Progress Report I - Peace Speech Project

Hongling Liu (hl3418), Haoyue Qi (hq2180),
Xuanhao Wu (xw2744), Yuxin Zhou (yz3904), Wenjie Zhu (wz2536)

October 26, 2021

I. Problem Definition and Progress Overview

In the current world of rising conflicts and wars, peacekeepers are actively conducting research on hate speech analysis in seeking to maintain robust and peaceful communities. Such hate speech dataset is accessible on the internet including but not limited to news articles, blogs and social media posts. Meanwhile, the peace speech, which is also an important yet available piece of language assets, commonly gets neglected. Capstone students in the past years had worked with Professor Coleman's team to conduct pilot studies that showed promising powers of the peace speech in distinguishing the peacefulness of societies. This year, we will continue the previous research with a much richer dataset and two new approaches.

First, we will explore the performance of more state-of-arts natural language processing (NLP) models on classifying English articles from high-peace and low-peace countries (*Figure 1-1, Task 1*). A high performance model would indicate the existence of language differences in the articles from countries with different peace levels. Different from the study conducted by previous years' capstone students, we will put our focus primarily on using the contextual text encoders like the Bidirectional Encoder Representation from Transformers (BERT) rather than bag-of-words to generate text embeddings, and then using that embeddings as inputs to explore different classification modeling approaches. Then, we will take a deep dive into studying how the interrelationship among words affect the previous classification results by evaluating the models' performance over randomly shuffled articles (*Figure 1-1, Task 2*). A drop in the performance would then prove that the relationship among words in one article to be a strong peacefulness indicator.

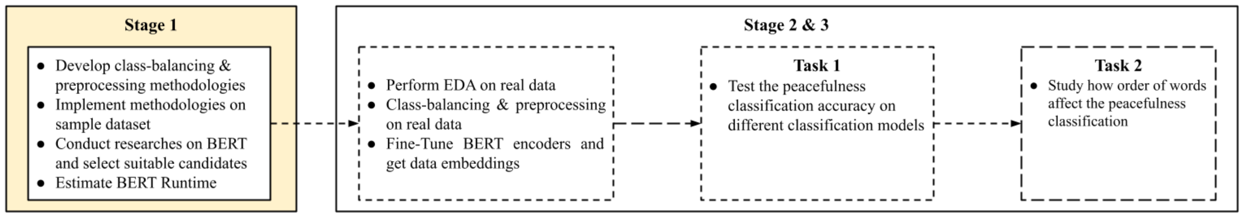

Stage 1
Develop class-balancing & preprocessing methodologies
Implement methodologies on sample dataset
Conduct researches on BERT and select suitable candidates
Estimate BERT Runtime
Stage 2 & 3
Perform EDA on real data
Class-balancing & preprocessing on real data
Fine-Tune BERT encoders and get data embeddings
Task 1
Test the peacefulness classification accuracy on different classification models
Task 2
Study how order of words affect the peacefulness classification

Figure 1-1 Progress Overview

Above is an overview of our project timeline. The highlighted stage is where we are. In the following sections, we will talk about our current work in detail, including data and programing platform introduction, class balancing methodology, data preprocessing pipeline, and an initial exploration on a basic neural network model using the BERT output focusing on studying the computation time of fine-tuning the pre-trained BERT encoder on our sample dataset.

II. Data and Computational Platform

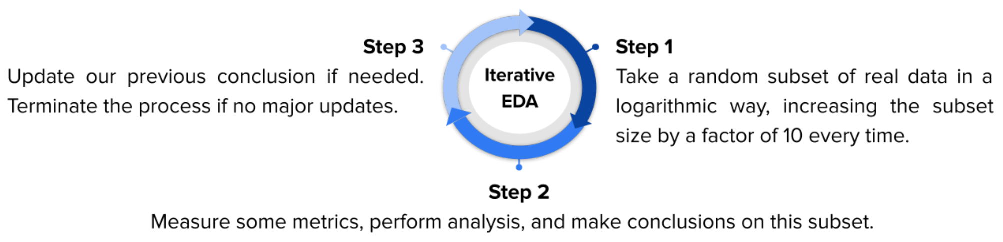
The real data is provided by LexisNexis and is stored on the AWS S3 bucket as .json files in two separate directories: *highPeace* and *lowPeace*. The *highPeace* is a directory with news articles from the top 10 high peace countries around the world based on the *rule-of-5*[1] metric, and the same logic follows for the other folder (see *Table 2-1* for the list of countries). Each .json object is one data point. There are ~33M articles (.json files) from high peace countries and ~57M articles from low peace countries. We will use AWS SageMaker cloud computation platform throughout this project, since it has a more compatible connection with S3 bucket and a configurable number of GPUs and other computational resources. Using a basic ml.t2.medium notebook instance with 2 virtual CPUs on AWS SageMaker, loading 1K .json files of ~15KB each into notebook memory takes 1 minute on average.

