Text Classification with Linear Models and BERT

1. INTRODUCTION

Text classification is one of many Natural Language Processing (NLP) applications that's gaining traction in the past decades. Online news and social media, which has been considered as one of the most powerful forces, often able to shift general population's political view and spark a manifestation of real-world phenomenon, presents itself not only as a gigantic source of text data but also as a domain where text classification can find important uses. Two of the most popular use cases that this project is focusing on, are classification of fake and real news as well as identification of toxic social media comments.

The challenges of building a robust NLP model starts with constructing a numerical representation of a text, termed vectorization.

Perhaps the most primitive embedding method is Bag of Words[1] (BoW) in which a document is represented by a vector $x = [x_1\ x_2 \dots x_d]$ with x_i corresponding to the count of the word V_i, the i-th word in vocabulary V, in the document. The vocabulary $V = [V_1\ V_2 \dots V_d]$ encompasses a subset or the entirety of words present in the dataset of many documents.

TF-IDF method[2] weighs each term based on its importance in the dataset. Term Frequency (TF) $tf_{i,j}$ is a frequency of occurrence of term i in document j, defined as follows:

$$tf_{i,j} = \frac{n_{i,j}}{\text{number of terms in document j}}$$

Inverse Document Frequency (IDF) idf_i is a measure of term i's importance in the dataset and is defined as

$$idf_i = log\frac{\text{number of documents}}{\text{number of documents with term i}}$$

TF-IDF score for each term can then be calculated as $tf_i df_{i,j} = tf_{i,j} * idf_i$. High score is assigned to terms with high frequency only in a few documents.

Once vector representations of documents are obtained, standard classification algorithms can be applied.

BERT (Bidirectional Encoder Representations from Transformers)

BERT is considered as the state-of-the-art in text classification. BERT is essentially a specifically designed neural network that is derived from transformers architecture. The full explanation of how transformers work can be found in the original paper[3]. Transformer was created to do sequence-to-sequence modelling, for instance to perform machine translation. While transformers consist of 2 parts: encoder and decoder, BERT is developed to generate a language model and hence only the encoder part is necessary.

BERT typically comes with models pre-trained with different corpuses for a problem termed Masked Language Model (MLM). In MLM, 80% of input words are replaced with [MASK] tokens, 10% with random words and the remaining 10% with the original words.

