CPSC 5310 Final Project: User Review Analysis of Google Play Store Apps

Mir Tafseer Nayeem
Graduate Student, University of Lethbridge
mir.nayeem@uleth.edu

November 27, 2015

1. Introduction

User review analysis is becoming a popular area of research. It generally focused on identifying opinion polarity [4]. App stores like Google Play [1] allow users to submit feedback for downloaded apps in form of star ratings and text comments. As of February, 2015, Google Play Store holds 1.4 million apps Android apps [2] both free and paid apps. Thousands of user's feedbacks are given every day on these apps. With this huge amount of apps the users have a wide selection range to buy or install. User reviews serves as a valuable source of information for evaluating a mobile app, for both new users and developers. But after few months of a new app is launched in the market, there could be over ten thousand textual comments from users. It is very challenging for a potential user to read all of the comments one by one. For example, very popular apps such as Facebook get more than 4000 reviews per day. The review quality also varies user to user. They may contain helpful advice, innovative ideas about features as well as insulting comments. A textual review generally holds a mixed sentiment .So it is very difficult to filter out the positive and negative feedback or retrieve the feedback for specific features [5]. This can be made easier with sentiment analysis more specifically user review analysis. I'll focus on 2 possible sentiment classifications of user reviews: positive and negative.

The project is divided into four parts **1)** Collecting reviews and making the labeled corpus **2)** Preprocessing and Feature Extraction **3)** Classification Phase with selected classifiers **4)** Evaluate the Experiment results. For implementation purposes I have used NLTK (Natural Language Toolkit) [3] it is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 100 corpora and lexical resources. Lastly, conclusion and future works are discussed. All the phases are summarized in figure: 01.

2. Web Crawling and Dataset Creation

While working on this project of review classification using machine learning, I realized that to train the algorithms I needed an already classified dataset for training and testing my classifier. After searching for a while on internet I didn't get any app review dataset to work on my project. Moreover, App's reviews and other user's reviews are two different things. App's reviews are normally short. On the other hand app reviews contains many app specific words like freezes, crashes, crashed, ads, popup, candy crush etc. While testing the reviews against the other review classifiers "i love this app" was resulting into negative sentiment and "this app freezes" was returning positive.

