Products review rating prediction from users' text reviews

Asterios Bampakis
MSc Data and Web Science
Computer Science Department
Aristotle University of Thessaloniki
maasterio@csd.auth.gr

Themistoklis Spanoudis
MSc Data and Web Science
Computer Science Department
Aristotle University of Thessaloniki
spanoudis@csd.auth.gr

***Abstract*—Product review rating prediction is the task of predicting a review rating from the free-form text part of a review. It's a relatively new machine learning task involving natural language processing, with potential benefits for e-commerce, review websites, forums, and social media platforms. We study this task as a supervised learning problem using data from Amazon's product reviews. We evaluate different techniques for data pre-processing in combination with different machine learning algorithms for the review rating prediction problem. We additionally investigate the use of word- and document-level embeddings in combination with the same machine learning algorithms.** [1]

***Index Terms*—Natural Language Processing, Rating review prediction, Review mining, Machine Learning**

I. Introduction

Online reviews have become omnipresent across e-commerce websites such as Amazon and eBay, across review websites such as Yelp and TripAdvisor, as well as across social networks such as Facebook. A review usually consists of a free-form text part and a star rating part usually in the scale of 1 to 5. These two features are provided independently by the reviewer. The aggregation of star ratings is often used by consumers as a quick way to filter out millions of products and services that are available throughout the web. Additionally, businesses and corporations use these ratings to gather feedback about their offerings and gain valuable market insights.

It can easily be observed that inconsistencies between the free-form text part and the star rating part of a review are not so uncommon. Such inconsistencies, whether occurring by mistake or being intentional, can lead to so called mismatched ratings that can alter the perceived overall evaluation of a product or service. This issue is amplified on offerings that have a small number of reviews, for example offerings that have just recently been introduced to a market. In these cases even a few mismatched reviews can be enough to alter the overall rating.

Review rating prediction is the task of predicting the star rating from a corresponding free-form text review. The main benefits of developing a model that can successfully perform this task is being able to assess user feedback more consistently, to extract a quantitative measure of user opinion from reviews consisted only of free-form text, and even to aid in conjunction with other specialized techniques in identifying fraudulent reviewers that are giving multiple highly mismatched reviews.

There are two key assumptions on which this work is based on. The first assumption is that the free-form text of a review is considered a much more informative source of a user's opinion about the offering compared to the given review rating. The second assumption is that the mismatching of review ratings is not systematic, but instead has a random nature.

We can say that the first assumption holds true, because in the free-form text part of the review we can generally find specific information about the user's experience with the product or service, as well as details about different aspects of the offering that may be important to the reviewer. Furthermore the second guesstimate is also valid, because empirically we can observe both reviews for which we would expect a higher corresponding star rating, and reviews for which we would expect a lower corresponding star rating. The latter assumption is also supported by the results presented by [1], where it was shown that average Google Play app ratings both increased for some apps and decreased for some others, when the original ratings were re-calculated with a model that was trained on human annotated examples that followed a specific rating guideline.

Training a review rating prediction model with a large amount of training examples is essentially finding a transformation from free-form review text to review ratings that is consistent with the average trend of the data. Since the majority of the data is considered to correspond to correct review ratings, such a model could then successfully be applied to identify review rating mismatches, to correct them, and to produce a quantitative measure from free-form text-only reviews, thus leading to the potential benefits mentioned earlier.

The rest of the paper is organized as follows: in section II we present the relevant work in this field, in section III we will analyze the dataset and the algorithms we relied on for the review rating prediction task. We will then present the findings from the various experiments in section IV and we

[1]**Source code**:
https://github.com/stergiosbamp/nlp-review-rating-prediction

conclude our work in section V.

II. Related Work

Although review rating prediction is a relatively new natural language processing problem, research has been conducted in this area.

