Automatic Star-rating Generation from Text Reviews

Chong-U Lim
culim@csail.mit.edu

Pablo Ortiz
portiz@csail.mit.edu

Sang-Woo Jun
wjun@csail.mit.edu

December 13, 2012

Contents

Chapter 1

Background

1.1 Introduction

For our final project, we designed a system that generates five-star ratings for a product based on a review of that product. Ratings are generated by classifying the overall sentiment of a product review on the five-star scale using the PMI-IR algorithm. In the actual implementation of the system, we constrained the domain of product reviews to those of mobile applications on the Google Play App store. It was necessary to constrain the system to a particular problem domain as different problem domains will produce different sentiments for the same words. The system we created can serve as a proof of concept for generating five-star ratings based on product reviews for products and services of various kinds.

1.2 Motivation

Accurate customer feedback is much sought after by commercial enterprises. The information is quite valuable and affects a comapny's bottom line by aiding in the design of new products/services or helping to fix those that have not been well-received. This being the case, many companies invite their customers to submit reviews for products they have used. The format of these reviews can vary widely from the "Like" format used by Facebook to lengthy surveys. Some formats, of course, are used more in practice than others.

One well-established format for customer reviews is that of five-star ratings paired with short text reviews. Feedback in this form allows a customer to both give an overall rating for a product/service and provide meaningful commentary about the product/service. Unfortunately, a problem related to the accuracy of the reviews comes up when feedback is formatted this way. A particular star rating can mean completely different things to

two different customers. For example, suppose there is one customer who sees the world in black and white and another customer who sees the world along the full five-star gradient. A one-star review from the first customer could mean anything between minor dislike and absolute hatred. The content of his/her text review would likely shed light on that. A one-star review from the second customer would definitely indicate absolute hatred. Though this example is quite extreme, it illustrates a flaw inherent to this system of obtaining feedback: some star ratings will not match the corresponding sentiment seen in the accompanying text commentary. For a commercial enterprise looking to collect accurate feedback from its customer reviews, this should be a matter of much concern.

To acquire more accurate information from customer feedback, we propose the removal of five-star ratings from these review formats entirely. Instead, we propose that the star ratings be generated in a uniform fashion from the sentiment of the text commentary.

1.3 Contributions

Chapter 2

Design

2.1 Data Collection

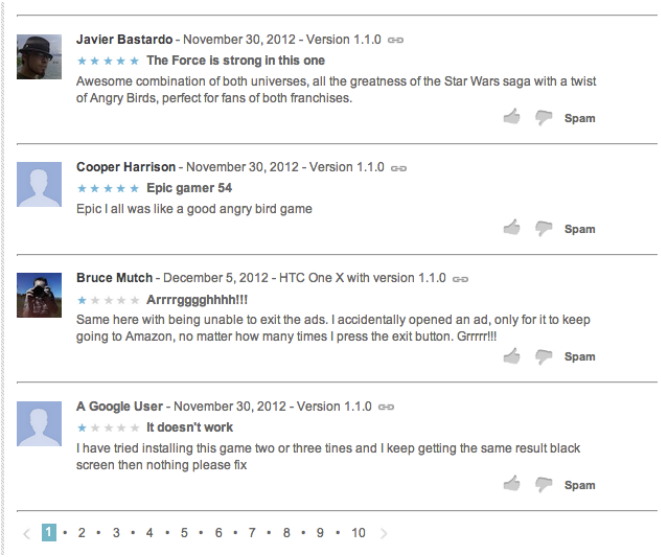

Figure 1: User Reviews on the Google Play Store

Google does not supply any publicly available Application Programming Interfaces (APIs) to query for data or results on the Google Play Store. Also, in searching for user reviews for a given application, one has to first navigate to the associated page and is at first initially presented with only the first 10 reviews, with the rest only asynchronously provided when a user clicks on the paginated links at the bottom (Figure 1).

However, attempts to automated the querying for more results were blocked by Google. As such, we had to manually obtain the data by first clicking through all the provided

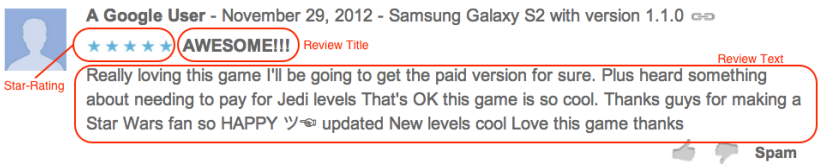

Figure 2: User Reviews on the Google Play Store

pagination links at the bottom in order to load all the user reviews. After which, we made use of Google Chrome's View Source functionality to extract all DOM elements between the `<html>` element. A python script was then used to locate each user review, and extract out 3 main features, namely, the **Review Title**, **Review Text** and the **Star-Rating** (Figure 2).

2.2 Phrase Extraction

2.2.1 Parts-Of-Speech (POS) Tagging

The text gathered as part of the **review text** often varies in length (i.e. number of sentences, number of words). Our aim is to extract out the important phrases in each review and use them to guide our calculation of the overall sentiment of the user's review. In order to do so, we first process the review text by splitting them into sentences, followed by determining the part-of-speech (POS) tags for each word in the sentence.

