Detecting Negative Comments on Social Media Platforms in Malay

Steven Limcorn Lee Jian Yang Joshua Yu Xuan Soo

June 3, 2021

Abstract

The advancement of technology in media has increased the number of social media users. This significant increase has also brought up negative influences such as posting negative comments on multiple social media platforms and this has been proven to have a bad influence on mental health. As a result, we proposed an idea of detecting negative comments on social media platforms utilizing various neural network architectures. The aim is to investigate the performance of different models on performing malaysian language sentiment analysis tasks. For baseline models, Logistic Regression and Multinomial NB were used, and for complex models, LSTM and BERT were used. Overall, we obtained the best results from Melayu BERT with a test accuracy of 86%.

We give consent for this to be used as a teaching resource.

1 Introduction

1.1 Problem

In an era where technological advancements are prominent, the number of users on social media platforms are increasing. As shown from the graph, there is a positive upward trend for the total number of users [1]. As a result, negative comments that appear on such platforms are also increasing. Consequently, this gives rise to negative impacts on public health, including but not limited to mental health issues such as suicide, depression, anxiety, etc.

Unlike the Malaysian language (Bahasa Melayu), most spoken languages, such as English (1.348 billion native speakers [2]) have a large amount of data and resources to work with and hence several deep learning models have been implemented to detect negative comments. From the figure [3], it is shown that Bahasa Melayu is estimated to have a total of 19 million native speakers which indicates a significant difference in popularity; which suggests limited data and resources that can be used for research purposes. Considering a lack of research in the NLP sector for the Malaysian language, this project further explores sentiment analysis in Malay.

The goal at the end of this project is to create a model that detects negative social media comments i.e Twitter, by utilizing a variety of models, experimenting on baseline models such as sklearn Logistic Regression (LR) and Multinomial Naïve Bayes (MNB) to advanced models such as LSTM and BERT.

1.2 Significance

As evidently shown from [4], it is stated that the general increase in social media users has a direct correlation to suicidal rates. Additionally, the report mentions that a proportion of suicidal rates are caused by Cyberbullying, which is an act of harassment towards others online by the usage of negative comments and threats.

Although the problem of Cyberbullying has existed since the existence of social media, there are no ideal solutions implemented to effectively counteract this issue. As mentioned earlier in the problem, models used to perform sentiment analysis have been created to detect positive and negative comments, but are mostly based on the English language. By extending the current model to be able to detect such comments on the Malaysian language, negative languages could effectively be removed.

2 Datasets

1. **Malaysian Corpus Dataset**

   This is a Malaysian corpus dataset from OSCAR [5]; A corpus dataset is a large collection of texts that contain authentic representations of the Malaysian language. OSCAR stands for **O**pen **S**uper-large **C**rawled **A**ggregated co**R**pus. OSCAR provides many distinct datasets for different languages, but this project only involves the Malaysian dataset. In this project, we used this dataset to pre-train our BERT models which will be further mentioned in the BERT section. This size of the dataset consists of 230,587 sentences and a total of 6,045,753 distinct words.

2. **Malaysian Twitter Sentiment Dataset**

   For fine tuning our models, the dataset we used is a pre-parsed *Malaysian Twitter Sentiment Dataset* obtained from Kaggle; the original dataset is available open source from Github [6]. The size of the dataset consists of 658,298 sentences.

3. **Custom Survey Dataset**

   For testing our models, we created a custom dataset of our own using Google Forms. We provided respondents with a few pictures and obtained a positive and negative comment for each of them. Then, we added the responses to a google sheets file, manually labelled them and downloaded it as a csv file. The dataset consists of 50 comments.

3 Baseline Models

Two types of traditional machine learning models were chosen as the baseline models and compared by their overall accuracy. We trained and validated the performance of each of these models on the *Malaysian Twitter Sentiment Dataset* [6], and further tested their accuracies on our own custom dataset.

3.1 Logistic Regression

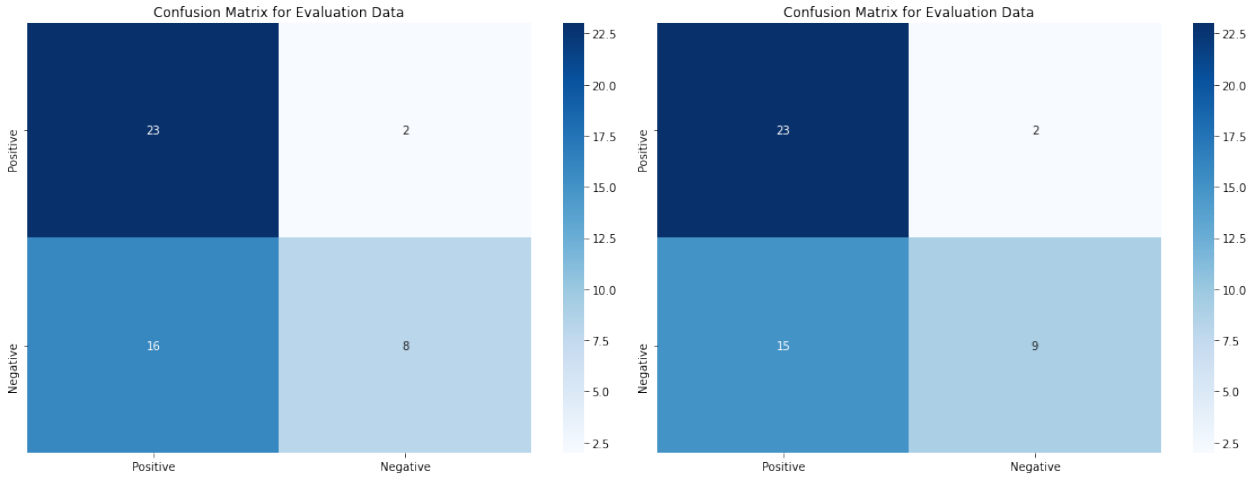
Logistic Regression is a staple in binary classification problems. Since sentiment analysis involves classifying sentences into positive or negative, it is suitable to run it as our baseline model. Two types of tokenizers were used; sklearn's CountVectorizer and TF-IDF Vectorizer. After performing hyperparameter tuning, we managed to achieve a validation accuracy of 79% by using TF-IDF Vectorizer, and a validation accuracy of 78% by using CountVectorizer.

