Multi-label Classification with EUR-Lex Dataset

Sebastian Hennig, Chetan Singh

15th June 2019

1 Motivation and Problem Statement

The motivation of given task of Multi Label Classification comes from the fact, that with radical increment in the availability of information, it becomes significantly more simple to discover problems related to a specific field to set up and investigate upon, For Ex. A dataset related to a specific problem consisting of high dimensional features and profound concepts with complex labelsets. In any case, as the process goes on effortlessly of discovering problems, diving into the arrangements to come up with a legitimate solutions for the same has turned out to be very intense. As finding those legitimate arrangements tending to those ideas is presently continued to be an obligatory advance.

The objective is to take care of an issue which is close by but somewhat not quite the same as the traditional classification problem, which typically includes binary or multi class classification. The dataset given falls in Multi Label Classification issue. A Multi Label Classification undertaking includes a dataset in which one instance is appointed with multiple (at least two) labels. For instance, a multi class arrangement would be to classify whether an animal appearing in a picture is a cat, a dog or perhaps some other animal, where each example of the Multi Class Classification will just have one mutually exclusive label. Aside from that, Multi Label Classification fundamentally comprises of an undertaking to relegate various labels to an instance like a given movie that can fall into every one of different classifications as action, adventure, drama, comedy, and so forth.

Assigning multiple labels to a particular instances is a significantly troublesome task and our goal for this task is to discover reasonable techniques to manage this issue appropriately. Hence we need to deal with the characteristics of the dataset regarding how it is structured and organized, how the proportions of the labels are balanced, the measurement and the dimensions it grows to and investigate numerous approaches to manage all these. Finally yielding them with ideal evaluation techniques, as the traditional strategies which are typically related to binary or multi class classification, tend not to suffice at this issue.

2 Data Set

The dataset given for the present undertaking is fetched from the EUR-Lex repository directly provided by TU Darmstadt. The EUR-Lex is a portal which gives free access to the European Union Law information in about 24 unique dialects. The dataset which we are going to utilize is an accumulation of documented archives in regards to European Union Law in a several categorization plans identified with labels related to a document. The labels allocated to the cases are fundamentally various aspects of European Law.

The dataset consists of text documents in .html format. Each document is classified with multiple labels with three different categorizations as given:

- EUROVOC Descriptor
- Subject Matter

- Directory Code

We opted for EUROVOC descriptor as our classification category because it contains a higher number of labels than the rest two classification categories and is asked for in the task.

Below we list some statistics related to the current dataset.

- Total number of documents: 19,940
- Total number of labels assigned: 1,03,629
- No. of unique labels assigned: 4,054
- Minimum number of labels assigned to a document: 1
- Maximum number of labels assigned to a document: 24
- Number of labels assigned to majority of documents: 6

