# Data 553 Project

Akash Dhatavkar, Christian Hluchy and Connor Lee

## 1. Introduction

Algorithms can be implemented in many ways in order to provide insights that can be used to improve the effectiveness of businesses. However, it is important that these algorithms provide robust and reproducible results, both, in the short term and in the long term. Any long term inconsistency could result in significant mistakes occurring during the decision making processes of a business. The 2016 paper "On the Automatic Classification of App Reviews" discusses the results of the various methods they employed to automatically classify app review types [1]. This paper will be referred to throughout this report as the paper. This report will discuss the methods employed to reproduce the results of three of the classification methods used in the paper, the robustness of the algorithms used in the paper, and any discrepancies between the results from the paper and from this report.

## 2. Pre-Processing Procedures

The datasets provided by the authors of the paper were used as the training sets. Each training dataset was tagged based on the file name (E.g. we created a column called 'Bug' in the bug_tt file and tagged the row as 1 if the comment was related to a bug, else it was 0). The testing dataset was sampled from the Data 542 project dataset. The data was filtered to contain only English comments.

We decided to use the following three classification techniques for the reasons discussed below:

- Bigrams: Highest Recall for Bug Reports and User Experience
- Bigrams + Bag of Words (BOW) - Stop Words + Lemmatize : Highest F1 Score for Bug reports, Feature Requests, and Ratings
- BOW + Lemmatize + Rating: Highest Precision for Feature Requests

For the Bigrams classification, a simple Bigram CountVectorizer was used for pre-processing. For Bigrams + BOW - Stop Words + Lemmatize the order in which the methods were applied was important since the Bigrams and BOW process reduced the data to a sparse dataframe. As a result lemmatization was applied first, then bigrams, bag of words, and stop words were applied with one function call through the parameters passed into the CountVectorizer function. Finally for BOW + Lemmatize + Rating, the user ratings included in both the training and testing datasets were used as the Rating attribute. The text was first Lemmatized and then a BOW model was run on the lemmatized words. In order to incorporate the ratings, we needed to convert the ratings column into a sparse matrix and then stack it onto the BOW model.

# 3. Development of Testing Dataset

In order to compare the results of the original paper a testing dataset was required, the testing set was a subset of manually classified reviews from the data provided during the Data 542 project (the sample dataset). The sample size of the data was calculated to be 384 using the Creative Research Systems sample size calculator [2]. The parameters passed into the sample size calculator were the confidence level of 95%, the confidence interval of 5, and the population size of 834,042 which was pulled from the sample dataset. The sample dataset was randomly sampled for the 384 reviews in order to produce the testing dataset. The testing data was broken into three 256 review blocks, two group members were assigned to each of the blocks, such that each review would be read, and classified, by two group members. The provided coding guide was used as a reference in order to standardize the category classification of reviews, the categories described were User Experience, Bug Report, Feature/Improvement Request, and Rating. Any review that had attributes belonging to multiple categories were classified as all of the applicable categories. Each group member then went through the 128 reviews that the other two group members had labeled. Any labeling that agreed was taken to be correctly labeled, any disagreements were resolved by the 3rd group member.

# 4. Prediction Performance

For our prediction, we decided to use the Naive Bayes Model as the paper demonstrated it to provide superior results [1]. The resulting Precision, Recall, and F1 Score from predicting the app classifications using this model on the new test data after each of the three text processing methods were applied to each category is shown in Table 1 below.

| | | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Bigrams | Bug | **0.29** | 0.71 | **0.41** |
| | Feature | **0.15** | **0.69** | **0.24** |
| | User Ex. | 0.41 | **0.62** | **0.49** |
| | Ratings | 0.75 | 0.68 | 0.71 |
| BOW + Bigrams - Stopwords + Lemmatization | Bug | 0.28 | **0.77** | **0.41** |
| | Feature | 0.13 | 0.56 | 0.21 |
| | User Ex. | 0.43 | 0.58 | **0.49** |
| | Ratings | 0.78 | 0.7 | 0.74 |
| BOW + Rating + Lemmatization | Bug | 0.25 | 0.64 | 0.36 |
| | Feature | 0.13 | 0.5 | 0.2 |
| | User Ex. | **0.45** | 0.55 | **0.49** |
| | Ratings | **0.8** | **0.73** | **0.76** |

Table 1: Summary of Prediction Results on New Test Data

The maximum values for each of these measures for each type of app classification are identified with bold numbers in Table 1. It should be noted that the maximum F1 score for Bug reports occured in both the Bigrams, and the BOW + Bigrams - Stopwords + Lemmatization cases, and that all three methods of text processing provided the same F1 score for app reviews that were classified as User Experience.

