Text Mining and Natural Language Processing

2024-2025

- **Alessandro Di Piano 535269**
- **Malgorzata Ozarowska 535414**

ICTC – Internet Comments Toxicity Classification

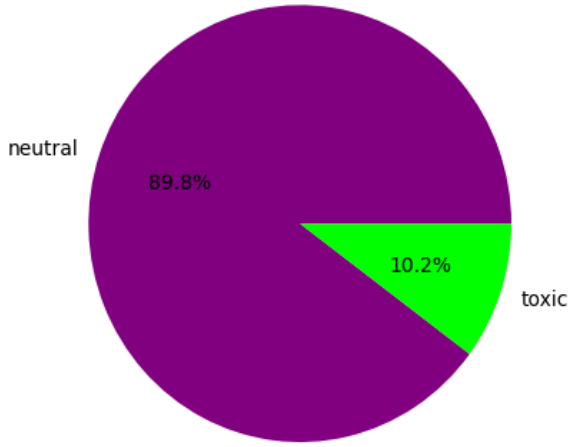
Introduction

The goal of the project is to build a text-processing model able to recognize and classify toxic behaviour of Internet users. For the purpose of this task, toxicity is defined as *comments that are rude, disrespectful or otherwise likely to make someone leave a discussion.*
Moreover, the model must be able to differentiate between various types of toxicity (e.g. threats, obscenities, insults or hate speech). Any instance of the dataset can belong to any combination of those classes.

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.
-task description on kaggle.com

Data

The dataset is a collection of posts (comments) from Wikipedia's editors' forum. The distribution of labels is very unbalanced, the dataset includes `159570` instances, but the majority of them do not belong to any of the positive classes.
Additionally, within the toxic class, some labels are severely underrepresented:

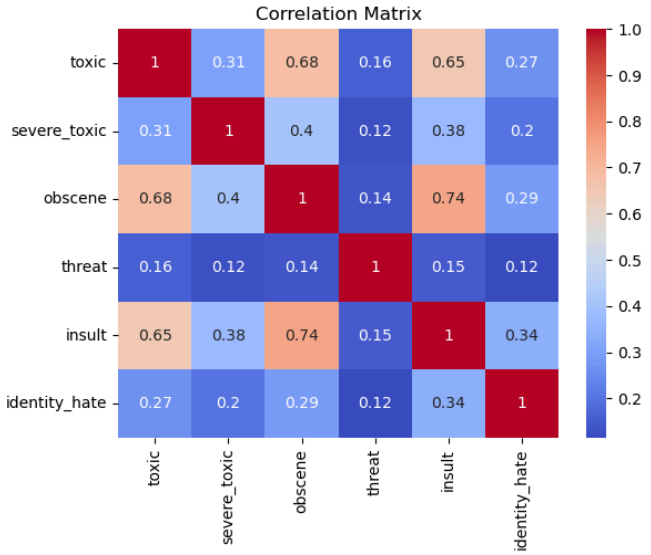


class distribution in the dataset

```
TEST SET
toxic: 1530 instances (9.59%)
severe_toxic: 160 instances (1.0%)
obscene: 845 instances (5.3%)
threat: 48 instances (0.3%)
insult: 788 instances (4.94%)
identity_hate: 141 instances (0.88%)
```

label distribution

Since the task is multi-label, some classes are highly correlated with each other:
note: many correlations in the data are not displayed here, since one-directional correlation (dependence), e.g. "any instance that is severely toxic is also toxic", is not considered in the matrix



Comments vary significantly in length, they require editing for efficient training

```
longest: 1235
shortest: 0
average: 32.07675212318438
median: 15
```

```
longest: 1807
shortest: 2
average: 79.08742757361695
median: 39
```

