Google Play Store Review: Mining Application Store to Interpret User Experience, relationship between Business and Technical Characteristics through Sentimental Analysis

Eshaan Kaul[1], Aditya Kaustubhan[2], Yudhveer Singh[3]

[1]*(Department of Computer Engineering, Pune Institute of Computer Technology, Pune, India)*

Abstract*—Sentiment analysis, also called as option mining, refers to the use of natural language processing, analysing texts and performing computational linguistics to identify and extract subjective information in source materials. Sentiment analysis is a popular technique for summarising and analysing consumers' textual reviews about products and services.Most sentiment prediction systems work by just looking at words in isolation, giving positive points for positive words and negative points for negative words and then summing up these points. But in that way, the order of words is ignored and important information is lost. In contrast, our learning model actually builds up representation of whole sentences based on sentence structure by considering a string of words as a feature (N-gram applicationroach). Here we have used Tri-gram (3-gram) wherein 3 words together are combined to get sentiment of sentence and does not get fooled as easily as compared to previous models. For example, our model learned that 'enjoy' and 'nice' are positive but the following sentence is still negative overall : "The application UI isn't very nice and i didn't enjoy using it much". Beyond just positive and negative reviews, we have also identified reviews which have bugs in them and given it a separate class called bug-related reviews so that applicationlication developers can view those first and try to fix the bugs in their applicationlication.

***Keywords*—Sentimental Analysis, Natural Language Processing (NLP), Text Classification, Android applicationlications**.

I. INTRODUCTION

Application stores provide a rich source of information about applications concerning their customer business and technically focused attributes. Customer information is available concerning the ratings accorded to applications by the users who downloaded them. This provides both qualitative and quantitative data about the customer perception of the applications. Business information is available, giving the number (or rank) of downloads and also price of applications. Technical information is available in the descriptions of applications, but it is in free text format, so data mining is necessary to extract the technical details.

Of course, this information may not be complete or fully reliable: customers may, for various reasons, leave reviews that do not reflect their true opinion. Either intentionally or unintentionally, developers may not be completely truthful about the technical claims made. It is not unreasonable to hope that broad observations about whole classes of applications may still prove to be robust; the large number of applications on which such observations are based tends to support robustness.

Software engineering researchers been able to access publicly available data that links all of these important attributes:

• The customers' opinions of software, in the form of the reviews they leave;

• The popularity of software, in the form of its rank and/or number of downloads.

Correlation analysis allows us to address fundamental questions for any application store, such as:

1) Do applications that cost the customer more tend to get a lower rating?

2) Do free applications get higher rating? OR

3) Are high rated applications necessarily popular?

4) Do the extracted features enjoy any of the above correlations?

With the ever-increasing volume of text data from Internet, databases, and archives, text categorization or classification poses unique challenges due to the very high dimensionality of text data, sparsity, multi-class labels and unbalanced classes. Many classification approaches have been developed for categorizing text documents, such as random forests. Due to its algorithmic simplicity and prominent classification performance for high dimensional data, Random Forest has become one the most commonly used methods for text categorization.

II. METHOD

In this study, we extracted two thousand five hundred textual reviews of different applications from Google Play (Android application Store) to be classified into as Positive or

Negative reviews. We extracted these from Heedzy tool. Using the implementation of a random forest classifier, which uses an ensemble of decision trees to classify text based on an averaged value of multiple decision trees or forest of decision trees. This algorithm is particularly designed for analyzing very high dimensional data with multiple classes whose well-known representative data is text corpus. We have taken the help of the python sci-kit library for this purpose.

Sklearn is used for summarizing reviews and pulling out features. We have set the max features to 5000 to limit the size of algorithm expansion. The features of detecting bug related reviews to add on top of the Positive/Negative reviews to help developers identify and make the changes accordingly thus helping in establishing a relationship between Business and its Technical Characteristics.

A. Data Extraction

The data has been extracted using the tool Heedzy which is a free open source tool to extract reviews from the Google Play Store, and new reviews can keep getting appended to the csv file using that online tool.

B. Data Pre-processing

- Reviews are stored in CSV format and are read by python code using the Pandas library which provides functions for easy reading and writing to files.
- BeautifulSoup is another library used for data pre-processing that removes any HTML tags from reviews so we have clean text data only. NLTK or natural language toolkit is used for data pre-processing as well to remove stop words such as 'a', 'an', 'is',etc which doesn't carry much meaning towards sentiment and this increases speed of the algorithm.
- We have also removed all punctuations from the reviews to make it faster and more efficient.

