Predicting Amazon Product Ratings with Review Analytics

Christina Chang [1] Samuel Chen [1] Sean Chen [1] John Paulus Francia [1] Zachary Markham [1] Jesse Patton [1] Dina Le-Hien Pham [1] Spencer Pham [1] Mehtab Randhawa [1] Vincent Tran [1] Samuel Weinstein [1] Sangwoo You [1]

Abstract

Previous research on Amazon review text focuses on sentiment analysis, but lacks connections to star ratings. In this paper, we built an Amazon star ratings classifier using textual data as input. We utilized term frequency–inverse document frequency (TF-IDF) and Doc2Vec to generate our features, and built an artificial neural network (ANN) with binary cross-entropy loss function to classify the features into 1 to 5 star ratings. We also cluster the text review data with K-means and visualize the resulting star ratings using t-Distributed Stochastic Neighbor Embedding (t-SNE) in order to understand the relationship between rating and review text. We report the accuracy of our binary cross-entropy loss function ANN to be 87% with TF-IDF as the feature selection method, and 86% with Doc2Vec. We find the ROC plots to be nonlinear and indicate that our ANN model predictions are better than chance. Our Amazon star rating categorization can be used in conjunction with sentiment analysis to better quantify Amazon review texts and develop a consistent rating system.

The Github repository is available here: https://github.com/smlchen/reviewClassifier.

1. Introduction

With the internet becoming more prominent in the last few decades, the amount of unstructured user-generated text data has been rapidly increasing. Classification of this unstructured text data for further analysis or predictive applications is important for web searching, information filtering, target marketing, medical diagnosis and sentiment analysis (Aggarwal & Zhai, 2012). An example of this text data comes from product reviews on Amazon that are written by buyers for other potential customers to read. These product reviews contain a star level rated by the buyers and product feedback, which is usually the first thing a potential customer sees when viewing a product, so it can influence whether or not they will end up purchasing the item. While many potential buyers rely on the overall star ratings to make an informative decision, these star ratings actually do not capture a lot of information of the products such as sentiments of the reviews.

Therefore, a lot of efforts in the field have been put into doing sentiment analysis of the reviews in order to evaluate a product more holistically. Prior work has tried to classify Amazon reviews on smartphones into "Positive", "Negative", and "Neutral" (Pankaj et al., 2019). Others have tried to include sentiments of anger, anticipation, disgust, fear, joy, sadness, surprise and trust (Singla et al., 2017). On the other hand, Bhatt and his colleagues have found a connection between positive/negative sentiments and high/low star ratings, but they did not account for the fact that one could write a good review but give a low star rating (Bhatt et al., 2015). In fact, because customers have different interpretations of each level in the five-star rating system, the reviews they write may not match with the star rating they assign. Therefore, the goal of this project is to build a predictor of star ratings based on the text reviews using a supervised ANN. We use K-means clustering, an unsupervised machine learning method, to see if the star ratings correspond to the clusters they were assigned to. We use K=5 because there are five classes of star ratings (one-star, two-star, three-star, four-star, five-star). Then we use TF-IDF and Doc2Vec as different feature engineering methods.

Through the K-means clustering (K=5), we have found that the features generated by both TF-IDF and Doc2Vec are not clustered together by star ratings. Additionally, using an ANN with binary cross-entropy as loss function, we are able to predict star ratings from text reviews with good accuracy. In this way, we can make sure that star ratings are consistent with the Amazon text reviews for future sentiment analysis.

2. Methods

2.1. Data Collection and Sampling

We obtain the base dataset that includes reviews from May 1996 to October 2018 from Jianmo Ni, a fourth year PhD student at UC San Diego (Ni et al., 2019). There are a total of 29 zipped files that represent each category in purchasable products on Amazon. The dataset holds various features such as reviews, overall rating and product ID for every product in a category. For this paper, we choose to focus on the review texts as input and overall ratings as output.

To randomly sample data values from a large database, we first read zipped JSON files from a directory holding all the Amazon data. Then, we remove duplicate review texts to prevent a skewed model. From each JSON file, we take 439 data values and parse them into a master CSV file. With 28 categories and 439 data values each, the master CSV file contains a total of 12,292 data rows.

2.2. Text Preprocessing

To clean and preprocess the text for TF-IDF and Doc2Vec, we use Python's Natural Language ToolKit (NLTK) library. We extract the "reviewText" column from the sampled Amazon dataset and remove all punctuation and newlines. Each sample's review text is then tokenized and lowercased. We assume that all of the review texts are written in English, and remove stopwords such as articles, pronouns, and prepositions, as well as non-English words and spelling typos. We then further process the remaining words, which consisted of mostly nouns, adjectives, and verbs, with lemmatization and stemming. Lemmatization reduces different word tenses and forms down to its morphological root, returning a valid word and decreasing the number of unique words, and therefore features (Nguyen et al., 2015). Stemming also reduces words down to their root form, but may not return a valid word since it strips inflectional prefixes and suffixes. Finally, numbers are removed and the text is rejoined to be passed into TF-IDF or Doc2Vec.