Supervised machine learning algorithms on data from “we8there.com” have been used by [2]. The data contained 6823 restaurant reviews consisted of free-form text as well as ratings (1-5) of different aspects (atmosphere, food, value, service, overall) of the restaurant being reviewed. They used a different model for predicting the rating of each different aspect from the free-form text. Essentially this is the same task repeated 5 times with the same features as input but different target ratings. They tried using uni-grams, bi-grams, word chucks, and part-of-speech chunks as features. They adopted the use of uni-grams as it was shown to be the best performing among the tested features. Another key finding was that addressing the task as a classification problem yielded better results than addressing it as a regression problem, even though the prediction of ratings may look like more as a regression problem. The “MaxExt” classifier was found to be the best performing model. The average ranking loss across the rating prediction for the 5 different aspects was 0.637.

[3] has researched similar approaches on the Yelp 2014 Challenge dataset. The research was limited only on restaurant reviews, which amounts to 706,646 reviews for 14,403 restaurants. The author tested 16 models which are all possible combinations between 4 features and 4 machine learning algorithms. Specifically the tested features were uni-grams, uni-grams plus bi-grams, uni-grams plus bi-grams plus tri-grams, and latent semantic indexing. The tested supervised machine learning algorithms were Logistic Regression, Naive Bayes classification, Perceptrons, and Linear Support Vector classification. Tf-Idf weighting was used to adjust the counts of the features. An important finding is that including all the available features results in diminishing returns in terms of performance compared to using only a percentage of the most common features. When using the Naive Bayes algorithm including more features after a certain point even resulted in worse performance. Using the cross-validation scores, the best performing model was the logistic regression algorithm in combination with the top 10,000 uni-grams and bi-grams as features with 64% accuracy. On the test set the same model achieved 54% accuracy and 0.92 Root Mean Squared Error (RMSE), while the linear support vector classification with the same features achieved 56% accuracy but 1.05 RMSE.

A different problem formulation was followed by [4] on a dataset consisted of 9,206 Amazon reviews for 8 films. They ignored 4-star reviews and turned the problem into binary classification, predicting whether a review has high (5-stars) or low (1, 2, or 3 stars) sentiment. They used the “NLTK” Python library to tokenize the sentences and to remove stop words, ignoring words with a count lower than 10. They tested the top-500 and top-900 words by Tf-Idf as well as the top-200, top-600, top-900 and top-1000 words by info-gain as features. They also tested using the top sentiment words with a count larger than 5 as features. They tried all resulting combinations of the above features with the Support Vector Machine classification and Naive Bayes classification algorithms. The best performing model on the training set was the Support Vector Machine classification algorithm using the top-600 info-gain words as features, with 81.9% accuracy, 90% precision, 92% recall and 90% F1-score. On the test set it achieved a 77% accuracy, which is slighlty worse than the same algorithm with the top-200 info-gain words as features that achieved 78% accuracy.

The review rating prediction problem has also been studied for Google Play apps by [1]. The key difference in this work is that they used human annotators to manual re-annotate a dataset of 8600 app reviews following specific annotation guidelines. They tried various machine learning algorithms from Weka as well as a deep learning approach. Specifically, from Weka they tried j48, Naive Bayes, OneR, AdaBoost M1, Logit Boost, Decision table, Decision stump, Ibk, and SMO, while the deep learning approach was a Dependency-based Convolutional Neural Network (DCNN). They experimented with using hand-crafted features and “GloVe” embeddings. A notable result was that most machine learning models performed better using the “GloVe” embeddings compared to the hand-crafted features. All models were trained on the human annotated data, and then tested on 115 new reviews provided by 23 users. The deep learning model was the best performing one, achieving 92% accuracy. The best from the machine learning models using hand-crafted features was the instance based classifier (Ibk), achieving 74.9% accuracy. The best from the machine learning models using word vectors was the OneR classifier, achieving 84.52% accuracy. A key finding was that the selected trained model was able to achieve comparably high performance on different review datasets (Amazon MP3 reviews, TripAdvisor reviews) without any additional training or fine-tuning. It was able to generalize from the human annotated examples to the other datasets, on which it achieved 88% and 86% accuracy respectively.

