NLP - Hate speech classification

Adrianna Klimczak, Adam Kowalczyk and Marcel Wenka

June 5th, 2022

1 Introduction

The goal of this project is to build a machine learning model for automatic cyberbullying detection and classification of the hate speech. The goal of this first milestone is to present the findings of literature and dataset analysis along with the proposed solution of the problem.

2 Problem overview

While social media offer many great communication opportunities, they also increase the vulnerability of young people to threatening situations online. Recent studies [12] shows that among youngsters cyberbullying constitutes an ever growing problem. Hence, automatic cyberbullying detection is also a task of rapidly growing interest. In recent years, the interest in this topic grew significantly in the Natural Language Processing and Machine Learning communities. It is both a challenging and extremely relevant problem. Especially when one takes into consideration how social networks have become a vital part of our lives and how dire the consequences of cyberbullying can be, especially among adolescents [12].

The goal of this project is to build a machine learning model for automatic cyberbullying detection and hate speech classification for polish language. The data will contain tweets collected from openly available Twitter discussions and forums. Final model should be able to distinguish the three possible classes of tweets:

1. 0 (non-harmful)
2. 1 (cyberbullying)
3. 2 (hate-speech)

Since there are many different definitions of hate-speech and cyberbullying there is a need to define the difference between them. The specific conditions on which we will be basing the annotations have been worked in the paper [10]. The main difference that distinguishes the cyberbullying from the hate-speech is towards whom the tweet is addressed. If it is addressed towards a private person then it will be labelled as cyberbullying and if it is addressed towards a public person/entity/large group it will be considered a hate speech [1].

The four basic phases in the hate speech classification problem are:

1. Preprocessing phase
2. Data representation phase
3. Detection phase
4. Classification phase

The first phase and perhaps one of the most important ones is the preprocessing. Among many things it involves: noise separation, labelling the data and extracting the features from raw data [2]. Twitter is a medium that has a lot of messages that can be meaningless so it it crucial to sort them out.

In the data representation phase the creation of model's tweet-topics occurs. It involves the categorisation of tweets into topics using possible different machine learning algorithms. The data preprocessed in this way is used as the training data in the detection phase's algorithms.

In the detection phase the clustering of tweets into the correct topic clusters takes place. The fourth phase is the classification phase and it can contain many possible methods and machine learning algorithms. This phase involves the analysis of the hate speech topics detected earlier and then the distinction of different classes [2].

3 Literature analysis

There is no doubt that research on hate speech classification is much more narrow compared to the social studies of this phenomenon. However, many important advances have been made in recent years. In this chapter, a brief overview of some of the most important natural language processing approaches to hate speech classification is being presented. For a more detailed overview of this problem we refer to the survey paper [2].

One of the most common methods used in NLP for hate speech classification are convolution neural networks. For example, Kern and Winter [15] have developed such model used for multilingual hate speech detection of tweets. The CNN was deployed in order to detect hate speech in Spanish or in English in the messages from Twitter. For feature extraction the model used word embedding technique. In the first place the model detected whether a tweet was hateful speech or not and then it decided on the severity level and target of the tweet. The CNN described in the paper produced very good accuracy and if compared to other baseline classifiers it performed better.

Another such example is Ribeiro and Silva [11]. In the paper they have proposed also a CNN architecture. The model was used for hate speech detection against immigrants and women on Twitter using pretrained word embeddings (GloVe and FastText). Here the model was also multilingual, also for tweets written in English or in Spanish. The task also consisted of two classification tasks. The first was to predict if a multilingual tweet that was targeted against women or immigrants was hateful or not. The second task was whether the target of hate speech was a group of individuals or a single individual. CNN's architecture turned out to perform better for tweets written in Spanish.

Another possible method used for hate speech classification is SVM. For example, it was described by Vega et al. [14]. In the paper they have implemented a SVM classifier to detect and classify hate speech against immigrants and women in Twitter. It was the same problem that was described in [11]. For feature extraction SVM used features representation model. Then the extracted features served as input into the SVM classifier. Superior performance of SVM have been noticed when compared to similar algorithms. Malmasi et al. [7] also described a system that was ensemble-based and used linear SVM classifiers in parallel. The problem was to distinguish and classify hate speech and profanity generally in social media.

Deep learning methods have been widely used in data mining and text classification fields, also for detection, classification and prediction of events like hate speech detection. For example, Komal Florio et al. [3] presented a deep learning approach for hate speech identification in Twitter data. Their approach included CNN and RNN classifier by using biGRU. The results presented in the paper showed that this approach could outperform other classifiers such as logistic regression.

Another possible approach to hate speech classification in Twitter data is to combine word embeddings and deep learning architectures. Word embedding is a specific technique for language modelling. In natural language processing word embedding performs vector representation of words context. There are many word embeddings techniques, some of which include fastText, BERT, Count Vectorizer, Hashing Vectorizer, TF–IDF Vectorizer, Word2Vec etc. Word embedding is used in order to enhance baseline classifiers' performance in fields such as sentiment classification and data analysis.