3.2 Multinomial NB

Multinomial Naive Bayes is a probabilistic learning algorithm used for classification problems based on Bayes' rule. By using the joint probabilities of the classes; positive or negative, the probabilities of the vocabulary itself, we can calculate the probability of a class given a specific vocab. Furthermore, Bayes' rule assumes that all vocabs are independent. After performing hyperparameter tuning, we managed to achieve a validation accuracy of 76% by using TF-IDF Vectorizer, and a validation accuracy of 77% by using CountVectorizer.

3.3 Issues

By testing our Logistic Regression and Multinomial NB models on our custom dataset, we managed to achieve these results. For Logistic Regression, we obtained 63% test accuracy and an F1 score of 72% by using TF-IDF Vectorizer, and a 65% test accuracy and an F1 score of 73% by using CountVectorizer.

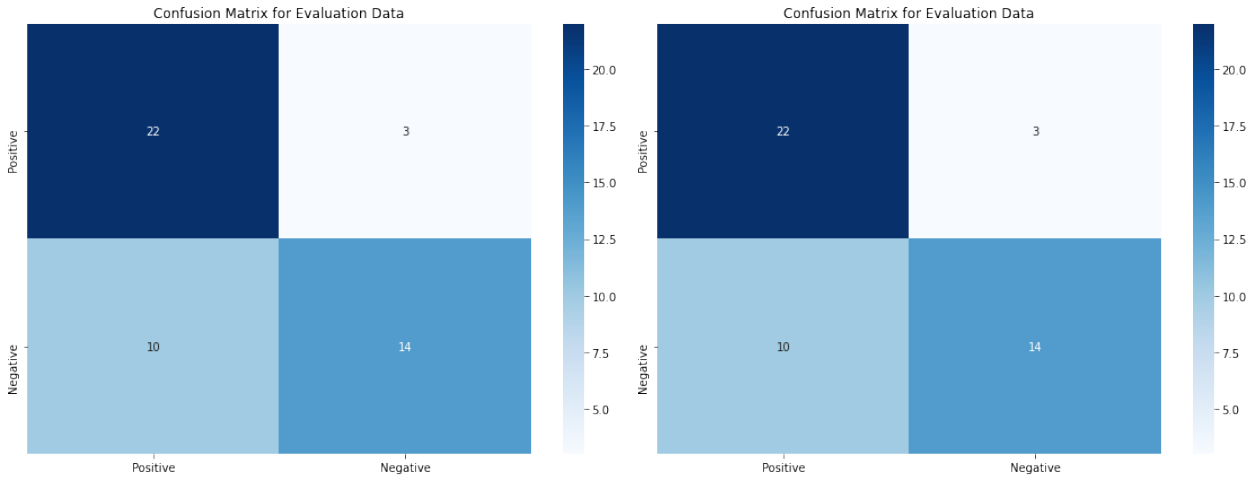


(a) TF-IDF

(b) Count Vectorizer

Figure 1: Results of Logistic Regression on Custom Survey Dataset

For Multinomial NB, we obtained 73% test accuracy and an F1 score of 77% by using TF-IDF Vectorizer, and a 73% test accuracy and an F1 score of 77% by using CountVectorizer.



(a) TF-IDF (b) Count Vectorizer

Figure 2: Results of Multinomial NB on Custom Survey Dataset

Judging from the results shown above, these are still considered significantly low and improvements can be made. As such, we explored several more advanced models to improve performance.

4 Simple LSTM

RNN with Long Short-Term Memory (LSTM), is an artificial neural net architecture designed to handle sequential related tasks such as NLP [7]. Memory cell in LSTM stores activation values from previous intervals and unlike RNN this provides extra information based on the previous iterations. Additionally, this also solves the vanishing gradient problem faced by RNN and due to this factor LSTM is a suitable choice for our classification task.

The dataset that will be used for training the LSTM model is the *Malaysian Twitter Sentiment Dataset* [6] . However, before proceeding into training our model, we first perform text preprocessing for our input data, which is an essential step for any NLP task.

