CS 263 Final Project Report
Sarcasm Detection

Abstract

In this project we explore 4 models to classify sarcasm in texts from the SARC dataset[2]. We use a mixture of classic and modern models to arrive at accuracies up to 73.1%.

1 Introduction

Sarcasm detection is a sub-problem of the sentiment analysis task. A statement is sarcastic when one states one thing, but usually means the exact opposite[3]. For example one might state "I love pies", when one actually means "I hate pies". This is quite confusing and people usually rely on contextual cues such as tone, logical absurdity, the author's personality and more disambiguate sarcasm. In NLP, this is very difficult to detect with a limited context and poses an interesting challenge which we explore in this project.

2 Data

The data, sourced from Princeton's Self-Annotated Reddit Corpus[10] (SARC v0.0) dataset[2], was generated by collecting comments from Reddit containing the "\s" symbol with which a user indicates sarcasm. The corpus contains 1.3 million rows each of which has the following fields:

Name	**Type**	**Description**
`comment`	string	User comment ranging from 1 to 10000 characters, labelled as either sarcastic or not
`parent_comment`	string	Parent comment that `comment` is in response to. Ranges from 1 to 40301 characters and is unlabelled
`label`	int	Target label which is 0 for non-sarcastic and 1 for sarcastic
`author`	string	comment author user name
`subreddit`	string	Forum that the comment is made in
`ups`	int	Positive votes from other users on comment
`downs`	int	Negative votes from other users on comment
`score`	int	Function sum of ups and downs
`date`	string	comment date
`created_utc`	int	comment timestamp

These rows contain 50% non-sarcastic and 50% sarcastic comments (see examples of each in Appendix B). Ultimately, mostly the `comment`, `parent_comment` and `label` columns were used so the data was pared down to these attributes. The data was also vectorized using the Term Frequency Inverse Document Frequency (TF-IDF) vectorizer[5] so that each word's frequency in all entries would be taken into account for their individual weights.

3 Methods

3.1 BERT

The Bidirectional encoder Representations from Transformers (BERT)[6] is a neural network that bases off of the encoder in the transformer architecture[14]. BERT is an extremely effective transformer that is leveraged by Google's search engine in almost every query. BERT is very effective because it trains the model by looking at all words at the same time as opposed to directional models that read each word sequentially. This allows for a much deeper contextual understanding. It is trained on two primary tasks: Masked Language Modeling which replaces 15% of the words in each sentence with a token then has the model attempt to predict the original word, and Next Sentence Prediction which inputs sentence pairs where 50% of the sentences are sequential and 50% are randomly paired and has the model attempt to predict if the two sentences are sequential[9].

For this project, we used the following existing implementations of these models:

- *bert-base*[1] - BERT model trained using wikipedia articles[7] and novels[17].

Training on large corpuses drastically increases accuracy. The issue is it becomes extremely resource intensive very quickly due to the training structure and the large number of weights to update. After pre-training the model the same architecture can be used for fine-tuning with minor modifications for use in more specific tasks like question answering[6] (Appendix C2 shows the high level architectureC.2).

3.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a family of neural networks that are used predominantly in vision-related tasks. This is due to the convolution and pooling operations being effective at capturing spatial relationships between features of an image. They have less of a track record in NLP[11], one of the reasons being that they need to pad their inputs to be of a fixed length, which makes them less effective for texts of high variance in length.

In the context of this problem we pre-processed the input by concatenating `comment` and `parent_comment` with a `<SEP>` token and then input the string into a frozen BERT model[1] to generate embeddings. We padded the embeddings to a uniform length of 512 and then input them into a CNN, applying 2D convolutions and pooling layers alternatingly. Lastly, we then applied 2 dense layers to obtain a classification. This initial model had 11 million trainable parameters and yielded a validation accuracy of 63.3% after 3 epochs of training with 10% of the data.

