Project Machine Learning
— Milestone 1 —

Johannes Constantin Keller, Moritz Wassmer, Ivan Akunevich

September 3, 2024

1 Introduction

This report is about the first milestone of Project Machine Learning in the winter term of 2023. The topic of our group is "Transformer-based Models for different NLP applications". More specifically, we implement the BERT (Devlin et al., 2019) architecture to classify toxic comments. For this milestone, our goal was to set up the baseline architecture of the model and perform a forward pass through the model, as well as exploring the specificities of our dataset. The data comes from the 2017 Kaggle toxic comment classification challenge (Jeffrey et al., 2017). It is a classification task, requiring to assign multiple labels for seven types of toxicity (including "non-toxic") to the comments.

The report is organized as follows: In section 2, we introduce the dataset with respect to the task in general, the characteristics, how we extract features and which further transformation steps are applied. In section 3, we describe our model, more specifically the BERT-architecture and review related literature. In the end, we discuss evaluation methods for the given task and provide first results from our baseline and discuss them.

2 Dataset

In this section, we briefly describe the characteristics of dataset and feature extraction approaches in neural networks, compared to classical ones and questions, regarding data. To download and process data, the *Hugging Face library*[1] was used. This allows, to process data faster and simplifies integration with workflows. Besides, it provides connection with transformer models.

2.1 Dataset Characteristics

The data set comprises 159571 comments from Wikipedia, obtained from Kaggle[2]. There are 6 types of labels: *toxic, severe toxic, obscene, threat, insult, identity hate.* Each label corresponds to subjective descriptions: *Toxic* represents very bad, unpleasant, or harmful comments. *Severe toxic* can be described as extremely bad or even offensive. *Obscene* is stated as disgusting by accepted standards of morality. Intention to cause pain or damage is *threat.* Disrespect is represented by *insult. Identity hate* corresponds to racism and other forms of hate speech against specific groups of society. Each comment can correspond to more than one label. A comment is considered *clean* when none of the others labels are *True*. The data set has neither missing values, nor duplicates. Since the data is taken from an open-source platform, and due to its nature, some comments can lack sense, thus impacting the quality of our data.

There is a huge difference between two categories: There are almost ten times more clean comments than toxic ones (143346 compared to 16225). Thus, the data set is unbalanced. From Figure 1, it can be observed, that the distribution of toxic label occurrences mostly falls into toxic and obscene. But as mentioned before, the comments can fall into more than one category.

[1] `https://huggingface.co/datasets/jigsaw_toxicity_pred`
[2] `https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/data`

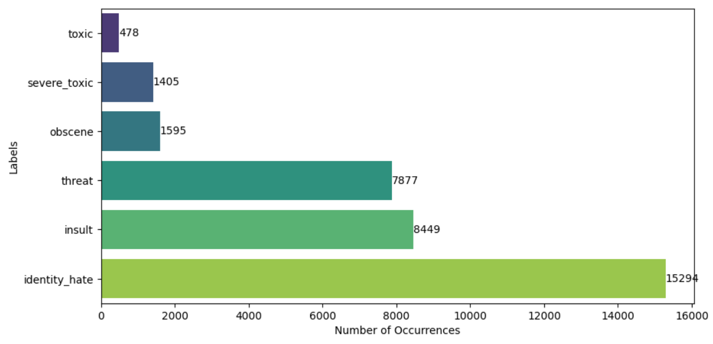

Figure 1: Distribution of toxic comments

A technique that is often used for visualizing data, is *word cloud* – a graphical representation of word frequency that gives greater significance to words that appear more frequently in a source text. The larger the word in the visual, the more common it is in the text. While comparing Figure 2 and Figure 3, the difference between the respective categories becomes clearly visible.

