## 

## 

## 

## 

## 

## 

## 

## Natural Language Processing

## Sentiment Analysis of Amazon Reviews

**Team:** Federico Cimini (CIS 5190), Liang-Yun Cheng (CIS 5190), Samuel Thudium (CIS 5190)

**Project Mentor TA:** David Yan

## **GitHub repository**: [AmazonFineFoodReviews](https://github.com/sthudium25/CIS5190-AmazonFoodReviews)

## Natural Language Processing

## Sentiment Analysis of Amazon Reviews

**Team:** Federico Cimini (CIS 5190), Liang-Yun Cheng (CIS 5190), Samuel Thudium (CIS 5190)

**Project Mentor TA:** David Yan

**GitHub repository**: [AmazonFineFoodReviews](https://github.com/sthudium25/CIS5190-AmazonFoodReviews)

#### 1) Abstract

Sentiment analysis is a popular introduction to natural language processing (NLP). This problem can be broken down into three simple steps: (1) clean input text, (2) learn embeddings for the words in present data, and (3) use these embeddings to classify the overall sentiment of new input text. We sought to build and improve upon the existing corpus of sentiment analysis notebooks that are prevalent on sites such as Kaggle. The sheer volume of these notebooks suggests that there are many ways to approach this problem, but typically there is no systematic comparison of the performance that these various methodologies have on a common dataset.

Thus, here we present a side-by-side performance of NLP and classification models on over 500,000 Amazon food reviews. Namely, we compare (1) NLP models with increasing fidelity to language structure, which are coupled with (2) classification models of increasing complexity. Overall, we were able to achieve quality results even using simpler models (TF-IDF + Logistic Regression), but saw an expected improvement using more complex models that take sequence information into account (word2vec + LSTM). This report aims to improve and expand upon the existing space of sentiment analysis projects.

#### 2) Introduction

To approach this problem, we used the “Amazon Fine Food Reviews”[[1]](#footnote-0) dataset from Kaggle to conduct model training and testing. This dataset consists of 568,454 Amazon reviews of food items; of interest to this study was the full review text, review summary, and star rating of each review. The full text of the reviews was cleaned and used as model input. The raw dataset contains star-review ratings from 1 to 5; these are skewed dramatically toward 5-star reviews (**Fig. 1A**). Given our knowledge of review systems, we opted to pool the five-scale rating classes into two, which we deemed “positive” and “negative”. The raw ratings were simplified as follows: scores 1 to 3 are encoded as -1, denoting a negative review, and scores 4 to 5 are encoded as 1, denoting a positive review. The pooled classes better reflected the overall sentiment that each review conveys; when customers give a 3-or-less-star rating, they typically have something negative to say about the product and conversely for 4- and 5-star reviews. Pooling slightly improved the balance between positive and negative classes, but the positive class still was much more abundant than the negatives (**Fig. 1B**). The imbalanced classes were handled with upsampling of the minority class in the training phase.

In addition to the baseline input of cleaned review text, we wanted to test the robustness of our chosen models to datashift. In particular, given models trained on whole reviews, we sought to interrogate the NLP and classification frameworks ability to predict sentiment given inputs from shifted distributions such as (1) review summaries, which are short phrases that act as titles for each review and (2) random word dropout from review text. These data shifts provide interesting challenges for sentiment analysis because they test the capacity of NLP models’ ability to capture features of user sentiment in a more restricted setting than the one on which the models were initially trained. An overview of summary statistics of the differences between the full review texts and review summaries is provided (**Fig. 1C**).

