A Comparative Approach to Multi-label Text Classification

Satvik Shetty, Gaurav Joshi, Ashis Kumar Sahoo

Department of Computer Science
North Carolina State University, Raleigh, NC 27606
{smshetty,gjoshi,aksahoo}@ncsu.edu

1 Introduction

Separating data into individual categories is called a 'Multi-Class' classification problem. Here every instance is put into one group or category. But there are several business use cases which require classifying the data into more than one groups or labels. For example, a photograph related to a crime scene may be categorized into the labels – "crime", "murder". This approach of classifying data into multiple categories or labels is called as a 'Multi-Label' classification problem. A multi-label classification problem arises in a variety of domains such as labeling multimedia resources such as images, videos and audio files, genetics/biology, and text classification such as emails, documents, reports, patents and the most popular – web pages. Classification of text into multiple labels is called a "Multi-Label Text Classification" problem.

In this project, we attempt to perform a Multi-Label Text Classification on news articles. The comprehensive dataset is provided by The Guardian (news agency) and is an ongoing data challenge called Growing Instability.

https://www.datasciencechallenge.org/challenges/2/growing-instability/

The goal is to correctly classify new news articles into various topics. This classification would help the team in question to focus and apply their resources on those areas pertinent to them. Our approaches to this problem would consist of applying algorithms such as Binary Relevance [1], Classifier Chains [2], Dependent Binary Relevance [3], OneVsAll using Naive Bayes [4] and OneVsAll using Stochastic Gradient Descent. We will then compare the performance of these multi-label text classification algorithms.

2 Business Use Case

Automated text classification has a multitude of business use cases that makes it an important problem in data science. Here are some business use cases relevant to the problem:

2.1 **Automated Ads suggestion:** The most popular example of multi-label text classification is the classification of web pages by google. If a web page is classified correctly, relevant ads (for example Google Ads) can be shown to the user, thereby increasing revenue.

2.2 **Automated Gene Sequencing:** This is aimed at predicting the biological function of genes. Each gene can express more than one function at once. Analyzing protein properties and gene expressions is a very costly task. The classification process is accelerated by using multilabel classification.

2.3 **Automated Review Categorizer:** Categorizing reviews and rating to relevant categories can be an important aspect for a lot of review and rating applications like Yelp. This high-level categorization of reviews into relevant categories can help users to understand why the reviewer rated a service as "good" or "bad". This information can help others to make a personalized choice.

2.4 **Medical Diagnosis:** A person might be suffering from multiple ailments at the same time. Based on the patient's medical history and the label interdependence, the diagnosis could indicate the diseases the patient is suffering from.

3 The Dataset

The dataset provided in the challenge consists of news articles spanning 16 years (from 1999 to 2014). It is provided as 32 JSON files, two for each year. Each JSON file consists of around 25000 articles. Four parameters are provided for each article

3.1 rowId: Unique Identifier for each news article
3.2 webPublicationDate: Date on which the article was published in DD-MM-YYYY format
3.3 topics: A list of topics to which this news is labeled. For example: "lifeandstyle", "artanddesign". The list of topics is empty for some articles.
3.4 bodyText: The actual text of news article.

A sample article looks as below:

```
"1999a_TrainingData_00002": {
      "webPublicationDate": "20-02-1999",
      "topics": [
      "culture",
      "media"
      ],
      "bodyText": "Experiments at the University of
Illinois, Kirsty Wark informed us in Tuning Into
Children (Radio 4), showed that …<truncated>"
}
```

4 Data Subsetting and Sampling

The data we were provided with contained more data than was required for our analysis. Also, since the competition is still live, the ground truth for the provided test data was not available.

Hence, for the sake of simplicity and to ensure the dataset would run on the limited resources available on our system, we take a subset of the data.

For the algorithms run on R using the mlr library (namely: Binary Relevance, Classifier Chains and Dependent Binary Relevance), we randomly selected 35% of a year's data (1999a) and divided it into the train (80%) and test set (20%).