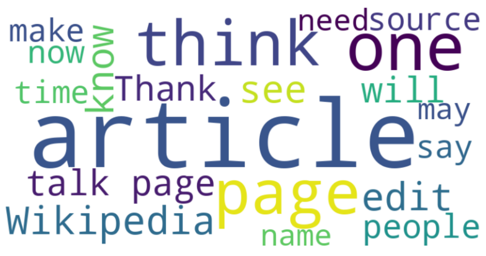

Figure 2: Word cloud for clean comments

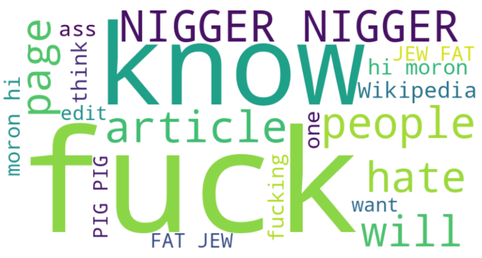

Figure 3: Word cloud for toxic comments

Figure 4 shows, that the average length of most of the labels, regarding the token representation, is somewhere between 75 and 90 tokens. The only outlier is *severe toxic* with an average length of around 133 tokens.

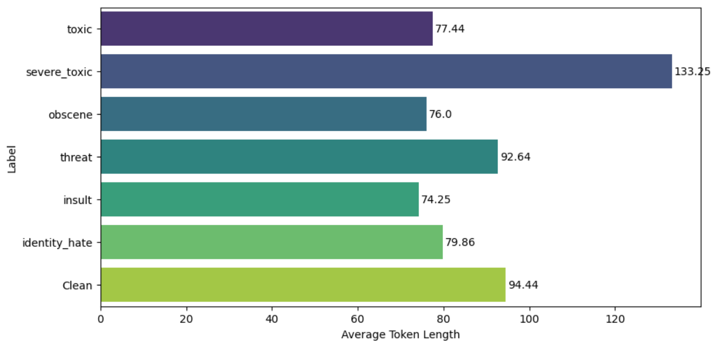

Figure 4: Average comment length in tokens per label

Feature Extraction and Transformations In order to classify the toxic comments, the classical methods for feature generation can be used. One of them is *Bag of Words* (BoW) (Qader et al., 2019). *BoW* represents the text as description of occurrences of specific words, defined in so-called *dictionaries*. An example can be: "I love cats", which will be represented as [1,1,1,0] — in the case, the dictionary has another word, not presented in mentioned phrase (encoded as 0).

Another approach, which is commonly used for text classification tasks, is *Term Frequency*(TF)-*Inverse Dense Frequency*(IDF) (Das and Chakraborty, 2018). *TF-IDF* eliminates some disadvantages of the previously mentioned method, by introducing a special metric, that evaluates each word not only by highest occurrence in the presented text, but at same time, by its rarity across the whole data collection.

The last method is *N-grams* (Cavnar et al., 1994), which is used to determine context by analysing neighboring words.

All these methods have limitations in the sense, that they treat words or phrases in isolation, ignoring the context. *Bidirectional Encoder Representations from Transformers* (BERT) uses trainable embeddings. That means, the vectors of each token are jointly trained in the given task, such as the other weights in the network. This gives a meaning to every token with respect to the task at hand. In contrast to classical embeddings based on words or characters, BERT splits the input sequences into *WordPiece-tokens*. The advantage of this so-called *WordPiece-embedding* (Wu et al., 2016), in comparison to learned word- or character embeddings is, that it learns to split up words into their parts. This makes it possible for the model to understand, that words like "household" and "housekeeper" have a shared meaning in the part "house". Which words are split up into pairs, is inferred by a score:

$$score = \frac{frequency\ of\ pair}{freqency\ of\ first\ element * freqency\ of\ second\ element} \tag{1}$$

This score makes the tokenizer more prone to keep a word together, that occurs more often in the text, than its respective parts and vice versa.

To compensate for the differences in length of the comments, padding tokens are added. Then, the BERT-specific CLS (classification) and SEP (separator) tokens are added. The CLS token is a special placeholder, which is necessary to obtain a representation which is not associated with other words in the sequence. The SEP token has can be used to differentiate between different input sequences. For example "toxic comment classification task" is encoded in tokens as: [101, 11704, 7615, 5579, 4708, 102]. To retain the original sentence, a decoding method can be used, that returns text and corresponding tokens: "[CLS] toxic comment classification [SEP]".

The other components that make up the BERT embedding, as part of the feature extraction, is discussed in section 3.2.

Good Features Definition Good features for machine learning in general (1) should appear often in the dataset and be representative of the underlying ground truth. (2) They should be easy interpretable for humans and have an obvious meaning. (3) They should be uncorrelated with other features, because otherwise these features provide more or less the same information. (4) Furthermore, good features should provide enough information as necessary about the data required for the machine learning task and have not mostly have the same value.