2.2.2 Pattern Matching

	First Word	Second Word	Third Word (Not Extracted
1.	JJ, JJS, JJR	NN, NNS	anything
2.	RB, RBR, RBS	JJ, JJS, JJR	neither NN nor NNS
3.	JJ, JJS, JJR	JJ, JJS, JJRS	neither NN nor NNS
4.	NN, NNS	JJ, JJS, JJRS	neither NN nor NNS
5.	RB, RBR, RBS	VB, VBD, VBN, VBG	anything
6.	DT	JJ, JJS, JJRS	anything
7.	VBZ	JJ, JJS, JJRS	anything

Table 3: Patterns of POS Tags for 2-Word Phrase Extraction

Given the tokenized form of the sentence, we proceed to extract 2-word phrases from the review via pattern matching against a set of phrase patterns, as shown in Table 3. The patterns of POS tags are extended from an original list proposed by Turney [**?**]. Our

extensions were required in order to better match style of reviews on the Google Play Store, and also to match the tokens used in the *NLTK* library. One notable addition was the addition of *superlative* adjectives (JJS), which we were emperically found to occur substantially in our dataset, with sentences such as " *. . . this game is* ***the best !*** *. . . .*" We also added comparative adjectives (JJS) to our matched tags. Figure 4 shows the results of our phrase extraction on a user review in our dataset.

```
text = "This is a great game but lags horribly. I'm
   constantly dying because it freezes up on me! Pleasseee
   do something to fix it!:)"
>>> get_phrases_from_text(text)
[[('a', 'DT'), ('great', 'JJ')], [('great', 'JJ'), ('game',
   'NN')], [('constantly', 'RB'), ('dying', 'VBG')]]
```

Figure 4: User Reviews on the Google Play Store

2.3 Phrase Sentiment

In order to gain insight regarding the user's sentiment based on his or her review, we make use of the extracted phrases from Section 2.2 as a measure of a user's sentiment towards the application. The basis of extracting information from the phrases is derived from the mutual information of the words contained within the phrase, against known words that we identified as having either *positive* or *negative* sentiment polarity.

2.3.1 Pointwise Mutual Information (PMI)

The Pointwise Mutual Information (PMI) between two words [**?**] is defined with the following equation:

$$PMI(word_1, word_2) = \log_2 \left(\frac{Pr[word_1 \; and \; word_2]}{Pr[word_1] \times Pr[word_2]} \right)$$

In the equation, the term $Pr[word]$ refers to the probability of the word appearing if we were to sample randomly from a text corpus, while $Pr[word_1 \; and \; word_2]$ refers to the probability of both words appearing together. The ratio between p(word1 & word2) and p(word1) p(word2) is thus a measure of the degree of statistical dependence between the words. The log of this ratio is the information gain from one of the words when the other is observed. The PMI score has an additional property in which it assigns a proportionally higher score to the words that appear infrequently in a given corpus, but with a higher likelihood of appearing together in the event they do. [**?**]

2.3.2 Semantic Orientation

The semantic orientation of a given phrase is thus given by the formula:

$$SO(phrase) = PMI(phrase, \texttt{<positive>}) - PMI(phrase, \texttt{<negative>})$$

In the formula, the terms `positive` and `negative` refer to words which are respectively associated with positive sentiments and negative sentiments.

2.3.3 PMI-IR

In Turney's implementation, he made used of the the words *"excellent"* and *"negative"*. Also, he introduces a technique called PMI-IR (Information Retrieval) in order to estimate the PMI values by using hit counts returned from search engine queries. He estimates the semantic orientation using the formula:

$$SO(phrase) = \log_2 \left(\frac{hits(phrase\ NEAR\ \text{“}excellent\text{”}) \times hits(\text{“}poor\text{”})}{hits(phrase\ NEAR\ \text{“}poor\text{”}) \times hits(\text{“}excellent\text{”})} \right)$$

The term *NEAR* in the equation above is a search query modifier which only returns results in which the items are located within a fixed number of words between each other.

As a basis for our implementation, we used a similar PMI-IR estimation for semantic orientation of the extracted phrases, but using Google as the search engine of choice, and replacing the *NEAR* operator with Google's corresponding *AROUND(n)* operator (where n indicates the maximum range of words of which the two items in the search could differ by.)

However, based on our initial findings, we identified two problems with this approach. Firstly, automated queries to Google go against its Terms of Service Agreement. Despite attempts to decrease the likelihood of being detected such as by limiting our queries to a random interval between 10-15 seconds, we got blocked after about 50-60 calls. A survey of other alternative search engines either resulted in similar limits enforced (Bing), or deprecated APIs (Yahoo, Altavista). This made the process a slow and cumbersome endeavour.

The second problem was associated with the accuracy of the calculated results, which are outlined in greater detail in Chapter 4. In short, the results failed to convincingly associate phrases with the correct polarity, partly due to the fact that the Google Search API does not actually return accurate values for the number of hits.

2.3.4 Corpus-Derived PMI

In order to find a method of calculating SO scores for phrases quickly, and without limit, we turned towards an offline SO estimator which uses the *NLTK* `movie_reviews` corpus. The corpus is a collection of user reviews from the *IMDB Movie Database* , and is separated into sentences associated with *positive* and *negative* reviews. In our implementation, we first created a probability distribution of words which appear in the positive reviews and negative reviews separately. Next, we estimated the semantic orientation of a phrase using the formula:

$$SO(word) = \log_2 \left(\frac{Pr_{pos}[word]}{Pr_{neg}[word]} \right)$$

Where the term $Pr_{pos}[word]$ is the probability of occurence of the word *word* using the frequency distribution model acquired from the *positive* corpus. The term $Pr_{pos}[word]$ naturally refers to the corresponding definition associated with the *negative* corpus. We can easily extend this to a phrase with two words, such as "very annoying" by calculating $SO(\text{“}very\text{”}) + SO(\text{“}annoying\text{”})$. This allows us to attain values such as $SO(\text{‘}excellent\text{‘”}) = 1.2432$, $SO(\text{“}poor\text{”}) = -0.9219$, $SO(\text{“}very\ annoying\text{”}) = -0.3952$ and $SO(\text{“}very\ happy\text{”}) = 0.6716$. Thus, this gives us a nice form in which positive phrases are given positive values, and negative phrases are given negative values. Also, the magnitude of a value indicates a stronger association with the polarity.

