Source code: https://github.com/drewherron/review-sentiment

Connor Baron-Williams
jcwill@pdx.edu

Drew Herron
dherron@pdx.edu

Nate Ruble
nruble@pdx.edu

Sentiment Analysis of Amazon Reviews

Author Contributions

Drew created the overall project template, including the 'review_sentiment' (main), loading, and plotting modules, and implemented a Bidirectional Encoder Representations from Transformers (BERT) Natural Language Processing model. Nate implemented a multilayer perceptron model. Connor implemented a Naïve Bayes classification model. All authors collaborated to make additional updates to the project template and reported methods and results for their respective implementations.

Introduction

Sentiment analysis is a critical component in Natural Language Processing (NLP), which involves the process of extracting or finding meaning from a given text. It is a process that is essential to today's development and advancement of intelligent machine systems pertaining to, among other functions, virtual assistants and chatbots, search engines, human-computer interaction, and, notably for this research, business insights. Businesses can utilize sentiment analysis to analyze textual data, such as customer reviews, to make intuitive decisions with respect to products or services. In the research conducted, three different methods, BERT NLP, Multilayer Perceptions, and Naïve Bayes classification, were trained and tested on a database of no more than 200,000 Amazon reviews (Ni, 2018) to gain a better understanding of each technique's ability to predict a customer's product rating by leveraging sentiment analysis (Herron et al., 2023).

Naïve Bayes Classification (NB)

The implementation of this method involved the use of two separate techniques, classification using the scikit-learn library's built-in Multinomial NB classifier and classification using the Natural Language Toolkit (NLTK) library's built-in NB classifier (Mogyorosi, 2022; Kargin, 2021). In utilizing sentiment analysis in conjunction with Multinomial NB classification, the training and testing dataset were transformed into a data frame (from the pandas library) of word count occurrences with respect to some chosen word features for each review. These word features were identified as the most commonly occurring words in the dataset. The data frame of word frequencies for each review were noticeably low, so they were retained as discrete values, demonstrating the beneficial use of scikit-learn's Multinomial classifier as opposed to the library's other Gaussian and Bernoulli classifiers, which rely on normalized or binary data, respectively. A similar process was conducted for NLTK's NB classification, although the training and testing dataset were transformed into a set of True or False occurrences with respect to the chosen features for each review, which is a required input for training and testing these types of classifiers.

The NLTK library also allows for the tokenization of specific parts-of-speech within a given text. In managing this process, unnecessary stop words (i.e., “the”, “is”, “a”, etc.) were removed, words were lemmatized to their root form (i.e., “running” becomes “run”), and the user was granted the choice to parse three different groups of parts-of-speech to be used as the word features. Word counts or occurrences (i.e., True/False) of these features could then be determined for each review. Parts-of-speech grouping options were set to adjectives and adverbs only, nouns, verbs, and interjections only, or a combination of all these listed speech types. Of these grouping options, the user was able to choose up to 1,000 of the most commonly occurring words to be used as the features (the number of features was limited to 1,000 due to system memory issues). Since NB classification assumes independence among features, the purpose behind these groupings was to see if either method's technique would perform better with respect to a part of speech. Adjectives and adverbs were chosen as a feature group since these types of words typically carry meaning in relation to description or feeling, which could be a good indicator for predicting a review's rating. Nouns, verbs, and interjections, as well as combining all the terms, were chosen with the intention of acting as contrary terms to measure whether feature type selection had any importance in the implementation of these two NB classification techniques.

These two built-in NB techniques were tested on two different formats of the dataset, balanced (i.e., each rating label has the same number of reviews) and unbalanced. The unbalanced dataset consisted predominantly of reviews with 5-star ratings (approximately 68%), so using the balanced dataset led to more insightful results. The two techniques were also tested on varying numbers of the three feature type selections, specifically 10, 50, 100, 500, and 1,000. Additionally, trained models from these executions, for both techniques, were then tested on other Amazon product category reviews (i.e., the models were trained on Amazon's Appliances reviews but tested on Amazon's Electronics reviews). With each decrease in the number of features chosen, all metrics for both techniques on all dataset and feature types typically dropped gradually. When trained models were tested on other Amazon review categories, accuracies dropped about 10 percentage points to minimums no less than

