

# **Technical Report – Assignment 3**

## **Zixuan Chen (zixuanc2@andrew.cmu.edu)**

### **0. Introduction**

In this assignment, we classify app reviews with machine learning approaches. The vision is to analyze user feedback from those app reviews and identify actionable information for the developers to enhance the quality of their apps. To achieve this, this work aims to train several different machine learning models to address a multi-classification problem, which is classifying each app review into one of four categories (bugs, features, rating, and user experience)[1].

About the developing environment, except the libraries showcased in the Lab, an additional nlp package calls “spaCy” is used for extracting word embedding. All the scripts are tested with Python 3.8.

To run the scripts, open “[Colab\\_A3.ipynb](#)” on a Google Colab runtime. Note that due to the limited computing resource on Colab, all models are trained with the default configuration. No model selection or parameter search is performed as they will take too much time on the Colab runtime. However, if you’d like to perform model selection, you could either uncomment the relevant code on Colab, or run the notebook for each training scenario individually in your local environment. Those notebooks are named “exp#\_XXX.ipynb”.

### **1. Approach**

#### **1.1. Dataset Manipulation**

The datasets at hand are individually self-contained, each consisting of the full data samples for a binary classification for a particular review category. For example, the “Bug” dataset consists of data labeled with either “bug” or “not\_bug”. The same things for the “Feature”, “Rating” and “User Experience” datasets. The challenge is to merge these four datasets into one, in which each review has a label of one of the four categories.

First, all reviews from the four datasets are put together for deduplication. Data are grouped by all features, including review comments and other metadata. Then, there is an iterative procedure to make sure that each distinct feature has one and only one positive label to match. We are not removing duplicates based on counts because even though some reviews appear multiple times in different datasets, their labels may be [“Bug”, “Not\_feature”, “Not\_rating”]. There is only one positive label, therefore it is considered a “valid duplication” and it should be kept in the final dataset. During the iterative procedure, we are mainly removing two types of data: (1) data with multiple positive

labels (e.g. both a “Bug” and “Rating”) and (2) data with conflict labels (e.g both “Rating” and “Not\_rating”). These two types of data will clearly create confusion and bias in our dataset.

Secondly, all the negative samples are dropped from the dataset because they do not contribute to our multi-class classification problem. A sample with a negative label cannot be identified as any category, therefore it makes no sense to keep it in the dataset. Furthermore, most of the remaining negative samples after deduplication are “valid duplicates”, which means there exists a sample with the exact same features but has a positive label. Thus, I believe it is safe to assume there is not much information lost after removing negative samples.

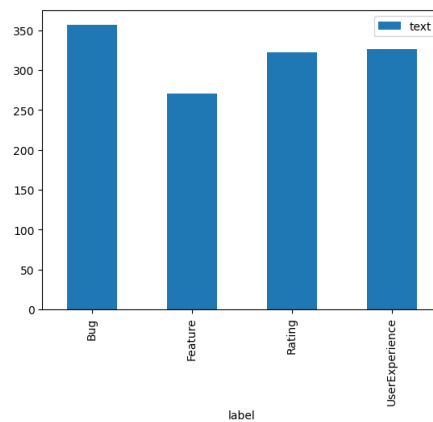


Figure 1. Label distribution of the final dataset

After data merge and manipulation, 141 duplicates and 1410 negative samples are removed. The final dataset has 1285 data points. Each has a distinct set of features and one label out of four categories. As shown in Figure 1, the final dataset has a balanced label distribution. See “utils/data\_loader.py” for detailed implementation.