## 4.1 Prediction Performance Comparison

The above test results are compared to the paper in Figures 1, 2, and 3 for Bigrams, BOW + Bigrams - Stopwords + Lemmatization, and BOW + Rating + Lemmatization respectively.

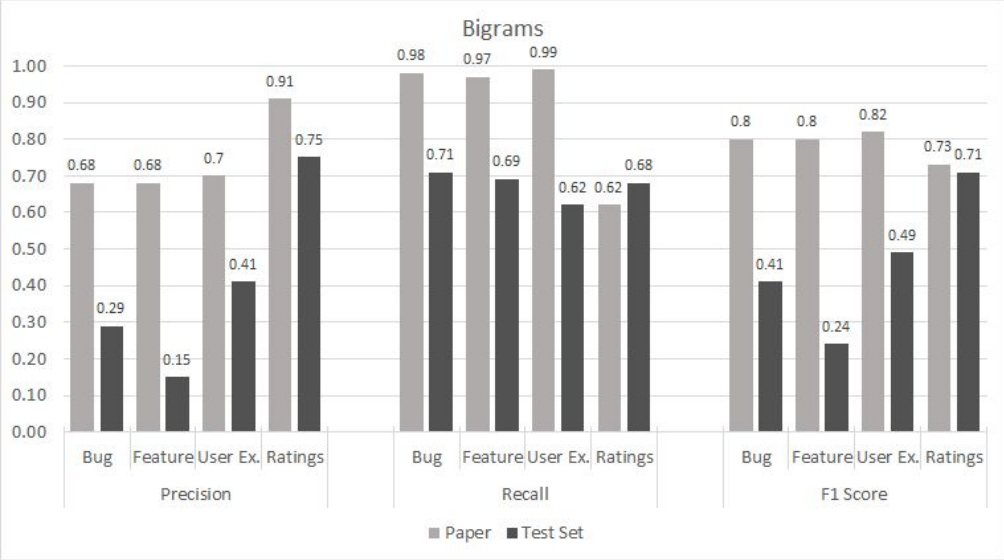


Figure 1: Bigrams Comparison of Results [1]

From Figure 1, above, it is clear that the prediction results from the test set, when Bigrams were applied, underperformed when compared to the results from the paper by a significant margin, with the exception of the Recall on Ratings (which outperformed the paper), and the F1 Score on Ratings (which underperformed when compared to the paper by 0.02). Additional points of interest include the Precision for Bug Reports (0.29), the Precision for Feature Requests (0.15), the F1 Score for Bug Reports (0.41), and the F1 Score for Feature Requests (0.24) which underperformed when compared to the paper by 0.39, 0.53, 0.39, and 0.56 respectively.

