PREDICTING MOBILE APPLICATION SUCCESS

TUCKERMAN, C.J.

Abstract. A machinery for downloading and extracting features about applications from the Google Play Store was developed and deployed, and the resulting data set was used to train three different models to predict the success of a mobile application; a naïve bayes based text classifier for the description of the application, a generalized linear model which categorizes applications as successful or not, and a linear regression which predicts the average rating of the application. The performance of the models is not sufficient to justify their use in driving investments in new applications, however interesting observations about the ecosystem, such as the current trend in photo sharing applications, are elucidated.

1. Introduction

Mobile applications have turned into an enormously profitable business, with revenue from mobile applications expected to exceed fifty billion USD by 2016 [6]. These profits are not distributed equally amongst developers, with forty-seven percent of developers making less than one-hundred USD, more than half of which make nothing at all [5]; creating a successful application is not easy. Luckily, an enormous amount of data about mobile applications is made available publicly by Google, Apple, and Microsoft by way of the websites for their app stores.

In this study, features extracted from the data made available on Google's Play Store website is used as input to two different models. Each model predicts the success of a given application, and interesting observations about their behavior are discussed.

2. Data Collection

Google makes data available about its applications on http://play.google.com. These pages contain data which can be extracted such the name of the application, the description, the number of installations, the average rating of the application, and many more features. In order gather as much data as possible without worrying about which features would eventually be used, a web crawler based on Scrapy [3] was created, deployed on Amazon Web Services (AWS) Elastic Compute Cloud (EC2), and the entire rendered DOMs of the application pages were downloaded to AWS Simple Storage Service (S3) and stored as HTML. The crawling job is currently still crawling, and at the time of training models had downloaded data for more than 1.3 million applications.

For each desired feature, a feature extracting function was written which, given an input HTML file, would output just the feature in question. For all feature extracting functions and all output HTML files, features were extracted and the resultant data is stored in AWS Relational Database System (RDS). As new output HTML files are downloaded, all registered feature extractors are automatically ran on it, and newly registered feature extractors are backfilled from the catalogue of previously downloaded HTML files.

3. Features, Preprocessing, and Labeling

Numerous feature extractors were developed: features extracted include the average user rating, number of {5, 4, 3, 2, 1} star ratings, the number of installations, the description, the name of the application, whether or not the developer is a top developer, the size of the Android Application Package (APK), when the most recent update was published, which Android SDKs are supported, the number of "+1"s on the

Date: December 12, 2014.

application, the price, and more. These extractors were written as a combination selectors [4] and regular expressions.

4. Principal Component Analysis

Principal component analysis was performed on the inputs to the GLM and linear regression models. While training these models, cross validation scores were lower than expected which might be explained through overfitting. In order to help alleviate some of this problem, principal component analysis was performed on the continuous features and the first two principal components were stored in the RDS database.

5. Text

Preparing the description for the naïve bayes algorithm involved using the NLTK in python [1] to both remove stop words and perform some basic stemming on the words in the description, bringing the feature vector dimension to on the order of one-half-million.

5.1. **Success Metrics.** The most obvious choice for a success metric would be revenue, however this information is among the small amount of information not available publically. Instead, we use number of installations and average user rating as a proxy for success, the distributions of which are seen in figure 1. Hoping to select the top five percent of applications as successful, a natural region of success is found. For some application x with average rating x_{score} and number of installations x_{installs}, the successfulness of an application, success x is defined as:

$$\text{success } x = \mathbf{1}\left\{x_{\text{score}} >= 4.5\right\} \cdot \mathbf{1}\left\{x_{\text{installs}} >= 5 \times 10^4\right\} \tag{1}$$

This region encompasses two clusters; one extremely high numbers of downloads and one with close to one-hundred-thousand downloads. The cluster containing applications with large numbers of downloads included those applications from developers such as Google, Facebook, and Snapchat. The cluster with a smaller number of installations contained a large variety of publishers but mainly consisted of very highly rated card games and highly rated applications targeting non-US markets.

