NLP Final Project: A Book Review Analysis with BERT
Group 3

Brian - Kuan-I Lu/ Bruce - Zixiao Jin/ Zhenghan-Fang/ Tyler- Chule Guan/ Joseph - Manho Yeung/ Daisy - Danxuan Zhao

Table of Contents

1. Abstract

In this project, our goal is to perform text analysis on amazon book reviews using NLP models, and perform a five-class classification predicting the review texts in one of the five ratings. The main approach of this project will be focusing on fine-tuning a BERT model, particularly a DistilBERT model. Based on comprehensive research, we made the decision to use DistilBERT as our base model due to its efficiency, balancing performance and the computational limitations of Google Colab. After fitting and fine-tuning the model, the final model we created achieved a decent performance, and we made multiple visualizations to assess the overall performance of the final model, including the metric and loss graphs and a confusion matrix heat map to display the result of a five-class classification task.

2. Introduction

The dataset used in the project is Amazon Books Reviews from Kaggle. This dataset includes 3 million reviews for more than 200,000 books from Amazon.com. Every review in the dataset has a rating of integers ranging from 1 to 5, which indicates how satisfied the reviewer is with the book.

The primary purpose of this dataset is to facilitate the development of NLP models that can understand and interpret user sentiments expressed in the review texts. By analyzing these reviews, models could gain insights into customer satisfaction and preferences, which can be valuable for tasks such as sentiment analysis and recommendation systems.

In this project, we aim to use the large dataset to build and improve an NLP model that can correctly infer the reviews' sentiments from their textual content. By evaluating the model’s performance, we hope to find out aspects of the training process, model architecture, and dataset that could affect the model's running time and accuracy. This comprehensive approach helps us gain a better understanding of NLP models and fine-tuning them for specific real-world tasks.

3. Related Work

Through our comprehensive research, we examined and compared the differences and characteristics among some of the most commonly used NLP models. From which we decided to work with BERT models out of all possible options. Then, knowing that Hugging Face contains numerous variations of BERt models, we assessed the pros and cons of each BERT model trying to decide on the version most suitable for our project. After the comparison, we explained the reason why we ended up deciding to use DistilBERT out of all possible models to tune from. To provide more details, we take a deeper look at the mechanics of DistilBERT and the methodologies used, along with mathematical explanation, to understand how DistilBERT works. Lastly, we explained the importance of freezing layers in the fine-tuning process, which we will be adapting such a method for this project.

3.1. Advatages of BERT Over Other Language Models

1. **Bidirectional Contextual Understanding**: Where other models may fall short, BERT excels. Unlike models such as GPT, which process text in a single direction (either left-to-right or right-to-left), BERT employs a bidirectional approach. This method allows BERT to fully capture the meaning of a word by considering its surrounding context, enabling a more nuanced understanding of entire sentences.
2. **Pre-training on Vast Datasets using MLM and NSP**: BERT is not just another language model; it's a powerhouse of language understanding, pre-trained on massive datasets. The introduction of Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) during its pre-training phase enables BERT to gain deep linguistic insights. This comprehensive pre-training, followed by fine-tuning on

specific tasks, makes BERT highly effective, even with limited task-specific data. Unlike older models that require extensive labeled datasets for each task, BERT leverages its pre-trained knowledge to excel in new challenges with minimal additional data.

3. **Contextual Embeddings**: Traditional word embeddings, such as word2vec or GloVe, treat words independently of their context. In contrast, BERT generates dynamic embeddings that vary depending on the word's context, enabling it to discern the nuanced meanings of polysemous words—something static embeddings cannot achieve.
4. **Scalability and Parallelism**: Unlike RNNs, which often face challenges in parallelization, BERT's transformer-based architecture allows for significant parallelism during training. This scalability makes BERT both efficient and fast, especially when handling large datasets.

3.2. Different Versions of BERTs

Since we know we're choosing BERT, we will show the characteristic, as well as the pros and cons of different BERT models to illustrate the reason why we choose DistilBERT over other BERT models:

1. **BERT Base**: With 110 million parameters and 12 layers, BERT Base is versatile for most NLP tasks like text classification and sentiment analysis. It balances performance and efficiency, making it the default choice for many projects.
2. **DistilBERT**: With only 6 layers, DistilBERT is a lightweight version of BERT, cutting parameters by 40% and boosts speed by 60%, while retaining 97% of BERT's accuracy. DistilBERT is Ideal for real-time applications on mobile devices and embedded systems.
3. **BERT Large**: With 345 million parameters and 24 layers, BERT Large offers better accuracy than BERT Base but requires more resources. Best for high-performance tasks like complex question-answering and large-scale text analysis.
4. **RoBERTa**: RoBERTa enhances BERT's pre-training, outperforming it in accuracy and large-scale classification tasks. It's used in applications like news recommendation and content review at Facebook.
5. **ALBERT**: ALBERT reduces parameters via weight sharing and embedding decomposition, maintaining performance in memory-constrained environments. Suitable for cloud applications and large-scale deployments.
6. **ConvBERT**: ConvBERT uses a convolution-based self-attention mechanism to improve efficiency and speed while maintaining accuracy. Ideal for fast-response tasks like real-time dialogue systems.