In our context, (1) good features (comments) are representative of their target categories, in order to provide a "understanding" of these, for the model. (2) They should be clearly understandable, have a low frequency of orthographic errors and have a similar length, to be comparable. (3) Good features are non-overlapping in target categories, even if it is a multi-label approach, to define clearly, what specifies a certain label. (4) Finally, they comprise *several attributes*, humans would consider toxic and not just *one*, like only keywords for instance.

Exploration of Questions The most interesting question about the data is maybe, what makes a model decide for certain labels? This can be explored at the fringes of this classifiability: What about *sarcastic* comments, that do not contain obvious keywords, a machine learning algorithm would certainly detect, but are to be labeled "toxic" by a human being? Another feature of relevance would be *fake facts* and *conspiracy theories* in this regard. At this point, one could observe, how much the model learned to infer toxicity from the context.

Regarding labeling, the separation between some categories of toxicity appears subjective. For instance, there might be cases in which it is hard to specify the difference in terms of keywords between "insult" and "obscene" or "toxic" and "severe toxic". The task for the model is hence, to reproduce a human "understanding" of these categories, meaning the human labeling, as there is no clear definition of a class available. Since multiple labels can be $True$ at the same time, the question arises: Why is not every comment also *toxic* which falls under one of the other 5 categories?

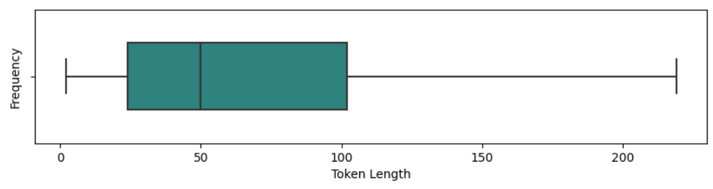

Figure 5: Distribution of tokens length

Another question, that can be asked, is: Which input size is sufficient for the model, to classify comments? From (Figure 5), it can be observed, that the majority of the data is represented in a range from 25 to 100 tokens per comment. This number can be also potentially decreased, due to the fact, that toxic comments in average don't exceed a length of about 90 tokens (cf. Figure 4). A balance has to be found between increasing the length, for better classification, and efficiency of the model.

3 Baseline Method

In this section, we briefly describe deep learning approaches prior to BERT and highlight the advantages of BERT. Afterwards we describe our baseline model.

3.1 Literature Review

Following, we compare transformers *Self-Attention* with *Recurrent Neural Networks* (RNNs), which were previously used for sequence modelling. Afterwards, we compare BERT to other Language models.

Self Attention vs. RNNs Prior to transformers, RNNs such as *Long short-term memory* (LSTMs) were *State of the Art* (SOTA) for sequence modeling, such as language modeling. The manner RNNs process sequential data, is by iteratively calculating the hidden state, which depends on preceding hidden states. This sequential computation makes parallelisation within training samples impossible. In contrast to RNNs, transformers Self-Attention mechanism is constant with respect to the input size, making the computation of a Self-Attention layer more efficient[3] compared to a recurrent layer. At the same time, long-range dependencies can be learned way easier when the paths, forward and backward signals have to go through between combinations of different inputs, is small (Vaswani et al., 2017).

Language Models Famous examples for preceding language representation models are *Embeddings from Language Models* (ELMo) (Peters et al., 2018) and the *Generative Pre-trained Transformer* (GPT) (Radford et al., 2018). Both of these models are *unidirectional architectures*, meaning every token can only use information from previous, either left or right, tokens. A main advantage of BERT is, that it uses the whole context (both preceding and succeeding tokens) at the same time, making it more suitable for *sentence-level tasks* (Devlin et al., 2019). Since toxic comment classification can be considered a sentence-level-task, it is reasonable to expect this model to be , regarding the classification performance potential, to unidirectional models.

3.2 Model Architecture

BERT is a language representation model, which stands for *Bidirectional Encoder Representations from Transformers* (Devlin et al., 2019). As the name suggests, it makes use of the *Transformer-Encoder Architecture* proposed by Vaswani et al. (2017). For this milestone, we are only going to introduce the high level concepts of BERT, instead of going into detail about the exact layers setup, parameters and computations.

