Hate speech classification

Paweł Flis, Karol Krupa, Rafał Chabasiński

June 2022

1 Introduction

Social media platforms such as Facebook, Twitter, Instagram and YouTube are experiencing a continuous growth in number of users. In recent years, partially due to COVID-19 pandemic even older generations turned to those platforms as a way to experience wider range of human interactions. Along with all the advantages social media brings, there are number of hazards related to harmful actions of some of its users encouraged by their perceived anonymity.

Especially recently hate speech has been under the scope of investigation to make the internet a safe and welcoming space for everyone. This topic garnered scientific attention due to the potential of applying machine learning for hate speech recognition, which would in turn help counteract cyberbullying. Although the very idea of using automatic censors is highly questionable their counselling can be of invaluable aid. In this case the hate speech detector accuracy is of the essence as too rigorous model could severely affect freedom of expression while high-tolerance one would maintain unsafe environments.

2 Literature analysis

Number of Natural Language Processing in conjunction with Deep Learning solutions have been proposed during the years, however majority of these solutions requires great amount of labelled data, which has to be manually labeled by a human, which eventually limits the cognitive capabilities of the model. It is also really hard to identify semantic nuances which often decide on the overall sentiment. There have been attempts to tackle the problem using statistical models such as n-grams, SVM or Logistic Regression. These methods require vector as an input, so the corpus of text has to be converted earlier - this is commonly done using methods such as word2vec encoder using semantic similarity of words or doc2vec for entire sentences.

The task at hand is hate speech detection in polish twitter data. Direct solution to the same problem were presented during PolEval2019 [5]. At a time statistical models such as SVM with tokenization or even Deep Learning solutions such as LSTM were able to achieve only minor improvements over baseline classifier which always predicts most common class.

Bidirectional Encoder Representations from Transformers model has shown a great prowess in various NLP tasks, in many cases outperforming its predecessors. Mozafari et al. [3] proposed a semi-supervised solution based on pre-trained BERT model with broad contextual knowledge and using transfer learning to fine tune a model on labelled samples from target domain of hate speech.

This approach considers two variants of BERT networks. $BERT_{base}$ having 12 layers and 110 million parameters and $BERT_{large}$ having double the layers and triple the parameters. The

output is a sentence representation in a 768-dimensional vector, which is an input to domain specific classifier.

In paper number of fine-tuning approaches are presented:

1. **BERT based fine-tuning**
   $BERT_{base}$ model with one fully connected layer at the top and softmax activation function with no hidden layers.

2. **Insert nonlinear layers**
   $BERT_{base}$ model with multilayer perceptron with two hidden layers at the top level.

3. **Insert Bi-LSTM layer**
   $BERT_{base}$ model with Bi-LSTM recurrent neural network and one fully connected layer on the output.

4. **Insert CNN layer**
   Output of each transformer from $BERT_{base}$ is concatenated to produce a matrix that is an input to CNN, later max pooling is performed and output is deduced using one fully connected layer once agian.

The authors performed data pre-processing to make the text input data more suitable for BERT model - emoticons were changed into <tags>, multiple word hashtags were separated and everything has been turned into lowercase. It is important to notice, that by doing that some relevant information may be lost, however we trust that it is a calculated decision and it is of benefit to used BERT model.

Version with CNN layer proved to perform the best across all metrics (Precision, Recalll, F1-score). It outperformed other solutions designed for the very same datasets used in [2] [7] [8]. It is worth noting that BERT with nonlinear layers performed surprisingly poorly(much worse than with no hidden layers) on dataset with samples labelled as Hate[1430] Offensive[19190] Neither[4163] so dataset heavily biased towards 'just' offensive content.

3 Exploratory data analysis

We've acquired our data from `http://2019.poleval.pl/tasks/task6`.
Training data was taken from section Task description, Task 6-2: Type of harmfulness. Test data was available in the later section 'Test data'.

The data row comprised of two elements - sentence and its classification to one of three classes. The classes were: 0 (non-harmful), 1 (cyberbullying) and 2 (hate-speech).

The condition to distinguish cyberbulling from hate speech was whether the harmful action is addressed towards a private person(s) (cyberbullying), or a public person/entity/large group (hate-speech).

3.1 Data examples

- non-harmful

    - anonymizedaccount anonymizedaccount Zaległości były, ale ważne czy były wezwania do zapłaty z których się klub nie wywiązał.

    - anonymizedaccount Tym w sejmie też? Banda hipokrytów i złodziei.

- cyberbullying

    - anonymizedaccount anonymizedaccount Czas pokaże kto jest kim, mącicielu, szkodniku, haha do gazu haha

    - anonymizedaccount anonymizedaccount anonymizedaccount Pis mial być inny, jesteś idiotą

- hate-speech

    - RT anonymizedaccount anonymizedaccount Pasujecie do siebie antypolskie kanalie.