3.1 Competition results

Specific methods that were the most successful in the competition regarding this tasks (as mentioned in [9]) were based on:

1. SVM

2. a combination of ensemble of classifiers from spaCy with tpot and BERT
3. fasttext

Worth mentioning is the fact that most of the participants used mainly lexical information represented by words (words, word embeddings,tokens, etc.). More sophisticated methods such as feature engineering or incorporating other features such as named entities, parts-of-speech or semantic features were not being applied [9].

4 Dataset overview

The dataset for this project comes from the PolEval competition, 2019, Task 6-2. It contains tweets collected from openly available Twitter discussions and forums. These tweets are subdivided into 3 categories:

1. 0 (non-harmful)
2. 1 (cyberbullying)
3. 2 (hate-speech)

4.1 Data acquisition

First the data was downloaded from the PolEval website and decompressed by using the following commands:

```
wget "http://2019.poleval.pl/task6/task_6-2.zip"
unzip "/content/task_6-2.zip" -d "/content/data"
```

Then it was loaded in python using the pandas library:

```
original_tags = pd.read_csv(
    '/content/data/training_set_clean_only_tags.txt', header = None)
with open('/content/data/training_set_clean_only_text.txt', 'r') as file:
    lines = file.readlines()
original_text = pd.DataFrame(lines)
df = pd.concat([original_text ,original_tags], axis=1)
df.columns =['Text', 'Tag']
```

4.2 Basic dataset information

The data contains 10041 rows of text and assigned classes. The classes distribution presents as follows:

Table 1: Class distribution

Class	Count
0	9190
1	253
2	598

Over 90% of the instances are non-harmful. 2.5% are examples of cyberbullying and the remaining 6% is hate-speech.

4.3 Word counts

The tweets differ in length. Actually, on average the lengths are different in different classes. Figure 1 shows word counts per class (the class is included in the title of the graph).

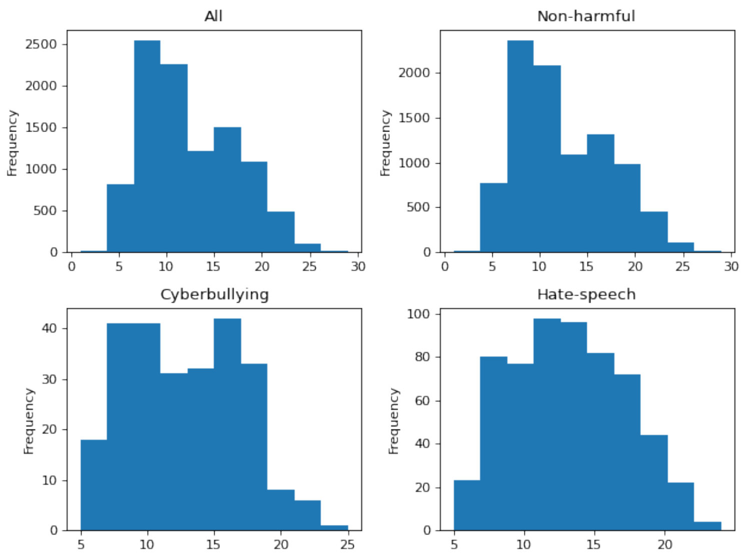


Figure 1: Word counts per class.

On average hate-speech and cyberbullying was composed of more words than the non-harmful tweets. The variation is rather similar. Notably, cyberbullying and hate-speech was never composed of less than 5 words, in contrast to non-harmful tweets, which were shorter on a few occasions.

4.4 Text length

All tweets followed a bell curve (normal distribution) when it comes to the length of text. Figure 2 shows text lengths per class (similarly to before, the class is included in the title of the graph).

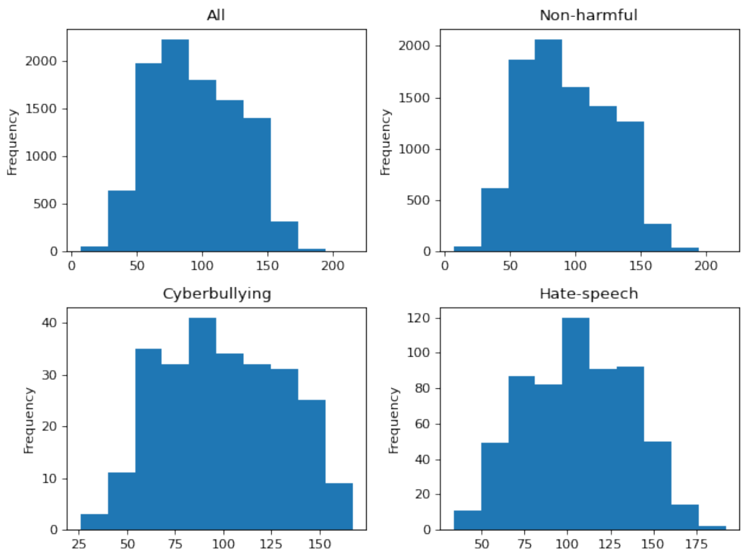


Figure 2: Text lengths per class.