Low Peace	High Peace
● Afghanistan ● Congo ● Guinea ● India ● Iran ● Kenya ● Nigeria ● Sri Lanka ● Uganda ● Zimbabwe	● Austria ● Australia ● Belgium ● Czech Republic ● Denmark ● Finland ● Netherlands ● New Zealand ● Norway ● Sweden

Table 2-1: List of Article Countries in Real Data

II.I. Exploratory Data Analysis (EDA) Methodology

Since the dataset is extremely large, the following issue might arise during this project: 1) The entire dataset does not fit into SageMaker Notebook memory, and 2) it's time and budget costly to repeatedly read & write files across S3 bucket and SageMaker. Therefore, it would be a challenge to perform EDA on this dataset entirely. To solve this problem, we proposed the following method in *Figure 2-1*:



Step 1
Take a random subset of real data in a logarithmic way, increasing the subset size by a factor of 10 every time.

Step 2
Measure some metrics, perform analysis, and make conclusions on this subset.

Step 3
Update our previous conclusion if needed. Terminate the process if no major updates.

[1] For each country, scores for indices such as Global Peace Index, the Positive Peace Index, etc., are computed. Then countries are placed into low-, mid-, and high-peace categories for each index. If a country has at least 5 indices categorized as low-peace, and no index in high-peace, then the country is low-peace. Same logic follows for high-peace.

Figure 2-1: Iterative EDA process

After a few iterations, the data distribution will converge to its true value within a confidence interval due to the increasing sample size per step. We will perform this iterative EDA on the *lowPeace* and *highPeace* data points separately, and compare the results in the end to draw some overall conclusions. Some metrics we would like to measure are: 1) article distribution across countries, and 2) average article word count. Since we didn't have access to the SageMaker environment and the data until the last week of stage 1, and the above EDA on a small set of sample data with 140 data points we got earlier would not be very insightful, we will hold on to more detailed EDA on the real data and present them together with next-step works in report 2.

II.II. Class Balancing Methodology

The sample dataset has 20 news articles from each of the 7 selected countries around the world, with roughly half of the countries identified as high-peace and half as low-peace. So the class distribution is balanced. However, there is a great imbalance between the number of articles in English from different countries in the real LexisNexis Dataset. For example, there are more than two hundred million articles in English from the United States, while there are only over one million articles in English from the United Arab Emirates. As the goal for our project is to create a classification model to distinguish between high-piece countries and low-piece countries, the imbalance between the number of articles from different countries will cause an imbalance between positive and negative samples and thus making the classification model very easy to overfit and perform badly on the test set among metrics like accuracy, precision and recall. To figure out the problem of imbalanced data, usually resampling is a good method and there are two types of resampling methods which are oversampling and undersampling. For our project, since there are a tremendous amount of articles in the LexisNexis Dataset and we have limited computing resources, we choose to use the method of undersampling to make training data balanced which means we randomly sample samples of bigger size to equalize positive and negative samples' size.

III. Text Pre-processing

Each news article contains a title and a content. Since the title is a summary with dense information related to the content, we concatenate the title to the content and treat this as the natural language data that we will work with in this project.