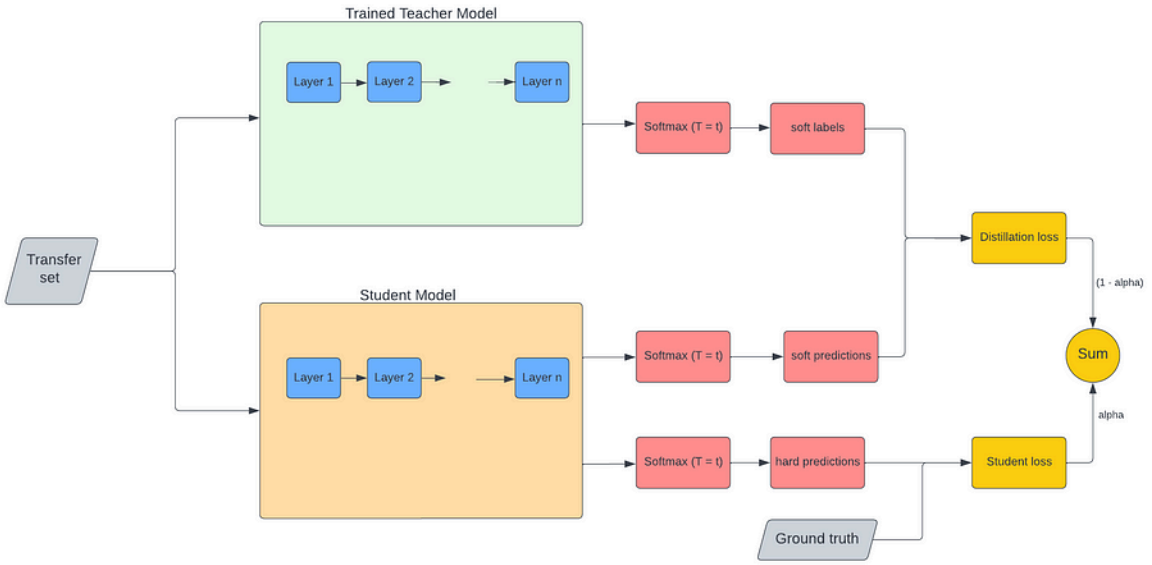
Notice that both DistilBERT and ALBERT are both light, meaning they have fewer parameters. However, we still prefer DistilBERT because, even though ALBERT does a great job of reducing the number of model parameters, DistilBERT is faster when making predictions. It also has a smaller model size and works better when computing resources are limited, which matches the scenario we're facing for this project. In short, DistilBERT is more manageable for students working on projects with their own computers.

3.3. How does DistillBERT work?

DistillBERT is introduced in the paper *"DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter"*. The requirements of big models such as BigBERT need high-performance computation, memory and time. Thus, it is not suitable for use on home computers or smartphones and other light weight devices. So, it is necessary to use DistillBERT, which reduces the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster.

DistillBERT introduced Knowledge Distillation. Knowledge Distillation Is a method where a large complex teacher model distills its knowledge and passes it to train a smaller student network to match the output.

The structure of Knowledge Distillation is as below:



(teacher model and student model)

The loss of the structure is **total loss = (alpha * Student model loss) + ((1 - alpha) * distillation loss),** which total loss is a combination of '**student model loss**' and '**distillation loss**'.

For further explanations,'student model loss' is the cross-entropy loss between the output of student model and ground truth (the label of data set), which 'Distillation loss' is the KL-divergence loss between the predictions of teacher model with t>1 and predictions of student model with t>1, where t is the temperature variable in SoftMax.

$$y_i(\mathbf{x}|t) = \frac{e^{\frac{z_i(\mathbf{x})}{t}}}{\sum_j e^{\frac{z_j(\mathbf{x})}{t}}}$$

(SoftMax)

Setting t>1 here is to amplify the difference between output from teacher model and student model because the probabilities are so close to zero if t=1. So, t>1 is better for calculating KL-divergence loss for distillation loss.

To Knowledge Distillation in BERT:

1. **Loss**: They used a linear combination of distill loss (Lce), Masked language modeling loss (Lmlm) and Cosine embedding loss (Lcos). They found that adding cosine embedding loss aligns the direction of teacher and student hidden vectors.
2. **Architecture**: DistilBert follows the same general architecture of Bert. The token type embedding, and pooler are removed while the number of layers gets reduced by a factor of 2. In modern frameworks, most of the operations are optimized and reducing the number of hidden dimensions has a small impact. So, in their architecture, the number of layers was the main determinant factor.
3. **Layer Initialization**: They initialized layers of students from the teacher by taking one layer out of two. Training and computation: For distillation training they used larger batches. Also, they used dynamic masking (Different masking pattern in different epochs) and without next sentence prediction objective. They trained Distil BERT on the same corpus as the original BERT model.
4. **Model Performance**:They compared the performance with Bert and ELMo encoder followed by two

Bi-LSTMs. Distil Bert retains 97% of Bert Performance on GLUE benchmark. The results are as follows:

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Also, Distil Bert is significantly smaller and is constantly faster. Number of parameters and inference time is shown below:

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

3.4. Freezing and Fine-Tuning

Training the entire model, meaning updating all the parameters in the BERT model to fit the training datasets, is computationally intensive and can sometimes lead to overfitting, especially when dealing with limited training datasets. A more efficient alternative is to freeze some of the earlier layers of the model, which are typically responsible for more general features, and fine-tune only the later layers, which are more specialized to different tasks.

