Is Text Summarization a Useful Tool for Classification?

Octavio E. Lima, Alexander Dong, Keren Makhervaks, Mariana Luna

{octavioe, adong9, kerenmak, mlunaz}@seas.upenn.edu

December 2022

Abstract

We investigate whether machine learning classification algorithms suffer a decrease in performance when classifying text summaries as opposed to the original raw text. Using a data set of Reddit Posts (with their abstractive summaries), we create extractive summaries using BART, and apply popular M.L. algorithms. The results show that, on average, the performance of the algorithms was slightly worse when classifying the extractive summaries compared to the raw text. However, the decrease in performance was not significant and did not affect the overall accuracy of the algorithms. Abstractive summaries did not yield equally satisfying results. This suggests that using extractive summaries as input for classification algorithms can be an effective way to reduce the amount of data without sacrificing performance.

Keywords: Transfer Learning, BART, Seq2Seq, S-BERT Embeddings, Random Forest, XGBoost, Big Data, Transformers, Classification

1 Introduction

Using Reddit data, this paper presents a study on the performance of machine learning classification algorithms when classifying text summaries as opposed to the original raw text with the use of two different embeddings. Our goal is to investigate whether using summaries and the type of embeddings used as an input to classification algorithms can reduce the amount of data needed to train the algorithms without significantly affecting their performance.

To conduct our study, we first create extractive summaries of the Reddit posts using BART. We then use two different types of word embeddings: `S-BERT` (as proposed by Choi et al, 2016) and `CountVectorizer` embeddings. We use these as input to classification algorithms and measure their performance using metrics such as F1, accuracy, and ROC. Lastly, We compare these scores with that of the raw sentences. We find that using extractive summaries with `S-BERT` or `Count Vectorizer` embeddings as input to classification algorithms can be an effective way to accelerate the process without significantly affecting their performance.

This research can be applied in a variety of settings where text classification is used, such as in N.L.P. applications, sentiment analysis, and topic modeling. For example, a company that uses M.L. algorithms to classify customer reviews or social media posts could benefit from using summaries of the texts as input — as it would reduce the amount of data needed to train the algorithms.

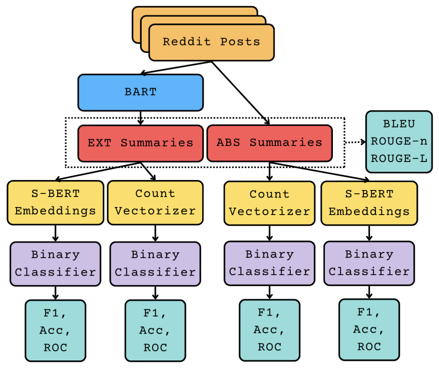

Figure 1: Workflow Diagram

2 Literature Review

In the paper *"NLP based Machine Learning Approaches for Text Summarization"*[1], Rahul et al

[1] Rahul, S. Adhikari and Monika, "NLP based Machine Learning Approaches for Text Summarization," 2020 Fourth International Conference on Computing Methodologies

present various approaches to generate summaries of large texts. The methods discussed produce either abstractive (ABS) or extractive (EXT) summaries. EXT summarization uses the same sentences as the original text, while ABS summarization focuses on the key concepts of the document. The paper also discusses different types of text summarization including single document, multi-document, query-based, and generic summarization. Various applications of text summarization in fields such as science, medicine, law, and engineering are mentioned. The paper then describes the use of supervised machine learning methods such as Naive Bayes, Random Forest, and SVD for generating EXT summaries. Deep learning techniques such as CNN, RNN, and sequence-to-sequence models are also discussed for both ABS and EXT summarization.

Similarly, in the paper *"Text Summarization Techniques: A Brief Survey"*[2], the authors note that extractive methods tend to produce better results than abstractive ones. The paper then outlines three steps commonly performed by summarizers: constructing an intermediate representation of the input text, scoring sentences based on that representation, and selecting a summary comprising a number of sentences. The paper also discusses two types of intermediate representation: topic and indicator representation. The paper concludes by discussing the evaluation of summarization systems and the challenges of applying these methods in practice.