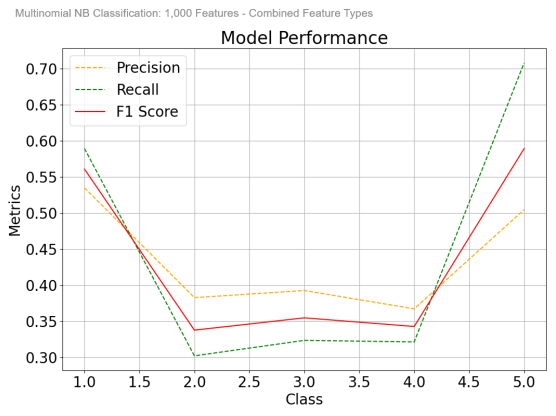
20%. Applied on the unbalanced dataset, collective accuracies ranged from 65.8% to 70.7% whereas accuracies from the balanced dataset yielded a range from 24.8% to 45.0%. While an accuracy of 70.7% was successfully found using 1,000 adjective and adverb features with the Multinomial NB classifier, it being generated from the unbalanced dataset was not necessarily indicative of a good model performance. These metrics were tested on 20% of the dataset (40,000 and 20,734 reviews for the unbalanced and balanced dataset, respectively). The following table shows top performance metric findings in relation to the accuracies and compares each technique on the balanced dataset.

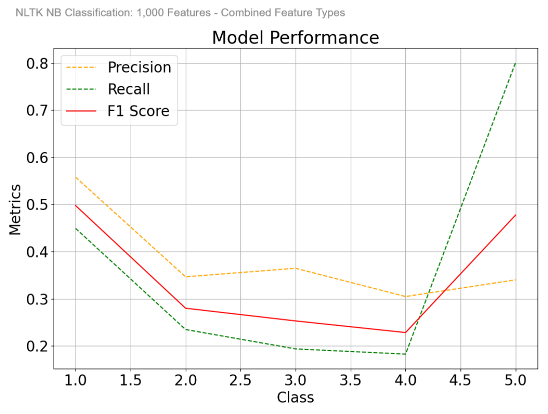
Feature Type	# of Features	Model Type	Accuracy	Precision	Recall	F1 Score
Adjectives/ Adverbs	1,000	MNB	**43.8%**	**42.4%**	**43.8%**	**42.3%**
		NLTK NB	37.2%	38.2%	37.2%	34.6%
	10	MNB	27.3%	30.0%	27.3%	25.4%
		NLTK NB	**27.8%**	**27.7%**	**27.8%**	**22.9%**
Nouns/ Verbs/ Interjections	1,000	MNB	**41.2%**	**40.1%**	**41.2%**	**40.2%**
		NLTK NB	35.9%	37.1%	35.9%	33.2%
	10	MNB	24.8%	25.5%	24.8%	19.1%
		NLTK NB	**26.8%**	**21.0%**	**26.8%**	**20.7%**
Combined	1,000	MNB	**45.0%**	**43.7%**	**45.0%**	**43.8%**
		NLTK NB	37.4%	38.2%	37.4%	34.7%
	10	MNB	27.4%	29.6%	27.4%	24.9%
		NLTK NB	**28.1%**	**29.1%**	**28.1%**	**24.0%**

As the table indicates, the Multinomial NB classifier performed better than the NLTK NB classifier, in terms of accuracy, on every feature type when executed with 1,000 features (all top comparative accuracies in bold). On the other hand, while it did not result in the NLTK NB classifier's highest accuracies, when applied to 10 features in all feature categories, it performed better than the Multinomial NB classifier. Notably, the overall recall matches the accuracy in all experiments, which, if executing correctly, could suggest that the models are correctly predicting instances for each class without a significant bias toward any particular class. Conversely, the overall precision in all experiments is less than 50%, which implies that the models did not perform well in correctly identifying positive instances for a given classification task. This is clearly seen in the resulting accuracies, where the highest accuracy, seen on 1,000 combined features with the Multinomial NB classifier, was only 45.0%. Otherwise, many of the accuracies in the 20% range illustrate an unlearned model making random predictions over the five possible star ratings.