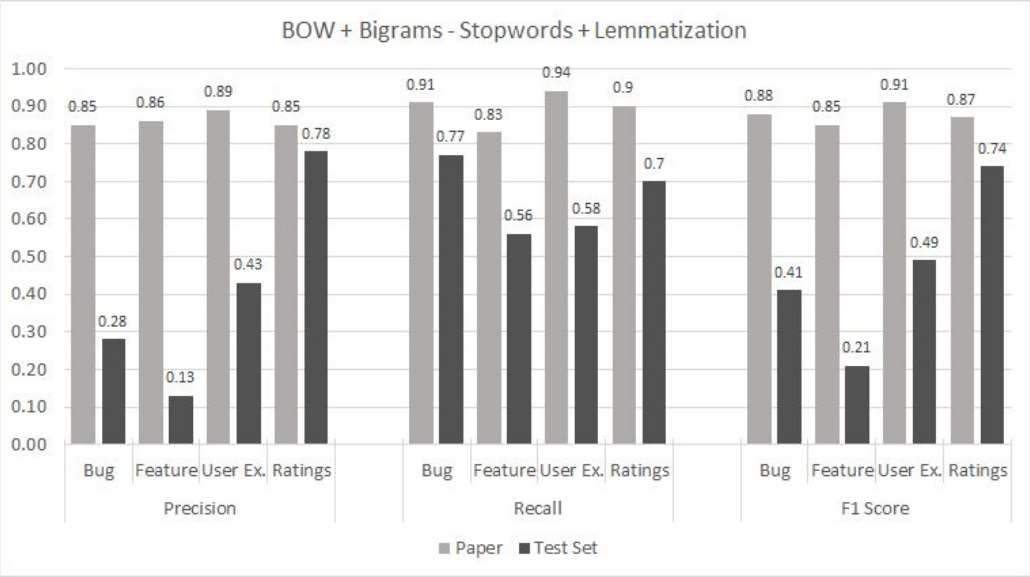


Figure 2: BOW+Bigrams-Stopwords+Lemmatization Comparison of Results [1]

From Figure 2, above, it is clear that the prediction results from the test set, when BOW+Bigrams-Stopwords+Lemmatization were applied, underperformed when compared to the results from the paper by a significant margin with the exception of the Precision for Ratings, which underperformed by 0.07. Additional points of interest include the Precision for Bug Reports (0.28), the Precision for Feature Requests (0.13), the Precision for User Experience (0.43) the F1 Score for Bug Reports (0.41), and the F1 Score for Feature Requests (0.21) which underperformed when compared to the paper by 0.57, 0.73, 0.46, 0.47, and 0.64 respectively.


Figure 3: BOW+Rating+Lemmatization Comparison of Results [1]

From Figure 3, above, it is clear that the prediction results from the test set, when BOW+Rating+Lemmatization were applied, underperformed when compared to the results from the paper by a significant margin with the exception of the Precision for Ratings, which outperformed the paper, and the F1 Score for Ratings which underperformed by 0.05. Additional points of interest include the Precision for Bug Reports (0.25), the Precision for Feature Requests (0.13), the Precision for User Experience (0.45) the F1 Score for Bug Reports (0.36), and the F1 Score for Feature Requests (0.2) which underperformed when compared to the paper by 0.63, 0.73, 0.45, 0.44, and 0.54 respectively.

A point of note with regards to Figures 1, 2, and 3 is that all three text processing methods performed reasonably well on reviews that were classified as Ratings, and on average performed better on Recall than on the Precision or F1 Score. The notably better results associated with the Ratings category could be attributable to various factors including the use of emotional language such as “I love it” or “I hate it”, or the relatively larger number of reviews that were classified as Ratings in the test set since, as shown in Table 2 below, the number of Reviews classified as Ratings was significantly higher than any of the other classification types.

| Category | Review Count | % of Total |
|---|---|---|
| Feature | 36 | 8.0% |
| Bug | 56 | 12.5% |
| User Ex. | 115 | 25.6% |
| Ratings | 242 | 53.9% |

Table 2: Summary of the Number of Reviews Per Category in the Testing Dataset

Table 2 also helps explain the notably poor performance when predicting Feature Requests, since Feature Requests account for only 8% of the total review classifications, as a result any misclassified Feature Request would result in a larger reduction on the prediction results than a misclassified Rating would.

# 5. Discussion of Library Robustness

Since we were unable to reproduce the results of the paper, with rather large discrepancies in some of the categories, we need to question what could be potential causes of that have led to these differences. Broadly speaking, most differences between the paper and the project are related to either the methodology or the actual data used to train and subsequently test the model.