2.3. Building TF-IDF and Doc2Vec Matrices

TF-IDF is used to measure how relevant a word is to a review text in a collection of Amazon reviews. The TF-IDF value increases proportionally to the number of times a word appears in the review and is offset by the number of reviews in the collection of Amazon reviews that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

To implement this method, we first write a function to compute a TF-IDF matrix given a dataframe. This function utilizes the class *TfidfVectorizer* from Scikit-learn library. This function returns the clean review texts from the previous phase into a list of strings. It then sets up the parameters for *TfidfVectorizer* in such a way that the matrix only considers words that are entirely alphabetical. Because there is a large amount of words from all the review texts combined, and our computational power is limited, the matrix only contains 1500 words that appear in no fewer than 20 review texts. In the end, this function returns the matrix in the form of a pandas dataframe. Lastly, we output this dataframe to a CSV file, along with the overall ratings for each review text as a separate column. We ten use the class *KFold* from Scikit-learn library to split the matrix into five sets of training and test data. We save every set of training and test data in CSV format in 5 separate folders for the next phase.

TF-IDF is defined as:

$$w_{i,j} = tf_{i,j} \times log\left(\frac{N}{df_i}\right)$$
$$tf_{i,j} = \text{number of occurrences of word } i \text{ in document } j$$
$$df_i = \text{number of documents containing word } i$$
$$N = \text{total number of documents}$$

On the other hand, Doc2Vec turns a review into a vector with a given number of dimensions. We use the function *Doc2Vec* from the class *gensim.models.doc2vec* in Gensim library. Because the input format for this function should be a representation of a document along with a tag, we also use the function *TaggedDocument* from the same class to transform the clean review texts. But before we transform the review texts to a list of documents with tags, we drop the null values in the dataset because there are empty strings that the function *Doc2Vec* cannot handle. Then we train the vectors of size 50 for every review over 100 epochs with a learning rate of 0.025. After each epoch, we decrease the learning rate by 0.0002 so that the vectors converge faster. Finally, these vectors are added to the original dataset in their own new column. We apply the same splitting method from TF-IDF on Doc2Vec, and we save every set of training and test data in CSV format in 5 separate folders for the next phase.

2.4. Clustering

Here, we use the K-means clustering algorithm, which is a form of unsupervised learning. The purpose of this algorithm is to aggregate similar data points into the same group. The number of groups, denoted by K, is a parameter in the algorithm. We decide to set K to five because there are five classes of star ratings. We want to see if reviews with the same rating will be placed in the same cluster. To do so, we use the function *k_mean* from the class *cluster* in Scikit-learn library. We first initialize the cluster centers with a method called "k-means++" in *k_mean*. This method of initialization initializes the cluster centers in such a way that speeds up convergence. After initialization, the algorithm iteratively performs calculations to optimize the position of each centroid. We run the K-means algorithm for a maximum of 300 iterations. Also, *sample_weight* in *k_mean* is set to default such that all observations are assigned equal weight. This procedure is done twice, one on the TF-IDF matrix and the other on the Doc2Vec vectors.

Furthermore, K-means clustering paired with dimensionality reduction provides a method to visualize the similarity between reviews for the entire dataset. We use t-SNE, a dimensionality reduction method, because it captures the local relationships and creates low-dimensional mappings. In other words, it preserves the non-linear structure of data. Observations that are close in high dimensions will be close to each other in a lower dimension visualization. This algorithm models the probability distribution of neighbors around each data point. The data in the high dimensional space is modeled as a Gaussian distribution, and the 2-dimensional data is modeled as a t-distribution. t-SNE attempts to find a 2-dimensional mapping that minimizes the difference between these two distributions (Maaten & Hinton, 2008). In our implementation of t-SNE, we use the function *TSNE* from the class *manifold* in Scikit-learn library. We set the number of nearest neighbors to 30, the learning rate 200, and the maximum number of iterations 1000.

2.5. Model Building, Testing, and Training

To construct a classifier to determine the review rating, we train two feed forward ANNs, one for each feature generation method used. A logistic regression model is also constructed for both sets of features. The neural networks are constructed the Keras library with TensorFlow as a backend. Both models use Adam as an optimizer, because Adam is computationally efficient and has low memory requirements (Kingma & Ba, 2015). For all models, there are five one hot encoded output nodes for the star ratings and sigmoid is used as the output activation function. We implement a parameter sweep to determine the best hyper parameters: hidden layer activation function, number of hidden layers, number of nodes per hidden layer, loss function, and whether or not to weight the data. The two common loss functions used are binary cross-entropy and MSE.