2.3.5 Text Semantic Orientation

Given that a review text often contains several sentences, which in part contain several phrases which match our phrase patterns, we calculate the semantic orientation of a given text by averaging the semantic orientation values of its constituent matched-phrases. We present the calculation of two reviews from our dataset, one corresponding to a positive user review (Figure 5) and another corresponding to a negative user review (Figure 6). We illustrate the phrase extracted together with its constituent POS tags and the results of the per-phrase semantic-orientation calculation and overall semantic orientation of the review.

Positive Review

Extracted Phrase	POS Tags	Semantic Orientation
is such	VBZ JJ	0.1573
an entertaining	DT JJ	0.3201
entertaining game	JJ NN	-0.1939
is great	VBZ JJ	0.6650
another great	DT JJ	0.4515
Average Semantic Orientation		0.4126

Table 5: Semantic Orientation Calculation for a Positive Review

Negative Review

Extracted Phrase	POS Tags	Semantic Orientation
a big	DT JJ	-0.3309
big fan	JJ NN	-0.7056
a huge	DT JJ	-0.2395
huge problem	JJ NN	-0.7483
not work	RB VBD	-0.0676
even uninstalled	RB VBD	-0.2645
then reinstalled	RB VBD	0.3341
again help	RB VB	0.0792
not stay	RB VB	-0.0695
Average Semantic Orientation		-0.2978

Table 6: Semantic Orientation Calculation for a Negative Review

2.4 Text Polarity

2.4.1 Bag of Words

A second method into gaining insight into the sentiments expressed within a sentence is to determine its polarity. The polarity may be either *positive* or *negative*, and it treats polarity using the **Bag of Words** (BOW) model [**?**]. In this model, a document is represented by the occurrences of words in it regardless their position in the document. This is implemented with the *NLTK* `movie_reviews` corpus by creating feature vectors

based on the appearance of words in each sub-corpus, with each sub-corpus forming a single training instance. We do this separately for both the *positive* and *negative* categories.

2.4.2 Naive Bayes Classifier

Classification is then performed by a Naive Bayes classifier [**?**, **?**] using the feature vector of word-appearances, using the following approximation:

$$Pr[polarity|w_1, \ldots, w_n] \approx Pr[polarity] \prod_{n=1}^{n} Pr[w_i|polarity]$$

The LHS of the equation is the probability of the polarity taking a certain value conditioned on the joint probability of occurrence of a given sequence of words $w_1, \ldots, w_n$. By using Bayes' Theorem, the result of the equation can be re-written based on the prior probability of attaining a certain polarity classificaiton $Pr[polarity]$, and the conditional polarity of generating a sequence of words $w_1, \ldots, w_n$ given a known polarity. By assuming conditional independence of the appearance of each word w_i given a *polarity*, we may represent the joint conditional probability as a product of individual probabilities of the occurrence of a word w_i given a *polarity*, $Pr[w_i|polarity]$, thus revealing the formula on the RHS of the equation.

fileid	...	abberline	ably	achieve	...	category
`pos/cv000_29590.txt`	...	`true`	`true`	`false`	...	`positive`
`pos/cv001_18431.txt`	...	`false`	`false`	`true`	...	`positive`
⋮	⋮	⋮	⋮	⋮	⋮	⋮
`neg/cv999_14636.txt`	...	`false`	`false`	`false`	...	`negative`

Table 7: Constructing Naive Bayes Training Data from NLTKs movie_reviews corpus

The *NLTK* `movie_reviews` corpus contains 1000 files associated with the *positive* sentiment, and 1000 files associated with the *negative* sentiment (identified by the category type `"pos"` and `"neg"`.) Table 7 shows how the training data was constructed using the `movie_reviews` corpus. Figure 8 illustrates the process of determining the polarity of a user's review. The text is first tokenized into individual word tokens and then converted into a feature vector. Finally, our classifier is used to classify the feature vector, in which it calculates the label `neg`, indicating a negative polarity in the user's review.

```
>>> sample_text
u'I was fine with having to exit ads on previous Angry Birds
    games, but this is a two -step requirement. You have to
   expand first and then exit the ad, and for me, using a
   Samsung Galaxy S3, it will only take me to the ad link
   when I touch the X. Broken ad exit button will not let me
    play. Uninstalling.'

>>> sample_text_tokens
['I', 'was', 'fine', 'with', 'having', 'to', 'exit', 'ads',
   'on', 'previous', 'Angry', 'Birds', 'games', ',', 'but',
   'this', 'is', 'a', 'two', '-step', 'requirement.', 'You',
    'have', 'to', 'expand', 'first', 'and', 'then', 'exit',
   'the', 'ad', ',', 'and', 'for', 'me', ',', 'using', 'a',
   'Samsung', 'Galaxy', 'S3', ',', 'it', 'will', 'only', '
   take', 'me', 'to', 'the', 'ad', 'link', 'when', 'I', '
   touch', 'the', 'X.', 'Broken', 'ad', 'exit', 'button', '
   will', 'not', 'let', 'me', 'play.', 'Uninstalling', '.']

>>> text_features = word_feats(sample_text_tokens)
{'and': True, 'ad': True, 'S3': True, 'Angry': True, 'it':
   True, 'two': True, 'have': True, 'touch': True, 'You':
   True, 'fine': True, 'previous': True, '-step': True, 'X.'
   : True, 'ads': True, 'for': True, 'Broken': True, 'when':
    True, ',': True, '.': True, 'to': True, 'only': True, '
   exit': True, 'take': True, 'was': True, 'is': True, 'then
   ': True, 'Samsung': True, 'I': True, 'but': True, 'me':
   True, 'link': True, 'play.': True, 'not': True, 'using':
   True, 'let': True, 'with': True, 'expand': True, 'a':
   True, 'on': True, 'Uninstalling': True, 'this': True, '
   button': True, 'having': True, 'will': True, 'games':
   True, 'requirement.': True, 'the': True, 'Birds': True, '
   Galaxy': True, 'first': True}

>>> classifier.classify(text_features)
"neg"
```