Benefits of Freezing Layers:

1. **Reduce Computational Load**: The computational requirements for fine-tuning are greatly reduced by freezing a large portion of the BERT layers. Fine-tuning only the last few layers can save time and resources while maintaining high performance.
2. **Prevent Overfitting**: Fine-tuning fewer layers also lowers the risk of overfitting, especially on smaller training datasets. Since the early layers of BERT capture more general linguistic features, freezing them ensures that these features remain stable and do not adapt too closely to the training data.
3. **Faster Convergence**: By reducing the number of parameters that need to be updated during training, freezing layers can result in faster convergence. This simplifies the optimization process, resulting in fewer epochs needed to achieve optimal performance.

4. Approach

After setting the intention and explaining the decision to use DistilBERT as our base model, we move on to exploring the fine-tuning process and illustrate the overall workflow of the model, as well as the project workflow, to demonstrate how we approach this problem. First, we will take a comprehensive look at all fine-tuning methods and choose the one that suits our project the best. Then, we will show how the fine-tuning process works by visualizing it with a flowchart. Finally, we will explain the overall structure of this project, from data cleaning to model building.

4.1. Methods of Fine-tuning DistilBERT

BERT is designed for masked word prediction and next sentence prediction tasks. Although it can already be used for some domain-specific tasks, we need to tune the model for higher accuracy in a specific domain. There are several most common approaches for tuning:

1. **Full fine-tuning (or simply "fine-tuning"):** All model parameters are updated. Relevant subtypes of fine-tuning include:

    a. Transfer learning: Update the layers of a pre-trained model to adapt to a new task.
    b. Knowledge Distillation: Use a new typically smaller model called a "student" to learn representations from a larger "teacher" for a new task.

2. **Parameter-efficient fine-tuning:** Only a small fraction of model parameters is updated while other parameters are frozen, some examples include:

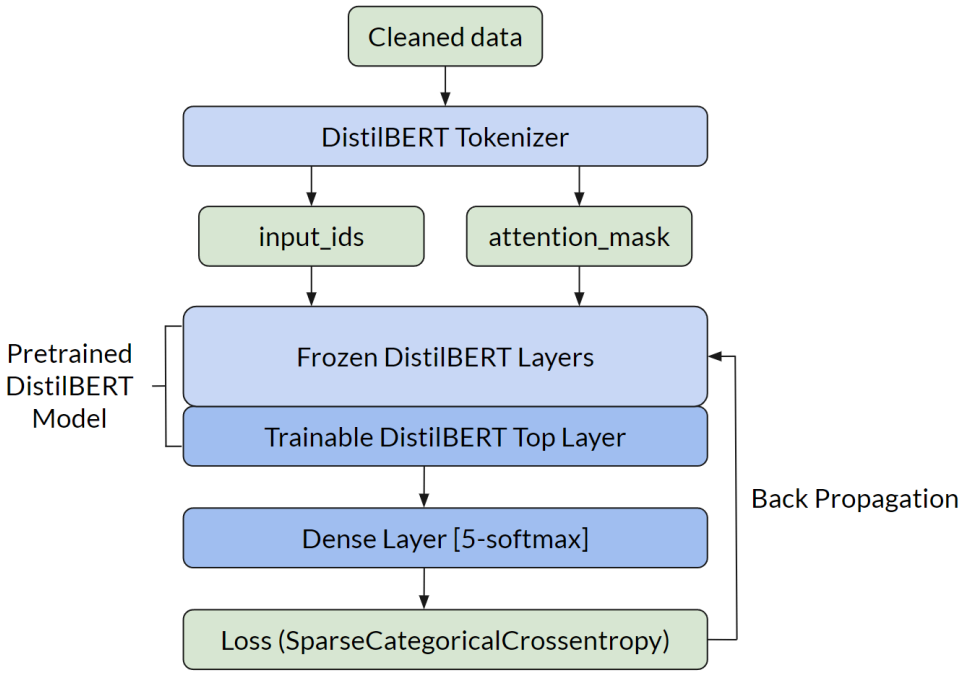
    a. Adapter-tuning: Inserts additional task-specific layers between the layers of pre-trained LLMs and only tunes parameters in the adapters.
    b. LoRA: Adaptors are low rank approximations of the original weight matrices (further reading) • Prefix-tuning: Prepends task-specific vectors to the model and only tunes parameters in the prefix.