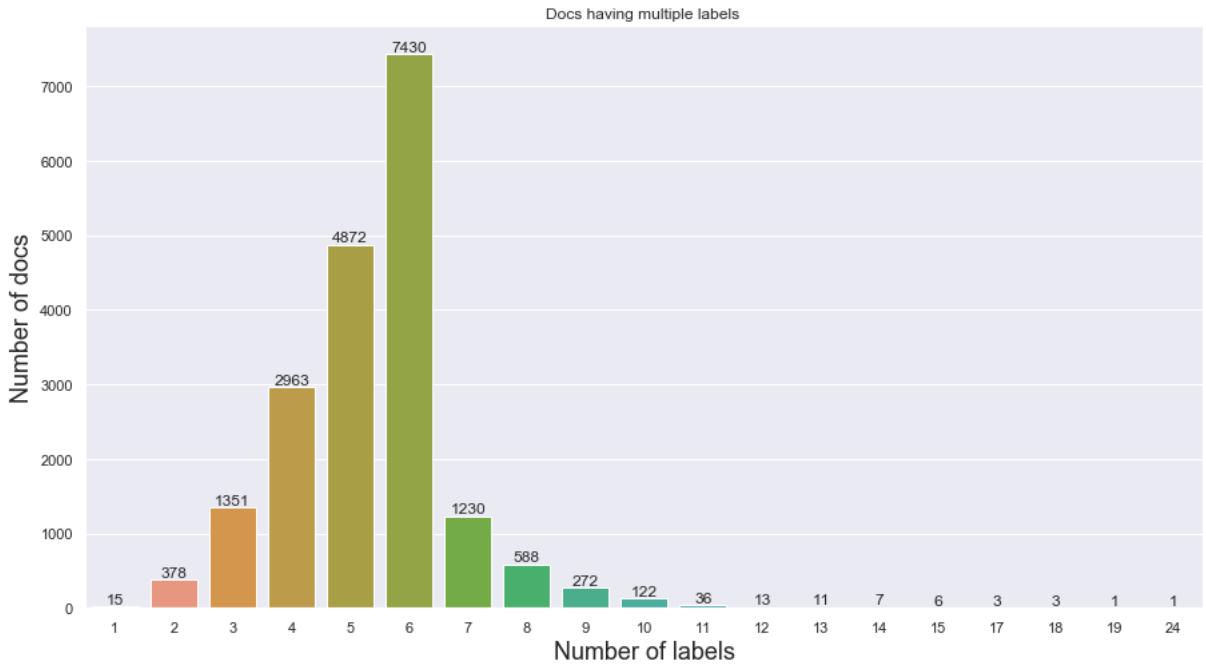

Figure 1: Number of labels distributed over all the documents.

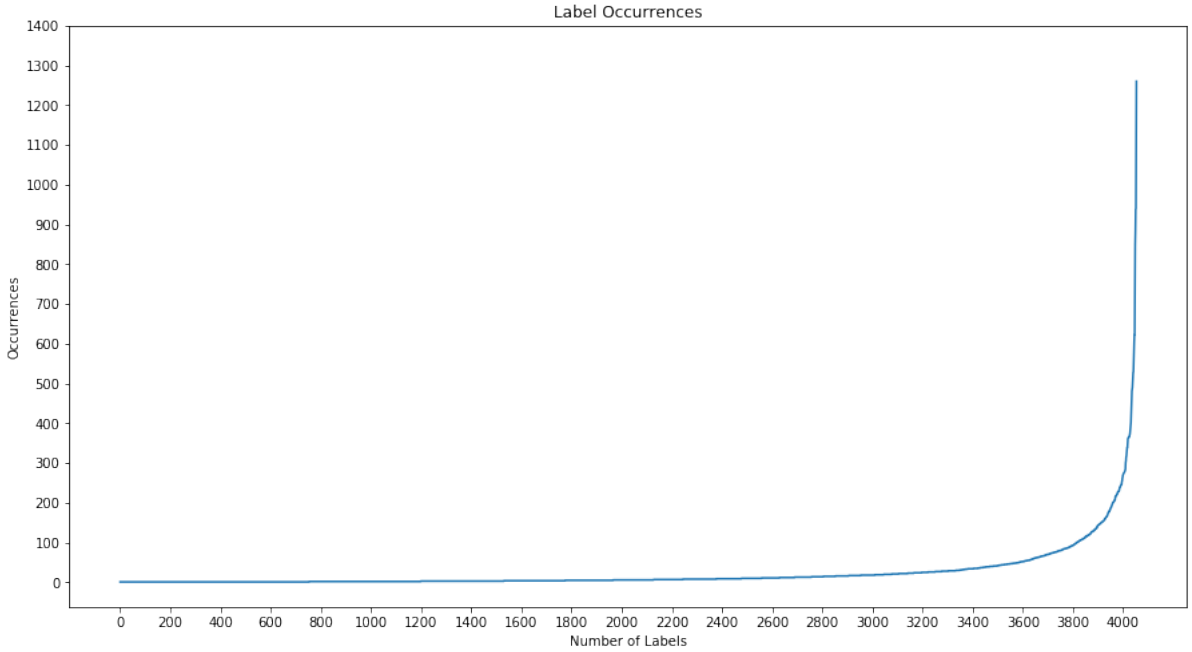

Figure 2: Label occurrences throughout all the documents.

