Predicting PlayStore App Rating (December 2015)

Naresh A., Naila B., and Ayush R. Aryal, *Graduate Student, Mississippi State University*

***Abstract*—Google Play Store is the most popular hub for android applications. With the increasing number of applications being uploaded in the store every day, users prefer to download apps based upon their previous 'Ratings'. The higher the ratings, it could be assumed that the application is more reliable and popular. Being in a competitive world of building android applications, it is the goal of all developers to maximize or increase the ratings of their applications. We have developed a system which would help developers to predict the rating of their newly uploaded applications based on their 'Attributes'. We have retrieved a public data set of existing play store applications till August 2015. Several attributes of these applications like applications size, name, category, description, price etc. have been used to build a training model that will be able to predict ratings for new applications. To build this training model we have used standard machine learning regression and classification methods. The learning performance of the model has been tested on the existing applications that already have their ratings. This model will be useful towards making continuous improvements of the new applications to boost their popularity.**

***Index Terms*—Google Play, PlayStore, Android Application, Ratings, Linear Regression, Classification, Text Vectorization, Residual sum of squares, variance score**

I. Introduction

THE use of mobile phones is rapidly increased in recent decades. Along with this increase in use, various operating systems and applications have been developed for mobile devices like Android, IOS, and Windows Phone OS etc. Android is one of the major operating systems used in mobile phones. It is well known for being open source and having a large variety of applications. Android serves a range of applications through its market place known as Google Play Store, previously Android Market. The applications of various categories like games, music, education, entertainment and others are available in the Play Store. These applications can be free or can cost some amount of money depending upon its developers. As play store is an open market place, users can publicly rate the applications by giving 'Stars' that range from one to five. These ratings are publicly available to any user downloading the application and are considered to be one of the major factors for measuring the success of the application. In recent times, user ratings have been majorly important for developers as well as users. For a developer, user ratings are the benchmark of how successful his application is in the real world. However, for a user it could be the main evaluation factor to decide whether to buy/download an application or not.

As user ratings are very vital factor for applications, being able to predict the ratings for newly created application would be beneficial to both developers and users. Our goal for this project was to use the publicly available information about the existing applications and predict the user ratings for new application using various machine learning algorithms. As Google does not publicly share the main database of the applications, dataset generated using web scraping tool was used. In this paper we will give an overview on various machine learning techniques that we have used and how they performed in the context of predicting the user ratings.

II. Dataset

As of November 2014, the Google play store has around 1,400,000 applications [1]. The playstore provides information about various features of applications like name, developer name, description, size, parental rating, user ratings etc. This information is publicly available on website but the whole database is not officially shared publicly. So, to prepare our dataset, we have used a publicly available project from github. This project [2] webscraped the playstore website and collected all the information about this website. Upon contacting the creator of the project we have been able to collect the crawled dataset. This dataset has information of nearly 1.2 million playstore applications. The whole data was in a single json file of size 6.1 gigabytes. We initially splitted the large json file into smaller chunks using file split commands in linux environment. The json files were later converted into csv files for ease of usage. In python development environment csv files can be efficiently handled and customized for available frameworks like 'Pandas'. These csv files were further used to generate the feature and label vectors to fit into the various machine learning models.

Fig 1 depicts a simple view of the json data we have that

This paper is submitted on December 9, 2015 as a part of course requirement of CSE: 4990/6990. The data used for the corresponding project has been acquired on request from the author of github.com repository GooglePlayAppsCrawler.

Naresh A is a current PhD student who is working as a Research Assistant in HPC2 lab of Mississippi State University, Starkville, MS 39762 (email: na542@msstate.edu).

Naila B. is a current PhD student who is working as a Research Assistant in HPC2 lab of Mississippi State University, Starkville, MS 39762 (email: nb921@msstate.edu).

Ayush R. Aryal, is a current Masters student who is working as a Research Assistant in HPC2 lab of Mississippi State University, Starkville, MS 39762 (email: ara305@msstate.edu).

was parsed for listing the features of the applications.