3. **Instruction-tuning:** This is a fine-tuning method using supervised samples, specifically a collection of tasks phrased as “instructions”. All model parameters are updated during instruction-tuning. It substantially improves zero-shot performance on unseen tasks.
4. **Reinforcement learning through human feedback (RLHF):** This is essentially an extension of instruction-tuning, with more steps added after the instruction-tuning step. The focus of the subsequent steps is to ensure models are aligned with human preferences.

In this project, since we don’t want to change every parameter in DistilBERT due to the limitation of our computational power, we will use the second introduced method, parameter-efficient fine-tuning. We reduce computational burden by unfreezing the first of the 6 layers of DistilBERT and keep the rest 5 layers untrainable. Additionally, we add a fully connected Dense layer on top of the DistilBERT model to perform classification tasks.

4.2. Model Workflow Illustration

Now that we've justified our decision of using DistilBERT as our base model, unfreezing only the top layer, and adding a fully connected dense layer. We want to show the workflow of our model to better illustrate the tuning mechanism:



Following the flowchart provided above, a previously cleaned data will be passed into the pre-defined DistilBERT Tokenizer, outputting a dictionary with two keys, input_ids and attention mask, to be ready to pass into the DistilBERT model. The output of the DistilBERT model moves on and gets passed into a fully connected dense layer with softmax as its activation function. Then, we calculate the loss, SparseCategoricalCrossentropy, and use back propagation to change the parameters. Notice that only the added dense layer and the top unfrozen layer of the DistilBERT model is trainable.

4.3. Project Workflow

The general steps to fine-tune DistilBERT for our tasks are as follows:

1. Load a pre-trained BERT model (distillBERT) and tokenizer (DistilBertTokenizer) from the Transformers library.
2. Load task-specific data set, pre-process and tokenize the data with tokenizer.
3. Define the model for the task by updating the last layer of the model for training.
4. Define the training hyperparameters and optimizer.
5. Train the model on the training data and evaluate the model on the validation data.
6. Save the fine-tuned model and use it for inference on the new data.

5. Experimental Results

5.1. Load Packages and Compare Base Models

In this project, we used NumPy and pandas for data manipulation, and matplotlib.pyplot for data visualization. For setting up and working with neural network models, we used the TensorFlow library. Additionally, since we will be tuning the pretrained BERT models, we also imported these models and tokenizers from the transformers package created by Hugging Face. Lastly, to perform train/test split, and to extract a subset of the entire dataset, we also imported the train_test_split function from sklearn.

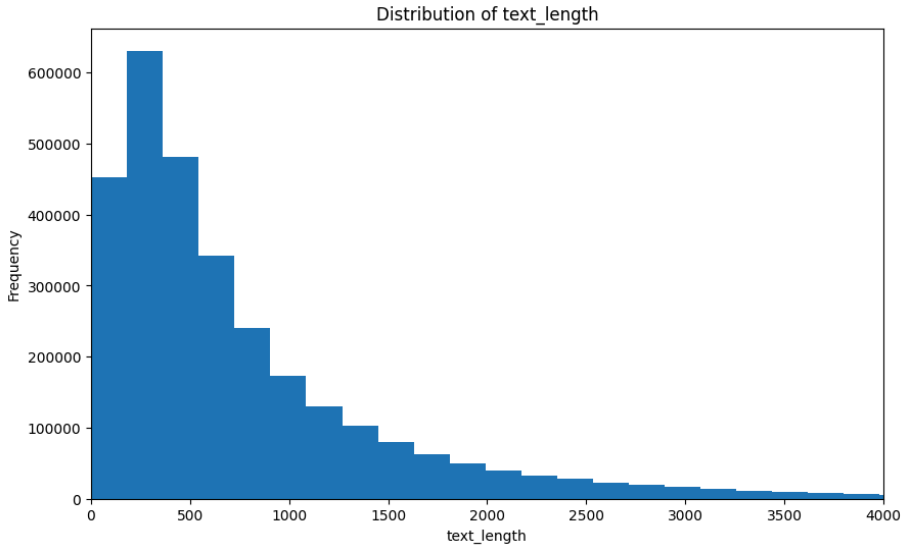
Additionally, to prove that DistilBERT is indeed parameter-efficient, we loaded both the DistilBERT model and the BigBERT model to examine the difference in the amount of parameters. From the results, we can see that DistilBERT has about 66 million parameters, which is already a significant amount. On the other hand, BigBERT has over 330 million parameters, which is around five times the amount of DistilBERT. By having substantially

more parameters in the model, our decision to base our model on DistilBERT, which is computationally more friendly, is justified given the environment and computational power we have access to.

