Recapitulate: Another Corporate Tool

Jared Deiner, Hung Hua, Jacob Morello, and Tran Ton
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332
{jdeiner3,hhua7,jmorello3,tton8}@gatech.edu

Abstract

We aim to build an application that processes and analyzes corporate IT tickets. The tool will enable teams and management to quickly perform a deep assessment of how tickets are being handled by providing users with a summary of the ticket contents as well as the attitudes and opinions of the customers towards how the ticket was handled. In this paper, we demonstrate implementations of sentiment analysis and text summarization using Natural Language Processing (NLP) techniques. While the work in this paper is not scoped to corporate/IT tickets, the work detailed in this report is a necessary precursor to our overall end goal.

1. Introduction

IT ticket handling plays an integral role in operations of businesses handling customer issues. Inefficient ticket handling could lead to service downtime and eventually impacts customers as well as business profitably. Until now, companies only have basic dashboard tools that show generic statistics such as the numbers of open and closed tickets. However, there are no tools that provide enough insights into how well the tickets are being handle and what are the issues if they are not. Therefore, we aim to build tool that enables non-technical personal, such as managerial staff, to quickly obtain an understanding of developer-customer interactions in a ticketing system. This requires the tool to provide a distillation of the content as well as a characterization of how the interaction went.

1.1. Background

Models have been developed separately for text summarization and sentiment analysis, but no existing system combines both.

We also searched for open-source IT ticket datasets that contain both summary and sentiment labels, but none was available. We believe this is due to the fact that many tickets likely contain proprietary information that cannot be released publicly. Therefore, we opted to use non-IT ticket related datasets to build our tool and achieve the task of text summarization and sentiment analysis. For text summarization task, we train the model with the CNN/DailyMail dataset which includes full articles to be used as the text to be summarized and a few highlight sentences to be used to choose the target summary. For sentiment analysis, we use IMDb reviews and Amazon product reviews datasets, where writers review text is processed as input to our models which sentiment is extracted from, and the ratings are used as ground truth labels for training (high rating is positive and low rating is negative).

Throughout the project, we used $pytorch$ and $huggingface$ transformers.

2. Approach

2.1. Sentiment Analysis

Sentiment analysis pertains to analyzing, interpreting, and ultimately classifying the attitude of a sentence or set of sentences; The most common set of interpret-able labels used for classification are positive, negative, or neutral. An overarching objective of Another Corporate Tool is to be able properly classify the sentiment of conversations between engineers/IT and customers with hope of using the sentiment data to help find interesting relationships between things such as language and turn-around time for fixing a customer issue. Before we can begin to ask questions such as, "What is the optimal type of language to use when interacting with support engineer so that my problem gets solved fastest?", we first need to be able to answer questions such as, "What does a positive sentiment look like?" - and this answer turns out to be objectively hard due to the subjective nature of language.

2.2. Sentiment Analysis Models

We chose the BERT transformer model, more specifically the BertForSequenceClassification model, as the foundation for our sentiment analyzer based on its authors claim that pre-trained BERT models can be fine tuned with the

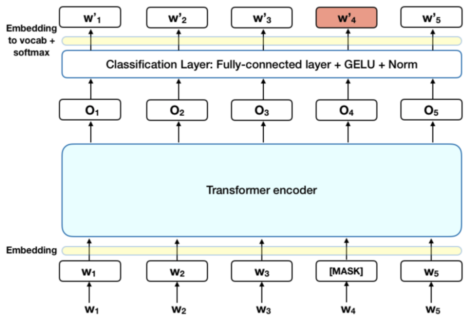
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
Transformer encoder
Embedding
[MASK]

Figure 1. BertForSequenceClassification architecture. Image taken from [1]

simple addition of just one extra output layer to ”create state-of-the-art models for a wide range of tasks”. The pre-trained BERT model out of the box is already trained by google using Wikipedia and Book corpus, which essentially means the model already has a wide range of vocabulary. To help fill in some knowledge gaps in the BERT models vocab we used a pre-trained BERT tokenizer which will break up unknown words into their known roots - an example of this being the word ‘breaking’ being tokenized into ‘break’ and ‘##ing’. The pre-trained BERT tokenizer we used is also in charge of encoding the input into a format readable by our model, taking care of assigning attention masks as well.