Figure 8: User Reviews on the Google Play Store

2.5 Hierarchical Classification

In our implementation, we aimed to provide a way to make use of both the semantic orientation (Section 2.3) and the polarity(Section 2.4) of a user review in order to generate

more accurate reviews on a 5-star rating scale. The approach we took makes use of a **Hierarchical Classifier** [**?**, **?**] in which the review to be classified is first classified based on its polarity. Next, we predict the exact star-rating based on the semantic-orientation of the review's extracted phrases.

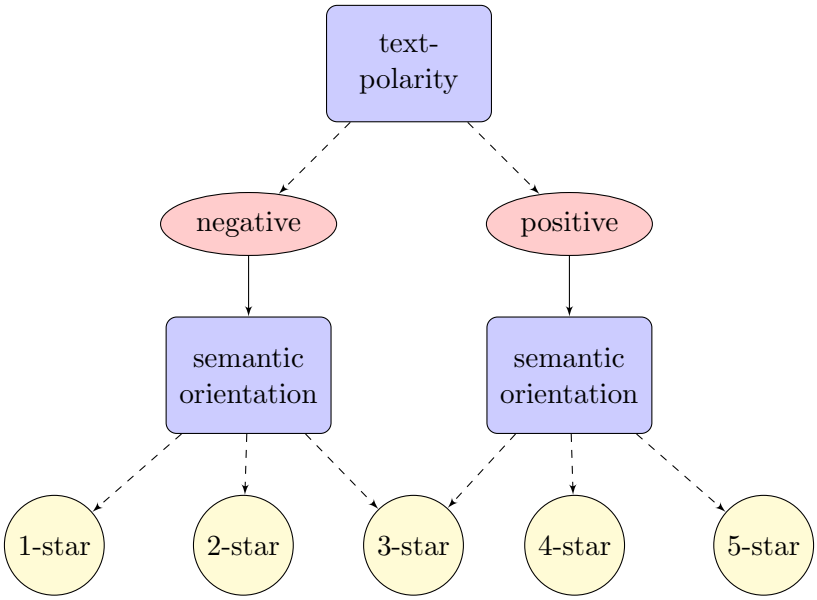

Figure 9: Hierarchical Classification of User Review

Figure 9 illustrates how our system is laid out. A review's text is first classified based on its polarity using a Naive Bayes classifier trained on the Bag of Words model. Once the polarity is determined, we narrow down the range for which a review's star-rating can achieve. A *negative* polarity restricts the star-ratings to $[1.0, 3.0]$ and a *positive* polarity restricts the star-ratings to $[3.0, 5.0]$.

In order to map the semantic-orientation (SO) of a review to an exact star rating, we needed a method of translating the SO value to a star-rating. We approached this using two methods: the first involved empirically determining two SO values for each polarity while the second makes use of an supervized machine learning technique involving **Support Vector Machines**.

2.5.1 Sentiment Thresholds

Once a review is determined to be *positive*, we are restricted to the star-rating values of 3.0, 4.0, and 5.0. If the SO value is below the first threshold, the review is given a star-rating of 3.0. If the SO value is between the first and second threshold, it is given a star-rating of 4.0. Finally, if the SO value is greater than the second threshold, it is given a star-rating of 5.0. A similar reasoning assigns SO values for the *negative* reviews to star-rating values of 1.0, 2.0 and 3.0.

2.5.2 Support Vector Machines

The second method we used involves the use of **support vector machines**, which have been shown to deal well in the domain of sentiment analysis and text classification [**?**, **?**]. Support Vector Machines (SVMs) are a form of supervised learning methods which can be used for classification or regression problems. In a binary classification example, we would train the SVM on a labelled dataset and if they they are linearly separable, the SVM will find a unique separation boundary in the form of a hyperplane with points falling on each side having different classifications. The separation boundary would be one in which the margin is maximized. The general formulation of SVMs as constrained quadratic programming problem is as follows

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & C\sum_{i=0}^{n}\xi_i + \frac{1}{2} \parallel \theta \parallel^2 \\ \text{subject to} \quad & y_i(\theta \cdot \mathbf{x}_n + \theta_0) \geq 1 - \xi_i,\ i = 1, \ldots, m \end{aligned}$$

where x_i represents each training data point, with y_i being its corresponding target classification. θ is the model, or parameter, of the classifier with offset θ_0, while C and ξ represent the penalty and slack variables respectively, which are used as tuning parameters in order to improve the classification results. In our design. we experimented with different feature vectors:

1. Semantic-Orientation Value as the only value in the feature vector.
2. Text Polarity and Semantic-Orientation Value as two values in a feature vector.
3. Title Polarity, Text Polarity and Semantic-Orientation Value as 3 values in a feature vector.

