CPTS 315
Luke Clancy, 11432090
Final Project Report

Google Play Apps Analysis Report

Introduction:

The Google Play App market is a black box. Google does not release information on the apps inside it, so many developers and potential entrepreneurs rely on heuristics and personal research to find gaps in the market. Thankfully, there are scrapers who have been collecting data automatically from the Google Play website. This information can then be utilized by machine learning engineers to provide useful and usable information for those entering this market.

I found a dataset in this arena provided by one of these stated scrapers. This data provides information on 47 thousand different apps. This by itself is not necessarily remarkable. What drew my attention beyond this was his claim that “I'm still working on more records (should be millions) and I will keep updating this whenever I can,” (Boulad). By providing data on a million different apps (especially when the market consists of just above two mission), a very clear image of this market appears. Once this data comes in a community will start to gather around this dataset, and I wanted to be one of the first to contribute.

Some of my initial questions when entering this project were:

1. What is the best way to parse through this text data?
2. What machine learning algorithm would be best to solve such a question?
3. How correlated are the description and category of an app to its success?
4. What can I learn about Data Mining in python?
5. How should text information be formatted for regression?

In order to answer these questions I fell into pitfalls, struggled through, and researched my way to a conclusion. At the beginning of my project I simply assigned each word to a position in the vector, and put it through a classifier. It took an embarrassingly long time for me to learn my mistake. I had more than 300,000 different unique words and

they contained no context as to their relevance within the sentence. I attempted to put this through multiple regressors before I realized the problem.

This is when I started looking into how to best process text data, and also where I utilized the python NLTK (Natural Language Tool Kit) library. The NLTK library contained the Rake class, which automatically decided how important a phrase is to the text as a whole. By using this library, I didn't have to store as much data, and I had a good way of deciding which data to hold on to based on importance.

My results found correlation between categories, description and the output. The results of my research are useful for further research on the dataset, and may be expanded for use in the industry.

Data Mining Task:

The main Data mining task was to properly format my input data so that it could be interpreted and understood by an MLP (Multi-Layer Perception) Regressor. The input I used is:

1. Primary Category
2. Secondary Category
3. Description

For the output of my project I produce a standard deviation over the error of my model. I compare this to the standard deviation of the target value. This measure assesses the accuracy of my model.

Part of my task was deciding on the model to use for classification. I decided on the MLP Regressor since it was continuous, complex, and could handle non-linear decision boundaries. Thats not to say I didn't try other models, but it was clear to me that the MLP Regressor was the ideal.

Although I generally understood the concept of what Natural Language parsing was, and how Neural Networks work, I had yet to utilize this information in a real world setting. I am glad to have learned these technologies in a real world scenario, and to have gained practical experience and knowledge. This practical application is what I set out to understand, and I succeeded in that regard. Through fighting my main challenges of how to properly format text data, how to deal with large amounts of data, and how to

configure a MLP-Regressor, I researched and implemented solutions to a real world application.

Technical Approach:

The categories are the simplest input into my project. For each category I have two featuresne for if the features is listed as a primary category, and one of the features is listed as a secondary. There are only a few categories, so it doesn't explode the feature size. Now onto the more complex solution of parsing the description.

The first thing I do to the descriptions is to send it through a language checker. The language detector used is from the library langdetect. Although I understand the utility in having multiple languages, other libraries do not work well with them, and end up exploding the feature size. In addition to this, the vast majority of applications are in English. As such I decided that foreign languages are outside of the scope of this project.

Once the language detect is completed, and all the non-English data is removed, we go on to use NLTK's Rake. Rake detects important phrases at the center of sentences and matches the word up with its corresponding importance. This all at once removed vast amounts of complexity and increased utility at the same time.

Now that we have the importance or each phrase to its paragraph, we can utilize this information to obtain the usefulness of each phrase to the model. By adding up the importance over all occurrences of a phrase, we have a general usefulness metric to gauge which features are more useful to the model than others. Using this information, we then cut out a **lot** of features.