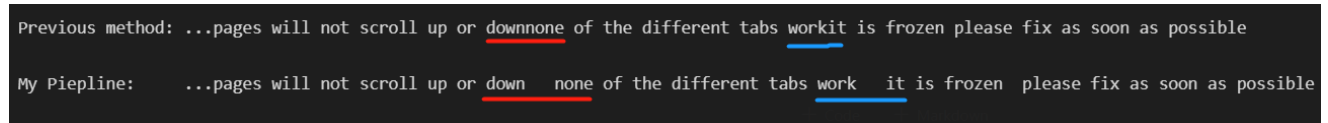
## 1.2. Text Processing

Textual features are extracted from “comments” of each app review. However, before feature extraction, we need to preprocess those text data. A single text preprocess pipeline was implemented for converting original comments into a simplified form which is easier for feature extraction. The pipeline contains 6 stages in total (implementation in “utils/text\_processor.py”):

- (1) Reverse contractions (e.g. asap becomes as soon as possible)[2]
- (2) Remove non-alphanumeric characters,
- (3) Convert to lowercase
- (4) Lemmatization[3]
- (5) Remove stop words

## (6) Remove extra spaces

This pipeline is an improvement from the original text processing procedure. In the original procedure, punctuations are replaced with an empty string, which would cause the words around them to be pieced together. However, our pipeline avoids this by replacing punctuations with a space, then removing all the extra spaces at the last stage.



```
Previous method: ...pages will not scroll up or downnone of the different tabs workit is frozen please fix as soon as possible
My Pipeline:    ...pages will not scroll up or down none of the different tabs work it is frozen please fix as soon as possible
```

Figure 2. Example of comparison between previous method and my text processing pipeline

After text processing, these strings need to be converted into vectors that can be processed by machine learning models. This project explored two methods for vectorizing text: TF-IDF and Word Embedding. TF-IDF uses frequency measures to capture word importance. It is simple and fast to apply. However, the corpus space could be huge even with text preprocessing, and each comment could use a very different set of words. Thus, the string vector is expected to be sparse with a lot of zeros in it and could make the models difficult to learn from. An alternative is the word embedding method, which captures both the syntactic and semantic meaning of a string [4][5]. With word embedding, we avoided the sparse matrix problem but were more exposed to the curse of high dimensionality.

### 1.3. Additional Metadata

Besides vectorized text, Maalej, et al suggested some additional numeric properties about the app reviews that could also help train the models. In their work, several metadata are collected and shown to be related to the review category [6]. For example, a high star rating value could indicate a user experience while a low star rating might be a sign of a bug report. In this assignment, 5 metadata are considered when training the models: star rating, length, tense (past or future), and sentiment score. I also experiment with using the positive and negative sentiment score instead of only the absolute sentiment score. However, the models trained with absolute sentiment score always outperform the models train with two sentiment scores. This also matches the results from Maalej, et al's work. Thus, in all the experiment settings, I am only using the absolute sentiment score.

### 1.4. Model Selection

Tree-based models support multi-class classification tasks natively. I selected Random Forest and XGBoost. Both are ensemble methods and complex extensions to tree-based models. The former utilized the concept of "majority votes" and statistically produce better predictions than a single fine-tuned decision tree. On the other hand, XGBoost considers the gradient of the results after each

iteration and greedily improves itself. In some settings, Decision Tree and AdaBoost are also trained and considered as baselines.

I also explored **Linear SVM** because, unlike tree-based models, it only supports binary classification. After all, It is possible to perform multi-classification using linear svm via the one-vs-rest strategy, and it is very easy to do so with scikit-learn [7][8]. It is worth mentioning that, unlike tree-based models, svm is sensitive to feature scaling. Therefore, standardization is performed for all numerical features.

## 2. Experiment Results

Settings are designed as combinations of 3 text representations (bag of words, bigram and NLP), 2 text cleaning configurations (remove or do not keep stopwords), and use of metadata (use or not use). Each setting is trained with 3 to 5 models. For simplicity, I am only reporting the result of the best model for each setting.