Additionally, in a paper[3] from ArXiv.org, the authors describe a state-of-the-art model for extractive summarization of documents using a recurrent neural network (RNN) that is trained using abstractive techniques. The model treats extractive summarization as a sequence-to-sequence classification problem, where each sentence is visited in the original order of the document and a binary decision is made on whether or not it should be included in the summary. The model uses a bi-directional gated recurrent unit (GRU) as the basic building block, and has two layers: a word-level RNN that computes hidden state representations for each word, and a sentence-level RNN that encodes the representations of sentences in the document. The model is trained using extractive labels, and is evaluated using different variants of the Rouge metric, showing it to be effective and interpretable.

and Communication (ICCMC), 2020, pp. 535-538, doi: 10.1109/ICCMC48092.2020.ICCMC-00099.

[2]Allahyari, Mehdi, et al. "Text Summarization Techniques: A Brief Survey." ArXiv.org, July 2017, https://arxiv.org/abs/1707.02268.

[3]Nallapati, Zhai, et al. "SummaRuNNer: A Recurrent Neural Network based Squence Model for Extractive Summarization of Documents" ArXiv.org, Novemeber 2016, https://arxiv.org/pdf/1611.04230v1.pdf.

Lastly, in the paper *"Embedding Performance on Downstream NLP Tasks"*[4], Hyunjin Choi et all find that the pre-trained BERT and ALBERT models can be fine-tuned to give state-of-the-art results in sentence-pair regressions such as semantic textual similarity (STS) and natural language inference (NLI). This paper explores sentence embedding models for BERT and ALBERT using modified BERT networks called Sentence-BERT (SBERT) and Sentence-ALBERT (SALBERT). The authors use cosine-similarity to evaluate the similarity between two sentence embeddings, and their results indicate that the CNN architecture improves ALBERT models substantially more than BERT models for STS benchmark. On average, SBERT and CNN-SBERT embeddings achieve a Spearman's rank correlation point of 84.49 and 84.34, respectively, indicating a high increase in performance on NLP tasks.

3 Experimental Design

3.1 Data

Our raw data[5] is available for download as a zipped JSON file. The file has a download size of 2.93 GiB and a data set size of 18.09 GiB. It consists of 3,848,330 Reddit posts, with an average length of 270 words for content and 28 words for the abstractive summary. We select two subreddit categories — `leagueoflegends` and `funny` — from the hundreds available in the original data. After cleaning and removing duplicates, we find a well-distributed data set with a total count of 2,360 observations for `leagueoflegends` and 1,771 observations for `funny`. This indicates that the data is representative of the two categories and that we have enough data to work with. This is important as our analysis and findings will not be biased towards one category over the other. More specific details of our final data may be found below.

3.2 Problem Specification

Formally, our problem can be defined as follows: Given a data set of texts and their summaries, apply a set of machine learning classification algorithms to the data. Evaluate the performance of the algorithms on the *(i)* raw text, *(ii)* abstractive summaries, and *(iii)* extractive summaries using two different types of word embeddings: *(i)* `S-BERT` and *(ii)* `Count Vectorizer`.

[4] Hyunjin Choi, et al. "Evaluation of BERT and ALBERT Sentence Embedding Performance on Downstream NLP Tasks" ArXiv.org, Novemeber 2016, https://arxiv.org/pdf/2101.10642.pdf.

[5]"Reddit: Tensorflow Datasets." TensorFlow, https://www.tensorflow.org/datasets/catalog/reddit

