

# Working Title Mobile Feedback Study

Kendall, Beatrice, Mei, Danny

## 1. OBTAINING THE CORPUS

**TODO:** Mei's android corpus

### 1.1 iOS Release Notes

We **TODO: word choice: downloaded** the release notes for the iOS applications in batches of 999 applications per day from the iTunes App Store . We chose to limit the number of applications downloaded due to concerns of overburdening the iTunes App Store . We found the iTunes App Store preferable to the App Store because the iTunes App Store saves the release notes for up to the previous 25 releases which allows us to develop an algorithm to automatically gather release notes on rotation so as not to trouble the iTunes App Store . The algorithm works in several steps.

The first step of the algorithm generates a list of application ids that will be downloaded for the day. Ids are stored in json format containing (1) the id number of the application, (2) the total number of reviews the application received last time its information was downloaded, (3) the date the application was last downloaded, and (4) the number of new reviews since the last time the app's information was downloaded. The algorithm chooses the top 999 id numbers with the earliest last downloaded date and generates a daily id file.

The next step retrieves all the html information from the iTunes App Store using the id numbers provided in the daily id file and stores this information in a file to be overwritten the next time the application with that id number is chosen. Between each subsequent call to the iTunes App Store , the algorithm waits 10 to 25 seconds.

After collecting the daily application information, the algorithm strips the html formatting from the html data and retrieves the json that contains the application metadata **TODO: ?**, this includes the release notes, the number of reviews, the genre of the application, the application name, etc.

The algorithm then extracts the release notes from each of the daily application and compares them with the existing release notes file. Each release is stored in json format containing (1) the version number, (2) the text of the release notes, and (3) the time and date of the release. Any new releases are added to the existing file and duplicate releases are discarded. If no file yet exists, all of the release note data from the daily application data is used to create one.

This step in the algorithm is also the "growing" stage. The iTunes App Store often recommends additional applications to customers on each page; the recommended applications are one or more of the following categories: (1) the top iPhone apps, (2) the top iPad apps, (3) other apps by the same developer, and (4) apps that customers who bought this application also bought. Our algorithm collects all of the recommended application ids and adds them to the id bank with an impossibly early last read date and no reviews to ensure the applications are read promptly and to maximize the reviews collected in the future.

The final step of the algorithm updates the complete list of ids by changing the last read date and the number of reviews. When updating the number of reviews the algorithm subtracts the previous number of reviews from the new number of reviews and stores this as the number of new reviews. This number is used in the review scraping algorithm so that the minimum amount of strain is put on the iTunes App Store .

Our mining algorithm was originally seeded with the top 50 iOS applications as determined by Distimo **TODO: cite** on December 30th 2014. Presently, the algorithm holds a total of 7000 application ids in its rotation.

### 1.2 iOS Reviews

The iTunes App Store does not store a significant number of reviews on each page accessed by the release notes algorithm so we deemed it necessary to create a second algorithm to accomplish this. Using the number of new reviews as determined by the release notes algorithm, we use a modified version of Kent Bye's script **TODO: cite** [] which scrapes a number of reviews for a single application from the iTunes App Store .

Reviews are stored on separate pages, with each page containing 10 reviews. The script calls down a certain number of pages and places the data from each review into json for-

mat containing (1) the title of the review, (2) the star rating given by the reviewer, (3) the username of the reviewer, (4) the application's version number being reviewed, (5) the date the review was given, (6) the body text of the review, (7) the percentage of people who found the review helpful, (8) the number of people who marked the review as helpful, (9) the total number of people who marked the review, (10) a link to the users profile, (11) the unique id number of the review, and (12) the country of the origin for the review.

Our algorithm scrapes a maximum backlog of 100 pages of reviews per application each run, which equates to up to 1000 reviews at one time. Each day the review mining algorithm is run until it reaches 14500 . Presently, the algorithm holds a has collected reviews for of 700 applications for a total of 7000 .

Due to the sheer number of applications and the limitations of compute power, we limited our initial analysis to 100 free\***TODO: Footnote - free does not mean without in game purchases** game applications from both the iTunes App Store and the Google Play store. The top applications from the iTunes App Store were taken from Distemo**TODO: cite** on February 2<sup>nd</sup>, 2015 and a complete list can be found at **TODO: url**. These applications were taken from the main mining rotation and were mined for both release notes and reviews daily.