Another recent approach [5] explored Recurrent Neural Network (RNN)-based models for the rating prediction problem. They used restaurant reviews from Yelp and e-book reviews from Amazon. The pre-processing included convertion to lowercase and removing stop- words, hashtags, URLS, dates, and times. “GloVe” embeddings were used as the features. Specifically, they choose the “GloVe” embeddings that have been trained on 660K words of Twitter, because according to the authors that dataset better reflects the language of reviews. They tried a simple Bidirectional Gated Recurrent Unit (Bi-GRU) neural network and a simple Bidirectional Long Short Term Memory (Bi-LSTM) neural network. They also proposed a Bi-GRU-based model with four additional layers. The total five layers of the proposed model are word embeddings, 1-d spatial dropout, Bi-GRU, concat maximum pool and average pool, and dense output layer. They used Amazon digital e-book reviews to evaluate the models. They also investigated the effect of balancing the percentage of the given star ratings using 10-fold random cross validation. In both cases the proposed model achieved better performance compared to the the other two state-of-the-art deep learning approaches. On the un-balanced data it achieved 69% precision, 70% recall, 68% F1-score, and 0.65 Root Mean Squared Error (RMSE), while on the balanced data it achieved 65% precision, 68% recall, 65% F1-score, and 68% RMSE.

III. Dataset & Models

A. Dataset

We use Amazon review data for our work which are extracted from datasets provided by [6]. These datasets come in JSON Lines format, where each line of the file is a valid JSON object that represents a review. From each JSON object we extract the free-form text of the review and the given star rating in order to form the examples that we will use to train and evaluate our models.

Each of the original files provides data for different product categories. In our work we choose to focus on the "software" product category. After extracting the examples and before proceeding with the pre-processing we drop any duplicate reviews that are present in our extracted dataset. Thus, starting from 12,805 reviews we end up with 10,662 reviews after removing all duplicates.

All tasks related to extracting the required examples for our work are done in Python. We use parameterized code that takes as input the URL of the original dataset and the fields of the JSON objects that correspond to the data and the target of the examples that we want to construct. The original dataset is then downloaded and unzipped, and the requested dataset is constructed. There is also the option to drop duplicates. This gives us the ability to create our dataset on the fly, which is helpful if there is a need to use any cloud services such as Google Colab notebooks.

B. Models

1) Pre-processing: As features for the machine learning algorithms we use the Tf-Idf weighted word counts. The values produced by Tf-Idf weighting are proportional both to the count of a word in a document, and to the inverse of the count of documents that the word appears in. We choose the Tf-Idf weighting as it's one of the most popular and tested term-weighting schemes, and has also been successfully used in the relevant work that we reviewed earlier.

We explore the use of 3 different tokenizers to process the free-form review text. Specifically, we try the built-in tokenizer of the TfidfVectorizer class from the scikit-learn library, the NLTK tokenizer in combination with the PorterStemmer, and the lemmatizer from the "en_core_web_sm" language model of the "spaCy" library. We make an end-to-end evaluation of different tokenizers in combination with different machine learning models for our task.

2) Machine learning models: Our approach for the review rating prediction will be modeled as a classification task rather than as a regression problem since the related work showed that approaching it as a classification task provides better results.

We try 3 popular supervised machine learning models for classification from the "scikit-learn" library. Specifically, we try the Multinomial Naive Bayes classifier (MultinomialNB), the Logistic Regression classifier (LogisticRegression) and the Linear Support Vector Machine classifier (LinearSVC).

The Multinomial Naive Bayes classifier is a popular algorithm for document classification problems. The multinomial variation assumes that the features are distributed multinomially, while the naive assumption assumes conditional independence between all features. Given a document that corresponds to the free-form text of a review, the Naive Bayes algorithm calculates the joint probability of the observed features for each possible review rating class. The predicted rating is the one that corresponds to the highest probability, when the prior of the ratings is also considered. The Naive Bayes classifier is a generative machine learning model, since its predictions are based on the probability of the observed features being generated by the different review rating classes. In practice some form of smoothing is also used on the counts of the training data to better generalize on unseen examples.