III.I. Methodology

Many encoders belonging to the BERT family: like BERT, ALBERT, RoBERTa, and DistilBERT are powerful enough to interpret inputs with special characters and numbers. Furthermore, according to Alzahrani et al.[1], BERT performs the best on a binary classification task when its input is raw text. Camacho-Collados et al.[2] also achieved a similar conclusion for general neural network models in topic categorization and sentiment detection tasks. Though preprocessing techniques such as removing stopwords is commonly seen in handling inputs for other NLP models, such methods will break the original input context and result in losing information. For example, the negation stopword "*not*" is often related to a shift in sentiment and might be related to the peacefulness pattern we are looking for. Since we are going to fine-tune a BERT encoder that has been pre-trained on raw natural language, overly preprocessing our data would only negatively impact BERT's ability in learning such text patterns. Yet, there are many named entities in the articles that might disclose the information such as the name of a city, an author who frequently writes reports for a country, or an organization that's doing business in a country. To avoid the encoder model from learning those biased aspects of the data, we would need to remove that information from our input. Besides that, there are two main types of English language people use across the world: American and British English. We need to convert all British spelling to American spelling so the spelling of the word won't disclose the region where the article comes from, and the two versions of a word can be mapped onto the same embeddings which reduces the sparsity in text mapping.

There are several steps we take to accomplish this task. The entire pipeline is shown in *Figure 3-1*. Note that the SpaCy English NLP model in step 3 comes with different sizes: sm, md, trf, and lg. Since we have limited computational resources, and the larger model doesn't have a significant advantage in named entity recognition (NER) tasks as shown in *Table 3-1*, we use the small model for NER. As stated, the targeted entities to be removed are those that might disclose country-related information: organizations, geographical names, and person names. The output of this pipeline will be English sentences ready to be fed into a pre-built preprocessor that is paired with each encoder in the BERT family. Both the BERT encoders and the paired preprocessors are available for download on Tensorflow Hub and Hugging Face.

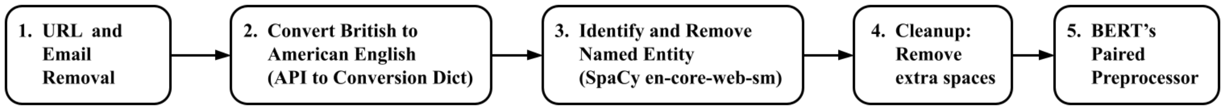
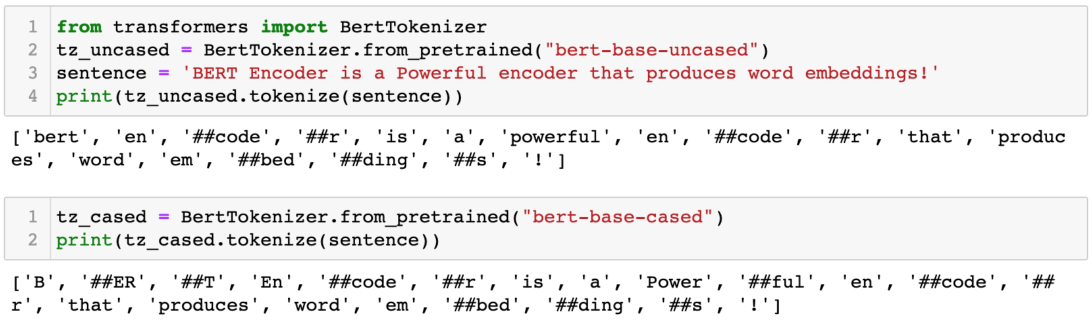

1. URL and Email Removal
2. Convert British to American English (API to Conversion Dict)
3. Identify and Remove Named Entity (SpaCy en-core-web-sm)
4. Cleanup: Remove extra spaces
5. BERT's Paired Preprocessor

Figure 3-1: Preprocessing Pipeline Design for Encoders in BERT family

Model	Size	Tokenization Accuracy	NER precision	NER recall	NER F1
en-core-web-sm	12 MB	1.00	0.84	0.83	0.84
en-core-web-md	43 MB	1.00	0.85	0.85	0.85
en-core-web-trf	438 MB	1.00	0.90	0.90	0.90
en-core-web-lg	741 MB	1.00	0.85	0.85	0.85
Source: SpaCy documentation (https://spacy.io/models/en)					