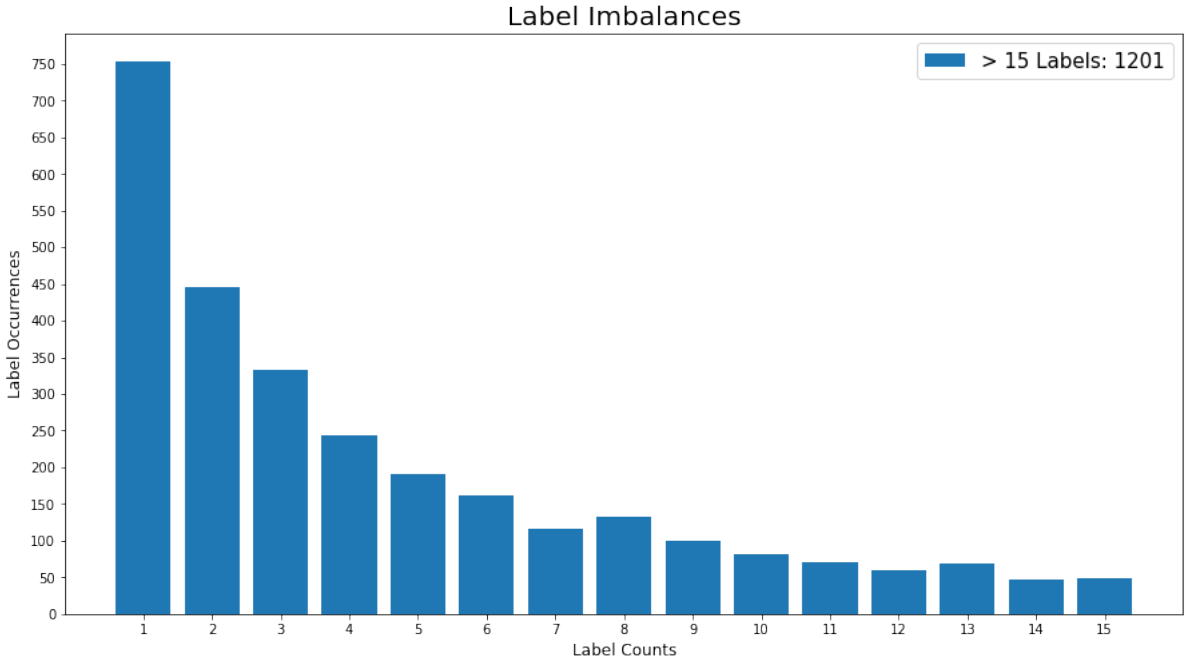

Figure 3: Label imbalances.

3 Concept

The aim is to solve the Multi-Label Classification task on the EUR-LEX dataset provided by TU-Darmstadt using two different approaches. Our first approach is using a Neural Network and the other is One vs. Rest Classification using Multinomial Naïve Bayes classifier.

To apply these methods as per our requirement, we first needed to transform the input data into a format that can be used to feed to the corresponding classifier. Therefore we opted to try different approaches. The simplest approach being *TF-IDF* bag of words representation for the

Multinomial Naïve Bayes. Another one is the Bi-Gram bag of words representation which offers some benefits since it catches sequential features like the word order that are basically inherent in textual data. It comes with a price though since it drastically increases the size of the resulting feature vector. For the neural network we use a dictionary to transform every word in the document to an index which we will later use in a look-up table. Since not every document has the same length we apply batch wise padding, always padding to the size of the longest document in the corresponding batch. This representation has the benefit of keeping the complete sequence order and is not increasing the feature vector size. Those feature vectors now need to be used for some model.

First we will build a neural network using a *LSTM* [HS97] model, this allows us to exploit the sequential nature of text data to a maximum since *LSTMs* are build for exactly that: sequential data. But we don't just put the indices into the network but passing them through an Embedding/Look-Up Layer beforehand. This Layer creates a meaningful word embedding feature vector for each word that has much richer information than just the index and hence provides better information for the actual *LSTM* layer. We then map the output through a *"tanh"* function to a number-of-labels sized output vector which should be equal to the labels. To make that possible we transform the labels to a vector as well. Here we use the one-hot-vector approach or since it is a Multi-Label problem it's 'multi-hot-vector'. This means every label is mapped to an index and if a document has that label, the label vector has a one at the corresponding indices and zeros otherwise. Since it is a Multi-Label classification problem we use the *Back-Propagation Multi-Label Learning (BP-MLL)* [ZZ06] cost function, to address this problem. BP-MLL tries to maximize the difference between correct and incorrect labels instead of trying to predict a copy of the exact output vector. This difference from approaches like Mean-Squared-Error or Log Loss create an advantage for multi-label classification tasks. To take care of the imbalances we additionally penalize the error of not so frequent instances heavier then the error for often occurring instances.