The Logistic Regression classifier is another popular algorithm used in a variety of classification problems. Contrary to the Naive Bayes classifier, the Logistic Regression classifier is a discriminative machine learning model. Given a document that corresponds to the free-form text of a review, the Logistic Regression classifier calculates the probability of each review rating using a probability function that depends on the features of the document. It essentially learns the probability of a sample free-form text belonging to a certain review rating class. In practice some form of regularization is also used on the weights to avoid overfitting.

The Linear Support Vector Machine classifier is the third algorithm that we examine for document classification. The Linear Support Vector Machine classifier makes predictions by finding a hyperplane on the multi-dimensional feature space that maximizes the margin between the classes that are being separated by the boundary hyperplane. Since a perfect linear separation is rarely possible, in practice the decision boundary is found by balancing between maximizing the margin and minimizing the number of misclassified examples.

3) Embeddings: We also research the effect of using vector embeddings as features instead of Tf-idf weighted word counts. We investigate both the use of word-level embeddings and the use of document-level embeddings. Specifically, we try "GloVe" embeddings pre-trained on Twitter data, training custom Word2Vec embeddings on our own dataset, and using Doc2Vec embeddings. For all three cases we use the "Gensim" Python library.

Embeddings aim to solve the limitations of the bag-of-words features. With bag-of-words all information regarding the ordering of the words is lost and the semantics of the words are ignored, which results in a representation where all words are equally distant. In contrast, Word2Vec embeddings are learned in a way that models word associations present in a corpus and as a result preserve part of underlying knowledge related to the structure of language. Additionally, Doc2Vec learns dense representations of documents that are trained to predict the words in a document.

When using the Doc2Vec embeddings which are document-level embeddings, we don't need any additional processing before using the embeddings as input features for the machine learning algorithms, since Doc2Vec always produces a constant size vector independent of the number of tokens present in the document. In contrast, when using word-level embeddings we need a way to aggregate the variable number of corresponding embeddings of each document into a feature space of constant size that can be passed as input to the machine learning algorithms. We adopt the simple but widely used solution of calculating a mean embedding of the same size from all individual word embeddings of each document.

We test the various embeddings in combination with the Logistic Regression classifier, evaluating their performance on the given task.

IV. Experiments

A. Effect of different pre-processing methods when using Tf-Idf weighted word counts

The first thing that we investigated was the effect that different pre-processing methods would have when using Tf-Idf weighted word counts as features for the machine learning algorithms. We measured the required classification time, which is composed of the time required for the training and of the time required for inference. We also calculated the weighted F1-score which can be used as a measure of overall performance when comparing the different pre-processing methods. The comparison of the pre-processing methods was done using 3 machine learning algorithms, the Logistic Regression classifier, the Linear Support Vector Machine classifier, and the Multinomial Naive Bayes classifier. The measured classification times and the calculated F1-scores are presented in figures 1 and 2.

As seen from figure 1 there is a large difference in the required time for the different pre-processing methods. The built-in tokenizer of the "TfidfVectorizer" class from "scikit-learn" is the fastest of the 3. When using the lemmatizer of the "en_core_web_sm" model from "spaCy" the required time increases by about one order of magnitude. Similarly, when using the "NLTK" library for the tokenization and stemming the required time increases by an additional order of magnitude. This holds true for all 3 machine learning algorithms examined.

In contrast, as seen from figure 2, no significant difference in performance in terms of weighted F1-score can be observed when using different pre-processing methods. This leads us to conclude that the specific pre-processing method used to tokenize and process the input text has no meaningful role, at least when using the Tf-Idf weighted word counts as features