    - anonymizedaccount Pierwszy priorytet zadowolić Jarkacza, drugi priorytet ssać kasę wszystkimi otworami, mają rozdwojenie jaźni

We can see that subjects are anonymized and are easily recognizable, which will be helpful for our algorithms.
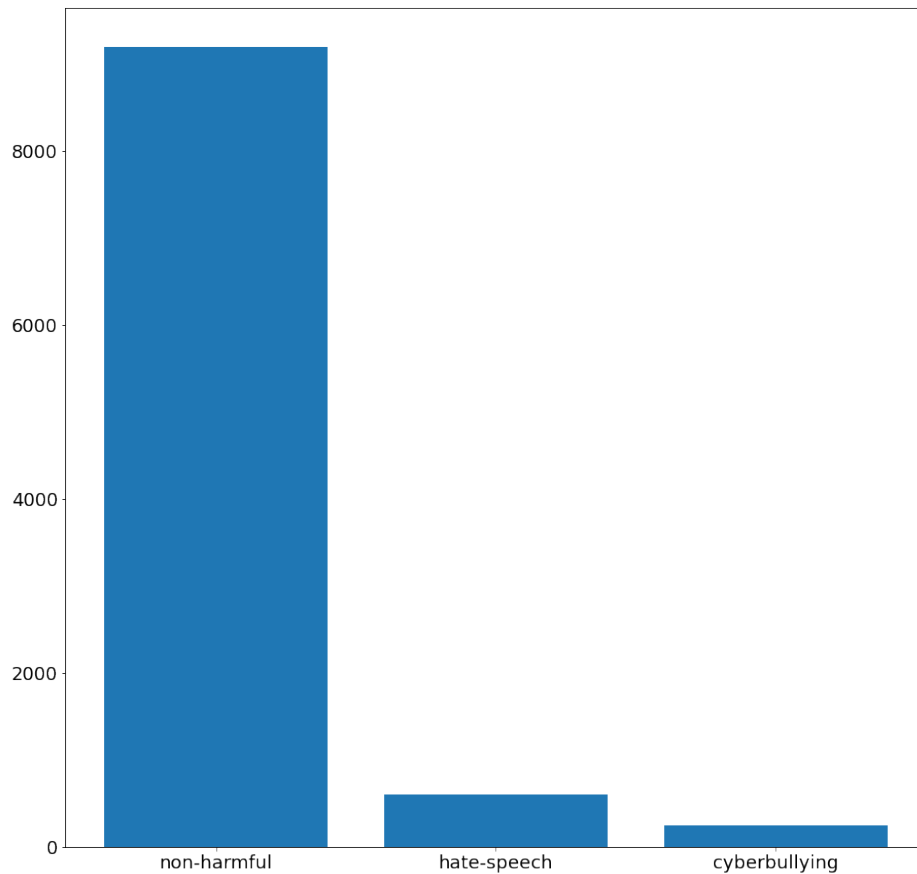
3.2 Label data split

Figure 1: Labels distribution.

In 1 we can see that distribution of classes in the dataset is very unbalanced. We'll have to consider that when creating our algorithm, by e.g.

- shrinking bigger data sets to the smallest one

- applying bigger weights to smaller datasets
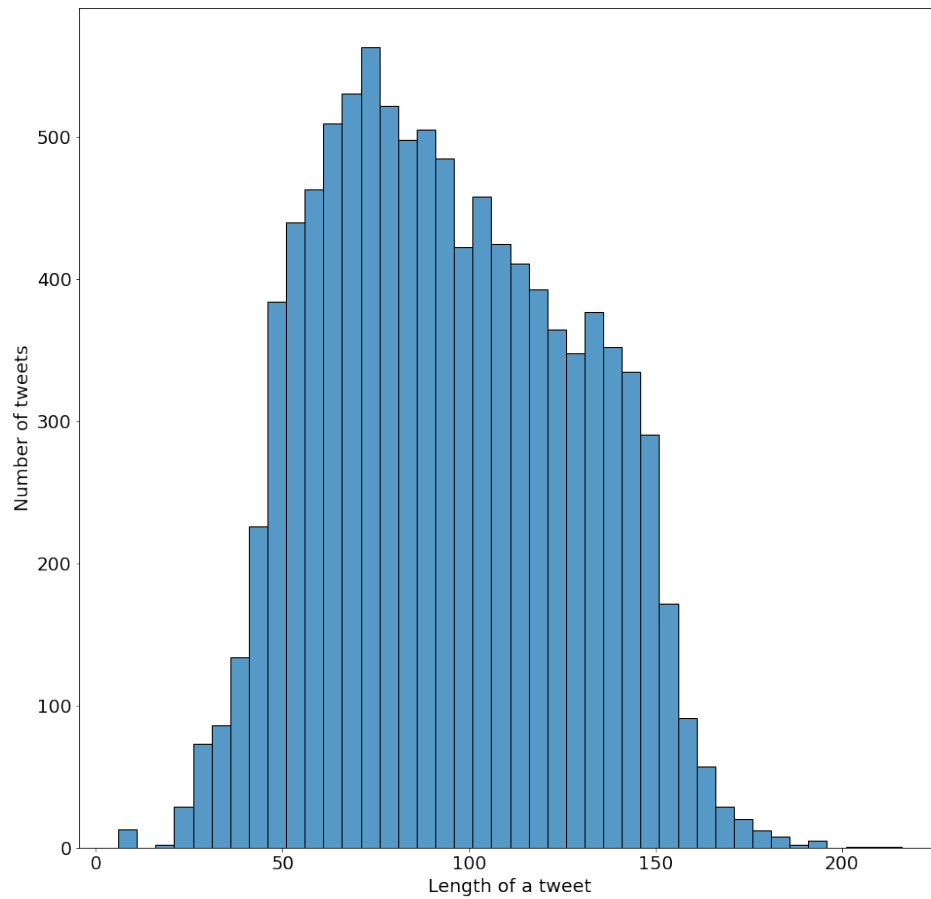
3.3   Data attributes

