BERT Sentiment Analysis

Executive summary

A DistilBERT model was tuned, trained, and evaluated to predict the sentiment of user reviews from websites such as product reviews from Amazon, movie reviews from IMDB, and business reviews from Yelp. The fully trained model was able to accurately label a review in text form as either positive or negative with an F1-score of 0.96.

Problem statement

In the face of staggering amounts of text-form data that require processing in the modern day, one of the major problems companies face is sentiment analysis. In this project, I fine-tuned a BERT model to predict whether a given product review was positive or negative, based on the text of the review itself. I fine-tuned and trained a DistilBERT base model on review data sets from Amazon, Yelp, and IMDB, sourced from Kaggle.

I used the HuggingFace Transformers library to implement the model, as well as the Optuna library to find the optimal hyperparameters.

Background

Natural Language Processing (NLP) is an increasingly vital field of Artificial Intelligence. Large Language Models (LLMs) have been dominating the headlines especially since ChatGPT rose to prominence in late 2022. GPT (Generative Pre-trained Transformers) models are a family of autoregressive Large Language Models. GPT models, as their name indicates, are intended to generate text by predicting the next word in a given sequence.

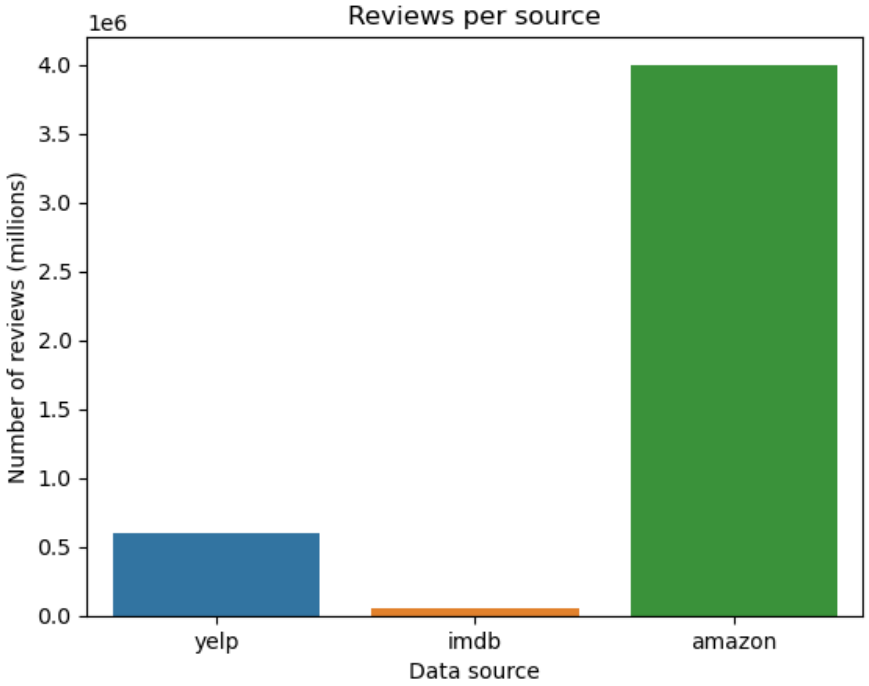
While GPT-based models are powerful, they are not the best option for every use case. BERT (Bidirectional Encoder Representations from Transformers) is an advanced model for certain specific use cases in NLP, such as text classification. For such tasks, BERT models can outperform GPT models despite using much smaller scale data–GPT-3, for instance, has 175 billion parameters, compared to BERT's 340 million (Bosley, 2023).

Data Sources

The data used to train my model was collected from three Kaggle datasets:

- Ilham Firdausi Putra's "Yelp Review Sentiment Dataset," containing nearly 600 thousand business reviews.
- Lakshmipathi N's "IMDB Dataset of 50K Movie Reviews," which did indeed contain exactly 50 thousand movie reviews
- Adam Bittlingmayer's "Amazon Reviews for Sentiment Analysis," containing over 4 million Amazon product reviews.