4.1 Preprocessing Data

1. **Tokenization**

   Tokenization is a common preprocessing task in an NLP problem. The idea behind performing tokenization is it breaks down sentences into tokens. Having space as our delimiter, we are able to segment sentences to individual words. This method is called Word Tokenization. Most neural network architectures that carry out NLP tasks handle raw text at a token level, which includes LSTM. With the tokens, we are then able to prepare our own Malaysian vocabulary.

   Two special tokens are introduced here, [PAD], that is used for padding on shorter sentences, and [UNK], to handle unknown words that are not in our Malaysian vocabulary.

2. **Lower Casing**

Lower casing the text data reduces the number of dimensions of the vocabularies. We performed a frequency counter on both occasions (with and without lower casing). The results we retrieved are 132,146 vocabularies without lower casing, and 127,924 vocabularies after lower casing. That is a significant drop of around 5,000 words.

Unfortunately dealing with the Malaysian language, there are no resources that remove or identify Malaysian stopwords. The only possible way would be to manually collect every stop word in the Malaysian language. However, due to the complexity of the task, it is difficult to remove them.

Punctuations in sentences are known to be unhelpful for NLP tasks, it can be considered as noises. Python3 has a library that provides predefined punctuations. With that, we are able to remove punctuations in the dataset.

3. **Padding**

   For our Twitter dataset, each twitter comment comes in varying sequence length. Since the LSTM model takes in sequential inputs that are the same length, input sequences are padded to a maximum length, and truncated if it goes beyond it.

   There are two ways in which we can add paddings to input sequences, padding at the beginning of a sequence (pre) and padding at the end of a sequence (post). In some cases, padding at the beginning gives better results while some might not. However, pre-padding sequences would be more suitable in LSTM due to it's long short-term memory behaviour [8]. We set a fixed maximum sequence length of 61, as that is the length of the longest comment.

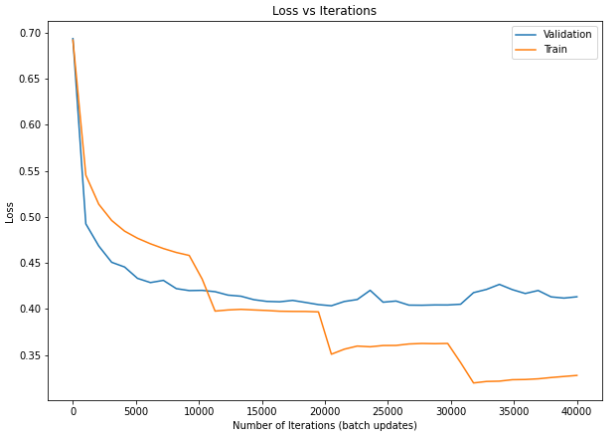
4.2 Training

The LSTM model used in our training has one embedding layer, 2 hidden layers each having 256 hidden neurons with a dropout of 0.5, and one fully connected linear layer.

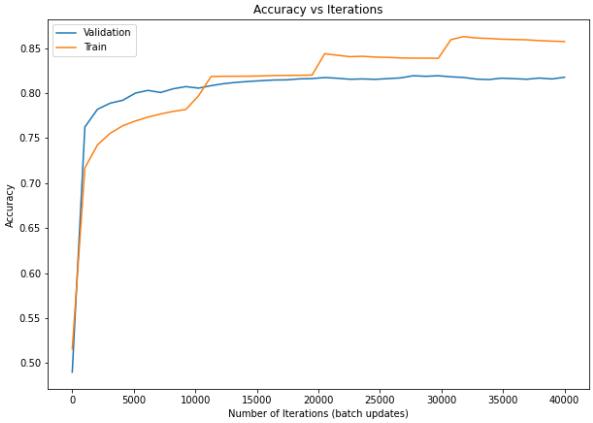
The setup of the pre-training of the LSTM are as below:

- Epochs: 4
- Criterion: Cross Entropy Loss
- Optimizer: AdamW, with weight decay of 0.01 and Learning Rate of 5e-3
- Scheduler: LambaLR scheduler
- Batch Size: 64, 128
- Sequence Length: 61