The next step to designing our models architecture was based on the choice of whether to do binary single-label classification i.e. solely classifying a sentence/set of sentences as positive or negative, or to do multi-class single-label classification i.e. classifying a sentence/set of sentences based on say a 1-5 star review rating. As you will see in the experimentation section, we chose all of the above, using a subset of Amazon product review datasets that could full fill both these classification problems, as well as provide data for labeling summarizing tasks as well. At this point we have covered that we were going to use a pre-trained BERT tokenizer to encode Amazon product reviews as input to be passed into a pre-trained BERT model - from here we experimented with extending the BERT model to have an extra dropout layer followed by a fully connected layer with outputs equal to the number of labels we wanted to classify on.

2.3. Extractive Summarization

A document $D = (s_1, s_2, \dots, s_n)$ is a finite sequence of sentences s_i, where each sentence $s_i = (w_1^{(i)}, w_2^{(i)}, \dots, w_{n_i}^{(i)})$ is a finite sequence of words. An *extractive summary* of D is a non-empty finite subsequence $D_S = (s_{i_1}, s_{i_2}, \dots, s_{i_k})$ of D, where $i_\ell < i_p \leq n$ for $\ell < p$. Thus, there are $2^n - 1$ extractive summaries of a document that contains n sentences (we do not consider the

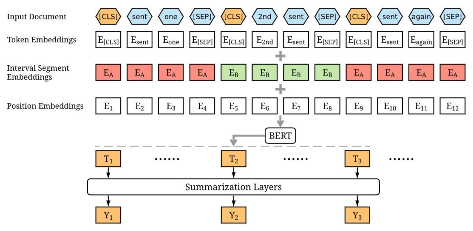

Figure 2. BERTSUM architecture. Image taken from [20].

empty summary to be an extractive summary). The goal of the *extractive summarization* problem is to find an extractive summary D_S of D such that (a) i_k is minimal in some meaningful sense, and (b) the semantic similarity between D and D_S is maximal in some meaningful sense. The prior two requirements are vague, so in order to make this a more concrete problem, we reduce these to (a) i_k is bounded above by some quantity less than n that depends on the application, and (b) maximizing the ROUGE-N (ROUGE-L, resp.) scores, where ROUGE-N (ROUGE-L, resp.) measures similarity by comparing the number of overlaps of N-grams (longest common subsequence of words, resp.) between the extractive summary and the ground truth summary (often called the *golden summary*).

2.4. Extractive Summarization Models

2.4.1 BertSum

In this project we considered BertSum [19, 22] as one of our potential models to produce extractive summaries. The BertSum model, in brief, feeds an input document D (which is initially preprocessed, to be explained) through a pre-trained BERT model and feeds the output of the final encoding layer of the BERT model into a summarization layer, which then finally produces the extractive summaries. See Figure 2 for an illustration of the architecture from [20].

To make the prior description more precise: For an input document D, we first preprocess it by prepending and appending a [CLS] and a [SEP] token, respectively, to each sentence s_i to get the tokenized sentence $([\text{CLS}], w_1^{(i)}, w_2^{(i)}, \dots, w_{n_i}^{(i)}, [\text{SEP}])$. The [CLS] token is often prepended once to the beginning of the document in other models, but for BertSum it’s prepended to *each* sentence in order to have a token that can be used to represent the sentence embedding after going the final encoding layer of the BERT model. The [SEP] token is, of course, used to separate the sentences of the document.