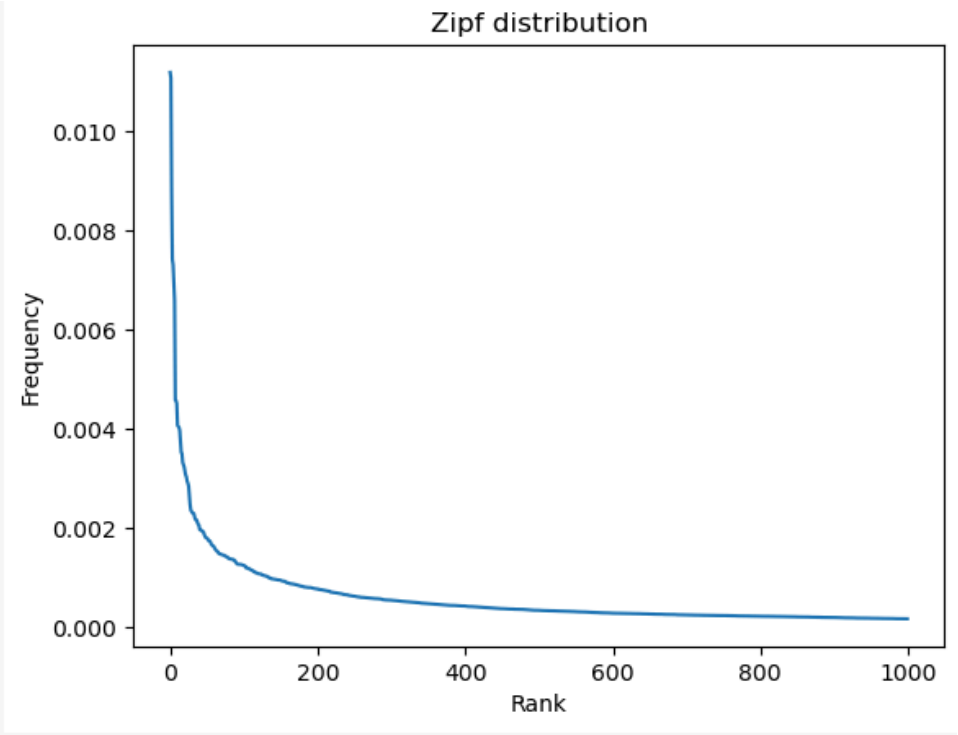
comment lengths in words (left) and subwords (right)

All comments in the dataset are in English, but the majority of them are written in various types of Internet slang. This feature of the dataset is a source of unique processing challenges, as the dataset has a lot more unique words than a text written in formal English would.

example of a comment containing almost no correctly spelled words. the drawing made using punctuation is clearly toxic for human evaluators, but difficult (if not impossible) to correctly interpret, even for very advanced models

The context of the original data gives rise to the SolidGoldMagicarp phenomenon, as users arguing with each other refer to one another by username, or mention repeatedly some entity (e.g. a film or a videogame). Both statistical and neural models learn those words, and because the words are not semantically related to the presence or absence of toxicity, they create noise for the classifier.

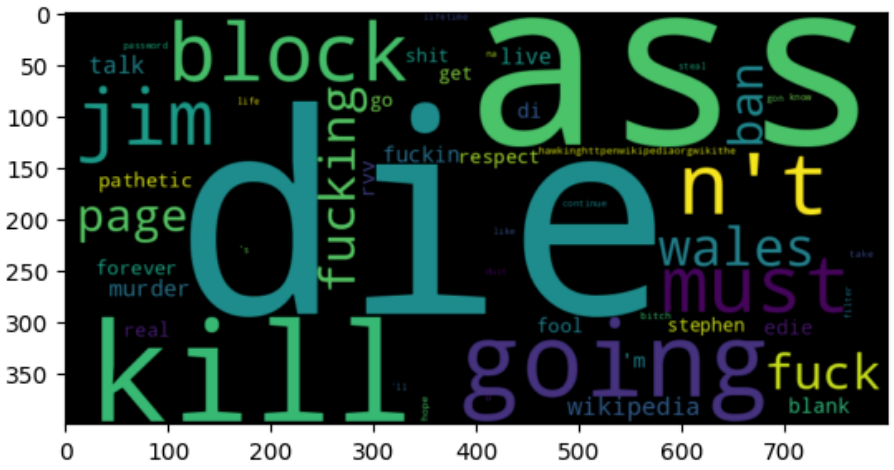
The distribution of words follows the Zipf law: after removing stopwords, the most frequent words of the dataset are either typical for the context from which the data was taken (e.g. “wikipedia”, “admin”, “edits”), or swear words and common insults.



word frequency/rank graph of the data after preprocessing

Classes are well characterized by their unique most-frequent vocabulary, which might be an explanation for the good performance of count-based embeddings (as shown later)

```
DATA_THREAT
['hi', 'back', 'last', 'warning', 'stop', 'undoing', 'edits', 'die', 'think', 'fagget', '
Number of all words: 14308
Number of unique words 2898
```



50 most frequent words for the “threat” label.