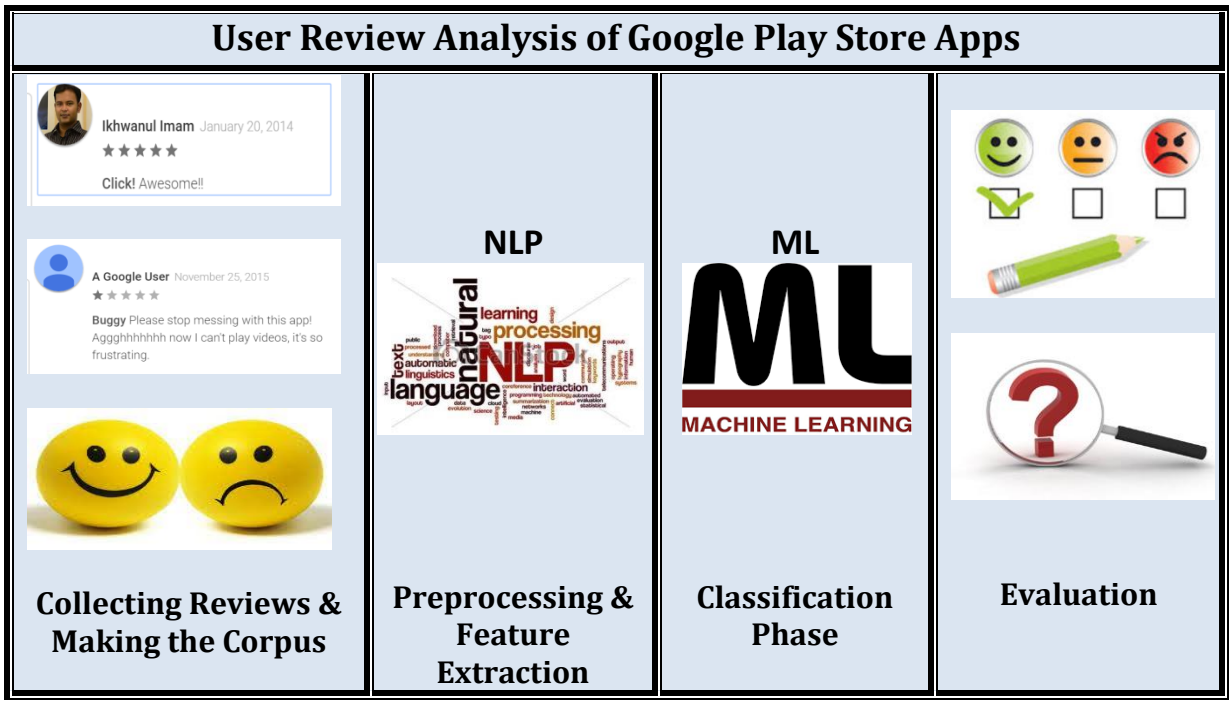

Figure 01: Review Analysis of App Store project Phases

I thought of using the reviews from the crawler DB but again there were two problems:

1. Reviews were not annotated (positive/negative)
2. Many apps data was country specific (different languages)

Finally, I decided to write a simple script to crawl the reviews from Google Play Store [1]. I took the top apps id of 3000 Apps from the crawler database on the server with their play store rankings between 1-10. I maintained the default sorting order i.e. most helpful reviews (The reviews and ratings marked helpful by others).

1. Collecting App IDs based on rankings, and
2. Downloading reviews for apps based on App IDs.

Due to Google's anti-spam, anti-DDOS policies, there are certain limitations on scraping the data:

1. Not all apps show up when querying for App IDs. For example, querying for "angry birds" does not return results for the popular "Angry Birds" game series.
2. Only a maximum of 4500 reviews can be downloaded for any app.

Common metadata that can be collected with the reviews include the star rating, the length, and the submission time. The star rating is a numeric value between 1 and 5 given by the user. Studies highlighted the importance of the reviews for the app success [6]. Apps with better reviews with better star ratings get a better ranking in the store and with it a better visibility and higher sales and download numbers [7]. So, for labeling purposes we assume that all 5 star rating reviews are labeled positive as the

reviews are small and 1 and 2 star ratings are labeled as negative. All the reviews which length is greater than 500 are ignored. However, for better training data we also ignored 3 star and 4 star ratings as they usually contain both of the labels.

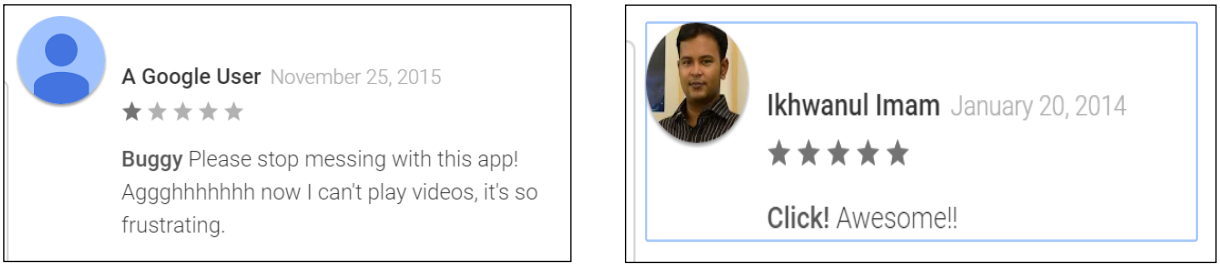

Figure 02: Example of (1 star review and 5 star reviews) collected from google play store.