Table 3-1: Comparison between SpaCy NER models of different sizes

Figure 3-2 shows a sample tokenization result from BERT-base-uncased and cased models. Note how the two tokenizers tokenize the word "Powerful" differently. Depending on which version of the BERT we use, its paired preprocessor will lower-casing all input letters for us if necessary. Then it will send the sentence to a WordPiece tokenizer that will perform a subword tokenization which leads to results similar to stemming and lemmatization. For example, the word "*embedding*" will be splitted into four tokens: "*em*"(prefix), "*##bed*"(root), "*##ding*"(suffix), "*##s*"(plural). Each of the four tokens in BERT will have its own word embedding, position embedding, and segment embedding. BERT sum up those embeddings internally to resemble the meaning of this word in the context it appears in. Therefore, we don't need to worry about stemming and lemmatizing words ourselves.



```
from transformers import BertTokenizer
tz_uncased = BertTokenizer.from_pretrained("bert-base-uncased")
sentence = 'BERT Encoder is a Powerful encoder that produces word embeddings!'
print(tz_uncased.tokenize(sentence))
```

```
['bert', 'en', '##code', '##r', 'is', 'a', 'powerful', 'en', '##code', '##r', 'that', 'produc
es', 'word', 'em', '##bed', '##ding', '##s', '!']
```

```
tz_cased = BertTokenizer.from_pretrained("bert-base-cased")
print(tz_cased.tokenize(sentence))
```

```
['B', '##ER', '##T', 'En', '##code', '##r', 'is', 'a', 'Power', '##ful', 'en', '##code', '##
r', 'that', 'produces', 'word', 'em', '##bed', '##ding', '##s', '!']
```

Figure 3-2: Sample Tokenization Result from BERT

III.II. Preprocessing Result on Sample Data

Before preprocessing	After preprocessing
Sadia Dada, Director Communications PMPKL, said, "The challenge of littering is not new to our country nor the efforts to combat it. Lack of awareness and infrastructure for disposal are key drivers abetting this bad habit adding that most important is how minor changes in the way we work can create room for each and every individual of society to be a part of it."\n\nPublished by HT Digital Content Services with permission from Daily Times. For any query with respect to this article or any other content requirement, please contact Editor at contentservices@htlive.com	, Director, said," The challenge of littering is not new to our country nor the efforts to combat it. Lack of awareness and infrastructure for disposal are key drivers abetting this bad habit adding that most important is how minor changes in the way we work can create room for each and every individual of society to be part of it." Published by with permission from.
* Highlighted parts are deleted in preprocessing	

Table 3-1 Sample article (id: 43749542108) before and after preprocessing

The output articles of the preprocessing pipeline for BERT are still human readable as the majority of the contexts are preserved as shown in *Table 3-1*. As will be explained later on in this report, we would like to focus mainly on the RoBERTa-base model from the BERT family. So we tokenize the result using the tokenizer from that model and generate the visualization below (*Figure 3-3*). We see that the article token count distribution is right-skewed with a median of 542.5 tokens before preprocessing and 365.5 tokens after preprocessing. The 3-4 outliers in each boxplot represent those extremely long articles, which make up only ~2% of the total sample data. Most of the pre-trained BERT models take 512 tokens in maximum, and we have ~64% of the articles that can fit in standard BERT input without truncation after running through our preprocessing pipeline, which increases ~15% from not having any preprocessing at all. For articles with token length above 512, we will lose ~30% of article information (tokens) on average. But according to Ganhotra et al.[3], BERT has a minimum performance drop when using a default tail-off truncation method. Though as suggested by Sun et al.[4], using a head-off or keep head-and-tail truncation method might lead to a lower error rate, the 0.2% of improvement in their experiment is not significant enough to persuade us to switch our focus to that topic. If time allows in stage 2 and 3, we can then try each method to see what works the best.