Starting with methodology, we made the decision to write our own code to pre-process the text, utilizing packages such as NLTK and scikit-learn. This was driven by our desire to learn new coding strategies, and the fact that the code from the paper was so poorly documented. If the

paper was more concerned with reproducibility, the authors should have spent time to explain what their various classes and functions did, instead of supplying the code undocumented. While stop words removal, lemmatization, etc are similar between the paper's code and ours, small differences could have had an impact on results. Further to this, the paper's results are the result of Monte Carlo cross validation on a curated data set, while the project used a simple train/test set methodology as stated in the project outline [1]. Other methodologies were made as consistent as possible, such as employing Naive Bayes for the binary classification.

The larger, and likely more problematic difference between the paper and the project was the difference in the test sets used to gauge model success. In building the data set for the paper, several stringent rules were applied [1]. These rules are of additional importance because the paper used cross validation, thus the characteristics of the training set are the same as the test set [1]. In curating the data set, effort was made to ensure things were balanced. Thus, there were equal representations of app reviews sourced from the Apple App Store and the Google Play Apps, free vs. paid apps, and star ratings (one through five). This rigour is wholly appropriate in building a good training data set, however when the test set is generated by a random 70/30 split of the data, the test set will inevitably be more balanced than it would be in a real world setting.

Conversely, for the test set used in the paper, many of the aforementioned criteria were not met [1]. To start, the data was sourced entirely from the Google Play Apps, and as far as we can discern no thought was put into paid vs free apps. This is of particular interest as the nature of criticism or praise could be influenced by any monetary value (ie. if a $10 app doesn't work, it's far more frustrating than having issues with a free one). When selecting the specific test set from the broader dataset, we simply chose a sample of 384 reviews at random.

While the sources of the data and any subsequent curating may challenge the reproducibility of the results, it's our view that the primary concern is attempting to replicate the model is the passage of time. Since we were instructed to train the model on the dataset from the paper, we were using app reviews from 2014 [1]. We then in turn used app reviews predominantly from 2018 as the test set. The market for apps has exploded over that duration, as world wide revenues for app sales have increased by over 260% between 2014 and 2018 [3]. As computing power of phones increased, the complexity of apps has changed, as have user expectations. It's reasonable to assume that using a training set from 2014 on 2018 apps reviews could cause problems. It is also reasonable to assume the possibility of "Ratings"' to have taken on potentially different meanings over the years. We know that the language used in app reviews are specific to the kind of app, so if our portfolio of apps changes, it's reasonable to expect that our reviews would change as well. The specificity is mentioned in the paper, as the authors comment on how gaming app reviews are particularly challenging for bug and feature requests, as the language is quite specific to the app category [1]. Overall, the disconnect between 2014 and 2018 is an example of drift, where small changes over time can accumulate and make a model less effective unless updated with new training data.

Even if the portfolio of apps in 2018 was similar to 2014, there's another challenge to reproducibility, which is simply the fact that language changes over time. Whether slang, or abbreviations, in four years small changes could influence classifications, especially for new words that don't exist in the training data at all. A final change that could impact the language with which reviews are written in 2018 vs 2014 is the increasing knowledge of how algorithms are utilized. As discussed in the online course, as soon as humans know how an algorithm is being employed (in this case compiling and flagging online reviews), they change their behavior. So an individual writing a review in 2018 may use specific language to raise the profile of a review, using knowledge they may not have had in 2014.

Turning to the robustness of the model presented in the paper, since we used our own code to classify the apps, we weren't in a situation to evaluate how well the author's code handled new data. As stated previously, the fact that the code was poorly documented meant that anyone hoping to assess the quality of the model would be required to sift through line by line and then read in their data in a similar manner. If we had decided to use their code, in testing the robustness it would have been preferable to better align the training data and test data, thereby removing the model drift mentioned previously.

# 6. Group Member Roles

Each group member manually classified 256 reviews, and provided a 3rd party review of the 128 reviews that the other two group members had classified. Each group member also wrote one of the three text pre-processing functions and participated in the writing of the report. The unique contributions performed by each group member are explained below.

Akash looked into the possibility of reusing the algorithms provided by the authors of the paper. He managed to convert the Python 2.7 code into Python 3 format and also created the BOW + Lemmatize + Rating classification algorithm.

Christian was responsible for the bigrams function. He looked into the tutorial recommended by Dr. Fard to familiarize the group with count vectorizer and binary classifications.