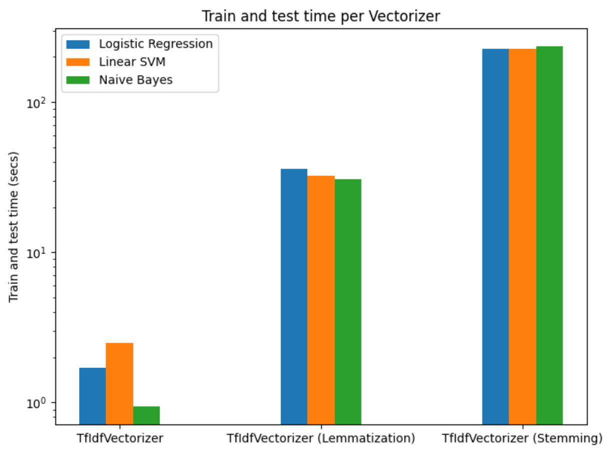

Fig. 1. Classification task time for different pre-processing methods and classification algorithms

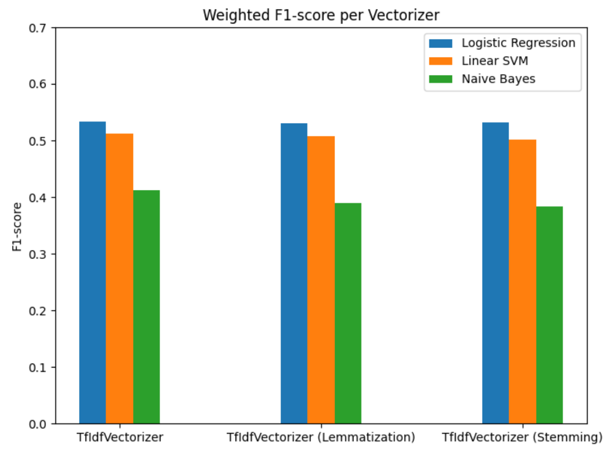

Fig. 2. Weighted F1-score for different pre-processing methods and classification algorithms

in combination with common machine learning classification algorithms.

B. Comparison of different machine learning algorithms for classification

The second thing that we investigated was the effect of different machine learning algorithms for classification on relevant performance metrics. For each algorithm we track standard classification metrics, Accuracy, Precision, Recall, F1-score, as well as the Mean Absolute Error (MAE). The algorithms that we evaluated was the Logistic Regression classifier, the Linear Support Vector Machine classifier, and the Multinomial Naive Bayes classifier. Grid search was used to tune the most important hyperparameters of each algorithm. For the pre-processing the "TfidfVectorizer" with its built-in tokenizer was used. The performance of each algorithm of the classification task in shown in table I.

TABLE I
PERFORMANCE OF DIFFERENT CLASSIFICATION ALGORITHMS (WEIGHTED METRICS)

Performance	Accuracy	Precision	Recall	F1	MAE
Logistic Regression	**0.5664**	**0.5391**	**0.5664**	**0.5337**	**0.6852**
Linear SVM	0.5577	0.5187	0.5577	0.5129	0.7074
Naive Bayes	0.4861	0.5076	0.4861	0.4125	0.9331

As seen from table I the best performing algorithm is the Logistic Regression classifier, which surpasses the others on all 5 tracked metrics. The second best with slightly worse performance is the Support Vector Machine classifier, while the Naive Bayes classifier comes last with substantial drop in performance. Although the Logistic Regression classifier is a relatively simpler classification algorithm than the Support Vector Machine classifier, it achieves slightly better performance, a fact that was also observed in relevant work.

C. *Effect of class imbalance*

The third thing that we investigated was the effect of class imbalance in the performance of the classification algorithms. Class imbalance as seen from relevant work is not a surprising phenomenon among review datasets. This also holds true for our dataset. The distributions of the classes can be observed in figure 3.