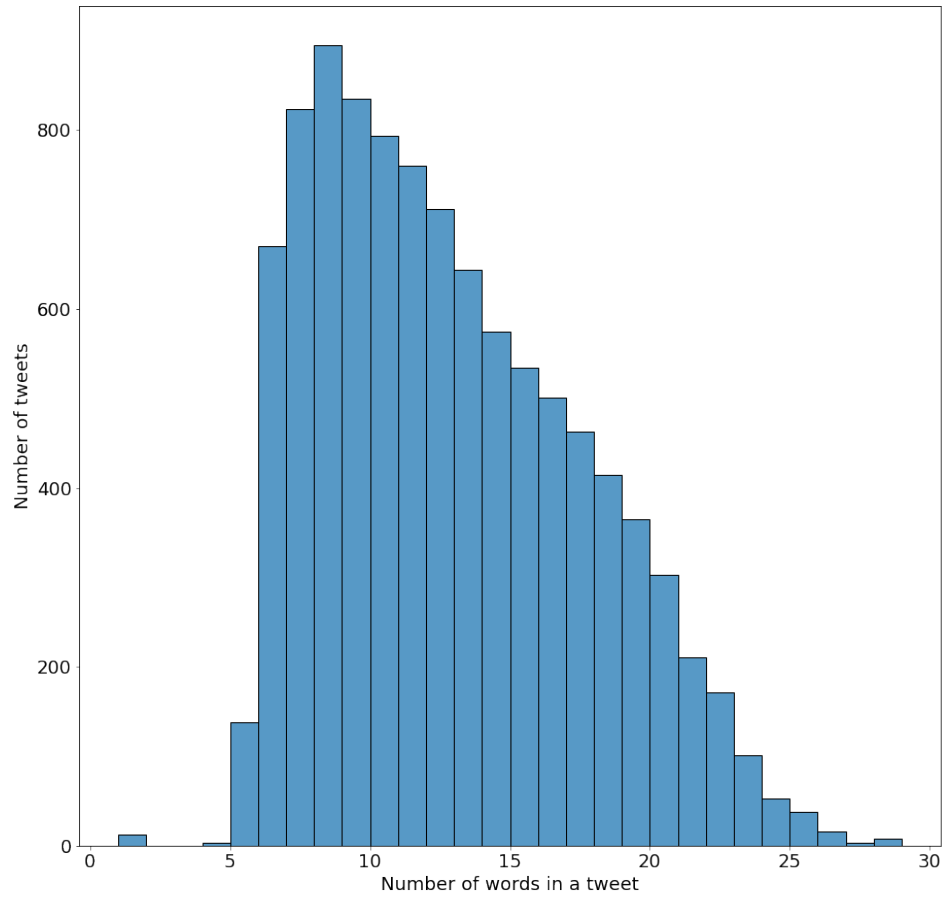Figure 2: Tweets lengths (i.e number of characters in a tweet).

Figure 3: Tweets word counts.