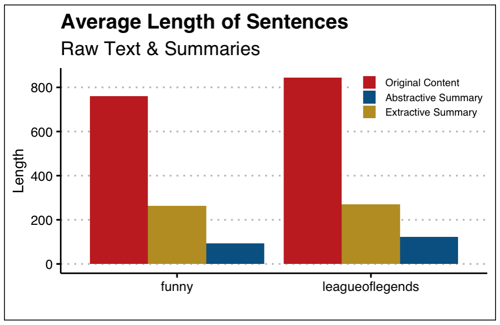

Figure 2: Mean Sentence Length by Subreddit

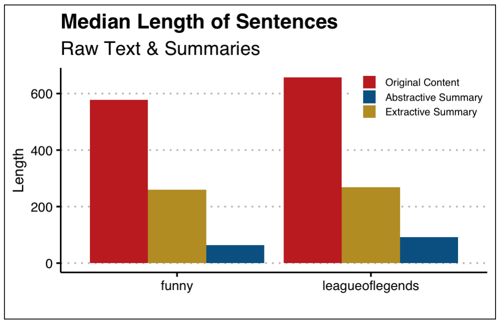

Figure 3: Median Sentence Length by Subreddit

Then compare the results to determine whether the type of summary (abstractive vs. extractive) and embedding affects the performance of the algorithms. The performance metrics used in the evaluation includes accuracy, precision, recall, and F1 score.

3.3 Evaluation Metric

First, we use two standard Text Summarization metrics to evaluate the quality of our abstractive and extractive summaries: ROUGE-1 FScore and BLEU Score. In summary, ROUGE-1 FScore measures the overlap between a generated summary and a reference summary, using precision and recall to compute the score. Conversely, BLEU measures the overlap between a generated summary and a reference summary, using a modified version of precision to compute the score. This modified version of precision is computed by taking the weighted average of the number of n-grams (sequences of n words) that match between the generated summary and the reference summary. In general, ROUGE1 FScore is considered to be more comprehensive than BLEU. This is because ROUGE-1 FScore computes both precision and recall, while BLEU Score only computes a modified version of precision.

Our extractive summaries are reliable, but their performance tends to decrease as the length of the sentences in the original text increases. This is because extractive summarization involves selecting and combining relevant sentences or phrases from the original text, so if the sentences are long, it can be more difficult to identify and select the most relevant information. Conversely, the abstractive summaries are often worse but more stable, meaning that their performance decreases at a slower rate as the raw text gets longer.

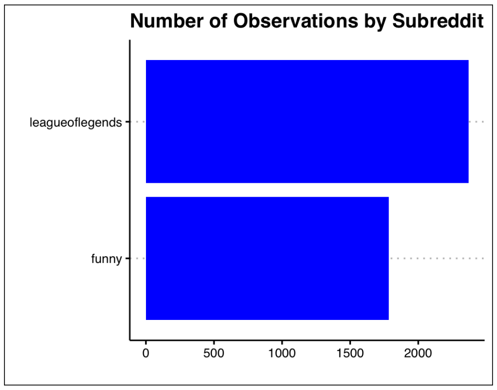

Figure 4: Observation Count by Subreddit

Our classification metrics are Accuracy (which measures the proportion of correct predictions), F1 Score (a measure of the balance between precision and recall), and ROC AUC (a measure of the ability to distinguish between positive and negative classes; ie, the area under the Receiver Operating Characteristic curve). Furthermore, our Text Summarization metrics are BLEU, ROUGE-n, and ROUGE-L, which are defined below.

$$\text{BLEU} = exp(\sum_{n=1}^{N} w_n \cdot \log p_n) \tag{1}$$

$$\text{ROUGE-n} = \frac{\text{\# Matching NGrams}}{\text{\# Ngrams in Reference Text}} \tag{2}$$

$$\text{ROUGE-L} = \frac{\text{Length of the LCS}}{\text{Length of Reference Text}} \tag{3}$$

where, on [1], BP is the Brevity penalty, N is the number of n-grams, w is the weight for each modified precision, and p is the Modified precision. On [3], LCS stands for "Longest Common Sequence". And Recall, Precision, and FScore are calculated on [2] and [3].