5.2. Load and Clean the Dataset

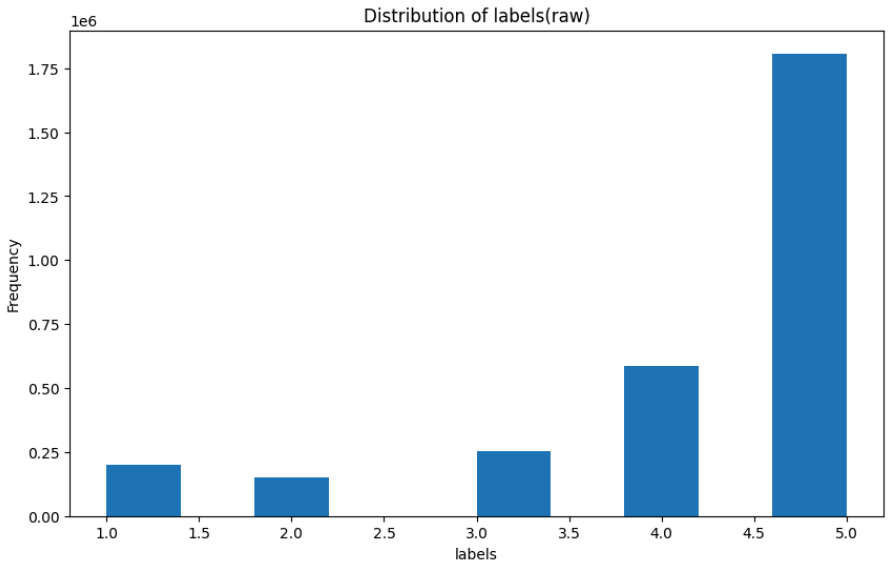
We loaded our dataset, Books_rating.csv, which contains comprehensive information on over 200,000 books with 3,000,000 reviews. For this project, we extracted only the text reviews and the corresponding scores to work with. The goal is to input the text reviews into the model, have the model analyze the text, and output the class of rating (integers ranging from 1 to 5) to which the text review belongs. After loading the dataset and extracting the columns useful for this project, we checked the data in terms of missing values and found that we have 8 observations that, for some reason, contain NaN values. Since the number of observations with missing values is trivial, we removed the rows with NaN values and continued cleaning.

Next, we examined the distribution of sentence lengths. In model building, we need to define the maximum sequence length to work with. By analyzing the distribution of sequence lengths, we can decide on a reasonable value for our maximum sequence length. It's important to consider not only the distribution to ensure that most sequences are complete but also the maximum sequence length that our environment and computational resources can handle. For example, if all sequences have lengths longer than 10,000 words, but setting the max_length to 10,000 would crash our environment, we will have to compromise and choose a smaller max_length. The goal is to optimize model performance while ensuring technical feasibility.

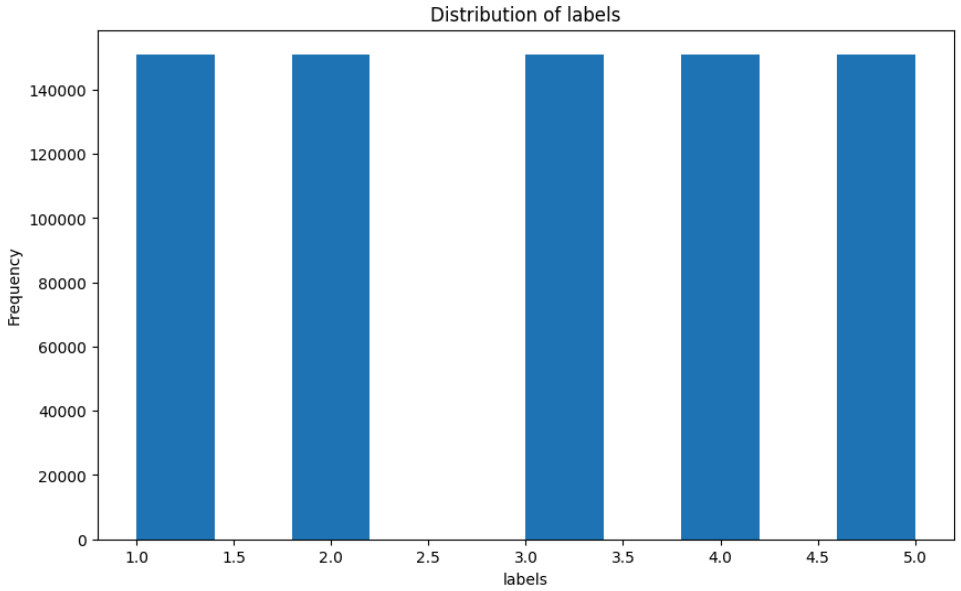


From the distribution of text lengths, we can see that most reviews are rather short, falling under 1,000 words. A reasonable maximum length would be somewhere between 1,000 and 2,000, depending on the completeness of sequences we want to work with for our model. However, after some trial and error, to make the best use of our available environment and to prevent excessive computational cost, we ended up limiting our sequence length to 800, which still covers a lot of our sequences completely.

Next, we examined the distribution of review scores (i.e., our outcome variable) to see if the distribution is ideal or imbalanced in any way.



Notice that the distribution of our outcome variable is extremely imbalanced, with a score of 5 being significantly more frequent than all the other classes. The next largest class, 4, has only approximately one-third of the amount of observations as class 5. Additionally, the smallest class, 2, has less than 10 percent of the observations of the largest class. A reasonable way to mitigate the effect of imbalanced outcome classes is to upsample or downsample the dataset. Since we have a large dataset with 3,000,000 observations, we downsampled our data to even out the outcome distribution.