We created histograms for length of a tweet and number of words in a tweet.
In both graphs (2 and 3) we can see some outliers in the data. In order to simplify our task and make our solution more reliable we should consider removing them.
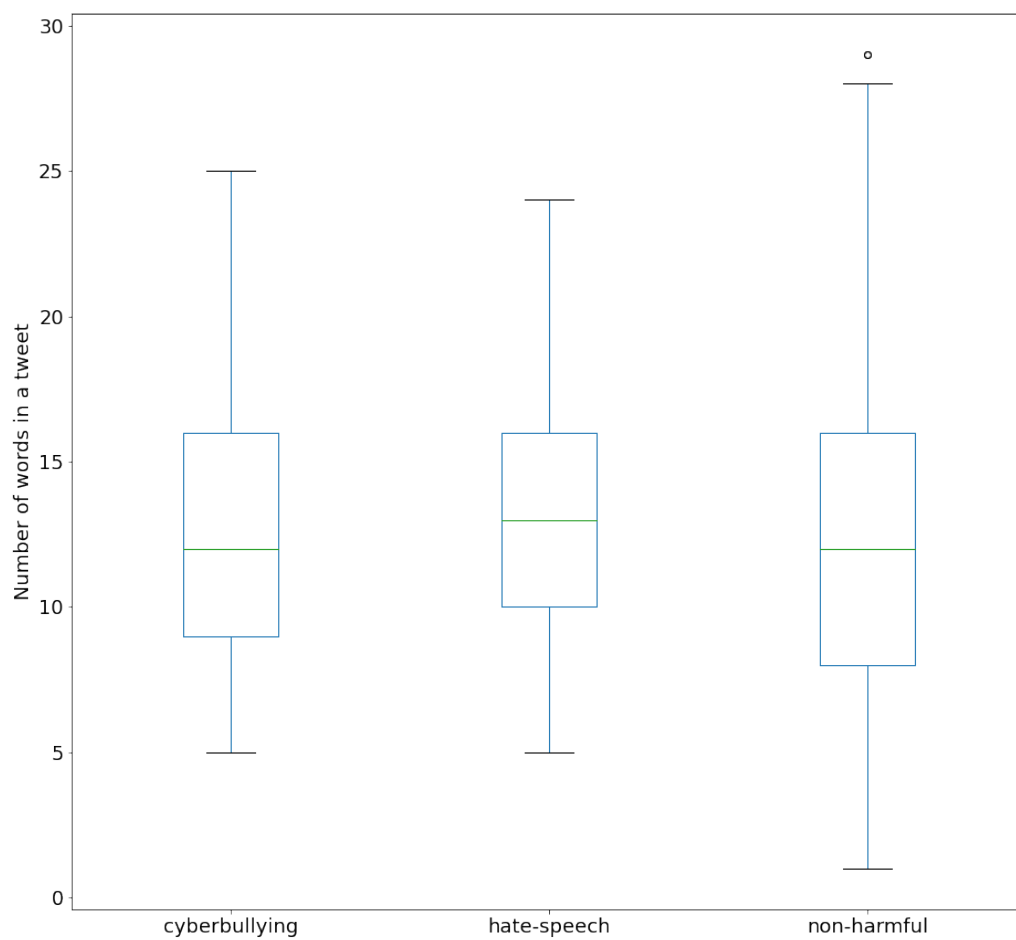
Figure 4: Tweets word counts box plot for different labels.

We've created boxplots for number of words in a tweet for all classes 4.
Unfortunately, different labels have quite similar expected values for number of words, so this statistic most likely won't help us.

3.4 Word cloud

Figure 5: Word cloud.

In 5 we can see that the most important topics in our dataset are football and politics. There are also some curse words which may help distinguish hate speech.

Some one letter words and specific stop words can be distinguished, so removing them might further help us remove noise from the data.

4 Solution concept

Preparing our solution to the problem is to prepare a program that, given an expression written by a human, returns a result of 1 if the statement is hate-filled and 0 otherwise.
In order to perform this task as well as possible, we need to:

1. Properly transfer expressions and sentences into a form understandable by a computer - that is, vectorize them using an appropriate tokenizer.

2. Model the language and its internal dependencies using certain language models.

3. Train the language models for our task, which is hate speech recognition.

The first two steps can be solved by choosing a pretrained model. A collection of such models is available at this link KLEJ-Benchmark. The first two steps are strongly related, since the trained model has been trained for selected embeddings and is unlikely to work for others.

An example model from the list above is HerBERT [4] and XLM-RoBERTa [1]. Both score very high on the CBD task [6], which is the task we solve. For this reason, we decided to select pre-trained models and then retrain them for our purposes. In addition, for the second model, the authors provided scripts that give the ability to train the model to a specific KLEJ-benchmark problem, including the CBD problem. The scripts are available here: scripts.

The computations will be done with google cloud graphics cards, on the google collab site. Verification will be done using Precision, Recalll, F1-score metrics.

In addition to this, research will be conducted on simpler models such as decision trees, SVMs, and naive Bayes. Various embedders such as count vectorizer, tfidf and spacy embedders will be tested.

5 Solution

5.1 Data preprocessing

During the development of the solution, a significant element was the proper preprocessing of the data. The proposed solution performs three-stage data processing:

1. Optional stemming of words in sentences using stemmer.

2. Main sentence preprocessing module consisting of:

   (a) Lowering letters
   (b) Unidecoding
   (c) Removing @anonymized_account phrase, dots and commas

3. Removing stopwords

The reprocessed datasets are stored in the solution repository in folder data in files: test_preprocessed.tsv, train_preprocessed.tsv, test_preprocessed_stem.tsv, train_preprocessed_stem.tsv.

5.2 Decision tree

At the beginning of the preparation of the machine learning solution to the problem posed, we started the tests by building a model closest to the rule-based system. The choice was to build a decision tree based on CountVectorizer with 1 element ngrams. We ran a series of tests in which we examined the effect of tree depth on the results obtained. Results can be seen in figure 6.
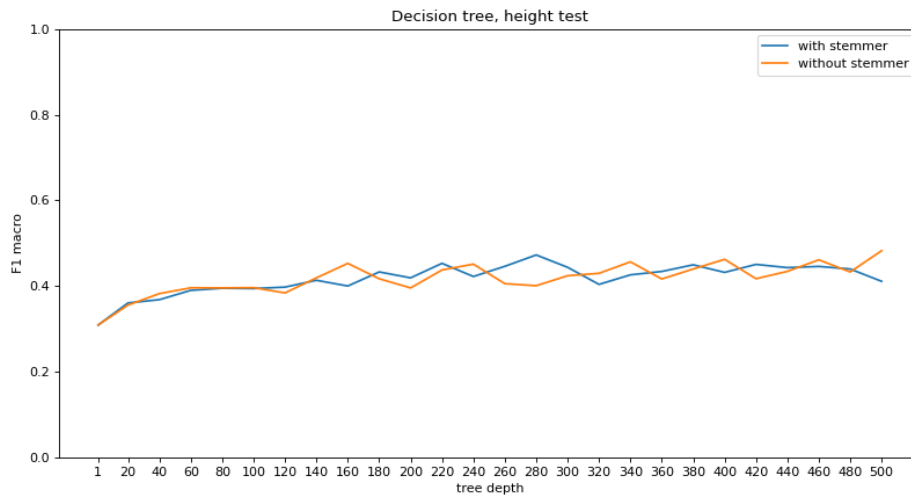