Methodology

Our approach was to try out different possible solutions at each step of the NLP-pipeline, to find the best performing configuration for this specific dataset.

I. preprocessing:
   1. basic- the standard procedure of cleaning the text of non-alphabetical characters (punctuation, digits, emoji) and removing stopwords
   2. leaving the text uncleaned, to let the model infer from all information available, not just words

II. tokenization:
   1. word- tokens separated by whitespace (training an instance of word-tokenizer from the nltk library)
      – this tokenization was performed on the text preprocessed using the 1st (basic) approach
   2. subword- characters in the text merged into word piece strings based on co-occurrence frequencies (training the WordPiece Tokenizer from tokenizers library)
      – this tokenization was done on the uncleaned text (2nd approach)

III. embedding:
   1. statistical methods: PPMI and Tf-idf
   2. word2vec (a pre-built model from the gensim library)
      – both statistical and word2vec embeddings, since they are based on word- co-occurrence, were built using the 1st (basic) preprocessing and word-tokenization

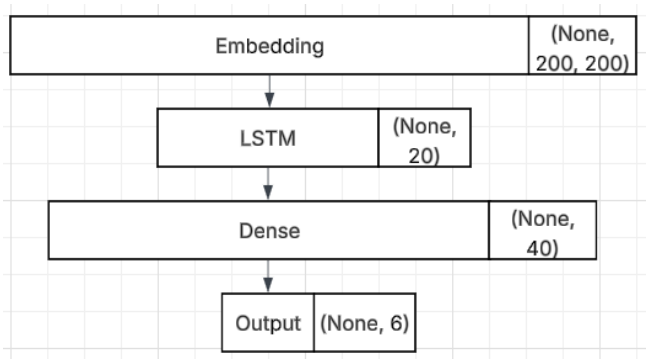
   3. task-specific embeddings trained directly by a neural model (via an Embedding layer)
      – based on subword tokenization

IV. model selection:
   1. Logistic Regression (an ensemble of six models, each a binary classifier trained on one label).
      – trained either on word2vec or PPMI embeddings

   2. a RNN with an LSTM layer – we trained four models, each with the same structure, differing by the choice of embedding matrix: pretrained (from word2vec, PPMI or Tf-idf) or trainable
      Each model was trained using binary cross entropy as loss, given six sigmoid-activated outputs in the last layer

      To counter the class imbalance problem, training instances were weighted, inverse-proportionately to the size of the smallest class they

belonged to. (the inclusion of weights in training led to an average 0.05 point improvement of f1 score for all models). Moreover, Adam optimization was used during training, and an EarlyStopping callback to prevent overfitting.



graph of the structure of RNN models: layer type + output shape

3. BERT was trained by importing the model, together with the fast tokenizer from the transformer dataset and then fine tuned for our classification task.
   the procedure for training bert follows a “pipeline”
   first we needed to decide how to tokenize the comment, meaning if dynamically or with a fixed padding. Because of computational bottleneck, we tokenized all datasets at once with fixed padding, then we needed to create dataloader to pass the input in batches to the model

   note that bert takes as input 2 arguments, the inputs ids and the attention masks, both as pytorch tensor

   then we trained the model using AdamW and the loss implemented in the model for multilabel classification i.e BCEwithlogitloss

   This loss function takes the raw logits for each class, applies a sigmoid internally to convert them into probabilities, and then computes the binary cross-entropy loss for each class independently. The loss values are averaged across all classes and across all samples in the batch, resulting in a single scalar loss value per batch.

   we take the average of all batches loss during the training visualization

Results and Analysis

statistical embeddings

ppmi

the following PPMI has shape 26616 rows × 26616 columns

...	26606	26607	26608	26609	26610	26611	26612	26613	26614	26615
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	5.901366	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	16.584991

the max value is 21.228848
Total elements: 708411456
Non-zero elements: 1470787
Sparsity: 99.79%

from the image above we can visualize the following flaws:

1) PPMI is biased towards rare words:

    - the highest PPMI is around 21, is achieved by the words at indexes (26490,26490), the words are both: “`bicth`”

      the word is misspelled and repeated 16 times consecutively in one comment, artificially inflating the PPMI score due to the rarity of the word and the repeated co-occurrence.