Text lengths of cyberbullying and hate-speech had a bit more variation with means at around 100 characters, whereas non-harmful tweets were usually shorter with the distribution being slightly left-skewed.

4.5 Word lengths

The distributions of word lengths were all left skewed as there were more shorter words. The distributions are shown in Figure 3.

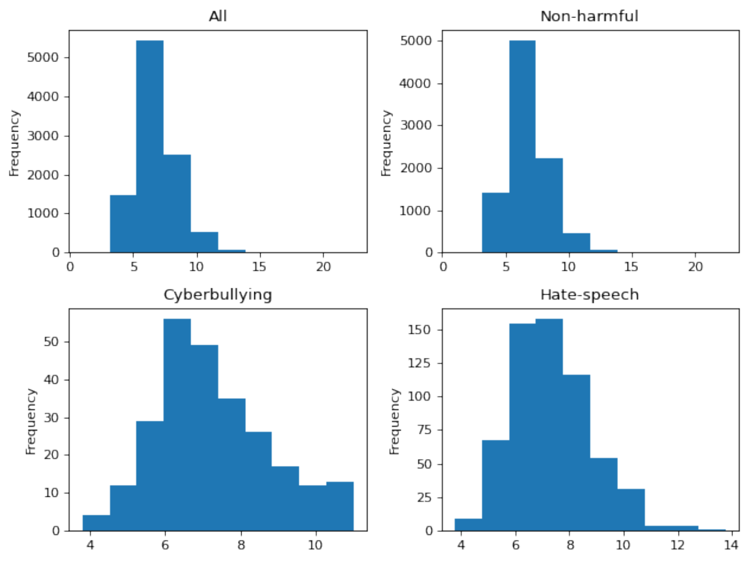


Figure 3: Word lengths per class.

Non-harmful tweets were mostly composed of 5-8 letter word as these are the most common word lengths in polish language [8]. Noticeably, cyberbullying contained more lengthier words. Again, the variation in cyberbullying and hate-speech was greater than in non-harmful tweets.

4.6 Stop words

The words which are generally filtered out before processing a natural language are called stop words. These are actually the most common words in any language (like articles, prepositions, pronouns nd conjunctions) and they do not add much information to the text. By removing these words, the low-level information is removed from text in order to give more focus to the important information. In other words, the removal of such words should not have any negative consequences on the model [5].

The most common stop words in the dataset where presented in Figure 4. These stop words were extracted using the python library spacy with locale set to polish (as shown in the code snipped below).

```
from spacy.lang.pl import Polish
nlp = Polish()
stopwords_count = []
for index, row in df.iterrows():
  for tok in nlp(row['Text']):
    if tok.is_stop:
      stopwords_count.append(tok.lower_)
```