Figure 6: Decision tree height comparison

Best obtained result for stemmed and not stemmed data are grouped in the table 1.

The performance of the decision trees increases during the initial phase of increasing the tree height, however, the fluctuations are quite high thereafter. This does not change the fact that

No.	Stemmed data	Best tree height	F1 macro	F1 micro
1	true	280	0.473	0.868
2	false	500	0.483	0.87

Table 1: Decision tree height comparison, results

good results have been achieved with the decision tree. However, it should be kept in mind that this solution is very dependent on the test set and does not take into account the meaning of the words, which can have very negative consequences when the vocabulary changes

As part of the development of the solution, we have also prepared an illustration showing how the decision making is done for an example tree of height 3. The result is available in figure 7.
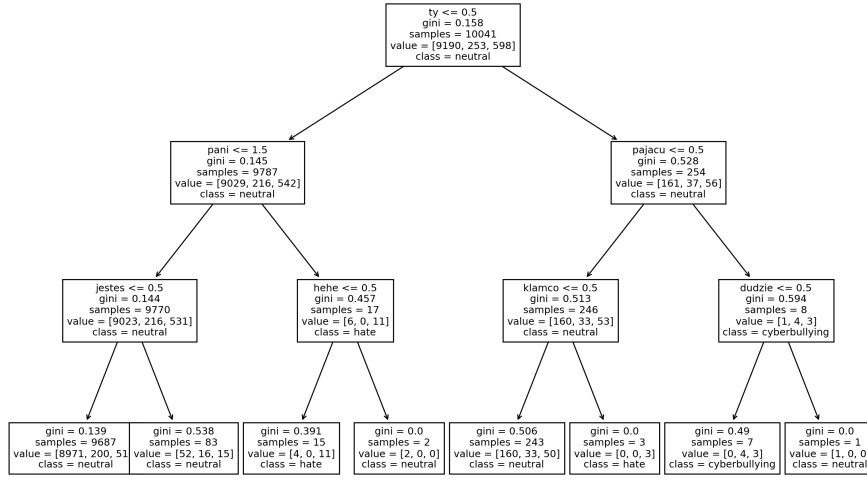

Figure 7: Example decision tree with depth = 3 and F1 macro 0.309

5.3 SVM and Naive Bayes

During the development of the solution, we experimented also with SVM models with three different kernels: RBF, Poly, Linear. In addition, we also considered the Naive Bayes model. To analyze these solutions, we proposed a text vectorizer computed from 1 or 2 element ngrams. Then the 5 or 10% most significant elements were selected. Moreover, each experiment was performed twice - for the set with and without the applied stemmer. The collected results can be found in the table 2.

It turns out that Linear SVM using 1 element grams and a 5% selector on the data with the stemmer applied, turned out to be the best solution from this set. In addition, it can be inferred

No.	model	ngram	selector_percentile	F1 macro (S)	F1 micro (S)	F1 macro	F1 micro
1	SVM RBF	1	5	0.365	0.873	0.350	0.871
2	SVM RBF	1	10	0.367	0.864	0.369	0.861
3	SVM RBF	2	5	0.399	0.869	0.403	0.871
4	SVM RBF	2	10	0.351	0.864	0.356	0.866
5	SVM Poly	1	5	0.398	0.875	0.366	0.873
6	SVM Poly	1	10	0.322	0.867	0.321	0.867
7	SVM Poly	2	5	0.339	0.87	0.339	0.869
8	SVM Poly	2	10	0.322	0.868	0.322	0.868
9	**SVM Linear**	**1**	**5**	**0.409**	**0.877**	**0.375**	**0.874**
10	SVM Linear	1	10	0.379	0.871	0.357	0.868
11	SVM Linear	2	5	0.389	0.872	0.389	0.872
12	SVM Linear	2	10	0.353	0.869	0.365	0.871
13	Naive Bayes	1	5	0.322	0.868	0.322	0.868
14	Naive Bayes	1	10	0.344	0.87	0.349	0.87
15	Naive Bayes	2	5	0.322	0.868	0.322	0.868
16	Naive Bayes	2	10	0.364	0.87	0.364	0.87

Table 2: Different SVM and Naive Bayes settings, results, S - stemmed data

from the table that only in five cases we obtained an improvement in performance after using stemmer. These are the rows 1, 5, 6, 9, 10.

