Decoding Phone Case Ratings with Sentiment Analysis

Jerry Chang, Sanjana Cheerla, Kara Schatz, Vincent Xin
lchang2@ncsu.edu,scheerl@ncsu.edu,kmschat2@ncsu.edu,vxin@ncsu.edu
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA

1 PROBLEM STATEMENT

Product reviews on Amazon are typically measured by a star rating from 1 to 5, usually followed with a textual review from the reviewer that supports their star rating. When buying a product online, reviews offer consumers a justification on whether or not they want to buy the product. Stars are a succinct way of telling the consumer how other's think of the product, but not all websites use a star system. Therefore, using a classification model to predict the star rating of a review can offer assistance to consumers that need a quick way of judging the sentiment around a product. For our models, we decided to focus specifically on phone case ratings because of the ease of access to a large number of data.

2 RELATED WORK

Natural Language Processing is one of the cornerstones of machine learning (ML). Libraries and methods of information extraction has progressed for over a decade. An article written by YuanHang Xiao et al. used one such library called TextBlob to obtain 2 different types of sentiment values from product reviews: polarity and subjectivity [8]. To specify, polarity represents a negative or positive view of the product, while subjectivity is the analysis of how influenced the text is by personal opinion or factual information. They also used Term Frequency-Inverse Document Frequency (TF-IDF) to obtain their features from the reviews. From their results, they discovered consumers preferred to be subjective to their experience of using a product. Quality descriptors were also a heavy influence on the rating level of the product. When it came to lower star reviews, design functions of the products were typically described to emphasize the essential functions they wish to see in the product.

In another article written by Shivaprasad and Shetty, they analyzed different methods of sentiment analysis and also how previous research has approached the problem on product reviews, specifically with ML versus lexicon-based approaches on feature selection and also classification [7]. In lexicon-based approaches, usually a dictionary or a set of words and phrases are analyzed on the text to gain a deeper understanding of the structure and meaning. In ML, the understanding of text or feature vectors are handled by the algorithm itself. The article explores different ML classification methods such Naive Bayes, Support Vector Machines, and Maximum Entropy. Overall they found that Support Vector Machines had a higher accuracy compared to the rest of the methods, but indicated that Support Vector Machines were more advanced compared to methods like Naive Bayes. Knowing how these different methods could perform on our own data can be beneficial towards achieving a more robust model for product review prediction.

We touched on ML classification, but we also need to consider deriving features from words. Aljuhani and Alghamdi explored the usage of three types of feature extraction techniques in their experiments [1]. Namely Bag-of-words, TF-IDF, and Word-embedding. Bag-of-words involves measuring the frequency of terms in text, while TF-IDF puts weights depending on the frequency of terms, with more frequency usually meaning less weight. We can also adjust specific qualities of the terms with n-grams. N is the measure of the amount of words in each term we want to observe. Word-embedding takes a different approach by creating word vectors that take into context that each word may be in, such as the surrounding words. From their results, they found that the Word-embedding method using Word2Vec obtained the best results with a 79.60% accuracy on a balanced dataset. Since the results had similar percentages between other methods, this means that we are encouraged to find the right method that works best with our dataset.

3 PROPOSED APPROACH

3.1 Novel Aspects

This project involves several novel aspects, which are described in this subsection. We hope to introduce a curated data set and answer the problem statement by explaining why a certain trend occurs.

3.1.1 Creating Dataset. One main novel aspect is creating our own data set through web scraping. We created our data set using mobile phone case product reviews from Amazon because we wanted to understand why customers prefer or do not prefer a certain phone case product. The data set creation is detailed in Section 4.

3.1.2 Further Analysis. Another aspect of novelty is our deeper analysis of reviews via sentiment analysis. Through this, we hope to identify how sentiment can be related to the star rating of a review, as well as common features of positive and negative reviews.

3.2 Method

Our proposed solution approach to the problem statement includes the following: dataset curation, model building, and analysis.

3.2.1 Dataset Curation. The first step includes creating a novel dataset as introduced in Section 3.1.1 and explained in Section 4.

3.2.2 Model Building. After researching several model-building methods, we settled on trying out several word embedding and neural network-based models. We would then compare our models and choose the best model to predict future reviews on.

For our *Word Embedding Models*, we pair an embedding method with a classifier. First, the embedding method is used to create word embeddings, and then the classifier is used to classify the reviews based on resulting embeddings. We used four different classifiers: k-NN, Decision Tree, Random Forest, and AdaBoost, and we paired each one with each embedding method described below.

(1) Word2Vec (W2V) [5]: Word2Vec is one of the most commonly used embedding techniques. We created our own library of word embeddings based on our dataset and used that library to create our training and testing word embeddings.
(2) Universal Sentence Encoder (USE) [2]: USE is the most widely used word embedding model in the TensorFlow hub. USE is pretrained, so it has been exposed to a variety of words.
(3) TF-IDF [6]: TF-IDF uses the frequency of words to determine their importance in the sentence. Then word embeddings are generated to reflect the importance.

In addition to *Word Embedding Models*, we will train *Neural Network Models*. A transformer is a deep learning model where every output element is connected to every input element, and the weightings between them are dynamically calculated based on their connection. Our selected transformer models are described below.