From initial findings, found that using the hierarchical classification approach, we only needed to use the SO-value in order to achieve accurate star-rating scores.

Chapter 3

Experiments

For each approach to generating mobile application review star-ratings we implemented, we executed a corresponding experiment. These experiments were knowingly conducted with limited data resources and severe time constraints. Our experimental setup and results should be judged with that in mind.

As noted in the previous chapter, we obtained two small corpora of Google Play application reviews. One corpus consisted of 1,218 reviews for free applications. The other corpus contained 1,751 reviews for paid applications. The development and test sets used to determine the effectiveness of each star-rating generator were gleaned from these corpora. The actual development and test sets used in each experiment often differed in their precise contents. The results obtained are no less meaningful because of it, however.

3.1 Experimental Setup: PMI-IR

To test the effectiveness of the PMI-IR approach in generating star-ratings, we first had to collect more data than what was at our disposal. For every phrase whose semantic orientation we wanted to analyze, we were required to provide search engine hit counts by the algorithm. We were largely unsuccessful in this endeavor.

Nevertheless, we were able to obtain a small set of data on which we ran experiments. From the first 50 reviews in the free application review corpus, we were able to get some of the hit counts we needed. Of the 50 reviews we wished to analyze, we were able to obtain usable data for 37 of them. It was on this subset of a subset of the test set that we ran our experiment. Unfortunately, when we tried to do the same for the first 50 reviews of the paid application review corpus, we were blocked from using the AJAX interface to Google's search engine after 3 queries. Given this unfortunate sequence of events, we were only able to experiment with the hit counts obtained from the free application

review corpus.

To generate star-ratings, we calculated the semantic orientation of each phrase in each review of our small test set. Then, by examining the produced values in a rather unsophisticated fashion, we set the boundaries for the range of semantic orientation values corresponding to each star-rating. These boundary settings are almost guaranteed to not be optimal. The manner in which they are chosen is simple trial and error based on a cursory glance at the semantic orientation values produced and the resulting accuracy of the star ratings produced by previous boundaries with respect to the original author's star-rating. The results of this experiment can be found in Chapter 4.

3.2 Experimental Setup: PMI-Concordance

After discovering that the PMI-IR method was not producing great results, we began looking at offline methods of calculating semantic orientation values. We settled on somehow making use of a corpus in a similar problem domain as the one we were analyzing (*NLTK* `movie_reviews`) to calculate semantic orientation values for words, phrases, and sentences.

Before using the method outlined in the previous chapter for Corpus-Derived PMI score calculation, however, we tried using the concordance of words in the movie review corpus to calculate the semantic orientation of words, phrases, and, sentences in a small data set. By concordance, we mean the measure of the relative distance between two words in a document. We thought this metric might act as a good substitute for search engine hit counts for comparing how close a particular word is in meaning to a known, very positivel or negativel polarized word. This turned out to slightly better than the PMI-IR appraoch.

As we no longer had to rely on search engine hit counts for words to perform experiments, we were able to perform a slightly more rigorous experiment using the PMI-Concordance method. As in the PMI-IR experiment, we took the first 50 reviews from both the free application and paid application corpora and placed them into separate test sets. We then calculated the semantic orientation values for all the phrases in all the reviews of each test set. Finally, using the same unsophisticated method used for the PMI-IR experiment, we set the boundaries for the range of semantic orientation values corresponding to each star-rating. The results of this experiment can be found in Chapter 4.

3.3 Experimental Setup: PMI-Hierarchical Classification

Our best and final approach to generating star-ratings for mobile application reviews was tested the most rigorously. Instead of using small subsets of the free and paid application

review corpora as the test sets for this experiments, we used the full corpora themselves. This gives the results of the following two experiments significantly more weight than the previous ones.

To compare two slightly different methods of generating star-ratings for mobile application reviews, we performed two different experiments.

3.3.1 Without SVMs

In the first experiment, we set the boundaries for the range of semantic orientation values corresponding to each star-rating by the same crude method used in all the previous experiments. In the process of doing so, we were able to characterize a great many aspects of the corpora we were working with. The results we obtained from this experiment were quite promising and can be found in Chapter 4.

3.3.2 With SVMs

In the second experiment, the optimal values for maximizing star-generation accuracy were calculated by SVMs. This boosted the quality of the results significantly. All the constituent elements of the hierarchical classifier combine to produce an excellent star-rating generator. With more time and resources, this method could probably be made competitive. The results we obtained from this experiment were excellent and can be found at the end of Chapter 4.