We then investigated methods to improve this baseline accuracy. Some simple ones were basic hyper-parameter tuning, which increased performance by 2-3% and using all of the training data, which improved it by a further 4%. We then investigated some other ideas such as unfreezing the BERT encoder, using different encoders[13] (including a sentiment-tuned RoBERTa model[8]), but generally did not see improvements. Lastly, we investigated the other fields of the dataset to determine if any of them could provide a better context (details in Appendix D). From these we chose the `author` and `subreddit` fields and input them into the dense layers together with the convolution outputs. This improved accuracy by a further 2%. The final architecture was 15 million parameters large and quite costly to train, taking around 2 days of GPU time (see Appendix C.1 for a visualisation).

3.3 Random Forest Algorithm

Random Forest is a flexible and easy-to-use supervised learning algorithm. It can be used for both classification and regression. A forest consists of trees and the more trees there is, the more robust the forest is. Random forest creates decision trees by randomly selecting data samples, predicts from each tress, and chooses the best solution by means of voting.[4] It works in four separate steps: first, selecting random sample from a dataset; second, creating a decision tress for each sample and predicting the result from each decision tree; third, performing a vote for each result; last, choosing the prediction result with most votes as the final result.

We choose random forest as a method due to its flexibility in use and dependable performance. However, since it is slow to generate prediction and hard to interpret, it likely will not be the most practical method.

In the fine-tuning phase of building random forest, we tested different max features for the best split, and different criteria to measure the quality of the split. Due to long training time, the training data size was reduced to 40% of final data size. Final evaluation (with more data) used the hyperparameters of the best result-as in bold.

max features	**criterion**	**avg accuracy**
sqrt	gini	**0.57025**
sqrt	entropy	0.56695
log2	gini	0.56785

Table 1: Accuracy for Random Forest Tuning

3.4 Support Vector Machine (SVM) Classification

A Support Vector Machine Classifier, or SVM, is an algorithm commonly used in Natural Language Processing to differentiate between two categories. This differentiation is defined by a hyperplane, also known as a kernel or decision boundary, which acts as a threshold for the decision-making process. The simplest of these kernels has a single dimension while more complicated ones can theoretically range up to an infinite number of dimensions. These kernels can further be divided into three types based on their shape: Linear, Polynomial, and Radial Basis Function.[12]

The kernel's final shape is determined by three optimization goals:

1. Maximize the Distance Margin (the distance from the decision boundary to the categorized data)
2. Maximize the correct classification of points in the training set
3. Optimize the distance of influence for any single training point

These optimization goals are controlled by the hyperparameters C and γ. C controls Goals 1 and 2 while γ controls Goal 3.

A small C optimizes in favor of the first goal and therefore does not prioritize minimizing misclassification. This results in a large number of mistakes in classification. Conversely, a large C optimizes in favor of the second goal and therefore results in decision boundaries with smaller margins.

The second hyperparameter, γ, is used only with non-linear kernels. In the instance where γ is used with a Radial Basis Function Kernel, it controls the distance of influence for any single training point.

In practice, a low γ value indicates a high similarity radius which results in larger groups of data points. On the other end, a high γ value causes overfitting since only points that are very close together are classified in the same class.

In regards to the relationship between the two hyperparameters, if one is too large, the effect of the other one on the decision boundary becomes negligible. Additionally, in the case of a linear kernel, only C can be used to optimize the decision boundary.[16] For this project, the optimal hyperparameters of 10 and 0.1 for C and γ respectively were found by iterating through all combinations of C and γ with values from 0.001 to 100 in powers of ten and comparing the maximum accuracy scores.

Finally, ThunderSVM was used to take the current SVM model and elevate it to a GPU-accelerated model. [15]

4 Results

Each of the models had varying challenges that resulted in the necessity for using different sizes of data to evaluate their individual effectiveness. BERT, CNN, Random Forest, and SVM models used approximately 5K, 1.3M, 50K, 1.3M data points respectively. The models were evaluated using

5-fold validation with each run documented in the tables below. Each iteration keeps track of four values: Accuracy, Precision, Recall, and F1 Score. In our final code, we also printed out training time and predicting time for efficiency measurement. The mean, maximum, and minimum data points are in bold for easy reference.