MNB		**Prediction**				
Actual		1	2	3	4	5
	1	**2471**	876	384	192	268
	2	1159	**1245**	895	460	357
	3	552	720	**1343**	945	587
	4	232	271	595	**1321**	1687
	5	203	138	201	676	**2956**

NLTK NB		**Prediction**				
Actual		1	2	3	4	5
	1	**1827**	555	208	178	1300
	2	863	**978**	562	445	1318
	3	326	692	**813**	736	1632
	4	139	392	467	**743**	2328
	5	121	206	180	337	**3388**





The above confusion matrices and charts for both techniques, representing when the models' accuracies were highest with the balanced dataset (1,000 features with combined feature types), illustrate specific performance metrics for each rating classification. The charts tell a consistent U-shaped story across all model experiments with respect to precision, recall, and F1 scores for each rating class prediction. All metrics dipped when classifying ratings of 2, 3, and 4, suggesting the overall randomness of each model's prediction capabilities. However, each model always depicted higher metrics when predicting class ratings of 1 and 5, with a balanced or unbalanced dataset. Overall, while scikit-learn’s Multinomial NB classifier typically performed better than NLTK’s NB classifier, performance in all experimental models showed poor performance and a tendency to seemingly classify ratings randomly as opposed to intelligently. Furthermore, experimenting with specific parts-of-speech features showed that the models performed best when using the combined feature types, which uncovered that focusing on adjectives and adverbs alone, at least in the context of this research, demonstrated no significant benefit in improving sentiment analysis.

Multilayer Perceptron (MLP)

In modeling an MLP, a small neural network was designed to take the frequency of meaningful words in a review and output the suspected rating given. The network was created to have two hidden layers of 100 neurons each. It was made using PyTorch and trained using cross-entropy loss as its loss function and stochastic gradient descent to decide the new weights. For the activation function, I chose ReLU in order to get sparse activation across the network. I decided this was important because the majority of words being fed to it would be specific to only certain types of reviews. For example, a reviewer is unlikely to say the word “Horrible” in a 5-star review. In addition, I chose to have hidden layers to get around the linear relationship issues that might arise with meaningful words that aren't strictly good or bad such as "much" or "really" which can affect a review in either direction depending on the other words in it.

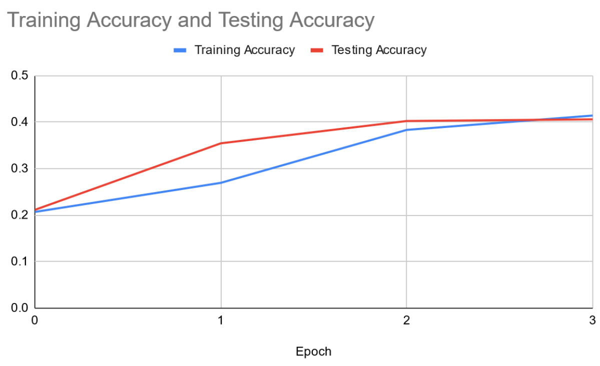
In order to find the best hyperparameters for this network I experimented with different configurations. I tried learning rates of 0.1, 0.01, 0.001, 0.0001, and even 0.00001. In addition, I varied the amount of meaningful words being fed to the network. I experimented with values of 50, 100, and 200 words. Ultimately the best hyperparameters I found were a learning rate of 0.0001 and 200 meaningful words. There seemed to be a direct relationship between the number of words fed to the network and the quality of its predictions. As for the learning rate 0.0001 seemed to consistently find the global minimum without taking more time than necessary, converging in only 3 epochs due to the small size of the model.

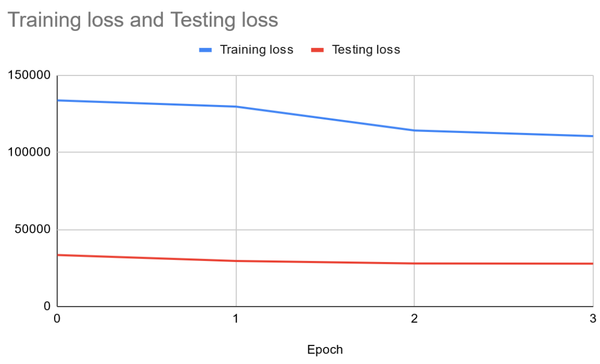
However, regardless of the hyperparameters the final results were ultimately disappointing. The network only managed to achieve a maximum accuracy of 40%. Which while better than the 20% of random guessing, is nowhere close to being actually useful in sentiment analysis. Below are the final test accuracy and loss, a confusion table displaying the results, and charts of the accuracy and loss during both training and testing:

	Prediction					
Actual		1	2	3	4	5
	1	2334	1463	734	317	268
	2	570	839	619	266	119
	3	298	710	1095	543	156
	4	173	418	848	1236	623
	5	733	785	899	1772	2916

Final Test Accuracy: 0.40609626700106105

Final Test Loss: 27798.469989474863





In the end, the results for the custom-made MLP were disappointing. I believe this is due to two factors, the size of the network, and the amount of data it was given. I think that the network was ultimately too simple, it only had 200 neurons total between input and output. Hardly enough to deal with the myriad of complex relationships between words that form our language. In addition, I think that the data it was given was simply not enough. Had the network been given far more word frequencies in addition to being more complex it probably would have performed much better.

BERT

BERT is a transformer-based machine learning model released by Google in 2018. It was originally available in two model sizes: BERT$_{BASE}$ and BERT$_{LARGE}$, although more sizes have since been developed. The sizes of BERT used in our experiments were:

- BERT$_{BASE}$: 12 encoders, 768 heads
- BERT$_{MEDIUM}$: 8 encoders, 512 heads
- BERT$_{SMALL}$: 4 encoders, 512 heads
- BERT$_{MINI}$: 4 encoders, 256 heads

- BERT$_{TINY}$: 2 encoders, 128 heads

BERT is a pre-trained model, and we used our Amazon review data to fine-tune the model - an example of *transfer learning*. For the first round of training we used BERT$_{BASE}$, with 100k reviews and 5 epochs. This training took 9 days on a relatively powerful desktop computer, with no (CUDA-compatible) GPU. Training time turned out to be the most limiting factor in the BERT experiments, and eventually we moved training to a GPU-based VM on Google Cloud, where training was ten times faster. Early results were not very promising, and throughout all of these experiments the final accuracy never broke 60% with BERT. However, the confusion matrices did reveal that things weren't as bad as they appeared. Even on a smaller training, as in this case training 5k reviews for 5 epochs on BERT$_{TINY}$:

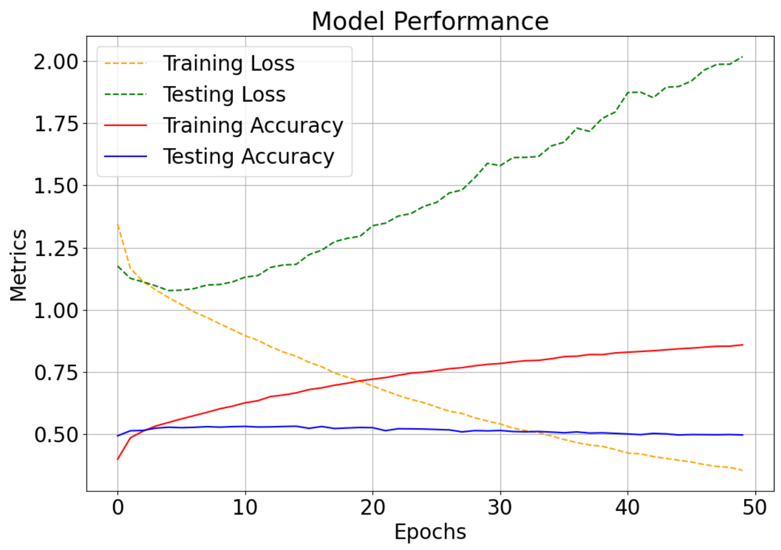
		Prediction				
		1	2	3	4	5
Actual	1	**118**	40	22	6	15
	2	45	**74**	70	14	9
	3	22	41	**88**	40	19
	4	2	6	22	**86**	79
	5	2	2	10	39	**129**

We can clearly see that the model is making good predictions, although in calculating our accuracy value being off by one star is as bad as being off by four. In this case, the final accuracy in testing was only 0.495. If we were to include correct predictions within a distance of 1 star, we arrive at an accuracy of 0.871.

This “within-one” accuracy is still very meaningful in terms of sentiment analysis, and is similar to a positive/neutral/negative predictor. Even humans could have trouble predicting whether a review text represents, for example, a 3-star rating or a 2-star rating.