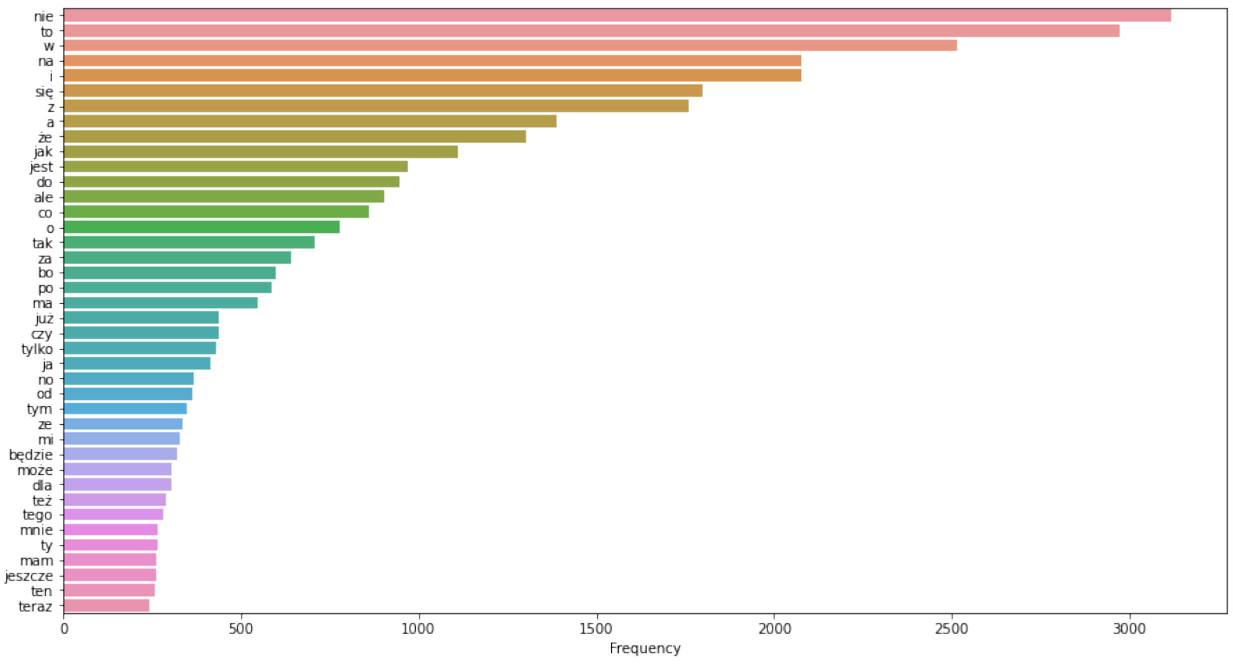


Figure 4: Most common stop words.

4.7 Non stop words

Non stop words are the words that remain after filtering out the stop words and punctuation. As tweets are very concise, because of their limit of 280 characters (which before November 2017 was 140) [4], they contain a lot of emoticons. Figure 5 shows the most common non stop words in all classes.

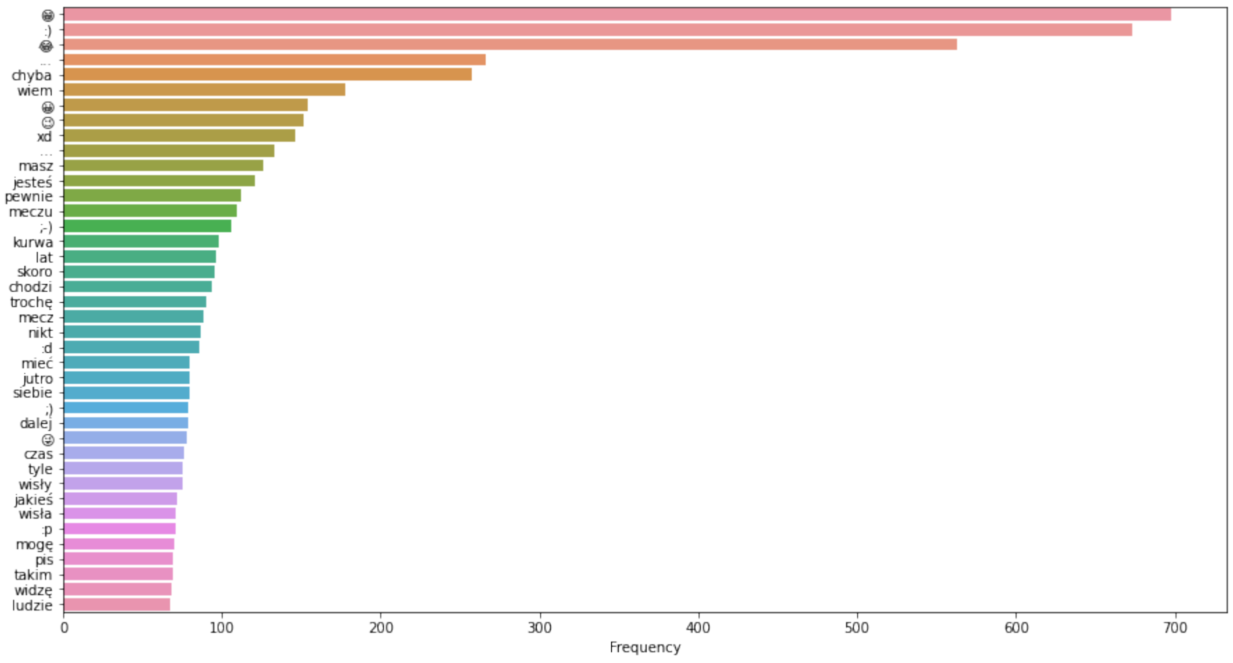


Figure 5: Most common non stop words in all classes.

Notably, some of these most common non stop words are swear words, even though over 90% of the tweets were not harmful. Additionally, it seems like spacy does not treat suspension points as punctuation as they appear as one of the most used tokens.

Figure 6 shows most common non stop words for cyberbullying and hate-speech.

