Università degli studi di Milano-Bicocca

Advanced Machine Learning

Final Project

Toxic Comment Classification Project

Authors:
Riccardo Frigerio - 852226 - r.frigerio12@campus.unimib.it
Mattia Napoli - 852239 - m.napoli16@campus.unimib.it
Giulia Tremolada - 861144 - g.tremolada8@campus.unimib.it

January 31, 2024

Abstract

Being able to properly label comments according to different toxicity levels comments is a critical component in the moderation of every forum-like application. A substantial section of this work was dedicated to word embeddings, including pre-trained ones like GloVe and FastText, which were subsequently combined with various models like Transformers, RNNs and CNNs. Results revealed that the quality of the embedding is essential for the performance and choosing the right one can be more effective than the choice of a model's architecture. Nevertheless, the model that surprisingly achieved the best outcome overall was the CNN which was the only one able to recognize all labels.

1 Introduction

The contemporary relevance of discussions and online dialogues has grown significantly. The act of openly expressing one's opinions makes each user susceptible to severe criticism, if not outright insult and humiliation. In addressing this phenomenon within the context of the Kaggle challenge [1], participants were tasked with constructing a model capable of labeling Wikipedia comments based on various categories of toxicity.

In response to the requirements of this challenge, a strategic decision was made to place greater emphasis to the embedding phase, with the objective of facilitating the subsequent definition and execution phases of the various designed models. The focus of this project therefore lies in the attempt to define robust embeddings that encapsulate and transmit a substantial part of the word information.

Beyond this, the adoption of classic architectures such as CNNs, RNNs, and transformers is contemplated in order to perform a comparative analysis to find the optimal one. Transformers are expected to deliver superior performance as they're explicitly designed for tasks of this nature. On the other side, CNNs could prove to be an efficient model, despite their limitation in capturing relationships between distant words due to the local structure of filters. Concerning RNNs, there is a belief that they occupy a middle ground, exhibiting performance superior to CNNs yet falling short of the transformative capabilities of transformers.

2 Datasets

The dataset provided by Kaggle consists of a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. There are seven colums which refer to the comment id, the comment text and the possible labels assigned to each one of them (there is one column with binary value for each label). A comment can be assigned to more than one label chosen among: "toxic, "sever toxic", "obscene", "threat", "insult" and "identity hate". It is also possible that comments are assigned none of those labels, so the labels' columns would be all zero.

The dataset was already split in train and test set when downloaded; the train set is composed of 159571 records while the test set includes other 153164. However, since it was a Kaggle competition, the test set comments and labels were divided in separate files and many comments were labelled as -1 to indicate that they were not used for scoring. Thus, while the train set was kept untouched, the test set comments and labels were merged into one dataframe and all comments with labels -1 were deleted. This resulted in only 63978 test records for the evaluation.

It was observed that for both the train set and test set the vast majority of comments had all zero labels and that the labels' distribution of both dataframes were extremely similar. These considerations remained sound even after further splitting the train set in order to obtain the validation set ($80\% - 20\%$ split). This was essential as validation records are used to estimate overfitting and manage the training phase of the models.

2.1 Preprocessing

Text preprocessing is an essential step in natural language processing that involves transforming raw text into a more suitable format for analysis and modeling. In this case, the performed operations aimed to delete useless information for toxic comment classification and to enhance those that are instead significant for the task. Therefore, the following preprocessing steps were taken: conversion to lower case, removal of inline characters, removal of numbers, removal of non-ASCII characters, removal of non-alphanumeric characters, removal of leading and ending spaces and removal of the underscore character within words. Stopwords, i.e. most common words in the English language that do not carry important meaning, were also removed.

Moreover, the lemmatization process was carried out so that words are reduced to their canonical or dictionary form.
An example of the effects of the described operations is showed below:

> D'aww! He matches this background colour I'm seemingly stuck with. Thanks. (talk) 21:51, January 11, 2016 (UTC)
>
> daww match background colour im seemingly stuck thanks talk january utc

2.1.1 Tokenization and Vectorization

As machine learning models cannot directly process data in a text format, the comments had to be converted in a numerical form first. The first step of this process is the *tokenization*: text is broken down in smaller units called tokens and a vocabulary for the dataset is built from them. This operation is applied after the preprocessing phase of comments.