[3]As long as the sequence length is smaller than the representation dimensionality (Vaswani et al., 2017)

BERT-Embedding The text sequences are first inputted into an *embedding layer* where the tokens (see section 2.1) are transformed into a high-dimensional representation. Since transformers do not know about the positions of the tokens (Vaswani et al., 2017), *positional information* of the tokens are injected, using sine and cosine functions. Also a *segment embedding* is added which can be used for the segmentation of sentences. Segment embeddings are equal for every non-padded token in a sequence for toxicity prediction, because we do not need to distinguish between different types of sequences.

Transformer-Encoder Bert consists of multiple layers of Transformer-Encoders (Vaswani et al., 2017). Each encoder consist of a variety of different layers like *layer normalization*, *dropout*, *non-linearities*, *linear layers*, *residual connections* and most importantly *multi-headed Self-Attention*. BERT-Base uses 12 layers of encoders.

Self-Attention allows a model, to assign varying degrees of importance to different elements in a sequence, enabling it to capture contextual information within the data. Multi-headed Self-Attention extends this idea, by using multiple sets of weights, called *Attention heads* in parallel, extracting relationships independently from different representational sub-spaces (Vaswani et al., 2017).

Toxicity-Head The last layers of the architecture take the representations of the BERT-Encoder and filter them, to only use the hidden representation corresponding to the CLS token, which is the special token to capture global representations (Devlin et al., 2019). We then use a linear layer to transform the representations into a vector, that matches the number of classes to predict (6). Usually one would also use a sigmoid layer to transform the output into probabilities, but since we are using *BCE loss with logits*, pytorch does this automatically (see section 3.4).

Implementation Since there are various different configurations of BERT, we chose to use the original *BERT-Base*. There are many ways to implement BERT in terms of code organization. Since we knew in advance, that we were allowed to use pre-trained weights, we decided to implement the model in a way, that enables loading the state dictionaries in a convenient fashion. These pre-trained weights where at this state of work not yet applied. Therefore, we organized the classes in a nested manner, using inner classes similar to *huggingface bert-base-uncased* [4]. Since we want to avoid passing down parameters in a nested way, which is prone to mistakes, we implemented most parameters as global constants. Depending on the need for *hyperparameter-optimization*, we might later implement a BERT configuration class.

3.3 Training Procedure

The general idea of the training follows the *Transfer Learning paradigm*. BERT is pre-trained on unlabeled text, to obtain bidirectional representations. Afterwards, the last layer(s) of the model can be replaced by a task specific head to adapt the model for different tasks. Subsequently, the model is fine-tuned on the specific task. In the following two paragraphs, the two training procedures (Devlin et al., 2019) are briefly described. We dropped the pre-training procedure from the submission, since it is not a requirement for the project. We plan to simply load the pre-trained weights instead of pre-train it from scratch. In our current implementation, we train our model from scratch on the toxicity task. Nevertheless, we are going to briefly describe the pre-training procedure.

Pre-training To obtain the representations, self-supervised pre-training on unlabeled text is performed by using two different tasks, namely 1) *Masked Language Modeling* and 2) *Next Sentence Prediction*. Both of these tasks are performed simultaneously. The datasets used for pre-training are *bookcorpus* (Zhu et al., 2015) and Wikipedia [5]. Masked language modeling refers to replacing some of the tokens in the input sequence by a masking-token. *Next sentence prediction* is a task which boils down to providing the model two sentences at the same time, and training the model on classifying, whether the second sentence succeeds the first sentence.

[4] `https://huggingface.co/bert-base-uncased`

[5] `https://dumps.wikimedia.org`

3.4 Toxicity Prediction

Since toxicity in the task is broken down into several categories (see section 2), while at the same time not being mutually exclusive, the problem at hand is called a *multi-label classification* task. Therefore the loss cannot be calculated as a "difference" between *one* distribution of the data and *one* distribution of the model for a data point, since the labels cannot be seen as states of a random variable.

Multi class Classification There are different ways to model this kind of task. A *classifier chain* (Read et al., 2011) is a method of transforming the classification into a chain of subsequent binary classifications. The outputs of previous classifiers are inputs to the next classifiers. Furthermore, there are approaches like the *transformation into a multi-class classification problem* (Spolaôr et al. (2013)). This comprises assigning new classes to each possible permutation of labels. There is also an approach called the *binary relevance* method (Read et al., 2011). It divides the task into multiple binary classification problems. Multiple classifiers are trained to solve the task for each label independently.