We also consider average user rating as a possible success metric to see how a relatively straightforward implementation of linear regression performs.

6. Prediction

6.1. **Naïve Bayes.** The first model attempted was to build a model which would classify an application as successful or not based on its description. A naïve based classifier text classifier was chosen, and the implementation was written in Java for use in a MapReduce pipeline over the text [2]. Cross-validation was performed so that the performance of the model might be measured, as seen in figure 2.

While not the best performing algorithm, the run-time performance of the algorithm was extremely good and allowed for quick changes, and some interesting results can be seen from the intermediate data of the algorithm.

$$p\left(x_{\text{“photo”}} = 1 | \text{success } x = 1\right) = .35 \tag{2}$$

$$p\left(x_{\text{“share”}} = 1 | \text{success } x = 1\right) = .31 \tag{3}$$

$$p\left(x_{\text{“download”}} = 1 | \text{success } x = 1\right) = .0001 \tag{4}$$

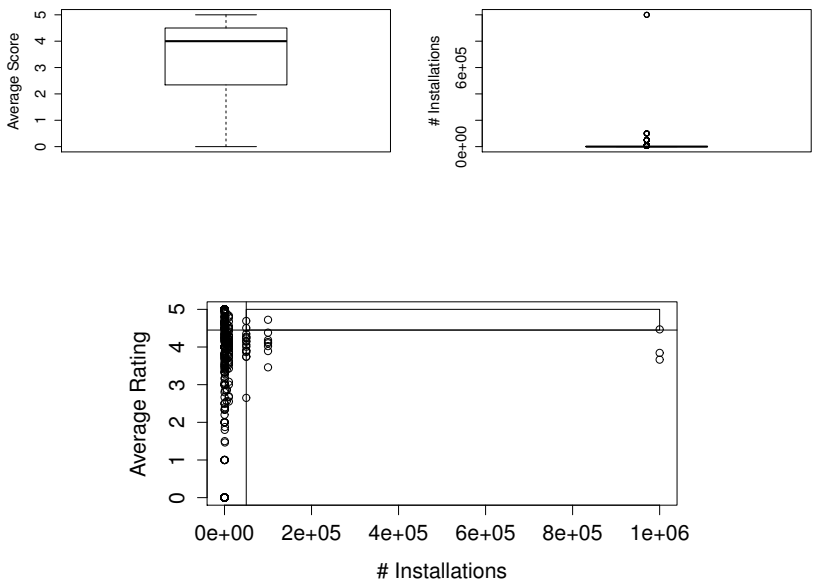

FIGURE 1. The distribution of average scores and numbers of installations.

FIGURE 2. Training curve for the naïve bayes model.

6.2. **GLM.** Given our definition of success x, one could say that success $x = 1$ is distributed Bernoulli with parameter $\phi = p(\text{success } x = 1) = .05$. We could then use our continuous feature vectors to train a generalized linear model to predict success x (in this case given the distribution, the GLM would be logistic regression). Cross validation was performed so that the performance of the model might be measured.

The cross-validation error is remains high, suggesting the possibility of overfitting. We instead switch to performing logistic regression on the first two principal components of the continuous features (described above).

The comparison between the two approaches is shown in the training curves in figure 3.

6.3. **Linear Regression.** Additionally, we try and use the continuous features to try and predict just the average rating of the application — one might hypothesize that predicting a popular application might be

FIGURE 3. Training curve for GLM models. (Left: Raw, Right: PCA)

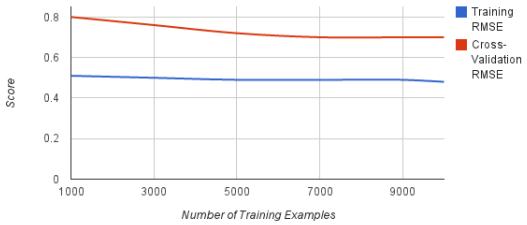

FIGURE 4. RMSE training curve for the linear model.

difficult whereas predicting whether an application makes users happy or angry might be easier. Figure 4 shows the performance of this model via the RMSE.