To mitigate this effect, and for computation efficiency, we did not consider words that co-occur only once in the PPMI.

note: co-occurrence of words with themself are not counted unless it happens that the same word is repeated inside the window size, as in this case

2) PPMI is almost completely zeros:
    - hence for computations we need to use specific algorithms
      e.g. scipy sparse matrix

tf-idf

Tf-Idf matrix has shape (25198, 56116) with the following characteristics:

Total elements: 1414010968
Non-zero elements: 936797
Sparsity: 99.93%
Max value: 1.0

- it shares the same problem as PPMI concerning sparsity
- it does not have the same problem with rare words

if we analyze the most important words in all documents i.e. the words with weight 1.0 we find the following:

There are 30 words with maximum weight, of which 24 are slurs, meaning that Tf-Idf prioritizes words that are highly distinctive but can also overemphasize toxic language.

This has the consequence, as we are gonna see later, of introducing biases.

LSA for PPMI and Tf-Idf

note: generally LSA is used as a term for term-document matrix while word embedding would be the correct name in case of term-term matrix

- There are different ways to solve both problems of PPMI (e.g. smoothing),
  In our work, we used Singular Value Decomposition (SVD), which helps simplify the matrix by keeping only the most important information. This makes the data less noisy, reduces the effect of rare words, and makes it easier to use for later tasks.

  statistics of the decomposed PPMI are: Total elements: 2661600
  Non-zero elements: 2661600

Sparsity: 0.00%
(26616, 100)

but most importantly, each dimension represents a latent semantic concept.

- For Tf-Idf we also needed to compute SVD since –even without rare words bias– the matrix was too sparse for efficient computations.
  Moreover, SVD helps capture deeper relationships beyond the raw tf-idf scores.

statistics of decomposed Tf-Idf

Total elements: 2519800
Non-zero elements: 2519400
Sparsity: 0.02%
(25198, 100)

In this section we are going to talk about latent semantic analysis and show how it worked in our dataset.

- LSA tries to uncover latent (hidden) semantic structure of words spread across the documents in case of Tf-Idf

- while for PPMI it aims to capture latent semantic relationships between words based on their co-occurrence patterns within a corpus i.e. it captures word similarity or associations more directly.

We analyzed the top K words (between 5 and 20) for some dimensions and found the following:

For PPMI the first dimension is related to toxicity, it contains slurs; while for example the third dimension is about nationality