Vectorization consists instead of creating, for every comment, a vector of integers, one for each token in the comment. The integer assigned to each token is the token's index in the tokenizer's vocabulary. Tokenization and vectorization are carried out contextually by Keras [2] TextVectorization layer. For efficiency reasons, the whole process was performed by fixing the number of words for every preprocessed comment to 200, so that longer comments will be truncated to this length and shorter comments will be padded with zeros during vectorization. The predefined length was chosen by observing that most comments have number of words less than 200; more precisely, just 1.98% of them are longer than 200 words and therefore will be truncated.

2.2 Data augmentation

Dataset preliminary analysis revealed a strong imbalance between the different labels. Furthermore, positive samples for toxic comments were always much less than samples of non-toxic ones, with differences reaching several orders of magnitude. This situation could invalidate the model by making it unable to capture the characteristics of the classes with lower frequency.

Some techniques for data augmentation include back translations, synonym replacement or random insertions/deletions. However, these solutions tend to produce vectors almost identical to original ones. To mitigate this behaviour, tactics in literature include the possibility of generating synthetic samples using specific algorithms. One of them is *Synthetic Minority Oversampling TEchnique* (SMOTE) [3] which works by selecting two sample from the minority class that are close in the feature space and drawing samples along the "line" between the two. In essence, given the vectorized dataset, it generates synthetic samples for the minority class by interpolating between existing instances.

A multilabel version of SMOTE was used, however, various tests shown that this approach was not well suited for the case at hand. Analyses indicated that the synthetic data lacked meaningful contributions and eventually achieved the opposite result: models were fooled by the new samples and failed to generalize at all. The explaination for this probably resides in the fact that alterations in the vectorized form of the comments fail to maintain the meaning and semantics of the original sample.

3 The Methodological Approach

3.1 Embedding Layer

Embeddings are a crucial component for the task: their purpose is to capture as much semantic and syntactic information as possible from each word by using real-valued vectors. This should facilitate the models' ability to learn from and generalize to different linguistic patterns.

The most straightforward approach was to create an embedding layer, which included a positional encoding, and let the model learn its parameters during the training process. While this provided the most flexibility for the current task, there was no guarantee that the model would be able to generate meaningful embeddings. Moreover, this led to higher execution times since an appropriate vector had to be built from scratch for each word of the vocabulary.

The natural step forward from this was the use of pre-trained embeddings. These are usually taken from models trained on very large collections of documents and should provide a richer representation of the words. Among the alternatives, GloVe embeddings [4] were chosen for few different reasons:

- even in their maximum vector size, 300, the most complete representation for each word, they were easily manageable in Colab;
- they provided a version based on 6 bilions tokens from Wikipedia, matching the context of the comments to classify; an alternative was based on 12 bilions tokens from Twitter, however it performed worse despite the bigger size, probably because data provided by Wikipedia is far more meaningful and suitable for the task.

Even after choosing the most promising GloVe configuration, many tokens in the train set, precisely 108835 out of 185790 (58%), didn't have a corresponding GloVe embedding and were consequently assigned an all-zero vector. Many of such words were essential to determine the toxicity of a comment, as they were mostly compound words made of individual bad words. A first attempt to tackle this problem was to assign to these "missing" tokens the same embedding as their longest substring found within GloVe words. However, this actually worsened the models' performance probably because of the very naive method used to extract one word from a compound one.

A second and more robust approach to solve the high percentage of missed embeddings was the use of FastText library [5]. The model behind this embeddings tries to extract meaning from the morphological structure of a word by treating it as an aggregation of subwords. The vector for a word is simply taken to be the sum of all vectors of its component char-ngrams. This enables FastText to generate embeddings even for *out-of-vocabulary* (OOV) words, which was a major flaw of the GloVe solution. In fact, being trained on Wikipedia like Glove, the model did not recognize 106944 words, still, it managed to generate non-zero vectors for the whole vocabulary. However, an unforeseen problem was that the pretrained model is 6.8 GB in size and takes 20GB of RAM to execute, thus the embedding matrix had to be built on an external machine and then uploaded back to Colab for use.