7. Conclusion and Further Discussion

The accuracy of the above discussed models show that there does exists some, albeit limited, predictive power in the eventual success of an application which can be garnered from the publically available information on the Google Play Store.

More interestingly, certain insights can be garnered from looking at the models themselves instead of just their output. It is shown that thirty-five percent of all successful applications contain the stem "photo" (eqn. 2) and thirty-one percent of all successful applications contain the steam "share" (eqn. 3) somewhere in their description. We are able to start to build a picture of the genres of applications in which users are interested by using this model.

Other interesting observation include those from the GLM model, such that a high price and long length of description are penalized, which is intuitive, but the degree to which they are important is surprising, as seen in figure 5.

8. Future

While the models were shown to be able to learn to some degree of success and interesting conclusions can be drawn from the data, it was not shown that our definition of success x actually correlates with any economic success. There exists companies, such as AppAnnie, which attempt to make available this data for commercial purposes. A developer interested in modeling the current state of affairs to determine which kind of application to develop, or an investor wishing to invest using models such as these would do well to subscribe to such a feed of information to ensure that success x correlates with revenue.

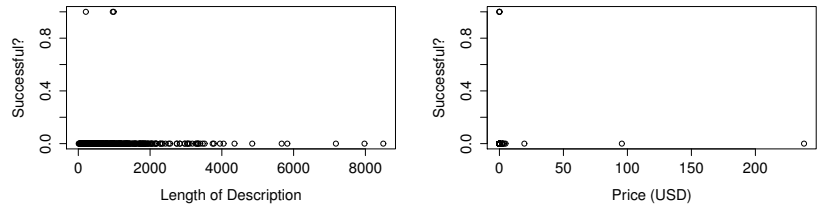

FIGURE 5. Success versus {length of description, price}

An additional future area of exploration is training on historical data. The most important conclusions were related to what the trained models say about the current state of affairs — it would be a good space to explore how trends and changes in the ecosystem could be explored using a similar procedure on historical data.

Additionally, during the exploration of the naïve bayes model, certain stems such as "download" (eqn. 4) were found to be present rarely in the descriptions of successful applications. Spot checking applications with the stem "download" in their description turned up numerous low quality applications that appeared to be spammy and very poorly rated. Building classifiers to find these types of low utility, potentially dangerous applications could help protect consumers from accidentally installing bad applications, or perhaps even lower their ranking in the search results.

Finally, spot-checking some of the misclassified applications by the naïve bayes model seemed to show that keyword stuffing (i.e. including unrelated, comma-separated keywords in the description to help with discoverability) was responsible for some of them. Exploring methods for detecting these non-sentence segments of the description would be a natural next step to remedying the problem.

ACKNOWLEDGEMENTS

The author would like to thank the CS229 course staff for their plurality of help, guidance, and direction for this project.

REFERENCES

[1] Bird, Steven. Loper, Edward. Klein, Evan. *Natural Language Processing with Python*. OReilly Media Inc. 2009.
[2] Maskey, Sameer. *MapReduce for Statistical NLP/Machine Learning*. 2012.
[3] *Scrapy*. http://doc.scrapy.org/en/latest/intro/overview.html 2014. Web. 01 Nov 2014.
[4] *Selectors*. *Mozilla Developer Network* Sept 2014. Web. 01 Nov 2014.
[5] Wilcox, Mark. Voskoglou, Christina. *State of the Developer Nation Q3 2014*. *Vision Mobile*. July 2014.
[6] *Worldwide mobile app revenues from 2011 to 2017 (in billion U.S. dollars)*. *Statista*. 2014. Web. 30 Nov. 2014. http://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/