4.3 Results

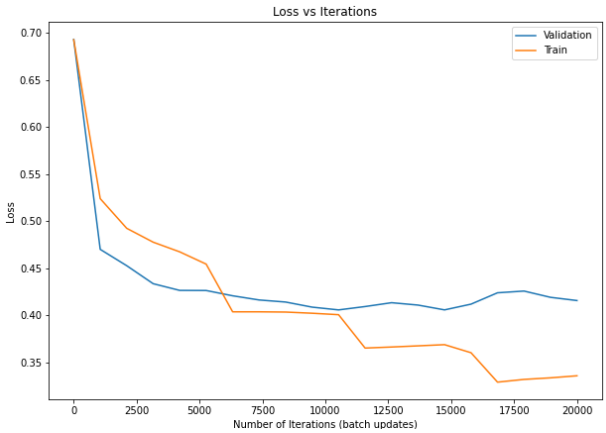


(a) Loss vs Iterations

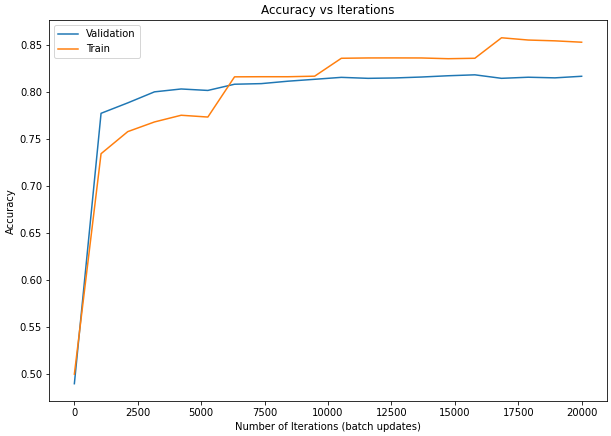


(b) Accuracy vs Iterations

Figure 3: Results of LSTM on batch size 64



(a) Loss vs Iterations

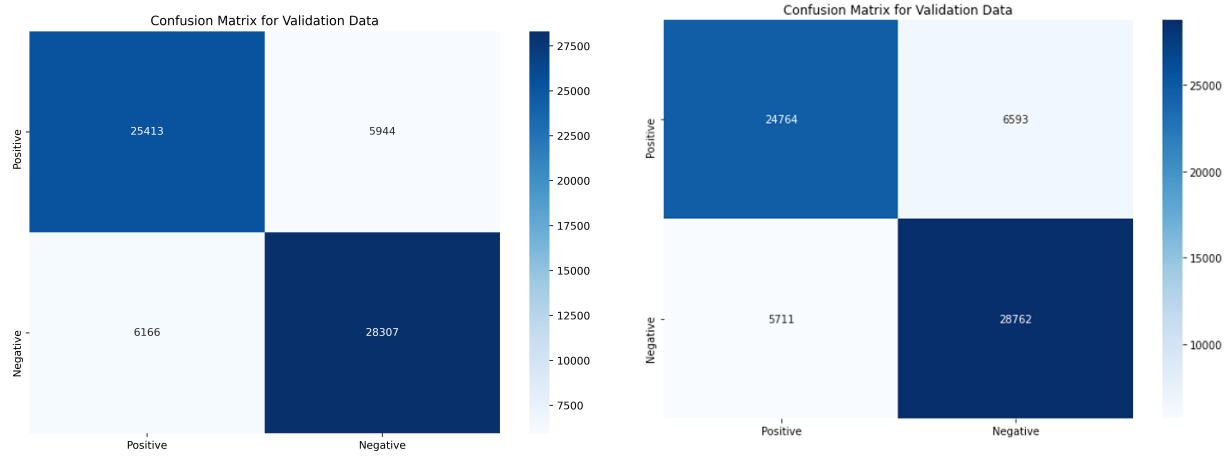


(b) Accuracy vs Iterations

Figure 4: Results of LSTM on batch size 128

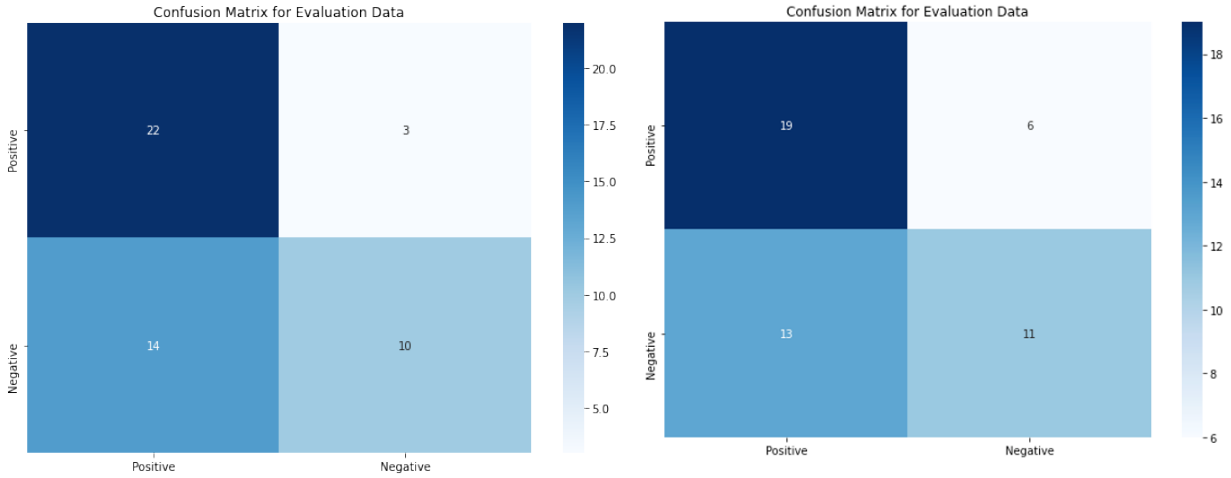
Fig.3a shows the training and validation losses for 4 Epochs (each epoch with 10,000 iterations) with batch size of 64, as well as the training and validation accuracies in fig.3b. Fig.4a is similar to fig.3a , except it trains in batch size of 128 (each epoch with 5,000 iterations). This is the same case between fig.4b and fig.3b.

Looking at fig.3a and 3b, we can observe that overfitting starts occurring slightly after 20,000 iterations (2nd Epoch). Additionally in fig.4a and 4b, we can also see that overfitting occurs right after 10,000 iterations (2nd Epoch as well). With that being said, training our model with 2 Epochs produces the best results. Hence, we proceed in using 2 epochs and performed tests.