### 1.3 Labeled Corpus

**TODO: Labeling the corpus, rules, and agreement percentage**

**Bugs:** The bugs category describes statements in release notes that correct flaws within the application. An example of a statement belonging in the bugs category is "Fixed a crash when closing documents while there was an active text insertion point."

**Enhancements:** The enhancements category describes statements in release notes that build on previously introduced features or improve performance, but do not correct overt flaws like those in the bugs category. An example of a statement belonging in the enhancement category is "Improved the performance of the Stroke and Fill layer styles."

**Features:** The features category describes statements in release notes that introduce new functionality or properties to an application. An example of a statement belonging in the enhancement category is "Added preference setting for prompting to save session information on close."

There is an additional miscellaneous category that we do not analyze that contains titles, headers, punctuation, other organizational type text, or duplications of release notes that are not considered in our analysis.

The classification of the statements was completed using qualitative thematic coding. Each of the classifications was done manually by the author and Hongyan Yi using the criteria detailed above. The authors took a random sample across the sample corpus of statements and both classified the same data to compare for agreement. The classifications were validated by reaching 79% agreement on 20% of sample











Emoticon	Emoji
:D	   
>:(	  
'--'	  

Table 1: A sample mapping of emoji to emoticons.









Emoticon	Unmatched Emoji
~:)	  
~:(	  
~:	 

Table 2: A sample mapping of unmatched emoji to emoticons. A '~' precedes each emoticon to indicate the emotion attached with that emoji is equivalent to a positive, neutral, or negative sentiment.

corpus - which is 734 release note statements.

## 1.4 Reviews

**TODO: bit about Meis data set**

### 1.4.1 Handling Reviews

Unlike release notes for applications, which generally avoid glaring misspelling and textual litter, user reviews are full of linguistic pitfalls that are not easy for computers to resolve. Though our methods are not able to account for sarcasm, we attempted to cleanse the data so that it retained the intent and the message of the reviewer, while being more properly constructed so the reviews could be analyzed. Below is steps of modification we took to adjust the reviews.

**Emoji:** Emoji are pictures used to convey emotions that can be used in electronic messages, much like emoticons on steroids. Originally from Japan, technology providers such as Microsoft, Apple, and Google added emoji as default characters sets to their mobile devices as their popularity grew and eventually standardized the Unicode codes for a set of common emoji in 2010.**TODO: cite - i got from wikipedia** As a result, emoji are common in reviews. Due to the platform-dependent meaning of emoji before standardization in October 2010, all emoji used in reviews before standardization are discarded.

Unfortunately, most sentiment analysis tools focus on emoticons **TODO: cite some papers and tools**, but very few recognize emoji. However, research has shown that emoticons contribute to more accurate sentiment analysis **TODO: cite** and we believe that emoticons and emoji serve similar purposes so we created a mapping between common emoji and emoticons to improve sentiment analysis.

**TODO: small example table with mapping** Our mapping attempted to match each emoji to a similar emoticon from SentiStrength's emoticon library**TODO: cite**. If the emoji was unable to be mapped to SentiStrength's emoticons, a new entry was added from Wikipedia's **TODO: cite** list of emoticons if the sentiment and appearance were similar. Finally, if an emoji did not match any widely recognized existing emoticon, three general categories were created to encapsulate "generally positive", "generally neutral", and "gener-

API	Time (s)	# Removed
No Language Checking*	7.06	65
Google Translate (via Goslate)	290.42	45
Langid	15.42	105
guess-language	26.765	327

Table 3: \*With no language checking the number of reviews removed include those with Unicode characters such as ... or pre-2010 emoji, which is why Google Translate appears to have retained extra reviews.

ally negative." Some examples of each category can be found in Table **TODO: ref.** In SentiStrength's original library there is no degree of emotion expressed by emoticons, only the values -1, 0, and 1. We followed this template and therefore do not distinguish different degrees that may appear to be present between 😞 and 😡, for example.

SentiStrength allows for multiple emotional associations between words and so most emoji tend to be either positive, negative, neutral, positive or neutral, or negative or neutral. There was one emoji that escaped this classification. The "face with look of triumph" emoji, 🏆, is used in both a positive and negative context. Unable to initially place this emoji, we looked through several tweets and found usage split between those using it to express a triumphant emotion and those using it to express an angry emotion (as a person with steam coming off of their head). Due to the extreme duality of the triumph/anger emoji, the authors decided that the textual context that this emoji would solely determine the sentiment and decided to classify its sentimental value as neutral.