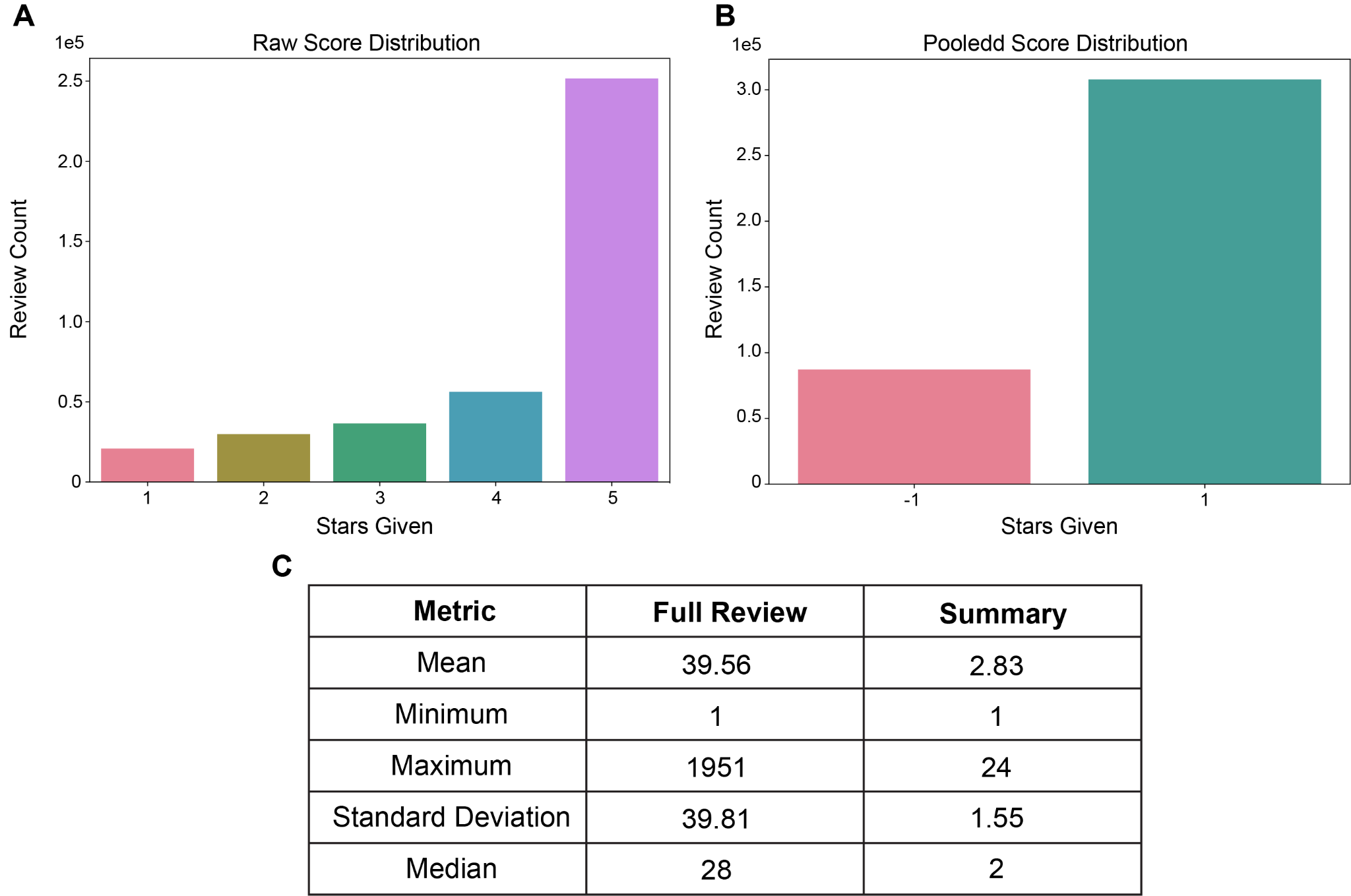


Figure 1.

**Implementation and Evaluation:**

Based on our review of existing resources, we proceeded to implement our systematic analysis of sentiment analysis methodologies into several discrete steps. Firstly, we had to clean the input text, removing punctuation, special characters, among other non-english words. Next, we had to pool the review scores into two general sentiment bins (-1: negative and 1: positive). Finally, we trained our NLP models on cleaned review text data that was balanced with respect to the label class. Learned word embeddings were used to transform raw test text for input into a series of classification models. To check the robustness of our models, we used trained NLP models to transform text from shifted distributions and then used the trained binary classifiers to predict sentiment. We monitored the efficacy of these models using two primary metrics: classification accuracy and ROC AUC, which describes how our models balanced the tradeoff between true positive rate and false negative rate. We compared data from shifted distributions to the metrics of each model trained and tested on the cleaned text data.

#### 2) How We Have Addressed Feedback From the Proposal Evaluations

We have gotten positive feedback on our previous milestones. We have asked about best ways to approach this NLP challenge, and we’ve gotten some ideas that we incorporated into the structure of our paper, including building a baseline model and then constructing additional more complex models that improve the evaluation metrics on top of the baseline. The most substantial feedback that we got in milestone 2 was the suggestion that we incorporate more significant hyperparameter tuning using grid search. We have incorporated this into each phase of the project as We have incorporated hyperparameter tuning and streamlined our datashift processes after evaluations.

#### 3) Background

For the baseline model, our work closely references the Kaggle notebook “Predicting Sentiment and Helpfulness” authored by Eugen Anghel[[2]](#footnote-1). Anghel conducted a simple text processing, followed by vectorizing the documents with 1-4 n-grams. Then, an embedding matrix is created using TF-IDF methodology. This embedding matrix is then used in logistic regression to predict sentiment. Some improvements considered include: more robust text cleaning, using full review text instead of summary text for training, incorporation of random up-sampling of minority class, and use of Multinomial Naive Bayes classifier for prediction.

In the next phase, to explore a more complex NLP model, we used “Amazon Fine Food Reviews: Sentiment Analysis”[[3]](#footnote-2) by Chirag Samal as the base for the “Word2Vec + LSTM” model. Unlike the author, we explored the possibility of training our own Word2Vec embedding, instead of using pre-trained embeddings. For training the embeddings, we referenced “Gensim Word2Vec Tutorial”[[4]](#footnote-3) by Pierre Megret and “word2vec and random forest classification”[[5]](#footnote-4) by Arunava.

Unlike most Kaggle notebooks where authors do not explore the effect of hyperparameters on model performance, we investigated the effect of tuning some hyper-parameters in logistic regression, Multinomial Naive Bayes classifier, LSTM, TF-IDF, and Word2vec. Moreover, instead of using one single metric to evaluate the performance of the models, we consider AUC, accuracy, and F1 of the test set. Lastly, we tested the robustness of the models by introducing data shifts in the test set, where summary and reviews with 50%, 75%, and 90% random dropouts are used to evaluate the performance of the models.

#### 4) Summary of Our Contributions

**4. 1 - Implementation contribution(s):**

Pre-Processing:

First, in order to reduce noise in the review text, the following text pre-processing steps were included: lemmatization, tokenization, and removal of stopwords, punctuation, hyperlinks, and special characters. This step is performed on both the full review text and summary text. The target column is simplified into +1 positive and -1 negative as described in the previous section. Before training the models, a random up-sampler was used to handle imbalance data, where minority class (target = -1) is up-sampled to match the majority class.