(a) LSTM Validation BatchSize 64 (b) LSTM Validation BatchSize 128

Figure 5: Confusion Matrix on validation dataset



(a) LSTM Custom Dataset BatchSize 64 (b) LSTM Custom Dataset BatchSize 128

Figure 6: Confusion Matrix on validation dataset

Batch Size	Validation Accuracy (%)	Testing Accuracy (%)
64	81.6	65.3
128	81.3	61.2

Table 1: Results from Validation and Testing

As we can observe in Table 1, having a batch size of 64 generally performs better than batch size of 128, with higher validation and testing accuracies.

From our final results, the Simple LSTM model trained on batch size 64 with 2 epochs gave the best test accuracy of 65.3%, compared to batch size 128 with 2 epochs. Comparing back to our

base models, we are yet to be able to obtain significantly higher accuracy. Therefore, we move onto our next model.

5 BERT

BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) is a transformer based architecture developed by Google which is designed to handle natural language processing tasks such as question answering and language inference [9]. BERT architecture is built up by stacked encoders in transformers. Unlike LSTM models, transformer based models like BERT capture context and meaning of words really well due to the self-attention mechanism. For this project, we used BERT for the masked language model, which is a BERT model with only maskedLM head.

5.1 Transformers

Transformer is a deep learning architecture that adopts the idea of attention mechanism, it is designed to handle natural language processing tasks such as translation and text summarization. Unlike LSTM which treats input data sequentially, transformers handle input data by running in parallel, which results in faster training time, no recurrences and long term dependencies. The attention mechanism function is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

Where Q, K, V in eq.1 represent the attention vector, keys of the query and value respectively. These values are then normalized by d_k, the dimension of either of the three vectors [10]. Transformer architecture consists of 2 halves which are encoder and decoder as shown from fig.7.

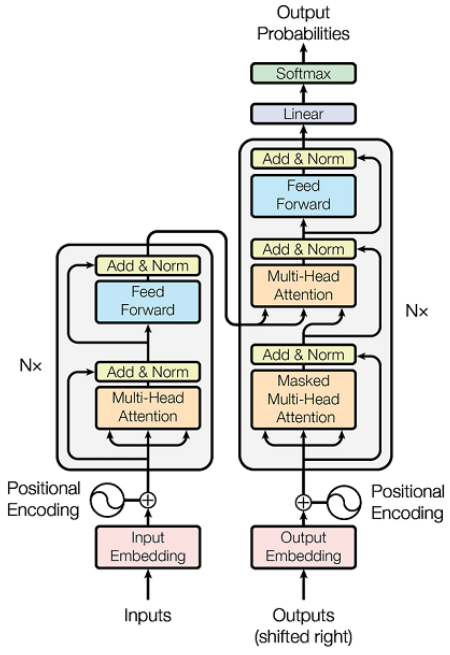


Figure 7: Architecture of Transformers [10]

Attention has been proven by *Bahdanau, Cho, Bengio* [11] to extract context and meaning of the input data. BERT architecture utilizes the encoders from transformers which includes the multi-head attention layer and hence BERT is a suitable model to handle this task.

5.2 Pre-processing

1. **Tokenization**

   For tokenization, we used **BertWordPieceTokenizer** from Hugging Face [12]. Instead of creating new subwords by merging from all possible characters according to their frequency occurrence in the corpus, **BertWordPieceTokenizer** selects the characters to be merged by considering the maximum likelihood of the training data once the subword has been added to the vocabulary. In other words, instead of merging subwords if their frequency occurrence is the highest, 2 subwords or characters are merged to form a new subword if and only if the probability of the new subword divided by the previous 2 subwords or characters are the greatest in the entire vocabulary. To train our tokenizer, we used the Malaysian Corpus Dataset from OSCAR. The vocabulary size after tokenization is 30,522.

2. **Text Grouping**

   Once tokenization is complete, we perform text grouping. Transformers require 3 inputs which are input IDs, token type IDs, and attention masks. Using the inputs, the corpus is encoded into its numerical representation with special tokens as well as attention masks. All encoded texts are split to fit within the sequence length of 128. Long sequences are split to multiple shorter sequences. Similarly, shorter sequences are concatenated together to form a longer sequence.

3. **Data Collation**

   As mentioned on section 5, BERT is trained by performing masked language modeling. In this step, random masking is applied to the sequences. The idea behind this is to use other tokens to predict the word represented by the masked token. In this project, each token has a 0.15 probability of becoming a label for training.

5.3 Language Model Pretraining

The first phase for training BERT is the pretraining phase, where it involves training BERT model from scratch on the selected corpus dataset. Instead of initializing random weights, we retrieved the BERT model that has been pretrained on the english corpus dataset.

5.3.1 Match Embedding

Another key point that needs to be mentioned about the pretraining process is a technique called match embeddings. This idea is proposed by Jeremy Howard [13] and has been proven to be efficient in pretraining language models. The process involves copying embedding representations that have matching tokens / words from the pretrained english BERT model to the Melayu BERT model. It is ineffective to initialize embeddings randomly for the same words / tokens that exist in the

pretrain english BERT model. Since the Malaysian Corpus Dataset has a significant amount of english words, we realise that this technique would be beneficial for the pre-training process.