(1) XLNet [9]: This is an auto-regressive language transformer that incorporates techniques from auto-encoder models like BERT while avoiding the limitations of auto-encoder models. This model learns from unsupervised text.
(2) BERT [3]: Bidirectional Encoder Representations from Transformers (BERT) is based on transformers, in BERT the model reads from left to right as well as right to left. Like XLNet, BERT also learns from unsupervised text.
(3) RoBERTa [4]: This is a re-implementation of BERT which uses the same architecture as BERT. However, in RoBERTa, certain hyperparameters, pretraining schemes, and embeddings have been tweaked.

3.2.3 Analysis. We plan to perform analysis through several methods. The first part of the analysis includes classifying unseen phone case reviews using our highest-performing model. Then we will perform sentiment analysis to understand how sentiments differ from negative, neutral, and positive reviews. We will also use keyword extraction on the reviews to create a word cloud that would explain the potential differences between each class.

3.3 Rationale

We wanted to explore various Natural Language Processing techniques with our data and understand how each model works. In addition, we wanted to compare the performance of word embedding techniques versus neural network models. For example, we want to encode words into word vectors, so we decide to use Word2Vec, USE, and TF-IDF, which are common word embedding techniques. Those algorithms encode each review into 512-dimension vectors as input features for our models. On the other hand, since we have three classes (i.e., -1, 0, and 1) for the targets, it is a classification problem. Thus, we select KNN, Decision Tree, Random Forest, and AdaBoost for our models, which can handle classification tasks.

4 DATASET

For this project, we built our own dataset consisting of product reviews scraped from Amazon. To automatically scrape the reviews, we used the `requests`[1] package in Python, along with Splash,[2] a service for rendering JavaScript. Together, these enabled us to send

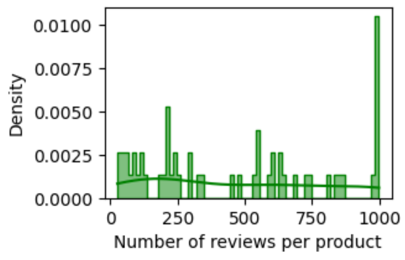


Figure 1: The distribution of the number of reviews per product in the dataset of Amazon reviews.

HTTP requests within our Python code to scrape Amazon.com without restrictions. The code used to generate the dataset is available on GitHub.[3]

First, we selected 55 different products from Amazon. To limit the scope of the dataset, all 55 products were phone cases. On the other hand, to get a broad range of products and reviews, each product was a different brand. Then, for each product we scraped up to 100 pages of reviews (if available); each page had up to 10 reviews. This resulted in a total of $25,745$ reviews over the 55 products. Figure 1 shows the distribution of the number of reviews per product in the resulting dataset. For each review, we extracted the product name, the review title, the rating (i.e., the number of stars out of five), and the review body. Table 1 shows a snippet of the dataset, including reviews for the popular phone case: OtterBox Commuter.[4]

In general, reviews that are have similar ratings can be especially difficult to distinguish. For example, two reviews that have ratings 1 and 2, respectively, may appear very similar based solely on the contents of the review body since they are both negative reviews. The same may be the case for reviews with ratings 4 and 5. These cases can even be difficult for humans to read and classify. Thus, we simplified the problem slightly by converting the 5-class classification problem to only a 3-class classification problem. Reviews with ratings 1 and 2 comprise the "negative" (-1) class, reviews with rating 3 comprise the "neutral" (0) class, and reviews with ratings 4 and 5 comprise the "positive" (+1) class. This classification problem is more feasible, while still providing valuable classifications for reviews. Table 2 shows the distribution of these three classes in our raw dataset. The resulting dataset is available on GitHub.[5]

5 EXPERIMENTS

5.1 Hypotheses

The main question that we seek to answer in our experiment is: given our set of models and classifiers, can we obtain an effective model that can predict the rating of Amazon phone case reviews? After obtaining our results, we want to answer another question by making a comparison between our different models and classifiers to see which method may be the best. One of our predictions is

[1]https://requests.readthedocs.io/en/latest/
[2]https://splash.readthedocs.io/en/stable/
[3]https://github.com/schatzkara/CSC_522_Group_23/blob/main/amazon_review_scraper.py
[4]https://www.amazon.com/OtterBox-Commuter-Case-iPhone-Pro/dp/B08DY8GJN4/ref=cm_cr_arp_d_product_top?ie=UTF8
[5]https://github.com/schatzkara/CSC_522_Group_23/blob/main/combined_generalized_rating.csv

Table 1: A snippet of the dataset of Amazon phone case reviews.

Product	Review Title	Rating	Review Body
OtterBox Commuter	Great case!	Positive (5)	Best case you can get!
OtterBox Commuter	I would purchase again	Positive (5)	I bought this as a gift and the person loved it. It's easy to put...
OtterBox Commuter	Great product.	Positive (4)	Easy to use.
OtterBox Commuter	A little less for a little more $$$	Positive (4)	I had a difficult time in adjusting the volume and shutting off...
OtterBox Commuter	Not practical	Neutral (3)	The product is sturdy but it is very difficult to work the side...
OtterBox Commuter	Several Nit Picks	Negative (2)	Senior Mechanical design engineer for over 40 years. Here are...
OtterBox Commuter	didn't last long	Negative (1)	It lasted about 21 month, i work as a plumber and fully...
OtterBox Commuter	Great for the price!	Positive (5)	Has held up at job sites, slim and durable. Great feature is the...
OtterBox Commuter	Doesn't fit my iPhone 12	Negative (2)	Not the right fit for my iPhone.
OtterBox Commuter	Good cheap not total garbage	Positive (4)	I think it's good. It protects my phone

Table 2: The number of reviews scraped for each class.