Chapter 4

Results

4.1 Results: PMI-IR

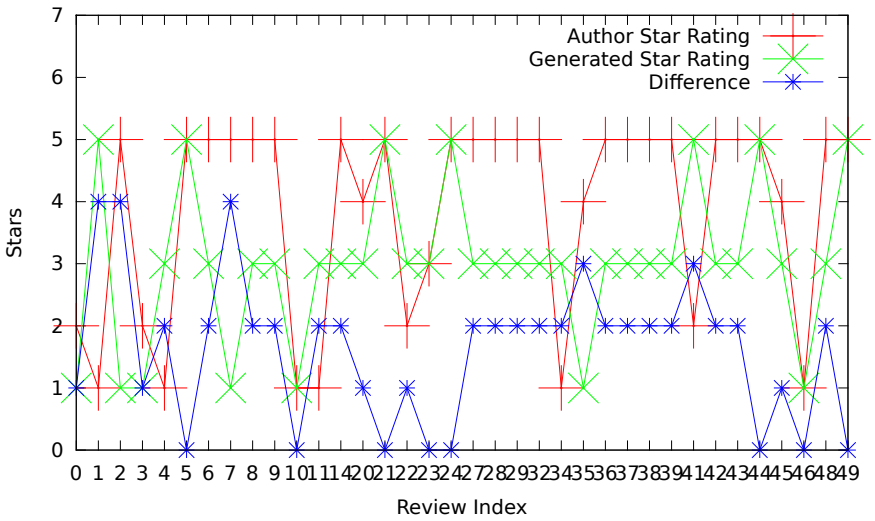

Figure 1: PMI-IR with free applications

The results of our first experiment in which we calculated the semantic orientations of phrases based on Google's search engine hit counts are shown above. As can be seen above, our ability at that point to reasonably match an application reviewer's star-rating was quite poor. The average absolute error was 2.04 stars. This meant that, if the reviewer had rated an application with a 3-star rating, we would, on average, generate a star-rating of 1-star or 5-stars. This is not good.

To make sure that the magnitude of the absolute error was not artificial, we went through

all the reviews for which our generated star-rating was off by 2 or more star-ratings. The only way in which the absolute error could have been artificial is if there were a significant number of reviews in our small test set of 50 reviews whose star-ratings did not match the sentiment of the review. Unfortunately, this was not the case, Our implementation of the PMI-IR method of generating sentiment scores was simply not good enough.

4.2 Results: PMI-Concordance

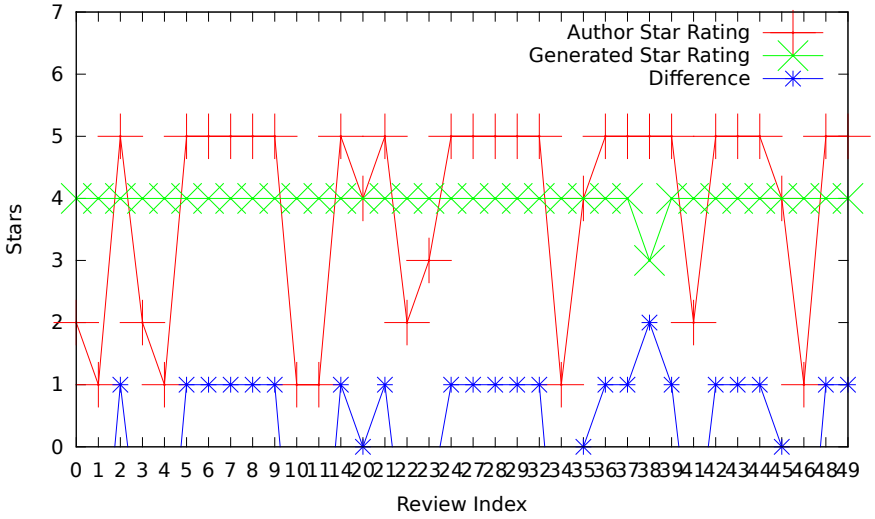

Figure 2: PMI-CD with free applications

The results of our second experiment in which we calculated the semantic orientations of phrases based on the concordance of words in *NLTK*'s `movie_reviews` corpus are shown above and below. Unlike the PMI-IR experiment, we were able to obtain results for 50 full reviews from both the free application review corpus and the paid application review corpus. Thus, the results are a bit more meaningful.

Compared to our previous experiment, we were able to match the application reviewer's star-rating much better. It was still nowhere near as good as we would have liked, especially given that the expanded data set we worked with had no incongruities between star-ratings and reviews as before. The average absolute error when generating a star-rating over the subset of free reviews was 1.39 stars. The average absolute error when generating a star-rating over the subset of paid reviews was 1.39 stars.

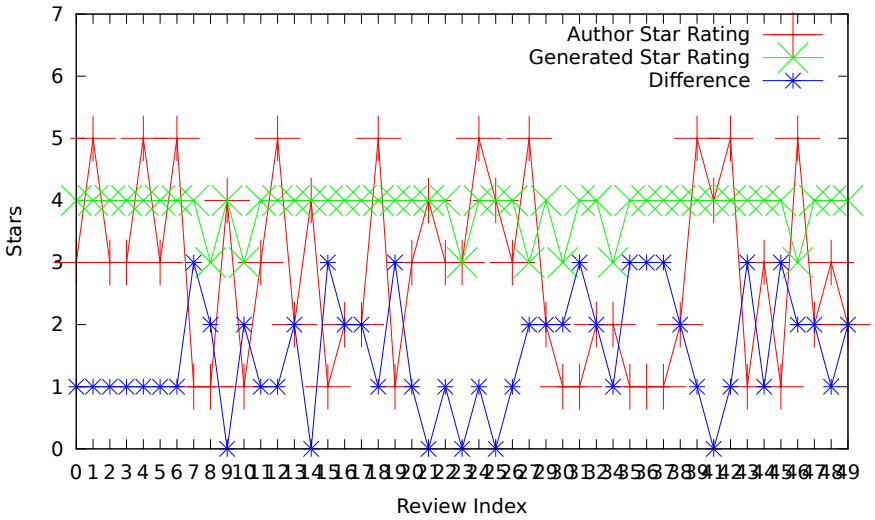

Figure 3: PMI-CD with paid applications

4.3 Results: PMI-Hierarchical Classification

4.3.1 Characterisation of Corpora

In our last set of experiments, we used the entirety of both review corpora to obtain our results. In the process of doing so we characterized the contents of the corpora for the sake of obtaining more accurate results. What follows is our characterization of the test set.

