NNNLPProject

Mateusz (Orda + Rzepecki)

June 21, 2023

1 Project plan

1.1 Introduction

A very good introducion and motivation was provided by PolEval (copied below).

Although the problem of humiliating and slandering people through the Internet has existed almost as long as communication via the Internet between people, the appearance of new devices, such as smartphones and tablet computers, which allow using this medium not only at home, work or school but also in motion, has further exacerbated the problem. Especially recent decade, during which Social Networking Services (SNS), such as Facebook and Twitter, rapidly grew in popularity, has brought to light the problem of unethical behaviors in Internet environments, which has been greatly impairing public mental health in adults and, for the most, in younger users and children. It is the problem of cyberbullying (CB), defined as exploitation of open online means of communication, such as Internet forum boards, or SNS to convey harmful and disturbing information about private individuals, often children and students.

To deal with the problem, researchers around the world have started studying the problem of cyberbullying with a goal to automatically detect Internet entries containing harmful information, and report them to SNS service providers for further analysis and deletion. After ten years of research [1], a sufficient knowledge base on this problem has been collected for languages of well-developed countries, such as the US, or Japan. Unfortunately, still close to nothing in this matter has been done for the Polish language. With this task, we aim at filling this gap.

In this pilot task, the contestants will determine whether an Internet entry is classifiable as part of cyberbullying narration or not. The entries will contain tweets collected from openly available Twitter discussions. Since much of the problem of automatic cyberbullying detection often relies on feature selection and feature engineering [2], the tweets will be provided as such, with minimal preprocessing. The preprocessing, if used, will be applied mostly for cases when information about a private person is revealed to the public.

The goal of the contestants will be to classify the tweets into cyberbullying/harmful and non-cyberbullying/non-harmful with the highest possible Precision, Recall, balanced F-score and Accuracy. In an additional sub-task, the contestants will differentiate between various types of cyberbullying, i.e., revealing of private information, personal threats, blackmails, ridiculing, gossip/insinuations, or accumulation of vulgarities and profane language.

1.2 Project plan

Create a model that recognizes cyberbullying in tweets in Polish. Extra: distinguish types of cyberbullying.

- Read problem description and look on the dataset.
- Implement dummy solution (no nets, some ifs, detections of some words etc.) as a benchmark.
- Read literature and decide what models we want to train
- Train the models and evaluate them
- analyze performance of the algorithms on each step and try to improve bottleneck
- read more literature to get inspiration for improvements
- implement improvements
- stretchgoal: try to solve problems of detecting other things in tweets using our algorithm

1.3 Relevant existing literature

- This paper about cyberbullying detection: link to arxiv
- PolEval task statement: link to PolEval website

1. Michal E. Ptaszynski, Fumito Masui. (2018). "Automatic Cyberbullying Detection: Emerging Research and Opportunities", IGI Global Publishing (November 2018), ISBN: 9781522552499.
2. Michal Ptaszynski, Juuso Kalevi Kristian Eronen and Fumito Masui. (2017). "Learning Deep on Cyberbullying is Always Better Than Brute Force", IJCAI 2017 3rd Workshop on Linguistic and Cognitive Approaches to Dialogue Agents (LaCATODA 2017), Melbourne, Australia, August 19-25, 2017

1.4 Models to try

- Convolutional Neural Network (CNN),
- Long Short-Term Memory (LSTM),
- Bidirectional LSTM (BLSTM)
- BLSTM with attention

1.5 Dataset

Dataset of tweets from PolEval. It consists of 10k tweets. Each consists of author, text and tag. The tag says if it was cyberbullying or not. The second version of this dataset distinguishes the difference between cyberbullying and hate-speech.

1.6 How to evaluate success

PolEval provides a script to compute four metrics: Precision, Recall, Balanced F-score and Accuracy. They also provide ranking of the competition. We will compare our solutions with the ones from the ranking as well as our benchmark.

1.7 Milestone

As a milestone, we would like to:

- Read literature to get more context and see which approaches work here and which don't
- Implement a dummy algorithm to see how performant that is
- Decide which model is the best to try first and train it
- Draw conclusions from the first model, see where the improvement can be done and what types of models we should focus on from that point

1.8 Reviewer comments

- Clear dataset
- Try the harder version
- Make an analysis of a trained model (how input flows through the network)
- Look at network architectures for the same problem in other languages
- Implement a few (2-3) dummy algorithms

2 Milestone presentation

2.1 What we did

- Got familiar with the dataset
- Implemented dummy algorithms
- One of the dummy algorithms took second place in the harder version of the problem
- However, for the easier version of the problem the results from dummies were below average

2.2 Next steps

- Networks architectures to check:
  - Simple network
  - RNN
  - LSTM
  - Transformer

- Code input using vector embeddings (with some variations for instance: avg of the sentence, min, max, fixed length of a tweet (taking the essence of the tweet may help here), etc.)
- Create a report showing how each algorithm performed, what gave what improvement and how the project proceeded
- Pipeline
  - Data reader - extracts tweets from the file, repairs etc.
  - Tweet to vector - changes tweet to vector form (we should try different approaches here)
  - Model - takes vectors and returns distribution on the output classes
  - Classifier - takes model and implements utilities for it (writing answer to files, selecting best answer from distribution etc.)