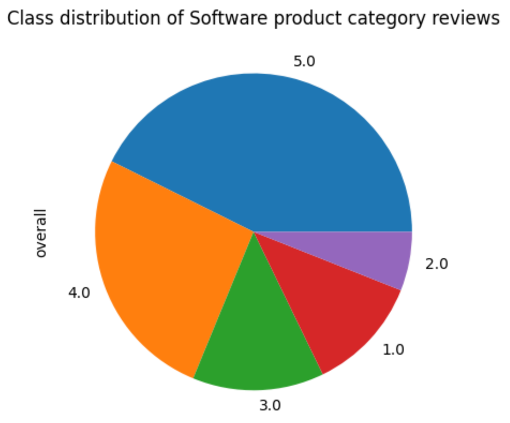

Fig. 3. Class distribution among reviews from the "Software" category

As seen from figure 3 the most common rating is 5-stars, with 2-stars being the least represented class in the dataset. Since the review examples are consisted of free-form text and because the size of our dataset is relatively modest, we resorted to using the "RandomOverSampler" from "imblearn". Using the oversampling approach all but the majority classes were sampled with replacement until the class distribution became even. The weighted F1-score was used to compare the performance of the classification with and without class balancing. The results are shown in figure 4.

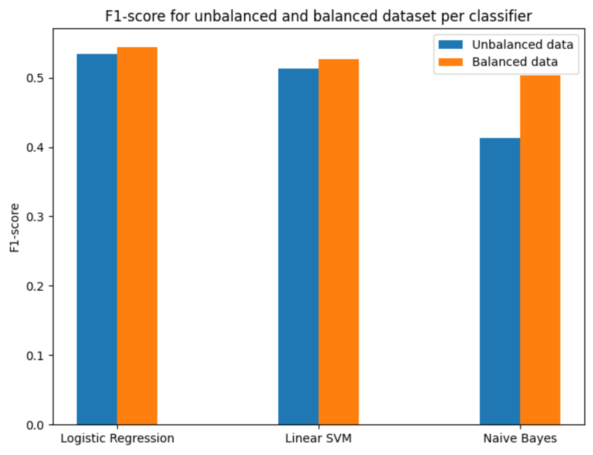

Fig. 4. F1-score for different classification algorithms before and after random oversampling

As seen from figure 4 balancing the classes results in improved performance in terms of weighted f1-score for all classification algorithms. The Naive Bayes classifier captures the most gains in performance after balancing the classes, while the Logistic Regression classifier and the Linear SVM classifier seem to be more robust to class imbalances. The performance of each algorithm on the balanced data is presented in greater detail in table II.

TABLE II
PERFORMANCE OF DIFFERENT CLASSIFICATION ALGORITHMS WITH CLASS BALANCING (WEIGHTED METRICS)

Performance	Accuracy	Precision	Recall	F1	MAE
Logistic Regression	**0.5436**	**0.5469**	**0.5436**	**0.5442**	**0.6818**
Linear SVM	0.5277	0.5283	0.5277	0.5264	0.7318
Naive Bayes	0.4933	0.5230	0.4933	0.5028	0.7937

As seen from table II the best performing algorithm is again the Logistic Regression classifier, with the Support Vector Machine classifier coming second. The Naive Bayes classifier is again last in performance, but the performance gap after balancing the data in considerably smaller.

D. *Effect of using embeddings as features*

The last thing we investigated was the effect of using different types of embeddings as features for the Logistic Regression classifier. We examined the use of pre-trained 100-dimensional "GloVe" embeddings, as well as training Word2Vec embeddings from scratch on the existing dataset. We also examined the use of Doc2Vec embeddings which are document-level embeddings, in contrast to the previous two that are word-level embeddings. For the comparisons we used the standard classification metrics, accuracy, precision, recall, and F1-score. The performance in the classification task when using embeddings compared to using the Tf-Idf weighted word counts in combination with the Logistic Regression classifier is presented in figure 5.

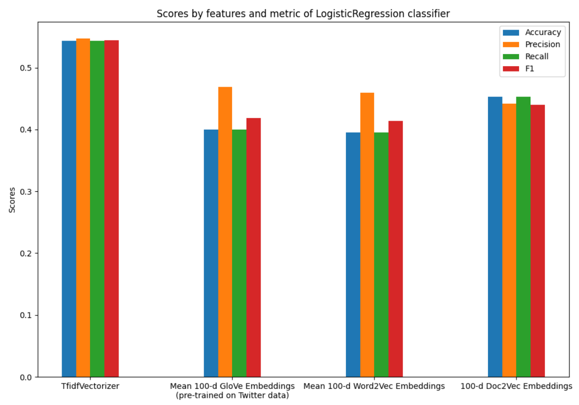

Fig. 5. F1-score for different embedding methods using the Logistic Regression classifier