For the algorithms run on Python using the scikit-learn package (namely: One Vs Rest Classifier with Naïve Bayes Classifier and One Vs Rest Classifier with Stochastic Gradient Descent Classifier), we took a year's data (2000b) for train set and another year's data (2000a) for test set.

5 Data Pre-Processing

Since the data given is in JSON format, we use `jsonlite` package to convert the files into a matrix

As a first step, we removed those rows from the data set that contained no topics We also removed those rows that contained no bodyText (which is the article in our case). This was done because these rows provided no new information that would help the training of our models.

Finally, we removed the first two columns of the dataset since they were not relevant to build our models.

Before the next process (feature extraction) executes, we apply various other functions to transform our data for building the models. These procedures come as parameters to the function we call for feature extraction hence need not be done separately. The following procedures are applied in the process: (1) Eliminating whitespaces, (2) eliminating stop words, (3) removing punctuations, (4) removing numbers, (5) lower casing, (6) Removing short words of length 2 or less, (7) TF-IDF: Term Frequency–Inverse Document Frequency

In R we use parameters in `DocumentTermMatrix` while in Python we use parameters in `TfidfVectorizer` for this purpose.

6 Feature Extraction

6.1 Feature Extraction in R

To train our models with the news articles, we need to extract relevant features out of this text. We derive word frequencies of all words appearing in an article, which is a relevant feature in this context. This is achieved by creating a `DocumentTermMatrix`, which is essentially a matrix of all documents vs all unique words appearing in the entire dataset. It stores the frequencies of the words appearing in each document. We utilize the Text Mining package (`tm`) in R for this purpose.

Table 1: Sample of Document Term Matrix generated in the process. Each row contains the frequencies of all words the document.

Words->	**british**	**first**	**former**	**government**	**may**	**months**	**never**	...
Document1	0	1	2	5	6	0	1	
Document2	1	2	1	0	0	5	1	
Document3	0	0	1	5	2	1	3	
…								

Taking all the unique words as features creates a huge matrix unsuitable for computation. So, we remove words that are too frequent and may not contribute to a better model. This is achieved by setting the sparse factor or sparsity to a high value (like 0.9) in the `removeSparseTerms` function.

Sparsity: The sparse factor or sparsity is one of the important parameters in generation of this matrix. This factor helps select the words that appear in only certain percentage of documents. A sparse factor set to 0.9 means that it removes those words that occur at most 10% of times in the entire document. This factor potentially fluctuates the model behavior since it increases or decreases the number of features involved in the model. A high value of sparsity retains most of the words whereas a low value eliminates words.

6.2 Feature Extraction in Python

For algorithms in python we used `TfidfVectorizer` from `scikit-learn` package to convert the article text to matrix containing count for each feature. We removed the stop words, used l2 norm to normalize term vectors, used ngram in range (1,3) and set different values to the parameters:

`min_df`: ignore terms that have a document frequency strictly less than the given threshold

`max_df`: ignore terms that have a document frequency strictly higher than the given threshold

We then extract the unique topics across the entire training data set and convert them into features. These are logical vectors that is a requirement for the worked upon classification algorithms.

7 Classification Approaches

To understand the problem in depth and create comparative statistics for arriving at the best method, we implemented several approaches to Multilabel Text Classification. Some of these methods were implemented in R and others in python. They are described below:

7.1 Binary Relevance using MLR Package in R

Binary relevance is the most straight-forward approach to Multi-Label classification. The basic idea of binary relevance is that that we carry out a 'binary' classification on each label/category to decide whether the label should be tagged to the document. For L unique labels given in the dataset, we transform the problem into L binary classification problems, one for each label. The binary classification problem can be solved by several algorithms. For our part, we have used the rPart (Recursive Partitioning and Regression Trees) algorithm.