All of these approaches were discarded due to complexity reasons for the baseline. We decided to go for a similar solution than the *binary relevance* method. But in contrast to train multiple BERT models for different classes, we adjusted the last layer of neurons, to match the number of classes in the dataset. Hence, each output neuron predicts the probability for one of the labels. The loss function is now a composite of multiple individual losses, which is described in the following paragraph.

Loss Function We construct the loss function by calculating the BCE (*binary cross-entropy*) between the distribution of each component of the output vector of the model $\sigma(x_n)$ and the corresponding distribution of the label vector $y_{n,c}$ for one data point. The individual values (see Equation 3) are averaged for each batch, to obtain a scalar loss.

To account for the model not disregarding rare classes, due to the observed class imbalance (see subsection 2.1), the loss is weighted, depending on the class c by multiplication with the ratio (see Equation 2). We use pytorchs *binary cross-entropy loss with logits*[6] as a loss function. The loss function applies the *log*() to the output of the *sigmoid function* σ, which is numerically more stable[7] than the standard *binary cross-entropy loss*. It is defined as:

$$p_c = \frac{dataset\ size}{number\ of\ comments\ labeled\ c} \tag{2}$$

$$\ell_{n,c} = -\left[p_c y_{n,c} \cdot \log \sigma(x_{n,c}) + (1 - y_{n,c}) \cdot \log(1 - \sigma(x_{n,c}))\right] \tag{3}$$

For fine-tuning, we train the pre-trained base model and the toxicity head together on the given task. This approach is the most feasible and is comparable to the proposal by Rezaei-Dastjerdehei et al. (2020). We consider researching more sophisticated loss functions for the next milestones.

Overfitting In the current state of the implementation, we are happy to see convergence (see subsection 4.3) and do not care about overfitting. In fact, we see it as an indication for having no mistakes within the pipeline. Nevertheless, we want to address this issue, since this is going to become an important part of this project. BERT provides several tunable hyperparameters, such as dropout probability or layer normalisation. We can also transform our architecture by reducing embedding dimensionality and number of encoder layers which would reduce model complexity.

4 Evaluation

In this section, we are at first going to describe our considered measures for evaluating our models, given the task. Afterwards, we discuss potential differences considering the input features, which are able to discriminate the different classes. In the end, we present the first results of our baseline.

[6]https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

[7]https://medium.com/dejunhuang/learning-day-57-practical-5-loss-function-crossentropyloss-vs-bceloss-in-pytorch-softmax-vs-bd866c8a0d23

4.1 Methods

In the baseline model, next to the loss function described in section 3, we consider multiple measures for evaluation. *Accuracy*: Due to the discussed class imbalance, this has to be interpreted with care. If class imbalance is present, a classifier can potentially increase its performance by always predicting the over-represented class and completely disregard the other classes. Nevertheless, it can be a useful measure for diagnostics. It is defined per class as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

with TP as *true positive*, TN as *true negative*, FP as *false positive* and FN as *false negative* classifications.

A more suitable measure is AUC-ROC (cf. Fawcett (2006)). It stands for *Area Under The Curve - Receiver Operation Characteristics* and tells, how much a model is able to separate between classes in a binary classification setting. If the AUC is close to one, the model is good at differentiating between two classes, if it is close to zero, it performs poorly. The ROC-curve is computed over several thresholds for the binary classifier and is plotted with TPR against FPR (Equation 5), with TPR as *True Positive Rate* and FPR as *False Positive Rate* (Fawcett, 2006).

$$TPR = \frac{TP}{TP + FN} \qquad FPR = 1 - \frac{FP}{TN + FP} \tag{5}$$

AUC-ROC is defined as the area under the ROC-curve, which can be found by approximation in discrete settings. In a multi-label classification setting, there will be N ROC-curves for N classes, each for thresholds for binary classification between the respective class and all the other classes. This can be averaged to receive a total score. This evaluation method is also the used metric for the Kaggle competition (Jeffrey et al., 2017).