After we downsampled our data, we observed that the distribution of our outcome variable is even. Additionally, despite the downsampling, we still have over 750,000 samples remaining, which is sufficient for the purposes of this project. In fact, the remaining 750,000 observations are more than we can handle. Initially, we used all 750,000 observations to fine-tune the DistilBERT model, but since the model is still complex and contains many parameters to train, using all of the observations is taking too much time and computational resources. Therefore, after some trial and error, we eventually decided to reduce our data to a subset of 10,000 observations, stratified by our outcome variable (review/score) to ensure the distribution of the outcome variable remains consistent. After creating a smaller dataset to work with, we can start to split and clean our data. Note that the train/test split is also stratified by the outcome variable to ensure balanced classifications.

Moving on, we want to clean the review texts by separating sentences based on delimiters and removing all

non-alphanumeric characters. Using the previously loaded tokenizer, DistilBertTokenizer, from the transformers package, we tokenized the cleaned data, setting the max_length to the previously defined 800 tokens.
As for the outcome labels, we created a dictionary mapping the five labels to indices. This step is crucial since later on, our model demands that the classes of our outcome variable be indices instead of labels of any type (int, str, etc.).

5.3. Selecting the Hyperparameters

When fine-tuning an NLP model using the functional API, we can select various parameters, from the embedding size of each layer to the number of epochs we want to train our model. In this case, the embedding size is determined by our base model, DistilBERT, so there is no need to define the embedding size for this model. Regarding the learning rate, since we are using Adam as our optimizer, which updates the learning rate for each batch, the initial learning rate value is less critical. Here, we chose a commonly used value based on previous work: **2e-05**. Due to limited computational power, we must select a smaller batch size, where larger sizes are preferable if the computation is manageable. In this case, we used a batch size of **8**, which is also a result of some trial and error, primarily from the environment crashing due to extensive memory use for each batch. As for epochs, since we are working with a large dataset where the model usually converges after only a few epochs when fine-tuning, we set it to **5** and apply an early stopping callback in case the model converges earlier than expected.

5.4. Construct and Train the Model

Based on the previously mentioned research, the most efficient way to train a BERT model is to unfreeze (make trainable) only the top layer and keep all other layers frozen. We implemented this exact method in our training process. In addition to unfreezing the top layer, we also added another dense layer that takes the class embedding, which the DistilBERT model outputs, as input. We used softmax as our activation function for our dense layer with an embedding size of 5, corresponding to the number of classifications for our outcome variable. This dense layer helps us perform the multi-classification task by producing five probabilities, which, when applied with an argmax function, determine the class to which the input text belongs. Using the parameters defined in Section 5.3 and the data cleaned in Section 5.2, we built and trained the model.

In terms of the loss function and metrics, we used accuracy as our metric to monitor the model performance, and we used sparse categorical cross entropy as our loss function.

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 attention_mask (InputLayer  [(None, 800)]             0         []
 )

 input_ids (InputLayer)      [(None, 800)]             0         []

 DistilBER_Pretrained (TFDi  TFBaseModelOutput(last_hid  66362880  ['attention_mask[0][0]',
 stilBertModel)              den_state=(None, 800, 768)            'input_ids[0][0]']
                             , hidden_states=None, atte
                             ntions=None)

 tf.__operators__.getitem (  (None, 768)               0         ['DistilBER_Pretrained[0][0]']
 SlicingOpLambda)

 dense (Dense)               (None, 5)                 3845      ['tf.__operators__.getitem[0][
                                                                 0]']

