PSL Research University

Master 2 IASD - NLP

Toxic Comment Classification

Authors :
Yassine Abou Hadid
Mehdi Bentaleb
Ismail El Hadrami

Supervisor :
Alexandre Allauzen

April 2023

Contents

1 Problem framing

1.1 Context and motivation

As of January 2023, there were 5.16 billion internet users worldwide, which is 64.4 percent of the global population. Of this total, 4.76 billion, or 59.4 percent of the world's population, were social media users.

The emergence of massive social networking websites like Facebook, Twitter and Reddit lead to millions of people posting content and commenting on them. People including children interact in the internet and there is a need to control the content and encourage a healthy environment and facilitate conversations, and ensure that they don't give up on stop expressing themselves.

Moderation of online discussions has become a key area of focus both for technology companies whose products include text interaction between users and from regulators since the nefarious aspect of negative online speech has shown serious impact on mental health. While technology companies employ armies of moderators, advancements in NLP using neural networks may provide larger degrees of automation and reduce the need for human moderation so that focus can be directed towards other content (videos, deepfake, propaganda) that cannot be automatically processed as efficiently as text.

The aim of this project is to explore the use of various machine learning models to detect and classify messages or comments that contain hateful speech, bullying, or other kinds of toxicity.

Our code can be found in this **repository**

1.2 Kaggle Challenge

Jigsaw and Google have founded Conversation AI team, a research initiative working on tools to help improve online conversation. One area of focus is the study of negative online behaviors. They've built a range of models and decided to create a Kaggle challenge to help them improve the current models. We have to build a model that's capable of detecting different types of toxicity like threats, insults and identity-based hate to help online discussion become more productive and respectful.

1.3 Dataset

In this project, we used the Wikipedia Comment Dataset that contains 159,563 comments with a category of toxicity for each one, labeled by human raters. There are 6 categories of toxicity : toxic, severe toxic, obscene, threat, insult, identity hate. A comment may belong to one or more classes and all comments are in English.

As seen below, we notice a few things - first, the training dataset is highly imbalanced with regards to comment classification labels. We see that almost 90% of the comments has no toxicity label. In addition, a brief investigation of the training set reveals that the percentages of toxic classifications are not even - 'toxic' ratings are much more prevalent (15,000 examples) with 'obscene' and 'insult' (around 8,000 each) at nearly half that amount and the rest at or below 1,500 examples each.