5.3.2 Training

The BERT model used as mentioned in section 5 is bert based uncased from hugging face [12] which has 110 million parameters and 11 encoders and is trained on Malaysian Corpus Dataset [5]. The setup of pretraining are as shown below:

- Epochs: 3
- Criterion: Cross Entropy Loss
- Optimizer: AdamW, with weight decay of 0.01 and learning rate of 2e-3
- Scheduler: LambaLR scheduler
- Batch size: 64
- Sequence Length: 128

5.3.3 LM Pretraining Results

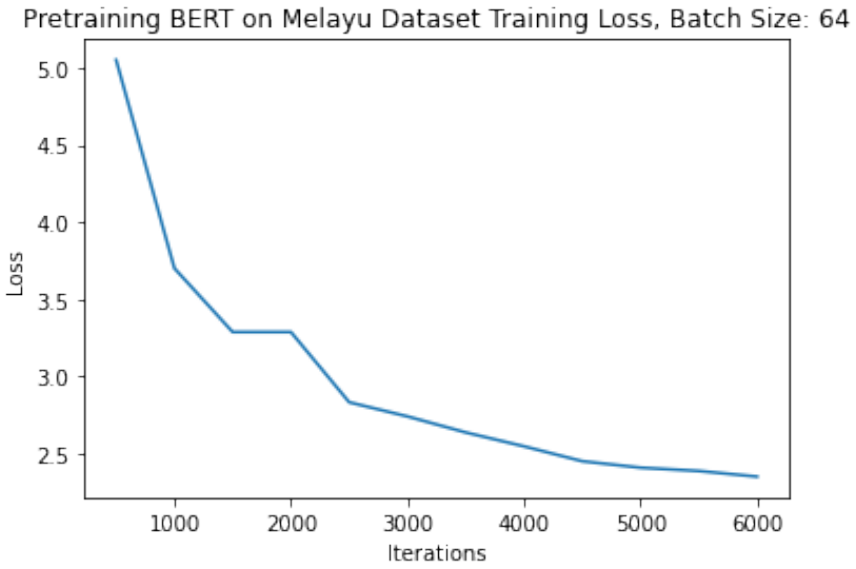


Figure 8: Training Loss vs Iterations

Fig. 8 shows the training loss as the number of weights updated per batch increases. We obtained a final validation loss of 2.24 and a training loss of 2.93, with a validation perplexity of 9.46. It can be seen from fig.8 that the loss might still decrease steadily by running a few more epochs; however, due to the limitation in computational power it is not possible to continue the training process.

5.4 Fine Tuning

The next procedure is fine tuning in which the pretrained language model is transfer-learned to a specific task, which in our case is sentiment analysis. The dataset used is Malaysian Twitter Sentiment Dataset. The pre-processing procedures are slightly different. Unlike pre-training which requires tokenization, text grouping and data collation, in the fine tuning phase only tokenization is required utilizing the tokenizers that were trained during LM pre-training phase.

5.4.1 Training

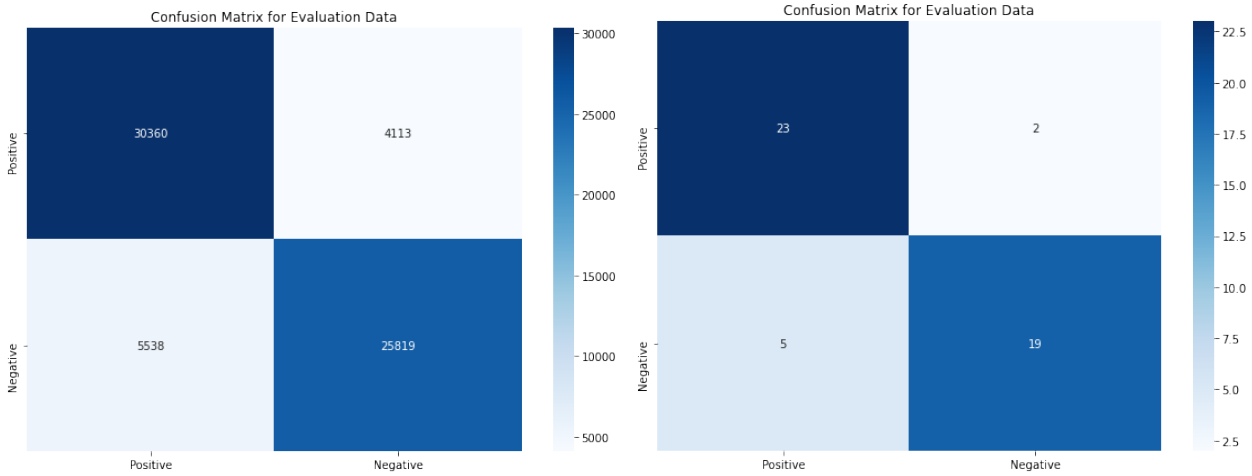
The pretrained language model is used for fine tuning by switching the pretrained model head with a BERT pooler head, a dropout and a fully-connected layer. The purpose of the fully-connected layer is to squeeze the hidden representation to the number of classes in the case of classification tasks.

