Multi-Label Classification of EUR-Lex Dataset

Anshu Daur	Ashwini G D	Kartik Prabhu	Sachin Nandkumar
M_Nr: 223853	M_Nr: 223687	M_Nr: 224132	M_Nr: 223685

June 18, 2019

Abstract

This report details the work done by our team on the task of Multilabel classification done on Eurlex data set. We considered OneVsRest and Labelpower problem transformation techniques with various classifier like Linear SVM, Multinominal Naive Base and SGDClassifier. The report discusses about the implementation and evaluation of classifiers. As the dataset is highly label unbalanced, problems faced in analyzing and working on the dataset during preprocessing and how it impacted our evaluation are also explained.

1 Motivation and Problem Statement

The real world problems faced during text categorization is the main motivation to choose a multi-label classification scenario. For example in a newspaper; an article about a movie involving communal issues can belong to more than one category like movies and politics. Using a traditional machine learning classification algorithm to classify the articles in a newspaper would end up classifying this article into Arts/Movies section. There are high chances that sometimes these articles might be misclassified and placed in different sections of the newspaper.

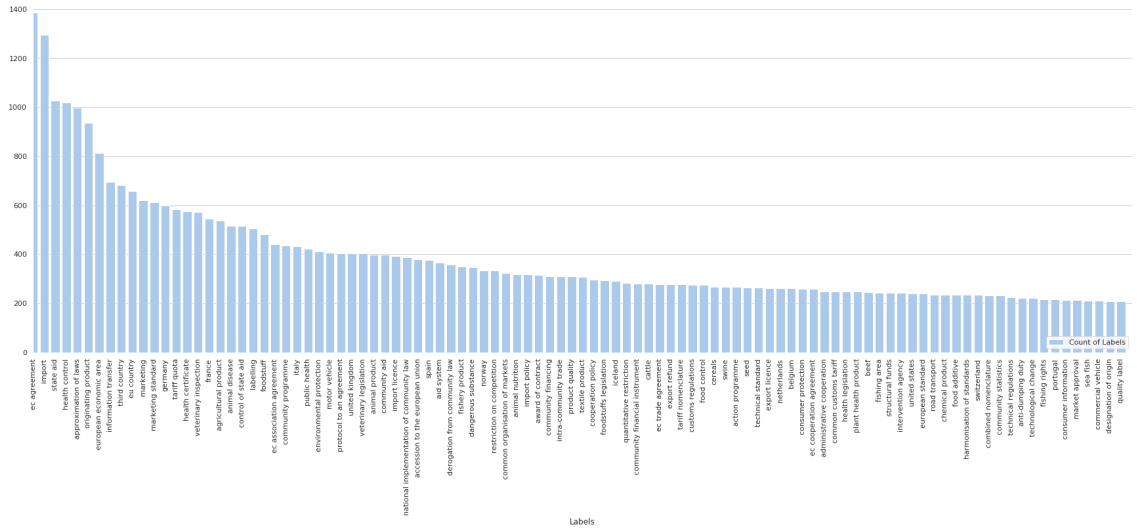
In order to avoid such confusions and difficulties, we can design a model which could classify the instances into one or more class labels at the same time. The EUR-Lex dataset offers one such multi-label classification problem. Along with the multi-label scenario described, there are many problems which are chained to this issue. It begins by dealing with the high dimensionality of the dataset. And as the number of labels increases, the computational cost of building the model also increase. One of the major issues to solve will involve handling a high imbalance of labels in the dataset. We might as well face several challenges while choosing the combination of labels due to its exponential growth. The dependency within the labels should also be taken care of during the design of the model.

The above-listed problems will be analysed on the EUR-Lex dataset and an attempt is made to handle them.

2 Dataset

The EUR-lex dataset is a collection of documents about European Union Law. The documents include treaties, international agreements, legislative proposals, case-laws, parliamentary questions and several other legislative documents. These documents are available in 24 dialects. But we chose to work with only English language dataset. It has 19940 legal documents in .html formal with 4000 labels which makes up the critical problem of multi-label classification. These 4000 labels belong to the "EUROVOC descriptor" category, which is a multilingual thesaurus. In order to check the distribution of labels in the dataset, we plotted the below graph that shows the count of each label (printed most frequent 100 labels).