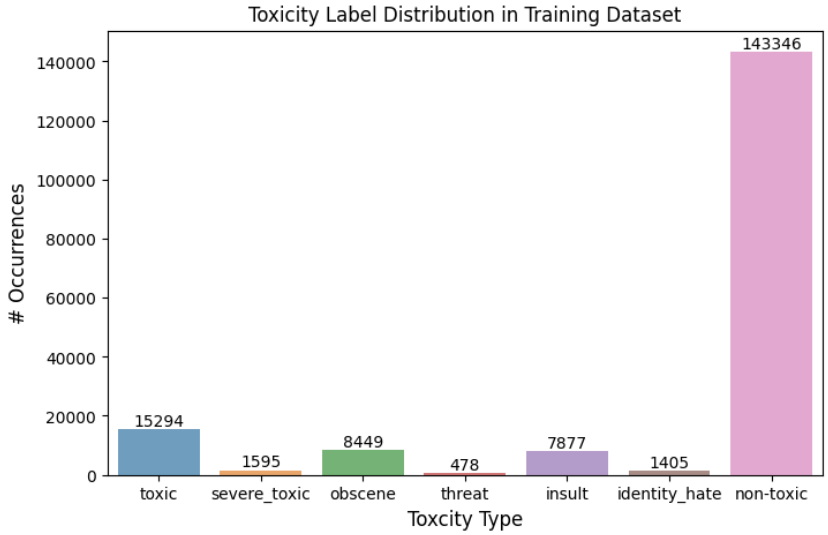

FIGURE 1 – Toxicity Label Distribution in Training Dataset

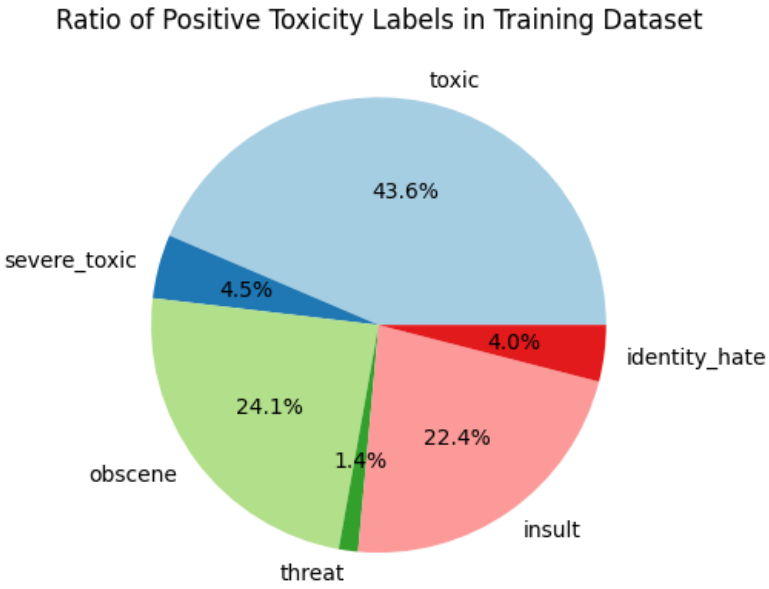

FIGURE 2 – Ratio of Positive Toxicity Labels in Training Dataset

We remark also that obscene and toxic comments are very correlated. It means that there is a high chance that we find insults in an obscene comment.

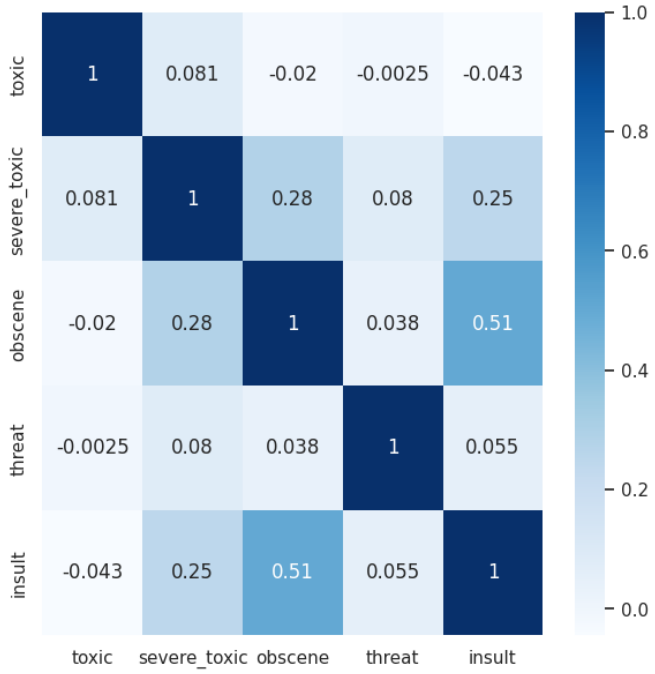

FIGURE 3 – Toxicity Label Distribution in Training Dataset

2 Experiments

2.1 Preprocess

The dataset is given in its raw, unprocessed form and necessitates several stages of cleaning prior to being able to undergo further processing. Consequently, a crucial initial measure involves examining the data and eliminating any irrelevant, confusing, duplicate, or potentially unprocessable portions.

Before breaking down the sentence into unique words by tokenizing the comments, we applied elementary processing steps, such as converting everything to lowercase to remove duplicates, specials characters such as numbers, removing stopwords and punctuation. We also filtered out non-ASCII characters after observing the results of feature engineering. We then lemmatize the comments and filter out comments with length below 3. Besides lemmatization, we also tried stemming but did not get a better result.