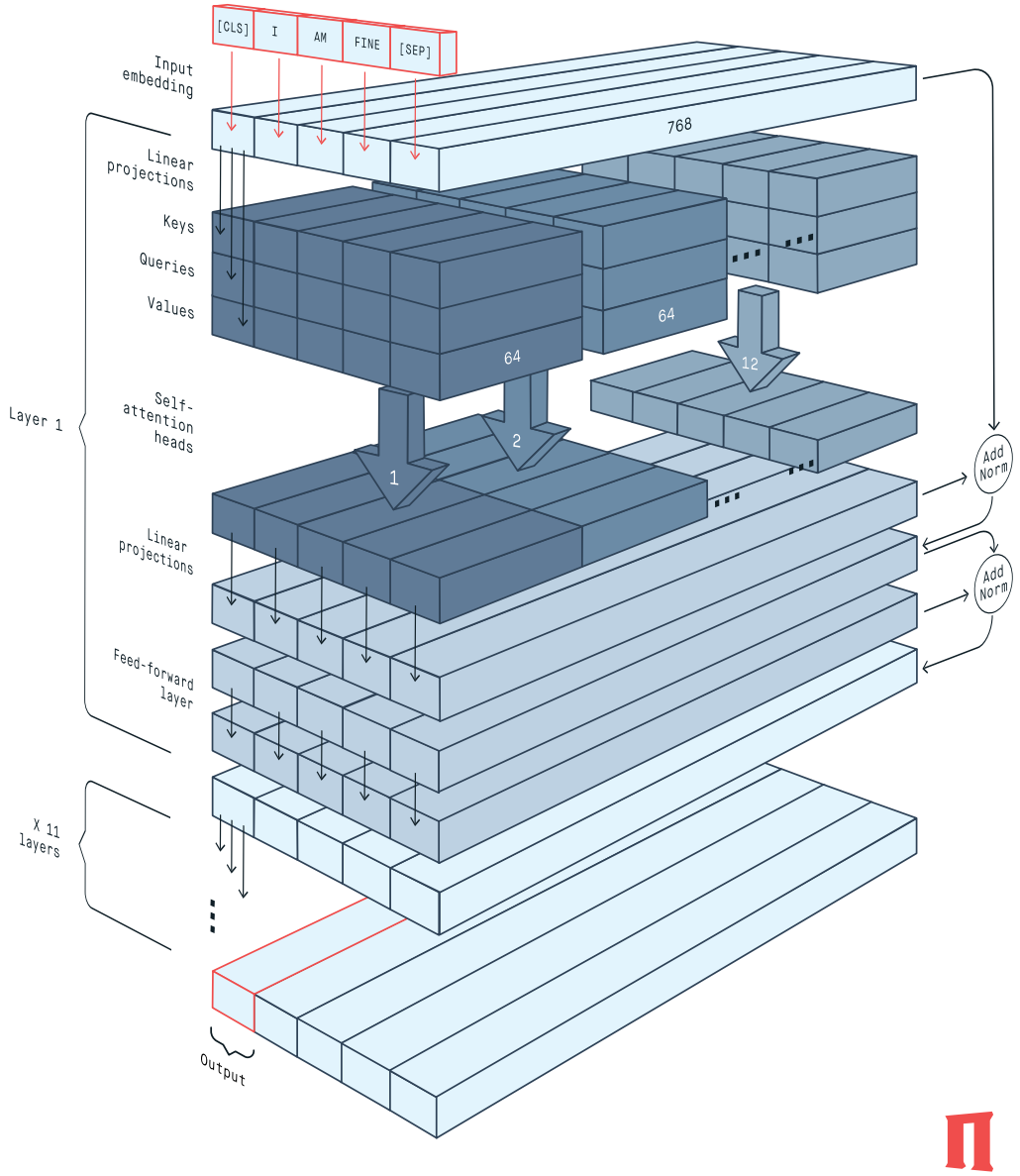

Figure 1. BERT English architecture[4]

BERT architecture

The full BERT architecture is shown in Figure 1. This section breaks down each layer of BERT and the intuition behind the mathematical operations involved.

Input encoding layer tokenizes input text and vectorizes each word into a $\mathbb{R}^{768}$ vector. BERT introduces [CLS] and [SEP] token to mark the beginning and end of sentences. Since the number of tokens must be fixed, input sequence is truncated or padded (with a specific padding token) to a fixed length specified during model training. The resulting $A \in \mathbb{R}^{768 \times n}$ matrix, where n is the number of tokens in an input sequence, is then passed to the next layer.

Keys, Queries and Values (KQV) layer performs 3 linear projections on the input encoding. The linear projections typically reduce the dimensionality from 768 to 64. This procedure is duplicated 12 times referred to as heads.

For i-th head, keys K_i, queries Q_i, and values V_i are defined as follows.

$$K_i = W_{i,K}A, \qquad Q_i = W_{i,Q}A, \qquad V_i = W_{i,V}A$$

Where $K_i, Q_i, V_i \in \mathbb{R}^{64\times n}$ and $W_{i,K}, W_{i,Q}, W_{i,V} \in \mathbb{R}^{64\times 768}$.

The keys and queries specifically can be thought of as projections focusing on different directions of vector space corresponding to different semantic aspects. Values are yet another $\mathbb{R}^{64\times n}$ projection performed similarly to keys and queries focusing on different semantic direction.

Self-attention heads are where keys, queries and values are combined to form contextual embeddings. First, for i-th head, a product of keys and queries are computed to produce an $\mathbb{R}^{n\times n}$ matrix

$$M_i = {K_i}^T Q_i$$

Where $M_i \in \mathbb{R}^{n\times n}$ and $K_i, Q_i \in \mathbb{R}^{64\times n}$. Then column-based softmax is applied to this matrix to produce $M_{i,softmax}$ where the sum of each column is 1. Finally, contextual embedding matrix B_i is produced from values matrix V_i in the following expression.

$$B_i = {V_i}^T M_{i,softmax}$$

Where $B_i, V_i \in \mathbb{R}^{64\times n}$. There are 12 self-attention heads each with its own keys, queries and values and the resulting matrices $B_i, \dots, B_{12}$ are concatenated to form B, an $\mathbb{R}^{768\times n}$ matrix.

Linear projections layer, as the name suggests, produces C, a linear projection of B

$$C = W_{BC}B$$

Where $C \in \mathbb{R}^{768\times n}$ and $W_{BC} \in \mathbb{R}^{n\times n}$. Addition followed by normalization was then performed on C to produce D.

$$D = norm(C + A)$$

Feed-forward layer then passes D through a hidden layer with dropout, maintaining the same dimensionality, following which D is added to the output of this hidden layer, termed E and finally normalized to produce F as described in the following expressions

$$D \xrightarrow{feed-forward} E$$

$$F = norm(E + D)$$

Where $D, E, F \in \mathbb{R}^{768\times n}$. F then goes into the second transformer layer starting with linear projections in the KQV layer. There are in total 12 transformers layers in BERT architecture.