The tokenized sentence is then transformed via a map f to get the input embeddings $s_i' = (f([\text{CLS}]), f(w_1^{(i)}), f(w_2^{(i)}), \dots, f(w_{n_i}^{(i)}), f([\text{SEP}]))$, where for a token w,

$$f(w) = \text{tok\_embed}(w) + \text{seg\_emb}(w) + \text{pos\_emb}(w) \tag{1}$$

where token_embed represents the token embedding, seg_emb represents the segment embedding, and pos_emb represents the positional embedding; these values were randomly initialized in the beginning of training, but was then fine-tuned. It is noted here that to distinguish sentences an interval segment embedding is employed, where there are two segment embeddings, say E_A and E_B, and seg_emb(w) is assigned to E_A if w comes from an odd-indexed sentence, and is assigned E_B if it comes from an even-indexed sentence; that is, the embeddings are assigned in an alternating fashion according to the sentence position in the document.

The preprocessed document $D' = (s'_1, s'_2, \ldots, s'_n)$ is then fed into the pretrained BERT model to obtain the embeddings $B(D') = (B(s'_1), B(s'_2), \ldots, B(s'_n))$ provided by the final encoding layer of the BERT model (we use B to denote the BERT embedding), where[1]

$$B(s'_i) = (B([\text{CLS}]), B(w_1^{(i)}), \ldots, B(w_{n_i}^{(i)}), B([\text{SEP}])) \tag{2}$$

Then, before going into the summarization layers we must decide on the embedding to represent each sentence. In BertSum, this sentence embedding is notated as T_i for sentence s_i, and it is defined as

$$\begin{aligned} T_i &= \pi_1 B(s'_i) & (3) \\ &= B([\text{CLS}]) & (4) \end{aligned}$$

where π_1 extracts the first component of the sequence.

We note here that BERT provides *contextual embeddings* (thanks to the self-attention mechanism), so $B([\text{CLS}])$ provides features associated with the entire sentence, and not just the [CLS] token itself. Also, it need not be the case that $T_i = T_j$ for $i \neq j$ even though notationally they appear equal (the more cluttered, but explicit, notation is $T_i = B([\text{CLS}], i)$ to associated with sentence s_i). Finally, the sentence embeddings $T_1, T_2, \ldots, T_n$ are then fed to the summarization layers to get the respective predicted scores $\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_n$. The sentences with the top scores are chosen as the extractive summaries (specifically for BertSum, 3 sentences are chosen).

There were three types of summarization layers that were used and measured: (a) Simple Classifier, (b) Transformer, and (c) LSTM.

[1]as these are contextual embeddings, we should've made explicit the dependence on the sentence it belongs to, such as $B([\text{CLS}], i)$ for sentence s_i, but we avoid this to reduce notational clutter.

For the simple classifier, a single linear layer with a sigmoid was used to generate $\hat{Y}_i$:

$$\hat{Y}_i = \sigma(W_0 T_i + b_0) \tag{5}$$

For the transformer, the output of the final encoding layer h^L (L denotes the number of layers, so h^L is the final one) of the transformer was used with a sigmoid classifier; hence for sentence s_i, the result is

$$\hat{Y}_i = \sigma(W_0 h_i^L + b_0) \tag{6}$$

It is noted that the sentence embeddings T_i still go through a positional embedding before going in the encoding layers of the transformer. Also, the decoder component of the transformer was not used.

Finally, for the model with the LSTM summarization layer, the sentence embeddings are sent to the the LSTM cell, and the cell provides at its output the vector h_i, and this is, like the other two cases, fed to a sigmoid classifier:

$$\hat{Y}_i = \sigma(W_0 h_i + b_0) \tag{7}$$

The summarization layers described above are jointly fine-tuned with BERT, and the loss used for the entire model is binary classification entropy between $\hat{Y}_i$ and the gold label Y_i [20].

2.4.2 Sentence.

The second approach we attempted was *Sentence.* [18,23], which compares document and sentence embeddings to determine if the sentence is part of the document's summary. We found this system promising because it uses a sentence transformer [25] at its core, which aligns with our sentence-level task.