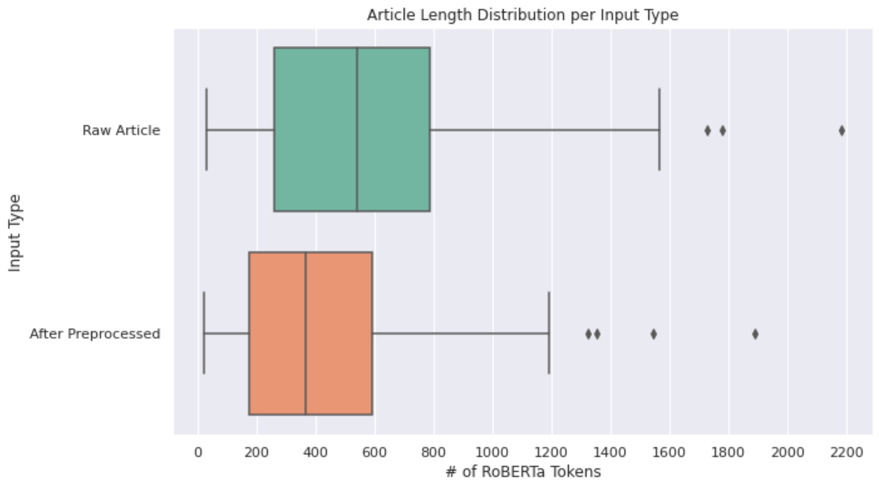


Figure 3-3 Preprocessing pipeline output length comparison

IV. Off-the-Shelf Model Exploration

IV.I. BERT Encoder Choice

As a state-of-the-art technique, there are many pre-trained BERT variations available on Tensorflow Hub and Hugging Face open source libraries that cover a wide range of sizes, languages, and fields of expertise. BERT's performance is directly impacted by the level of similarity between the data used for pre-training and the actual data. Our dataset contains mainly English news articles, so we can either choose a BERT that has been fine-tuned using news articles or start with a regular model from the BERT family and fine-tune it ourselves. The latter choice will require more computational resources, so the prior would be our first choice. We did some research online and narrowed down the candidates to the following three models listed in *Table 4-1*.

Model	Platform	Dataset	Parameter	Word Bank	Past Performance
RoBERTa-base	HuggingFace (Official)	Pretrained on: ● English Wikipedia, BookCorpus, CC-News, OpenWebText, Stories	125M	50K	SST-2 Score: 94.8 MRPC Score: 90.2
experts/bert/wiki_books/onli	Tensorflow	Pretrained on: ● English Wikipedia, BookCorpus Fine-tuned on: ● QNLI Task	110M	30K	MRPC Score: 90+
textattack/distilbert-base-uncased-ag-news	HuggingFace (OpenSource)	Pretrained on: ● English Wikipedia, BookCorpus Fine-tuned on: ● Adversarial Attack Approach on News Category Classification	66M	30K	ag_news sequence classification: 94.7

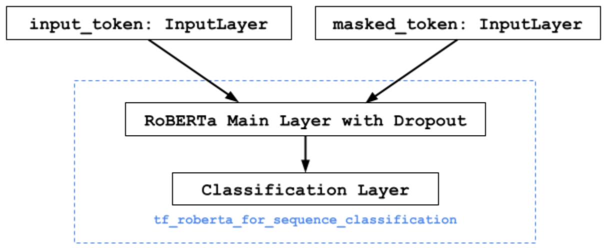
Table 4-1: BERT Encoder Candidates

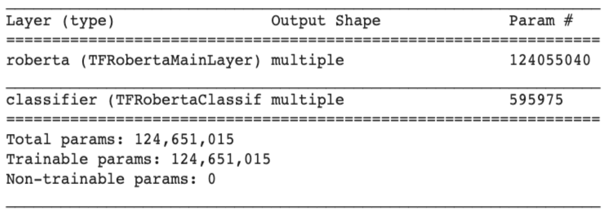
As mentioned in a previous section, we finally picked the RoBERTa-base as our first choice. Besides the fact that this model has many solid papers that are supporting it, it is managed by the HuggingFace Team and is more reliable than open source models. It could have a higher ability in interpreting news articles due to 63 million CC English news articles in its training set. In addition, we have more information on its past performance on different General Language Understanding Evaluation (GLUE) tasks. We hypothesize that if the articles truly have some peaceful indication power, then the peacefulness might be captured by the article's sentiment. The model's high score on the SST-2 (binary sentiment analysis) task qualifies it. Meanwhile, it also has a relatively prominent performance on the MRPC (English news article semantic binary comparison) task, which directly reflects its capability in encoding information in the news articles. Although this model is the heaviest among all three candidates, its capability and reliability outweigh this disadvantage. In the next step, we will mainly focus on exploring that model in more detail.

IV.II. Computation Time Estimate for Fine-Tuning a BERT Encoder

Since we will run all codes on SageMaker, and the computation could be costly, we did a test using the 140 sample data points to see how long RoBERTa will run. Since the SageMaker environment is not fully set up yet, we did the test with Google Colab 1-GPU runtime.