5.4   Multilayer perceptron

The next stage in the development of our solution was to prepare an MLP network with a structure consisting of the following parts:

- Linear layer of size input_size x 200

- Dropout layer 0.2

- ReLU

- Linear layer of size 200 x 50

- Dropout layer 0.2

- ReLU

- Linear layer of size 50 x 3

- Softmax

During the development of the solution, we prepared an environment to train MLPs along with various vectorizers. We tested the following text vectors as part of our testing:

- CountVectorizer with ngrams of size 1, with selector 5%

- CountVectorizer with ngrams of size 2, with selector 5%

- TfidfVectorizer

- Spacy pl_core_news_md vectorizer

- Spacy pl_core_news_lg vectorizer

Each of these tests was performed on stemmed data and without applied stemmer. Each test was run with the Adam optimizer, with an initial learning rate of 1e-4, at 20 epochs, with the CrossEntropy loss function. Training data were reported with batches of size 10. We also implemented a custom subsampler that rejects 80% of sentences with a label 0 (neutral sentence) - this operation was needed, otherwise the network learned to predict only one class. Also, in the beginning of every experiment random seed was set to same value (1).

The collected test results are shown in the table below.

No.	Vectorizer	F1 macro (S)	F1 micro (S)	F1 macro	F1 micro
1	CountVectorizer, ngram=1, percentile=5	0.472	0.874	0.485	0.876
2	CountVectorizer, ngram=2, percentile=5	0.441	0.874	0.427	0.877
3	**Tfidf**	**0.481**	**0.815**	**0.500**	**0.828**
4	Spacy, pl_core_news_md	0.438	0.816	0.430	0.787
5	Spacy, pl_core_news_lg	0.435	0.800	0.436	0.803

Table 3: MLP with different vectorizers, results. S - stemmed data, results for best F1 macro score epoch

The best solution turned out to be the one using the tfidf vectorizer on the data without the stemmer applied.

The figure 8 shows the training processes for each experiment.

One can see on the graph that the training process varies between methods up to about 5 epochs. After that, the process is similar, but the differences in F1 macro statistics are about a few percentage points.

5.5 Deep learning

To solve the task using deep learning methods, we used RoBERTa deep machine learning models along with scripts prepared to train them for the KLEJ benchmark prepared by Allegro. The available scripts are here.

During the development of the solution, we used the Collab Pro environment (Google's computing cloud) and the Tesla graphics card that was assigned to us there.

Solution preparation consisted of:

1. Prepare the training and testing collection for the CBD task format from the scripts described above. Their Internet hate detection task involved distinguishing between neutral and hate speech. In our case, we have 3 classes: hate, cyber-bullying, and neutral. Moreover, the sets for this task appear in two files: tags and text. Obtaining the intended format required combining these files in tag tab text format.

2. KLEJ benchmark script modification. The Task CBD described above had 2 categories and was not set up for post-training testing. Because of this, it required code modification. The modified tests.py file is available in the solution repository.
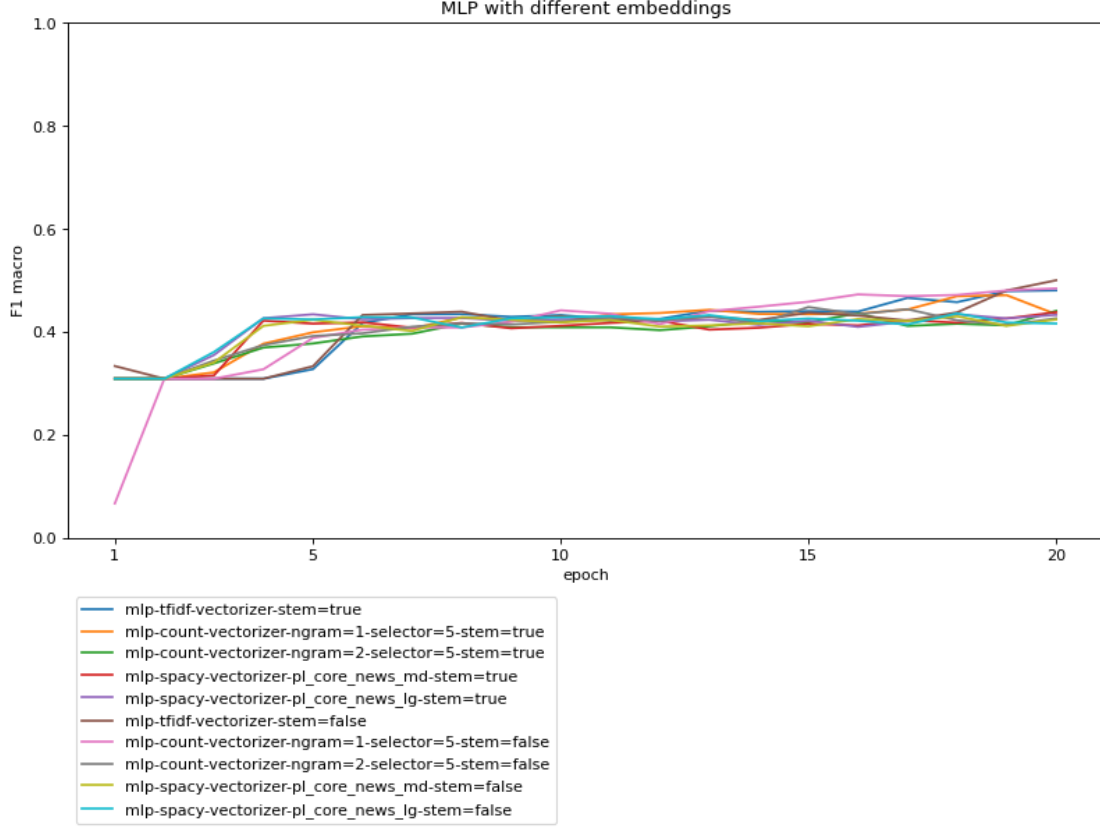
3. Downloading models on Google Collab.