We used the CNN/DailyMail dataset for training and evaluation, which includes a full article, as well as a few corresponding "highlight" sentences. These highlights aren't explicitly drawn from the article, so processing must be performed to construct pseudo-ground truth for the dataset. This is done by sentence segmentation of the articles and selecting the sentences that are sufficiently close to at least one sentence in the highlight based on a Rouge score threshold. The sentence segmentation is performed using a statistical language model from *spacy*. We borrowed the core of this from *Sentence.*, but refactored it to increase performance and portability.

The next step is preparing the sentences and documents for input to the encoder. We stayed with the default encoder that *Sentence.* uses, a small version of SBERT. First, both the document and a candidate sentences are tokenized, and each token is mapped to a numerical id that the transformer supports. Then, they are passed to the encoder to produce a

384-dimensional embedding. In $Sentence.$, these are multiplied together element-wise, and the resulting vector is concatenated with the two input embeddings to form the combined embedding. Finally, that embedding is passed to linear layers for binary classification. As with the pre-processing code, we borrow the core from $Sentence.$, but modified it to support configuration of size and number of linear layers and how to combine embeddings.

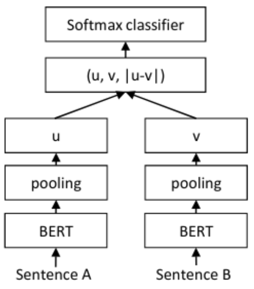

Figure 3. During training, SBERT combines the BERT embeddings of two sentences (figure source: [25]).

The design for this architecture is likely motivated by the structure of SBERT itself: as Figure 3 shows, SBERT is trained by combining the BERT embeddings of two input sentences, and passing the combined embedding through linear layers to a classifier.

A notable modification that we made was to change the element-wise multiplication to an addition. The element-wise multiplication between the embeddings might be understood as providing some similarity information to the network, since it only needs a summation to become an unnormalized cosine similarity. However, in practice it didn't perform very well. The justification for addition instead is based on intuition from word2vec, in which linear combinations of vectors can have interpretable semantic meaning. The idea here is that the linear combination of the document and sentence embedding produces a vector that's close to the "true" summary for that document in semantic space. Ultimately, this modification yielded the best results, as will be discussed below.

3. Experiments and Results

3.1. Sentiment

In our experiments using BertForSequenceClassification we trained our model using two differing classification tasks: binary single label classification and multi-class single label classification - each type of classification task having their own problems.

We primarily trained our model on a subset of Amazon customer product review datasets. [2] related to digital music and video games. We begun our experimentation by pre-processing these datasets and extracting only the relevant features within the context of sentiment analysis: for each sample, we extracted its review text, to be encoded and passed as input to our model, and overall rating, to be formatted and used as our ground truth label. The overall rating, used as our label, could take on any integer values 1,5

3.2. Multi-Class Single Label

In order to properly classify against using 5 possible outputs, we converted the labels for each sample into a length 5 one-hot encoded vector where the index of the ground truth value was set to 1. We then added an additional output layer to the model of size 5 and took the softmax of the output across the 5 outputs returned by the model as its prediction. We used CCE (Categorical Cross Entropy) as our loss function since BCE would be restricted as samples belonging to one class do in fact inform the model about how likely they are to belong to another.

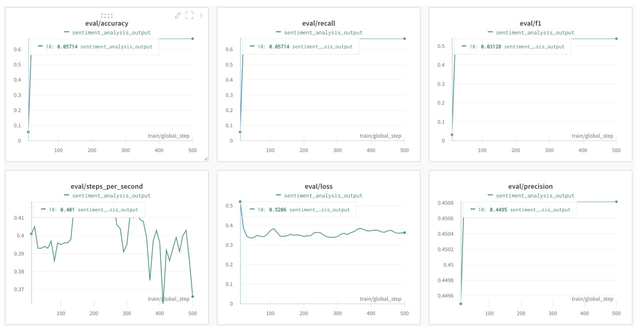

Figure 4. Multi-class model - Accuracy, Recall, F1, Precision Loss curves during training with a smaller learning rate of 1e-05, a in-balanced training set biased towards 5 star reviews, with mini-batches of size 2, over 500 steps.