We used the well-packed *TFAutoModelForSequenceClassification.from_pretrained* function from HuggingFace transformers to load in the RoBERTa model we selected in the last section. This *ModelForSequenceClassification* object class would take care of adding the classification head to the RoBERTa's main layer for us, so we only need to configure this object according to our requirement during initialization, and no further model structure modification is needed. This basic fine-tuning structure takes two inputs: tokenized input IDs and attention masks. Inside the model, there is a RoBERTa-base encoder layer (*Figure 4-1: Main Layer*), a dropout layer with 0.1 dropout rate, and a final dense layer with width 7 (*Figure 4-1: Classifier Layer*), which is the number of countries in our sample dataset.


input_token: InputLayer
masked_token: InputLayer
RoBERTa Main Layer with Dropout
Classification Layer
tf_roberta_for_sequence_classification



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
roberta (TFRobertaMainLayer) multiple                  124055040
_________________________________________________________________
classifier (TFRobertaClassif multiple                  595975
=================================================================
Total params: 124,651,015
Trainable params: 124,651,015
Non-trainable params: 0
_________________________________________________________________
```

Figure 4-1 BERT fine-tuning model structure and summary

Common BERT models can take 512 input length in maximum. But due to the limited runtime memory in Colab, we have to shrink that to 128 for this test. This computation limit would not be a problem in SageMaker, because the runtime memory and computation resources over that platform would be configurable. As a result of this shrink in input length, the per sample training time we see in this trial would be lower than it should be when using 512 as the max input length in the actual training process. But the runtime complexity should still remain the same, which will be explained later. Then we took a random set of data from the sample dataset in a logarithmic way, increasing the sample size by a factor of 2 every time. For each sample size, we ran 5 trials with 20 epochs per trial and 10% validation split per epoch, then used the average runtime of the 5 trials to get a value plotted on the graph. We can see from *Figure 4-2*, the runtime is linearly increasing with the input size. As input size doubles, the runtime is also roughly doubled. In fact, we would expect to see a linear runtime complexity in fine-tuning a BERT model. Because we have a fixed number of parameters inside the model, and we would update this set of parameters for a fixed number of times per token in each sample and each epoch. The sample length is also capped by a max length, which is also a constant. So the runtime should be constantly proportional to the input sample size and input length. This plot (F*igure 4-2*) gives a proof for that and shows that even when we have a large number of inputs, BERT's runtime is still manageable.

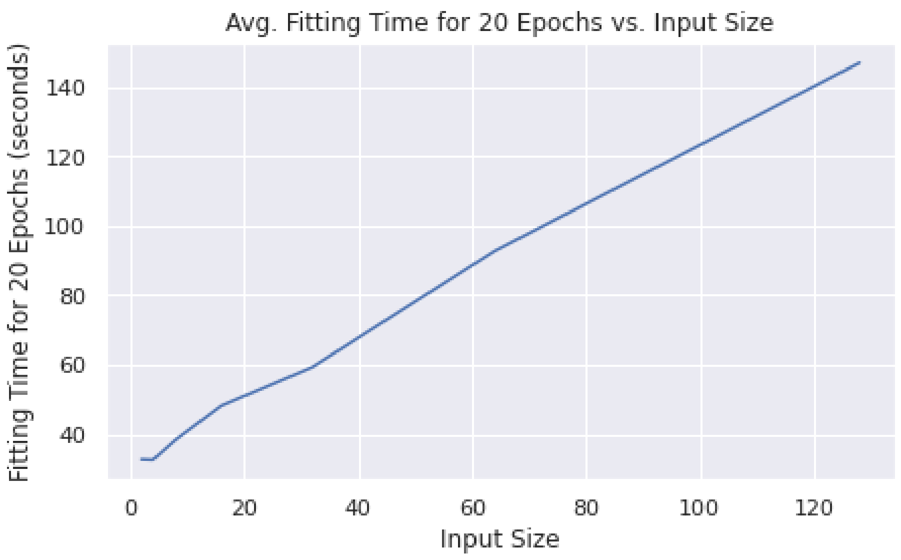


Figure 4-2 Average Fitting for 20 Epochs (seconds) vs. Input Size

Meanwhile, this test also gives us some valuable estimates on the model's performance. As mentioned previously in the model setup specifications, we gave this model a 7-class, instead of binary, classification task. It obviously further increases the task difficulty than what we actually need it to accomplish. The validation accuracy of this simple neural network model reaches a highest score of 92% on its last set of trials, and reaches a highest score of 100% at least once in each of the first three preceding trial sets. Given the small set of training samples in this test, its performance is not very stable until we increase the sample size to 64. When input size reaches 128, the average validation accuracy reaches 87%. This result looks nice, especially when taking the small input size, shrinked input length, and increased task difficulty into account. Another insight we get from this experiment is that using 128 as max input length might give a satisfying accuracy level in an easier binary classification task with a larger set of real data to learn from. So we might not need to set the max input length to 512 later in a real experiment, and that could save us more computational time and resources.