The setup of pretraining are as shown below:

- Epochs: 3
- Criterion: Cross Entropy Loss
- Optimizer: AdamW, with weight decay of 0.01 and learning rate of 3e-5
- Scheduler: LambaLR scheduler
- Batch size: 128
- Sequence Length: 128

5.4.2 LM Fine Tuning Results

After fine tuning the BERT model, we validated and tested our model.



(a) BERT Fine Tuning Validation

(b) BERT Fine Tuning Test

Figure 9: Confusion Matrix on validation dataset

For validation, we obtained a training loss of 0.36 and a validation loss of 0.4. In terms of accuracy, the validation accuracy is 85% and the F1 score is 84%. For testing, we achieved 86% accuracy and the F1 score is 87%.

Overall, when comparing performance to our baseline models, we can see a significant improvement in terms of accuracy.

6 Multilingual BERT

mBERT (Multilingual BERT) is a BERT based model that is developed by Google which is a BERT model that is trained on 100 different language corpus (including the Malaysian language). The significant difference between normal BERT model and mBERT is in the number of parameters which is 167 million parameters. Due to the significant difference of parameters,a TPU (Tensor Processing Unit) is required to train mBERT. mBERT is used as an upper benchmark to compare with our BERT model.

6.1 Fine Tuning

Multilingual BERT does not require pretraining as it has been trained on the Malaysian language corpus, although the corpus dataset used might be different. The process of fine tuning is exactly the same as fine tuning the BERT model section 5.4, the only difference is that it utilizes the Multilingual BERT tokenizer retrieved from Hugging Face.

6.1.1 Training

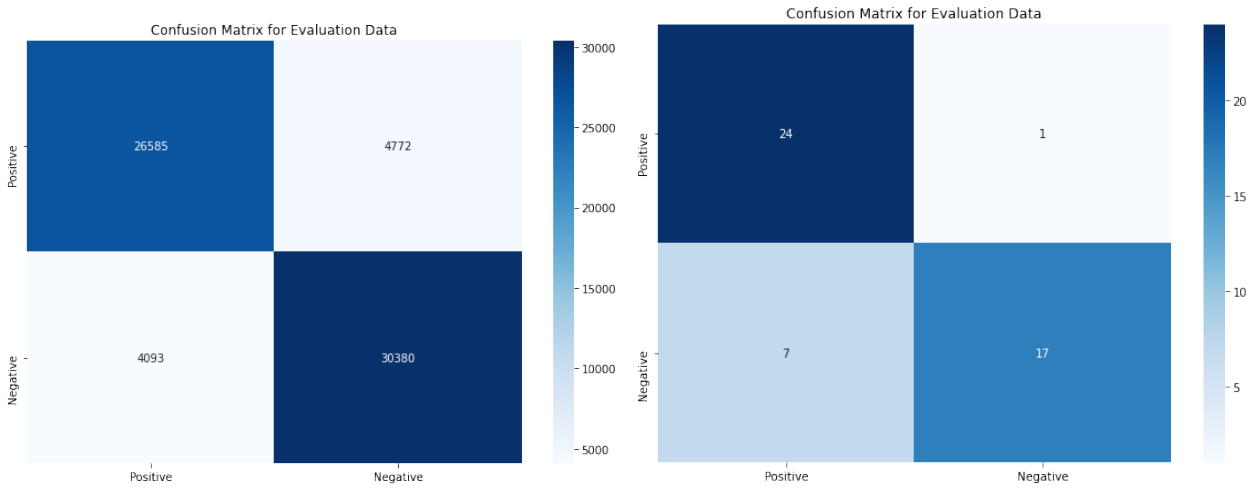
Similarly like training BERT, the head is replaced with a BERT pooler layer, as well as a drop out and fully-connected layer.

The setup of pretraining are as shown below:

- Epochs: 3
- Criterion: Cross Entropy Loss
- Optimizer: AdamW, with weight decay of 0.01 and learning rate of 3e-5
- Scheduler: LambaLR scheduler
- Batch size: 128
- Sequence Length: 128

6.1.2 LM Fine Tuning Results

As for fine tuning results on the Multilingual BERT model, we obtained these results.



(a) mBERT Fine Tuning Validation

(b) mBERT Fine Tuning Test

Figure 10: Confusion Matrix on validation dataset

The training loss is 0.31 and validation loss is consistently 0.38 for all epochs. In terms of accuracy, the validation accuracy is 87% and the F1 score is 86%. Additionally, the test accuracy is 84% and the F1 score is 86%.

Similar to our BERT model, our Multilingual BERT model also shows a significantly better performance as compared to the baseline models. As for the performances between BERT and Multilingual BERT, it is evident that the accuracy for Multilingual BERT during validation is slightly higher than BERT. However, the test accuracy for BERT is slightly higher than Multilingual BERT.

7 Discussion

7.1 Conclusion

We performed various models starting from the baseline to sophisticated models. After conducting several experiments on various models, these are the results obtained running the best models for different architecture, testing on the custom survey dataset.