Figure 1: Distribution of 100 Frequent labels in the Dataset



3 Concept

3.1 Feature Vector Representation

The most basic way of selecting feature would be words which have the highest frequency(tf). But as mentioned before, stopwords like 'a', 'the', etc, may have high frequency but gives no information. Even after removing stopwords, there are words which appear too many times in all documents like institution which will be not useful to the user. So, we used Tf-idf, along with term frequency, we want to give importance to how unique the words are across the dataset. Hence we take the product of tf and idf of words which gives us how frequent the word is in a document and how infrequently it has occurred across the corpus. Higher the Tf-idf, the better information the word gives about the document. We used tf-idf on unigram and bigram

Disadvantage: tf-idf doesn't consider the position of the word in the document, neither does it take semantic meaning with respect to neighbouring words.

3.2 Problem Transformation Methods

Most of the classifiers for NLP in Machine Learning predict single label problems effectively while multi-label classification is still an area of interest. In order to handle the multi-label scenario in our dataset, we transformed the problem by using Problem transformation techniques namely OneVsRest and Label Powerset before calling the classifier.

- **OneVsRest multi-label strategy** : The multilabel algorithm accepts a binary mask over multiple labels. The result for each prediction will be an array of 0s and 1s marking which class labels apply to each row input sample. In case of Eur Lex data set, this may lead to overfitting.
- **Label Powerset** : For Eurlex dataset, since the labels present provide and insight about what the document is describing. So we thought of using LabelPowerset transformation method as a wrapper for classifier so that it transforms the multilabel dataset into multiclass dataset by using the labelset of each instance as class identifier. So if there are labels present in the dataset, we will run the classifier for 2 power L combinations of labels to find correlations in the data.

3.3 Classification

- **Linear SVM**: SVM with a linear kernel function which supports sparse matrix and multi-class/multilabel when implemented with OneVsRest classifier. Hence this was the simplest and a good choice to be executed for Eurlex dataset.

- **Multinomial Naive Bayes**: One of our obvious choices was to try Naive Bayes Classifier due to its success rate even though the features are considered to be independent of each other. Multinomial NB classifies documents using the bag-of-words approach and is suitable for discrete features which in our case was generated using TF-IDF vectorizer. It still assumes that a word's context and its position is completely independent of its occurrence like that of the normal NB.
- **SGDClassifier**: This approach allows linear classifiers with the normal stochastic gradient descent training. The implementation allows working with the sparse matrix as well which is clearly the case under consideration. It also allows partial fitting of the dataset to the classifier which could be considered when the dataset is huge and there are significant memory issues.

4 Implementation

4.1 Dataset Pre-processing and preparation:

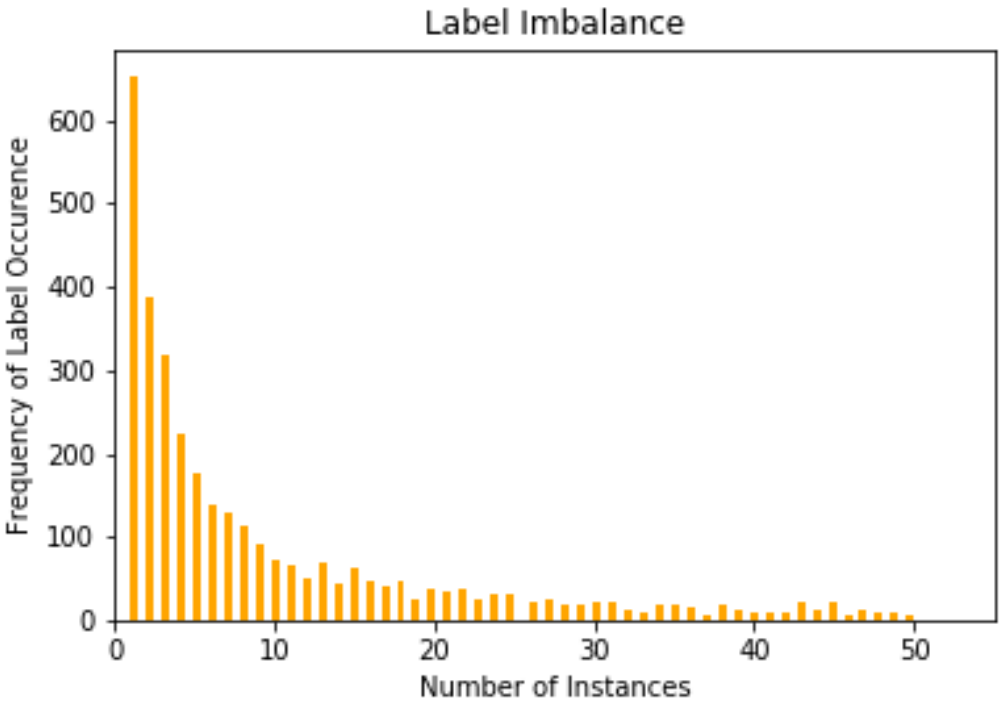
As part of preprocessing, actual text data is extracted by parsing HTML files using BeautifulSoup API (bs4 library). Content lies inside the "class:texte" and the labels are generated from Eurovoc Descriptors present as links in the HTML files.