Binary cross-entropy is defined as:

$$CE = -\sum_{i=1}^{C} t_i log(s_i)$$

T_i = the ground truth of class C

s_i = the output of the ith output node.

A sigmoid activation function is used.

The mean squared error is defined as:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

$y_i =$ observed values

$\hat{y_i} =$ predicted values

$N =$ number of data points

The ground truth class (1 - 5 stars) of the samples is one-hot encoded so that a multi-class model could be built.

2.6. Plotting ROC and Precision-Recall Curves

The receiver operating characteristic (ROC) curve plots the true positive rate against the false positive rate and is used to help determine the best hyperparameters for the model (Metz, 1978). Individual rating ROC curves (Figure 3) are produced for the tested models and five-fold cross validation is performed to produce mean ROC curves for each rating with an error of 2 standard deviations. The sum of the areas under the curves is used to evaluate the best model parameters and is primarily used to determine if there was a difference between Doc2Vec and TF-IDF matrix, as well as the mean squared error and cross-entropy loss functions and logistic regression models. The functions "roc_curve" and "auc" from the class *Metrics* in Scikit-learn library are used to determine the true positive rate and false positive rate and the area under the curve.

Precision-Recall curves (Figure 4) for each rating were also produced using five-fold cross validation with error an of 2 standard deviations. Again the sum of the areas under the curves were used to help determine the best model. Precision-Recall curves are a better indicator of predictor performance than ROC curves when there are large class imbalances where Precision is the ratio of true positives to the sum of true positives and false positives and recall is the sensitivity(Saito & Rehmsmeiser, 2015). The function "precision_recall_curve" from the class *Metrics* in Scikit-learn library was used.

3. Results

Figure 1 is a histogram of the distribution star ratings of our sampled dataset. One-star reviews make up 4.8% of the samples, two-stars 4.1%, three stars 8.6%, four-stars 16.3%, and five-stars 66.2%.

The t-SNE plots (Figure 2) visualizing K-means clustering demonstrate that the clusters generated from the TF-IDF matrix and the Doc2Vec matrix yield different results. The t-SNE plot using the TF-IDF matrix shows that there are two large clusters that have quite a bit of overlap, and three smaller clusters that are relatively distinct. As for clustering using the Doc2Vec matrix, there are three large clusters and two small clusters. These

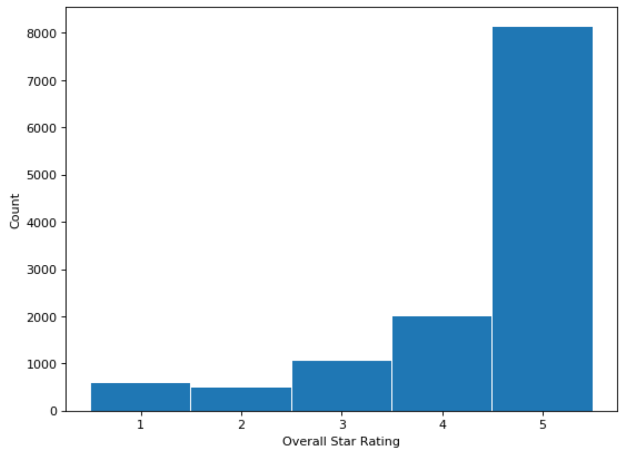


Figure 1. Overall star rating for sampled Amazon data.

clusters have more separation when compared to clustering with the TF-IDF matrix, however, there is still some overlap.

Based on the ROC curves created for the models that are trained during the perimeter sweep, the best hyperparameters for the TF-IDF matrix are: 1 hidden layer, 4 nodes per hidden layer, sigmoid as the hidden layer activation function, and binary cross-entropy as the loss function. Also, the best parameters for the model trained with the Doc2Vec samples are: 2 hidden layers, 32 nodes per hidden layer, sigmoid hidden layer activation functions, and binary cross-entropy as the loss function. In all cases, the models that are trained with unweighted data perform better than those trained with weighted data.

The PR curves are appropriate when the classes are imbalanced, while the ROC curves are better for balanced classes. In our case, because we do not have an even distribution of samples across all star rating classes, our classes are imbalanced. As we can see from the PR curves (Figure 4), both the model trained on the TF-IDF matrix and the model trained on Doc2Vec vectors predicts five star ratings the best. As for the rest of the star rating classes, both models performed poorly.