The raw dataset contained online links (identified as starting with ”http ://”), IP adresses (also identified with a regular expression) and emojis that needed to be removed.

2.2 Bag of words/TF-IDF

Bag of words (BoW) is a statistical language model used to analyze text and documents based on word count. It is based on a vocabulary of known words and a measure of their presence in the document. The vectorization in BoW vector gives features that unique words and feature values are word counts. There are two popular techniques associated with the bag-of-words approach : CountVectorizer and Tf-IDFVectorizer.

2.2.1 CountVectorizer :

It is a technique that converts a collection of text documents into a matrix of token counts. It builds a vocabulary from all the unique words in the documents and assigns an index to each word. Each document is then represented by a vector where each element corresponds to the count of the respective word in the document. The resulting matrix is often referred to as a "term-document matrix."

2.2.2 Tf-IDFVectorizer :

Tf-IDF is a more advanced technique that considers the importance of words in a document within a larger corpus. It calculates a weight for each word by multiplying the term frequency (how often the word appears in a document) by the inverse document frequency (how rare the word is across all documents in the corpus). This approach helps in giving higher weights to words that are more informative and discriminative.

In our case, we opted to use Tf-IDF to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. We also tried CountVectorizer. However, it is not performing as well as Tf-IDF.

2.3 Embeddings from scratch (LSTM)

In this section, we did an embedding for the comments from scratch. We did it following these steps :

- Cleaning phase : lowercase - remove spaces and punctation - remove all characters that are not numbers, letters or not part of the ASCII code
- Tokenization
- Removing the stop words (using NLTK)
- Lemmatization
- Padding to have the inputs of the same size (needed for lstm models.

In this section, we took a baseline model a neural network with Long Short-Term Memory(LSTM). As a reminder, LSTM is designed to handle the problem of vanishing gradients in traditional recurrent neural networks. It uses a combination of memory cells and gates to regulate the flow of information through the network and selectively remember or forget information over long sequences. The gates are controlled by sigmoid activation functions that output values between 0 and 1, which determine how much information should be passed on to the next cell state.

2.4 GloVe

GloVe (Global Vectors for Word Representation) is a popular method for generating word embeddings, based on matrix factorization techniques that leverage the co-occurrence statistics of words in a corpus to learn word embeddings. The GloVe algorithm operates by constructing a matrix of word co-occurrence counts based on a corpus of text. This matrix is then factorized using singular value decomposition (SVD) to generate a lower-dimensional representation of the co-occurrence matrix that captures the semantic relationships between words. The resulting word embeddings capture both the syntactic

and semantic relationships between words in a corpus. The key idea behind GloVe is that words that frequently co-occur in a corpus have similar meanings, and this can be used to learn high-quality word embeddings.

Here's an example of how GloVe generates word embeddings for the sentence : "The cat sat on the mat." The co-occurrence matrix will look something like this :

words	the	cat	sat	on	mat
the	0	1	0	1	0
cat	1	0	1	1	0
sat	0	1	0	1	1
on	1	1	1	0	1
mat	0	0	1	1	0

The rows and columns of this matrix correspond to each word in the sentence, and the cells represent the number of times each pair of words co-occurs in the sentence.

This matrix is then factorized using SVD to generate a lower-dimensional representation of the co-occurrence matrix that captures the semantic relationships between words. The resulting word embeddings can be used in a variety of NLP tasks, such as sentiment analysis, machine translation, and information retrieval.