=================================================================================================
Total params: 66366725 (253.17 MB)
Trainable params: 30927365 (117.98 MB)
Non-trainable params: 35439360 (135.19 MB)
_________________________________________________________________________________________________
```

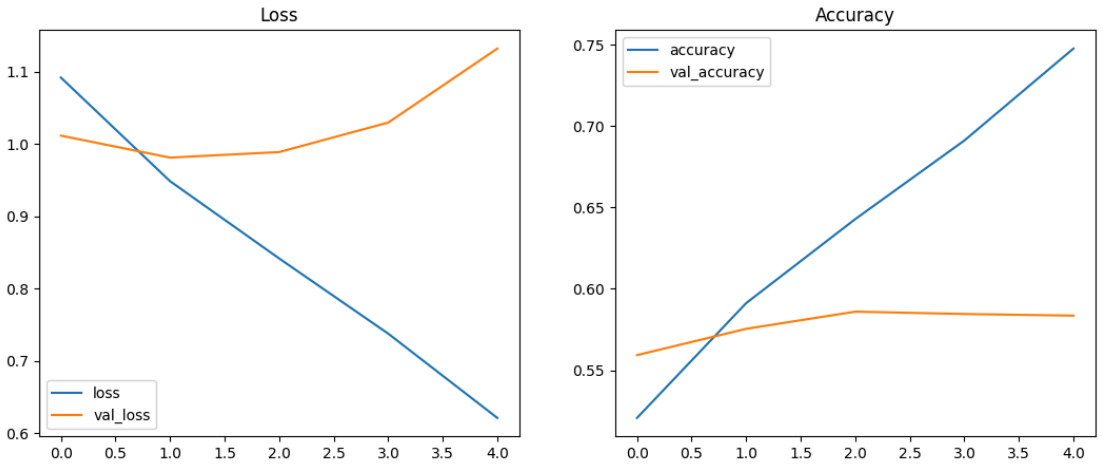
In our model summary, we can see that even after freezing all layers except the first one of our DistilBERT, we still have 30,000,000 parameters to train. This is still a significant number of parameters, which could potentially

lead to overfitting our training data.

5.5. Assess Model Results

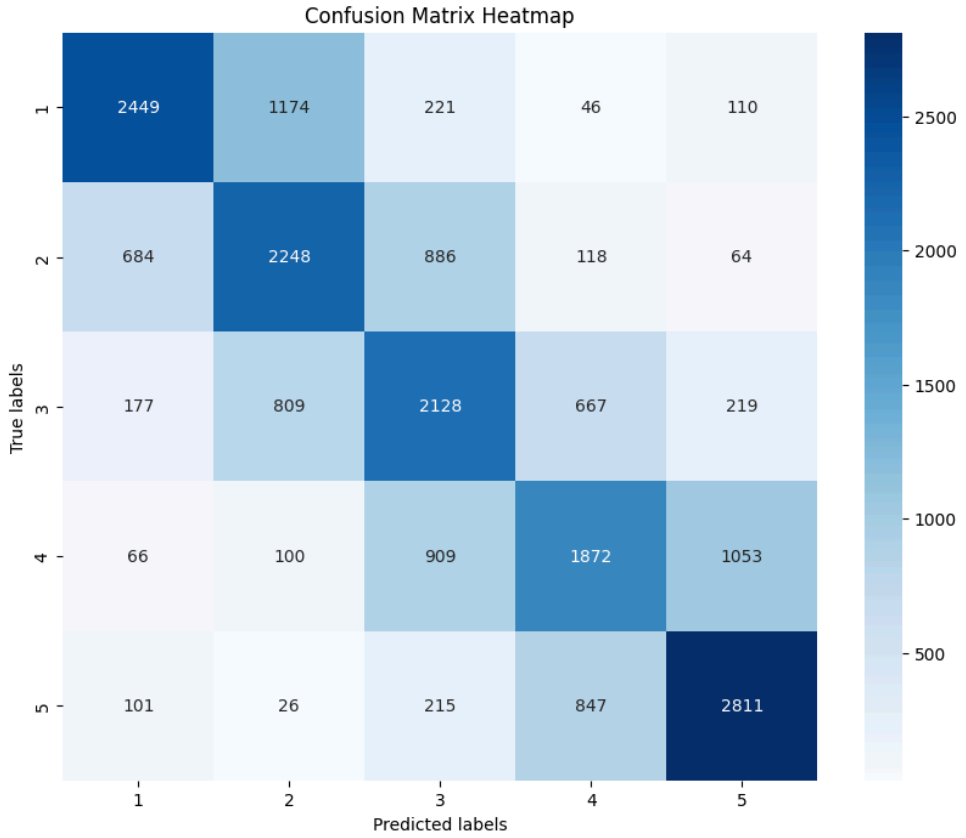
The model fitting process took more than 7 hours to run, with 5 epochs and restoration of the best epoch in terms of the loss function, which was the second epoch. During the model fitting process, the training accuracy increased steadily, reaching 0.74 in the last epoch, whereas the testing accuracy peaked in the second epoch at 0.586 and remained at the same level for the remaining epochs. This is a clear sign of overfitting, so we defined the model with the lowest testing loss as our final model.

Below is a graph of the loss and accuracy for both the testing and training data:



From the loss graph, we can see that the training loss decreased gradually with each epoch. However, our testing loss reached its lowest point at the second epoch and started increasing with each subsequent epoch. This is a clear sign of overfitting, likely due to the complexity of the model and the insufficiency of the training data to create a model that generalizes well enough to determine the rating based on the text.

Regarding accuracy, the testing accuracy remains around 0.58, which, given the nature of the dataset, can be considered a decent performance, though it clearly shows room for improvement. A good way to assess whether this accuracy indicates a well-performing model could be to have humans try to perform the same task on a large dataset and compare their performance with that of the model. While binary classification might be simple, a five-class classification task like this can be challenging to achieve remarkably high accuracy.



Here is the confusion matrix heatmap demonstrating our best model's prediction performance. By looking at the confusion matrix heat map, we can see that our model is particularly good at predicting texts with scores of 1 and 5, reaching accuracies of 0.704 and 0.660, respectively. However, the predictions for the remaining classes (2, 3, and 4) have approximately the same chance of being correct or predicting the neighboring classes. For example, when the model tries to predict class 3, it still makes most of its predictions correctly on class 3 but tends to make errors by mistaking these class 3 texts as class 2 or class 4. Overall, the confusion matrix still has significantly darker cells along the diagonal, showing traits of a well-performing prediction model.