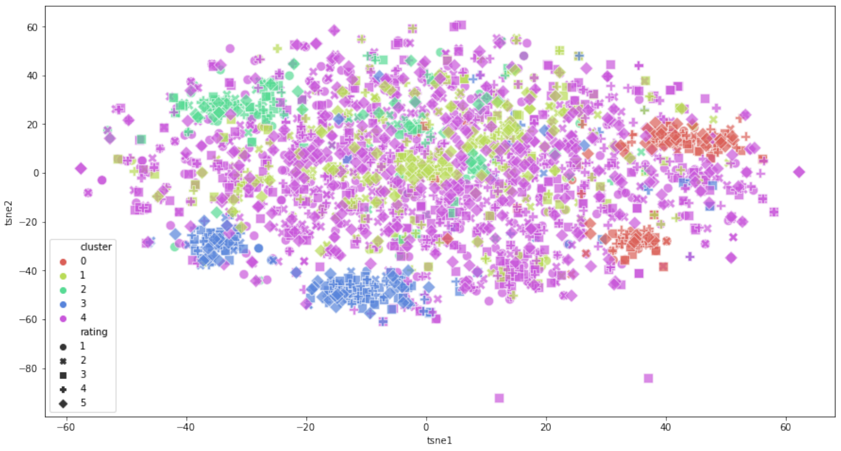
Table 1. The accuracy for testing data for each fold for TF-IDF and Doc2Vec.

Fold	Accuracy	
	Doc2Vec	TF-IDF
1	0.8854	0.875
2	0.8842	0.879
3	0.8841	0.881
4	0.8843	0.872
5	0.884	0.876
Mean	0.8844	0.877
std	0.0005	0.003

4. Discussion

Two major concessions have to be made to cope with the large volume of data. First, out of around two million reviews available to us, we only work with around 15,000 samples. It is important to note that the files used in this project are a smaller subset of a much larger database, but it should not affect the outcome of this project. Our samples are selected randomly from the original two million entries within the constraint of selecting an equal number of samples from each included product category. This is done to limit ourselves to a more practical number of samples and eliminate any potential bias from oversampling the more popular product categories. Two unintended consequences of this approach, however, is that one category, appliances, is removed due to its excessive amount of duplicate data. If this category were to remain, there would have been a maximum of 140 data values per category to preserve a uniform amount across all categories. The total number of data values to represent all the categories on Amazon would have resulted in only 140 * 29 = 4060 data values. By removing the appliances JSON file, the total number of data values will increase to 12,292 with over 28 categories, about three times the amount of original data values which is around 0.7% of the total number of reviews available.

Second, because the TF-IDF approach entails creating a new feature for every unique word present in a review, we find that even after a modest



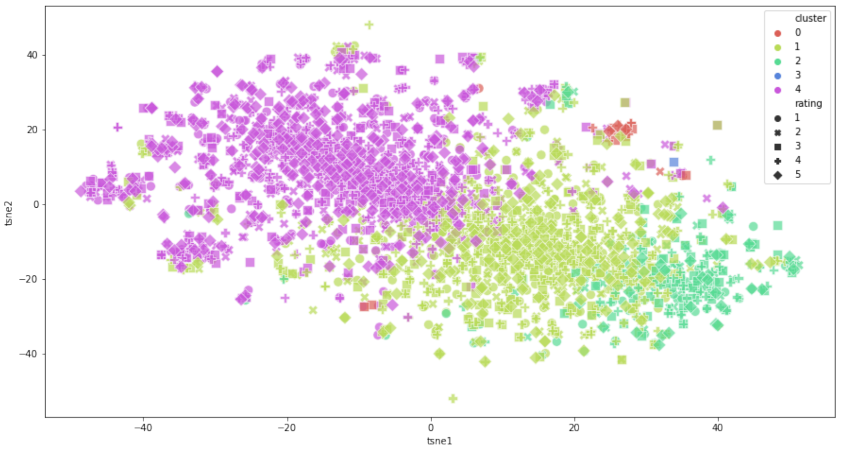


Figure 2. The top figure is the K-means clustering plot using t-SNE reduced Tf-idf data. The bottom figure is the K-means clustering plot using t-SNE reduced Doc2vec data.

amount of cleaning we are left with several thousand different features on which our models had to train. This is too burdensome of an approach with our small computers, so further cleaning is done by using both lemmatization and stemming, which both reduce inflectional words down to their roots. We remove over ten-thousand features from our TF-IDF matrix (meaning over ten-thousand unique words are removed from our review texts). Many of these features are typos and words in different tenses or forms, but have the same base meaning. Therefore, removing these words will not affect our results.