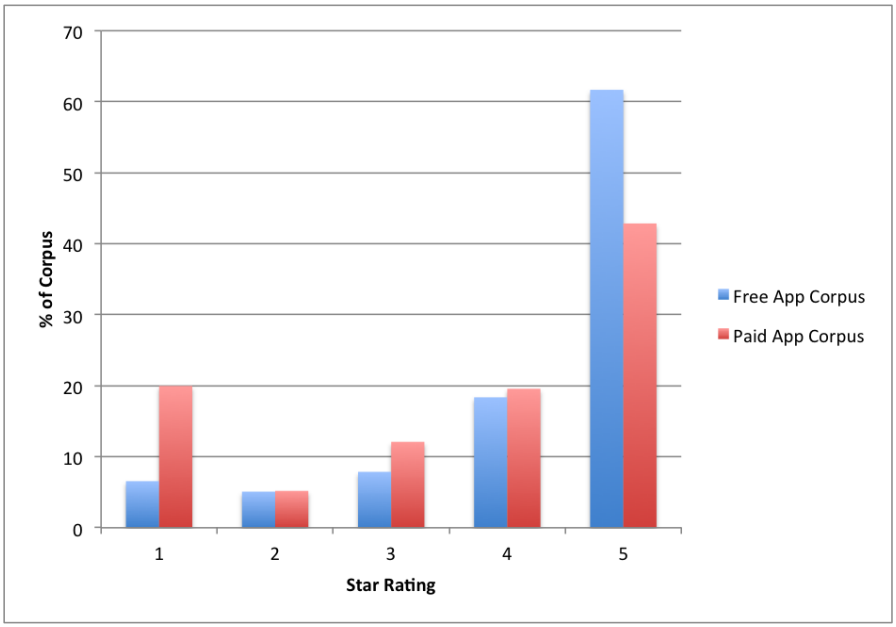

Figure 4: The distribution of star-ratings among reviews in each corpus

As shown above, our test set has a tremendous number of positive reviews. If we were to guess a star-rating between 3 and 5, we would be correct most of the time. It is interesting to note that reviewers seemed more willing to assign 5-star-ratings to free applications and more willing to assign 1-star ratings for paid applications. This may be indicative of how forgiving people are of the quality of an application based on its cost. As the total number of reviews from both corpora numbers only 2,969, it is not really possible to make sweeping sociological remarks based on the figure shown above.

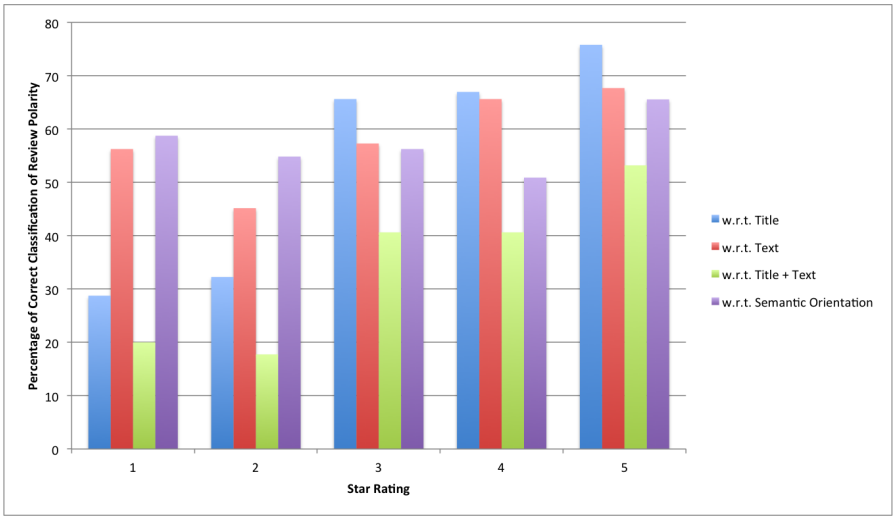

Figure 5: Accuracy with which different indicators produced correct review polarity from free app corpus.

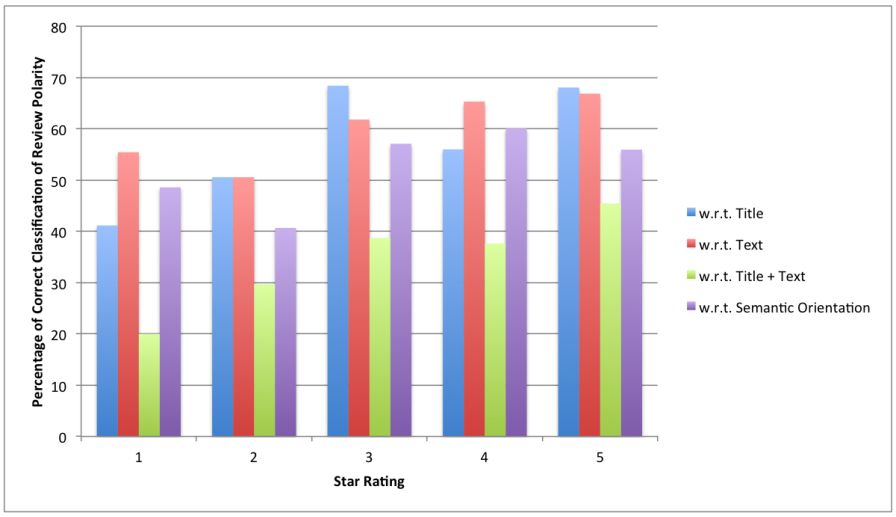

Figure 6: Accuracy with which different indicators produced correct review polarity from paid app corpus.