More data and a larger model did seem to have the strongest correlation with more accurate predictions. BERT$_{MEDIUM}$ performed better than BERT$_{SMALL}$, which performed better than BERT$_{MINI}$, etc., and on the same models training on 15k reviews performed better than training on 10k reviews. Training BERT$_{TINY}$ on 100k reviews resulted in a “within-one” accuracy of 0.91, while BERT$_{BASE}$ hit 0.93 on 25k reviews after the same number of epochs.

There also didn’t seem to be any benefit from training beyond 5 epochs, at least with these hyperparameters. Longer trainings would show classic signs of overfitting, as seen in this plot (with continuously falling training loss and increasing testing loss):



To test that our models would generalize well and weren’t overfitting to one particular type or style of review, BERT$_{SMALL}$ was trained on 50k reviews in the “Appliances” category, and then tested on reviews from the “Luxury/Beauty” category. The results showed that our sentiment analysis model would likely perform well across categories (and potentially beyond product reviews):

	Prediction					
Actual		1	2	3	4	5
	1	**1230**	601	109	18	20
	2	400	**1128**	398	51	18
	3	119	556	**999**	229	54
	4	21	69	433	**1001**	556
	5	17	31	58	310	**1574**

This represents a testing loss of 1.075, a testing accuracy of 0.59300, and an accuracy within 1 star of 0.9415. This is especially surprising considering BERT never reached these accuracies when testing within the category trained on.

Training on an unbalanced dataset did result in generally higher prediction accuracies, but this doesn't mean the learning was more effective. Unbalanced data includes a disproportionate number of high ratings, so effectively our training data and test data both have a bias that probably won't help the model's generalizability.

We used a batch size of 8 by default, and in our experiments no other batch size was as effective. Smaller batches were nearly as good, but not quite. The program would crash with larger batch sizes.

Using the Adam optimization algorithm, the learning rate for most of these experiments was 2e-5 (0.00002). A few experiments were run with a much higher (although still small) learning rate of 2e-3, and the accuracies dropped to around 0.2, no better than chance.

Due to the slow training, most of these experiments were for comparing various sizes of models, number of inputs, and number of epochs. If we were to devote more time to this, I’d want to pick a very large dataset and a very large model, and vary each training primarily in the learning rate and number of epochs.

Conclusion

Processing data effectively and model training time for sentiment analysis proved to be substantially challenging in the implementation of the Naïve Bayes, Multilayer Perceptron, and BERT models. The Multilayer Perceptron and Naïve Bayes methods both used similar data processing techniques and yielded prediction accuracies in the 40% range, with Naïve Bayes performing only slightly better with the execution of the Multinomial NB classifier. While the built-in models themselves could have led to this poor performance, it is more likely that the amount of data, feature selection, and possibly even word parsing may not have been adequate. Additionally, BERT’s training time took more than a week, restricting the number of experiments that could be conducted. However, while BERT performed slightly better with hard classification in all cases as compared to the other two methods, it demonstrated significant improvements when applied to fuzzy, or within-one star, classification. BERT reached hard and fuzzy classification accuracies as high as 59.3% and 94.2%, respectively, and notably did so when tested on separate Amazon review categories. Overall, BERT, being a pre-trained model with fine-tuning adjustments to process Amazon reviews, proved to perform substantially better than the other two methods and demonstrated its superior capabilities relating to sentiment analysis.

References

J. Ni, (2018). *Amazon Review Data*. Retrieved 2023, from https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/

D. Herron, N. Ruble, & C. Baron-Williams, (2023, November). *review-sentiment: Machine learning with Amazon reviews*. Retrieved December, 2023, from https://github.com/drewherron/review-sentiment

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding’, CoRR, vol. abs/1810.04805, 2018.

P. Bhargava, A. Drozd, and A. Rogers, ‘Generalization in NLI: Ways (Not) To Go Beyond Simple Heuristics’, arXiv [cs.CL]. 2021.

I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, ‘Well-Read Students Learn Better: The Impact of Student Initialization on Knowledge Distillation’, CoRR, vol. abs/1908.08962, 2019.

M. Mogyorosi (2022, September 1). *Sentiment analysis: First steps with Python’s NLTK library*. Real Python. https://realpython.com/python-nltk-sentiment-analysis/

K. Kargin (2021, November 20). *NLP: Tokenization, Stemming, Lemmatization and Part of Speech Tagging*. Medium. https://medium.com/mlearning-ai/nlp-tokenization-stemming-lemmatization-and-part-of-speech-tagging-9088ac068768