Model	# of Parameters	Validation Accuracy(%)	Test Accuracy(%)
Simple LSTM (batch size 64)	34M	82	65
BERT	110M	85	86
Multilingual BERT	167M	87	84
Logistic Regression (TfidfVectorizer)	230K	79	63
Logistic Regression (CountVectorizer)	230K	78	65
Multinomial NB (TfidfVectorizer)	-	76	73
Multinomial NB (CountVectorizer)	-	77	73

Table 2: Results from Validation and Testing

As shown from Table 2, although the validation accuracy of BERT is lower compared to Multilingual BERT, BERT has the highest test accuracy. The pretraining of BERT might be the reason behind this better performance since the corpus might contain different vocabs compared to the

Multilingual BERT. The multiple corpus trained on Multilingual BERT might also have an effect on the accuracy as the vocab is now limited due to multiple corpus vocabs. All deep learning models surpass the baseline traditional machine learning models in validation accuracy; however, comparing all the models judging from the test accuracy LSTM performs the worst.

Considering the massive difference in number of parameters and training time, only slight improvements can be seen judging from the accuracy score. This is due to the fact that transformer based models perform better in sequence-to-sequence problems. Although running on a classification problem is a good benchmark to test the performance of models, training with a dataset that has more number of classes to be classified might be a better benchmark test.

7.2 Limitations

To train our models as efficiently as possible, we used a GPU (Tesla P100, 16GB VRAM) and a TPU from Google Colab. Despite using this method, we still faced an issue relating to long training times. For Simple LSTM, the process took 30 minutes for 4 epochs. For BERT, the pre-training process took 5 hours for 3 epochs, whereas the fine tuning to sentiment text classification aspect took 4 hours and 30 minutes. This brings a total duration to 9 hours and 30 minutes. For Multilingual BERT, we did not have to perform pre-training as this process was already implemented for the model, however, the duration for fine tuning to sentiment text classification took a longer process of 6 hours for 3 epochs. Since we perform training on Google Colab, this meant that we had to continuously be connected to Google's online servers. Due to unforeseen circumstances, we have experienced technical difficulties and termination errors that resulted in restarting the training procedure.

As mentioned above, we can observe that the duration taken for training these models are especially long. Hence, we were unable to perform additional hyperparameter tuning to improve the model's results, additionally with a lack of computational power and the limit set by Google Colab per each account. Based on the result obtained, we were still able to show that the complicated models performed slightly better overall than the baseline models.

8 Further Work

Besides LSTM, BERT and Multilingual BERT, we also explored the ULMFiT technique that was introduced by Jeremy Howard [13]. We used AWD-LSTM for the ULMFiT technique test and managed to pretrained the model with the Malaysian Corpus Dataset. However, we have faced a few issues during the tokenization phase when we perform Target Task LM Fine Tuning [13], and it turned out to be a problem when using the Fastai [14] framework on Google Colab. As a result, we did not manage to work with this model.

Looking from papers like [15], it is shown that the model that is trained was compared to XLM models. Unlike mBERT which is a multilingual model, XLM is a cross-lingual model. It is trained with 2 techniques which are Masked Language Model (same as BERT models) as well as Causal Language Model [16]. As a result, it is shown to perform better compared to mBERT. This might be a better upper benchmark compared to mBERT.

References

[1] "Twitter: number of users worldwide 2019-2020 — statista," Statista, 2019. [Online]. Available: https://www.statista.com/statistics/303681/twitter-users-worldwide/

[2] M. Szmigiera, "Most spoken languages in the world — statista," Statista, 2017. [Online]. Available: https://www.statista.com/statistics/266808/the-most-spoken-languages-worldwide/

[3] I. Ghosh, "Ranked: The 100 most spoken languages around the world," Visual Capitalist, 02 2020. [Online]. Available: https://www.visualcapitalist.com/100-most-spoken-languages/

[4] D. D. Luxton, J. D. June, and J. M. Fairall, "Social media and suicide: A public health perspective," *American Journal of Public Health*, vol. 102, pp. S195–S200, 05 2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3477910/

[5] OSCAR, "Oscar," OSCAR. [Online]. Available: https://oscar-corpus.com/

[6] h. , "huseinzol05/malay-dataset," GitHub. [Online]. Available: https://github.com/huseinzol05/malay-dataset/tree/master/dumping/twitter

[7] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2222–2232, 10 2017. [Online]. Available: https://arxiv.org/pdf/1503.04069.pdf

[8] D. Reddy and S. Reddy, "Effects of padding on lstms and cnns." [Online]. Available: https://arxiv.org/pdf/1903.07288.pdf

[9] J. Devlin, M.-W. Chang, K. Lee, K. Google, and A. Language, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, G. Brain, G. Research, L. Jones, A. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2015. [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf

[12] "Tokenizer summary — transformers 3.0.2 documentation," huggingface.co. [Online]. Available: https://huggingface.co/transformers/tokenizer_summary.html

[13] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018. [Online]. Available: https://arxiv.org/pdf/1801.06146.pdf

[14] J. Howard and S. Gugger, "Fastai: A layered api for deep learning," *Information*, vol. 11, p. 108, 02 2020.

[15] B. Wilie, K. Vincentio, G. Indra Winata, S. Cahyawijaya, X. Li, Z. Lim, S. Soleman, R. Mahendra, P. Fung, S. Bahar, and A. Purwarianti, "Indonlu: Benchmark and resources for evaluating indonesian natural language understanding." [Online]. Available: https://arxiv.org/pdf/2009.05387.pdf

[16] G. Lample and A. Conneau, "Cross-lingual language model pretraining." [Online]. Available: https://arxiv.org/pdf/1901.07291v1.pdf