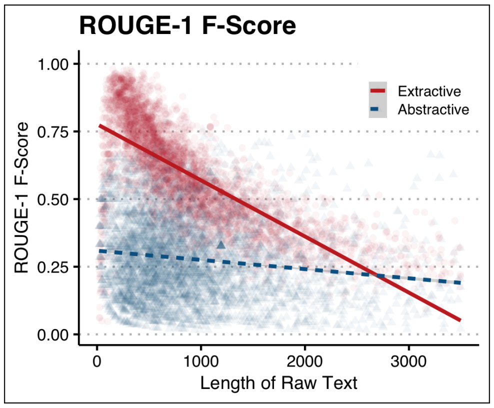

Figure 5: ROUGE1 FScore by Sentence Length

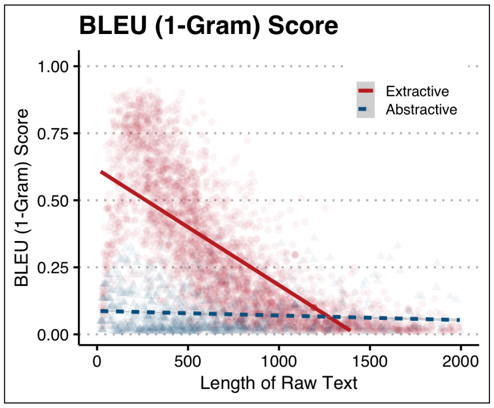

Figure 6: BLEU Score by Sentence Length

3.4 Simple Baseline

We use a BART seq2seq model to create extractive summaries of sentences using the `transfomers` library in Python. The BART model is pre-trained on a large corpus of sentences, and it is used to generate extractive summaries of the sentences in the corpus.

After generating the extractive summaries, we use a simple embedding, `CountVectorizer` and we feed these to Random Forest and XGBoost classification models to categorize the sentences based on the raw texts, extractive summaries, and abstractive summaries. The classification models are trained on the raw reddit posts, where the sentences are labeled with the appropriate subreddit categories. The models are then used to predict the subreddit categories for the sentences in the corpus, using the raw texts, extractive summaries, and abstractive summaries individually as input.

The results of this research design provide insights into the effectiveness of using BART-generated extractive summaries for categorizing sentences (as opposed to abstractive summaries) when compared to the original sentence. It also provides a comparison of the performance of the Random Forest and XGBoost classification models on the different types of text inputs and the use of different types of word embeddings. This information can be used to improve the performance of classification models in N.L.P. applications. It also compares the difference in running time among the three types of inputs.

4 Experimental Results

4.1 Published Baseline

To analyze the performance of the two types of summaries, we first use BLEU, ROUGE-n, and ROUGE-L metrics. As mentioned in Section 3.3, the extractive summaries tend to have more reliable scores, which tend to decrease as the original sentences get longer (as seen in Figure 5 and 6). On the other hand, the abstractive summaries are shorter and more stable relative to the length of the raw texts, but their performance is generally worse.

Next, we investigate the binary classification performance of both summary types using `Count Vectorizer`[6] embeddings as we feed the sentences into the two classification algorithms (Random Forest and XGBoost). When using a `Count Vectorizer`, the input text is first tokenized into words/phrases, then a count is taken of the number of times each word/phrase appears in the text. The count is then used to create a numerical representation of the text, which is then used as the input in those two classification models.

We observe that the classification performance using the abstractive summaries is lower than that of the extractive summaries because abstractive summarization involves generating a new, condensed version of the original text, while extractive summarization involves selecting and combining relevant sentences or phrases from the original text. This means that the model has to do more work and make more decisions when classifying an abstractive summary, which can lead to lower performance. Additionally, because raw text contains more information than a summary, both Random Forest and XGBoost perform better on the original raw sentences above all (as expected).

Table 1 displays the performance of the two classifiers based on the different types of text inputs.

[6]Sharma, Yashika. "Understanding Count Vectorizer." Medium, The Startup, 24 May 2020, https://medium.com/swlh/understanding-count-vectorizer-5dd71530c1b.

Model	Data to Classify	Accuracy	F1 Score	ROC Score	Time (sec)
Random Forest	All Content	0.9140	0.9442	0.9727	1.51
	Extractive	0.8603	0.8601	0.9261	0.632
	Abstractive	0.7056	0.6999	0.7965	0.262
XGBoost	All Content	0.9321	0.9433	0.9910	1.54
	Extractive	0.8637	0.8633	0.9318	0.655
	Abstractive	0.7179	0.7139	0.8038	0.257

Table 1: Classification Model Results using Count Embeddings

4.2 Extensions

We first use classification as an additional metric to evaluate the extractive and abstractive summaries. Then we evaluate the use of different word embeddings, `S-BERT` and `Count Vectorizer`, to increase classification accuracy and analyze classification run time to see the effects of summarization on classification on a larger scale.