3. Preprocessing and Feature Extraction

I preprocessed the text data separated from each other by a single space to remove the noise for better feature extraction. Afterwards, we extract the features from the reviews by applying a collocation finding algorithm and aggregating features by their meaning.

a) Preprocessing

1. **Removing the numbers:** Numbers don't carry any sentimental information. So numbers has been removed at the first place.
2. **Removing the Punctuations:** Punctuation marks (. , ; ( ) ? : etc) are removed because they don't contain any useful information.
3. **Not Lowered Down:** All the words are not lowered down because, all uppercase words like AWESOME, BEST, LOVE IT, RIP OFF are highly sentimental words. It has been seen that lowering down all the words will decrease the accuracy by 2%. However, without lowering the whole word I only lowered down the first character if not all UP_CASE to reduce the overall feature size because a feature word can be present at the beginning as well as the other places for instance "Loved it" and "I just loved it"; here the feature word is "love". Classifier will consider "Love" and "love" as two different features if it was not lowered down.
4. **No Spell Correction:** User reviews have typos and as well as contractions (U, coz, awsm , gr8 , Plz etc) . Contractions are not typos, they are sentimentally rich tokens. It has been seen that Spell checker algorithms converts some of the contractions into dictionary words and eventually reduces the performance by 1.3%.
5. **Things Not done:** combination of punctuation marks represents the emoticons (like ":)" means ☺ and ":(" means ☹) which can be a good feature for classifications of reviews. Moreover, repetitions (like soooooooo happyyyyyyyy.., greattttt…. , looooved it , plzzzzz) can also be a good feature. For the sake of simplicity I ignored these features to be included in the feature vector.

b) Feature Extraction

Review analysis does not require further preprocessing; the feature extraction requires two additional steps stopword removal and Lemmatization for better feature selections. Intelligent feature selection will eventually increase the accuracy, we will see it later.

1. **Stopword removal:** Stopwords [8] are words that are generally considered neutral words, the kind of words that are devoid of sentiment on the other hand they are the common words present in almost every sentences; so including them would greatly increase the size of the feature without improving precision or recall. NLTK [3] comes with a stopwords corpus that includes a list of 128 english stopwords (I , we , it , have , a , the , not , have , should , very , down , off etc). I removed the stopwords only for unigram features (single word as a feature).
2. **Lemmatization:** We use the Wordnet [9] lemmatizer from NLTK for grouping the different inflected forms of words with the same part of speech tag which are syntactically different but semantically equal. This step reduces the number of features. With this process, for example, the terms describing the verbs “fixing”, “fixed”, and “fixes” are grouped into the term "fix".
3. **Stemming is not used:** Stemming reduces each term to its basic form by removing its postfix. Stemming does not consider the linguistic context of the term and do not uses dictionaries. Stemmers just operate on single words and therefore cannot distinguish between words which have different meanings depending on part of speech. For instance, lemmatization recognizes “good” as the lemma of “better” and stemmer will reduce “goods” and “good” to the same term. Both lemmatization and stemming can help the classifier to unify keywords with same meaning but different language forms, which will increase word feature count. It has been observed including both of them together will decrease the accuracy by 3.6%. So I have used only lemmatization.
4. **Unigram Features:** Taking every single word as features. I excluded the stopwords because all words are independent of each other in case of unigram feature. It have been seen that excluding the stopwords will increase the accuracy of unigram features by 2.73%.
5. **Bigram Features:** People normally use positives words in negative reviews, but the word is preceded by “not” (or some other negative word), such as “not great”. And since the classifier uses the bag of words model, which assumes every word is independent, it cannot learn that “not great” is a negative. Moreover the sequences of words that co-occur more often than by chance, called collocations. For instance, “great app” as a phrase is probably more influential as a feature than the separate terms “great” and “app. I have used the collocation finding algorithm provided by the NLTK toolkit for extracting bigram features from the user reviews. In addition to that, other words such as “rip off”, “slow down”, “very good” those are highly sentimental bigrams but “not”, “off”, “down” and “very” etc are included in the stopwords list. Including the stopwords for bigram finding will decrease the accuracy by 3%. So I decided not to exclude the stopwords in case of bigram collection finding.

