**Comparison of algorithms classifying text articles including the problem of extreme classification.**

Grzegorz Meller, Mateusz Rock
Gdansk University of Technology, Data Engineering

## 1. About the project

The goal of our project was to implement different classification algorithms, for article classification, and compare results of their outcomes. For this task we used Reuters News dataset from Kaggle website: https://www.kaggle.com/astoeckl/newsen.

The second goal was to check if selected algorithms (used for classifying Reuters News articles) would be able to perform so called “extreme classification” on a very large dataset. For this we used wiki10-31k dataset taken from Manik Varma website: http://manikvarma.org/downloads/XC/XMLRepository.html.

## 2. Selected classification algorithms

In the first stage of our project we decided that we will implement popular and very often presented in literature algorithms used for data classification. These are the following:

- K-nearest neighbors (kNN)
- Naive Bayes classifier
- Support Vector Machines (SVM)
- Artificial Neural Network

The detailed working explanation of each algorithm is presented in the report from the first stage of the project. All of this algorithms were used for classifying Reuters News, and all of them gave good results (more on this in the further part of the report).

Parameters of K-nearest neighbors and Artificial Neural Network were selected using error and trial method. For example in kNN we found optimal n number (which happened to be 30) by increasing in each iteration its’ number by one and checking if it gave better or worse results on validating set. In Neural Network we found number of epochs equal 5 in order to get best results on validating set. In order to avoid overfitting we added one Dropout Layer set on 20%. What is more, after research we found that for text classification the best type of neural network is LSTM. Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. It can not only process single data points, but also entire sequences of data.

Unfortunately, for the the dataset Wiki 10-31k, selected algorithms were unable to deal with such amount of data and the problem of multi-label classification. Wiki 10-31k dataset has 14146 rows, assigned to 101938 possible categories. Each time we tried to fit neural network with data from this dataset we were getting errors that more 10% CPU memory would be used, and always refused to work on. Even reducing batch size to 1 and reducing number of layers to minimal was not able to fit the model. So we tried with cutting a little bit the dataset, so that the model could accept it. Unfortunately, in order for the model to adopt the dataset, we had to cut it down very heavily, and once it was able to fit the model, it gave ridiculously low results, which were unacceptable for us.

With this experience we had to do a research on more sophisticated algorithms, used for extreme classification. In the extreme classification repository, we have seen that LEML algorithm is able to perform efficiently this kind of tasks. Because it is very complicated algorithm, we decided not to write it by ourselves from the beginning, but use ready implementation found on github: https://github.com/AnthonyMRios/leml. Also explanation on how LEML works can be found here: http://proceedings.mlr.press/v32/yu14.pdf.

**3. Data Preprocessing**

At the beginning, both of the datasets were divided into training, validation and test sets. The proportions for the divisions are as follows: training set - 60%, validation set - 20%, test set - 20%. Training set is used to fit the parameters of the model.The model is trained on the training dataset using a supervised learning method. Validation set is used to tune the hyperparameters of a classifier. For example, in neural networks, a hyperparameter can be, the number of hidden units. Test set is used for final model evaluation, which is not done on validating set in order to prevent overfitting.

To fit algorithms we had to transform text from articles into numerical values. For this we used **tf-idf vectorizer**. It is intended to reflect how important a word is to a document. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

For neural network, we had to convert input and output into numerical matrices. For input we used Tokenization. Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens and than putting them into appropriate numpy array. For output we used **One-hot encoding** method. It creates a matrix, each column represents specific label, and in rows there are zeros and ones. Zero represents that row does not belong to specific category and one, that it belongs.

For wiki10-31k dataset we didn't have to vectorize input and output, because dataset provided by extreme classification repository was already fully prepared in form of numerical values. The only task we had here, was in a few for loops split provided text document into features and labels, by searching in each row for characters like ‘,’ or ‘:’ and put into appropriate numpy arrays.

**4. Selected evaluation method**

We are measuring our results from created models with the F1 score. It considers both the precision and the recall of the test to compute the score. Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive. Recall actually calculates how many of the actual positives our model capture through labeling it as positive.
Formula:

$$F_1 = \left( \frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

**5. Results**

Reuters dataset

| Algorithm | F1 score of validating set | F1 score of testing set | Time |
|---|---|---|---|
| kNN (n=30) | 0.86 | 0.78 | 41 seconds |
| Naive Bayes | 0.82 | 0.74 | 12 seconds |
| SVM | 0.88 | 0.80 | 705 seconds |
| Artificial Neural Network (epochs=5) | 0.80 | 0.72 | 616 seconds |

Wiki 10-31k dataset

Leml algorithm was learning on training data for approximately 2 hours (exactly 6834 seconds), using 100% CPU power and returned the following precisions on validating set, which implementation were added by the authors of leml algorithm:

Precision 1: 0.39
Precision 2: 0.31
Precision 3: 0.25

We also added implementation for calculating F1 score (from sklearn library), and after compiling it gave approximately 0.52 score on testing set.

**6. Comments on the results**

On the Reuters dataset the best result (0.80) was obtained by the SVM algorithm , but it took the longest time (705 seconds). The second best result (0.78) was obtained by kNN algorithm and it took only 41 seconds, so about 17 times faster than the SVM.The third result (0.74) was obtained by Naive Bayes and toke 12 seconds.The worst result (0.72) was obtained by the neural network and it took 616 seconds.

The worst algorithm assuming the time and test result is the neural network. Neural networks require more data preprocessing steps, than the other algorithms that we have used. There is no optimal rule for determining the structure of artificial neural networks. A good neural network models require bigger amounts of data to be trained on, probably that's why it got the weakest results.

The best algorithm assuming the time and test result is the kNN algorithm. One of the reasons why it works so well can be the fact that kNN does not build any model, it simply tags the new data entry based learning from historical data. New data entry would be tagged with majority class in the nearest neighborhood. The kNN algorithm on the other hand is also computationally expensive, because it has to store all training examples and compute the distances to them, but the Reuters dataset seems to be small enough for the algorithm to cope with the disadvantage.

On the Wiki 10-31k dataset we used only the LEML algorithm , because the other algorithms were not able to accept such a large set of data. LEML model gave a surprisingly good result, but on the other hand it used a lot of computing power and it took a lot of time for training.

**7. Summary**

Basing on our results from the 5 algorithms, we can say that algorithms like SVM, kNN, Naive Bayes and Neural Network can with ease work on dataset that has about 40000 articles with one category assigned to each. They fail when it comes to the extreme data classification and large datasets. To sum up, we made the main task of the project to categorize the extreme dataset wiki 10-31k, but we didn't use algorithms, which at the beginning we assumed we would use for this task.