Connor was responsible for the Bigrams + BOW - Stop Words + Lemmatize functionality. He also did various proof of concept tests to confirm that creating the desired functionality in Python 3, rather than using the provided Python 2.7 code, was feasible. Through this proof of concept work he was able to get functionality for Bigrams, BOW, Stop Words, Lemmatization and Stemming working. The Stemming functionality was ultimately not implemented into the project.

# 7. Additional Comments

Through the process of developing the pre-processing code, and training and testing the model various additional points of interest were identified. These additional observations are discussed below.

Originally some minor pre-processing including changing all review text to lower case was considered. However this was abandoned due to a notable decrease in Precision, Recall and F1 score occuring when the text was changed to all lower case prior to applying any of the text processing (bigrams, BOW, etc…). This was surprising as one would expect that by changing all letters to lowercase would reduce the total number of unique words, and increase the occurrences of more common words in the Bigrams and the Bag of Words processes, which would eventually lead to an improvement in the prediction capabilities of the Naive Bayes model. From this observation it appears as though the use of upper/lower case in reviews is in some way related to the type of review. Presumably bug reports and ratings would be more likely to have capital letters as people tend to express anger or excitement in text by typing in capital letters.

Our test set was randomly sampled, as a result, there was a disparity in the number of reviews from each group. Ratings had the bulk of the reviews and hence we could see better results achieved for Ratings.

Recognizing that reviews are often specific to the category of app, a natural next step would be to create models specific to a given category, then test their ability to classify into the four categories. For example, knowing that gaming reviews have specific language, it would be interesting to train a model solely with that category of app and see what changes arose in terms of precision and recall.

In attempting to reproduce the results of the paper, a major source of error was the subjective nature of the labelling process, which involved a large human element. It was good to see the rigour put forth to curate a “golden set” with peer agreed labels, but the coding guide given to the master’s students would have benefitted from clarification [1]. In the coding guide, an example of each classification is provided but at no point is an example given where a review can be classified as more than one option. For example, under user experience the review “Great for daily readings, I love it.” is given, with no mention that it could be classified as rating as well. To improve this process we’d recommend having multiple examples where a review fit more than one classification. Without clear direction, labellers are bound to make a high level assumption and then apply it across hundreds of reviews.

# 8. Conclusion

The robustness of an algorithm, and the accurate reproducibility of results is of extreme importance to businesses when the results of the algorithm are being used to make high impact decisions. As a result it is important to test the robustness of algorithms. One way to do this is to have a third party implement the same algorithm, and attempt to obtain the same results. This report covered the implementation of the algorithm discussed in the 2016 paper "On the Automatic Classification of App Reviews" discussing the automatic classification of app reviews using various Python 3 libraries, instead of the Python 2.7 library produced by the authors of the paper [1]. The results from the Python 3 implementation, in most cases, significantly underperformed the results from the paper. There are various possible explanations including the evolution of the english language between the times the training and testing sets were collected (2014 and 2018 respectively), the comparatively small number of reviews used for testing the Python 3 model, and the fact that the training data was collected from both the Apple App Store and Google Play Apps, while the testing data was just collected from Google Play Apps.

Additional investigation is required in order to identify if the library from the paper is sufficiently robust to accurately predict app review categories now. Such an investigation should start by generating a larger dataset of categorized app reviews consisting of apps from both the Apple App Store and Google Play Apps. These reviews should be confined to a set time period, and they should be randomly split into training and testing sets. This would reduce the errors introduced by the time frame, review quantity, and app store inconsistencies.

# 9. References


[1] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016.

[2] "Sample Size Calculator," *Sample Size Calculator - Confidence Level, Confidence Interval, Sample Size, Population Size, Relevant Population*. [Online]. Available: https://www.surveysystem.com/sscalce.htm. [Accessed: 07-Feb-2020].

[3] W. B. Hansen and L. M. Scheier, "Specialized Smartphone Intervention Apps: Review of 2014 to 2018 NIH Funded Grants," *JMIR mHealth and uHealth*, vol. 7, no. 7, 2019.