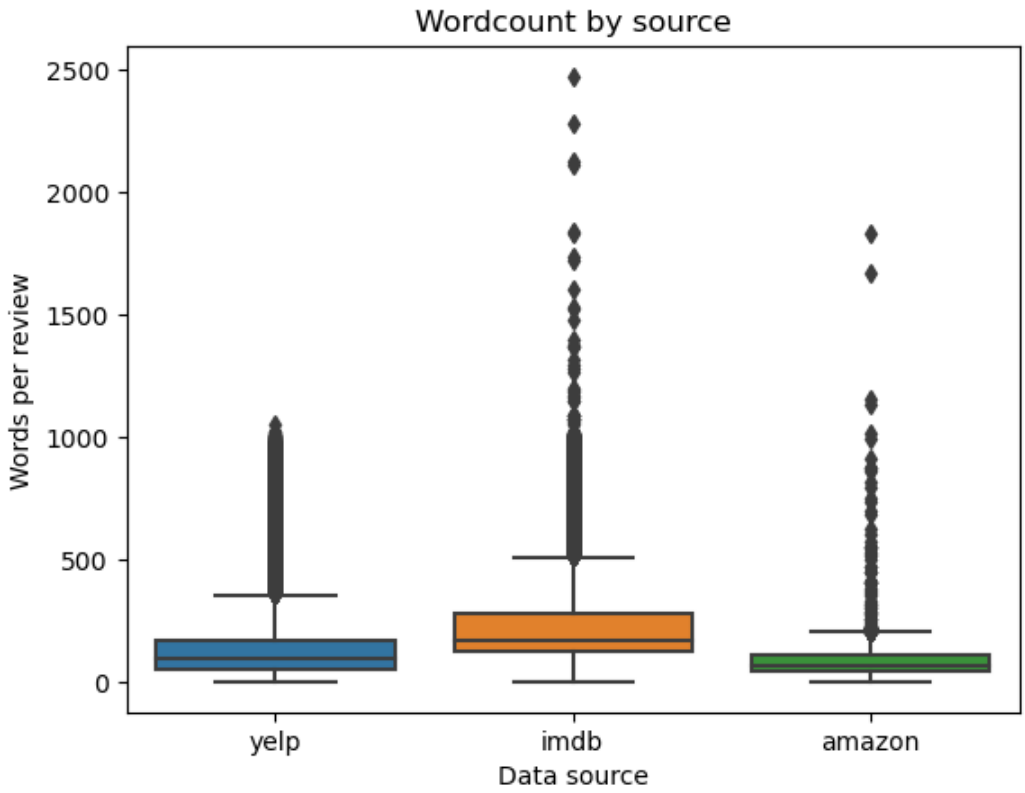
Figure 1 displays the distribution of data points from each website.



(Fig 1)

Each dataset contained the text of reviews, together with a label stating whether the review was positive (associated with a high star rating), or negative (associated with a low star rating). Although each set had slight differences in formatting that needed to be resolved before they could be combined, they were each perfectly balanced, with exactly 50% positive and 50% negative sentiment.

The lengths of the reviews varied by source, as shown in the box plot in Figure 2.



(Figure 2)

As the plot shows, most of the reviews sit in a very narrow range in the low hundreds of words, with Amazon having the shortest reviews overall and IMDB having the longest. Many high-word count outliers exist for each source, with Yelp having the least extreme outliers (rarely over one thousand words) and IMDB having the most extreme, with a few in the two thousands.

As it turned out, Yelp had over 300 reviews containing only one word, while neither of the other data sets contained any.

I used a stratified train/test split after combining the data, and used a relatively small sample of the data (half a million data points) in the final model.

Preprocessing

Model choice

The final model used was HuggingFace's 'distilbert-base-uncased' model, a smaller, faster version of the original BERT model that uses a strategy of randomly masking 15% of the words in the input, and predicting the masked words. Its training objectives involve generating hidden states as close as possible to the original BERT model (Sanh, 2019). The uncased model does not distinguish between capital and lowercase letters in the input text. This model was chosen for its increased efficiency, without significant loss of accuracy.

Details

First, the input text was tokenized with HuggingFace Transformers' Autotokenizer with the following code:

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased')
```

The tokenizer created a data frame sampled in Figure 3.

	index	input_ids	attention_mask
0	886662	[101, 2307, 7692, 1024, 2023, 2003, 1037, 2307...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1	112830	[101, 2054, 2019, 12476, 24385, 999, 1032, 105...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
2	2320316	[101, 2204, 2096, 2009, 6354, 1006, 2005, 2026...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
3	566879	[101, 1045, 2293, 2023, 2173, 1012, 1045, 2134...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	1666132	[101, 2987, 1005, 1056, 2377, 2006, 1037, 4966...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

(Fig 3)

The "input_ids" feature is a list of each word, subword, item of punctuation, etc, turned into a number. For instance, in the first element above, the token "2023" represents the word "this" while the token "1024" represents a colon.

The "attention_mask" feature contains a 1 for each used token, and a 0 for each "[PAD]" token. [PAD] tokens were used for reviews shorter than the maximum length allowable by the tokenizer (512 tokens), so that the lengths of the reviews would be uniform. Reviews longer than the maximum length were truncated.

Inspection revealed that the token "100" mapped to "[UNK]", a token for a word not recognizable to the tokenizer. Closer examination of some reviews with unknown tokens indicated that the most common cause of unknown tokens was excessively repeated characters. One example was a reviewer emphasizing their satisfaction with the taste of some pasta by typing "mmmm..." with 157 m's.

These were found with the following code:

```
has_unknown = []
for i, input in enumerate(test_inputs['input_ids']):
  for tok in input:
    # Searches for an [UNK] token
    if tok == 100:
      # Prints entire line with padding characters removed
      print(f"input {i}: {tokenizer.decode(input).replace(' [PAD]','')}.")
      has_unknown.append(i)
      break
```

Hyperparameter Tuning

Before a model can be trained, there are several hyperparameters that must be selected, such as the learning rate, the number of training epochs, and the gradient accumulation steps. Simply guessing at the best values is far from optimal, so we use hyperparameter tuning. This is the process of trying out a number of different options for each hyperparameter on a subset of the data, to see which one performs the best. For this project, I used the Optuna library–a library created specifically for hyperparameter tuning.

Sample space

"learning_rate", 1e-5, 5e-5, log=True
"num_train_epochs", 1, 3
"gradient_accumulation_steps", 1, 8
"per_device_train_batch_size", 4, 16
"evaluation_strategy", ['steps', 'epoch']
"per_device_eval_batch_size", 4, 16
"warmup_steps", 100, 500
"weight_decay", 0.0, 0.1

Much of the code I used was based on tutorials from HuggingFace's Transformers library (Wolf et al, 2020), as well as Optuna's documentation (Akiba 2019). With Optuna, you specify a minimum and maximum value for each parameter, and the search attempts to find the optimal value within those bounds.

In the case of this model, the best hyperparameters found were the following: {'learning_rate': 4.122342215733177e-05, 'num_train_epochs': 1, 'gradient_accumulation_steps': 2, 'per_device_train_batch_size': 12, 'evaluation_strategy': 'epoch', 'per_device_eval_batch_size': 5, 'warmup_steps': 391, 'weight_decay': 0.08139944860406301}

Training and Evaluation

During the training process, there were certain tradeoffs that had to be made between the millions of data points available and the limitations of time and computing power. As such, only half a million data points were used for training, with about half that number used for evaluation. Despite this, the model performed exceptionally well, with an F1 score of 0.96. F1 score was used as the evaluation metric, since the two labels were functionally equivalent, with neither false positive nor false negative minimization being more important.

Technical Challenges and Solutions

On the first few tries, each attempt to train the model was met with a Key Error based on a randomly selected row. A search of several forums indicated that several people have had this trouble with the Transformers library over the years, but none of them received replies. The

Debugging Notebook was created with the goal of attempting the training on a small subset of the data, and a dive into the library's documentation revealed that the training model was expecting a PyTorch Dataset object. Along with a few other changes to the format of the data, such as a change of column names and label values, as well as resetting the indices of both the training and eval data, this solved the problem. I was then able to finally post a solution on the forums, so that the next person who runs into this same error will have a solution.

Another challenge faced was Google Colab's 24-hour runtime limit. Even with a subset of the data, training the model would have taken longer than that if it weren't for parallel processing enabled by the Accelerate library, as well as the DataLoader from Torch.

Conclusions and Future Work

Even with a limited amount of time and computing power, the DistilBERT model was able to reach remarkably high levels of performance. In the future, it might be interesting to see whether a model trained with the full training set of over 4 million total data points would have any significant improvement. It would also be interesting to compare various BERT models, such as a cased, rather than uncased, model, or a full BERT model.

One notable limitation of the data sets is that they include only the most polarized reviews, with neutral reviews unrepresented. Another is that there were a nonzero number of non-English reviews. Language identification before use in the model was somewhat beyond the scope of this project, but that could be a potential future extension

Another intriguing possibility would be to assess the model's ability to generalize across unseen data sources.

Appendices

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD

Mitchel Bosley, et al. "Do we still need BERT in the age of GPT? Comparing the benefits of domain-adaptation and in-context-learning approaches to using LLMs for Political Science Research." (2023).

Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". *ArXiv* abs/1910.01108. (2019).

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, & Alexander M. Rush. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing.

Data sets

Bittlingmayer, Adam. Amazon Reviews for Sentiment Analysis. 7, Kaggle, www.kaggle.com/datasets/bittlingmayer/amazonreviews.

N, Lakshmipathi. IMDB Dataset of 50K Movie Reviews. 1, Kaggle, www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

Putra, Ilham Firdausi. Yelp Review Sentiment Dataset. 2, Kaggle, www.kaggle.com/datasets/ilhamfp31/yelp-review-dataset.

Software and tools:

- HuggingFace Transformers
    - Implements the BERT models; critical for training and evaluation
- Jupyter Notebooks
    - Facilitates Python code display and documentation on Git repository
- Google Colab
    - Offers cloud computing with extra memory and GPUs, vital when handling large datasets beyond local machine capabilities
- Pandas
    - Manages and organizes the data into DataFrames for analysis
- Dask
    - Provides efficient handling for larger-than-memory datasets.
- Optuna
    - Conducts efficient hyperparameter tuning
- Scikit-Learn
    - Utilized for splitting the data into train and test sets for model validation
- Torch
    - Supports training and saving of models with required data structures
- Accelerator
    - Enables use of parallel processing, speeding up the model training process