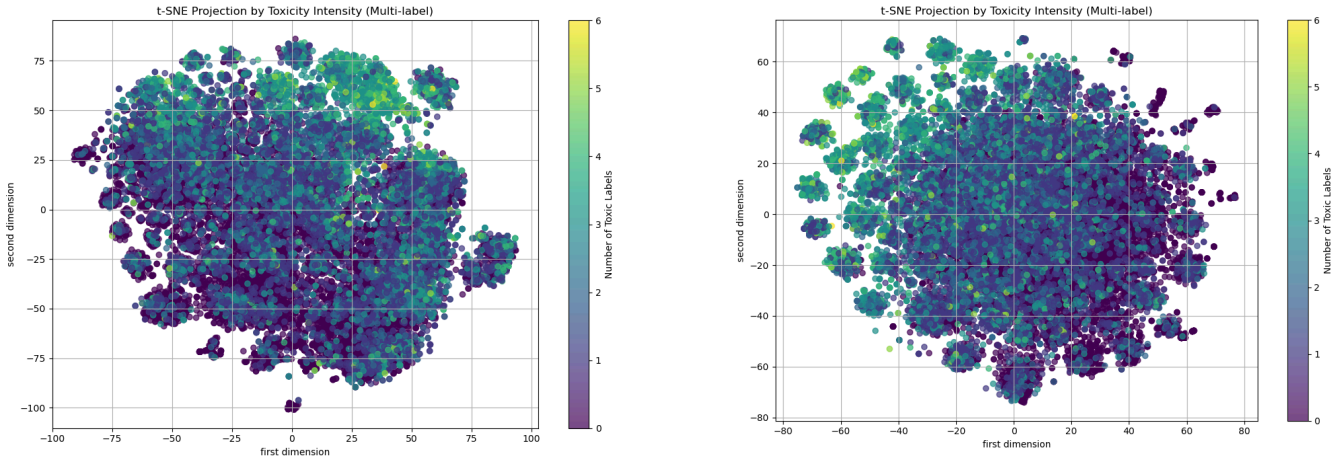
```
Top 20 words for dimension 3:
Word: state, value: 21.611066818237305
Word: country, value: 20.214658737182617
Word: south, value: 20.067211151123047
Word: american, value: 18.865007400512695
Word: british, value: 18.472301483154297
Word: north, value: 17.650806427001953
Word: world, value: 17.509180068969727
Word: united, value: 17.49245834350586
Word: west, value: 17.15933609008789
Word: national, value: 17.055753707885742
Word: city, value: 16.510847091674805
Word: born, value: 16.007598876953125
Word: states, value: 15.998442649841309
Word: government, value: 15.706395149230957
Word: countries, value: 15.617169380187988
Word: nation, value: 15.552037239074707
Word: please, value: -15.347258567810059
Word: land, value: 14.963475227355957
Word: european, value: 14.886735916137695
Word: greek, value: 14.804723739624023
```

for tf-idf instead we found that SVD yields far less interpretable outputs, for example:

the image on the right seems to be capturing controversial commentary but it's not easy to discern the meaning

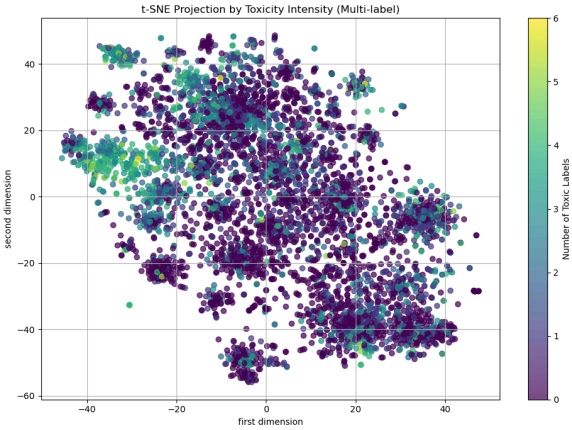
```
Top 15 words for dimension 1:
Word: yao, value: 0.5687743938980834
Word: ridiculed, value: 0.5683696062990689
Word: degradation, value: 0.568369452061588
Word: pwr, value: 0.568369452061588
Word: forged, value: 0.568369452061588
Word: fuckheads, value: 0.568369452061588
Word: loyal, value: 0.568369452061588
Word: ragad, value: 0.568369452061588
Word: socalled, value: 0.5680882434677226
Word: booze, value: 0.5125865909638081
Word: 'possible, value: 0.5096593176156058
Word: ideot, value: 0.49788069467513457
Word: raven, value: 0.4811273468438051
Word: batard, value: 0.45007983546063546
Word: cissonia, value: 0.44786979819766215
```

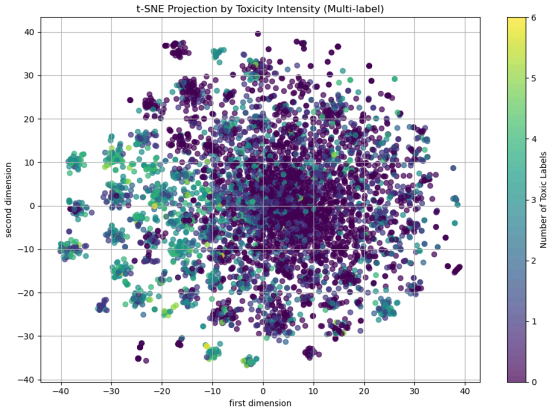
hence PPMI+SVD seems to be better for word embeddings and for the specific task of this project, but this will be analyzed better as follows:



on the left we have a T-SNE plot made by the PPMI+SVD transformed in document embeddings through averaging (dot-product)

on the right we have the plot of the Tf-Idf matrix+SVD





same image as above but with less comments, all of them where done setting perplexity at 50

they seem to represent comments similarly,
more conclusions will be drawn during prediction using neural models

model performance

The metrics used to evaluate models were:
– F1 macro– a combination of precision and recall that treats all classes equally, regardless of their frequency, making it suitable for imbalanced dataset.