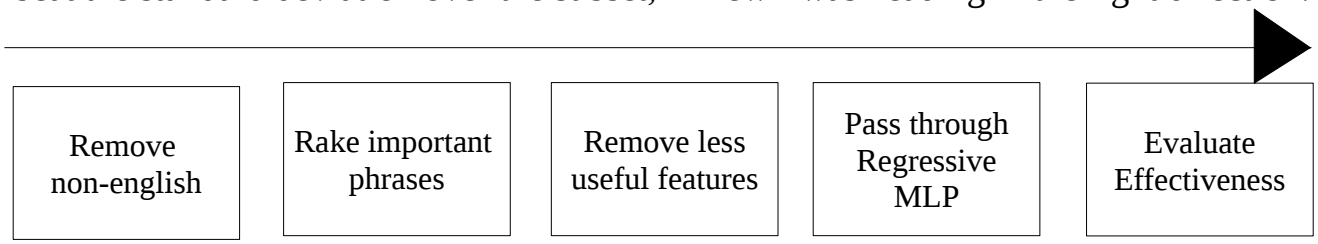
```
Cutoff = .1
maxFeats = 5000
#preliminary dimensionality cutting through deleting irrelevant features
WBKeys = list(wordBucket.keys())
while len(wordBucket) > maxFeats:
    for key in WBKeys:
        if cutoff > wordBucket[key]:
            wordBucket.pop(key)
    cutoff += .2
    WBKeys = list(wordBucket.keys())
    print('WB len: ' + str(len(wordBucket)))
```

I also included a TruncatedSVD after the features were reduced to 5000. I would have inputted more than 5000 features into the TruncatedSVD, however, the SVD was having

problems with large amounts of features and constantly produced technical issues when this happened. These technical issues slowed my progress.

After separating my data to create a testing subset, finally we are ready to learn using the MLP Regressor. After multiple trials, I settled on a MLP with hidden layers of size (200, 300, 200, 100). This provided me with both an extensive range of entry nodes as well as the decision power of a MLP with four layers.

After I get the output from my model, I test it over the testing subset and acquire the standard deviation over that subset, and the standard deviation of my guesses. Once I beat the standard deviation over the subset, I knew I was heading in the right direction.



Evaluation Methodology:

As my output is a continuous variable, evaluation was done using the standard deviation of my output and the real value. This was compared against the standard deviation in general (output and average) to ensure results were apparent. Improvements were verified against previous results.

Results:

Unfortunately the results of my research are underwhelming. Given the size of the test data and multiple trials, I have verified that I have beat the standard deviation by .18 points (when predicting app rating).

When predicting the rating of apps:
Standard Deviation from Average: 1.47
Standard Deviation from Prediction: 1.29

In cases where someone is evaluating a large set of applications, my research may be immediately useful. But in the general case for predicting results over a single app, this research is not immediately useful due to the lack of accuracy increase over the standard deviation. This research may also be useful as the basis of future research on the dataset.

Discussion:

These results can be built upon to create information that is useful to business owners. Currently a lot of information is lost in feature reduction. Information may be regained by implementing further text parsing techniques, or by implementing other SVD based alternatives to feature reduction.

It is worth exploring the effect of regularization on the different Rake Importance values before vectorization. This would have the effect of giving each app equal importance on the MLP Regression model.

Lessons Learned:

By far the most important thing I learned from this process is to do research before implementation, otherwise you will waste a lot of time. This happened twice in my project. Once was with extracting text features. I did not understand the importance of preprocessing text. That was solved using NLTK's Rake. The second time it was with the Neural Network (MLP Regressor). Neural Networks are not good at handling high dimensions of features. Before this project began I assumed they would handle that automatically. Now I attempt to reduce as much as possible before handing that data to the Neural Network.

Acknowledgements:

1: Boulad, Adam. "Google Play Apps Data." *Kaggle*. https://www.kaggle.com/adamboulad/google-play-apps-data.