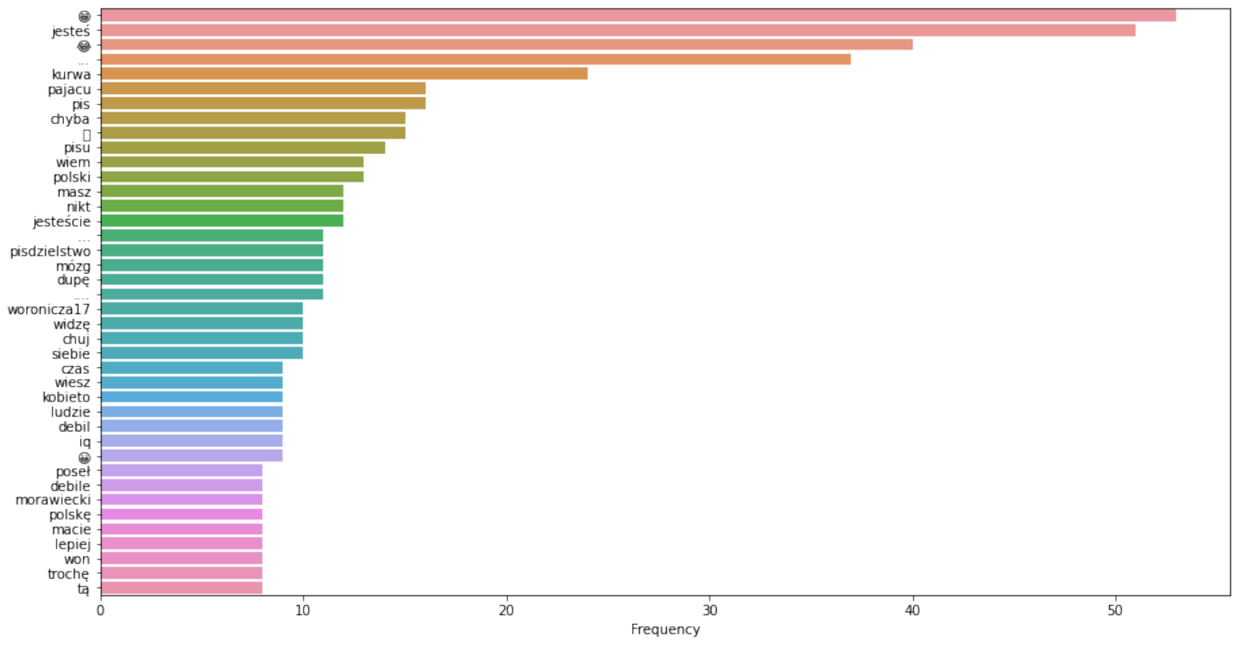


Figure 6: Most common non stop words in cyberbullying and hate-speech.

Obviously, there are much more swear words and they upper way up the list. Additionally, there appear words connected with politics:

- ”PiS” – ruling party in Poland
- ”Woronicza 17” – address of the national TV broadcaster
- ”poseł” – member of parliament
- ”Morwiecki” – polish prime minister

4.8 Preprocessing

Preprocessing included a few steps:

1. removing whitespaces
2. tokenizing the text
3. removing retweets (marked with the ”RT” keyword)
4. removing mentions to other accounts starting or ending the tweet
5. removing punctuation
6. removing stop words
7. converting the text to lowercase
8. removing emoticons

The code for all aforementioned steps is presented below:

```
text = ' '.join(text.split())

doc = [tok for tok in nlp(text)]

if str(doc[0]) == "RT":
    doc.pop(0)

while str(doc[0]) == "@anonymized_account":
    doc.pop(0)
while str(doc[-1]) == "@anonymized_account":
    doc.pop()

doc = [t for t in doc if t.text not in string.punctuation]

doc = [tok for tok in doc if not tok.is_stop]

doc = [tok.lower_ for tok in doc]

doc = [RE_EMOJI.sub(r'', str_text) for str_text in doc]
```

5 Proposed solution

We plan on using word embeddings techniques such as:

- TF–IDF Vectorizer
- Word2Vec

Then we want to compare several machine learning models. We are certainly going to try support-vector machine and convolutional neural networks. We would like to analyse these techniques and come up with the most accurate combination. We might also try some ensembling methods to increase the model accuracy.

Additionally, we do not rule out modifying the preprocessing steps as possibly some additional tweets might get removed and some tokens might be added (for example the @ mention).

6 Extended solution

Our initial solution was inspired by the model which won the PolEval competition. This solution used a combination of non-contextual embeddings to achieve the highest score among the contenders. Later, we realised that BERT was a very fresh idea at the moment of the competition and was probably not as thoroughly tested. We opted for combining embeddings from non-contextual and contextual models in order to utilise the benefits of both.

7 Experiments

Because of the high imbalance in the number of class instances we used micro and macro average F1 score. For the classification task micro average is equivalent to accuracy. However, macro average F1 score (or macro F1 score) is computed by taking the arithmetic mean (unweighted mean) of all the per-class F1 scores. This means that this method treats all classes equally regardless of their support values. In all experiments described in this chapter, we provide the F1 scores for just the test set, while the models were fitted only on the training set.

To cope with the high imbalance of classes all experiments included using weighted categorical crossentropy or other weight-based loss calculation methods.

We started by getting familiarised with the non-contextual models. We used TFIDF to set up a baseline score against which we would later compare our results. We loaded, preprocessed and modelled the data using SVM with a linear kernel. This part did not include many experiments as it was very time consuming to train because of the high number of output columns. The vectors returned by TFIDF had over 20,000 columns. Linear SVM with embeddings from TFIDF obtained micro F1 score of 87.7% and macro F1 score of 49.9%.