For our second model the one vs. all Multinomial Naïve Bayes classifier we first need to augment the dataset to create a binary classification task for each label in the dataset. We then train a binary MNB classifier on each label instance where the positive examples are all documents containing that label and the negative examples are the rest. We use a multinomial approach since this one is known to be best for textual data. This method makes two assumptions: First that the word order doesn't matter since this information is lost in the Bag of Words representation aswell as in the tf-idf Version. Second one being that all features (words) are independent of each other and following a multinomial distribution. That is not equal to the distribution that zipfs law is suggesting but it's better fitting than the Gaussian approach. These assumptions do not correspond to the actual characteristics of text data since for example word order can change the meaning of a sentence. Therefore we are not able to perfectly represent our text data via this assumptions but they are close enough to get meaningful inputs for our Naïve Bayes classifier. One way to introduce some attention to word order is using a bigram representation of the words which we also use as aforementioned. To infer the labels of an unseen document we run the document through all the classifiers and assign the label to it if the corresponding classifier outputs a 1. To take care of the imbalances we also want to adapt the one vs. all approach to an one vs. some approach. This means we will have a proportional amount of negative examples to positive examples for each MNB classifier. This avoids cases that have 1 positive example and about 19900 negative examples which will most likely result in just returning a zero for every possible document.

4 Implementation

For all of our source code we used python version 3.7.3. The additional libraries that we used are listed in the following paragraphs marked with double quotes. This helps to understand which library was used for which task. A list of all used libraries can be found in the appendix.

The dataset is organized in .html format, comprising of its textual contents and labels. We originally scraped the textual contents and the EUROVOC Descriptor labels individually from the documents utilizing tools like *"BeautifulSoup"* accessible in python. At that point we joined the scraped contents and their particular labels to a two dimensional list represented as a solitary

instance. This procedure is rehashed until we eventually scraped and consolidated every one of these instances and saved it in a tabular arrangement.

As the majority of the data is in textual format, we opted not to keep any numerical values or any kind of non alphabetic symbols in the feature set since words containing them are not part of the English language and we only consider English documents in this dataset. We removed all the English stop-words and then used Porter Stemmer to stem all the individual tokens from the contents by using *"NLTK"* library accessible in python. As applying those will help us to filter out non important words and simplify the features for our models by mapping similar words to the same stems. We can do this since in the large majority of cases words with the same stem have the same meaning and/or context. Also due to the shrinkage of the feature space we end up with marginally less processing time.

As should be obvious from Figure 3. the dataset is profoundly imbalanced, to handle this issue we favoured two situations. In first case we dropped the labels which occurred less in the dataset, and furthermore we utilized oversampling to make copies of instances identified with the labels, which occurred less and push ahead with both of the strategies to see which one performs best. To not loose to much data we didn't delete the complete instances that contained infrequent labels in the downsampling approach. We only deleted the infrequent label from the labelset of that instance. Only if the labelset is empty after the removal of the infrequent label we will delete the complete instance. For the oversampling we also didn't just copy the complete instance since this would not change the label ratio in an effective manner. To prevent copying already frequent labels and pushing their frequency even further, we removed all labels but the infrequent one in the copy. This ensures that only infrequent labels get more frequent and not the whole labelset.

To transform the preprocessed documents and labels into feature vectors we created dictionaries for all the words occurring in the all the documents and labels. Since the documents are supposed to be English we filtered out all words containing non-Latin letters. In the dictionaries the keys are the words/labels and the values are the indices that they are assigned to. We also build a bigram dictionary which is build analog to the word dictionary with the difference that instead of using single words we iterated over all pairs of words that are next to each other. For the Bigram dictionary we threw out all bigrams that only occurred once which reduced the feature vector size from roughly 3M to 1.7M. Which is still enormous compared to the word dictionary size which is around 160K. Furthermore we calculated the IDF for each word and saved it in a dictionary to use it later for the *TF-IDF* vector representation. For the labels we also calculated a dictionary of penalized values which mapped each label to the inverse of its frequency e.g. if a label occurs in 30 documents the value assigned here would be 1/30. We use this value later for the penalization in the loss function. The actual transformation to the feature vectors is done during the runtime of the model since the feature vectors are the ones that cause trouble when it comes to memory constraints. So far the libraries used are *"pickle", "re", "math" and "random"*.