4. Classification of User Reviews

Document classification is a popular technique in information science, where a document is assigned to a certain class. A popular example is the email classification as “spam" or “no spam”. In our case, a document is a single review. The Binary classification labeled with either “pos” or “neg” for a single user review. The basic form of document classification is called bag of words (BOW).

a) Bag of Words

The bag of words model is the simplest method; it constructs a word presence feature set from all the words of an instance. This method doesn't care about the order of the words, or how many times a word occurs, all that matters is whether the word is present in a list of words. The NLTK classifiers expect dict style feature sets, so we must therefore transform our text into a dict. The classifier creates a dictionary of all terms in the corpus of all reviews and calculates whether the term is present in the review of a certain type and how often. Supervised machine learning algorithms can then be trained with a set of reviews (training set) to learn the review type based on the terms existence and frequency.

b) Classifiers Used in the Project

Supervised machine learning algorithms can be used to classify the reviews. The idea is to first calculate a vector of properties (called features) for each review. Then, in a training phase, the classifier calculates the probability for each property to be observed in the reviews of a certain type. Finally, in the test phase, the classifier uses the previous observations to decide whether this review is “pos” or “neg”. This is called a binary classification.

Naive Bayes [11] is a very popular algorithm for binary classifiers [11], which is based on applying Bayes’ theorem with strong independence assumptions between the features. It is simple, efficient, and does not require a large training set like most other classifiers.

Decision Tree learning is another popular classification algorithm [11], which assumes that all features have finite discrete domains and that there is a single target feature representing the classification (i.e., the tree leaves) [11].

Finally, the multinomial **logistic regression** (also known as maximum entropy or MaxEnt) [11] is a popular algorithm for multiclass classification. Instead of a statistical independence of the features, MaxEnt assumes a linear combination of the features and that some review-specific parameters can be used to determine the probability of each particular review type.

c) Training Set vs Test Set

The app reviews corpus has 2,861 positive files and 2,764 negative files. We’ll use 3/4 of them as the training set, and the rest as the test set. This gives us 4218 training instances and 1407 test instances. The classifier training method expects to be given a list of tokens in the form of [(feats, label)] where feats is a feature dictionary and label is the classification label. In our case, feats will be of the form {word: True} and label will be one of ‘pos’ or ‘neg’. For accuracy evaluation, we can use nltk.classify.util.accuracy with the test set as the gold standard. Moreover, we experimented with

other ratios and with 10-Fold Cross Validation method. The results reported in this project are based on one run with the 75:25 split ratio.

d) Evaluation Measures

Accuracy is not the only metric for evaluating the effectiveness of a classifier. Two other useful metrics are **precision** and **recall**. These two metrics can provide much greater insight into the performance characteristics of a binary classifier.

1) Classifier Precision

Precision measures the exactness of a classifier. A higher precision means less false positives, while a lower precision means more false positives.

$$Precision = \frac{TP}{TP + FP}$$

2) Classifier Recall

Recall measures the completeness, or sensitivity, of a classifier. Higher recall means less false negatives, while lower recall means more false negatives.

$$Recall = \frac{TP}{TP + FN}$$

5. Experimental Results and Discussions

The overall goal is to study how accurately the classification techniques described from Section 4 can predict the user review's class as "pos" or "neg". This includes answering the following questions:

- **Feature Selection:** Which feature or combination of features works well in classification of app reviews?
- **Classification algorithms:** Which classifier algorithm works better (N. Bayes vs. D. Tree, vs. Logistic Regression)? Will combination of these algorithms with voting outperform the individual?
- **Performance & data:** How much time and training data are needed for an accurate classification and is there a difference when number of features varies?

To answer these research questions we conducted a series of experiments the results are summarized in table I.

Feature Selection	Classification Models														
	Naïve Bayes					Logistic Regression					Decision Tree				
	A	P		R		A	P		R		A	P		R	
		PP	NP	PR	NR		PP	NP	PR	NR		PP	NP	PR	NR
Unigram	93.8	**90.2**	98.2	98.4	**89.0**	91.6	86.6	98.4	98.7	84.2	**87.2**	87.3	87.0	87.5	86.8
Bigram	96.3	**96.6**	96.1	96.2	**96.5**	96.1	95.8	96.4	96.6	95.6	86.3	86.2	86.9	87.3	86.4
Unigram + Bigram	**96.5**	96.1	96.9	97.0	95.9	96.0	94.7	97.6	97.7	94.3	87.4	88.0	86.8	87.1	87.6

Table I: Experiment results of different classification Models with different feature selections

By looking at the above numbers it is clearly understood that naïve bayes outperforms compare to other two models. Naïve Bayes and Logistic Regression are quite close to each other. Not only the accuracy, precision and recall of decision tree classifier is low but also it takes too much time to classify compare to other two methods.