– ROC AUC – a measure of the model's ability to give positive instances higher predictions than negative (a metric suggested in the dataset description).

model	threshold	F1	ROC AUC
RNN(sub)	0.7	0.52	0.93
RNN(w2v)	0.7	0.49	0.95
RNN(PPMI)	0.7	0.5	0.95
RNN(Tf-Idf)	0.7	0.5	0.93
BERT	0.4	0.67	0.99

validation scores of each model trained in the project

- micro vs macro average:
  macro average optimizes performance regardless of class imbalance

Threshold	Precision	Recall	Accuracy
0.20	0.5401	0.9114	0.8484
0.30	0.5691	0.8927	0.8588
0.40	0.5916	0.8773	0.8668
0.50	0.6137	0.8562	0.8743
0.60	0.6413	0.8391	0.8833
0.70	0.6677	0.8089	0.8899
0.80	0.6980	0.7679	0.8962

example of bert using micro averaging

Since we care about performance of underrepresented classes, we will use the threshold tuned on the macro-average for single comment inference. The only different thresholding is the one used for bert, that we are going to set 0.7, same as for every other model.
The following conclusions can be drawn from model performance:

1. Logistic Regression models do not benefit from more semantically rich embeddings (word2vec was trained with a window of 5, while PPMI a window of 2), because the format of the data processed by those models (one aggregate vector for each instance) hides any semantic relationships between words in a comment.
   The fact that PPMI embeddings lead to less linear separability between classes is not visible in the performance of the models, but the model trained on PPMI representations required almost `10x` more iterations to converge.

2. Depth and recurrent processing leads to an improvement regardless of embedding type.
   All RNN models reached similar levels of performance, but as will be shown later, their predictions on specific instances differ. This suggests that each model learned to exploit the features of a given embedding, but they all reached the same limit.
   An additional issue for the model trained to learn its own embeddings was the problem of dimensionality (the proportion between the number of trainable model parameters and the size of the training dataset). Other RNN models all used pretrained embeddings, reducing their number of parameters drastically, which presumably allowed them to train other layers more accurately.

3. The best model for this task is BERT, as to be expected, a transformer model pre-trained on large corpus did achieve higher results.
   As we will show later this results are not only visible in validation and test set metrics, but also during single comment inference and evaluation on the test set.

single comment inference and model biases

We start by testing the model on possible biases. To do so, we will use words seen in comments that might be flagged as inappropriate because of the context in which they were used in the dataset.

1) "*Freddy mercury was a gay man":*

models	toxic	severe toxic	obscene	threat	insult	identity hate
RNN(sub)	0.567	0.002	0.023	0.001	0.171	0.540
RNN(w2v)	0.803	0.035	0.217	0.008	0.554	0.802
RNN(PPMI)	0.974	0.025	0.179	0.003	0.588	0.967
RNN(Tf-Idf)	0.843	0.089	0.404	0.035	0.555	0.823
BERT	0.692	0.003	0.007	0.004	0.097	0.221

probability of outputs for every model

We can see that every model during training learnt to associate a single word, which is by itself not problematic, to a label.
Every RNN model learnt to associate the word with both *toxic* and *identity hate*, the two statistical embeddings and word2vec also learnt a correlation with *insult*.

Despite having a high probability for toxic, BERT does not flag it as identity hate or insult.

The mistake of the subword-trained model is a lot less pronounced than for other models. This is most likely caused by the fact that, dealing with uncleaned text, this model can learn the typical formatting of insulting comments, which is different from the text used above.