V. Goals and Next Steps

In the last few days of stage 1, we got access to the mentioned real dataset stored on AWS. Since we are not familiar with a variety of services provided by AWS for large data computing, fluently operating these data on SageMaker would be a foreseen challenge in stage 2. We will keep coordinating with the tech support from Amazon to sort this out.

Besides that, a high-priority task for the next step is to implement everything we presented above on the real dataset, including: an exploratory data analysis, class balancing by downsampling, train-test split, and applying our pre-processing pipeline to these selected samples. Ideally, the pre-processing results would be stored in S3 bucket for later re-use. Then we will re-create the stated RoBERTa fine-tuning pipeline using the HuggingFace package in SageMaker, and fine-tune it on real data in a binary classification task. This time, we will monitor the precision, recall, and F1 score in addition to accuracy throughout the fine-tuning process. We expect a qualifying RoBERTa encoder to reproduce its best performance reported in the GLUE tasks on our dataset, that is to get an average of ~95% in the metrics stated above.

After having the encoder trained, we will encode the articles and get their embeddings. When BERT's own paired tokenizer tokenizes the input articles, it sticks a special [CLS] token to the beginning of the sentence to mark the starting point of a document. In the end, when the fine-tuned BERT encodes a sentence, the embedding for this [CLS] token would represent the embedding of the entire input document. We will take this embedding as our input to other classification models. Four top modeling directions are neural network, tree based models, SVM, and regression-based models. We will investigate each model and study how the pros and cons (*Table 5-1*) of each model affects the overall model performance. After having detailed analysis on the above topics, we can then design a word shuffling algorithm for task 2, and send the shuffled articles to the same set of encoders and models to get the performance comparison.

Model	Pro	Con
Neural Network	● Easy to implement given the framework we have built when testing the runtime	● Heavy model ● Requires more computational resources
Tree-based Models	● High performance ● Make no assumptions of the data ● More interpretable if features' meaning are known	● Can overfit data ● Requires more computational resources
SVM	● Effective in high dimensional spaces.	● Might not converge on large dataset ● Might be difficult to find a good kernel function
Regression-based Models	● Easy to implement	● Performance depends on whether data is linearly separable

Table 5-1: Comparison between models

VI. Contribution

- **Hongling Liu**: Main contributor of research and literature review on BERT.
- **Haoyue Qi**: Main contributor of exploratory data analysis and class balancing method. Co-contributor of text data preprocessing.
- **Xuanhao Wu**: Main contributor of exploratory data analysis and visualization methods and performance metrics.
- **Yuxin Zhou**: Main contributor of preprocessing on sample dataset and researching on BERT and AWS S3\SageMaker.
- **Wenjie Zhu**: Main contributor of BERT pipeline implementation and runtime analysis, setting up weekly meetings and managing progress.

References

[1] E. Alzahrani and L. Jololian, “How Different Text-preprocessing Techniques Using The BERT Model Affect The Gender Profiling of Authors,” *arXiv.org*, 2021. https://arxiv.org/abs/2109.13890.
[2] J. Camacho-Collados and M. T. Pilehvar, “On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis,” *arXiv.org*, 2017. https://arxiv.org/abs/1707.01780.
[3] J. Ganhotra and S. Joshi, “Does Dialog Length matter for Next Response Selection task? An Empirical Study,” *arXiv.org*, 2021. https://arxiv.org/abs/2101.09647.
[4] C. Sun, X. Qiu, Y. Xu, and X. Huang, “How to Fine-Tune BERT for Text Classification?,” *arXiv.org*, 2019. https://arxiv.org/abs/1905.05583.