However, most text cleaning procedures use lemmatization or stemming and not both depending on the application, since they both produce similar results (Balakrishnan & Lloyd-Yemoh, 2014). We choose to perform stemming after lemmatization to further reduce the number of words since there are issues with the number of features in the TF-IDF matrix. However, stemming has the issue of crudely striping off prefixes and suffixes, sometimes resulting in non-English words (ie. “this” and “expense” became “thi” and “expens”, and words like “try” and “many” were converted to “tri” and “mani”). We also note that some of the non-English words are treated as typos and are removed in a later text preprocessing step, reducing the majority of our input data and removing connections between certain words, which end up affecting our Doc2Vec results.

In terms of Doc2Vec, it is based on Word2Vec, which operates on individual words. The vectors from Word2Vec are generated by a neural net

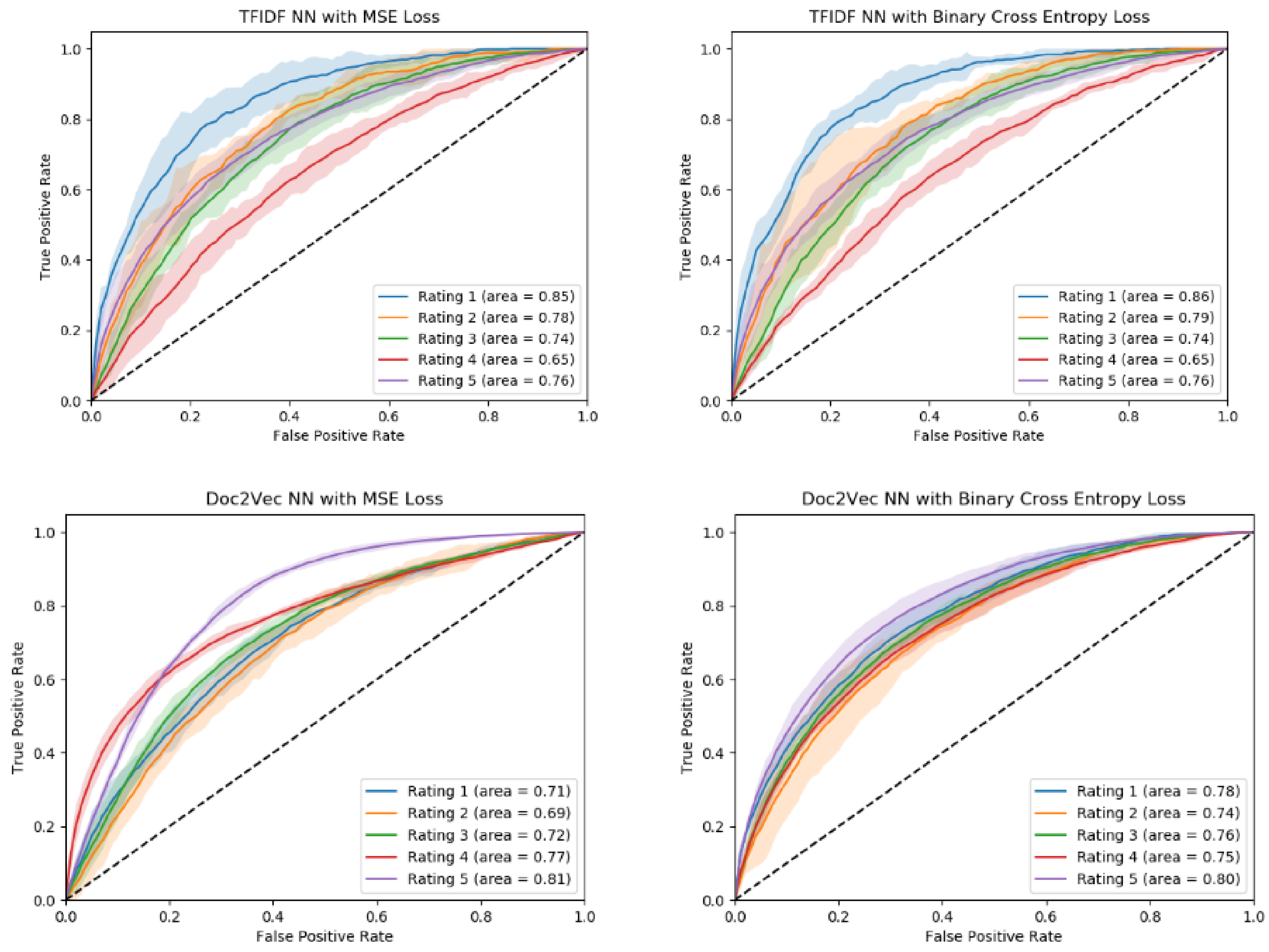


Figure 3. This plot shows the ROC curves for Doc2Vec and TF-IDF for both binary cross-entropy and MSE loss functions. Each curve is the mean of five-fold cross validation testing. The error is two standard deviations from the mean.

such that certain grammatical relationships can be expressed via mathematical operations on the vectors themselves and similar words (as measured by which words appear nearby) are close together in space. For example, the vector for “shirt” could be represented as the vector for “cloth” added to the vector for “torso” (Mikolov et al., 2013). On the other hand, Doc2Vec works by running Word2Vec but each sample has an additional feature (unique to each document) that is trained at the same time as the words. So the reason that Doc2Vec is used rather than Word2Vec in this paper is that we care more about the reviews as a whole rather than the individual words in every review.