The difference in running time between the classification model using `Count Vectorizer` embeddings for the extractive summaries and raw text was small and measured in miliseconds, this is a substantial finding and important as the size of the data scales up. For example, our data consisted of only 4,131 observations, so the $>50\%$ decrease in running time was not significant. However, if the data were to increase to millions or billions of observations, that same $>50\%$ decrease in running time could translate into significant time savings. This is because the running time of the classification model would grow linearly with the size of the data, so a small decrease in running time for a small data set would become much more significant for a large data set.

Next, we investigate the binary classification performance of both summary types using `S-BERT` embeddings as we feed the sentences into the two classification algorithms (Random Forest and XGBoost). `Sentence-BERT` embeddings use a Siamese-network-like architecture to provide two sentences as an input. These two sentences are then passed to BERT models and a pooling layer to generate their embeddings [7]. Computing the `S-BERT` embeddings takes longer than the simple `Count Vectorizer` word embeddings. This makes the run time of the prediction much larger, but we also see an increase in accuracy, F-1 and ROC scores.

After conducting a thorough analysis, we have found that the performance of both the Random Forest and XGBoost algorithms was significantly better when using `S-BERT` (sentence-bert) embeddings compared to `Count Vectorizer` embeddings. This was the case for all three text input types (raw text, extractive summaries, and abstractive summaries) that we tested, as seen in Figure 7. However, this increase in performance came at a great cost. The classification models using `S-BERT` embeddings took significantly longer to run than those using `Count Vectorizer`. For example, the models using `S-BERT` took 284 seconds to run on the raw text, 54 seconds on the abstractive summaries, and 116 seconds on the extractive summaries. In comparison, the models using `Count Vectorizer` only took 1.52, 0.260, and 0.640 seconds to run on the same text inputs, respectively.

Model	Data to Classify	Accuracy	F1 Score	ROC Score	Time (sec)
Random Forest	Raw	0.996	0.995	0.999	202.06
	Extractive	0.944	0.935	0.9778	82.74
	Abstractive	0.826	0.824	0.909	38.97
XGBoost	Raw	0.998	0.998	0.999	365.68
	Extractive	0.950	0.949	0.987	150.72
	Abstractive	0.844	0.844	0.929	69.54

Table 2: Classification Model Results using S-BERT Embeddings

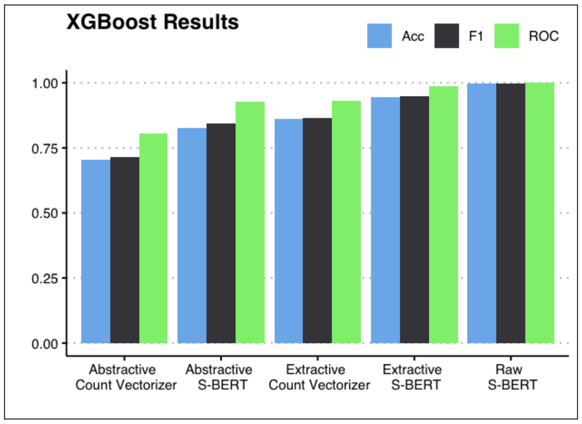

Figure 7: XGBoost Results by Input Type

While the improved performance of the models using `SBERT` embeddings is impressive, the increased running time is a disadvantage. Nonetheless, this comparison between the performance of the two classification algorithms, using `SBERT` and `Count Vectorizer` embeddings, is important for other reasons.

Firstly, it provides a robustness check to our study. By using two different types of embeddings, we can confirm that our results are not specific to just one type of embedding, and that they are likely to be generalizable to other similar approaches. Secondly, the comparison allows us to evaluate the relative performance of the two different algorithms when using different types of embeddings. This can help us to understand which type of embedding is more effective for a given classification task, and can guide future research in this area.