Modeling:

Then, three text embedding techniques were explored, TF-IDF, Word2Vec, and pre-trained BERT. Lastly, logistic regression, Multinomial Naive Bayes classifier, and LSTM were used to compare the performance of various classification techniques.

**4.2 - Evaluation contributions:**

The performance of all models was compared, followed by the performance of these models after significant data shifts: word dropout and summary text instead of full review text. The aim of this comparison was to determine the robustness of the implemented models.

#### 5) Detailed Description of Contributions

**5.1 - Implementation contributions:**

**5.1.1 - Text cleaning:**

Before we could pass text data into our models, the review text had to be cleaned. Our review of existing implementations of sentiment analysis procedures showed that authors typically do not perform extensive text cleaning. We opted to set a stricter threshold for a word to be deemed ‘cleaned’. To this end, we removed the following elements from the review texts: hyperlinks, digits, apostrophes, punctuation, and stopwords. Additionally, we converted all words to lowercase and performed lemmatization to truncate or swap words into their base forms.

After performing this rigorous text cleaning, we dropped duplicates from the data set as well as any rows that contained no review text after cleaning.

Using the cleaned text data, we generated wordclouds to visualize the words used with greatest frequency and grouped on sentiment class (positive or negative) (**Fig. 2**).



Figure 2.

Based on the wordclouds generated, we could not determine differences between sentiment classes. Thus, the models described here are likely making sentiment classification decisions based upon much more nuanced understandings of the text data than the macro-view of the wordclouds.

**5.1.2 - TF-IDF**

We began our comparison of NLP and classification models by defining our baseline performance. To do so, we trained a TF-IDF count-vectorizer and logistic regression model. To determine the optimal parameters for TF-IDF, we performed Grid Search cross validation on different ngram\_ranges.

The relative performance of the vectorizer with each parameter was determined by coupling the cross validation process with a OLS-logistic regression model. After performing cross validation using this pipeline, we observed that the optimal configuration for the ngram\_ranges parameter was (1,2), (which means unigrams and bigrams). This optimized model was used as our baseline.

**5.1.3 - Logistic Regression**

We next sought to find the optimal parameters for the logistic regression model used to make the sentiment prediction given the best TF-IDF model parameters discovered. Again, cross validation was performed, this time on the regularization parameter, C. Furthermore, the model regularization was set to the L2-norm. From the scikit-learn documentation[[6]](#footnote-5), the regularization parameter, C, is inversely proportional to the regularization strength. A range of values were investigated which corresponded to very strong regularization up to very weak regularization. Results will be summarized in the contribution section.

**5.1.4 - Multinomial Naive Bayes**

In our review of the literature surrounding sentiment analysis, we found that Naive Bayes Classifiers are commonly used. In particular, Mutlinomial Naive Bayes has been shown to perform well in sentiment classification tasks[[7]](#footnote-6). As before, we performed GridSearch cross validation to tune this model; the hyperparamenters of the sklearn implementation are limited[[8]](#footnote-7), but the ‘fit\_prior’ parameter, which is a boolean value telling the model whether or not to learn class prior probabilities, and the ‘alpha’ parameter, which controls additive smoothing of the model to handle the issue of zero probability common to Naive Bayes. After cross validation, the best parameters were chosen and used to refit a new model to the entire training set. This trained model was used to test on the held out review texts as well as shifted test sets.

**5.1.5 - Word2Vec**

Next, we explored another word embedding method Word2Vec, which aims to capture the semantic closeness of words. The two parameters we explored were *min\_count* and *window*. The parameter *min\_count* denotes how many times the word should appear in the entire training corpus in order to be considered a vocabulary word in the model. The values explored were 10, 20, and 50. The parameter *window* is the maximum distance between the current and predicted words in a sentence. The values tested were 2 and 4. Unlike the previous model, since scikit-learn's GridSearch API for Word2Vec has been deprecated, we manually constructed 6 models and compared its performance. The Word2Vec embedding was coupled with OLS logistic regression to predict review sentiment.

**5.1.6 - LSTM**

With the best result from Word2Vec, the trained embedding was used to train the LSTM model. Again, it is expensive to tune the numerous hyper-parameters available for an LSTM model. We explored one hyperparameter *SpatialDropout1D(x)*. Unlike the traditional dropout technique that considers element-wise random dropout, *SpatialDropout1D(x)* randomly drops out a dimension of the feature map of all the channels. SpatialDropout is an effective means of encouraging the learning of independent feature maps[[9]](#footnote-8). We considered values [0.2, 0.3, 0.4].

**5.1.7 - BERT**

An additional model we tried running was a classification model using a pre-trained version of BERT: BERT\_base\_cased. This model was trained on BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia (excluding lists, tables and headers)[[10]](#footnote-9). It takes into account if words are UPPER or lowercased, since we believe it is important in this context as a word like “GOOD” can have a more significant meaning than “good”. We intended to use this state-of-the-art technique as a way of having an additional baseline to compare our other, more fine-tuned models on. This notebook from Kaggle titled “Sentiment Analysis using BERT”[[11]](#footnote-10) was adapted for our dataset.