For the implementation of our neural network we used *"tensorflow 1.13.1"* and an implementation of the *BP-MLL* loss function in tensorflow from this repo:

https://github.com/vanHavel/bp-mll-tensorflow

To add the penalization term we had to adapt the *BP-MLL* function. This function was originally defined as:

$$E = \frac{1}{|Y||\overline{Y}|} \sum_{(k,l)\in Y\times\overline{Y}} \exp(-(p_k - p_l)), \tag{1}$$

where Y is the set of labels that a document x is associated with. $\overline{Y}$ being the complementary set to Y and p_i is the prediction of the i-th label based on the instance x.
To implement the penalization, we changed the formula as follows:

$$E = \frac{1}{|Y||\overline{Y}|} \sum_{(k,l)\in Y\times\overline{Y}} \frac{1}{f_k} \exp(-(p_k - p_l)), \tag{2}$$

Where f_i is the number of occurrences of label i with respect to the complete dataset. By that we reduce the error for labels that occur often and keep the error the same for labels that only occur once in the whole dataset. Thus penalizing rare labels compared to frequent labels. To implement this, we changed the source-code of the bp-mll implementation.

To handle the high dimensional feature vectors with constrained memory resources we build a generator function which only translates the current batch which is fed to the neural network into the feature vector representation. So we only load the whole dataset, where the documents are strings and the labels are a list of strings, into memory and then generate the memory intensive vector representations on demand.

For the Multinomial Naïve Bayes approach we additionally used the libraries *"scikit-learn", "random" and "itertools"*. Again we use a generator approach which only builds the final feature vectors, here TF-IDF vectors, when they are needed for training. Next we can choose if we go for the One vs. Rest approach or the One vs. Some approach. The One vs. Rest is the typical text book approach where we just use the documents corresponding to a label as positive samples and all other documents as negative ones. Where as in the One vs. Some approach we count how many positive instances where found for one label and then multiply that number with some scalar. The result is the number of documents we randomly sampled from the dataset that remains when removing the positive instances for that label.

To not run in an out of bound exception we need to be careful that the most occurring label times the scalar value does not exceed the number of available negative examples. This allows us to assign an equal class distribution to all classifiers where we still manage to represent the skewed ratio of positive to negative samples for each of the labels. We then proceed to train one Multinomial Naïve Bayes classifier for each label in the label set using the dataset created for each label as described before. The trained classifiers are then saved in an array and written to disk. For inference we load the model array and run all 4000 classifiers on each instance. Since the output is either 0 (negative example) or 1 (positive example) we can concatenate the results of all classifiers to a vector and get the predicted label vector which we can compare to our truth label vector.

5 Evaluation

In this Section we will talk about how we divided our dataset into training and test data. Next we shortly mention how we can get a prediction from the neural network output. We will then introduce some evaluation measures and finally compare our models with different scenarios and parameters.

5.1 Data Split

We split the dataset up into 80% training and 20% test data. We thought about incorporating a validation set with 10% reducing the training data to 70%. But the scikit-library provides a fit functionality to train your model, which does not have intermediate results. Hence the validation set has no use for the Multinomial Naïve Bayes approach. For the neural network it is an common approach to check the validation error after every epoch to implement early stopping and avoid overfitting. But in our case it didn't make much sense. Due to time and hardware constraints we couldn't train more then 3 epochs per model and therefore the risk of overfitting is negligible. Since it didn't make sense to use the validation set for either of our models we merged it back into the training set and ended up with the 80/20 ratio of training to test data.

5.2 Neural Network Output

When we put instances in our neural network we receive a vector the size of the labelset. But the output is not zeros and ones but some floating point number between -1 and 1 (due to the nature of the tanh function). To get to a 0,1 representation we define a threshold of 0. So all values that are greater then 0 will be assigned a 1 (label belongs to that instance) and all other will be assigned a 0 (label does not belong to that instance). We choose the value 0 for our threshold since it is suggested by [ZZ06]. The authors actually trained another classifier to determine the optimal threshold but unfortunately we didn't have the time and resources to look into it and went with the default value.

