NLP, Spring 2023

Kirill Kuznetsov, Evgenii Evlampev, Danil Davydyan
k.kuznetsov@innopolis.university,
e.evlampev@innopolis.university
d.davydyan@innopolis.university

Link to github

Link to Google Drive with Embeddings

Note: To run the code you have to download embeddings from google drive and change variable for path to embeddings to proper path in your local computer, variables: embeddings_dir.

Toxic Comment Classification

1. Introduction

In this project, we will explore the problem of toxic comment classification, which involves automatically identifying comments that contain toxic language. This is an important task in the field of natural language processing (NLP) and has numerous applications, including content moderation, sentiment analysis, and online safety.

We will use a dataset provided by the Jigsaw Toxic Comment Classification Challenge on Kaggle, which contains a large number of comments that have been labeled as “toxic”, “severe_toxic”, “obscene”, “threat”, “insult”, “identity_hate”. Using this dataset, we will train and evaluate machine learning model to classify these comments. We will also explore different techniques for preprocessing the data and extracting useful features from the text.

The goal of this project is to develop a machine learning model that can accurately classify toxic comments in real-world online platforms.

2. Review of existing approaches

In this section, we will review some of the existing solutions to the toxic comment classification problem.

1. Bag of words approach: One of the earliest approaches to text classification involved representing each comment as a bag of words and using a simple machine learning algorithm such as logistic regression or naive Bayes to classify the comment as toxic or non-toxic. This approach has the advantage of being easy to implement and interpret, but it often suffers from poor performance due to the lack of contextual information and the inability to capture the meaning of the text.

2. Convolutional neural networks (CNNs): CNNs have been successfully applied to various NLP tasks, including text classification. In the context of toxic comment classification, CNNs can be used to learn features from the text at different scales and capture important patterns in the text.

3. Recurrent neural networks (RNNs): RNNs are another popular type of neural network that have been used for text classification. In the context of toxic comment classification, RNNs can be used to model the sequence of words in a comment and capture the contextual information in the text.

3. Our solution and experiments

For our project we used the prepared dataset from kaggle with labeled comments.

3.1 Data collection and processing.

Our code revises toxic words to the right format and that helps the word embedding layer in the model recognize more toxic words. We have severals step in preprocessing:

- cleain_text: delete some useless characters like digits and punctuation marks.
- revise_cuts: delete cuts like "you're" or "i'm" and so on.
- revise_star: due to stars often used to disguise bad words, we create a set of the most popular bad words and check: "does the word in bad if delete all start?", if yes then delete stars.
- revise_triple_and_more_letters: delete char duplicates.
- revise_redundancy_words: delete redundancy of bad words.

Also we have added new features to the data such as “caps_vs_length” - ratio of uppercase to lowercase letters, “words_vs_unique” - ratio of unique words to overall number of words. Probably, these features will help us to get better accuracy.

3.2 Architecture and implementation

Our model for text classification uses a combination of pre-trained word embeddings from two different sources: FastText and GloVe Twitter. Structure of ANN:

- First layer: concatenated fasttext and glove twitter embeddings. Fasttext vector is used by itself if there is no glove vector but not the other way around. Words without word vectors are replaced with a word vector for a word "something". Also, I added additional value that was set to 1 if a word was written in all capital letters and 0 otherwise.

- Second layer: SpatialDropout1D(0.5)
- Third layer: Bidirectional CuDNNLSTM with a kernel size 40. We found out that LSTM as a first layer works better than GRU.
- Fourth layer: Bidirectional CuDNNGRU with a kernel size 40.
- Fifth layer: A concatenation of the last state, maximum pool, average pool and two features: "Unique words rate" and "Rate of all-caps words"
- Sixth layer: output dense layer.

First, we create a tokenizer to get tokens from words from train and test dataset and then use embedding to get vectors from tokens.

Second, we initialize an embedding matrix with zeros, with dimensions (nb_words, 501), where 501 is the total size of the embedding vector. The code generates embedded vectors from both embeddings: FastText and GloVe Twitter embeddings. Then embeddings are concatenated and added to the embedding matrix. If a word is not present in either embedding, the code uses the embedding for the word "something" instead which also was used for NaN values in initial data. If a word is misspelled or not present in the embeddings, the code tries to find a corrected version of the word using custom spell correction which was shown and implemented in the second assignment.

The model architecture is defined in the `get_model()` function. The input to the model consists of two inputs: a sequence of text data (inp), and an additional feature (features_input). The text input is first passed through an embedding layer, which uses weights from previously embedded words from train data.

The resulting embedding is then passed through Bidirectional LSTM layer, which allows the model to capture both forward and backward contexts in the input text. Then It will go to the second layer: Bidirectional GRU layer, which has the same idea of ‘memorizing’ context from further words and from close words. The outputs of the GRU layers are concatenated with the features input, output from max pooling(input to it is output from GRU) and output from average pooling(input to it is output from GRU also), then it passed through a dense layer to produce the final classification output.

The model is trained using binary cross-entropy loss and the Adam optimizer, with accuracy as the evaluation metric. The model is compiled and returned at the end of the function.

4. Evaluation and Conclusion.

- Evaluation:

  To evaluate our model we use a method which was shown in kaggle competition, its ROC AUC score - Compute Area Under the Receiver Operating Characteristic Curve score. We implement a function to get a score and then evaluate train and test data. We have trained the model to classify toxic comments with accuracy 0.97 on the training set, and 0.95 on the test set with ROC AUC score. So, our model can successfully detect types of toxicity and be used in real life. For example, filter out negative comments on a website by the degree of toxicity.

```
4040/4040 [==============================] - 3265s 806ms/step - loss: 0.0958 - accuracy: 0.9739

499/499 [==============================] - 137s 272ms/step
Test score: 0.9525899970815279
Done
```

- What have we learned:

  We have learned how to properly preprocess data and found out new methods for preprocessing. We understood how to properly define the architecture of a model and implemented it for Text Classification problems. We learned how to use FastText and GloVe Twitter embeddings together.

- Conclusion:

  For our project, we used embeddings from FastText and GloVe Twitter and implemented a deep learning model consisting of an embedding layer, bidirectional LSTM and GRU layers, and additional features. We also applied various preprocessing techniques to the data, including cleaning the text, revising cuts and redundancy words, and adding new features.

  Our experiments showed that our model achieved high accuracy on the test dataset. This suggests that our approach to preprocessing and model architecture is effective for toxic comment classification.

- Team members contribution:
  - Kirill Kuznetsov - preprocessing, tokenization, choose embedings, creation model with layers and add some comment to ipynb file.
  - Evgenii Evlampev - preprocessing, creation model, training model, writing a report.

- Danil Davydyan - spelling correction, add comments to ipynb, research different solutions, help with training a model.