Model	***Run 1***	***Run 2***	***Run 3***	***Run 4***	***Run 5***	***Mean***
BERT	0.490	**0.516**	0.514	**0.488**	0.503	**0.502**
CNN	**0.731**	**0.718**	0.721	0.722	0.719	**0.722**
Random Forest	0.528	**0.533**	**0.525**	0.530	0.528	**0.529**
SVM	**0.480**	0.506	0.499	0.504	**0.527**	**0.503**

Table 2: Accuracy for variable data points

Model	***Run 1***	***Run 2***	***Run 3***	***Run 4***	***Run 5***	***Mean***
BERT	0.490	**0.484**	**0.514**	0.512	0.503	**0.501**
CNN	0.721	0.709	**0.734**	0.710	**0.708**	**0.716**
Random Forest	0.328	**0.330**	**0.308**	0.316	0.320	**0.320**
SVM	**0.485**	0.515	0.498	0.510	**0.527**	**0.507**

Table 3: Precision for variable data points

Model	***Run 1***	***Run 2***	***Run 3***	***Run 4***	***Run 5***	***Mean***
BERT	1.000	1.000	1.000	1.000	1.000	1.000F
CNN	0.740	0.721	**0.742**	**0.719**	0.729	**0.737**
Random Forest	**0.136**	0.126	0.119	**0.119**	0.125	**0.125**
SVM	**0.641**	0.2098	0.208	**0.1999**	0.525	**0.357**

Table 4: Recall for variable data points

Model	***Run 1***	***Run 2***	***Run 3***	***Run 4***	***Run 5***	***Mean***
BERT	0.658	**0.653**	**0.679**	0.677	0.669	**0.667**
CNN	0.730	0.715	**0.737**	**0.714**	0.718	**0.723**
Random Forest	**0.192**	0.182	**0.172**	0.173	0.180	**0.180**
SVM	**0.552**	0.298	0.293	**0.287**	0.526	**0.391**

Table 5: F1 Score for variable data points

5 Evaluation

From our experiments we can see that neural network-based approaches appear to be the most effective at classifying sarcasm in this dataset. There is however, still room for improvement. The accuracy of all models could likely be improved by exploring more pre-processing and removing outliers/non-sense datapoints. For example, simply by removing the 0.1% longest comments, the maximum length shrinks from 10000 characters to 363. Doing so would benefit particularly expensive models like BERT where the biggest limiting factor in this project was resources. This also benefits training speed as models like the CNN can make better tradeoffs between padding and trimming to the maximum sequence length.

References

[1] Bert-base-cased · hugging face. `https://huggingface.co/bert-base-cased`.

[2] Sarc 0.0 dataset. `https://nlp.cs.princeton.edu/SARC/0.0/`.

[3] Sarcasm.

[4] Sklearn random forest classifiers, 2019.

[5] M. Chaudhary. Tf-idf vectorizer scikit-learn, Jan 2021.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[7] W. Foundation. Wikimedia downloads.

[8] J. Hartmann, M. Heitmann, C. Siebert, and C. Schamp. More than a feeling: Accuracy and application of sentiment analysis. *International Journal of Research in Marketing (Forthcoming)*, 2022.

[9] R. Horev. Bert explained: State of the art language model for nlp.

[10] M. Khodak, N. Saunshi, and K. Vodrahalli. A large self-annotated corpus for sarcasm. *CoRR*, abs/1704.05579, 2017.

[11] Y. Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[12] A. Kumar. Svm (support vector machine) for classification, Jul 2020.

[13] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[15] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen. ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research*, 19:797–801, 2018.

[16] S. Yıldırım. Svm hyperparameters explained with visualizations, Oct 2020.