Class	Number of Reviews
Negative	3, 899
Neutral	1, 912
Positive	19, 935

that the more complex the model, such as with a Neural Network, the more effective it will be in rating prediction. Finally, we want to analyze the model and understand what words or phrases are more commonly found in certain clusters of predictions rather than others.

5.2 Experimental Design

For a high-level depiction of the entire experimental process, see Figure 4 in Appendix A. We implemented our methods and ran all our experiments in Python. Our code is available on GitHub.[6] As described in Section 3.2, we used two different types of models: (1) *Word Embedding (WE) Models* and (2) *Neural Network (NN) Models*. These are discussed in Sections 5.2.1–5.2.2, respectively.

5.2.1 Word Embedding Models. Each *Word Embedding Model* consists of (1) an embedding method and (2) a classifier (see Section 3.2.2). We denote by *E-C* the model that uses embedding method *E* and classifier *C*, e.g. W2V-kNN. We used three embedding methods and four classifiers, for a total of 12 models. Details on the embedding method and classifier implementation are given below.

Embedding Methods. As described in Section 3.2, we used three different embedding methods in our models: Word2Vec (W2V), Universal Sentence Encoder (USE), and Term Frequency-Inverse Document Frequency (TF-IDF). In our experiments, we used pre-implemented versions of these models from various Python libraries. We used the Word2Vec method provided by the `gensim` library,[7] the USE method provided by the `TensorFlow` library[8], and the TF-IDF method provided by the `scikit-learn` library.[9]

[6]https://github.com/schatzkara/CSC_522_Group_23
[7]https://radimrehurek.com/gensim/models/word2vec.html
[8]https://tfhub.dev/google/universal-sentence-encoder/4
[9]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Classifiers. As described in Section 3.2, we used four different classifiers in our models: kNN, Decision Tree (DT), Random Forest (RF), and AdaBoost (AB). For each classifier, we used the pre-implemented version from the `scikit-learn`[10] library in Python.

5.2.2 Neural Network Models. We used three different *NN Models*: XLNet, BERT, and RoBERTa. For each model, we used the pre-implemented version from the `Simple Transformers`[11] library in Python.

5.2.3 Data Preprocessing. Our main preprocessing step aimed to resolve issues with class imbalance. To this end, we upsampled the "neutral" class and downsampled the "negative" and "positive" classes such that we had 3, 000 total samples of each class. This version of the dataset is available on GitHub as well.[12]

We conducted our experiments on the original dataset, see Section 4, as well as the balanced dataset so we could analyze the impact of the class balancing on our model performance, see Section 6.1.1 for details. We call these datasets ***Reviews-Complete*** and ***Reviews-Balanced***, respectively.

Moreover, we also applied several text preprocessing methods to the dataset using the `gensim`[7] library. We applied many standard text preprocessing steps, for example, transforming strings into lowercase, removing tags, replacing punctuation characters with spaces, removing repeating whitespace characters, and removing digits from strings. Unfortunately, we found that these preprocessing steps did not have a significant impact on model performance, so we opted not to apply them to the datasets used in our experiments, see Section 6.1.3 for details.

5.2.4 Training and Evaluation. To classify reviews, we used both the *WE Models* and the *NN Models*, for a total of 15 models. We trained the *NN models* for five epochs.

To train and test our models, we used a random 80-20 split of each dataset; 80% (the training set) was used in the training phase to train each model, and the remaining 20% (the testing set) was used in the testing phase to evaluate the model. For the ***Reviews-Complete*** dataset, the training set consisted of 3, 105 negative reviews, 1, 553 neutral reviews, and 15, 939 positive reviews, and the testing set

[10]https://scikit-learn.org/stable/
[11]https://simpletransformers.ai/
[12]https://github.com/schatzkara/CSC_522_Group_23/blob/main/balanced_data_set_3000.csv

consisted of 794 negative reviews, 359 neutral reviews, and 3, 996 positive reviews. For the ***Reviews-Balanced*** dataset, the training set consisted of 2, 418 negative reviews, 2, 390 neutral reviews, and 2, 392 positive reviews, and the testing set consisted of 582 negative reviews, 610 neutral reviews, and 608 positive reviews.

To evaluate the performance of our models, we used several standard classification performance metrics. First, we used precision, recall, F1-score. Because we have three classes, we compute these metrics separately for each class. For each class i, ***precision*** (Eq. 1) is the proportion of reviews that the model predicted to be in class i that are actually in class i, and ***recall*** (Eq. 2) is the proportion of reviews that are in class i that the model predicted to be in class i.

$$Precision_i = \frac{\text{\# class } i \text{ reviews predicted as class } i}{\text{\# reviews predicted as class } i} \tag{1}$$

$$Recall_i = \frac{\text{\# class } i \text{ reviews predicted as class } i}{\text{\# class } i \text{ reviews}} \tag{2}$$

F1-score (Eq. 3) is the harmonic mean of precision and recall and is commonly used to balance the tradeoff between the two metrics.

$$F1\text{-}score_i = 2 \times \frac{Precision_i \times Recall_i}{Precision_i + Recall_i} \tag{3}$$

To aggregate the results for each class, we computed the macro average of these metrics (precision, recall, and F1-score). The ***macro average*** (Eq. 4) is the mean of the scores for the three classes.

$$MacroAverage(Metric) = \frac{\sum_i Metric_i}{3} \tag{4}$$

Finally, we used ***accuracy*** (Eq. 5) as a metric for the overall model performance, which is the proportion of all reviews that were correctly classified by the model.

$$Accuracy = \frac{\sum_i \text{\# class } i \text{ reviews predicted as class } i}{\text{\# reviews}} \tag{5}$$

5.2.5 Sentiment Analysis. After creating our models and selecting the best model, we performed sentiment analysis on the data. First, we used the best-performing model and annotated 5, 000 unseen reviews. This resulted in a total of around 11, 000 reviews annotated with negative, neutral, and positive class labels. Then, we created three clusters corresponding to the annotation labels of negative, neutral, and positive. Next, we performed sentiment analysis on the clusters to understand how much positive/negative sentiment is in each cluster, as well as the entire dataset.