An obvious drawback of binary relevance approach is that it considers each label independent of the other and does not factor in the correlation between labels. In spite of this drawback, binary relevance gives a reasonably good result in various scenarios.

7.2 Classifier Chains using MLR Package in R

This method trains consecutively the labels with the input data. The input data in each step is augmented by the already trained labels (with the real observed values). It is different from `BR` in that the attribute space for each binary model is extended with the $0/1$ label relevances of all previous classifiers; thus forming a *classifier chain*. Therefore, an order of the labels has to be specified. At prediction time the labels are predicted in the same order as while training. The required labels in the input data are given by the previous done prediction of the respective label.

7.3 Dependent Binary Relevance using MLR Package in R

Dependent binary relevance (DBR) models are similar to the CC models but instead of single/simple chained dependence in CC, the DBR model uses a cyclic structure i.e. each label is trained with the real observed values of all other labels. In prediction phase for a label the other necessary labels are obtained in a previous step by a base learner like the binary relevance method.

7.4 One Vs Rest Classifier from Scikit Learn in Python

This strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. This method is computationally efficient and provides easy interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier.
For this project, we have used the OneVsRestClassifier along with MultinomialNB for Multi-label Naive Bayes classification and SGDClassifier for Stochastic Gradient Descent classification from Scikit-learn package in Python.

7.4.1 One Vs Rest Classifier with Naïve Bayes Classifier using MultinomialNB:

Naive Bayesian algorithm is efficient. For the real-time system, it has a wide application such as Spam filtering system. For the task of multi-label classification, we can use the NB classifier (using MultinomialNB classifier present in Scikit-learn). NB classifier has a naive assumption of class conditional independence, i.e. the effect of a feature value on a given class is independent of the values of other features.

The multi-label classification problem is transformed into the combination of several single-label NB classifiers.

7.4.2 One Vs Rest Classifier with Stochastic Gradient Descent Classifier using SGDClassifier:

This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule. This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model fits a linear support vector machine (SVM). The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero-vector using either the squared Euclidean norm L2.

8 Performance Metrics

The training has articles and a list of corresponding labels. A new, unseen test set is used to evaluate the performance of our classifiers. The predicted tags are compared with the true tags and average Precision and average F-Measure is calculated for all the classifiers.
The Precision, Recall and F-Measure for the recommender h, on dataset D is calculated as follows.

$$Precision\ (p) = \frac{TP}{TP + FP}$$

$$Recall(r) = \frac{TP}{TP + FN}$$

$$F - Measure = \frac{(2 * r * p)}{r + p}$$

Where,

TP -> True Positive

FP -> False Positive

FN -> False Negative

8.1 For algorithms implemented in R:

Table 2: Test results for CC, BR and DBR methods evaluated in R with varying sparsity values.

	Sparsity	Precision	Recall	F-Measure
BR	0.93	71.02%	39.24%	50.54%
	0.90	69.69%	33.28%	45.04%
	0.80	56.85%	9.51%	16.29%
	0.50	54.03%	2.32%	4.46%
CC	0.90	67.72%	32.62%	44.03%
	0.80	43.46%	16.93%	24.37%
	0.50	28.87%	14.50%	19.31%
DBR	0.80	31.44%	30.88%	31.16%
	0.50	20.24%	35.98%	25.91%

From the above results, we can see that as we increase the sparsity, all three, Precision, Recall and F-Measure improve. This shows that we should set the sparsity at a higher value (say 0.99) as it is likely that words that occur at most 1% times would not be helpful in predicting the label. However, our system was unable to handle the computations at a sparsity greater than 0.93 for Binary Relevance, 0.9 for Classifier Chains and 0.8 for Dependent Binary Relevance.

Analyzing these results further, we see that while the BR model results in a high precision, the F-Measure being a weighted harmonic mean of the precision and recall, provides a better picture. The F-Measure on a relative scale is better for DBR than for BR and CC. This is a strong indicator that the DBR model might outperform the other two approaches at a higher sparsity value. However, this does so at the cost of higher computations, space and time complexity. Thus, BR would be a better choice when resources and time is of a concern.