To perform classification task additional 512-neuron hidden layers were attached to the end of BERT block as illustrated below:

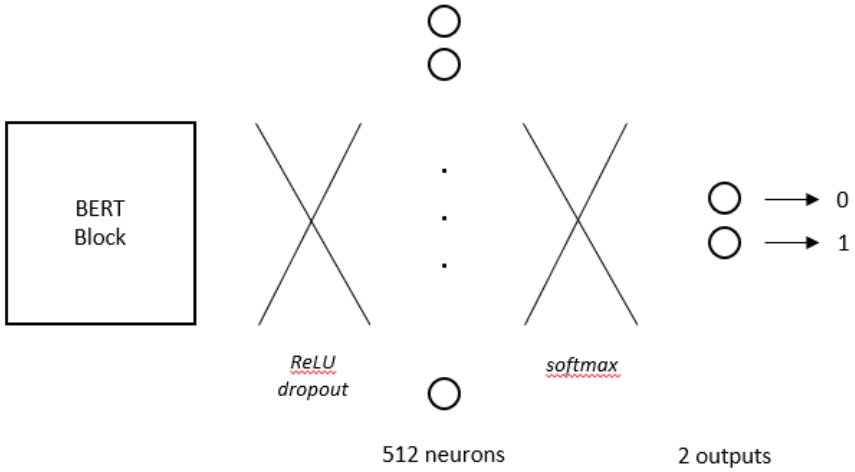

Figure 2. BERT Classification layer architecture

2. DATA

Two datasets are analyzed in this project, both are taken from Kaggle:

- News dataset, containing 17903 fake and 20826 real news articles (https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset)
- Comments dataset from Jigsaw, containing 144277 non-toxic and 15294 toxic comments (https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data). The full version of dataset contains other categories which are not used in this project: severe toxic, obscene, threat, insult, and identity hate.

3. METHODOLOGY

3.1. News dataset

3.1.1 Preprocessing

Preprocessing of the text data is summarized in steps listed in order as follows:

1. Expansion of contractions using “contractions” package (https://pypi.org/project/contractions/)
2. Splitting of each document into sentences using NLTK’s Punkt Sentence Tokenizer
3. Conversion to lower case
4. Tokenization using NLTK’s word_tokenize
5. Lemmatization according to each word’s Part Of Speech (POS) tag, which consists of 4 categories: adjective, adverb, verb and noun.

Additionally, removal of the source of the article needs to be done as real news articles in the dataset starts with it, for example “Washington (Reuters) - ”. This is done with regular expression to remove everything up until the first ‘-‘ (dash) encountered.

3.1.2. Non-deep learning classification

The News dataset was split 80:20 into train and test set. BoW and TF-IDF vectorization were performed using Scikit-Learn with default parameters (i.e. unigram and consisting of the entire vocabulary found in the corpus). For each vectorization technique, logistic regression, linear SVM and SVM with RBF kernel classification models were built also using default parameters specified in Scikit-Learn. No elaborate optimization of hyperparameters was done, in favor of attempting a more challenging dataset as well as exploring BERT, which is considered as the current best standard for text classification.

3.2. Comments dataset

3.2.1. Preprocessing

Preprocessing of text data was done using the same procedure as News dataset in section 3.1.

3.2.2. Non-deep learning classification

Similar to News dataset Bow and TF-ID vectorization were performed in combination with logistic regression, linear SVM and SVM with RBF. Comments dataset contains only ~9.4% toxic comments therefore comparison between uniform class weight and 'balanced' class weight is drawn. Due to time constraint, only the best model had the hyperparameter(s) optimized.

The dataset is split 80:20 into train and test set, after which the train set is again split 80:20 to perform validation. Random state for each splitting is fixed to ensure comparability between models. The performance on validation set, particularly precision, recall, f1-score and ROC-AUC score, are used to select the best non-deep learning classification model.

3.2.3. BERT

BERT comes with its own tokenizer which can process raw text, therefore it was used instead of the preprocessing steps described for the non-deep learning classification. Huggingface's "transformers" package was used, specifically "bert-base-uncased" which was pre-trained on lower-case English corpus. Three models were trained with parameters listed below:

Parameters	**BERT v1**	**BERT v2**	**BERT v3**
Loss function	Cross entropy	Cross entropy	Cross entropy
Class weight	Balanced	Balanced	Balanced
Optimizer	AdamW	AdamW	AdamW
Number of tokens	128	256	128
Learning rate	10^{-5}	10^{-5}	10^{-7}
Batch size	32	32	32
Number of epochs	10	10	10
Classification layer	(768, 512)	(768, 512)	(768, 512)
BERT parameters frozen	Yes	Yes	No

Table 1. BERT model parameters

Train, validation, and test split were done exactly as described for non-deep learning classification in the previous section. Within each training, model with lowest validation loss is saved to prevent overtraining. Validation losses were compared to select the best BERT model.