In this paper, we compare Doc2Vec with TF-IDF. As opposed to TF-IDF, the strength of using Doc2Vec lies in its ability to take account of context. This allows Doc2Vec to attempt to represent the grammatical meaning of a document as opposed to only looking at word frequency (Mikolov et al., 2013). To make full use of this characteristic each document used ought to remain mostly unaltered from the original written English (i.e., stopwords should be left in). In our case though, the data we run Doc2Vec on had been heavily cleaned from its original form, including performing lemmatization, stemming, and stopword removal. In fact, we clean the data so much

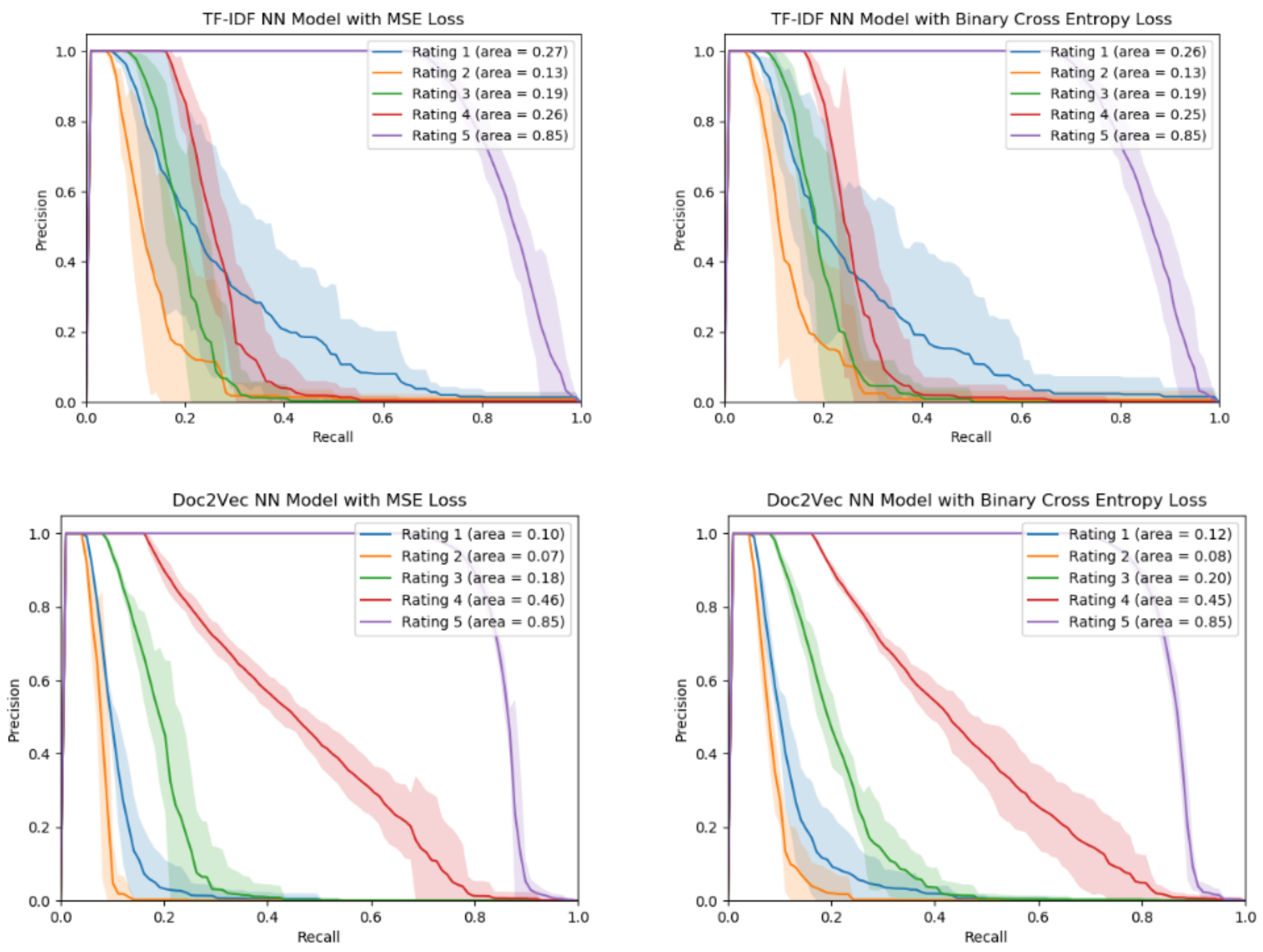
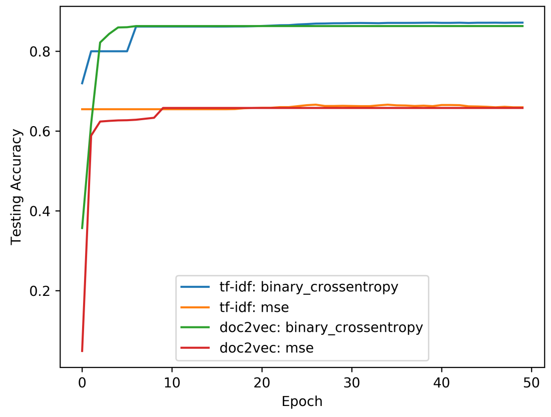