[7]Kotamraju, S. (2022). An Intuitive Explanation of Sentence-BERT. Towards Data Science. https://towardsdatascience.com/an-intuitive-explanation-of-sentence-bert-1984d144a868

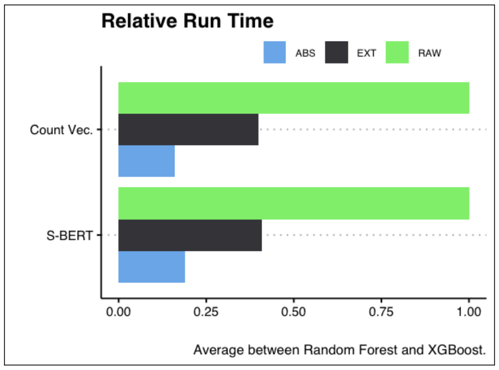

Figure 8: Classification Running Time

Fig. 8 displays the relative running times by Embedding type, where the 0-1 scale represents the running time of the summaries compared to raw sentences. E.g., if it takes 10 seconds to run the model on raw sentences using Count Vectorizer, 3 seconds on extractive summaries (EXT), and 1 second on abstractive summaries (ABS), then the relative running time of the raw sentences would be 1.0 (100%), the relative running time of the extractive summaries would be 0.30 (30%), and the relative running time of the abstractive summaries would be 0.10 (10%). This is important because it shows the relative difference in running time between the different types of summaries and raw sentences, which may be significant as the data increases. The raw sentences are set as 100% on the scale. Fig. 9 shows the run times using `S-BERT`.

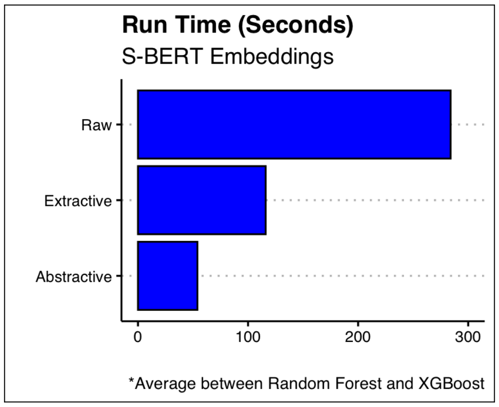

Figure 9: Classification Running Time, S-BERT

We conclude that while `SBERT` does give great performance improvements when compared to `Count Vectorizer`, the increase in running time overshadows this. Therefore, for our data, it may be sufficient to use `Count Vectorizer` embeddings as we still get strong performance results. However, it is important to note that perhaps for more complex and less easily separable data, `SBERT` may be necessary to obtain sufficient performance results.

4.3 Error Analysis

One common theme of misclassification is that of `funny` subreddits with "gaming terminology" that are misclassified as part of the `leagueoflegends`. This occurs in both our Random Forest and XGBoost models. There are many funny stories in this subreddit category, and descriptive terminologies like "monster" cause some stories to get misclassified. This is because `leagueoflegends` is a sci-fi multiplayer online battle arena video game, so terms such as "monster" would likely be mentioned within its subreddit category. For example consider the following extractive summaries that both get misclassified:

```
1. He cried out like a girl, screamed that
I should lock up my "crazy monster",
jumped in his truck and sped off.

2. Goal differential would be determined
on the scores of USA v Germany and Ghana v
Portugal games .
```

Summary 1 is part of a funny story that a user is telling. This is getting consistently classified as `leagueoflegends` despite the type of word embedding and classifier we use. Summary 2 describes an unlikely soccer turnout and the use of the word "score" is mostly attributed to `leagueoflegends`. Thus, we can see that the use of descriptive gaming language is a common error in our classification.

Using our best classification model, XGBoost we can see the rates of false positive (predicting `funny` when the real label was `leagueoflegends`) and false negative (predicting `leagueoflegends` when the real label was `funny`).