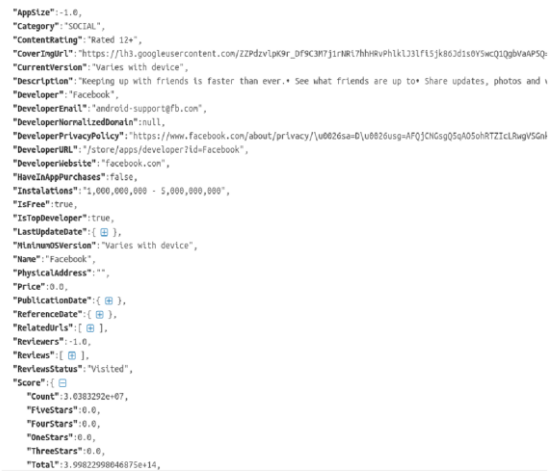


Fig:1 Attributes of PlayStore applications in json format.

III. Data Preprocessing

Data is the core of any learning algorithm. Before training a learning model with a specific set of data, it needs to be made sure the data will produce expected result from the model. If the data contains redundant, noisy and unusable information then we cannot expect a model to give us decent results. This is why most of the learning process goes through this stage of data preprocessing. Data preprocessing itself has different stages. Among them we have mainly focused on 'Data reduction' so that we can remove the unnecessary app attributes which do not affect the ratings of the apps and 'Data transformation' to extract information from both numerical and non-numerical data to finally construct the feature vectors..

A. Data Reduction

Several application attributes like the cover image url of the app, developers email address, physical address of the developer, url of the application etc. do not have enough significance to affect the ratings of the corresponding app. So through manual scrutinizing we have eliminated such features and shortlisted 13 features for constructing the feature set. These attributes are namely AppSize, Price, IsTopDeveloper, HaveInAppPurchase, IsFree, PublicationDate, LastUpdateDate, Installations, Category, Developer, Name, ContentRating and Description.

B. Data Transformation

Among our short listed features AppSize and Price already had numerical data as values so these did not need any kind of transformation. IsTopDeveloper, HaveInAppPurchase, IsFree,Several, these three had Boolean (True and False) values as measurement which were replaced by '0' and '1's. Attributes like PublicationDate, LastUpdateDate, and Installations needed some different kinds of transformation to be effective on the feature vectors. These date fields were modified such that number of months since that date to current date can be calculated and considered as feature values. The remaining textual attributes like Category, Developer, Name, ContentRating and Description needed to be vectorized based on word count.

Vectorization resulted in huge number of columns for some of the features which were further reduced using analytical techniques of dimensionality reduction.

IV. Text Vectorization

Text vectorization is a technique of generating numerical feature vectors corresponding to the terms in a text document [3]. With an assumption that any target or dependent variable is a function of document (a bunch of text) that is associated with the variable, text document is also considered as independent variable. Because machine learning algorithm does not process raw strings of text to generate an estimator, a text document needs to be vectorized.

Basically there are two methods of translating a document to its corresponding numerical vectors: 1) Term Frequency Vector and 2) Term Frequency-Inverse Document Frequency Vector

Term Frequency Vector is generated by counting occurrences of a particular term or word t in a document *d*, which is called term frequency, $tf(t,d)$. For example, if a term t occurs for a frequency f, then the vector value for the term is $f_i = tf(t,d)$.

In general, for all terms t_i that is selected to be a valid term that exist in a document d_j, term frequency vector is V_j, an array of all f_i.

$$V_j = [f_1, f_2, f_3 \dots f_n] \qquad (1)$$

Term frequency Vector approach emphasizes frequent terms however it diminishes the rare terms which are empirically more informative than the higher frequency terms [3]. In a classification problem [3], highly frequent terms what are present in the entire corpus (set of documents) of the documents would be less beneficial. This problem of term frequency is solved by term *tf-idf* weight that is used to generate term-frequency-inverse document frequency vectors.