**5.2 - Evaluation contributions:**

**5.2.1 - Tuning TF-IDF n-gram parameter**

Using a logistic regression with standard parameters, Grid Search with cross validation was applied to try out different ngram ranges. The best ngram range was (1,2), which includes unigrams and bigrams, contrasting with our expectations (the bigger the ngrams, the more context and the better the predictions), but it appears that in this case having less context around each word proved to be more informative. This improves the performance of the notebook we were basing our modeling on, since they used (1,4) for their ngram ranges (**Fig. 3**).

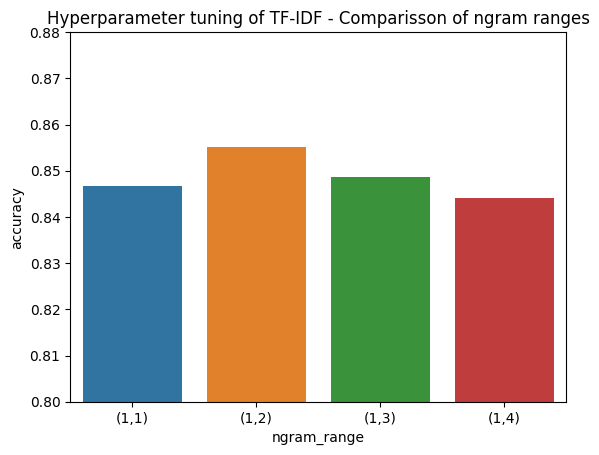


Figure 3.

Other hyperparameters of TF-IDF could have been analyzed and tuned, such as maximum and minimum document frequency. Due to time and resources constraints we decided to use the default parameters.

The accuracy of our baseline model is therefore 0.856

**5.2.2 - Tuning Logistic Regression on optimal TF-IDF embeddings**

Using the best parameters for TF-IDF defined above, count-vectorization was performed and used as input to cross validation of the Logistic Regression model. As mentioned above, the regularization parameter was tuned in conjunction with L2. This resulted in better-than-expected performance given the models used. Using accuracy and ROC AUC metrics, we saw that the best regularization parameter for logistic regression was 10 (Figure 4). More regularization (as C → 0) resulted in poorer performance, while the improvements gained from less regularization (as C → infinity) diminished above C=10. After refitting to the entire training set using the best parameter, the accuracy on held out test data was: 0.895 and ROC AUC was 0.856[[12]](#footnote-11). It is interesting to note that while both the training and validation curves for accuracy and ROC AUC show near optimal results, that the final test accuracy was much lower. This may suggest that these plots show overfitting to the training data, though the fact that the cross validation metrics do not show a similar decrease.