[17] 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.

6 Appendix

A Project Code

The project code is available for download via the following URL: `https://github.com/NiklasZ/cs263-final-project`

B Data Examples

Below is an example of a non-sarcastic data sample and a sarcastic one. It should be noted that the SARC dataset was not sanitised so examples may contain some objectionable content.

Field	**Value**
`comment`	Mine auto renewed without asking me the other day.
`parent_comment`	Thank you, Jagex, for being cool Jagex has always been a cool company, and we could thank them for tons of things, but I just noticed that they don't charge your card as a recurring subscription. So many companies today do scummy things where they try to get you to sign up for something so they can bill you monthly and hope you'll forget about it, but Jagex only runs your subscription for as long as you purchased. I know it's nothing huge, but it's something I appreciate.
`label`	0
`author`	yeaweckin
`subreddit`	2007scape
`ups`	-1
`downs`	-1
`score`	3
`date`	2016-11
`created_utc`	1477964417

Table 6: Non-sarcastic example

Field	**Value**
`comment`	Exactly, no reason whatsoever.
`parent_comment`	To make predators think they're poisonous?
`label`	1
`author`	Bifi323
`subreddit`	oddlysatisfying
`ups`	-1
`downs`	-1
`score`	1
`date`	2016-11
`created_utc`	1477994394