A. Term-frequency-inverse Document Frequency Vector:

Let a document be defined as $D = \{d_1, d_2, d_3, \dots, d_n\}$, where *n* is the number of documents in a *C*. $Idf(t)$ of a term *t* is defined as:

$$Idf(t) = log \frac{|D|}{(1 + N_t)} \qquad (2)$$

where, N_t is the number of documents where term *t* appears, and $Idf(t)$ is non-zero, and $|D|$ is number of total documents in a corpus. A new function $tf\_idf(t)$ is defined as:

$$tf_{idf(t)} = tf(t,d).idf(t) \qquad (3)$$

A high value of (2) is obtained when a term has high *tf(t,d)* in a given document *d,* and a low document frequency of the term in the whole corpus [3].

B. *Steps for Vectorization*

Document vectorization consists of three specific techniques [4]: 1) Tokenization, 2) Counting, and 3) Normalization. Tokenization is process of segregating a document into different components called words or terms and assigning each token an integer identifier. Each word or term is called a token. To tokenize a given document, separators like white-spaces, punctuation marks could be referenced. It can be done with or without considering other words in the context of the word. Counting involves finding the occurrences of terms or words and assigning frequencies to them. Token Normalization is method by which each of the vectors corresponding to a document is expressed to have magnitude of unity. In general, if V is a vector space, normalized vector is obtained by dividing V by its magnitude $|V|$. There are normal forms: L2-norm, and L1-norm.

C. *Attributes that needed Vectorization*

Five attributes of an application that needed text vectorization are:

1. Name: It is name of an application. Name is a *multilingual* corpus—it consists of small number of words that could be in multiple languages.
2. Description: It is textual description of an application. It is also multilingual corpus that consists of multiple words of larger number of words.
3. Category: It represents a category that an application belongs to. There are 41 different categorical names and all belong to English terms making it a *monolingual* corpus.
4. Developer: Also a *multilingual* corpus like name
5. ContentRating: *monolingual* corpus like Category

D. *Methods used for vectorization:*

In our project, two types of vectorization methods are used. In first method, Sklearn's CountVectorizer with stop words in English, Chinese, German, Arabic, French, Japanese and Korean were used. In second method, CustomStemVectorizer with stop words in those languages was used. Latter method produces stem or base of the words in the document by employing Nltk's stemmer utility. Finding base of a word could be done in the context of the word's occurrence. However, in this project, only base of a word without caring its context is adopted. In both of the cases, token that represents very basic and relatively very less important words of English language and other languages and numerical values, punctuations and other symbols in {*, _} are ignored as stop words or characters by the vectorizers [5].

E. *Analysis of Sparsity of the Vectorized Matrices*

Sparsity of a vectorized matrix is ratio of total non-zero values to total values in the matrix. Sparsity decreases as number of terms having zero term frequency increases. High sparsity of a vectorized matrix is desired. However, normally very low sparsity is obtained when processing large documents.

Two different vectorization ways are implemented for the above attributes of the applications. For each of the following vectorizations, stops word of languages like English, Russian, Greek, Latin, Chinese, Japanese, Korean and Arabian are used to disregard syntactically redundant words in the languages. However, more than 30 languages appear across the documents describing description of the application. Numerical terms that appear in description and that represent year or price or number of installations, and so on are excluded consciously to reduce the size of the vectorized matrix.

1) *Stemmed Vectorization*

In stemmed vectorization, stem or base of the word are generated for every word that appears in a document. This reduces the size of feature vectors because two or more words having the same stem are excluded. In this case, stemmer utility of NLTK python library is used to determine stem of an English word. Stemmer utilities for other languages could have also been used.

2) *Un-stemmed Vectorization*

In un-stemmed vectorization, each word is counted as single token and no stemming of the word is performed. This approach significantly increases the size of vectorized matrix as each word represents a unique feature irrespective of their origin or base.