3 Final report

3.1 Introduction

The problem description is in the project plan. This section will focus on our thoughts, troubles and solutions while implementing, testing and tuning models and the whole pipeline.

3.2 Abstract idea

The pipeline of the project is described in the milestone presentation.

3.2.1 Data reader

This part of the pipeline was mostly implementation.

At first it has to clear the data. By default, we delete just some remainings after parser inaccuracy, like HTML inserts. The motivation behind that is as follows. Most of other irrelevant parts of tweets should be truncated during translation to vectors or ignored by models regardless if they appear or not.

The second step here is tokenization. For all models but BERT-based, we used NLKT built-in tokenizer. For BERT-based model we used BERT-specific tokenization, which converts text into more complex structure which is lated plugged in into BERT layer.

Here it is worth to mention about one more thing about data preprocessing. We saw that the dataset has unequal distribution over classification classes and more complex networks tended to return the most frequent class. Thus, we augmented the dataset, by copying some of the samples, so that each class had the same number of representatives. This improved performance of LSTM, BiLSTM and RNN incredibly.

This section allows us also split train data into the real train and validation datasets.

3.2.2 Tweet to vector

At first, we had to select embeddings. We decided to use some ready embeddings, trained on big corpus, to have a solid basis there. We finally chosen one of the models trained on NKJP and Wikipedia among the ones available here: http://dsmodels.nlp.ipipan.waw.pl/

The embeddings turned out to work pretty nice. We restricted them to our corpus to reduce the size of the file and work on that more conveniently (for other data we just have to replace the relevant file). Something that could be done in the future is to try some different embeddings.

We had to further decide how to change tweet to vector of embeddings. One of the things to considered was that some networks work only with fixed-length input. We tried a few approaches:

- take the prefix of the tweet
- take the average of embeddings of all words
- take all embeddings of all words in the tweet and fill with zeros if needed

Some of considered models are able to take different sized input. Thus, for them we simply transformed all words from tweet to embeddings and feed the models with them.

3.2.3 Models

We tried the following models:

- naive Bayes
- word counter (our own heuristic based on naive Bayes idea)
- simple NN (a few linear layers with ReLu as activation)
- LSTM
- BiLSTM
- simple RNN
- BERT-based NN (partially pretrained model, based on transformers)

The selected models vary from very simple ones to very complex ones. We did that on purpose to be able to see if heavy machinery improves the result and if it does then how big is the improvement. This selection of models enables us to see how different ideas perform with our task. We decided to run each of the complex models in its own jupyter notebook. These notebooks contain final results and sometimes also some analysis of model performance.

3.2.4 Classifier validation

This was pretty simple. For all RNN we had a few choices of classifying a tweet. Namely, for a given result array (one value for each word) we needed to classify a tweet. We used the following approaches:

- take last element (performed poorly)
- take sum over all elements

- take max over all elements

Final model validation is described in section 'How to evaluate success' and in PolEval task description (linked above).

The last two approaches performed on a similar level.

3.3 Model performance

We optimized cross entropy which was not the loss PolEval wanted us to maximize (PolEval's one was not differentiable). Because of that we used performance on validation set as interpretation of how well the model behaves and we didn't focus only on the loss minimalization. We saw that good performance on validation didn't always imply good performance on test set. This is most likely a consequence of a small dataset.

While comparing models based on loss and validation score we focused on loss descrease as well as fluctuations in validation score. The latter sometimes was "randomly" increasing and decreasing what could mean that the model is not learning how to solve the task, but is lucky from time to time.

3.3.1 Naive Bayes

Very classical classification model.

Performs quite well on the second task and not so bad at the first one. Because of that, at first, its performance seemed to be a potential challenge for the networks.

3.3.2 Word counter

The idea was very similar to naive Bayes. For each word, we calculate the distribution meaning which fractions of appeariances of this word happens in every class. Next, for each class, we pick top K words with biggest fraction for this class.

To classify new tweet, we look at the words it consists of. Each of them potentially voices for the resulting class (proportionally to mentioned fractions).

Performs very good on second task. However, it fails miserably on the first one.

3.3.3 Simple NN

The first Neural Network we tried was the simple network. It consists of few linear layers with ReLU activation.

The network learns pretty fast. One needs to watch out to avoid overfitting. It beats both dummy approaches on the first task and achieves score between them in the second one. However, it definitely loses with some of the more complex networks.