In Figure 4, you can see that the models loss began oscillating while stagnant at 67 percent accuracy - this is likely due to the model having trouble differentiating between 4 star and 5 star reviews in a dataset that was already imbalanced in favor of 5 star reviews. We believe that with more data, or by better balancing the dataset instead of randomly sampling from it, we could eventually train the model to make this distinction. We chose to ultimately choose binary classification to be the better approach, relying on task dependent pre-processing of labels to split them into positive/negative sentiments, since we constantly had the issue of the model differentiating between similar star reviews.

3.3. Binary Single Label

After observing the results in the multi-class single label, we decided to try binary classification. To properly train a model for the binary classification task we first needed to sanitize the labels with values of 1-5 into either positive or negative ground truth values. Using our own domain knowledge we decided to set any review between 1-3 stars as negative and 4-5 stars as positive. Through figure above, we

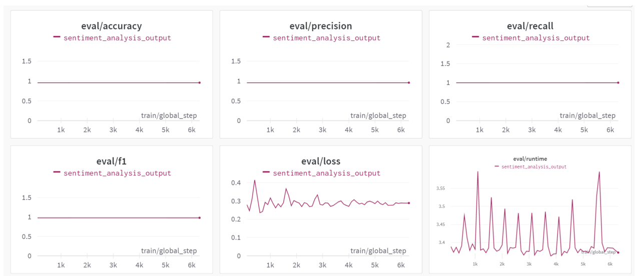

Figure 5. Binary Class model - Accuracy, Recall, F1, Precision, Loss curves during training with a smaller learning rate of 1e-05, a in-balanced training set biased towards 5 star reviews, with mini-batches of size 1, over 6000 steps.

can see that the validation metrics are between 0.95 and 1 which is very good and hence do not change much during training. The validation loss started out at 0.4 and gradually decrease and stabilized around 0.288.

The performance of the binary class is much better than of the multi-class model. This could be explain intuitively that when given a customer review to classify, it is difficult even for a human to distinguish between 5-star and 4-start review, as well as 1-star and 2-start reviews but it's in general, we can agree that if the ratings are between 3 and 5 stars, the customers like and have a positive opinions about the products. But if the reviews are 1-star and 2-star, it is quite certain that the review writers do not like the product. In addition, for the purpose of our application, it is sufficient and straightforward to classify if the attitudes of the customers shown through the interaction records is positive or negative.

Model	accuracy	f1	precision	recall	loss
Multi-class	0.67	0.53	0.45	0.67	0.36
Binary	0.95	0.97	0.95	1.0	0.32

Table 1. Metrics for sentiment analysis models.

3.4. Summarization

3.4.1 BertSum Experiments and Results

In our experiments using BertSum for extractive summarization, we, like [20], trained the model using the CNN/Dailymail dataset with each of the the three different types of summarization layers. We used the same hyperparameters used in the original paper [20] but with one distinction: the training steps. We used only 10000 training steps, as opposed to [20]'s use of 50000 due to our intention of tuning hyperparameters with a relatively small fixed training step count of 10000.

We tested the trained models on the amazon reviews dataset [2] using the hyperparameters stated prior, and the results can be seen in Table 2, where the results are ROUGE F1 scores, so both precision and recall of the overlapping n-grams are taken into account. It is noted that the amazon reviews dataset has only single sentence highlights to be used to choose the target summaries (oracle summary), while BertSum, by default, produces 3 summaries. Thus, we modified BertSum so that it produces only 1 sentence (the sentence with the highest score) instead of 3 to have better precision.