Both vectorization techniques were to process the text describing name, description and category of applications and the sparsity of the matrix generated is compared. Languages were grouped into two categories: In first category, pure English words only and in second category English words including words in all other languages. In both cases, stemmer for the English words was used, and stop words used were of English, Russian, Arabic, Japanese, Korean and Chinese. With sample size of 10K documents, following result of sparsity is obtained.

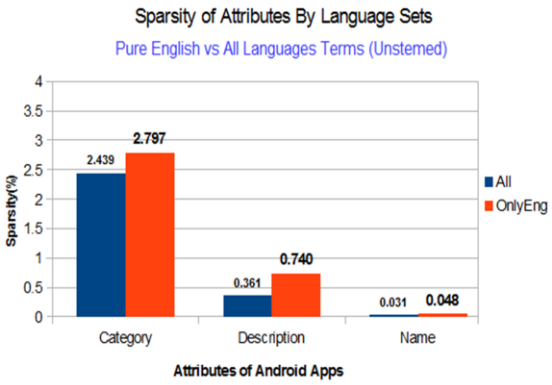


Fig:2 Sparsity of vectorized matrices for Category, Description, and Name documents considering words in all languages and pure words in the English only without stemming the words in the English only without stemming the words in English

As indicated in above figure-2, when taken pure English words without stemming, the sparsity is higher across all corpora Category, Description and Name.

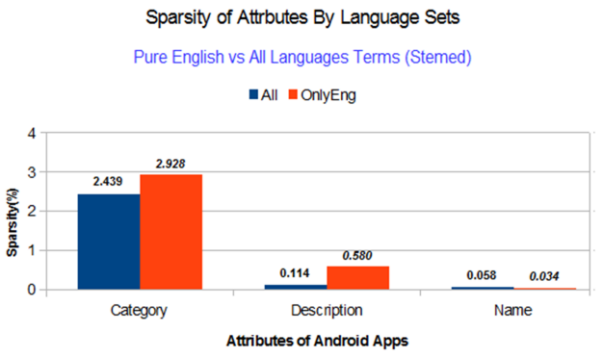
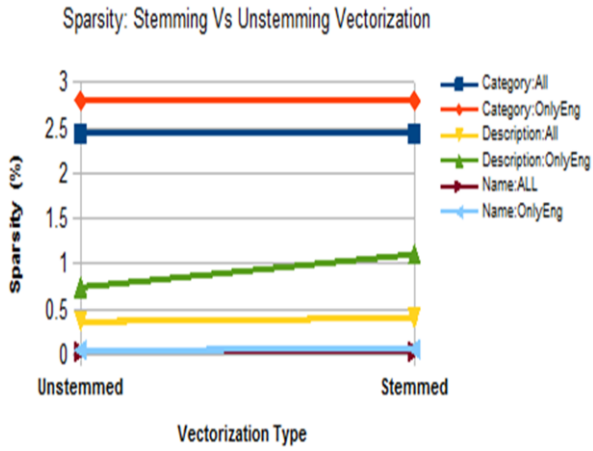


Fig: 3: Sparsity of vectorized matrices for Category, Description, and Name documents considering words in all languages and pure words in the English only taking stemmed words in the English.

As indicated in the above figure-3, sparsity is high when you consider words or tokens only in the English languages. For category attribute, though there are only tokens in the English language, a difference in sparsity is observed. It is so because not all tokens in the documents belonging category are pure English words, for example words could contain symbol '_'.

Description is a multilingual corpus. Sparsity is high if words only in English are considered. Name is multilingual corpus. Sparsity is higher when words in all languages are considered.



In the above figure-3, sparsity increases when English words are stemmed for Description and Name. It does not change in case of category corpus.