4. EVALUATION AND RESULTS

4.1. News dataset

Preprocessing step is illustrated below:

Before:

WASHINGTON (Reuters) - The head of a conservative Republican faction in the U.S. Congress, who voted this month for a huge expansion of the national debt to pay for tax cuts, called himself a "fiscal conservative" on Sunday and urged budget restraint in 2018. In keeping with…

After:

the head of a conservative republican faction in the you.s . congress , who vote this month for a huge expansion of the national debt to pay for tax cut , call himself a " fiscal conservative " on sunday and urge budget restraint in 2018 . in keep with…

Lower casing and lemmatization seems to work properly in general apart from the fact that "U.S" is transformed into "you.s". Unfortunately, there does not seem to be a simple workaround, therefore, it is kept as it is for further analysis.

Below is the summary of quick exploration of BoW and TF-IDF vectorizer combined with logistic regression, linear SVM and SVM with RBF kernel. Normal threshold of 0.5 probability is used for all models.

Model	Accuracy	Precision	Recall	F1-score
Logistic regression, BoW	0.985	0.984	0.982	0.983
Logistic regression, TF-IDF	0.982	0.985	0.974	0.980
Linear SVM, BoW	0.982	0.981	0.990	0.980
Linear SVM, TF-IDF	0.989	0.990	0.985	0.988
RBF-SVM, BoW	0.979	0.983	0.971	0.977
RBF-SVM, TF-IDF	0.987	0.990	0.982	0.986

Table 2. Test results of linear models with default parameters on News dataset

There is not enough differentiation between vectorization methods and classification algorithms since all of them appear to have no problems distinguishing between fake and real news with high accuracy. This warrants a closer look at the text data from both classes.

Sample real news:

…japan be prepare to acquire precision air-launched missile that for the first time would give it the capability to strike north korean missile site , two source with direct knowledge of the matter say . japan plan to put money…

Sample fake news:

… decide to literally beg his many fan on twitter for money to fund his campaign . invoke his crooked hillary theme , he tweet out to all the neo-nazis , klan member , misogynist, …

It seems that there might be a difference in writing style and topics between real and fake news. This correlation is generally not straight-forward to visualize, however, some indication can be observed from a comparison of TF-IDF feature means between real and fake news.

Real

term	mean
say	0.065913
trump	0.044357
state	0.030544
president	0.026121
republican	0.023803
house	0.021259
government	0.019999
year	0.017630
united	0.017390
party	0.017002

Fake

term	mean
trump	0.059594
image	0.026407
say	0.024158
people	0.020457
clinton	0.020199
just	0.018506
like	0.017681
president	0.017344
obama	0.017086
make	0.016630

Table 3. 10 terms highest mean TF-IDF score in real news and fake news

It can be seen that there is a difference in the nature of words appearing in real news and fake news. For instance, the word “clinton” and “obama” which correspond to political opponents of Trump have higher TF-IDF frequency in fake news. The word “trump” itself appears more frequently in fake news. The author recognizes the possible bias existing in the acquisition of real and fake news articles and does not attempt to make any political statement or conclusion.

4.2. Comments dataset

In order to obtain better comparison between algorithms, a harder classification problem is required. Therefore a second dataset is included in this project.

Comments dataset is tougher to classify for two reasons:

1. There is a severe imbalance in the dataset. Only ~9.4% of comments are considered toxic.

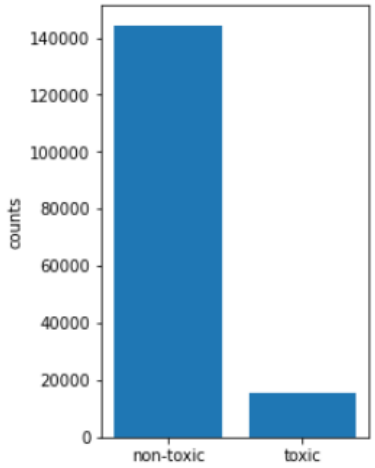

Figure 3. Distribution of toxic and non-toxic comments

2. Similar conversational language style in both toxic and non-toxic comments.

4.2.1. Non-deep learning classification

Baseline performance was established using BoW and TF-IDF with logistic regression. Linear SVM and RBF-SVM. BoW and TF-IDF vectors are generated with all words found in the corpus after stop-words removal, totalling 179429 features. The following are the validation results:

Model	Accuracy	Precision	Recall	F1-score
Logistic regression, BoW	0.958	0.851	0.680	0.756
Logistic regression, TF-IDF	0.954	0.923	0.571	0.706
Linear SVM, BoW	0.954	0.769	0.740	0.754
Linear SVM, TF-IDF	0.959	0.872	0.670	0.758
RBF-SVM, BoW	0.935	0.968	0.336	0.499
RBF-SVM, TF-IDF	0.957	0.923	0.601	0.728

Table 4. Validation results of baseline linear models

The models prioritize accuracy and tend to predict non-toxic more as evident from much lower recall compared to precision score. It might be desired to sacrifice precision for higher recall, particularly in the context of comments gatekeeping, especially if there is a manual moderation somewhere in the filtering pipeline. One way to do this is to introduce 'balanced' class weight where the weight of a particular class is set to be inversely proportional to its frequency in the dataset. The balanced class weights are 0.553 for non-toxic comments and 5.217 for toxic comments.