Figure 2: Word cloud before preprocessing

- Remove HTML tags and read relevant text starting from "Classification" text in the HTML files
- Generate labels from EuroVoc Descriptors present in the HTML files
- Convert all words to lowercase
- Remove the words with numbers and alphanumeric characters except for hyphen ("-") in it as compound words are usually used with a hyphen.
- Perform lemmatization using lemmatize method of nltk which will reduce inflectional form of a word to its base form. For e.g. administrator and administration are reduced to admin (base for of both words)
- Perform Stemming using Porter stemmer, which transforms words with same semantics into their stem words.
- Remove stems of length 1 and 2

- Remove stopwords, using the default set of words from NLTK library. Also, we considered few stopwords from Eur-lex stopword list(determined from the word cloud as shown above). This removes the most frequently used common words which will give us less or no information about the document, and thus helps us focus on important words.
- Skipped documents which had empty strings and contained error message saying "There is no English version of this document available since it was not included in the English Special Edition".
- Imbalance in label set was analysed by considering the count of labels against the number of instances in which they occur. We tried visualizing the same over a histogram. Below snippet indicates the occurrence of all the labels over 19K documents.

Figure 3: Label Imbalance across the 19K documents



- Save preprocessed filename:[Content:Labels] in matrix form using Python dictionary. And keep appending the content and label of the files in the dictionary. We used binary representation(0-1) of the document containing the label. The matrix was then stored in a dataframe (Data structure offered by Pandas library in Python for efficient computations) and printed to CSV file for the next step i.e. model training and classification.

Figure 4: Word cloud after preprocessing

Below snippet represents data format which has 19331 rows indicating the count of documents and 3943 columns indicating the labels and its count.

	index	content	door-to-door selling	working environment	dissemination of community information	fruit juice	cinema	brussels region	foreign enterprise	enlargement of the union	...	reference price	community trunk route
0	31987Y0128(01)_EN_NOT	['updat version text commiss regul no septemb ...	0	0	0	0	0	0	0	0	...	0	0
1	31987Y0612(01)_EN_NOT	['resolut of consult committe restructur steel...	0	0	0	0	0	0	0	0	...	0	0
2	31987Y0704(03)_EN_NOT	['council resolut june consum safeti c council...	0	0	0	0	0	0	0	0	...	0	0
		['commiss opinion januari											

Figure 5: Data format

The word-frequency plot for the preprocessed data is shown below.

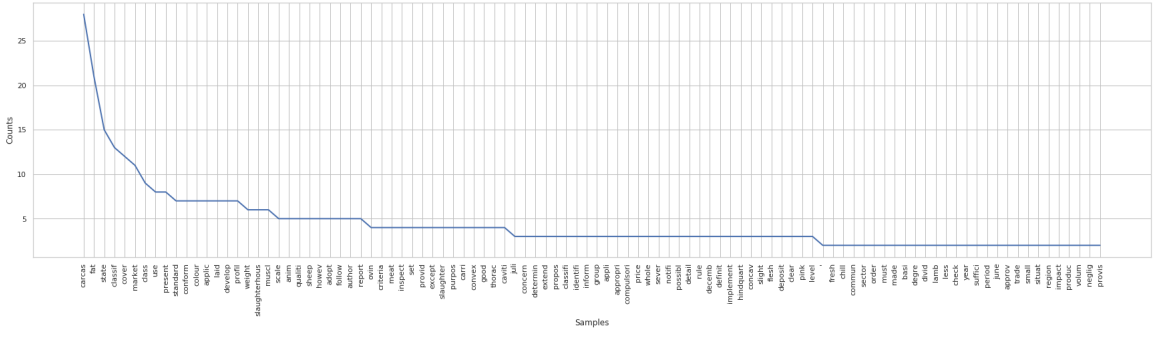


Figure 6: Word-frequency plot

The preprocessed data is now split into train and test using the function *train_ test_ split()* from the model selection library of sklearn [*sklearn.model_ selection.train_ test_ split()*]. The *test_ size* parameter inside the function indicates the percentage of data that should be held out for testing. We have set the *test_ size* parameter as mentioned in section 5.1.