V. Linear Regression

Linear regression is one of the simplest statistic approaches to build relationship between variables. In linear regression a model is formed which represents a relationship between a dependent variable denoted by '*y*' and one or more independent variable denoted by '*x*'. From this relationship the unseen values of *y* are predicted. If there is only one independent variable '*x1*', the model can be called 'Simple linear regression' and there are multiple independent variables involved to predict the target variable '*y'* then it can be called 'multiple linear regression'.

The model can be represented as the following function:

$$\hat{y} = \beta 0 + \beta 1 x1 + \cdots + \beta nxn + \epsilon \qquad (4)$$

Here the value of y is linearly dependent on the values of $x1,\ldots..xn$. $\beta 0, \beta 1 \ldots \beta n$ these parameters are known as coefficients which assigns different weights to the values of *x*. ϵ is the error term mainly because of noise found in the data.

Linear models are visualized by scatter plots (Fig 5) where the model tries to generate a line that best fits the data points.

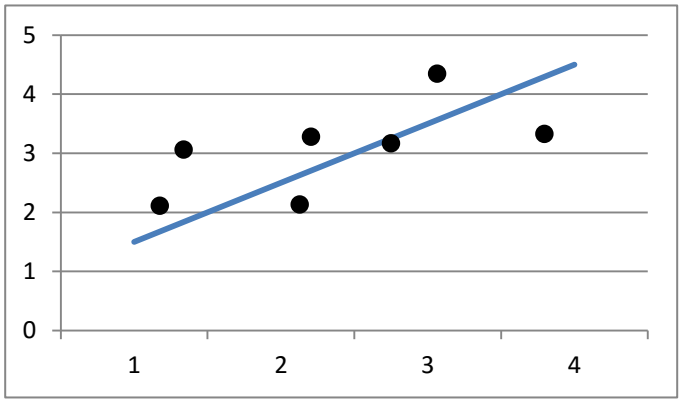


Fig 5: Scatter plot to visualize linear regression model.

A. *Cross validated prediction*

After vectorization of some of the attributes namely Name, Category and Description there were more than around thousand columns for these each category. One of the reasons was playstore contains applications from all over the world in different languages. Different languages have different words thus constructing a large feature vector. In order to reduce computation time and memory usage we have done text vectorization on the 'English words' only and also reduced our dataset to have the information of 3000 applications. Then to reduce the data dimensionality we have done Pricipal Component Analysis (PCA)[5] to reduce our feature number to 100.

Before training the linear model with this data we have randomized the dataset by k-fold cross validation technique. In case of linear regression 4-fold cross validation was used so that each time there is a 75-25 split for train and test data. We have ploted the cross validated outputs (measures vs predicted) of app scores to visulaize how our model fits the data.