```
Label: 1 , Text: Dont bother with this one .  As a huge GLH fan I couldnt even make it through the first half of this book .
1/1 [==============================] - 0s 165ms/step
Prediction: 1
Label: 4 , Text: As its title promises this book is in fact very well illustrated with hundreds of color as well as black an
1/1 [==============================] - 0s 122ms/step
Prediction: 4
Label: 2 , Text: The Vampire Lestat reads as a very slow philosophical autobiography .  Ive read lots of 1000 novels that mo
1/1 [==============================] - 0s 119ms/step
Prediction: 2
Label: 1 , Text: I read both quotRed Dragonquot and quotSilence of the Lambsquot and very much enjoyed them .  This book I t
1/1 [==============================] - 0s 120ms/step
Prediction: 2
Label: 3 , Text: An OK read but seems to get bogged down in the middle .  However if youre into Jeal Auels seriesyoull want
1/1 [==============================] - 0s 121ms/step
Prediction: 3
```

Finally, we randomly selected five sample texts from our testing data. We reviewed the model's predictions as well as the correct classes these texts belong to. From the results, we can see that the model performs well at predicting the rating score of a book based on the texts. Even when the model misses the correct answer, the prediction is still close to the actual value.

6. Conclusion

In this project, we were tasked with text analysis and classification prediction using NLP models. From our knowledge, the most powerful approach is fine-tuning a BERT model to perform the classification task. The reason for choosing BERT over numerous other language models is based on comparisons and research on all the known language models. However, even after deciding to approach our project by fine-tuning a BERT model, there are still multiple BERT models in the transformers package to choose from, each designed for different targeted tasks with various pros and cons. Therefore, to prevent wasting time on excessive trial and error, comprehensive research and comparison were necessary before starting the computation of the project.

We referred to multiple authoritative sources to conclude that, given the scope and tools available for this project, the best BERT model to base our work on is the DistilBERT model. Compared to BERT Base, DistilBERT has 40% fewer parameters but loses only 3% of the capability. Since we're running our project on Google Colab with limited RAM and GPU resources, using DistilBERT as the base of this project became an ideal method. In our research, we also discovered that the most efficient way to fine-tune a DistilBERT is by unfreezing the top of the 6 layers, leaving the remaining layers untrainable. By doing this, we can avoid altering all parameters, which is computationally expensive, while still reshaping the model to some extent. This approach prevents the added dense layer from being the only part modified according to our training data, which could over-generalize the data patterns and yield poor performances.

In the data cleaning process, we first determined the appropriate maximum sequence length we wanted to work with based on the distribution of sequence lengths of our input texts. We then examined the distribution of the outcome variable and found it severely imbalanced. Consequently, we decided to downsample the data to have the same number of observations for each class of the rating scores. Throughout the initial attempts, we discovered that our environment could not handle the large amount of data we had. In response, we decided to work with a stratified subset of the original data, allowing us to successfully train and fit our model.

Regarding model result and performance, the final model achieved an accuracy of 0.58 and a sparse categorical cross-entropy of 0.98 on our testing data. Given the nature of this task, which is five-class classification, a result like this is acceptable and can be considered quite good. To take a deeper look at the model's prediction performance, we also drew a confusion matrix heat map and found that our model performs really well at predicting classes 1 and 5, but sometimes mistakenly predicts neighboring classes for all classes.

Evidently, there is some room for improvement in this project. First, we clearly overfit the model, which could be due to having a too-complicated model for our data, as well as the scarcity of the data itself. It is important to note that we compromised by not using all the available data to train our model. If we had a more steady and robust method of fitting the model, we could have worked with more of our data and potentially exceeded the current model's performance. Also, note that the accuracy of the best-performing model significantly increased when we made the first layer of the DistilBERT model trainable, rather than only training the added dense layer. If we could handle the computational burden, unfreezing all the layers with an adequate amount of data would likely outperform the model we developed in this project. With more computational power, we could also use BERT Base or even BigBERT to achieve highly accurate predictions.

In summary, this project has been a valuable experience in learning how fine-tuning an NLP model works and understanding the trade-offs involved. Throughout the project, we constantly had to make decisions that balanced computational feasibility and model performance. The result we produced can be considered a well-balanced combination of the two. In future studies and projects, we are excited to try out more methods, perhaps with more computational power at hand, to fully reach the potential of not just BERT, but any large language models we work with in the future.

7. References

1. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *in Proc. of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Workshop*, 2019, pp. 1-6.
2. G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

3. A. Mallya, "Understanding DistilBERT in depth," Medium, Aug. 23, 2020. [Online]. Available: https://arunm8489.medium.com/understanding-distil-bert-in-depth-5f2ca92cf1ed. [Accessed: Aug. 20, 2024].
4. O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky, "Revealing the Dark Secrets of BERT," arXiv preprint arXiv:1908.08593, 2019.
5. J. Lee, R. Tang, and J. Lin, "What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning," *arXiv preprint arXiv:1911.03090*, 2019.