Model	ROUGE-1	ROUGE-2	ROUGE-L
B+Classifier	9.61	2.24	8.63
B+Transformer	9.74	2.39	8.86
B+LSTM	9.55	2.18	8.55

Table 2. Results of BertSum with different summarization layers (table format from [20]). Used default hyperparameters from [20] with the exception of training steps, where we used 10000 (as opposed to 50000). Fine-tuned using CNN/Dailymail [3] and the results shown are from from testing on the amazon reviews dataset [2]. Scores are ROUGE F1.

From Table 2, the best performing model is the model that uses the transformer as its summarization layer. Although the scores certainly do not, this matches, at least qualitatively, the results from [20], where the transformer summarization layer also performed best. We took this model, and tuned a number of hyperparameters, but our results show that for most adjustments, the results are worse by 1-3 points (e.g., from 9.74 to values as low as 6.88).

3.4.2 BertSum Analysis

First, the results from Table 2. There are, firstly, two obvious reasons for the poor results. The first reason is due to the training steps. As noted before, we used a reduced training step count of 10000, which we believe is a contributing (not significant) reason for the poorer score. One way we verified this is by testing the model trained with 10000 steps on the CNN/Dailymail dataset with [20]'s result of using 50000 steps. See Table 3, which gives us confidence to our suspicion.

The second reason for the poor results shown in Table 2 is the amazon reviews dataset, where all of the highlighted summaries are primarily single sentence reviews. See Table 3 for two samples. As we can see, the highlights (amazon review) can be overly short, leading to a chosen oracle summary with poor ROUGE score, even though our model's output summary can have a similar semantic meaning.

Training Step Count	ROUGE-1	ROUGE-2	ROUGE-L
10000	33.11	15.26	29.62
50000	43.25	20.24	39.63

Table 3. Result of BERTSUM+Transformer with same hyperparameters with the exception of training step count. Second row taken directly from [20]. Both trained and tested on CNN/Dailymail. Scores are ROUGE F1.

Our Models Summary	Highlight (to select oracle)
too short for the price, too long for the story	meh
many of the plays were ones I was not familiar with, so I learned a thing or two myself	great as a read aloud

Table 4. The first sample is about a review concerning a book about soldiers. The second sample is a review about a kindle book about a play. Model outputs are lower-cased as BERT-uncased is used in BertSum.

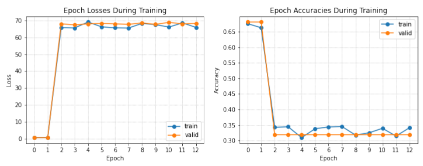

Figure 6. Loss and accuracy curves during training with the original Sentence. architecture.

3.4.3 Sentence.

Initially, we mirrored the original configuration of *Sentence.*: a 2:1 ratio between out-of-summary sentences and in-summary sentences and a hidden-layer size of 784 before the classifier. However, as the training curves show in Figure 6, this did not achieve good results. The model initially achieves an accuracy of about 0.67 before flipping to 0.33 in just two epoch; the first interval classifies everything as out-of-summary, while the second interval classifies everything as in-summary.

The configuration with a balanced training set (unmodified validation set), smaller learning rate, and additive rather than multiplicative embedding combination demonstrated improvement during training, though the results were not ideal either. As Figure 7 shows, the model consistently improves on the training set; however, it overfits and begins to do poorly on the validation set.

We believe the reason a smaller learning rate (by two orders of magnitude) helped is that there is a small boundary between an in-summary and out-of-summary embedding in semantic space, so the model requires very granular changes to be able to discriminate between them.

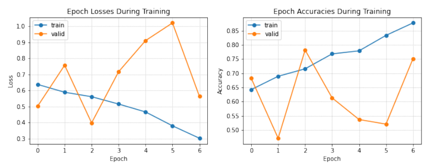

Figure 7. Loss and Accuracy curves during training with a smaller learning rate, a 1:1 balanced training set, and an additive rather than multiplicative embedding combination.

Overall, the greatest impediment was lack of computing power. Due to insufficient resources, we were unable to train on the full set of data within a reasonable amount of time; all of these results are on subsets of their respective partitions in the original dataset.