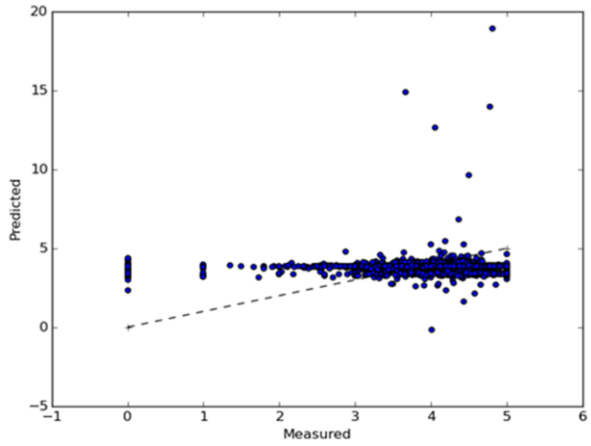


Fig: 6: Using all features

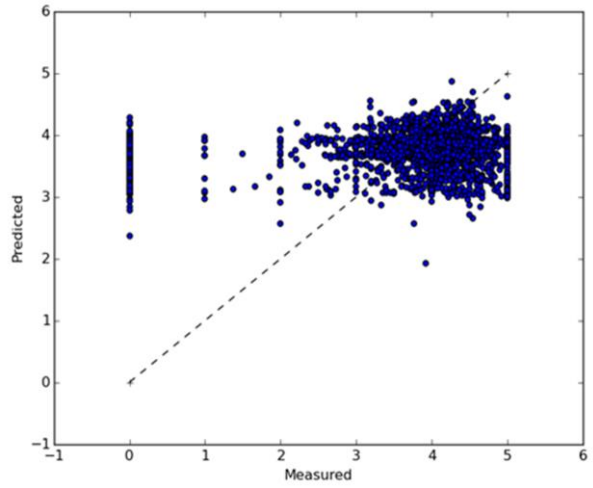


Fig: 7: Without higher dimensional features

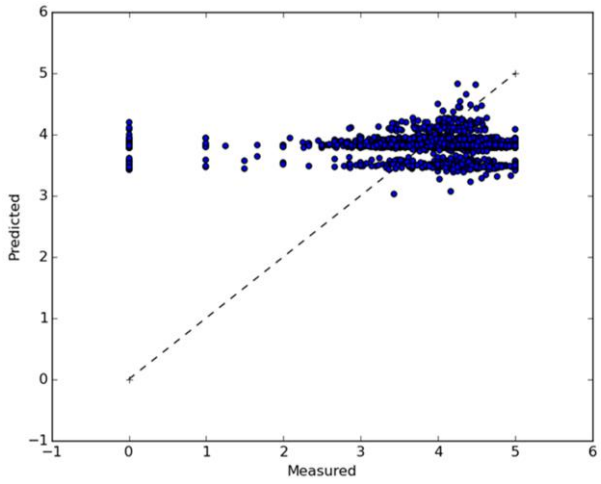


Fig: 8: Without Vectorized features

To see how feature sets are affecting our models we have first calculated the prediction using all of our selected features (Fig:6). Then we removed the features that have high dimensional feature vectors (Fig:7). In the latter case we did not do any PCA as the number of features were not that much large. Finally, we have tried to predcit the scores using only the numerical values where no vectorized features were used (Fig:8).

B. Performance of the linear model

For performance measurement of the model we have used the least square methods and the variance score for different set of train and test data. In the least square method we calculate the residual sum of squares of the error ϵ where

$$e = y - \hat{y} \qquad (5)$$

Thus the residual sum of square (RSS) can be represented as:

$$RSS = e_1^2 + e_2^2 + \cdots + e_n^2 \qquad (6)$$

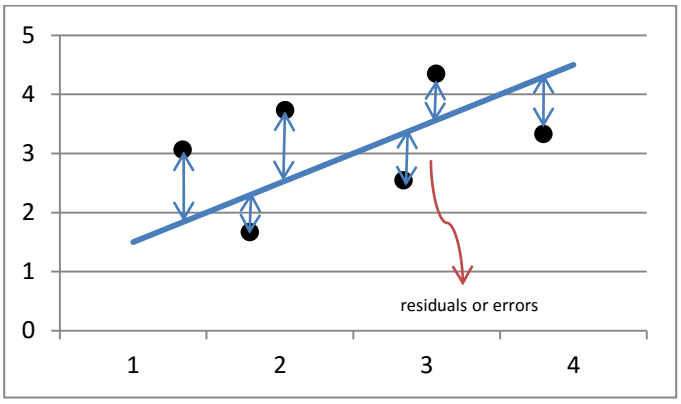


Fig 9: Residuals or error measurement

Variance score is calculated using scikit learn function LinearModel.score() [5]. It returns the coefficient of determination denoted by R^2. R^2 can be represented as follows:

$$R^2 = \left(1 - \frac{u}{v}\right) \qquad (7)$$

Where

$$u = \sum (y - \widehat{y})^2 \qquad (8)$$

$$v = \sum (y - \bar{y})^2 \qquad (9)$$

The best possible score is 1 meaning the model can predict perfectly. A score of 0.0 means the model's prediction does