Table 7: Sarcastic Example

C Model Architectures

C.1 CNN

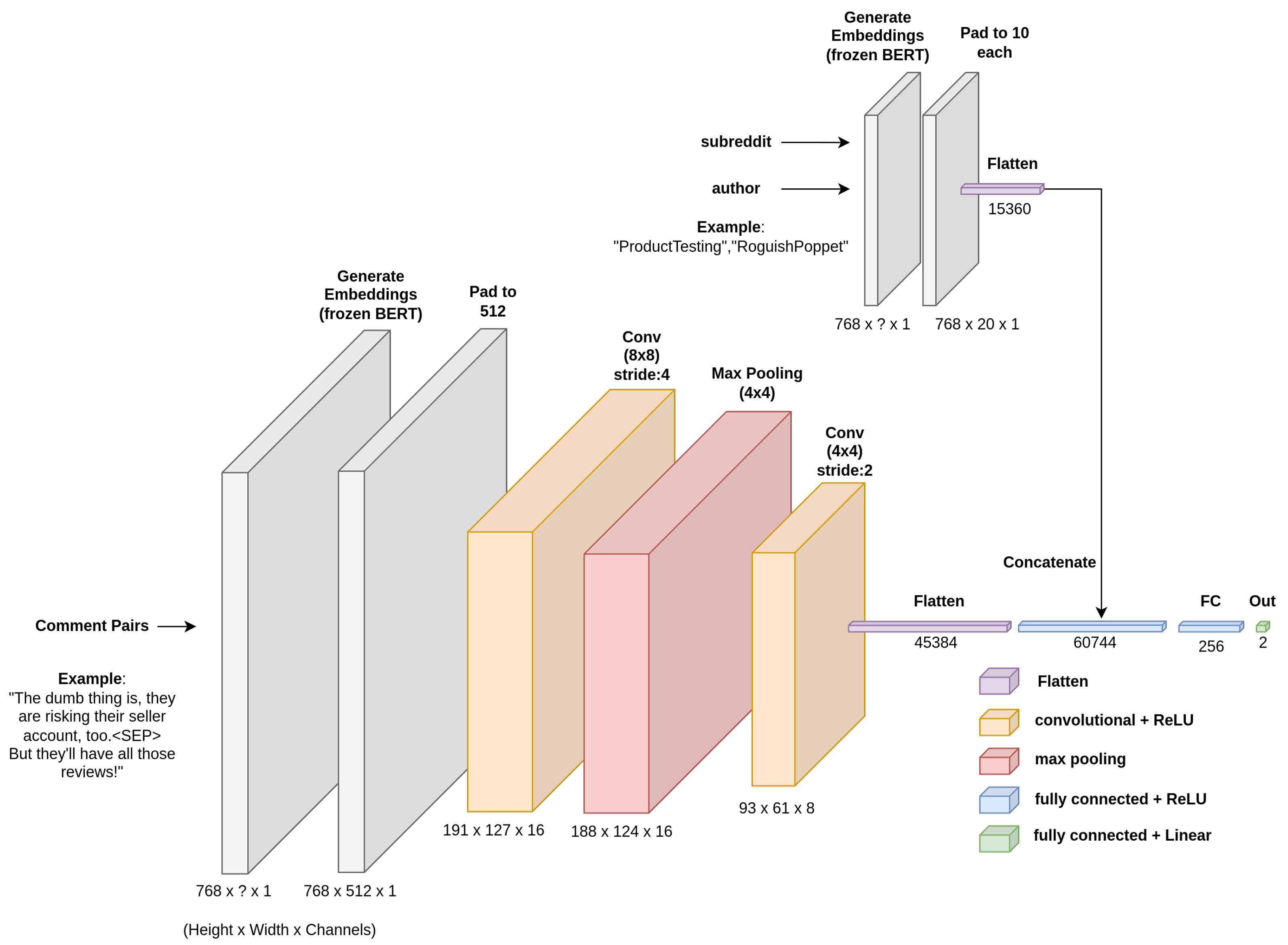


Figure 1: CNN Architecture. We put the comment pairs through the convolution layers and then concatenate the output with the extra fields. These are then passed on to the fully-connected layers.

C.2 BERT

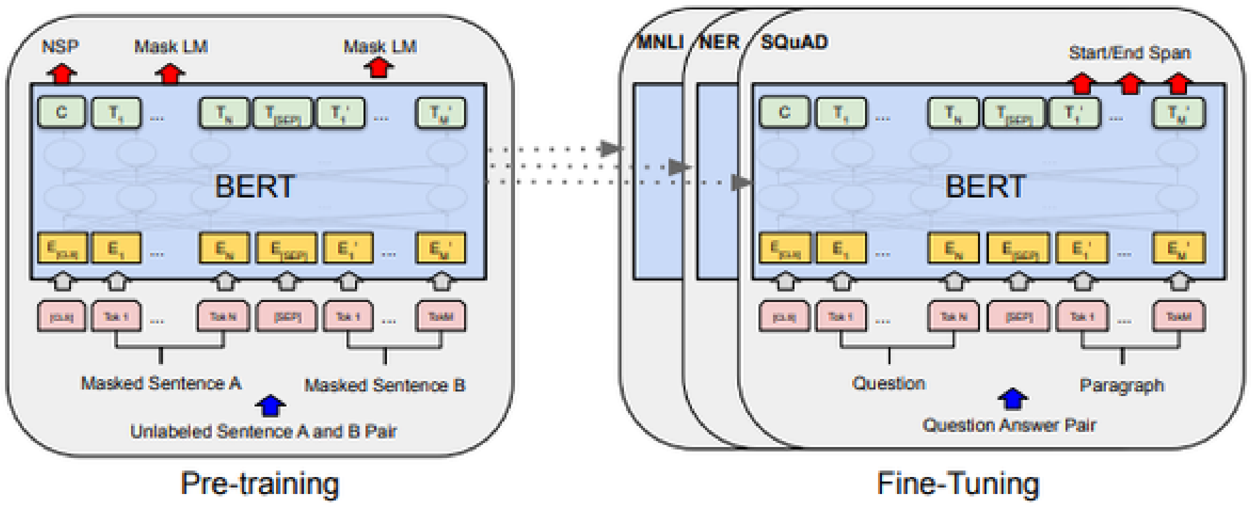
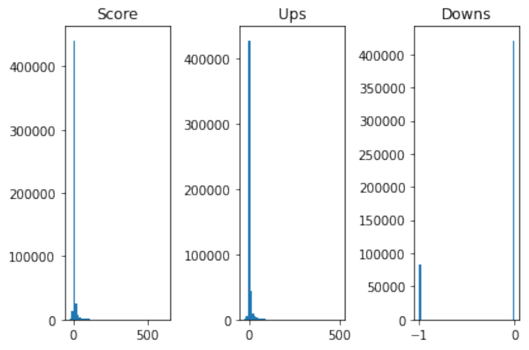