13

Figure 8: MLP training processes

4. Beginning model training.

5. Generating labels for test data.

6. Running evaluating script.

We tested two models while conducting the experiments. The first one is RoBERTa-large-v2, a model trained on a corpus of about 200GB of data. When training this model, we used the following python script call:

python run_tasks.py –arch roberta_large –model_dir roberta_large –train-epochs 2 –tasks KLEJ-CBD –fp16 True –max-sentences 8 –update-freq 4 –resample 0:1,1:1,2:1

The training process took approximately 320 seconds (2 epochs) on a Tesla P4 16GB.

The parameters mean that we used the roberta_large architecture, the model is in the roberta_large folder, we want to train the model for 2 epochs, the task we are doing is TASK-CBD, we want to do the calculations with less precision (meaning they will be done faster), the maximum number of sentences in the GPU at one time is 8, the update rate is 4, the input resampling is 1 for class 0 (neutral), 1 for class 1 (cyber-bullying), 1 for class 2 (hate).

Next, we tested the samplings as 1:15:10 and 0.75:5:3. In addition, we tested the performance of the RoBERTa-base model trained on a 20GB body, which is 10x smaller than the RoBERTa-large-v2.

	micro-average F-score	macro-average F-score
base-075-5-3	88.10	57.24
large-v2-075-5-3	90.00	58.63
large-v2-1-1-1	90.70	51.48
large-v2-1-15-10	89.50	54.21

Table 4: RoBERTa with different subsampling methods, results

The results obtained can be found in 4.

Accuracy was about 90% in each case, however, it is not very reliable due to the huge advantage of the class of neutral statements.

The results obtained by the script authors were 66.77% and 74.16% for the base model and the large model, respectively. Note that the task they performed is significantly simpler, because they only had to distinguish between negative and neutral statements. In our case there is a neutral statement and a negative statement in two variants, which is a more difficult task.
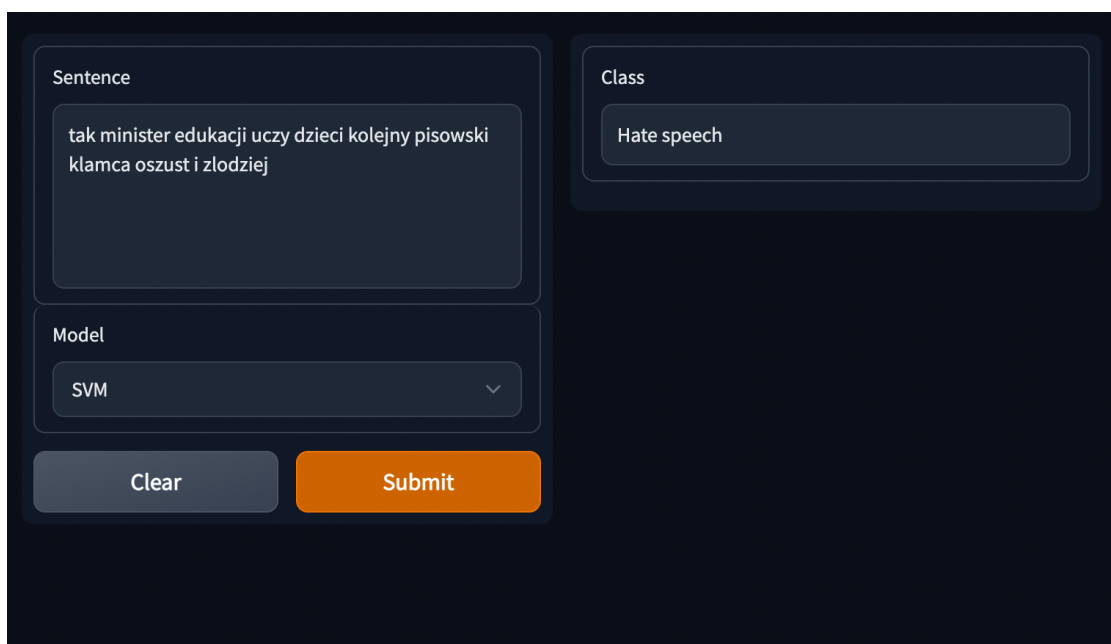
We conducted an experiment using the RoBERT-large-v2 model along with 0:0.75, 1:3 resampling (used by the authors) for a class 2 CBD task. The results obtained are: precision 76.64%, recall 61.19%, balanced F-score 68.05%, accuracy 92.3%, so much better than in case of solution for 3 class problem.