5.3 Measures

Here we will have a look at the different measures used for evaluation namely Hamming Loss, Jaccard Score, F1-Micro, F1-Macro and Accuracy and discuss why they are useful for a multilabel case. But first we will shortly discuss why it is acceptable to not have perfect matches with the true labels and still get good results. In multi label problems there are often many classes that fit to a subject. To predict all correct would be perfect but we don't need to aim that high since we got multiple results to choose from. Imagine a document being about relationships. If in a binary classification task, our classifier would suggest that this document is not about relationships, we would have a big problem. But in multi label scenario even if it is not tagged with the label relationships it could still have the labels love,boyfriend,girlfriend,trust,loyalty assigned by the classifier. Even if the true labels have the relationships label and don't have the loyalty label, which would be an error for the perfect match scenario, we end up with a pretty good understanding what the document is about and can infer the missing information.

5.3.1 Hamming Loss

The hamming loss is defined as the fraction of wrongly classified labels to the total number of labels over the complete dataset.

$$\mathrm{HL} = \frac{1}{|D||L|} \sum_{l=1}^{|L|} \sum_{i=1}^{|D|} P_{i,l} \oplus T_{i,l}, \tag{3}$$

where D is the dataset, L is the labelset, P is the prediction matrix and T is the truth matrix. An entry $a_{i,j}$ written as $P_{i,j}$ when the entry is from the prediction matrix and $T_{i,j}$ when it's from the truth matrix, has the value 1 if the l-th label is assigned to the i-th document else 0. The $\oplus$ is the XOR operation. This measure is useful for multilabel tasks since it doesn't penalize small mistakes too harshly when the vast majority of labels is predicted correctly. Where traditional binary or even multiclass scores check for perfect matches, in the multilabel case it won't matter too much if not all labels are perfectly assigned as long as the majority is correct.

5.3.2 Jaccard Score

The Jaccard Score measures the similarity of two finite sets using union and intersection.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{4}$$

where A is the set of labels that belong to an instance and B is the predicted set of labels for that instance. Since the union gives all elements of both sets we get the total number of pairwise distinct labels. The intersection of the two sets gives the items that are equal to each other which in our case are the correct predictions. So the score is the fraction of the correct positive predictions to the sum of the correct positive predictions and all wrongly classified predictions. For this score larger values are bigger since we get a 1 for two equal sets. This score is similar to the hamming loss regarding the softer penalization of not perfect fits but is more challenging since it counts only positive matches. This eliminates the possibility to just set every label to zero and still get a good score since the majority of labels is zero anyways. This is especially true for tasks like this

one, where the labelset is huge and there is a maximum of 24 labels per instance. Hence setting everything to zero gives you a maximum error of $\frac{24}{4000}$ using the Hamming Loss. This behaviour will be punished using the Jaccard Score.

5.3.3 F1-Micro

This measure is a adjusted method of the standard F-measure as for example defined in [Sas07]. To handle the multiple labels the confusion matrices are calculated for every label separately as they would be, if that label would be a binary classification task. To calculate precision and recall we then proceed to add up the necessary values of all confusion matrices to receive a score that represents the contribution of each of the labels to the final result. With the micro averaged-precision and -recall we continue as usual to calculate the micro F1 score.

5.3.4 F1-Macro

The F1-Macro measure is very similar to the F1-Micro measure. In fact it only differs in one point. Instead of using all necessary values of all confusion matrices to calculate the precision and recall, we get the precision by calculating the precision of each label separately and then adding up all the precision scores. This value is then divided by the total number of labels which gives us the macro precision. The macro recall score is handled analogue. The F1-Macro is then calculated as stated in section 5.3.3.

5.3.5 Accuracy

Accuracy measures the ratio of perfect matching predictions to the total number of points in the dataset. Again, this is not a very useful measure since in a multi label scenario we don't need perfect fits to retain good results as stated before, but it is interesting to check how many perfect matches our models can produce.