As expected the combination of unigram and bigram will perform better. So our assumption was correct. Unigram model perform poorly compare to other two models because of independence of each words. Let's explain it in a good detail.

(- + = -) ("not" + "good") = bad ("-ve" word/feature)

(- - = +) ("not" + "bad") = good ("+ve" word/feature)

Positive precision is higher in Bigram model and lower in unigram model. Lower precision means more false positives. This can only be occur when someone use a positive word in a negative review like the previous example. Vice versa is also for the negative recall when negative word is used in a positive review. In case of unigram model, it cannot differentiate them .On the other hand bigrams can; that's why it has the higher PP and NR. Vice versa also true for the NP and PR .The conclusion could be that there are certain common words that are biased towards the pos label, but occur frequently enough in the neg feature sets to cause mis-classifications. To correct this behavior, I used most informative words as a feature.

a) Choosing High Informative Features

A high information feature is a word or group of words that is strongly biased towards a single classification label. The low information words are words that are common to all labels. Eliminating these words from the training data can actually improve accuracy, precision, and recall while also decreasing the size of the model, which results in less memory usage along with faster training and classification. The reason this works is that using only high information words reduces the noise and confusion of a classifier's internal model. If all the words/features are highly biased one way or the other, it's much easier for the classifier to make a correct guess.

A word that occurs primarily in positive movie reviews and rarely in negative reviews is high information. For example, the presence of the word "AWESOME" in an App review is a strong indicator that the review is positive. Let's observe the classifiers performance on different numbers of informative features in Table II.

High Ranked Features	Classification Models														
	Naïve Bayes					Logistic Regression					Decision Tree				
	A	P		R		A	P		R		A	P		R	
		PP	NP	PR	NR		PP	NP	PR	NR		PP	NP	PR	NR
10 Unigrams	**56.3**	53.8	100	100	11.1	60.9	56.6	99.3	99.8	20.7	**86.5**	86.4	86.6	87.2	85.8
100 Unigrams	92.8	88.4	98.5	98.7	86.6	**92.9**	88.6	98.5	98.7	86.8	87.7	88.6	86.7	87.0	88.4
1000 Unigrams	**95.8**	94.4	97.4	97.6	94.0	95.6	93.8	97.7	97.9	93.3	87.2	87.6	86.7	87.1	87.2
10000 Unigrams	95.6	93.7	97.8	98.0	93.1	95.1	92.6	98.1	98.3	91.8	87.4	87.5	87.2	87.7	87.1
15000 Unigrams	95.3	93.2	97.8	98.0	92.6	94.8	91.8	98.4	98.6	90.8	87.1	87.2	87.0	87.5	86.6
200 Bigrams	96.3	96.6	96.1	96.2	96.5	96.1	95.8	96.4	96.6	95.6	86.3	86.2	86.9	87.3	86.4

Table II: High informative features and their performance in different classifiers

Important thing to here is that decision tree performs well with small number of informative features. Logistic regression performs well with 100 unigrams. Navie bayes outperforms others with 1000 unigrams. After that including the features decreases the accuracy. More features create more confusion to classify as we have seen previously.

Let's observe the some top informative features (unigram + bigram) of the classifiers used in the evaluation in Table III. Somewhat surprisingly, the top words are different for all the classifiers. This discrepancy is due to how each classifier calculates the significance of each feature. However, some of the features are common among the classifiers means they are informative among the high informative ones. It's actually beneficial to have these different methods as they can be combined to improve accuracy. Next I will use the classifiers combined with voting.