We decided to use a pre-trained model for performing sentiment analysis. Specifically, we used the pipeline from the `transformers`[13] library in Python with the base pre-trained model. This model is from Hugging Face and specifically uses the "distilbert-base-uncased-finetuned-sst-2-english" model. This pre-trained model performs an analysis of positive versus negative labels, providing the label as well as a confidence score from 0 to 1 of how confident the model feels with the assigned label.

In addition to performing sentiment analysis on the clusters, we also created word cloud graphs to understand the most common words used on reviews of different classes. To increase the usability of the word cloud, we removed stop words and added application-specific stop words like "phone," "case," and "phonecase."

[13]https://huggingface.co/docs/transformers/index

6 RESULTS AND DISCUSSION

6.1 Model Performance

6.1.1 The Impact of Class Imbalance. We first trained and tested each of our 12 *Word Embedding (WE) Models* on the ***Reviews-Complete*** dataset. This gave us a baseline for our results, allowing us to analyze the impact that preprocessing had on the models. The summary results for each model are shown in Table 3a, i.e., the macro averages of the precision, recall, and F1-scores for each class, along with the overall model accuracy.

To further understand our models, we analyzed the model performance for the three individual classes: negative, neutral, and positive. For the sake of space these results are shown in Appendix B.1 in Table 6. From these results, we noticed that the models tend to perform substantially better on the positive (+1) class as compared to the other two classes. Given the class distribution of our training and testing sets (see Section 5.2.4), we hypothesized that this may be due to class imbalance, which led us to generate the ***Reviews-Balanced*** dataset (see Section 5.2.3).

Next, we trained and tested each of our 12 *WE Models* on the ***Reviews-Balanced*** dataset. The summary results for each model are shown in Table 3b, i.e., the macro averages of the precision, recall, and F1-scores for each class, along with the overall model accuracy. Just as for the models trained on the ***Reviews-Complete*** dataset, we also report the specific model performance for the three individual classes in Appendix B.1 in Table 7.

First, by comparing Tables 3a and 3b, we can see the impact of the class imbalance. The results on models with Word2Vec as the embedding method were comparably poor in using both datasets,[14]. If we focus on the models using USE or TF-IDF, while the accuracy of these models using the balanced dataset did not improve much from the unbalanced dataset, the precision, recall, and F1-scores improved overall. Accuracy is heavily impacted by class imbalance, so it is not surprising that our accuracy stayed roughly the same in most cases. Table 6 shows more performance details, which highlight that our models performed well with respect to accuracy on the ***Reviews-Complete*** dataset mainly because they predicted most reviews to be in the positive (+1) class. Since the positive class is so much more prevalent in this version of the dataset, the accuracy is still high, despite incorrectly classifying many neutral (0) and negative (-1) reviews. This explains the behavior that we see with the accuracy metric.

In the presence of class imbalance, precision, recall, and F1-score are much more robust metrics, and we saw that our model performed significantly worse with respect to these metrics on the ***Reviews-Complete*** dataset, as compared to the ***Reviews-Balanced*** dataset. In fact, Table 7 indicates that our USE and TF-IDF models do not have a strong preference for any of the three classes, that is, they perform roughly the same on all three classes, which is the desired behavior.

Overall, we conclude that class imbalance did, in fact, have a substantial impact on our models. We take this as validation of our decision to build the ***Reviews-Balanced*** dataset, and deduce that this dataset is more appropriate for training robust models that have no preference or performance-discrepancy between classes.

[14]We will discuss this more later in this section.

Table 3: The precision, recall, F1-score, and accuracy for the 12 *Word Embedding Models* on the two datasets. The macro average of the scores for the three classes (negative, neutral, and positive) are given for precision, recall, and F1-score. The highest score in each column for each embedding method is underlined; the highest score in each column overall is shown in bold.

(a) Model performance on the *Reviews-Complete* dataset.

Model	Precision	Recall	F1-score	Accuracy
W2V-kNN	0.375	0.349	0.348	0.685
W2V-DT	0.259	0.333	0.291	0.776
W2V-RF	0.259	0.333	0.291	0.776
W2V-AB	0.259	0.333	0.291	0.776
USE-kNN	0.602	**0.552**	**0.572**	0.829
USE-DT	0.509	0.513	0.511	0.762
USE-RF	0.542	0.495	0.507	**0.843**
USE-AB	0.603	0.545	0.544	0.841
TF-IDF-kNN	**0.615**	0.357	0.340	0.784
TF-IDF-DT	0.519	0.514	0.516	0.775
TF-IDF-RF	0.548	0.455	0.471	0.829
TF-IDF-AB	0.579	0.485	0.504	0.819

(b) Model performance on the *Reviews-Balanced* dataset.