3.3.4 LSTM

This network was hard to train. Firstly it was learning to print only 0 or to print only 1. After dataset augmentation it slowly started learning. The learning process was not so quick. The network was tested on many setup of hyperparameters to finally perform well. This model had the best hyperparameters tuning which yield very good performance.

3.3.5 BiLSTM

This network was implemented after LSTM, thus we had a lot of intuitions about hyperparameters what made the learning process way easier. We tested the network on many parameters inspired by LSTM's parameters.

3.3.6 Simple RNN

This was implemented similarly to BiLSTM. This newtork had huge problems with learning. Neither big nor small models were able to decently decrease the loss. They also didn't perform super well on the test set.

3.3.7 BERT-based NN

At the end we wanted to see if this pretrained model, based on famous transformers, can perform even better than the previous one. We started to work on it at the very end, as some kind of interesting bonus, so we didn't spend a lot of time on this approach.

We used one of pretrained BERT models, which was learned on 102 languages with biggest Wikipedias. The input here is a bit different. Instead of default embeddings, this model uses more complex input instances, which can be built by some (also pretraind) sibling module.

The idea was to treat this BERT model as first layer of network. It returns tensor of size 768, which should be somehow mapped to output of size equal to number of classes. The further idea was to keep this part as simple as possible and solve this issue using just single linear layer.

However, it turned out that it is not so easy to just learn it and obtain good results. We tried several tricks there, some of them same as for previous models (data augmentation, playing with complexity of added layers, ...). Finally, we exceeded Google Collab gpu limit and gave up, as we also didn't have much more time. We still belive that this approach can yield really good results, but it is definitely not so straightforward to obtain them and needs more time.

3.4 Detailed algorithms analysis

Each of the algorithms has its own notebook with analysis of the algorithm in the repo. The files consists mostly of implementation of what's described above plus runs of the algorithms, sometimes with different parameters, and score computations along with some analysis of what was happening.

3.5 Final results

Final results for our final models are presented in Table 1. LSTM would win both competitions!!!

Model name	2 classes	3 classes
Naive Bayes	33.18	84.20
Word counter	8.51	87.50
Simple NN	39.37	87.20
LSTM	62.89752650176679	89.1
BiLSTM	58.77551020408163	88.2
Simple RNN	50.65789473684211	85.1
BERT-based NN	31.52	?
PolEval best results	58.58	87.60

Table 1: Final results

3.6 Most harmful tweets according to LSTM

Which tweets excited our model the most? Warining! This subsection contains lots of phrases which are considered... harmful.

- anonymizedaccount anonymizedaccount ty pajacu idź sie doucz , bo jesteś tepym gnojkiem i pojecia nie masz co sie działo za czasów olszewskiego .
- anonymizedaccount ty pieprzona pisowska gnido , worku gówna i żenady .
- anonymizedaccount anonymizedaccount anonymizedaccount zreszta ty chuja zobaczysz , kutasa ziobry najwyzej
- anonymizedaccount anonymizedaccount i znowu łżesz jak bura suka . ue mówi całkowicie co innego niż wy do swoich moherów .
- anonymizedaccount anonymizedaccount o czym ty bredzisz . jak nic nie wiesz o sprawie to sie po prostu nie odzywaj i nie rób z siebie durnia .
- anonymizedaccount anonymizedaccount anonymizedaccount ty załgany łachudro , jakby nie brejza to nikt by o tym nie wiedział .
- anonymizedaccount anonymizedaccount anonymizedaccount tak minister edukacji uczy dzieci . kolejny pisowski kłamca , oszust i złodziej .
- anonymizedaccount anonymizedaccount kolega jest debilem , zawsze marsze w ramach protestu sa batdziej agresywna , niz w ramach lizania dupy wladzy
- jebac chuju , lewusów , nikt mi nie bronil ale to przesada
- anonymizedaccount anonymizedaccount anonymizedaccount pluli na kobiet , ciagali , dzicz chora z nienawisci , cywilacja jest zakaz dziczy

3.7 Conclusions

- In both tasks simple algorithms achieved pretty nice results. Simple NNs were not able to significantly beat them. Heavier machinery provided by deep learning was able to perform noticibly better, but training it to do so was very hard, took a lot of time and the training on (finally found) proper parameters took some time as well.

- Training a complex neural net is not easy. It required a lot of effort, analysis of what's happening to make the network perform on the level of dummies. From that point improving started being easier, but yet it was not automatic.
- Validation set was not much similar to the test set. Good results on validation did not imply reasonable results on test set. This could be caused by the small size of the dataset.
- Many of the models were overfitting and most of the best versions of models were trained in the first few epochs.
- Optimizing not a differentiable function by a neural net is significantly harder than a differentiable function.
- In recursive NN we have to analyze the result for all words instead of the result for the whole tweet.
- It also turned out that learning some general pretrained model to solve our specific problem is not as straightforward as it may seem and may require some additional time and effort.