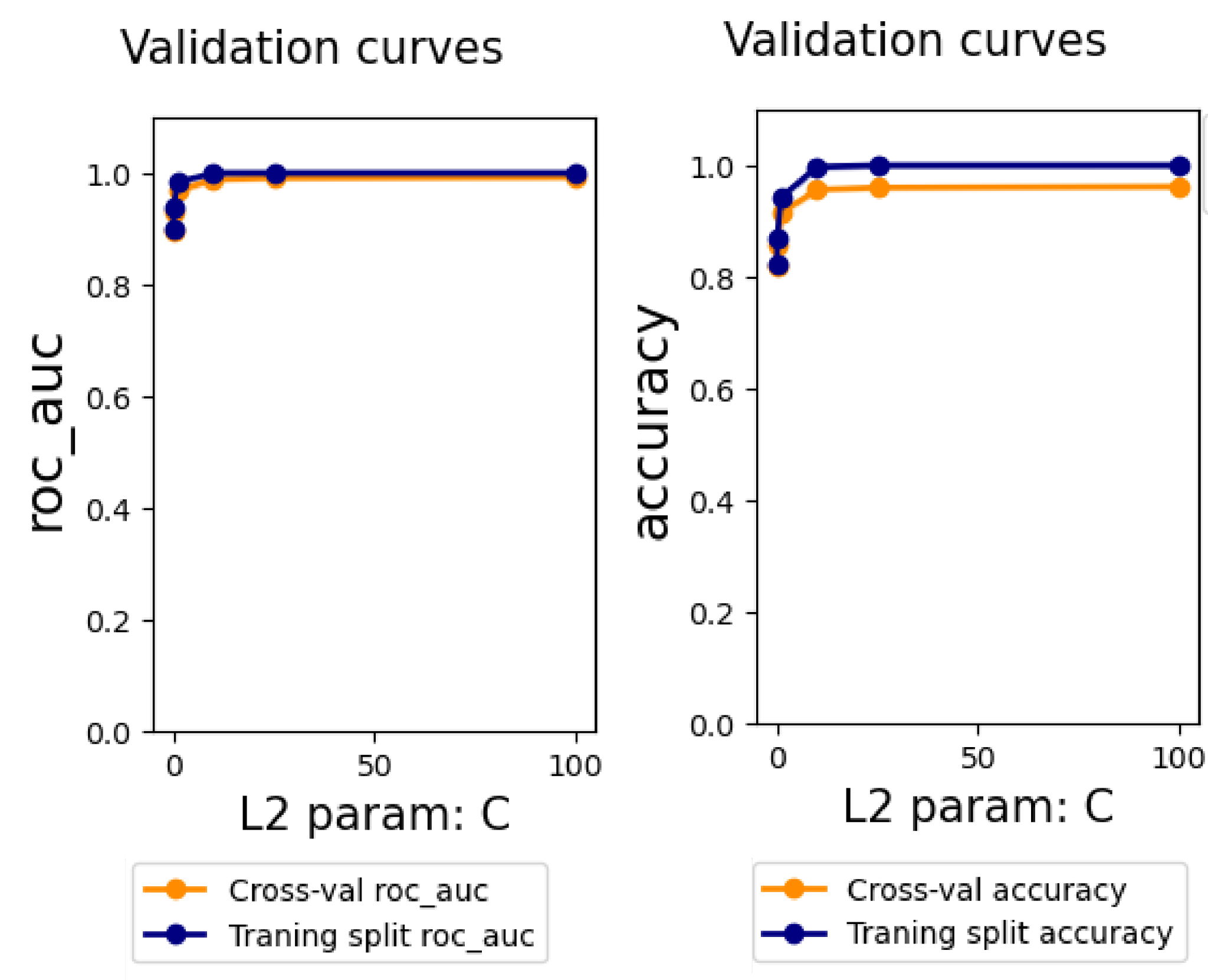


Figure 4.

**5.2.3 - Pairing TF-IDF with Multinomial Naive Bayes**

Logistic regression did surprisingly well, but as mentioned previously one of the commonly used classification models used for sentiment analysis is Naive Bayes. More specifically, under naive bayes we make the assumption that the probability of obtaining the current document (in our case review text) is in which is represented as a set of features . Naively, we make the assumption that the features are independent and thus that probability can be calculated as the product of individual conditional probabilities: [[13]](#footnote-12). However, as before, we were not able to find quality examples of notebooks in which Multinomial Naive Bayes was tuned. We also noted long training times for the logistic regression model described above and were curious if the naive assumption of MNB would result in faster training.

Indeed, upon tuning the two trainable parameters under sklearn’s implementation of MultinomialNB, we saw significantly faster training times (data not shown); however, performance actually declined slightly in comparison to logistic regression (**Fig. 5**). The best parameters were determined to be ‘alpha’=0.1 and ‘fit\_prior’=True. After refitting to the entire training set using the best parameter, the accuracy on held out test data was: 0.819 and ROC AUC was 0.874. Once again, we see far superior results in the cross validation metrics than in the test set. Given this is a recurring pattern, we hypothesize that the random split that was used to produce the training and testing sets happened to partition the data such that the samples in the test set were notably more difficult than in the training set.

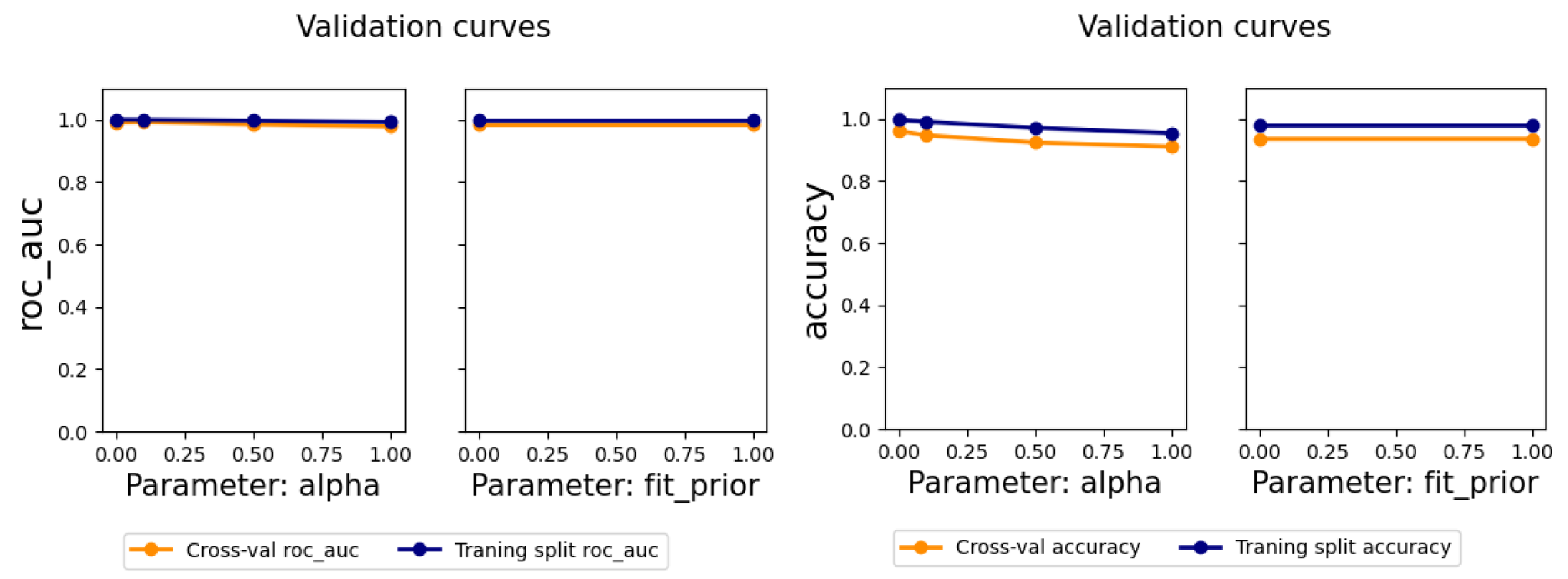


Figure 5.