Model (balanced)	Accuracy	Precision	Recall	F1-score
Logistic regression, BoW	0.945	0.677	0.822	0.742
Logistic regression, TF-IDF	0.943	0.662	0.838	0.739
Linear SVM, BoW	0.942	0.670	0.782	0.722
Linear SVM, TF-IDF	0.947	0.696	0.798	0.744
RBF-SVM, BoW	0.938	0.639	0.805	0.713
RBF-SVM, TF-IDF	0.959	0.839	0.714	0.771

Table 5. Validation results of models with balanced class weight

The obvious trend is balanced weight produces higher recall and lower precision. Interestingly, RBF-SVM sees a major F1-score improvement from using TF-IDF and balanced class weight. For further analysis, balanced class weight is focused on.

Analysis of the unigram BoW and TF-IDF terms shows a distinction between non-toxic and toxic comments as shown below. The author apologizes for inappropriate terms displayed.

BoW

Non-toxic		Toxic	
term	mean	term	mean
article	0.500405	fuck	0.800902
page	0.377732	suck	0.313587
wikipedia	0.310160	wikipedia	0.240486
talk	0.263549	shit	0.238917
use	0.219779	like	0.238786
make	0.195755	nigger	0.223944
edit	0.186509	hate	0.172682
just	0.175690	page	0.168955
like	0.173403	know	0.162286
say	0.163817	faggot	0.162286

TF-IDF

Non-toxic		Toxic	
term	mean	term	mean
article	0.027063	fuck	0.064007
page	0.024510	shit	0.021041
talk	0.022387	suck	0.018965
wikipedia	0.018847	like	0.016618
edit	0.015978	bitch	0.016268
just	0.013288	stupid	0.015156
use	0.013194	wikipedia	0.014304
make	0.013064	just	0.013693
thanks	0.012260	stop	0.013346
think	0.012170	asshole	0.013079

Table 6. 10 terms highest mean BoW and TF-IDF score in real news and fake news

Another parameter to look at is the number of features generated by the vectorizer. CountVectorizer and TfidfVectorizer support selection of n features with the highest term frequency across the corpus.

Model (balanced)	Accuracy	Precision	Recall	F1-score
Logistic regression, n=100000	0.942	0.651	0.843	0.735
Logistic regression, n=50000	0.941	0.645	0.847	0.733
Logistic regression, n=25000	0.939	0.635	0.854	0.728
Logistic regression, n=10000	0.934	0.610	0.859	0.713
Logistic regression, n=5000	0.928	0.583	0.865	0.697

Model (balanced)	Accuracy	Precision	Recall	F1-score
Linear SVM, n=100000	0.944	0.672	0.809	0.735
Linear SVM, n=50000	0.941	0.653	0.814	0.725
Linear SVM, n=25000	0.938	0.634	0.833	0.720
Linear SVM, n=10000	0.930	0.595	0.845	0.698
Linear SVM, n=5000	0.923	0.565	0.852	0.679

Model (balanced)	Accuracy	Precision	Recall	F1-score
RBF-SVM, n=100000	0.959	0.829	0.723	0.773
RBF-SVM, n=50000	0.959	0.821	0.737	0.777
RBF-SVM, n=25000	0.959	0.806	0.749	0.776
RBF-SVM, n=10000	0.956	0.779	0.762	0.770
RBF-SVM, n=5000	0.953	0.747	0.767	0.757

Table 7. Validation results with different number of features generated by TfidfVectorizer

Recall seems to increase for all algorithms with decreasing number of features, however since the evaluation is not done with cross-validation this trend might be specific for the current training and validation split. RBF-SVM seems to experience lower decay in F1-score as number of features decreases compared to the other two algorithms which is in line with the intuition that RBF-SVM typically suffers from overfitting problem and lowering variance while increasing bias in the model typically affects the model in a positive way.

Since RBF-SVM takes > 50000 iterations to converge, regularization parameters optimization is only done for logistic regression and Linear SVM.

Regularization parameter optimization

Logistic regression and Linear SVM were optimized with 5-fold cross validation.