GloVe and FastText embeddings are usually frozen during training for efficiency reasons. However, they can be made trainable in a second phase to fine-tune them for the task at hand in order to suite more the dataset. To perform a thorough evaluation of the perfomances and characteristics, each chosen model has been trained in the following configurations:

- token and positional embedding
- first training phase with frozen GloVe embeddings, plus fine-tuning

- first training phase with frozen FastText embeddings, plus fine-tuning

3.2 Models

For a machine learning model the task of distinguishing between toxic and non-toxic comments requires a good understanding of the role that each word plays in the sentence. An immediate choice in this sense was the use of Transformers. The built-in attention mechanism is made exactly for such cases and should be able to weight words within the sentence according to the others. The final model was a single transformer block with multihead attention which was used as an encoder for subsequent dense layers.

The next category of examined models was Recurrent Neural Networks as text can be seen as sequential data. Both *Long Term Short Memory* (LSTM) and *Gated Recurrent Unit* delivered good results but the best performance was achieved by a *bidirectional* LSTM (Bi-LSTM). While in a traditional layer the input is only processed forward, a Bi-LSTM adds another independent LSTM layer that process it backward and combines the result of both. This allows to capture the context from both previous and following words.

Both transformers and RNNs had problems with comments labeled as "threat" or "identity hate", the two less represented labels in the whole dataset. An attempt to fix this was the introduction of Convolutional Neural Networks. The convolution operation is able to extract features independently of their position in the sentence and thus may be able to better learn the characteristics of such comments. The final model uses the typical combination of one 1D convolutions followed by a Max Pooling Layer before the fully connected network.

All three different models have been tested against the previously explained embeddings in order to find the best combination for the task.

Because of considerable class imbalalance, overfitting is expected. Employed techniques to avoid this were the addition of *Dropout* layers in every model and the *EarlyStopping* callback. In particular, the EarlyStopping callback's parameters were set differently according to the training phase: during fine-tuning the minimum expected improvement was reduced to match the lower learning rate.

4 Results and Evaluation

All three models, combined with the embeddings, were initially evaluated according to the ROC-AUC measure as this was the metric used by the Kaggle competition. However, due to the pronounced class imbalance, the AUC metric for model comparison may not truly represent the ability of the models to recognize toxicity. Models that perform well on majority class instances but poorly on minority class ones can still achieve good AUC scores. To address this limitation, further analyses were carried out based on Precision, Recall, and F1-measure to find out how the models learned the dataset and to determine the best model overall. Whenever pre-trained embeddings are used, results are split in before and after the fine-tuning.

Table 1: AUC Metric (in %) of Models

(a) Positional

Model	
Transformer	95.5
Bi-LSTM	96.1
CNN	96.1

(b) GloVe

Model	Frozen	Fine Tuning
Transformer	95.9	96.5
Bi-LSTM	96.6	97.0
CNN	95.9	96.2

(c) FastText

Model	Frozen	Fine Tuning
Transformer	95.3	96.8
Bi-LSTM	96.2	97.1
CNN	96.8	97.3

Table 2: F1-Measure (in %) of Models with Positional Embeddings

	Toxic	*Severe Toxic*	*Obscene*	*Threat*	*Insult*	*Identity Hate*
Transformer	61	5	67	0	58	0
Bi-LSTM	63	36	70	0	61	0
CNN	65	24	68	0	62	0