C. GUI and Front-End Processing

We have built a web based dashboard that displays the count of positive and negative reviews as well as bug related reviews using bar charts animated with the help of ChartIST javascript.

Business analyst also has option of viewing all the reviews and checking which reviews had bugs and what they were on the UI. The AJAX function call is used to read CSV output file and check for changes and display that on graph.

We have built the entire dashboard on HTML and bootstrap which is a framework for HTML and CSS.

III. PROPOSED SYSTEM

So far we were finding the sentiment analysis using N-Gram. The only issue with the N-gram model was that to pre-determine the hyperparameters. Hyperparameters are basically the number of decision trees, max number of features, etc. So, thus far we were hard coding these parameters. Through our analysis, we understood the behavior of the system. The accuracy ratio was scaling up/down approximately logarithmically until a certain threshold value. Then it became stagnant for the remainder of the time. This brings us to two conditions namely :

1) Less Random : This states that the number of decision trees extracted from the sub-matrix are less in number and the proper sentiment of the sentence cannot be derived. It might give an invalid result.

2) More Random : This gives us the result with proper accuracy. But the only drawback to this is that the complexity of of the program will increase drastically. More space and time will be needed (for the system).

So, to overcome the first problem, we can add more and more number of training data to the system manually. And to overcome the second problem, we can use the following approach : Instead of executing all the decision trees we can setup a threshold level. After this level the accuracy ratio will always remain constant i.e. it won't have any affect on the system (No matter how many decision trees you execute).We have also divided the untrained sentence into number of words. First we start from 1 word (of a sentence, i.e. 1-Gram). We've have set a threshold of 'x' for that 1 word. Again we have to take into consideration that the trained data set can generate equal to or more than 'x' decision trees(so as to not make it Less Random). We record the classification as '0' or '1'(say A1). On the basis of all the 'x' decision trees. Then we go on to find the sentiments of the combination of two words using 'x/2' decision trees. We store the result again (say A2).We will then compare the two results. If both of them give the same(or almost) results, then we can stop the execution. Or else, we would have to go forward (A3,A4,..).

We will continue with the iteration till the entire sentence(after removing stop words,etc) becomes an individual token for classification. And the result(say An) will be our desired output by flooring or ceiling it. If the result is 0.5, then our either our training data set is not large enough or the user has put in a neutral comment. But this will only happen in the rarest or rare cases. We will stop the iterations at a step if our value is in a complete discrete form i.e. '0' or '1' only. Note that in our first step we will select those decision trees only that have a match with the tokens in the untrained data set and the features in the matrix using an efficient string matching algorithm (preferably KMP).Consider the following example of the untrained data : This application is not bad and useful. After removing the stop words : application not bad useful. So, our tokens will be formed of application, bad and useless in the first iteration. So from our feature extraction consider we get the classifier of the various tokens as :

1) application : 0

2) not : 1

3) bad : 1

4) useful : 0

Note that we have taken positive sentiment as 0 and negative sentiment as 1. Our result will be 0.50 (A1) with the help of 'x' decision trees. Thus it is pretty unclear what the result is going to be.In our next iteration, we will be going through two words

at a time using 'x/2' decision trees (because 'x' trees won't be required because of too much overhead). Possible combinations :

1) application not : 1

2) not bad : 0

3) bad useful : 0

At this step our value of A2 will be 0.33, thus leading toward a positive review. But still there is a huge difference b/w A1 and A2.

So we go towards the next iteration i.e A3(using 'x/4' decision trees). We get two combinations :

1) application not bad : 0

2) not bad useful : 0

So A3 gives a value of '0'. Thus we will end with our computation. And classify the untrained data as a positive review. After this step we will use this result for training future untrained data as well.

IV. Architecture Diagram

In this 3-tiered architecture (FIG 4.1), we use MongoDB as our database to store the CSV file. This file to be labelled is read by Random Forest module along with training data CSV file. Output is saved in a CSV file which is read by AJAX function call and results are displayed on a graph using ChartIST.

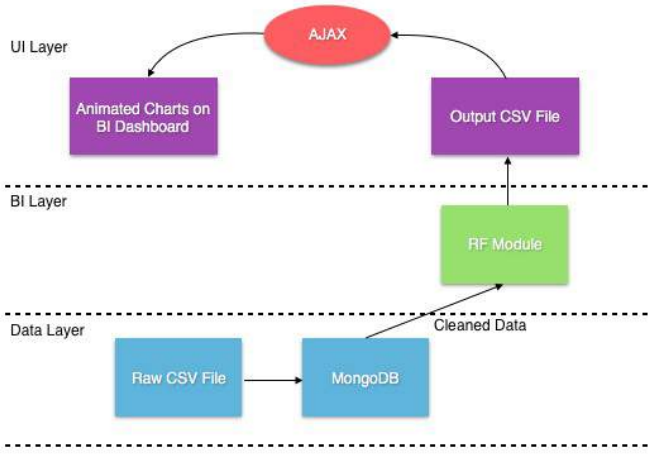

FIG: 4.1

V. Results

After performing tests on our data set using Random Forest Algorithm, we found it to have a classification accuracy of 94.75% . (Refer FIG 5.1)