Model	Precision	Recall	F1-score	Accuracy
W2V-kNN	0.374	0.366	0.363	0.365
W2V-DT	0.441	0.334	0.164	0.324
W2V-RF	0.330	0.334	0.165	0.324
W2V-AB	0.441	0.334	0.164	0.324
USE-kNN	0.695	0.685	0.688	0.685
USE-DT	0.763	0.757	0.757	0.757
USE-RF	**0.839**	**0.837**	**0.836**	**0.836**
USE-AB	0.637	0.639	0.637	0.638
TF-IDF-kNN	0.733	0.479	0.433	0.483
TF-IDF-DT	0.767	0.766	0.766	0.766
TF-IDF-RF	0.834	0.834	0.834	0.834
TF-IDF-AB	0.624	0.619	0.618	0.618

6.1.2 Word Embedding Model Performance. Now, we compare the performance of our various *WE Models* on the ***Reviews-Balanced*** dataset. The models with Word2Vec as the embedding method performed poorly in all cases. Given that this is a 3-class classification problem, these models performed roughly on par with a random classifier, which would have an expected accuracy of 0.333. From this, we deduce that Word2Vec is not well-suited for our problem. In fact, Word2Vec is an older technique, so it is not surprising that it was unable to perform as well as its younger counterparts.

The models with USE and TF-IDF as the embedding method performed well in most cases, with the USE models performing slightly better. For both embedding methods, RF was the strongest classifier, achieving an F1-score and accuracy of about 0.83. Thus, we conclude that USE was the strongest embedding method and RF was the strongest classifier, based on our experiments, making USE-RF our strongest *WE Model*. In Section 6.1.5, we will compare USE-RF with our *NN Models*.

6.1.3 The Impact of Text Preprocessing. As discussed in Section 5.2.3, we also applied several text preprocessing steps to our data. Then, we reran our models on the preprocessed data and compared the results to those discussed above. We found that there was no significant effect on the model performance, so we decided not to apply those preprocessing steps in the rest of our experiments.

6.1.4 Neural Network Model Performance. Next, we trained and tested each of our three *Neural Network (NN) Models* on the ***Reviews-Balanced*** dataset. Here, we only used the balanced dataset because of our observations in Section 6.1.1 that this dataset was better suited for training robust models. To test how the number of epochs impacts model performance, we trained each model for one epoch and for five epochs. The summary results for each model trained for five epochs are shown in Table 4, and the model performance for the individual classes is shown in Appendix B.1 in Table 8. For the sake of space, the corresponding results for each model trained for only one epoch are shown in Appendix B.2 in Tables 9–10.

Table 4: The precision, recall, F1-score, and accuracy for the three *Neural Network Models* on the *Reviews-Balanced* dataset. The macro average of the scores for the three classes (negative, neutral, and positive) are given for precision, recall, and F1-score. The highest score in each column is shown in bold.

Transformer	Precision	Recall	F1-score	Accuracy
XLNet	**0.878**	**0.877**	**0.877**	**0.877**
BERT	0.876	0.874	0.874	0.874
RoBERTa	0.875	0.871	0.872	0.871

Our results indicate that training for five epochs significantly improved the model performance. Since more epochs means that the model gets to "see" each data object more and thereby learn more from each data object, more epochs does typically improve performance, as we saw here. However, some models have long training times, as was the case for our models, so there is a tradeoff between time and performance. Since we saw such good performance after only five epochs, we decided to stop our training there. However, it would be interesting to train our models for more epochs in the future to see if the performance would continue to increase.

Overall, all three *NN Models* performed strongly and roughly on par with each other, with XLNet having a slightly stronger performance than both BERT and RoBERTa. Moreover, we see in Table 8 that these models perform roughly the same on all three classes, which is the desired behavior.

6.1.5 Word Embedding versus Neural Network Models. One of our main hypotheses was that more complex models, i.e., *NN Models* would perform strongest for our classification problem. In comparing USE-RF, the strongest *WE Model* and XLNet, the strongest *NN Model*, we see evidence that this hypothesis is correct. XLNet performed about 0.04 or 4% better than USE-RF with respect to all four performance metrics (see Tables 3b and 4).

Table 5: Some reviews misclassified by our XLNet model.

Actual Class	Predicted Class	Review
Positive	Negative	I love this case however after one drop on the ground it has a tear in it so be careful. It protected my phone though.
Positive	Neutral	I put my phone in my back pocket for work. I really enjoy this case but over time the top of the case breaks from coming in and out of my pocket.
Negative	Neutral	I like it easy to hold and seems pretty protective however the case keeps sticking to the back of the phone and it makes it looks like there is water in it. I don't care for that but all in all I like it.
Negative	Positive	2 weeks using and is completely dirty and didn't come out. Great!

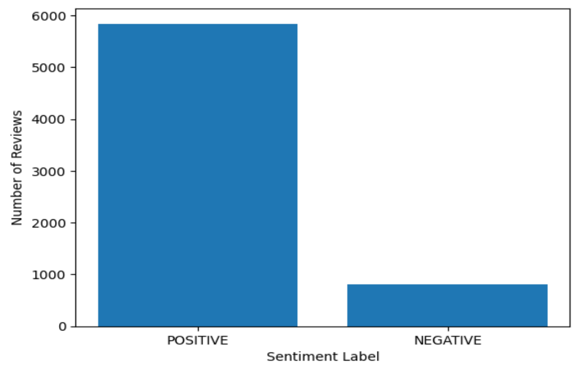
6.2 Sentiment Analysis

In this Section, we ran our trained XLNet model, the top performing model (see Section 6.1.5), on the remaining unseen reviews on which it was not trained, we call this dataset the ***Reviews-Unseen*** dataset (i.e., the reviews in ***Reviews-Complete*** but not in ***Reviews-Balanced***). To better understand the model's behavior, we analyzed its outputs in a variety of ways.