**5.2.4 - Word2Vec**

Next, unlike TF-IDF which only captures the significance of words in a corpus, Word2Vec aims to capture the semantic closeness of words. With the hyper-parameters tested, we can observe the *window* of 4 on average performed better than *window* of 2. With being able to use more words in the prediction, the model is able to better capture the context. Moreover, by having more words in the vocabulary (with a lower *min\_count*), the model generally performed better. However, the trend is not conclusive. Overall, we found that the best parameter combinations would be *min\_count* = 10 and *window* = 4 (**Fig. 6)**.

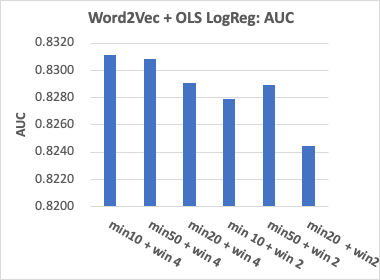
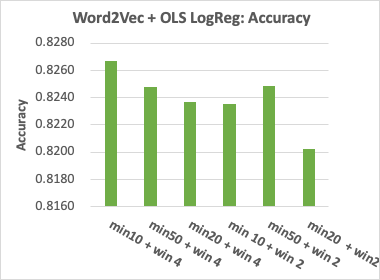
 

Figure 6.

Interestingly, Word2Vec embeddings with logistic regression did not perform better than TF-IDF with logistic regression. It is possible that we have not found the optimal hyper-parameters (**Fig. 7)**. It will be good to set up a more extensive search for optimal hyper-parameter combinations.

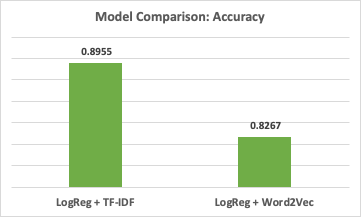
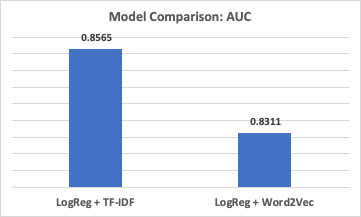


Figure 7.

**5.2.5 - LSTM**

Next, using the best Word2Vec embedding obtained from the previous section, we can already observe a significant improvement with the incorporation of bi-directional LSTM (**Fig. 8**), which uses both previous words to predict the next word and the latter words to predict the previous word. With almost identical LSTM structure but with our own Word2Vec embedding, we were able to obtain an accuracy of 0.90 on the test data; whereas the Kaggle notebook we referenced only obtained an accuracy of 0.83 on the train data.

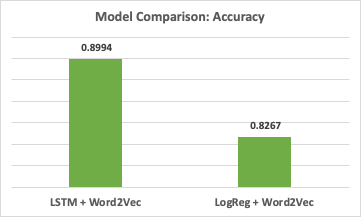
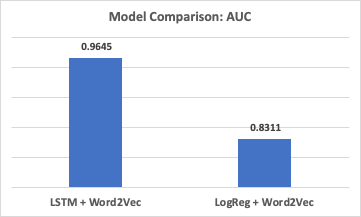


Figure 8.

With the *SpatialDropout1D( )* hyperparameter, we can observe that while it can improve the model performance (**Fig. 9**), the trend is not conclusive. It will be interesting to compare the effect of the *SpatialDropout1D(x)* layer compared to the *dropout* parameter within the LSTM layer.

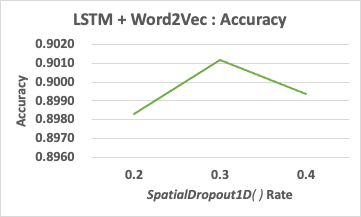
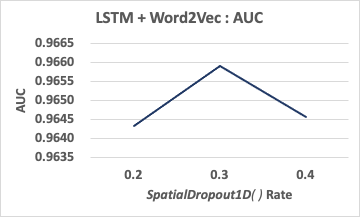


Figure 9.

**5.2.6 - BERT (‘bert-base-cased’)**

When applying BERT to our dataset we got a test accuracy of 0.8712 and AUC score of 0.8205. Looking at the results it looks like our model is not very good at predicting reviews with negative sentiment (**Fig. 10**). While this is an adequate accuracy, it is a lower result than expected, and we believe it was due to three reasons:

1. In order to manage the memory requirement from notebooks we had to limit our entire database to 50,000 reviews (12.5%). With more resources available the model should be trained on the entire dataset.
2. No hyperparameter tuning was applied due to the extreme time duration of processes. By adjusting hyperparameters and improving the architecture of the model better results could be obtained.
3. The pre-trained model was trained on books and wikipedia articles that can be quite different from the language used on user reviews. A different pre-trained BERT model should be used, maybe one trained on Google News which could have more “sentimental” vocabulary.