Later, we used a contextual model. We downloaded BERT from Darek Kłeczek's github repository [1]. Then, we used a neural network with one hidden layer. Unfortunately, we only managed to obtain macro F1 score of 45.1% and micro F1 score of 82%. This solution did not manage to outperform simple TFIDF model.

At this point we should mention we used a different preprocessing method for BERT and TFIDF. BERT was not trained to handle emojis, whereas for TFIDF it is best not to differentiate between lower and uppercase. The exact differences can be seen in Table 2.

	BERT	TFIDF
Stripping leading and trailing whitespaces	x	x
Replacing all whitespaces with spaces		x
Removing retweet tags ("RT")	x	x
Removing leading and trailing responses and mentions	x	x
Removing emojis	x	
Removing punctuation		x
Removing stop words		x

Table 2: Differences in preprocessing

Next, we decided to swap BERT for RoBERTa. RoBERTa iterates on BERT's pretraining procedure, including training the model longer, with bigger batches over more data, removing the next sentence prediction objective, training on longer sequences, and dynamically changing the masking pattern applied to the training data [6]. We downloaded the RoBERTa model from Sławomir Dadas's github repository[2].

We have tested five different classification models using RoBERTa:

1. SVM Linear
2. SVM RBF
3. RandomForest
4. Scikit MLP
5. Custom MLP

The results can be seen in Table 3. In terms of F1 micro score the Scikit MLP achieved the best results (88.5%) but when taken into consideration the F1 macro score the Custom MLP outperformed other models with a score of 53.1%. We believe in our case the F1 macro score is more important due to the high imbalance in the classes. The worst model both in terms of F1 macro was Random Forest (31.6%). It seems that it was not able to learn the pattern based on that data since the score is almost equal to the one achieved with guessing. When it comes to Scikit MLP we believe that the hyper parameters were not adjusted enough and the model could perform better.

Custom MLP was a TensorFlow solution, which we fine-tuned in respect of the number of layers and neurons. The problem with MLP was vast overfitting. Even though, we used dropout after every layer, we could not prevent the model from overfitting after just a few epochs, that is why the model was trained just for 3 epochs. The final network structure that we came up with can be seen below.

1. Dense (filters = 512, activation = relu, kernel_initializer = he_normal)

[1]https://github.com/kldarek/polbert
[2]https://github.com/sdadas/polish-nlp-resources

	SVM Linear	SVM RBF	Random Forest	Scikit MLP	Custom MLP
F1 micro	86.1%	81.2%	86.7%	88.5%	83.9%
F1 macro	52.0%	52.2%	31.6%	47.9%	53.1%

Table 3: RoBERTa (base) results

2. Dropout (rate = 0.25)
3. Dense (filters = 64, activation = relu, kernel_initializer = he_normal)
4. Dropout (rate = 0.35)
5. Dense (filters = 3, activation = softmax, kernel_initializer = he_normal)
6. Adam Optimizer (loss = categorical_crossentropy)

Then we moved onto RoBERTa large, the results can be observed in Table 4. Most of the models achieved better results than with RoBERTa base. The only model that performed worse was Random Forest (F1 micro worse by 0.1 pp and F1 macro by 0.7 pp). It comes as no surprise since we already have observed that it performs very poorly on this dataset and is in fact random. The best model here in terms of F1 macro score was SVM RBF. Despite its simplicity SVM is a very powerful machine learning model, especially when using the kernel trick such as RBF.

Even though we put a lot of work into fine-tuning the MLP solution, we were not able to outperform the SVM. The architecture had to be slightly altered compared to RoBERTa base, because of the increased size of embeddings. We actually trained the model for a few more epochs and greatly increased the dropout rate in both layers to 65%.

	SVM Linear	SVM RBF	Random Forest	Scikit MLP	Custom MLP
F1 micro	87.5%	84.4%	86.6%	89.0%	85.2%
F1 macro	52.5%	57.1%	30.9%	50.8%	54.4%

Table 4: RoBERTa (large) results

Next experiment we conducted was taking best models in terms of F1 micro and macro from RoBERTa base and large experiments in order to compare the embedding times and the size of the models. The results can be seen in Table 5. The best models from the base experiment were Scikit MLP and Custom MLP and from large SVM RBF ans Scikit MLP. What can be seen is the fact that not only RoBERTa base is much smaller (3.4 times smaller than RoBERTa large) but also much faster. The embeddings for RoBERTa base computed in 10 up to 15 minutes while for large it was almost 30 minutes. Nonetheless, in our final solution we decided to go with RoBERTa large because the time and size was bearable for us and both F1 scores were much higher. In order to save time we have saved the embeddings for each of the models on the drive which enabled us to just load them instead of computing them every time.