Features	**Classification Models**		
	Naïve Bayes	**Logistic Regression**	**Decision Tree**
Unigram	waste = True neg : 78.8	AWESOME = True pos : 63.9	**amazing = True pos : 76.**3
	excellent = True pos : 77.9	**crap = True neg : 58.8**	terrible = True neg : 59.2
	amazing = True pos : 76.3	BEST = True pos : 53.3	**crap = True neg : 58.8**
Bigram	(u'highly', u'recommended') = True pos : 60.2	(u'very', u'useful') = True pos = 67.0	**(u'LOVE', u'IT') = True pos : 63.9**
	(u'not', u'working') = True neg : 56.5	**(u'LOVE', u'IT') = True pos : 63.9**	(u'Not', u'happy') = True neg : 57.6

Table III: High Informative features for the classification algorithms

b) Combining classifiers with voting

One way to improve classification performance is to combine classifiers. The simplest way to combine multiple classifiers is to use voting, and choose whichever label gets the most votes. The individual classifiers should use different algorithms; the idea is that multiple algorithms are better than one. I am going to use a NaiveBayesClassifier class, a DecisionTreeClassifier class, and a MaxentClassifier class, all trained on the highest information words of the app reviews corpus. These were all trained in the previous sections, so I will combine these three classifiers with voting.

High Ranked Features	**Max Vote Classifier**				
	A	**P**		**R**	
		PP	**NP**	**PR**	**NR**
10 Unigrams	60.9	56.6	99.3	99.8	20.6
100 Unigrams	92.8	88.5	98.5	98.7	86.8
1000 Unigrams	95.8	94.4	97.4	97.6	94.0
10000 Unigrams	95.7	93.8	97.8	98.0	93.3
15000 Unigrams	95.5	93.3	97.8	98.0	92.9
200 Bigrams	**96.4**	96.9	95.9	96.0	96.8

Feature Selection	**Max Vote Classifier**				
	A	**P**		**R**	
		PP	**NP**	**PR**	**NR**
Unigram	93.8	90.4	98.1	98.3	89.3
Bigram	96.5	96.9	96.0	96.1	96.8
Unigram + Bigram	**96.7**	96.4	96.9	97.0	96.2

Table IV: Experiment results of max vote classification Model with different feature selections

Max vote classifier outperforms all the previous classifiers in terms of accuracy precision and recall in different combination of features.

6. Conclusion and Future works:

Readers might think that why the classification accuracy is significantly higher. Remember I assumed all 5 star reviews are likely to be positive and all 1 or 2 star reviews are negative. Using this assumption I have labeled the corpus. This is somewhat the justification of the assumption. In future, I will try to device a mechanism to annotate the 3 or 4 star reviews in the corpus and evaluate the performance. Moreover, Interactive systems can be developed to summarize and visualize these reviews which will be beneficial for both the parties. (a) Without reading every comment end-users can use these summaries to choose the apps with the best user experience according to specific features (b) App developers can use these summaries to improve the quality, re-implement missing features, fixing bugs etc.

REFERENCES

[1] https://play.google.com/store/apps

[2] http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[3] http://www.nltk.org/

[4] Hu, M., & Liu, B. (2004a). Mining and summarizing customer reviews. In Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining (pp. 168–177). Seattle, WA.

[5] Hu, M., & Liu, B. (2004b). Mining opinion features in customer reviews. In Proceedings of the nineteenth national conference on artificial intelligence, sixteenth conference on innovative applications of artificial intelligence AAAI 2004 (pp. 755–760). San Jose.

[6] H. Li, L. Zhang, L. Zhang, and J. Shen. A user satisfaction analysis approach for software evolution. In Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on, volume 2, pages 1093–1097. IEEE, 2010.

[7] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. App store analysis: Mining app stores for relationships between customer, business and technical characteristics. Reseach Note RN/14/10, UCL Department of Computer Science, 2014.

[8] https://en.wikipedia.org/wiki/Stop_words

[9] G. A. Miller. WordNet: a lexical database for English. Communications of the ACM, 38(11):39–41, 1995.

[10] S. Bird, E. Klein, and E. Loper. Natural language processing with Python. O'reilly, 2009.

[11]Intelligence, Artificial. "A modern approach." Russell and Norvig (2003).