Our emoji library can be found at **TODO: add url**

**TODO: include link list of emojis and how they rate in terms of positivity. This is ridiculous!!!!**

## 2. SENTIMENT SPELL CHECK

Research in sentiment analysis **TODO: cite sentistrength paper** shows that increased levels of sentiment can be related to repeated letters. The sentiment analysis tool SentiStrength operates under the assumption that a single repeated letter indicates a misspelling, but two or more repeated letters denote a intensification of the sentiment **TODO: cite.**

## 3. LANGUAGE CHECKING

## 4. CLASSIFIER - NMB AND LDA

### 4.1 Multinomial Naive Bayesian Classifier

#### 4.1.1 Preprocessing the Data

Before classifying any of the data we first normalize language to a more generic level so that a classifier can try to extract meaning. Our first step removes formatting noise that may make texts easier for humans to read but serves no purpose for a computer. Th type of linguistic noise our preprocessing focuses on removing is topic headers, distinct bullet points, different verb forms, and stop words. The following is the steps executed to prepare the data for classification.

If the data source is release notes, the noise related to section headers within release notes is removed because indicators

...  
BUG FIXES

-A crash that occured when an image is sent to  
Afterlight from another app has been fixed.  
-Other small bug fixes and improvements.  
-Fixed a small saving bug.  
-a small fix to a location data bug

...

Figure 1: A sample section in release notes. Notice the two different uses of the phrase "bug fixes": one as a divider and one as something shipped in the release.

may be removed later in the preprocessing steps. Figure 1 represents an example of a section header in release notes found in the application Afterlight. **TODO: The last two statements are from different release notes notes than the first two and the header, I have added them in to serve as a running example.** The statement "BUG FIXES" should not be considered for classification under the Bug label, while the "bug fixes" in "Other small bug fixes and improvements" is labeled as a Bug. Reviews do not usually follow the list format common in release notes, therefore we skip removing section headers when processing user reviews.

Automatic classifiers would potentially distinguish the words "fixed", "fixes", "Fixed", and "fix" as four different meanings, so the next few steps try to normalize language for the classifiers. After removing initial unrelated statements, all of the text is converted to lowercase and non-alphanumeric expunged. This gets rid of the distinction between "fixed" and "Fixed" as well as removing any extra bullets or symbols such as "►".

Then we stem the words using the Porter stemmer from the nltk. A stemmer transforms words from their conjugated or inflected forms to the root of the word. After transformation by the Porter stemmer, "fixed", "fixes", and "fix" are all reduced to "fix". Stemming the words allows the classifiers to make associations on the concepts conveyed by the words, not based on the tense or variation of the word.

The final step in preprocessing the corpus removes stop words. Stop words are words that are necessary for the mechanics of natural languages, but do not further understanding of abstract content. Stop words commonly include prepositions, articles, pronouns, and adverbs. We use a stop word list provided by Beatrice .

In addition to a general language stop word list, we also created a mobile stop word list. This list contains the names of all of the applications in our corpus (e.g. Facebook, TriviaCrack, Afterlight, etc.) . This prevents our training set from over-focusing on the classifier on applications associated with a label. For example, application "BuggyApp" has a lot of reviews or release notes that mention "bug" or "fix" in the form of "BuggyApp has too many bugs. Don't buy it!" or "fixed a bug that made BuggyApp slow". When generating a model with the training set, the classifiers will associate "BuggyApp" with the "bugfix" topic. When classifying another application not in the training set, "SuperApp-

	Precision	recall	f1-score
Bug	0.91	0.91	0.91
Enhancement	0.63	0.91	0.75
Feature	0.69	0.45	0.54
Miscellaneous	0.77	0.39	0.5

Table 4: Overall precision is 0.722

Bugs", the classifier will more weakly connect a bugfix and the "bugfix" topics because the data from "SuperAppBugs" does not contain references to "BuggyApp".

## 4.2 LDA

## 5.

## 6. ACKNOWLEDGMENTS

Pooria Emojis for latex <https://github.com/alecjacobson/coloremoji.sty>