As seen from figure 5 the pre-trained "GloVe" embeddings have lower performance that the Tf-Idf weighted word counts. This can be attributed partly to the low vocabulary overlap between the embeddings and our dataset, with 38.55% of the vocabulary of the dataset not being present in the vocabulary of the embeddings. It can also be party attributed to the averaging of multiple embeddings in order to have a fixed size of input features. The performance of Word2Vec embeddings is also lower than the Tf-Idf weighted word counts. This is also attributed party to the averaging method applied on the word-level embeddings, but also on the limited size of the dataset. We would expect an increase in performance on a larger dataset. Finally, Doc2Vec embeddings perform better than both "GloVe" and Word2Vec embeddings, but still worse than the Tf-Idf weighted word counts. Perhaps document-level embeddings perform better than word-level embedding when the available data is less than what is required for these methods to scale to their full potential.

V. Conclusions

A. Conclusions

In this paper, we studied and researched the problem of review rating prediction from free-form review test, which was formulated as a multi-class classification problem. We tried different pre-processing and tokenization methods for the vectorization of the text in combination with different classification algorithms. We found that while the pre-processing methods didn't have an effect on the performance of the algorithms, the built-in tokenizer of the "TfidfVectorizer" was orders of magnitude faster than the other examined pre-processing methods and should be preferred, at least when applying standard machine learning algorithms. We also found that the Logistic Regression classifier was the best performing one, with the Linear SVM classifier being slighty worse, and the Naive Bayes classifier having considerably worse performance on the unbalanced dataset. We tried balancing the data, which resulted in slight performance improvements for all three classification algorithms, with the ranking between the algorithms remaining the same. The Logistic Regression classifier and the Linear SVM classifier seem to more robust to class imbalances than the Naive Bayes classifier. We also examined the use of embeddings, which achieved worse performance than the Tf-Idf weighted word counts. For the "GloVe" embeddings this was partly attributed to the low vocabulary overlap, with 38.55% of the vocabulary of the dataset not being found in the vocabulary of the embeddings, as well as perhaps to the averaging used on the embeddings to result in a fixed size feature input. For the Word2Vec embeddings apart from the use of the averaging method, decrease in performance was also attributed on the small number of available data. For the Doc2Vec embeddings the lower performance was solely attributed on the small number of the available data, since Doc2Vec embeddings are document-level embeddings and don't require the use of the averaging method. We also conclude that document-level embeddings might have potentially better performance than word-level embeddings when both are trained on limited data.

B. Feature work

One of the directions for feature work is the application of the best methods on a product category that has a much larger number of reviews. Something that we also consider an interesting approach is the application of the methods on a dataset that consists of multiple product categories, as well as training the models on one or more product categories and testing them on one or more unseen product categories. Finally, we consider that the application of state-of-the-art models such as BERT with some fine-tuning on the specific dataset, could potentially provide interesting outcomes.

References

[1] R. Aralikatte, G. Sridhara, N. Gantayat, and S. Mani, "Fault in your stars: an analysis of android app reviews," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2018, pp. 57–66.

[2] N. Gupta, G. Di Fabbrizio, and P. Haffner, "Capturing the stars: predicting ratings for service and product reviews," in *Proceedings of the NAACL HLT 2010 workshop on semantic search*, 2010, pp. 36–43.

[3] N. Asghar, "Yelp dataset challenge: Review rating prediction," *arXiv preprint arXiv:1605.05362*, 2016.

[4] M. Kavousi and S. Saadatmand, "Estimating the rating of the reviews based on the text," in *Data Analytics and Learning*. Springer, 2019, pp. 257–267.

[5] B. H. Ahmed and A. S. Ghabayen, "Review rating prediction framework using deep learning," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2020.

[6] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 188–197.