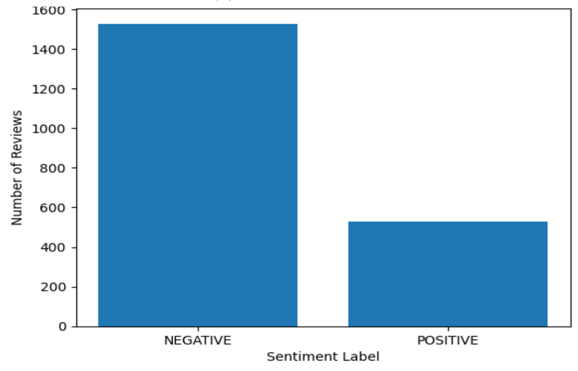
First, we manually analyzed several reviews from the ***Reviews-Unseen*** dataset that were misclassified by XLNet to identify any trends in the misclassifications. Table 5 shows a few such instances. The first two rows show positive reviews that were classified as negative and neutral, respectively. In both cases, we can see that the review has somewhat mixed sentiments, i.e., it starts out positive and then turns negative, so it is difficult to classify. It is possible that the model is thrown off by keywords like "however" and "but" because these words usually indicate something negative is being shared. The last two rows show negative reviews that were classified as neutral and positive, respectively. The review in the third row follows the same trend as the misclassified positive reviews: it is a mix of positive and negative sentiments. The fourth row actually shows an instance of sarcasm, which is known to be a difficult case for machine learning models to handle. Overall, these insights reveal one key limitation of our model: it has difficulty when mixed sentiments appear in the review text. Because this is a particularly difficult situation for classification, this model behavior is not surprising.

Next, we performed automatic sentiment analysis on the ***Reviews-Unseen*** dataset, see Section 5.2.5 for details. The sentiment analysis assigned a sentiment label to each review: either positive or negative sentiment. Naturally, we would expect reviews to receive sentiment labels that match their class. To this end, we grouped the reviews in each class by their sentiment, as shown in Figures 2a–2c. For the positive and negative classes, we can see that the sentiment labels align well with our intuition, i.e., the positive reviews have mostly positive sentiment while the negative reviews have mostly negative sentiment. For neutral reviews, we see a mix of positive and negative sentiment with negative sentiment being more prominent. This is an interesting trend for neutral reviews, and is somewhat expected. Overall, our sentiment analysis reveals that sentiment is strongly related to the review class and may be a good classifier in the future.

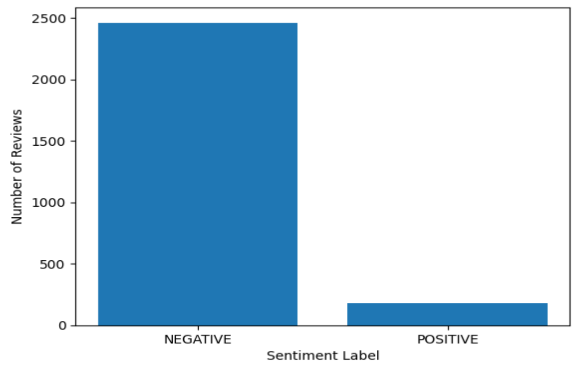
Finally, we created word cloud of the positive, neutral, and negative reviews in the ***Reviews-Unseen*** dataset, which are shown in Figures 3a–3c, respectively. Word clouds show the words used in the reviews, where the size of each word corresponds to the frequency of use. The word cloud for the positive class seems to align very



(a) Positive class

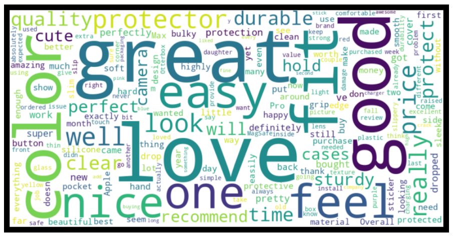


(b) Neutral class

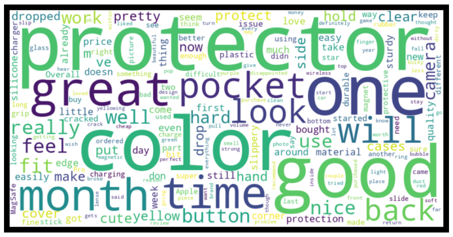


(c) Negative class

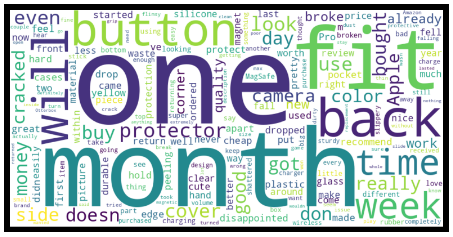
Figure 2: The distribution of positive and negative sentiment in reviews of each class in the *Reviews-Unseen* dataset.