	base		large	
	Scikit MLP	Custom MLP	SVM RBF	Scikit MLP
F1 micro	88.5%	83.9%	84.4%	89.0%
F1 macro	47.9%	53.1%	57.1%	50.8%
Embedding time	10-15 minutes		25-30 minutes	
Model size	0.623 gigabytes		2.11 gigabytes	

Table 5: RoBERTa: base vs large

One of the last experiments was the a combination of TFIDF with RoBERTa. The results can be seen in Table 6. A combination of TFIDF, RoBERTa and SVM Linear achieved an impressive F1 micro score of 89.6% and 56% F1 macro. While SVM RBF with RoBERTa has achieved the best F1 macro score, it achieved a bit worse F1 micro score than SVM Linear. What is worth mentioning here is the fact that SVM RBF was way slower (it run for almost 2 hours) while MLP took only 10 minutes and SVM Linear as little as 5. That and the fact that SVM Linear achieved very good results in terms of both F1 macro and micro influenced our decision to choose it as a final solution.

	SVM Linear		SVM RBF		Custom MLP	
	F1 micro	F1 macro	F1 micro	F1 macro	F1 micro	F1 macro
TFIDF	86.1%	52.0%	81.2%	52.2%	83.9%	53.1%
RoBERTa	87.5%	52.5%	84.4%	57.1%	85.2%	54.4%
Combined	89.6%	56.0%	84.8%	57.0%	86.6%	56.7%

Table 6: RoBERTa + TFIDF

Additionally, as the last experiment, we decided to test the HerBERT model, since according to the KLEJ ranking [13] it is the best all-round Polish BERT model. Unfortunately, we did not manage to obtain any results that clearly outperform RoBERTa.

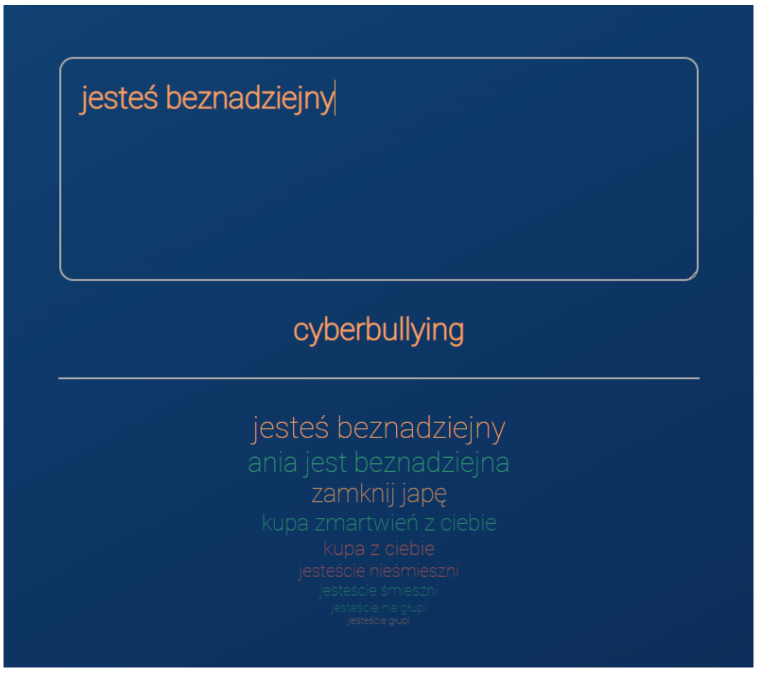
8 Deployment

The interface is very simple and written in vanilla JavaScript. The user can write his input in a provided text box and after clicking enter a request is sent to the REST API, which uses the pretrained model to classify the text. The predicted class is then returned in the response and shown under the box.

The history of previous texts is also visible under the box. Additionally based on the class the text was assigned to, the text will have a different colour:

- non-harmful – medium sea green
- cyberbullying – sandy brown
- hate speech – tomato

The screenshot of the UI can be seen in Figure 7.



jesteś beznadziejny

cyberbullying

jesteś beznadziejny
ania jest beznadziejna
zamknij japę
kupa zmartwień z ciebie
kupa z ciebie
jesteście nieśmieszni
jesteście śmieszni
jesteście nie głupi
jesteście głupi

Figure 7: User Interface

The REST API was written in python using a package called Flask. It has just one endpoint ”predict”, accepting query parameter ”text”, which is used for inference. For both website and API CORS is enabled.

The final solution is dockerized. There are two docker containers:

1. web-app – http-server hosting the web application
2. rest-api – RESTful API connected to the prediction model

Both containers are lightweight, they contain the bear minimum to run the required services. They share their respective ports in order to allow for communication. The ports are:

1. 8080 – for the Web Application
2. 5000 – for the REST API

Building and running the whole solution it as simple as typing (with docker installed):

```
docker compose up −d
```

9 API Benchmark

Last, we conducted an API performance benchmark. We chose 15 random sentences equally distributed among the classes. Then, we used a Postman Collection and Test Runner to measure the performance. We also tested the status and provided response for each request. The API provided the expected output in all 100% cases.