8.2 For algorithms implemented in Python:

We tried with different values of max_df and min_df and found that the best results were obtained with min_df = 0.001 (ignore terms that appear in less than 0.1% of documents)
max_df = 0.9 (ignore terms that appear in more than 90% of documents)

We tried the models with different alpha (laplace smoothing parameter in case of MultinomialNB and regularization multiplicative term in case of SGDClassifier) values in the classifier.

Table 3: Test results for SGD and NB methods evaluated in python with varying alpha values.

	Alpha	Precision	F-Measure	Recall
SGD	**0.00001**	**84.26**	**71.92**	62.73
	0.0001	89.52	67.57	54.26
	0.001	95.54	41.94	26.86
	0.01	93.58	49.11	33.29
NB	0.00001	75.25	64.92	57.08
	0.0001	73.9	66.19	59.94
	0.001	**70.84**	**67.5**	64.1
	0.01	66.84	67.57	68.3

From the above data, we get following observations:

For SGD Classifier, we observed that the best result is obtained with data obtained with `TfidfVectorizer` parameters df_min = 0.001 and df_max = 0.90 and alpha = 0.00001. As, we increase the value of alpha, the precision improves but F-Measure decreases since the classifier makes less predictions for the labels.

For NB Classifier, we observed that the best result is obtained with data obtained with `TfidfVectorizer` parameters df_min = 0.001 and df_max = 0.90 and alpha = 0.001. As, we increase the value of alpha, the precision decreases but F-Measure increases since the classifier makes more predictions for the labels.

9 Future Scope

9.1 Other Methods

In the due course of this project we also came across several other potential methods of Multi-Label classification methods. One such method was suggested by Mark J. Berger [5] which used word embeddings and word order from Word2Vec and GloVe. This method employed convolutional and recurrent neural networks on word embeddings to solve the multi-label text classification problem. We also looked at approaches involving Mixure Models trained by EM (Expectation Maximization) by Andrew Kachites McCallum [6]. These methods may be evaluated in future for a comparative study.

9.2 Performance Improvements

Performance was a limiting factor in comparing the various models involved in the program. While we got reasonably good output, we could run tests on higher performance machines to achieve even better results.

10 Conclusion

Based on our experiments within the constraints of our machine performance, the One Vs Rest Classifiers gave the best classification of labels for test set. The best results for F-Measure are achieved with One Vs Rest Classifier Stochastic Gradient Descent Classifier with alpha value = 0.00001. If we decrease the value of alpha, the precision increases but, less predictions are made and as a result, the F-Measure decreases.

On the other side, DBR method gave most promising results. But we could only evaluate it for a maximum sparsity of 0.8. We believe that it might outperform the other methods at a higher sparsity value but also at the cost of higher computing.

11 References

[1] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas: Multilabel Text Classification for Automated Tag Suggestion.

[2] Jesse Read, Bernhard Pfahringer, Geoff Holmes and Eibe Frank: Classifier chains for multi-label classification

[3] Elena Montañes, Robin Senge, Jose Barranquero, José Ramón Quevedo, Juan José del Coz, Eyke Hüllermeier: Dependent binary relevance models for multi-label classification.

[4] Zhihua Wei, Hongyun Zhang, Zhifei Zhang and Wen Li, Duoqian Miao: A Naive Bayesian Multi-label Classification Algorithm with Application to Visualize Text Search Results

[5] Mark J. Berger: Large Scale Multi-label Text Classification with Semantic Word Vectors.

[6] Andrew Kachites McCallum: Multi-Label Text Classification with a Mixure Model Trained by EM.

[7] Grigorios Tsoumakas, Ioannis Katakis: Multi-Label Classification: An Overview

[8] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, Saso Dzeroski: An extensive experimental comparison of methods for multi-label learning.