(a) Positive class



(b) Neutral class



(c) Negative class

Figure 3: Word clouds of the reviews for each class, where word size corresponds to frequency of use.

well with our natural expectations as the most common words are positively connotated, e.g., "great," "love," and "nice." On the other hand, the most common words in the word cloud for the negative reviews don't seem to have a particular connotations. However, some of the other words in this cloud are negatively connotated, e.g., "broke," "flimsy," and "disappointed." We would have naturally expected words like these to appear more frequently. Overall, the word clouds give us good insight into the types of things people say in reviews of different classes.

7 CONCLUSION

We learned through this process that in order to achieve a well-rounded prediction model, it is paramount to experiment with many different methods. It is hard to find a one-size-fits-all solution, especially when it comes to complex classification problems. We found out that the best model for our dataset was the Neural Network model XLNet. However, as an Neural Network model, it is difficult to understand the reasons for its predictions, not to mention that training these models took at least three hours for each epoch. This also does not mean that using a Neural Network will always be more accurate than using more primitive techniques, but in our case of classifying phone case reviews, it was the better choice. Choosing the best model is simply a measure of the time you are willing to experiment with different techniques that align with the expectations for your results.

Another outcome from this project was learning more about how to combine different techniques to build an effective pipeline. In order to reach a desired result, we combined existing techniques to form our own unique solutions. For example, we scraped our own dataset, and experimented with word embedding techniques like Word2Vec, Universal Sentence Encoder, and TF-IDF, and combined these with classification techniques to classify reviews in our dataset. We also learned to make use of tools like Jupyter Notebook and Google Colab to help accelerate the model creation and testing process.

8 MEETING ATTENDANCE

(1) April 8th, 8-9pm: everyone
(2) April 15th, 8-9pm: everyone
(3) April 20th, 9-10pm: everyone
(4) April 23nd, 4-5pm: everyone

REFERENCES

[1] Sara Ashour Aljuhani and Norah Saleh Alghamdi. "A Comparison of Sentiment Analysis Methods on Amazon Reviews of Mobile Phones". In: *International Journal of Advanced Computer Science and Applications* 10.6 (2019).
[2] Daniel Cer et al. *Universal Sentence Encoder.* 2018. arXiv: 1803.11175 `[cs.CL]`.
[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: 1810.04805 `[cs.CL]`.
[4] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019. arXiv: 1907.11692 `[cs.CL]`.
[5] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: 1301.3781 `[cs.CL]`.
[6] "TF–IDF". In: *Encyclopedia of Machine Learning.* Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 986–987.
[7] T. K. Shivaprasad and Jyothi Shetty. "Sentiment analysis of product reviews: A review". In: *2017 International Conference on Inventive Communication and Computational Technologies (ICICCT).* 2017, pp. 298–301.
[8] Yuanhang Xiao, Chengbin Qi, and Hongyong Leng. "Sentiment analysis of Amazon product reviews based on NLP". In: *2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE).* 2021, pp. 1218–1221.
[9] Zhilin Yang et al. "XLNet: Generalized autoregressive pretraining for language understanding". In: *Advances in neural information processing systems* 32 (2019).

A PROCEDURE OVERVIEW

Figure 4 shows a high-level overview of our entire process.

B ADDITIONAL MODEL RESULTS

B.1 Model Performance on Individual Classes

Tables 6–8 show model performance on the individual classes for (i) the *WE Models* on the ***Reviews-Complete*** dataset, (ii) the *WE Models* on the ***Reviews-Balanced*** dataset, and (iii) the *NN Models* on the ***Reviews-Balanced*** dataset, respectively.

B.2 Results on Training Transformers for One Epoch

Tables 9–10 show the overall performance and the class performance, respectively, of the *NN Models* when trained for one epoch.

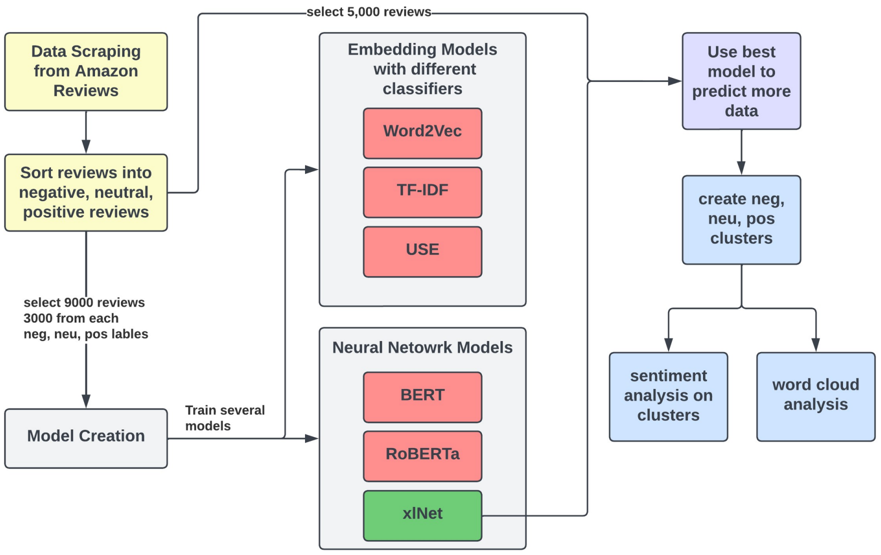

Data Scraping from Amazon Reviews
Sort reviews into negative, neutral, positive reviews
select 5,000 reviews
select 9000 reviews 3000 from each neg, neu, pos lables
Model Creation
Train several models
Embedding Models with different classifiers
Word2Vec
TF-IDF
USE
Neural Netowrk Models
BERT
RoBERTa
xlNet
Use best model to predict more data
create neg, neu, pos clusters
sentiment analysis on clusters
word cloud analysis

Figure 4: A high-level overview of our experimental procedure from dataset curation to analysis.