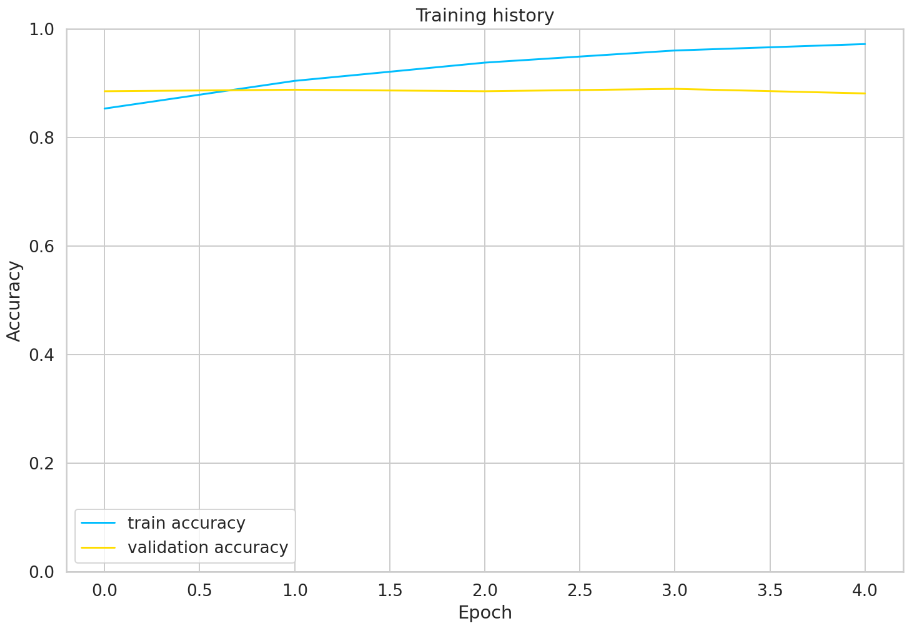


Figure 10.

**Effect of Data Shifts:**

The effect of data shift was rather interesting. We hypothesized that “summary” text would generally yield lacking results due to the limited number of words that the model can use to predict the sentiment. This hypothesis is reflected in the decreasing AUC and accuracy that we observe in the random dropout samples. However, surprisingly, “summary” text was able to generate similar AUC and accuracy results as random dropout of 50% of the full review text (**Fig. 11, Fig. 12**). In retrospect, while the summary text is generally concise, customers often use precise words to capture their feelings about the products, such as “best food”, “worst product”, etc; therefore, despite having limited words, overall, summary text is able to provide enough context for the models to make good predictions.

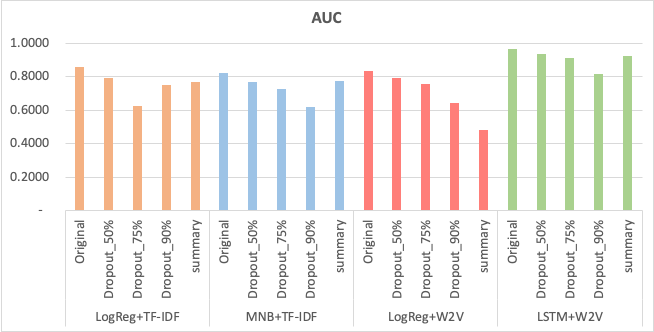


Figure 11.

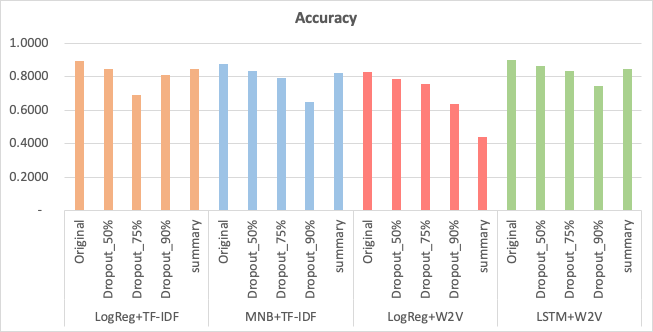


Figure 12.

#### 6) Compute/Other Resources Used

In order to complete this project, we utilized AWS resources for data storage as well as model training and testing. Raw reviews data was cleaned in Google Colab, which is then exported and uploaded to an Amazon S3 bucket in order to be used with Amazon SageMaker. We trained, tuned, and tested our models on Amazon Sagemaker notebook instances.

#### 7) Conclusions

The “Amazon Fine Food Reviews” dataset was uploaded at least 6 years ago. Over the years, many have attempted to analyze the data with a variety of NLP models. Upon closer inspection of the Kaggle notebooks, one can soon realize that most notebooks explore only 1 model with limited hyperparameter tuning. Reading over our original proposal, we can observe that we have come a long way in terms of understanding the different aspects that can influence the performance of a sentiment analysis model, from the rigor of the text processing, to the word embedding technique, to the final classification model.

We initially only planned on exploring the use of TF-IDF and LSTM for conducting sentiment analysis. However, overtime, we learned of the various models that can be implemented, and were able to deploy a workable Word2Vec, Multinomial Naive Bayes classifier, and BERT. While a lot of resources are available on Kaggle, we still spent a lot of time updating the code (some packages were updated) and fitting the code to our dataset. In addition, as the model complexity increased, the training time also increased. Also, since the more complex models were not compatible with the scikit-learn GridSearch method, we were unable to experiment with more hyperparameter combinations.

In the future, one can explore the use of pre-trained BERT embeddings that may be more closely aligned with review data, such as Google news[[14]](#footnote-13). Given the polarized nature of news content, it is possible that a model trained on this corpus of data would perform better than the BERT model we picked initially. Further, future work can be focused on hyperparameter tuning with the existing models as most of the complex models have myriads of hyperparameters that can be explored.