We took two models from this best-performing configuration to run on the held-out test partition: the model from epoch 2 (best performance on validation set) and the model from epoch 6 (best performance on the training set). As expected, the model from epoch 2 is able to generalize better to the test set, achieving a higher score, shown in Table 5.

Model	ROUGE-1	ROUGE-L
Best Valid Loss	24.78	16.66
Best Train Loss	23.51	15.52

Table 5. Result of Sentence. on the first 500 documents of the test partition of CNN/DailyMail.

4. Conclusion

We ended with models that work on both aspects of the task we set out to solve, but performance was unsatisfactory and while we could to some extent adapt the Amazon reviews dataset to fit, we didn't have a compelling dataset for the specific problem we were interested in. A critical avenue of future work is to construct a dataset that includes both summarization and sentiment analysis so that a combined system can be meaningfully evaluated.

Additionally, since this is intended to be a user-facing application, actual user studies are necessary for future development, which prompts a number of questions: could abstractive summaries be more semantically meaningful? If so, does that outweigh their lack of explainability relative to extractive summaries? Is summarization actually useful for a manager, or would a higher-level view such as sentiment-tagged topic clusters be more valuable for triage?

References

[1] https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270. 2

[2] http://jmcauley.ucsd.edu/data/amazon/links.html. 4, 5

[3] https://github.com/abisee/cnn-dailymail. 5

[4] https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b.

[5] https://towardsdatascience.com/extractive-summarization-using-bert-966e912f4142.

[6] https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[7] https://pytorch.org/tutorials/beginner/nn_tutorial.html.

[8] https://gluebenchmark.com/.

[9] https://openreview.net/pdf?id=rJ4km2R5t7.

[10] https://pytorch.org/hub/huggingface_pytorch-transformers/.

[11] https://huggingface.co/transformers.

[12] https://neptune.ai/blog/how-to-code-bert-using-pytorch-tutorial.

[13] http://jalammar.github.io/illustrated-bert/.

[14] https://mccormickml.com/2019/07/22/BERT-fine-tuning/.

[15] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[17] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III au2, and Kate Crawford. Datasheets for datasets, 2021.

[18] Cloudera Fast Forward labs. Extractive summarization with bert. https://github.com/designingwithml/mlconcepts/tree/main/extractivesummarization, 2021. 3

[19] Yang Liu. https://github.com/nlpyang/BertSum. 2

[20] Yang Liu. Fine-tune bert for extractive summarization, 2019. 2, 3, 5, 6

[21] Yang Liu. Fine-tune BERT for extractive summarization. *CoRR*, abs/1903.10318, 2019.

[22] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders, 2019. 2

[23] T.J. Gaffney Melanie Beck, Victor Dibia. Summarize. https://github.com/fastforwardlabs/summarize., 2021. 3

[24] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[25] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. 3, 4

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Student Name	Contributed Aspects	Details
Jared Deiner	Multi-Class/Binary Sentiment Impl and Analysis	Wrote scaffolding for training/testing pipeline for BERT models, imported and sanitized datasets for training/testing, wrote generic dataloader and custom dataloaders for Amazon and IMdB datasets, extended pretrained BERT model to add additional layers.
Tran Ton	Sentiment Implementation and Analysis	Trained and tested Amazon Product reviews datasets, implemented hyperparameter tuning, analyzed results and ran experiments
Jacob Morello	Extractive Summarization	Preprocessing code for CNN/Dailymail; training loop for Sentence. network; metrics, evaluation, and visualization code for Sentence; several experiments with various configurations for Sentence. (see google drive links in README).
Hung Hua	Extractive Summarization	Trained and tested BertSum model with the classifier, Transformer, and LSTM as the summarization layers. Modified existing BertSum code to process Amazon reviews data, and to output single sentence summaries. Tuned hyperparameters of BertSum model. Experiementation and Analysis of BertSum results.

Table 6. Contributions of team members.