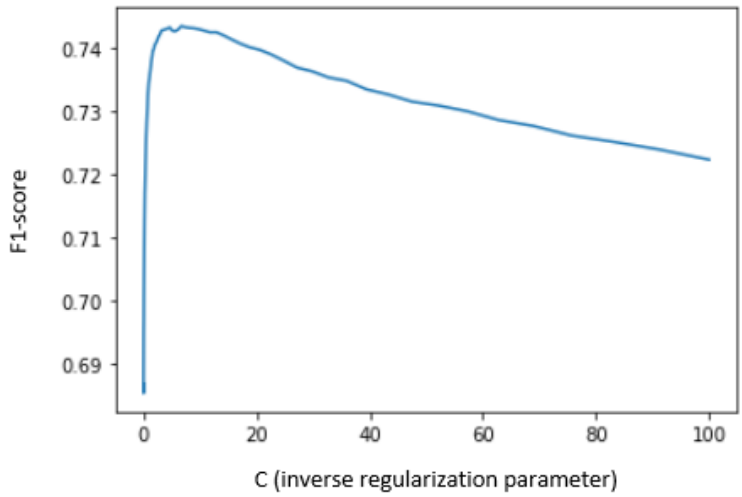

Figure 4. F-1 score against C for logistic regression

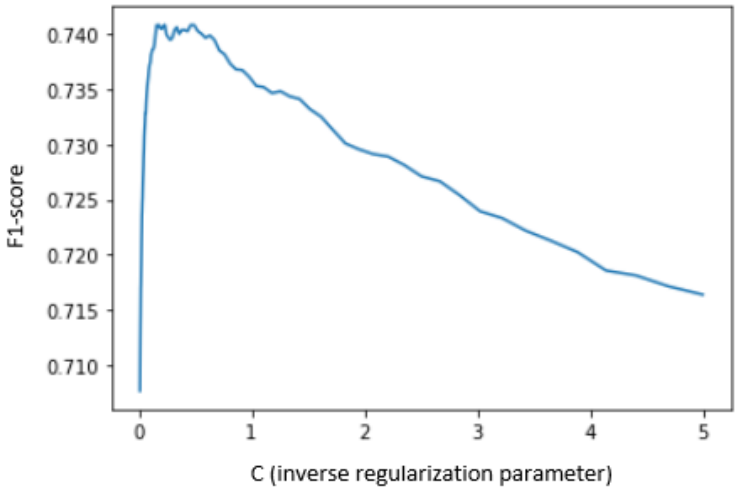

Figure 5. F-1 score against C for linear SVM

Optimum C of 6.734 and 0.231 were obtained for logistic regression and linear SVM, respectively. The results of the optimized algorithms on validation set (for comparability with the other models so far) are summarized in the following table.

Model (balanced)	**Accuracy**	**Precision**	**Recall**	**F1-score**
Logistic regression, optimized	0.944	0.672	0.809	0.735
Linear SVM, optimized	0.948	0.699	0.810	0.750

Table 8. Validation results of optimized logistic regression and linear SVM

Even though the validation F1-scores of optimized algorithms are lower than those of many other algorithms so far, they might generalize better to the test set since they are obtained with cross validation as opposed to just one version of validation set. However, for illustration purpose RBF-SVM with n=50000, which corresponds to the highest F1-score obtained on validation set, is picked as the best model to be compared with BERT (in section 4.2.3).

4.2.2. BERT

Three BERT models were trained according to the parameters listed in the methodology section. Token lengths of 128 and 256 were chosen as they seem to be appropriate considering the word count distribution in the training set. The training of each model took >14 hours.

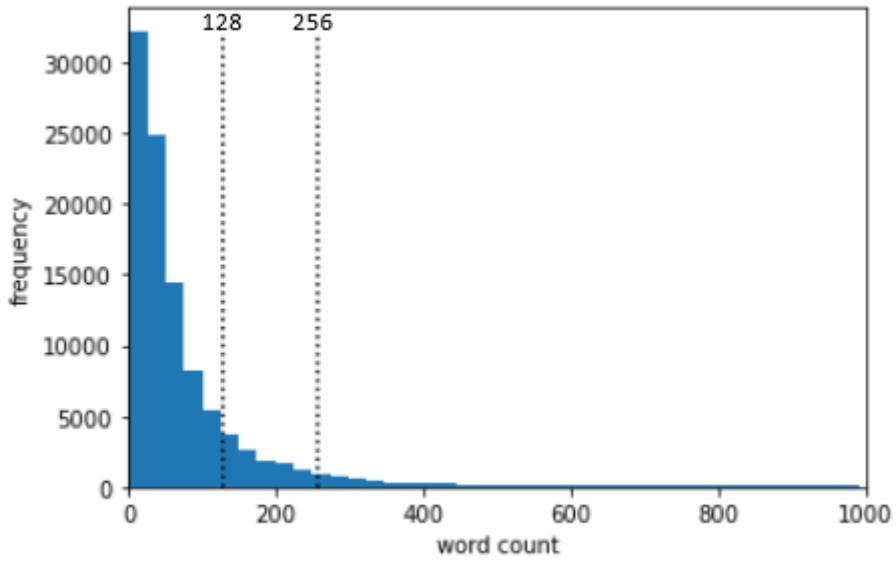

Figure 4. Word count distribution in the Comments training set

Validation losses of the three models are plotted.