2) *“stupid european”*

models	toxic	severe toxic	obscene	threat	insult	identity hate
RNN(sub)	0.967	0.130	0.784	0.069	0.834	0.726
RNN(w2v)	0.931	0.124	0.610	0.087	0.703	0.793
RNN(PPMI)	0.971	0.063	0.705	0.028	0.826	0.962
RNN(Tf-Idf)	0.954	0.224	0.733	0.115	0.698	0.315
BERT	0.986	0.063	0.744	0.009	0.894	0.745

All of the models–aside from the Tf-idf based one– correctly classify the statement as toxic, identity hate and insult.
We can notice that all models learnt a correlation between “stupid” and the label *obscene*.

Upon checking the comments that contained the word “stupid”, it’s noticeable that the word is used together with stronger slurs flagged as obscene.

For the identity label, we can see that 4 models out of 5 classified it correctly, the reason is the following:
Tf-Idf + SVD did not succeed in creating informationally rich latent dimensions as PPMI did, the latter has a dimension for geographical area, as can be seen in the LSA section above. word2vec is semantically richer than Tf-Idf so a prediction closer to PPMI was to be expected, while BERT was trained on a bigger corpus and learns “context” from the phrase.

3) *“the monkey likes books”*

models	toxic	severe toxic	obscene	threat	insult	identity hate
RNN(sub)	0.223	0.017	0.044	0.007	0.081	0.284
RNN(w2v)	0.917	0.121	0.544	0.081	0.509	0.175
RNN(PPMI)	0.696	0.017	0.121	0.022	0.326	0.090
RNN(Tf-Idf)	0.728	0.154	0.394	0.186	0.403	0.252
BERT	0.392	0.001	0.011	0.001	0.039	0.002

The above table summarizes the problem of neural models trained with static embeddings and standard preprocessing (described above) which is that they tend to focus on single words inside documents disregarding a more complex structure.

A neural model trained with subword tokenization and without removal of stop-words does mitigate the effect, possibly because of a richer vocabulary and task dependent embeddings

BERT on the other hand learns a more complex structure thanks to attention mechanism, learning a pattern–the context– which is not considered by static embeddings.

Conclusion

In the project, we built and evaluated models on three different levels of complexity: a linear model (Logistic Regression), a deep neural network (RNN) and a transformer model (BERT). For each of those models, we developed an appropriate preprocessing pipeline, with a focus on the choice of embeddings. We then analyzed the performance of those models, focusing on the influence of different preprocessing choices and the interpretability of predictions.

Division of work on the project:
Malgorzata Ozarowska:
– text preprocessing, data preprocessing for different models
– word-frequency analysis
– word2vec embeddings
– Logistic Regression models
– RNN models

Alessandro di Piano:
– PPMI and Tf-idf matrices, SVD
– TSNE
– Latent Semantic Analysis
– BERT model and tokenizer
– model prediction comparisons

AI Assistance

The AI tool used during the development of the project was ChatGPT-4o.

In Ozarowska’s part of the project, the lines in the code marked with the #chat comment were generated using the following prompts:

```
1. how to perform the train test split on multilabel data
2. how to check if any digits appear in a string
3. how to convert a pandas column of textual entries into a txt file
4. how to append a list with a pre-defined number of zeros
5. how to show a covariance matrix
6. how do i convert a series of arrays into an array or a tensor
7. is it possible to give weight to instances in a multilabel classification
with binary cross-entropy loss?
```

for the part done by Di Piano, the following prompt were used:

1) how to apply a formula on only the non-zero entries of a sparse matrix
2) How does a T-SNE plot work, what are the argument and how to plot multilabel instances
3) given the following code (T-SNE plot), make it 3D
4) how to do latent semantic analysis of dimensions in svd reduced matrices?
5) how to check gpu availability and how to pass from cpu to gpu in training bert
6) is it possible to not pad every comment to a given length? If yes, how to do it for bert?
7) how to compute the forward pass using pytorch and what loss does bert use
8) how to make bert prediction usable in the following code?
9) print the prediction of the models in a nice tabular formatting

- In the code there are functions where is specified “chat helped” and “chat did this”.

Additionally, the same LLM was used to explain error messages in the debugging stage of the project.