6   Deployment

A user interface application was prepared to manually test and use the models we prepared earlier. It was created using gradio library which allows for straight forward creation of ML demos. Previously trained models have been stored in the cache and when you select a model, the corresponding one is loaded. The user can enter any sentence and after pressing the "Submit" button, the classifying method is called and the information whether the sentence is Neutral, Cyber-Bullying or Hate speech is displayed. To run the application first install requirements:

```
$ pip install −r requirements.txt
```

Then run the only cell in python notebook file *deployment.ipynb*. Application will be simultaneously deployed on local http port.

Figure 9: User Interface application

7 Summary

The presented problem of sentence classification in the perspective of neutral/cyberbullying/hate classes consisted in evaluating to which class a given expression belongs.
In the literature analysis section, we analysed similar problems (to ours) in the literature. We've also described latest solutions currently used in natural language processing when solving similar tasks.

The data prepared for the task was analyzed in these regards: proportion of different classes in the set, lengths of tweets, numbers of characters and the numbers of words. We also checked how the number of words in a tweet depends on the class. Then, an analysis was done on the most frequent words in the text after removing the stopwords.

The next stage of the work was to prepare a machine learning solution performing the classification of sentences into netural/cyberbullying/hate classes. First we prepared data for training and testing in both versions - with and without stemming. The first model we tested was a decision tree, which allowed us to build a rule-based system. Here we analyzed the CountVectorizer embedder with 1 element ngrams and checked how trees behave when their size is changed. We performed tests on data with and without stemming. We also prepared visualization of the tree.

We then ran tests on SVM and Naive Bayes models. Here we prepared an embedder CountVectorizer with ngrams of 1/2 element and a selector of 5/10%. We analyzed the behavior of SVM with different kernels with and without stemming data.

In the next stage of solution development, we tested the behavior of the Multilayer Perceptron model with CountVectorizer embedders with 1/2 element ngrams + 5/10% selector, Tfidf, spacy pl_core_news_md and spacy pl_core_news_lg. For MLP training, we also needed to perform subsampling of the training set.

In the final stage of development, we tested the RoBERT model using scripts prepared by Allegro for the hate classification task, but based on 2 classes. We adapted the code to our 3 class problem and ran tests for the RoBERTa-large and RoBERTa-base model with different subsampling options for the classes.

The problem with CountVectorizer-based embedder solutions is that they depend on a set of training words - word meanings are not taken into account. More complex solutions take this into account. Deep learning-based solutions perform much better than models such as SVMs or decision trees, but the process of training them takes much longer, and the environment itself requires GPU hardware. RoBERTa deep learning model is a complex model with many parameters, so it can model language dependencies in an advanced way. For this reason, it performs much better than other solutions.

The project was able to present a number of solutions to solve the problem of classifying sentences into netural/cyberbullying/hate classes. Tests of these solutions have been conducted. In the further development of the solution many possibilities arise such as different processing of the input data, testing other embedders and checking the performance of other deep learning models.

References

[1]   Sławomir Dadas, Michał Perełkiewicz, and Rafał Poświata. "Pre-training Polish Transformer-Based Language Models at Scale". In: *Artificial Intelligence and Soft Computing.* Springer International Publishing, 2020, pp. 301–314. ISBN: 978-3-030-61534-5.

[2]   Thomas Davidson et al. "Automated Hate Speech Detection and the Problem of Offensive Language". In: *ICWSM.* 2017.

[3]   Marzieh Mozafari, Reza Farahbakhsh, and Noël Crespi. "A BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media". In: *COMPLEX NETWORKS.* 2019.

[4]   Robert Mroczkowski et al. "HerBERT: Efficiently Pretrained Transformer-based Language Model for Polish". In: *Proceedings of the 8th Workshop on Balto-Slavic Natural Language Processing.* Kiyv, Ukraine: Association for Computational Linguistics, Apr. 2021, pp. 1–10. URL: https://www.aclweb.org/anthology/2021.bsnlp-1.1.

[5]   Maciej Ogrodniczuk and Łukasz Kobyliński, eds. *Proceedings of the PolEval 2019 Workshop.* Warsaw, Poland: Institute of Computer Science, Polish Academy of Sciences, 2019. URL: http://2019.poleval.pl/files/poleval2019.pdf.

[6]   Michal Ptaszynski, Agata Pieciukiewicz, and Paweł Dybała. "Results of the PolEval 2019 Shared Task 6: First Dataset and Open Shared Task for Automatic Cyberbullying Detection in Polish Twitter". In: Institute of Computer Science, Polish Academy of Sciences, 2019, p. 89.

[7]   Zeerak Waseem and Dirk Hovy. "Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter". In: *NAACL.* 2016.

[8]   Zeerak Waseem, James Thorne, and Joachim Bingel. "Bridging the Gaps: Multi Task Learning for Domain Transfer of Hate Speech Detection". In: 2018.