Table 3: Evaluation Metrics (in %) of Models with GloVe Embeddings

Label (count)	*Model*	*Frozen*			*Fine Tuning*		
		Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Toxic (6090)	Transformer	54	80	64	55	83	67
	Bi-LSTM	57	80	66	55	84	67
	CNN	54	82	65	53	85	65
Severe Toxic (367)	Transformer	21	5	8	32	33	32
	Bi-LSTM	36	38	37	37	39	38
	CNN	35	13	19	32	48	38
Obscene (3691)	Transformer	65	65	65	63	73	68
	Bi-LSTM	69	64	66	62	74	68
	CNN	66	66	66	62	73	67
Threat (211)	Transformer	0	0	0	0	0	0
	Bi-LSTM	0	0	0	0	0	0
	CNN	45	14	22	38	39	39
Insult (3427)	Transformer	57	67	61	57	72	64
	Bi-LSTM	63	63	63	59	72	65
	CNN	60	63	62	56	72	63
Identity Hate (712)	Transformer	0	0	0	73	29	41
	Bi-LSTM	69	36	47	66	47	55
	CNN	73	31	44	61	53	57

5 Discussion

Results will be discussed by analyzing the performance of each model on the same embedding configuration, specifically starting from the standard embedding layer and moving on to GloVe and FastText. Finally, global considerations on the behaviour of models and embeddings will be provided.

Table 4: Evaluation Metrics (in %) of Models with FastText Embeddings

Label (count)	*Model*	*Frozen*			*Fine Tuning*		
		Precision	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Toxic (6090)	Transformer	51	87	64	50	90	64
	Bi-LSTM	57	79	66	55	86	67
	CNN	60	78	68	54	86	66
Severe Toxic (367)	Transformer	32	33	33	31	54	39
	Bi-LSTM	35	18	24	37	41	39
	CNN	42	27	33	40	42	41
Obscene (3691)	Transformer	74	54	63	58	81	68
	Bi-LSTM	72	64	67	62	79	69
	CNN	71	62	66	60	78	68
Threat (211)	Transformer	0	0	0	0	0	0
	Bi-LSTM	0	0	0	0	0	0
	CNN	32	11	17	34	25	29
Insult (3427)	Transformer	60	48	54	50	74	60
	Bi-LSTM	61	59	60	56	73	63
	CNN	69	57	62	59	72	65
Identity Hate (712)	Transformer	0	0	0	46	13	21
	Bi-LSTM	63	5	10	59	22	32
	CNN	80	16	27	69	45	54

5.1 Standard Keras Embedding Layer

The embeddings obtained during models' training phase using the standard Keras embedding layer are by definition specific to the Kaggle challenge and its data. However, F1-scores from table 2 showed that all models failed to capture labels with the smallest amount of samples, "Threat" and "Indentity Hate". It would seem that word representations built with this approach didn't capture enough semantic meaning for models to generalize well. This is exactly the reason why GloVe embeddings pre-trained on Wikipedia data were introduced. It is also clear that the Trasformer model had the worst performance overall and suffered particularly from class imbalance since it had poor recognition of even the “Severe Toxic” class, which is itself quite small. The poor achievements of Trasformer are also confirmed by the AUC metric in table 1. This type of embedding eventually set the baseline for the following considerations about other solutions to generate vectors from words.

5.2 GloVe Embeddings

The use of GloVe pre-trained embeddings did actually improve most models performance with respect to training embeddings from scratch, as table 3 convey. Even if neither AUC nor F1-measure for bigger classes changed much, the recognition of the less represented labels such as “Threat” and “Identity Hate” was definitely better. While the Trasformer model still assigned an all-zero prediction to comments with the mentioned labels, Bi-LSTM and CNN were able to recognize and classify correctly at least some of them, especially those in the “Identity Hate” class. In particular, the CNN identified comments of both critical classes and the LSTM instead behaved better on “Identity Hate” but didn’t identify the “Threat” class. The CNN showed the most promising results in this sense and the Trasformer model confirmed its limited contribution to the task.