- Classification of Abstractive Summaries:

  FP: $\frac{1{,}121}{4{,}131} \approx 0.27$

  FN: $\frac{1{,}727}{4{,}131} \approx 0.42$

- Classification of Extractive Summaries:

  FP: $\frac{471}{4{,}131} \approx 0.11$

  FN: $\frac{92}{4{,}131} \approx 0.02$

We see a significant increase in misclassification using abstractive summaries on both the false positives and false negatives. Thus, we can see that, overall, the extractive summaries are less error-prone in regards to classification. On abstractive and extractive summaries, the false positive error is more prevalent than the false negatives. This means our model makes more mistakes identifying subreddits as `funny` when they are supposed to be a part of `leagueoflegends`.

5 Conclusions

In conclusion, our study demonstrates that extractive summarization can be a useful preprocessing step for classification tasks, as it allows for a significant reduction in the amount of data and running time required without sacrificing performance. We conducted a robustness check by using two different types of word embeddings — `S-BERT` (also known as `Sentence-BERT`) and `Count Vectorizer` — to measure the performance of our classification model.

Our findings showed that while the model using abstractive summaries was the fastest, it resulted in a significant decrease in performance as measured by F1, Accuracy, and ROC scores when compared to the model using extractive summaries with both types of embeddings. In contrast, the model using extractive summaries with both `S-BERT` and `Count Vectorizer` was much faster than the model run on raw texts, and only caused a minor decrease in performance. It's worth noting that the models using `S-BERT` embeddings took substantially longer to run compared to those using `Count Vectorizer`. `S-BERT` embeddings are a powerful tool that, given a complicated classification task, can help the model increase its performance drastically.

These results suggest that using extractive summaries as input for classification algorithms can be an effective way to reduce the amount of data needed without sacrificing performance, and can also reduce the running time of the classification algorithms. Lastly, this could be applied, for instance, by a company that handles a large volume of customer service inquiries through email or chat. The company could use extractive summaries of the inquiries, and feed them into a classification model that would indicate the type of inquiry based on the summarized content.

6 Appendix

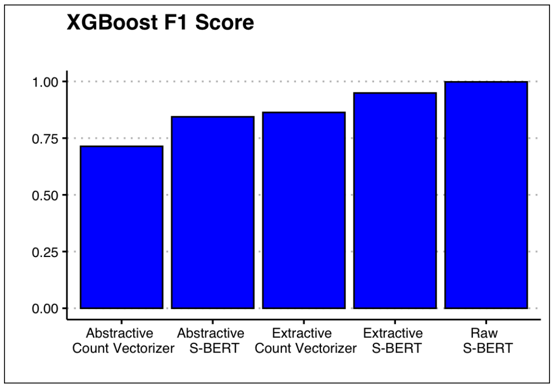

Figure 10: XGBoost Classification Results by Input Type

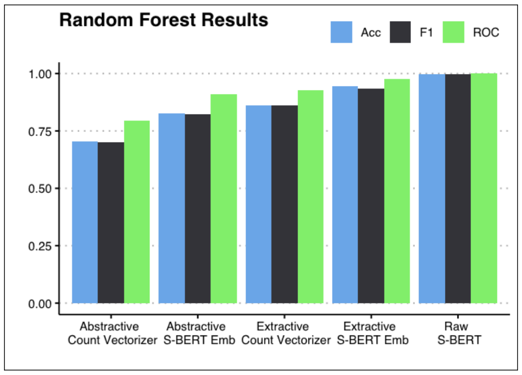

Figure 11: Random Forest Results by Input Type

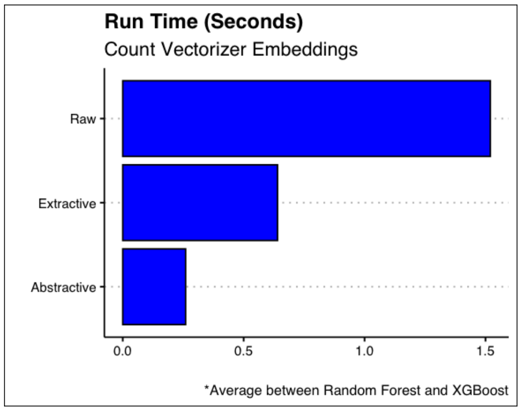

Figure 12: Random Forest Results by Input Type