TABLE I
PERFORMANCE USING ALL FEATURES

4 fold cross validation	iter1	iter2	iter3	iter4
Residual sum of squares	1.27	1.38	1.36	1.10
Variance score	0.56	0.31	0.40	0.23

TABLE II
PERFORMANCE WITHOUT USING HIGH DIMENSIONAL FEATURES

4 fold cross validation	iter1	iter2	iter3	iter4
Residual sum of squares	1.36	1.33	1.18	1.20
Variance score	0.49	0.38	0.19	0.25

TABLE III
PERFORMANCE USING ONLY NUMERICAL FEATURES

4 fold cross validation	iter1	iter2	iter3	iter4
Residual sum of squares	1.10	1.27	1.30	1.33
Variance score	0.37	0.45	0.12	0.24

not really depends upon he input features. This value can also be negative meaning the model performs really bad on the test data. The RSS and scores that we have for our data is given in Table 2,3 and 4[5]:

VI. CLASSIFICATION

Classification is a statistical technique used to identify the category of a new observation based upon the training data whose categories are already known. In the context of our project we have various features that were numerical, boolean or textual. The usable numerical features were directly used, the boolean features were changed to zero or one and the textual features were vectorized using tf-idf approach. The user ratings of applications were categorized into five classes (1-5) such that ratings ranging from 0-1 has label 1, ratings ranging from 1-2 has rating of 2 and so on. The categorized user ratings were used as labels for machine learning models.

As the data had large number of features, after vectorization Principal Component Analysis (PCA) were used to reduce the dimensionality of data. PCA does a linear dimensionality reduction on the data using Singular Value Decomposition and keeps only the most significant vectors of the data such that the data is projected to lower dimensional space [5]. In order to make our performance results reliable we used Kfold Cross Validation technique to train and test various classifiers. This method is an improvement of data holdout method for cross validation. In holdout method the data is simply splitted into training and test set in certain ratio which is used to train and test the models. The results can highly depend upon the nature of data in the training and test in case of holdout method. However, in Kfold Cross Validation, the data is splitted into k subsets and holdout method is repeated k times. In every fold, k-1 subsets are used to train and remaining one is used to test the data. In this method every data point gets an opportunity to take part in both train and test. The disadvantage of this method was that the training of the models needed to be done k number of times which increases the computational time of the overall classification process [6].

PCA was done by varying number of components and Kfold cross validation approach was implemented to analyze the performance of various classifiers. We utilized Scikit's library implementation of Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest Classifier, Logistic Regression Classifier, SGD Classifier, Ridge Classifier, Decision Tree Classifier, K Nearest Neighbors Classifier and Linear SVC to compare the performance of these classifiers. The performances of these classifiers were compared using the accuracy scores. The accuracy score is the ratio of sum of true positive values and true negative values to the total number of samples n. The accuracy score can be well explained in context of supervised learning with two possible classes. In case of two classes, the overall result can be represented as a 2x2 confusion matrix with values true positive (tp), true negative (tn), false positive (fp) and false negative(fn) as shown in table below:

TABLE IV
CONFUSION MATRIX

	Predicted	
True Values	True	False
True	TP	FN
False	FP	TN

The accuracy, precision and recall of for n samples can be calculated following the given equations of (10), (11) and (12):

$$accuracy\ (a) = \frac{(tp + tn)}{n} \tag{10}$$

$$precision\ (p) = \frac{tp}{(tp + fp)} \tag{11}$$

$$recall\ (r) = \frac{tp}{(tp + fp)} \tag{12}$$

We have used accuracy as the criteria for evaluating performance of our classifiers [7]. The accuracies of various

classification models with various numbers of components of PCA are as follows:

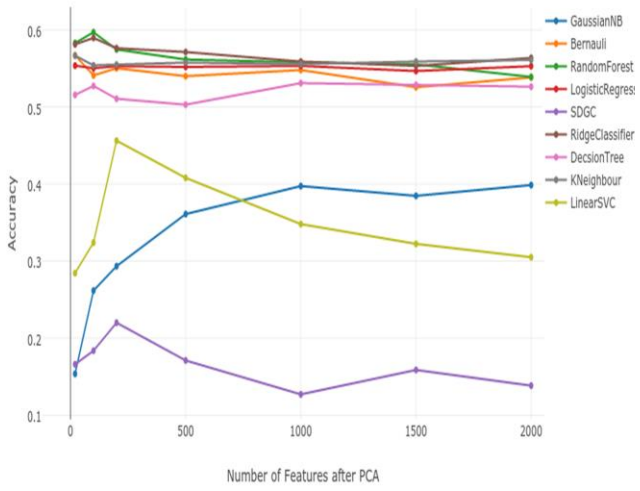


Fig.10. Accuracies for various classifiers

The classifier had better performance with accuracy scores in range of 55 to 60% when the PCA was done to 100 components. Thus, there was need for improving the performance of the classifier. The classifiers that had better performance were chosen and Grid Search Cross Validation Technique was used to find the optimal parameters for the classifier. The grid search technique uses the combination of all possible parameters provided and evaluates the performance of the model. The grid search technique helps us to identify which set of parameters gives best performance. In our context, Random Forest, Ridge Classifier, K nearest neighbor Classifier, Logistic Regression and Decision Tree had good accuracy scores. So, Grid Search CV was used to find out the best combination of parameters and evaluate their performance. The performance of these classifiers before and after the tuning is listed in the table below:

TABLE V
INCREASE IN ACCURACY AFTER TUNING

	Accuracy Before Tuning (%)	Accuracy After Tuning (%)
Random Forest	59.7	69.33
Ridge Classifier	58.9	64.7
KNN Classifier	55.4	62.11
Logistic Regression	55	65.7
Decision Tree	52	60
	Number of Components	*100*
	Data Points	*3000*

After using Grid Search CV, we were able to improve the performance of various classifiers as shown in table above. The Random Forest Classifier was able to predict the ratings of test data set with an accuracy of nearly 70%. However, we had certain limitations while computing these results. The computational resources we had were only able to handle the vectorization of 3000 data points. Also, the text vectorizer excluded all words that were not in English language.

VII. CONCLUSION

The hardware infrastructure we used was not enough for vectorizing the whole dataset of 600K applications so we had to use only 3000 data points in order to limit the computational time and memory usage. Increasing the data point may have resulted in more efficient training model having higher accuracy rate. The text vectorizer we used was also filtered to use only the English words to avoid memory issues. So, significant portion of data may have not been considered for computation. In future we have plans to do parallel processing to handle all available data and also use feature correlation between features sets to find out most usable features

ACKNOWLEDGMENT

We would like to thanks the instructor Dr. Somya Mohanty, of the course CSE 4990/6990, where we were introduced to the world of big data and machine learning. We are thankful for giving us the opportunity to implement various machine learning tools in our project where had our hands-on experiment of working with big data and machine learning.

REFERENCES

[1] "Google Play", https://en.wikipedia.org/wiki/Google_Play, (current December 8, 2015)
[2] M. Lins, https://www.github.com/GooglePlayAppsCrawler
[3] C. S. Perone, "Machine Learning: Text Feature Extraction", http://blog.christianperone.com/2011/10/machine-learning-text-feature-extraction-tf-idf-part-ii/, December 4, 2015
[4] M. Bhoge., "Python Scikit-learn to simplify Machine Learning : Bag of words To TF-IDF", *http://www.datasciencecentral.com/profiles/blogs/python-scikit-learn-to-simplify-machine-learning-bag-of-words-to, September 25, 2013*
[5] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
[6] J. Schneider, https://www.cs.cmu.edu/~schneide/tut5/node42.html
[7] C. Elkan, "Evaluating Classifiers", http://cseweb.ucsd.edu/~elkan/250Bwinter2012/classifiereval.pdf, January 20, 2012