5.4 Results

We trained the neural network model aswell as the MNB model on both datasets, the undersampled and the oversampled one to compare the results. For the MNB model we additionally used the ove vs. all and one vs. some approach with scaling factor 12 for each of the both datasets. The results are shown in figure 4. Since some of the values are close to one and others are in the range of 1×10^{-3} to 1×10^{-2} we used a logarithmic score to better visualize the results. We just have to keep in mind that through the scaling the gaps get larger as we come closer to 1. That means small deviations in the top third of the graph are actually bigger then they are shown, compared to distances between two points in the middle or bottom third. Also since the log of zero is undefined we had to handle zero values that occurred in the accuracy measure by setting them to a small value. So all scores having a value of 1×10^{-6} are actually zero values. All scores except the Hamming Loss get bigger the better the result is. Only the Hamming Loss is better the smaller the score is.
Now that we know the properties of our scores and diagram we can have a look at the actual results. We can see that the neural network approaches perform worst for the all scores except the hamming loss. In the hamming loss though the neural network with the undersampled training data scores best. The bad performance can be explained by the simplicity and size of the network as well as the insufficient training time of only three epochs. An explanation why the undersampled neural network outperformed all the other approaches on the hamming loss could be that it just set everything to zero and therefore eliminating the chance of any false positives. And since the number of labels assigned to an instance is tiny compared to the complete labelset, the impact on the total error of getting each correct label as a false negative is also limited. The other approaches which are more likely to assign a label to an instance therefore also have a higher chance of making false positive errors additionally to the false negatives.
We can also see that the undersampled version performed better than the oversampled version in all cases. That can be attributed to the lower number of labels that remain for the undersampled dataset and the fact that the copies don't provide new information in the oversampling case. Interestingly though the oversampled MNB models where the only ones that could produce some exact matches. We also found that the one vs. some approach, even if the results are not much