The GloVe word embeddings for the words in the sentence "The cat sat on the mat" might look something like this :
the : [0.2, 0.4, -0.1, 0.3, -0.2]
cat : [0.5, -0.1, 0.4, 0.2, -0.3]
sat : [-0.1, 0.3, -0.2, 0.5, 0.4]
on : [0.3, 0.2, 0.5, -0.1, 0.1]
mat : [-0.2, -0.3, 0.4, 0.1, 0.2]

2.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) is another popular neural network model for NLP tasks, including language understanding and sentiment analysis. BERT is trained on a large corpus of text data, such as Wikipedia, to learn contextualized word embeddings that capture the meaning and syntax of words in a sentence. BERT uses a transformer architecture that processes the input text in both directions, from left-to-right and right-to-left, to capture the meaning of the entire sentence. BERT is pre-trained on a large corpus of text using two unsupervised learning tasks :

— Masked Language Modeling : BERT randomly masks some of the words in a sentence and then trains the model to predict the masked words based on the context of the other words in the sentence.
— Next Sentence Prediction : BERT is trained to predict whether a pair of sentences are consecutive or not.

Once pre-trained, BERT can be fine-tuned on a specific NLP task by adding a task-specific layer on top of the pre-trained BERT model. For example, for sentiment analysis, a classification layer can be added on top of BERT to predict the sentiment of a given sentence.

Here's an example of how BERT generates contextualized word embeddings for the sentence "I went to the bank to deposit some money." The word "bank" can have different meanings depending on the context of the sentence. BERT generates a contextualized

embedding for each occurrence of the word "bank" based on the surrounding words in the sentence. For example, the embedding for "bank" in the phrase "I went to the bank" will be different from the embedding for "bank" in the phrase "I saw a fish in the river bank". This contextualized word embedding allows BERT to capture well the meaning and syntax of words in a sentence.

3 Results

As the data in hand is highly imbalanced, our main metric for measuring model performance is F1-score, and since we have 6 labels, the F1-score would be the average of 6 labels. We will also take other metrics into consideration while evaluating models, e.g, Hamming loss and recall.

3.1 Baseline Models

We choose Naive Bayes as our baseline model, specifically Multinomial Naive Bayes. Also, we want to compare between different models, especially models that perform well in text classification. Thus, we choose to compare Multinomial Naive Bayes with Logistic Regression and Linear Support Vector Machine.

Model	Hamming Loss
Multinomial NB	0.0269
Logistic Regression	0.0256
Linear SVC	0.0284

Figure 4 presents a comparison between these different models after training them and see how these models perform on the test data.

Notice that Multinomial Naive Bayes does not perform as well as the other two models, while Linear SVC in general outperforms the others based on F1 score.

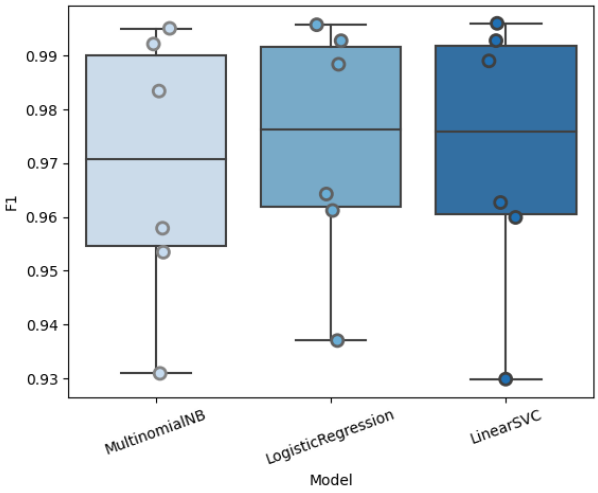

FIGURE 4 – F1 score boxplots

As seen in the preliminary section, the main concern is the imbalanced data, we decide to manually adjust class_weight for the models to see if we can achieve better results.

Since Logistic Regression and Linear SVM are performing better, we will focus on these two models. For display purpose, we will only include average F1 score, Recall, and Hamming Loss for comparison.

Model	F1-Score	Recall	Hamming Loss	Training Time
Logistic Regression	0.947	0.934	0.065	4.511
Linear SVC	0.951	0.941	0.058	5.162

Now we decide to do grid search to seek for the "optimal" hyperparameters for the basic models that we've chosen. Later we will make comparison based on the best model from each algorithm, since we have 6 different labels, tuning models for each label would be time expensive, so we will use the most common label "Toxic" to tune hyperparameters.