The histogram of request time can be seen in Figure 8. The mean time equalled 224.35ms with 21.05 standard deviation. The outliers never exceeded 376ms. Additionally, we tested the average time for invalid request. A request is invalid if ”text” parameter was not provided. The average Bad Request time was less than 20ms.

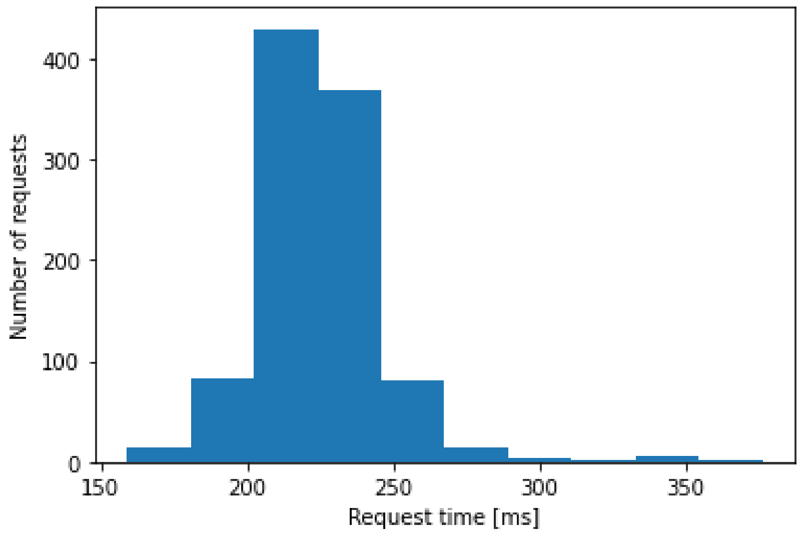


Figure 8: User Interface

10 Conclusion

We are very pleased with the provided solution. We believe multiple language processing approaches were analysed and the final model had a very adequate accuracy. In fact, the final model outperformed the competition winning solution by achieving both better micro and macro F1 scores. There certainly are possibilities for further improvement. Most notably, the provided dataset was limited in size with many duplicates (retweets and answers). Provided more training data, the model could most probably achieve even better results.

Additionally, the solution is ready for cloud deployment if needed. By dockerizing the services no extra configuration is needed.

References

[1] PolEval 2019. Task 6: Automatic cyberbullying detection, 2019.

[2] Femi Emmanuel Ayo, Olusegun Folorunso, Friday Thomas Ibharalu, and Idowu Ademola Osinuga. Machine learning techniques for hate speech classification of twitter data: State-of-the-art, future challenges and research directions.

[3] Komal Florio, Valerio Basile, Marco Polignano, Pierpaolo Basile, and Viviana Patti. Time of your hate: The challenge of time in hate speech detection on social media. *Applied Sciences*, 10(12), 2020.

[4] Kristina Gligoric, Ashton Anderson, and Robert West. Adoption of twitter's new length limit: Is 280 the new 140? *CoRR*, abs/2009.07661, 2020.

[5] Chetna Khanna. Text pre-processing: Stop words removal using different libraries, 2021.

[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[7] S. Malmasi and M. Zampieri. Challenges in discriminating profanity from hate speech., 2018.

[8] Tomasz Moździerz. Długość przecietnego polskiego wyrazu w tekstach pisanych w świetle analizy korpusowej. *Acta Universitatis Lodziensis. Kształcenie Polonistyczne Cudzoziemców*, 27:177–192, December 2020.

[9] 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.

[10] Michal E. Ptaszynski and Fumito Masui. Automatic cyberbullying detection: Emerging research and opportunities, 2018.

[11] A. Ribeiro, N. Silva, and F-HatEval. Semeval-2019 task 5: Convolutional neural networks for hate speech detection against women and immigrants on twitter. *Proceedings of the 13th International Workshop on Semantic Evaluation*, 2019.

[12] H. Rosa, N. Pereira, R. Ribeiro, P.C. Ferreira, J.P. Carvalho, S. Oliveira, L. Coheur, P. Paulino, A.M. Veiga Simão, and I. Trancoso. Automatic cyberbullying detection: A systematic review. *Computers in Human Behavior*, 93:333–345, 2019.

[13] Piotr Rybak, Robert Mroczkowski, Janusz Tracz, and Ireneusz Gawlik. Klej: Comprehensive benchmark for polish language understanding, 2020.

[14] E.A. Vega, J.C. Reyes-Magaña, H. Gómez-Adorno, and G. Bel-Enguix and. Mineriaunam semeval-2019 task 5: Detecting hate speech in twitter using multiple features in a combinatorial framework, 2019.

[15] K. Winter, R. Kern, and Know-center. Semeval-2019 task 5: Multilingual hate speech detection on twitter using cnns. *Proceedings of the 13th International Workshop on Semantic Evaluation*, 2019.