Figure 4. This plot shows the PR curves for Doc2Vec and TF-IDF for both binary cross-entropy and MSE loss functions. Each curve is the mean of five-fold cross validation testing. The error is two standard deviations from the mean.

that about 100 samples end up with empty review texts, all of which have to be eliminated before using Doc2Vec. Essentially, much of the context that Doc2Vec would have thrived on had been previously removed. In light of this, it is unsurprising that our Doc2Vec classification ends up with an accuracy comparable to that of the grammatically-naive TF-IDF classification as seen in Figure 5.

We group the reviews using K-means clustering to see if the reviews will be clustered into their respective rating class. Thus, we create scatterplots of the K-means clustering results by reducing the dimensionality of the features as shown in Figure 2. For each data point, the color corresponds to the assigned cluster, and the shape corresponds to the actual star rating. We notice that clusters in the Doc2Vec clustering plot do not overlap as much as the clusters in the TF-IDF clustering plot, for which we believe this is because Doc2Vec preserves the grammatical relationship between words, whereas TF-IDF does not. Because of this, more information is kept in the numerical representation of the reviews using Doc2Vec rather than TF-IDF. We expect reviews with the same star rating to be assigned to the same cluster. In other words, we hypothesize that each cluster would primarily have the same shapes. In both the TF-IDF and the Doc2Vec clustering plots, every cluster



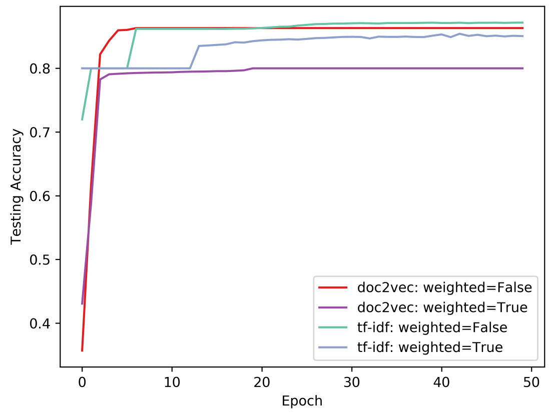


Figure 5. The top plot shows the testing accuracy plotted against epochs for two models (Doc2Vec and TF-IDF) with weighted and unweighted. The bottom plot shows the testing accuracy plotted against epochs for two models (TF-IDF and Doc2Vec) with two different loss functions (binary cross-entropy and mean squared error).

has a mix of all the five shapes. This means that the star ratings are not clustered into the same group and hence, the star rating does not seem to be consistent with the textual content of the review. This supports our motivation to make a consistent predictor of the rating by using the review text.

When we are building the models with different hyperparameters, we realize that when we use categorical cross-entropy as the loss function, the loss quickly becomes NaN. We then begin experimenting with other loss functions and find that binary cross-entropy yields models with good accuracy (85%). It is for this reason that we choose sigmoid as the output layer activation function.

We find that the loss function is the most sensitive parameter in determining accuracy. Figure 5 shows the accuracy of the models with the different loss functions. It is conclusive that binary cross-entropy is on average 20% more accurate compared to mean squared error. According to research conducted at RWTH Aachen University, "The Cross Entropy criterion allows [one] to nd a better local optimum than the Square Error criterion" (Golik & Ney, 2013). This is because the mean square error is not a convex function. Therefore, MES is liable to find a local minimum instead of the global minimum. binary cross-entropy on the other hand is a convex function and so is more likely to find the global minimum.