Model	F1-Score	Recall	Hamming Loss	Training Time
Logistic Regression	0.971	0.971	0.028	3.262
Linear SVC	0.973	0.973	0.026	54.655

Overall, we see that LogisticRegression slightly outperforms LinearSVC while taking considerably more time to train.

3.2 Boosting Methods

Since Ensemble learning helps improve machine learning results by combining several models and allows the production of better predictive performance compared to a single model, we want to see if ensembling could help us achieve better results.

To ensemble different models, we firstly tried some models based on tree boosting, then use a voting classfier to ensemble one of the boosting model with the basic models in previous parts.

Model	F1-Score	Recall	Hamming Loss	Training Time
AdaBoostClassifier	0.967	0.969	0.030	33.282
GradientBoostingClassifier	0.969	0.971	0.028	133.029
XGBClassifier	0.972	0.973	0.026	93.702

Since XGBoost classifier outperforms other two boosting models, we decide to go ahead with it.

Our ensemble model consisted of a Hard Voting Classifier composed of XGBoost, Linear SVC and Logistic Regression. Next, we can see its performance on the test set (without hyperparameters tuning).

Model	F1-Score	Recall	Hamming Loss	Training Time
Ensemble	0.973	0.973	0.026	106.558

Note : Ensemble model worked very well but still could not outperform LinearSVC since we did not tune the hyperparameters for the ensemble model.

3.3 What went wrong ?

1347 samples were misclassified as non-toxic which were actually toxic. We analyzed why the model couldn't recognize these words.

To do so, we first passed these raw comment strings through the same tokenizer and checked the common tokens. *ucking* is a common word in the test set, and it seems our classifier hasn't learned to classify it as toxic. It is interesting to note that this token wasn't common in our training set. That explains why our model couldn't learn it. It also gives us some food for thought on how we can improve our model further, such as going deeper on feature engineering (spelling corrector, sentiment scores, n-grams, etc) and using more advanced Deep Learning models.

3.4 Deep learning techniques

In this section, we're testing 3 approaches :

— a baseline approach that utilized a Bidirectional Long Short-Term Memory (LSTM) network with embeddings trained from scratch.

— a variation of the baseline approach that incorporated Glove's pre-trained embeddings with the Bidirectional LSTM architecture.

— the well-known BERT model, which is capable of producing state-of-the-art results on a range of NLP tasks, including text classification.

For the BERT model we use the already pretrained model distilbert-base-uncased and then we feed the embeddings to a feed forward network.

The models below have all been trained on 10 epochs using Adam optimizer and with Binary Cross Entropy loss

Model	Val F1-Score	Val Loss
LSTM (embeddings trained from scratch)	0.9681	0.0070
LSTM (GloVe embeddings)	0.9620	0.0085
BERT	0.978	0.06

4 Conclusion

In conclusion, our research project successfully implemented machine learning and deep learning models on the Toxic Comment Challenge Dataset, yielding convincing results. We were able to effectively classify toxic comments with a high level of accuracy, contributing to the field of natural language processing and online content moderation.

We were able to leverage machine learning and deep learning models in automatically detecting and flagging toxic comments, which can significantly enhance online community safety and promote healthier digital interactions. The models demonstrated exceptional performance in classifying various types of toxic comments, including those containing hate speech, personal attacks, and offensive language.

However, reflecting on our research, there are several aspects that we could have approached differently to further improve our findings. Firstly, we could have explored more advanced pre-processing techniques to enhance the quality of the input data. By carefully analyzing and cleaning the dataset, we could have potentially eliminated noise and biases that may have influenced the model's performance.

Furthermore, the inclusion of additional features or meta-data related to the comments, such as user information, timestamps, or contextual information, could have potentially improved the model's ability to understand the context and subtleties of toxic comments. This could have led to more nuanced classifications and a better understanding of the factors contributing to toxic behavior in online communities. Additionally, we could have investigated alternative deep learning architectures or ensemble methods to further enhance the model's performance.