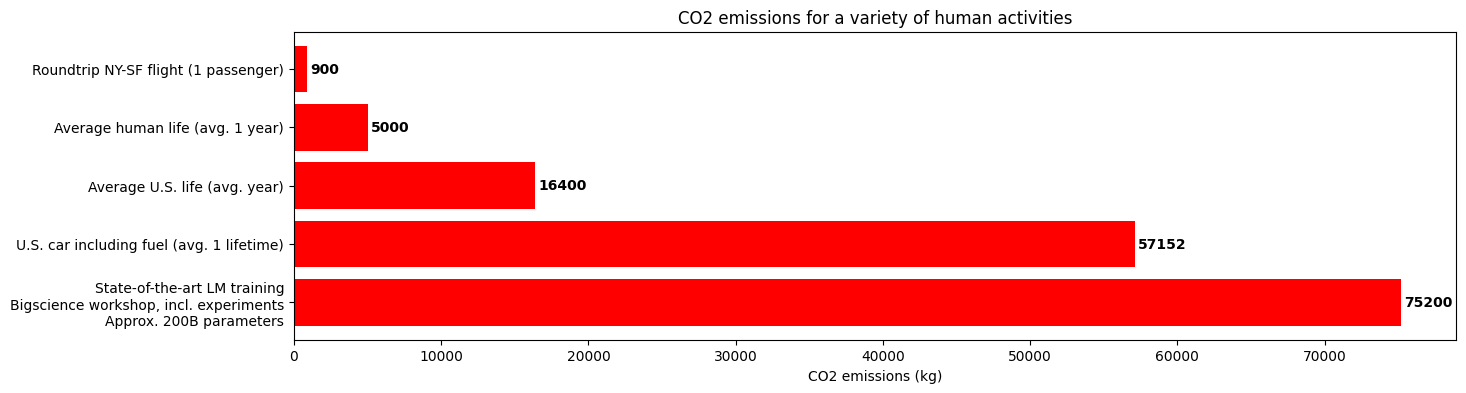
We hope that this report improves upon the space of sentiment analysis projects and provides a systematic interrogation of several common natural language processing techniques and complementary classification models that can serve as a guide to future data scientist students conducting sentiment analysis projects. There are innumerable methodologies to explore in this space and documenting the factors that influence model performance in sentiment classification settings will allow us to gain a better understanding of how machines interpret language.

**Social Impact:**

NLP models are significantly vulnerable to bias in data. They can relate certain words with context and sentiment from biased data that may cause an impact on real people. These models could treat certain words or language expressions related to vulnerable populations as having a negative sentiment, which can impact the performance of users and businesses that buy and sell these food products if implemented unchecked.

**Environmental impact:**

State of the art transformers and other deep learning techniques release a significant amount of carbon dioxide into the atmosphere. Sharing and using pre-trained models like word2vec and BERT helps reduce the impact of this activity. The following graph shows CO2 emissions of ML training versus other human activities[[15]](#footnote-14):



**8) Roles of team members (1-2 sentences each):**

**Federico Cimini:** Federico helped build the initial notebook for data cleaning and preprocessing, and the word clouds. He tuned hyperparameters of TF-IDF and ran the BERT model.

**Sam Thudium:** Sam produced the visualizations for label distributions and text summary statistics. He also set up the AWS environment, tuned hyperparameters for logistic regression and multinomial naive bayes, and wrote helper functions used in several notebooks.

**Liang-Yun Cheng**: Liang worked on using various Kaggle notebooks to set up the Word2Vec and LSTM model. She also explored the effect of various hyperparameters on each of those models.

1. Amazon Fine Food Reviews; <https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews> [↑](#footnote-ref-0)
2. Eugen Anghel: <https://www.kaggle.com/code/eugen1701/predicting-sentiment-and-helpfulness> [↑](#footnote-ref-1)
3. Chirag Samal: <https://www.kaggle.com/code/chirag9073/amazon-fine-food-reviews-sentiment-analysis> [↑](#footnote-ref-2)
4. Pierre Megret: <https://www.kaggle.com/code/pierremegret/gensim-word2vec-tutorial> [↑](#footnote-ref-3)
5. Arunava: <https://www.kaggle.com/code/arunava21/word2vec-and-random-forest-classification> [↑](#footnote-ref-4)
6. Sklearn log reg: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html> [↑](#footnote-ref-5)
7. Multinomial NB paper: <http://paper.ijcsns.org/07_book/201903/20190310.pdf> [↑](#footnote-ref-6)
8. Sklearn mulitnomialNB: <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html> [↑](#footnote-ref-7)
9. <https://arxiv.org/pdf/1411.4280.pdf> [↑](#footnote-ref-8)
10. <https://huggingface.co/bert-base-cased> [↑](#footnote-ref-9)
11. <https://www.kaggle.com/code/prakharrathi25/sentiment-analysis-using-bert> [↑](#footnote-ref-10)
12. See model results file for model: https://github.com/sthudium25/CIS5190-AmazonFoodReviews [↑](#footnote-ref-11)
13. “Speech and Language Processing”, Daniel Jurafsky & James H. Martin. 2023. https://web.stanford.edu/~jurafsky/slp3/4.pdf [↑](#footnote-ref-12)
14. Google News BERT pre-trained: https://huggingface.co/fse/word2vec-google-news-300 [↑](#footnote-ref-13)
15. <https://huggingface.co/learn/nlp-course/chapter1/4> [↑](#footnote-ref-14)