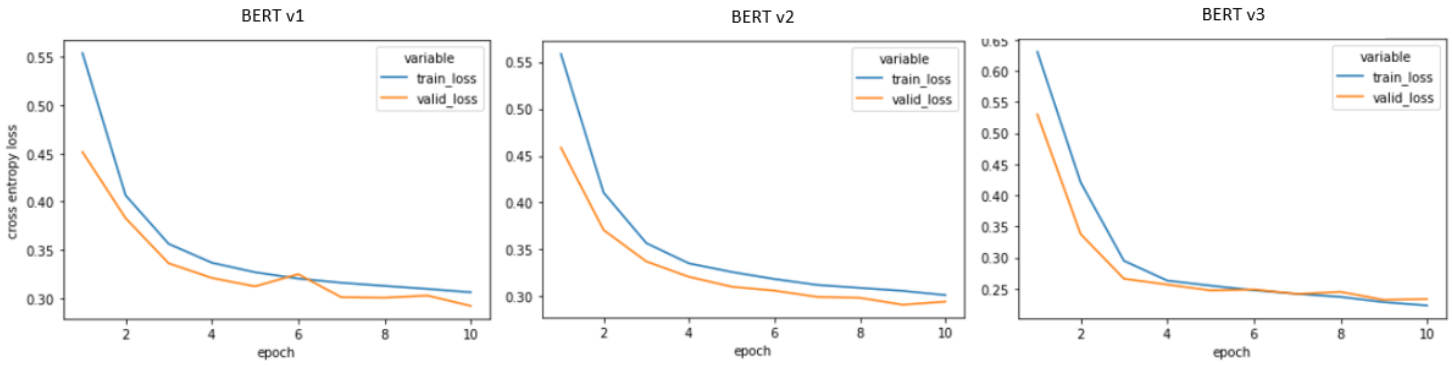

Figure 5. Training and validation losses of three BERT models

Increasing the number of tokens does not seem to have any significant impact on validation loss. This is possibly due to most comments having fewer than 128 words and hence 256-token features consist of mostly padding tokens which do not add any information to the model.

Unfreezing BERT parameters, however, significantly improves validation loss at the cost of longer training time and the risk of overfitting. Learning rate had to be reduced by a factor of 100 otherwise validation loss shoots back up immediately after the first epoch. This is because backpropagation updates all the weights that are not frozen.

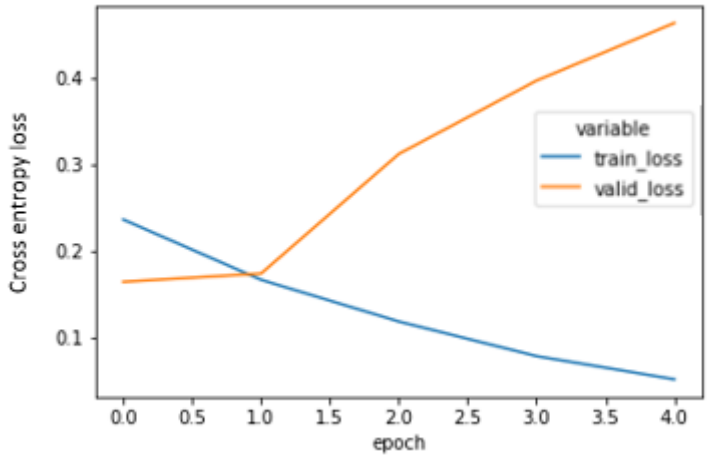

Figure 6. Preliminary BERT trial with 200 tokens and learning rate of 10^{-5}

Validation results are summarized in the following table:

Model	Accuracy	Precision	Recall	F1-score
BERT v1	0.884	0.441	0.805	0.570
BERT v2	0.886	0.449	0.850	0.587
BERT v3	0.957	0.738	0.856	0.793

Table 9. Validation results of BERT models

BERT v3 with parameters (weights) unfrozen and jointly trained together with classification layer produces much better validation results.