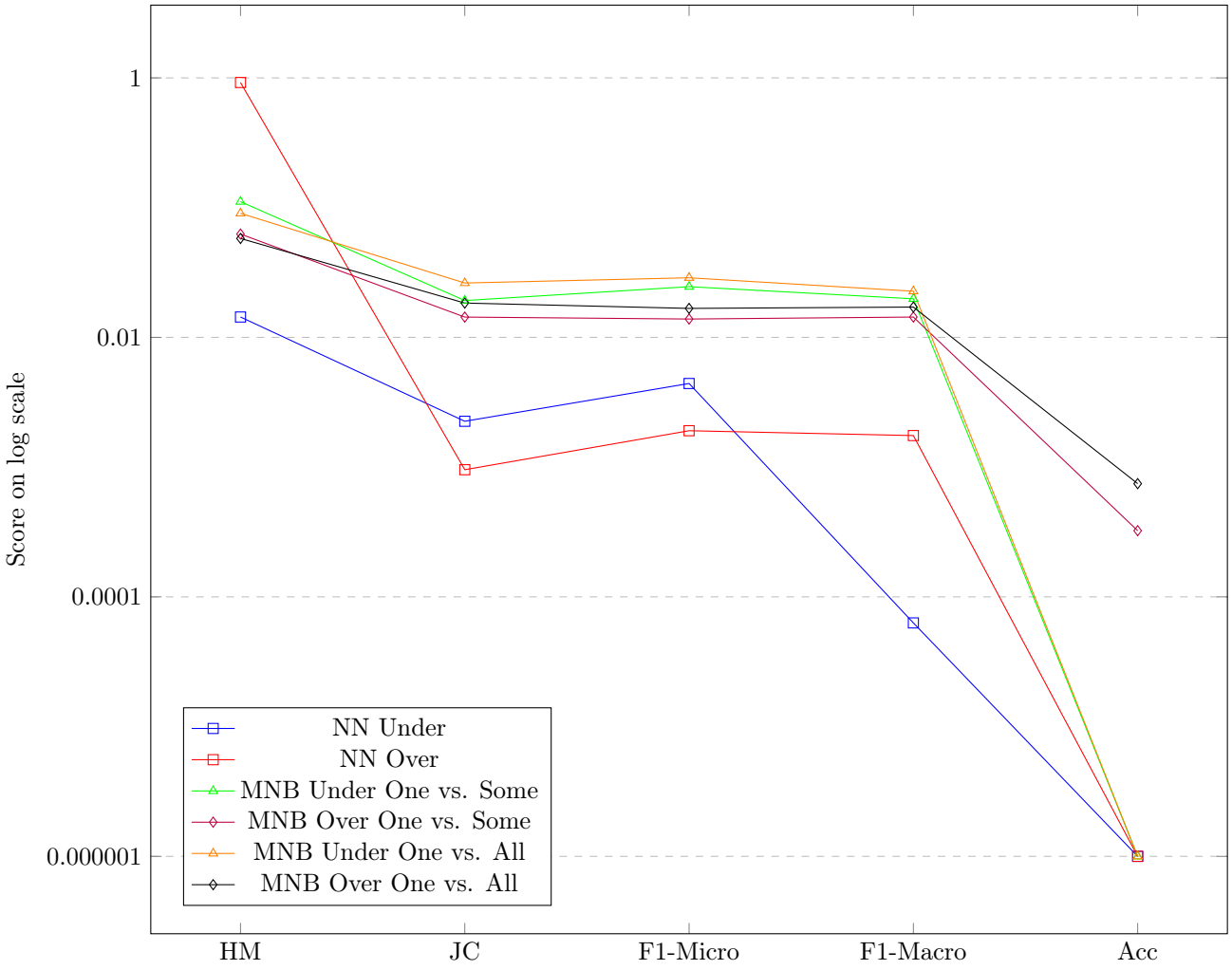

Figure 4: Results of all models on different measures where HM is Hamming Loss, JC is Jaccard Score and Acc is Accuracy. The log score does not allow zero values hence the zero values for accuracy are set to 1×10^{-6}

worse, cant compete with the one vs. all approach. This is probably due to the fact that the testset underlies the same distribution as the training set. While the one vs. all approach takes this into account by using the same distribution as the test set, the one vs. some approach changes the distribution of negative to positive examples to some constant scalar for all labels.
Unfortunately we didn't have enough time to test the bigram representation for the MNB models which could have given another improvement on the scores. It is expected that with more time, we could have tested more parameters and approaches for the neural network aswell as the MNB classifier which most probably would have led to better results.

6 Conclusion

In this work we implemetned two different approaches to solve the multilabel classification task on the EURLEX dataset regarding the EUROVOC descriptors with highly imbalanced class distributions. We used a neural network approach and a Multinomial Naive Bayes approach and found that the MNB performs better. We also applied various measures to handle the imbalances by penalizing the loss function and using under- and oversampling. We found that undersampling gives better results in our case. Also we introduced a one vs. some approach for the MNB classifier which could not outperform the one vs. all approach.

The main challenge of this project was the limited time and hardware resources for the quite

intensive model training workload. The neural network was trained for 24h on each dataset which resulted in only 3 epochs. Similarly problems occurred with the MNB model where it took around 36h to train all the 4000 models on 19000 documents each in the one vs. all approach. That time limit also hindered us from trying the bi-gram representation since the feature space is a several magnitudes larger than the tf-idf representation and hence would probably take around 3-5 days to finish to train all 4000 classifiers.

Other approaches could be implemented with more time and hardware resources. Just running the neural network for more epochs could already drastically increase the performance. We could also increase the model size or use some state of the art neural network architectures like the Transformer [Vas+17]. Further we could compare the penalized BP-MLL to the original BP-MLL to see if the penalization is chiming in with the loss function. For the MNB model we could test other scalar values for the one vs. some approach that come closer to true distribution of the dataset. Using tf-idf values for the bi-gram representations or using trigrams would also be feasible approaches to try.

Another interesting approach that smoothly handles class imbalances would be to divide the labels into buckets based on the frequency of occurrence. So for example all 700 labels that only occur once would be thrown in one bucket and would be assigned a label 'bucket 1'. Then we could build nicely balanced buckets in the top level. So the first classifier would just classify in which bucket the instance is. We then would need to train second level classifiers for each bucket to get the final label. With a clever bucket choice we would end up with balanced datasets on the top level and in the buckets aswell. But here again, for time reasons this is out of scope.

In summary, it can be stated that given the time limitation and the rather long processing time for the models we tried to face the challenges in as many ways as possible but there is still plenty of possibilities for improvement.

References

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[Sas07] Yutaka Sasaki. "The truth of the F-measure". In: *Teach Tutor Mater* (Jan. 2007).

[Vas+17] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[ZZ06] Min-Ling Zhang and Zhi-Hua Zhou. "Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization". In: *IEEE Trans. on Knowl. and Data Eng.* 18.10 (Oct. 2006), pp. 1338–1351. ISSN: 1041-4347. DOI: 10.1109/TKDE.2006.162. URL: http://dx.doi.org/10.1109/TKDE.2006.162.

A Libraries used

BeautifulSoup, NLTK, pickle, re, math, random, tensorflow 1.13.1, https://github.com/vanHavel/bp-mll-tensorflow, scikit-learn, itertools