Based on the problem transformation technique used for each method of implementation, we used *sklearn.multiclass.OneVsRestClassifier* library for OneVsRest and *skmultilearn.problem_ transform.LabelPowerset* library for LabelPowerset which acts as a wrapper to the classification algorithms we have used. For the classification process, we used 3 algorithms as mentioned in section 3.c. To implement SVM with linear kernel we made use of *sklearn.svm.LinearSVC*. Similarly, we used *sklearn.linear_ model.SGDClassifier* and *sklearn.naive_ bayes.MultinomialNB* libraries to implement SGDClassifier and Multinomial Naive Bayes respectively.

For evaluating the model, we made use of the measures as mentioned in section 5.2. The following libraries, *sklearn.metrics.accuracy_ score, sklearn.metrics.hamming_ loss, sklearn.metrics.precision_ score, sklearn.metrics.f1_ score, sklearn.metrics.balanced_ accuracy_ score.*

5 Evaluation

The distribution of labels in content is random and sparse, so we divided the dataset into 7:3 and used 70% data for training and 30% for testing.

5.1 Evaluation measures

5.1.1 Accuracy:

Accuracy gives an estimate of the number of correct predictions obtained from total prediction by the classifier. For Eurlex dataset class imbalance is very high, accuracy is not a good measure for estimation as the model will over-fit for the labels which are present in most of the documents. Also if a class is represented too many in one of the train/test split but not in others, it will return too many true negatives. Also, when we use accuracy, we assign equal cost to false positives and false negatives. When the data set is imbalanced - say it has 99% of instances in one class and only 1% in the other - there is a great way to lower the cost. Predict that every instance belongs to the majority class, get an accuracy of 99%. Thus accuracy, is not a balanced evaluation metric, especially for multilabel classification.

5.1.2 Balanced Accuracy:

The balanced accuracy in binary and multiclass classification problems deals with imbalanced datasets. It is defined as the average of recall obtained on each class. For Eurlex dataset, recall plays an important role as it will tell how many actual labels (true positives) were predicted. Thus we selected both accuracy and balanced accuracy for evaluation measures to compare the importance of each of the metrics and justify why accuracy is not always a good measure.

5.1.3 Hamming Loss:

It's the fraction of labels that are incorrectly predicted. We have used sklearn.metrics.hamming_loss(ytrue, ypred, labels=None, sample_weight=None). Here, ytrue : Labels already present in Test fold ypred : Labels predicted by the classifier

5.1.4 Precision:

It is a useful measure when the classes are very imbalanced as it tells how successful was the classifier while making correct predictions. we have used precision score method of sklearn to calculate precision and then performed averaging over the values of precision we used

5.1.5 F measure:

F measure finds the trade-off between precision and recall. We have used f1score method by sklearn api using micro and macro as averaging parameters suitable for multilabel classification. Micro

averaging calculates weighted harmonic mean by taking label imbalance into account whereas macro averaging will take unweighted harmonic mean of precision and accuracy.

6 Evaluation Results

Figure 7 and Figure 8 indicates the comparative results of our model for various algorithms, problem transformation techniques and evaluation metric chosen as part of our project.

Initially we tried applying Linear SVC algorithm with OneVsRest as a wrapper,with unigram text feature selection. To further interpret our results, applied the same set of algorithm and wrapper technique, but with bigram as a feature selection parameter. We could observe that our model provided better results with unigram as feature selection parameter. We expected the bigram selection to provide better results, which failed; indicating less co-relation between the words in the documents and also the reduced number of features selected.

In "LinearSVC Optimised", we tweaked parameters like "min_df" and "max_df" which sets a threshold to the selection of features. Parameter max_df ignore the terms that have document frequency higher than the given threshold, where as min_df ignore the terms that have document frequency lower than the given threshold. But with this setup, we did not have any variation in our evaluation results because even after handling label imbalance by reducing the unimportant labels the distribution of labels is very random and does add value to our evaluation.

For better interpretation and understanding of our problem statement, we tried several other algorithms in combination of two different problem transformation techniques. Observing the results of these implementations, we could see that our methods could not completely fulfill the requirements of our problem statement. It might be due to the improper handling of label imbalance issue.