4.2.3. Threshold optimization

Threshold can be adjusted to yield different classification result. Up until this point default threshold of 0.5 was used. In this section two additional thresholds are explored for the best model, BERT v3:

- Threshold to produce 2447 comments predicted as toxic (2447 is the actual number of toxic comments in validation set)
- Threshold to produce best validation F1-score

To produce 2247 toxic predictions, threshold needs to be shifted to 0.845 as shown in the following plot.

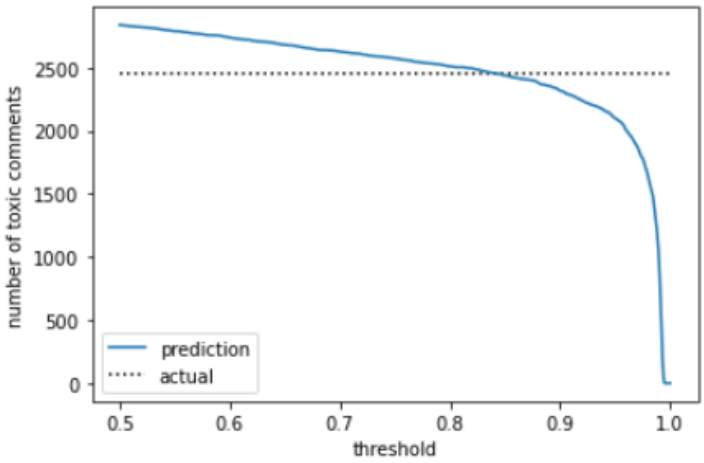

Figure 7. Number of toxic comments (predicted and actual) with different probability thresholds

F1-score can also be optimized for the validation set by plotting it against different thresholds.

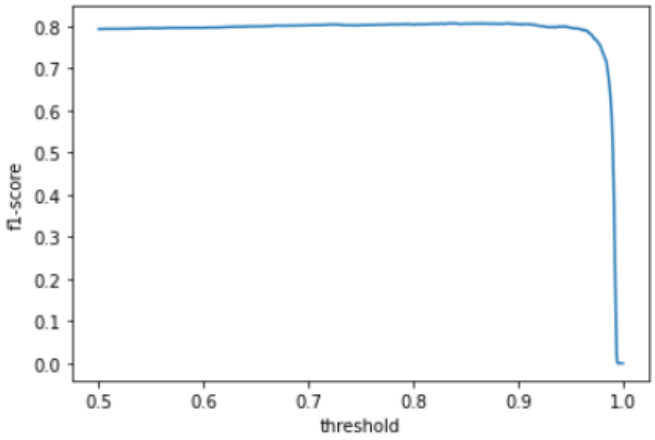

Figure 8. F1-score with different probability thresholds

Threshold of 0.891 produced maximum F1-score. BERT v3 validation results with different results are shown below.

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
BERT v3, default threshold	0.957	0.738	0.856	0.793
BERT v3, threshold=0.845	0.963	0.805	0.805	0.805
BERT v3, threshold=0.891	0.964	0.822	0.791	0.806

Table 9. BERT v3 validation results with different thresholds

4.2.4. Test results

Performance metrics of BERT v3 with different thresholds on test set are summarized in the following table.

Model	Accuracy	Precision	Recall	F1-score
BERT v3, default threshold	0.955	0.732	0.836	0.780
BERT v3, threshold=0.845	0.961	0.802	0.784	0.793
BERT v3, threshold=0.891	0.961	0.817	0.765	0.790

Table 10. BERT v3 test results with different thresholds

5. CONCLUSION

BoW or TF-IDF vectorization combined with linear models such as logistic regression, linear SVM and RBF-SVM is sufficient to classify News dataset with test F1-score >0.98.

For Comments dataset, BERT model with unfrozen parameters outperforms logistic regression, linear SVM and RBF-SVMwith F1-score of ~0.8 on validation set and ~0.79 on test set.

There are several other methods that might improve the model's accuracy. The use of "large" BERT instead of the "base" model used in this project sometimes yield better performance at the expense of longer training time (BERT-large has 24 encoding layers with 336M parameters, while BERT-base has 12 encoding layers with 110M parameters). It has also been reported that introducing additional layers (called adapter) after each transformer block can improve BERT's predictive power[5].

6. REFERENCES

1. Harris, Zellig (1954). "Distributional Structure". *Word*. 10 (2/3): 146–62.
2. Spärck Jones, K. (1972). "A Statistical Interpretation of Term Specificity and Its Application in Retrieval". *Journal of Documentation*. 28: 11–21.
3. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. *Attention Is All You Need.* https://arxiv.org/abs/1706.03762
4. BERT English architecture. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/english-bert
5. Sina J. Semnani, Kaushik Ram Sadagopan, Fatma Tlili. BERT-A: Fine-tuning BERT with Adapters and Data Augmentation. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15848417.pdf