Table 6: Details of the performance of the 12 *Word Embedding Models* on the *Reviews-Complete* dataset on each of the three classes: negative (-1), neutral (0), and positive (+1). We report the precision, recall, and F1-score for each class for each model.

Model	Precision			Recall			F1-score		
	-1	0	+1	-1	0	+1	-1	0	+1
W2V-kNN	0.168	0.174	0.782	0.154	0.045	0.848	0.161	0.071	0.814
W2V-DT	0.000	0.000	0.776	0.000	0.000	0.999	0.000	0.000	0.874
W2V-RF	0.000	0.000	0.776	0.000	0.000	0.999	0.000	0.000	0.874
W2V-AB	0.000	0.000	0.776	0.000	0.000	0.999	0.000	0.000	0.874
USE-kNN	0.671	0.251	0.884	0.555	0.159	0.943	0.608	0.195	0.912
USE-DT	0.492	0.153	0.881	0.495	0.178	0.867	0.494	0.165	0.874
USE-RF	0.773	0.000	0.851	0.499	0.000	0.987	0.606	0.000	0.914
USE-AB	0.639	0.286	0.885	0.636	0.045	0.953	0.638	0.077	0.918
TF-IDF-kNN	0.684	0.375	0.786	0.068	0.008	0.996	0.124	0.016	0.879
TF-IDF-DT	0.517	0.163	0.876	0.500	0.156	0.885	0.508	0.159	0.881
TF-IDF-RF	0.814	0.000	0.830	0.370	0.000	0.994	0.509	0.000	0.905
TF-IDF-AB	0.633	0.253	0.851	0.437	0.056	0.963	0.517	0.091	0.904

Table 7: Details of the performance of the 12 *Word Embedding Models* on the *Reviews-Balanced* dataset on each of the three classes: negative (-1), neutral (0), and positive (+1). We report the precision, recall, and F1-score for each class for each model.

Model	Precision			Recall			F1-score		
	-1	0	+1	-1	0	+1	-1	0	+1
W2V-kNN	0.321	0.437	0.363	0.454	0.369	0.276	0.376	0.400	0.314
W2V-DT	0.324	1.000	0.000	1.000	0.002	0.000	0.489	0.003	0.000
W2V-RF	0.323	0.667	0.000	0.998	0.003	0.000	0.488	0.007	0.000
W2V-AB	0.324	1.000	0.000	1.000	0.002	0.000	0.489	0.003	0.000
USE-kNN	0.641	0.612	0.832	0.696	0.657	0.702	0.667	0.634	0.762
USE-DT	0.750	0.716	0.882	0.754	0.826	0.691	0.752	0.767	0.751
USE-RF	0.792	0.826	0.900	0.885	0.823	0.803	0.836	0.824	0.849
USE-AB	0.626	0.528	0.756	0.694	0.485	0.738	0.659	0.506	0.747
TF-IDF-kNN	0.857	0.398	0.945	0.237	0.974	0.225	0.371	0.565	0.364
TF-IDF-DT	0.742	0.754	0.805	0.747	0.815	0.735	0.745	0.783	0.769
TF-IDF-RF	0.820	0.820	0.864	0.844	0.834	0.824	0.831	0.827	0.843
TF-IDF-AB	0.555	0.571	0.745	0.680	0.497	0.681	0.612	0.531	0.711

Table 8: Details of the performance of the three *Neural Network Models* on the *Reviews-Balanced* dataset on each of the three classes: negative (-1), neutral (0), and positive (+1). We report the precision, recall, and F1-score for each class for each model.

Transformer	Precision			Recall			F1-score		
	-1	0	+1	-1	0	+1	-1	0	+1
XLNet	0.866	0.832	0.936	0.879	0.851	0.900	0.872	0.842	0.918
BERT	0.869	0.820	0.939	0.857	0.866	0.898	0.863	0.842	0.918
RoBERTa	0.892	0.796	0.938	0.829	0.876	0.908	0.859	0.834	0.923

Table 9: The precision, recall, F1-score, and accuracy for the three *Neural Network Models* on the *Reviews-Balanced* dataset when trained for only a single epoch. The macro average of the scores for the three classes (negative, neutral, and positive) are given for precision, recall, and F1-score. The highest score in each column is shown in bold.

Transformer	Precision	Recall	F1-score	Accuracy
XLNet	0.806	0.803	0.804	0.803
BERT	0.807	0.802	0.803	0.802
RoBERTa	**0.812**	**0.806**	**0.808**	**0.806**

Table 10: Details of the performance of the three *Neural Network Models* on the *Reviews-Balanced* dataset on each of the three classes: negative (-1), neutral (0), and positive (+1), when trained for only a single epoch. We report the precision, recall, and F1-score for each class for each model.

Transformer	Precision			Recall			F1-score		
	-1	0	+1	-1	0	+1	-1	0	+1
XLNet	0.785	0.717	0.916	0.792	0.744	0.873	0.788	0.730	0.894
BERT	0.792	0.710	0.918	0.775	0.770	0.860	0.783	0.739	0.888
RoBERTa	0.796	0.705	0.934	0.775	0.770	0.872	0.785	0.736	0.902