4.2 Class Differences

One obvious, distinctive feature for the model are certain keywords (see Figure 4). Comments including words like "dickhead", are very likely to be insults. But given certain contexts, it might not be an insult: For example, the sentence "he's not a dickhead" is not an insult. Nevertheless the model should be able to attend to the word "not" in this case and classify this correctly. On the other hand, toxic comments can also be sarcastic, which we expect to be harder for our models to detect.

4.3 Baseline results

After some iterations of debugging the model, we were able to see first signs of convergence. Initially, we forgot the residual connections, which resulted in deeper BERT variants with a large number of layers not converging properly. The accuracy was stuck at around 50 percent or worse, whereas configurations with only one transformer encoder were able to show first signs of success. We assumed it to be a sign of vanishing gradients(cf. Hochreiter (1998)), even though we did not inspect the model in detail.

For our final run, we ran a training of ten epochs on the entire training data set, using 512 comments per batch and a sequence length of 64 tokens. The baseline model did converge at an average training loss of 0.41 and an accuracy of around 92 percent in the ninth iteration. The model did classify around 88 percent of the $True$-labels correctly (TPR), which means, the weight-balancing had an effect. The test results support these findings, in the ninth iteration, the average training loss was slightly higher with 0.59, a TPR of 88 percent and an accuracy at 87 percent even.

5 Discussion

In summary, even though we did not apply the pre-trained weights from the BERT-Base model yet, the training runs exceeded our expectations of achieving only a mere forward pass. The baseline model at this point showed signs of the ability to separate the classes successfully.

The average comment length of around 90 tokens (see Figure 5) is above the maximal sequence length of 64, which we used for our first evaluation runs. We expect the results to be better, once we use a higher sequence length for the model, such that more text can be processed in the case of longer comments. Using the pre-trained BERT-Base weights in the next milestone, we do not only raise the sequence length to 512 tokens, but do also profit from the pre-trained representations as a better initialization for the weights.

The main challenge in this classification task is the multi-label approach combined with the high imbalance between the labels. Several techniques to solve these issues were discussed above. It could hence be interesting, to explore the results of a classifier chain with weighted loss functions in comparison to the binary relevance method with a single weighted loss function.

For the next milestone, we plan to implement a more sophisticated evaluation measure like ROC-AUC. We will also take a closer look at which of the classes the model predicts better, in contrast to only regarding aggregated measures over all classes. Also, we want to research, if there are more suitable loss functions.

Once everything is set up, we then plan on doing hyperparameter optimization, to obtain the full potential of the model. This also means, we are going to split up the test set into test and validation to avoid leaking information about the test set into the model.

References

W. B. Cavnar, J. M. Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175, page 14. Las Vegas, NV, 1994.

B. Das and S. Chakraborty. An improved text sentiment classification model using tf-idf and next word negation. *arXiv preprint arXiv:1806.06407*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2005.10.010. URL `https://www.sciencedirect.com/science/article/pii/S016786550500303X`. ROC Analysis in Pattern Recognition.

S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998. doi: 10.1142/S0218488598000094.

S. Jeffrey, E. Julia, D. Lucas, M. Mark, and C. Will. Toxic comment classification challenge, 2017. URL `https://kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge`.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In M. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL `https://aclanthology.org/N18-1202`.

W. A. Qader, M. M. Ameen, and B. I. Ahmed. An overview of bag of words; importance, implementation, applications, and challenges. In *2019 international engineering conference (IEC)*, pages 200–204. IEEE, 2019.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, December 2011. ISSN 1573-0565. doi: 10.1007/s10994-011-5256-5. URL `https://doi.org/10.1007/s10994-011-5256-5`.

M. R. Rezaei-Dastjerdehei, A. Mijani, and E. Fatemizadeh. Addressing imbalance in multi-label classification using weighted cross entropy loss function. In *2020 27th National and 5th International Iranian Conference on Biomedical Engineering (ICBME)*, pages 333–338, 2020. doi: 10.1109/ICBME51989.2020.9319440.

N. Spolaôr, E. A. Cherman, M. C. Monard, and H. D. Lee. A comparison of multi-label feature selection methods using the problem transformation approach. *Electronic Notes in Theoretical Computer Science*, 292:135–151, 2013. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2013.02.010. URL `https://www.sciencedirect.com/science/article/pii/S1571066113000121`. Proceedings of the XXXVIII Latin American Conference in Informatics (CLEI).

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.