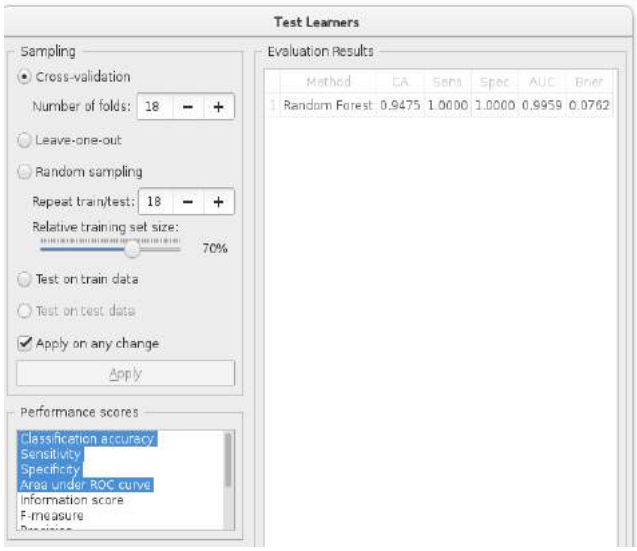

FIG: 5.1

We can also see from Figure 5.2 the scatter plot graph that all bug related reviews have come under negative sentiment reviews which shows a deep analysis into sentiment of user.

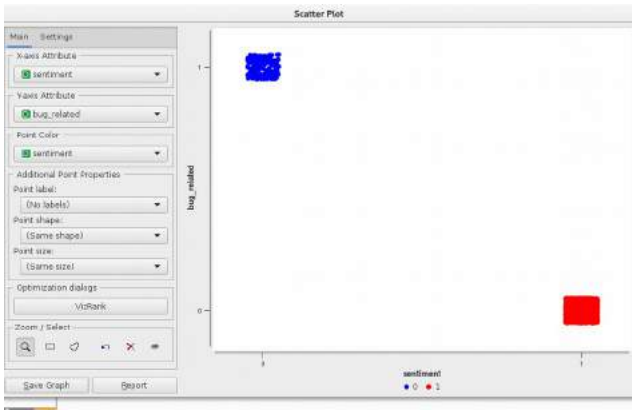

FIG: 5.2

VI. Conclusion and Future Scope

Application stores provide a software development space and market place that are fundamentally different from those to which we have become accustomed for traditional software development: the granularity is finer and there is a far greater source of information available for research and analysis. Information is available on price, customer rating and, through the data mining approach presented in this paper, the features claimed by application developers.

We have introduced a method to extract, from application store descriptions, usable information about the features of applications that captures some of the technical aspects of the applications in the store. We evaluated our approach on both the free and the non-free applications in the Blackberry application Store.

The degree of correlation between rating, price and popularity is different for different application categories, as one might expect and as we report in detail in the paper. Our analysis indicates that there is a strong overall correlation between the ratings given to applications by their users and their popularity

In future, we also intend to investigate predictive models of customer evaluations, and the interplay between functional and non-functional properties of applications, and the data available in application stores.

VII. REFERENCES

[1] *Kincaid, Jason. "Android Market: 10 Billion applications Served So Far, And Another 1 Billion Each Month". Techcrunch. Dec, 2011. http://techcrunch.com/2011/12/06/android-market-10-billion-applications-served-so-far-and-another-1-billion- each-month/*

[2] *Baoxun Xu, Yunming Ye, Xiufeng Guo and Jiefeng Cheng, An Improved Random Forest Classifier for Text Categorisation, Journal of Computers, VOL. 7, December 2012.*

[3] *Anthony Finkelstein, Mark Harman, Yue Jia, William Martin, Federica Sarro and Yuanyuan Zhang, application Store Analysis: Mining application Stores, UCL Department of Computer Science, 5th September, 2014.*

[4] *Jiawen Liu, Mantosh Kumar Sarkar and Goutam Chakraborty, Oklahoma State University, Stillwater, OK, USA, Feature-based Sentiment Analysis on Android application Reviews Using SAS® Text Miner and SAS® Sentiment Analysis Studio, SAS Global Forum 2013.*