The above statements notwithstanding, after completing all of our plots, the accuracies of the models trained with binary cross-entropy and MSE do not fully agree with the corresponding ROC curves. Upon further investigation, we find that the problem results from the fact that we use "accuracy" as the metric when training the model instead of "categorical_accuracy." While a multi-class model can be trained by binary cross-entropy using TensorFlow, the use of "accuracy" as a metric has undefined behavior. After discovering this, we use "catagorical_accuracy" as a metric to train a single model using binary cross-entropy and the corresponding best hyperparameters, and find that the "catagorical_accuracy" of the model is indeed around 65%. Thus, the near 90% accuracy that we report above is not the true accuracy of the model, but rather, a result of undefined TensorFlow behaviour.

When building the ANN, there are issues regarding how our data is more biased towards 5 star ratings. When we first start playing with the hyperparameters of the ANN, we notice that most of the models have an accuracy of about 65%. After investigating, our classes are significantly imbalanced. Figure 1 shows that 66.2% of the data are samples with a ground truth class of five star ratings. So, we decide to weigh the data in the hopes that this will minimize the effect of the class imbalance. The inverse of the class frequencies are used to weight the corresponding classes. This gives rating class with fewer samples a higher weight. However, Figure 5 showcases that this approach is not successful and yields a less accurate result. An optimal solution will be to use a more balanced sample based on star ratings.

The ROC plots show that the neural network models are better at predicting than chance. The ROC for the logistic regression models for both Doc2Vec and TF-IDF models do not perform better than chance. And while the accuracy for the models trained using binary cross-entropy loss function show a large improvement over those trained with the MSE loss function there is no significant improvement to the ROC curves. Also, the standard deviation for the Doc2Vec models is lower than the TF-IDF models.

As for future work, we would like to return to data collection and sampling and balance the dataset by star rating in order to improve our model training. We would like to modify our data cleaning to retain more words in the corpus. A modest data cleaning process will retain more of the grammatical context of the reviews, and will lead to a better numerical representation of the reviews. Also, we would like to move the process of removing non-English words before lemmatization and stemming, instead of after lemmatization and stemming because these text standardization processes may convert some words to non-English words. We are only able to use a subset of the entire Amazon review dataset as well as the reduced number of features because we are limited by the computational power of our computers. In the future, we would like to use the entire dataset and its features by running our code on more powerful hardware.

We are interested in further exploring natural language processing through sentiment analysis. The results from our rating predictor can be paired with sentiment analysis in order to better understand the relationship between ratings and sentiments of reviews. Another potential addition for this project would be to incorporate the sentiment ratings of reviews into our classifier in order to get better rating predictions.

5. Author Contributions

Chen S. and Tran V. were in charged of Data Collection and Sampling. Chang C. and Pham D. performed text preprocessing. Chang C. also performed K-means clustering on the data. Chen S. and Zachary M. built the TF-IDF matrix and Doc2Vec vectors. Francia J., Patton J., Randhawa M., and You S. built, trained and tested the neural network model. Finally, Patton J., Pham S., and Weinstein S. plotted the ROC and PR curves.

6. References

Aggarwal, C. C. and Zhai, C. (eds.). *A Survey of Text Classification Algorithms*. Springer, Boston, MA, 2012.

Balakrishnan, V. and Lloyd-Yemoh, E. Stemming and lemmatization: A comparison of retrieval performances. *International Association of Computer Science and Information Technology*, 2014.

Bhatt, A., Patel, A., Chheda, H., and Gawande, K. Amazon review classification and sentiment analysis. *International Journal of Science and Information Technologies*, 6(6):5107–5110, 2015.

Golik, P. and Ney, H. Cross-entropy vs. squared error training: a theoretical and experimental comparison. *Proceedings of the Annual Conference of the International Speech Communication Association*, pp. 1756–1760, 2013.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. *ICLR*, 2015.

Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, pp. 2579–2605, 2008.

Metz, C. E. Basic principles of roc analysis. *Seminars in Nuclear Medicine*, 8(4), 1978.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *ICML*, 2013.

Nguyen, H., Veluchamy, A., Diop, M., and Iqbal, R. Comparative study of sentiment analysis with product reviews using machine learning and lexicon-based approaches. *SMU Data Science Review*, 1(4), 2015.

Ni, J., Li, J., and McAuley, J. Justifying recommendations using distantly-labeled reviews and fined-grained aspects. *Empirical Methods in Natural Language Processing*, 2019.

Pankaj, T., Pandey, P., Muskeen, A., and Soni, N. Sentiment analysis on customer feedback data: Amazon product reviews. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing*, 2019.

Saito, T. and Rehmsmeiser, M. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3), 2015.

Singla, Z., Randhawa, S., and Jain, S. Statistical and sentiment analysis of consumer product reviews. *IEEE*, 2017.