In order to accurately generate star-ratings for reviews, we had to be able to accurately determine the polarity of a review. To determine the best indicator of a review's polarity, we compared the star-rating assigned by the reviewer to four different indicators: the polarity of a review's title, the polarity of a review's text, the polarity of both the review's title and text if they were the same, and the calculated semantic orientation of the review's text. As shown above, the polarity of a review's text is the best indicator on average of a review's polarity. At first, we mistakenly believed that the title was a

better indicator of the overall polarity of a review, but, after collecting and analyzing the data, we came to the correct conclusion. The result was a much better star-rating generator.

Since our goal was not to model what star-ratings reviwers assign to mobile applications but to instead be able to generate star-ratings uniformly based on the sentiment of the text of a review, we decided to assess the accuracy of our star-rating generation a bit differently. To assess the accuracy of our hierarchical classifier, we first determined how well our generated star-ratings matched up to each review's original star-rating by star-rating. The accuracy for each star-rating was granted equal weight and averaged together to determine our overall accuracy. Then, we assessed the validity of this value by getting a sense of how well reviewers classified their own reviews into a star-rating.

4.3.2 Without SVMs

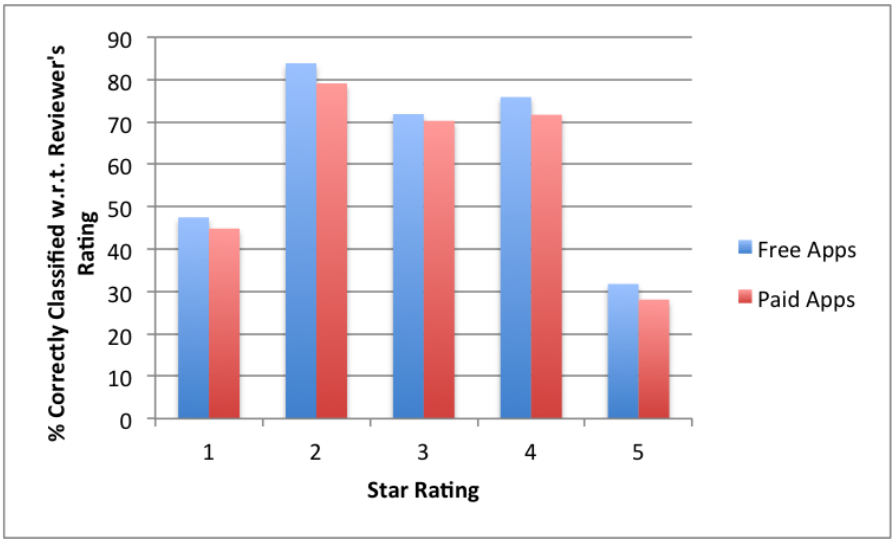

Figure 7: Accuracy of generated star-ratings w.r.t star-ratings of original reviewers.

The results shown above were obtained as described in Section 2.5.1. The polarity of each review was determined by the polarity of review's text and the threshold values for the star ratings were obtained via the same crude method used in previous experiments. The results above show a clear, vast improvement over our other attempts to generate star-ratings. Without using SVMs we acheived an accuracy of 46.6% for the free applications and 47.7% for the paid applications.

4.3.3 With SVMs

Below are the results for the free applications after training the thresholds using SVMs.

```
--------------------------------------------------------------------------------
Free Apps

----------------------------------------
Assuming we correctly classify the the positive reviews, if we use
semantic-orientation to calculate the exact star rating, we get
the following results:

Accuracy: 0.70 (+/- 0.00)

----------------------------------------
Assuming we correctly classify the the negative reviews, if we use
semantic-orientation to calculate the exact star rating, we get
the following results:

Accuracy: 0.58 (+/- 0.03)

Paid Apps
----------------------------------------
Assuming we correctly classify the the positive reviews, if we use
semantic-orientation to calculate the exact star rating, we get
the following results:

Accuracy: 0.57 (+/- 0.00)

----------------------------------------
Assuming we correctly classify the the negative reviews, if we use
semantic-orientation to calculate the exact star rating, we get
the following results:

Accuracy: 0.79 (+/- 0.00)

--------------------------------------------------------------------------------
```

Chapter 5

Analysis

5.1 PMI-IR

As far as we know, the PMI-IR method is a perfectly reasonable method for generating star-ratings. Unfortunately, the tiny amount of data we were able to obtain did not provide us with enough data to make a good model for generating star-ratings accurately. Previous work has shown that this method produces promising results [**?**]. This work was performed almost a decade ago, however, and search engines have become far more sensitive to automatic queries. To collect enough data to accurately assess and use the PMI-IR method would require either intimate access to a search engine's inner workings or enough time to manually obtain all the hit counts required from the browser interface to a search engine. Either way, this method was not a good fit for our project and it was good that we discovered this early.

5.2 PMI-Concordance

Talk about difficulties. (Why it didn't work. Why we tried it, etc.) We did better with this approach than the previous one but it still was not good enough.

5.3 PMI-CUE

Talk about why it works. Talk about things we observed from the data. (We don't have enough data to make general conclusions). Considerable improvement over both methods. Training the thresholds with SVMs made this even better.

Chapter 6

Conclusion

Appendix A

Datasets