Setting	Model	F1 Bug	F1 Feature	F1 Rating	F1 User Experience	Accuracy
BOW-stopwords	XGBoost	0.661	0.5135	0.6609	0.5316	0.6062
BOW+Bigram-stopwords	Random Forest	0.6875	0.4545	0.6667	0.6437	0.6321
NLP-stopwords	Random Forest	0.6316	0.3492	0.7059	0.6136	0.601
BOW-stopwords+meta	XGBoost	0.7213	0.5195	0.6526	0.6957	0.658
BOW+Bigram-stopwords+meta	Random Forest	<b>0.7313</b>	0.5075	0.7033	0.7447	0.6891
NLP-stopwords+meta	XGBoost	0.6774	0.3889	0.7083	0.7234	0.6425
BOW+stopwords+meta	LinearSVM	0.7227	0.6234	0.6337	0.6966	0.6736
BOW+Bigram+stopwords+meta	LinearSVM	0.7213	<b>0.641</b>	0.6939	<b>0.75</b>	<b>0.7047</b>
NLP+stopwords+meta	Random Forest	0.6308	0.3279	<b>0.7327</b>	0.6596	0.6166

### 3. Conclusion

The setting that produced the best performance is Bag of Words and Bigram, plus metadata (rating, tense, length, sentiment score), and the stop words are kept during text cleaning. Among the models trained, the Linear SVM has the best overall accuracy and F1 scores. The Random Forest and XGBoost models have accuracy of 0.68 and 0.67 respectively. Experiments also show that parameter search did not improve the performance. The major factor that has contributed to its performance is keeping stop words. This might seem counter-intuitive, especially for the Bag of Words methods, because stopwords are supposed to appear in every document, thus providing little information to distinguish them. However, as Maalej, et al mentioned, some common stopwords are actually strong indicators of a review category (e.g. “should” may signify a feature request) [6]. Therefore, stopwords could be informative for this task, and results show that keeping them indeed improved the performance.

The worst performance is produced by NLP and without stopwords. Linear SVM, Random Forest, and XGBoost models have accuracy of 0.51, 0.6 and 0.58 respectively. It makes sense that Linear SVM works poorly on NLP methods, because the word embedding vectors are much more complex than bag of words. It could be too complex for linear model to handle. Another interesting finding is that the training score of settings using the NLP approach are usually much higher than others. This is saying that all models are terribly overfitting word embeddings. Finally, it didn't consider any of the metadata. Some information like tense are clearly helpful, but it cannot be captured from the text representation, since it is removed during lemmatization.

The results are greatly influenced by settings. As mentioned, the settings are designed as combinations of 3 methods of text representation, whether or not to include metadata and whether or not to keep stopwords. Adding metadata had a significant improvement for all three types of text representation. Keeping stopwords also improved the performance of bag of words and bigram, but it worked poorly for word embedding. Performing parameter search did not show significant improvement in the performance, the results are almost identical to training with the default parameters. Furthermore, my modification for the text cleaning pipeline also contribute greatly to the performance, because it disentangled many words which could be informative.

Finally, if given more time, I would do two things differently:

- (1) Experiment with smaller feature size by either performing feature selection or reducing the text vector length
- (2) Try a different approach: train a regression model for each category, each outputs a confidence score between 0 and 1. When predicting, output the label which has the highest confidence score.

## Bibliography

1. [https://en.wikipedia.org/wiki/Multiclass\\_classification](https://en.wikipedia.org/wiki/Multiclass_classification)
2. <https://github.com/kootenpv/contractions>
3. <https://en.wikipedia.org/wiki/Lemmatisation>
4. [https://en.wikipedia.org/wiki/Word\\_embedding](https://en.wikipedia.org/wiki/Word_embedding)
5. <https://lvngd.com/blog/spacy-word-vectors-as-features-in-scikit-learn/>
6. Maalej, W., Kurtanović, Z., Nabil, H. *et al.* On the automatic classification of app reviews. *Requirements Eng* **21**, 311–331 (2016). <https://doi.org/10.1007/s00766-016-0251-9>
7. <https://www.baeldung.com/cs/svm-multiclass-classification#:~:text=Multiclass%20Classification%20Using%20SVM&text=The%20idea%20is%20to%20map,into%20multiple%20binary%20classification%20problems.>
8. <https://scikit-learn.org/stable/modules/svm.html>