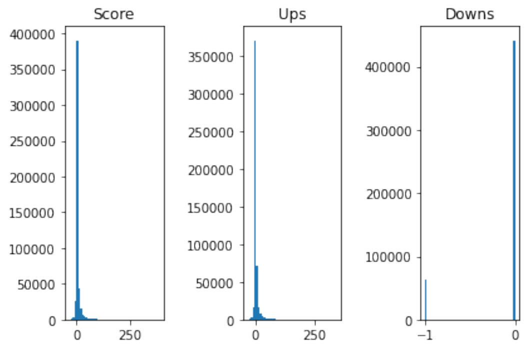
Figure 2: BERT Architecture. Overview of pre-training and fine-tuning tasks [6]

D Investigation of other Data Fields

We considered the `author`, `subreddit`, `ups`, `downs` and `score` fields as possible candidates to incorporate as inputs to model prediction. We first investigated the ups, down and score of `comment` and found that they are very similarly distributed across classes (see Figure 3). This does not make them suitable discriminators so we did not consider them further. The `subreddit` field was more promising and there is a differing rate of sarcasm across the forums (see Figure 4). The `author` field was a bit odd as almost all authors have exactly 50% genuine and 50% sarcastic comments. This was likely artificially done during the collection of the SARC dataset.



(a) Genuine Comment Score Distributions



(b) Sarcastic Comment Score Distributions

Figure 3: Comparison of scoring fields across classes. Generally, genuine comments are rated slightly more positively with a mean `score` of 6.1 than sarcastic comments with a mean score of 5.6.

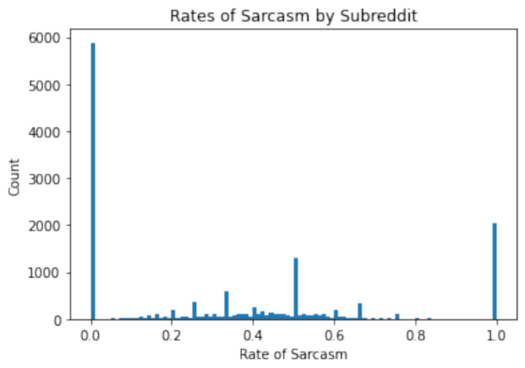


Figure 4: Sarcasm rates counted across subreddits. 0 means no sarcastic comments, whereas 1 means only sarcastic comments.

E Training Hyper-parameters

E.1 CNN

Name	**Setting**	**Notes**
Training epochs	6-10	This varies from run to run as we employ early stopping once validation accuracy degrades.
Optimizer	AdamW $\gamma = 0.0001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\epsilon = 1\mathrm{e}{-8}$ $\lambda = 0.01$	See `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html` for parameter definitions.
Max sequence length	512	Maximum number of embedding vectors the model can handle.
Batch size	10	-
Other	-	Other hyper-parameters are indicated in the model diagram.

F BERT-Metrics

While recording metrics for the BERT model it was noted that the RECALL column was outputting 1.0. The figure below shows the equation used to calculate Recall. For this recall values calculated to equal 1.0 it is suspected that the value of False Negative summation for each epoch must have been 0.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

It is not understood at this time why the False Negatives were not recorded yet the False Negatives used to calculate Precision were. This does mean that the F1 score recorded for the model is also invalid as it is dependent on Recall score as shown in the equation below.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Further investigation will have to be conducted to determine the source of the error in calculation.