With respect to the results in Figure 7, the model was trained one label at a time where as w.r.t the results provided in Figure 8, the model was trained on the whole labelset at once. Only major difference was the runtime taken by each of the models to learn the data.

tf-idf	Problem Transformation Technique	Algorithm	Hamming-loss	Accuracy	Macro-f1	Micro-f1	Balanced_accuracy_score	Precision
Unigram	OneVsRest	LinearSVC	0.00167	0.9983	0.6741		0.643	0.534
		Linear SVC Optimized	0.00161	0.998	0.673	0.998	0.644	0.514
Bigram		LinearSVC	0.00169	0.9983	0.6579		0.626	0.532
		Linear SVC Optimized	0.00165	0.998	0.6528	0.998	0.625	0.472

Figure 7: Model Results(One label at a time)

tf-idf	Problem Transformation Technique	Algorithm	Hamming-loss	Accuracy	Macro-f1	Micro-f1
Unigram	OneVsRest	MultinomialNB	0.002	0.007	0.027	0.202
		LinearSVC	0.002	0.048	0.268	0.517
		SGDClassifier	0.002	0.023	0.121	0.393
	LabelPowerset	MultinomialNB	0.003	0.013	0.011	0.053
		LinearSVC	0.002	0.166	0.423	0.528
		SGDClassifier	0.004	0.031	0.07	0.156
Bigram	OneVsRest	MultinomialNB	0.003	0.006	0.041	0.263
		LinearSVC	0.002	0.031	0.178	0.442
		SGDClassifier	0.002	0.016	0.082	0.331
	LabelPowerset	MultinomialNB	0.002	0.135	0.336	0.469
		LinearSVC	0.002	0.135	0.336	0.469
		SGDClassifier	0.004	0.027	0.048	0.109

Figure 8: Model Results

7 Conclusion

Data preprocessing went well and we did not face many issues while dealing with parsing and cleaning of the dataset. Classifiers that we selected to solve the problem returned the similar results. We think that might be because of random distribution of labels for all documents. Better performance could have been achieved using principle component analysis for feature space reduction (dimensionality reduction) and handling of sparse label matrix before feeding it to the classifier. Also, stratified method could not work because of sparse label matrix, hence random generation of folds using inbuilt python method was used. We think that after performing PCA and handling sparse matrix, stratified could be applied to generate meaningful test and training folds. In short, more analysis on handling of dataset could have provided better results.

We also ran into memory issue while working with the dataset and our system crashed(Windows 7, i7 processor, 8GB RAM) while working with LabelPowerset and classifierChains along with LVC classifier. The runtime complexity was very high which required atleast 8 hours to run the classifier and randomforest ran for two days and did not return any results. Runtime complexity could have been handled using parallel processing during prediction by the classifiers but we are not sure how it can be approached as of now.

8 User guide for execution

The user should run the executor file with the following parameter:

```
For Executing: python executor.py <RunType> <Source> <Destination> <ClassifierType>
RunType = 1 {For preprocessing}, 2 {For training}
Source = Folder path of training files (only for preprocessing)
Destination = Folder path for preprocessed files
ClassifierType = 1{For OneVsRest LSVM Unigram (single label at a time)}
               = 2{For OneVsRest LSVM Bigram (single label at a time)}
               = 3{For OneVsRest LSVM Unigram with tfidf parameter min_df=0.01, max_df=0.8 (single label at a time)}
               = 4{For OneVsRest LSVM Bigram with tfidf parameter min_df=0.01, max_df=0.8 (single label at a time)}
               = 5{For OneVsRest LSVM Unigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 6{For OneVsRest LSVM bigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 7{For OneVsRest MultinomialNB Unigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 8{For OneVsRest MultinomialNB bigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 9{For OneVsRest SGDClassifier Unigram tfidf parameter min_df=0.01, max_df=0.8 }
               = 10{For OneVsRest SGDClassifier Bigram tfidf parameter min_df=0.01, max_df=0.8 }
               = 11{For LabelPowerset LSVM Unigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 12{For LabelPowerset LSVM bigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 13{For LabelPowerset MultinomialNB Unigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 14{For LabelPowerset MultinomialNB bigram with tfidf parameter min_df=0.01, max_df=0.8 }
               = 15{For LabelPowerset SGDClassifier Unigram tfidf parameter min_df=0.01, max_df=0.8 }
               = 16{For LabelPowerset SGDClassifier Bigram tfidf parameter min_df=0.01, max_df=0.8 }
```

Figure 9: Execution arguments for the program

Please make sure that source path is given only for preprocessing and not for training. The destination path should have minimum memory space of 400MB as 10 .csv files will be generated from 19940 documents. All the evaluation will take place simultaneously and there is no need to specify it. You can find the whole code here: https://github.com/kartikprabhu20/Multi-label-Classification