GloVe Embeddings were set as trainable to obtain better performances using fine-tuning: a solid starting point was already built and appropriate adjustments to the current task could be made. Results did indeed improve for all models, both in terms of AUC and specific metrics: while F1-scores of larger classes remained consistent, both Precision and Recall of the critical labels increased for all models that were able to recognize them. Moreover, fine-tuning seemed to favor recall over precision, even those whose F1-Measure remained stable. The most significant improvement was obtained by the Trasformer model, which was able to identify the “Identity Hate” class; in addition, its overall F1-score performance almost reached the one of Bi-LSTM and CNN, which in this case have very similar behaviour.

The models’ performances shown with fine-tuning confirmed that embeddings have a much bigger impact on results than a specific network architecture does. AUC values are in line with previous considerations but indicate the Bi-LSTM model as the best performing one. Nevertheless, the CNN model was still the only one that could recognize some “Threat” comments and therefore remained the leading architecture.

5.3 FastText Embeddings

As it already happened with GloVe, table 4 outlines that fine-tuning incremented AUC, F1-measures (still favoring recall) and the number of recognized classes with respect to the frozen version. Specifically, CNNs exhibited the smallest increase and Bi-LSTMs represented an intermediate scenario, displaying a modest gain. Notably, for Transformers, fine-tuning proved again to be particularly effective, yielding a one-percentage-point increase in AUC.

However, fine-tuned FastText Embeddings gave surprisingly similar results, if not slightly worse, to the fine-tuned GloVe approach, both in terms of F1-score and in terms of classes recognized by models. Just the CNN performance sensibly improved and that is consistent with the AUC results, which show best value for this configuration. FastText embeddings were supposed to add considerable value to toxic comments classification because of the subwords recognition mechanism. The disappointing performance could be due to the fact that out-of-vocabulary words do not have that great of an impact on predictions or that the generated embeddings for these word still don't seize the intended semantic. GloVe embeddings seem to better capture the global context and are of higher quality in this sense. The same motivation stands behind the fact that the boost in AUC performances obtainted with fine-tuning is greater than the one obtained with GloVe. This highlights the fact that Glove embeddings had higher starting quality for the current challenge and needed less improving.

5.4 Evaluation of the Methodological Approach

The proposed approach mainly focused on improving performance through the introduction of different word embeddings, while keeping the models simple. This perspective was proven to be quite effective since varying only the examined layer equally improved all models.

Data augmentation was not applied in the end because the methods tested weren't adding notable progress. However, a more extensive research among standard text data augmentation procedures could be useful as future works to contrast class imbalance, which was the biggest challenge of this task. Data cleaning was also kept basic; more advanced procedures could be implemented in order to additionally enhance significant expressions.

As stated previously, proposed models were fairly simple and a comparison with more sophisticated ones like BERT could be relevant. The findings of this project imply the following recommendations: it is advisable to concentrate efforts on embeddings first and eventually on model architecture.

6 Conclusions

Regarding the best choice for the embedding layer, it was largely demonstrated that fine-tuned pre-trained embeddings reached a level of text semantic comprehension unmatched by their frozen version and by the Keras layer trained from scratch. Both fine-tuned GloVe and FastText enabled most models to recognize even the smaller classes, proving that quality of embeddings is essential in such tasks. However, the former seemed to represent words in a more incisive way even though the latter managed to achieve higher AUC scores. As they performed very similarly, choosing one over the other will be stricly dependent on the task to be performed.

The chosen architectures proved to fit data differently. Specifically the transformer struggled the most with high label imbalance while Bi-LSTM and CNN were able to manage it. Ultimately, the Convolutional Neural Network was deemed as the most appropriate model for toxic comment classification challenge as it was the only one able to recognize all toxicity levels.

References

[1] Toxic comment classification challenge. [Online]. Available: https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge

[2] Keras: deep learning for humans. [Online]. Available: https://keras.io/

[3] Handling data imbalance in multi-label classification (mlsmote). [Online]. Available: https://medium.com/thecyphy/handling-data-imbalance-in-multi-label-classification-mlsmote-531155416b87

[4] Glove: Global vectors for word representation. [Online]. Available: https://nlp.stanford.edu/projects/glove/

[5] fasttext. [Online]. Available: https://fasttext.cc/