AI6122 Text Data Management and Processing Assignment

Balraj Singh Bains
Nanyang Technological University
balraj001@e.ntu.edu.sg

Brandon Chua Shao Jie
Nanyang Technological University
bran0026@e.ntu.edu.sg

Ron Kow Kheng Hui
Nanyang Technological University
s190103@e.ntu.edu.sg

Samuel Samsudin Ng
Nanyang Technological University
sa0002@e.ntu.edu.sg

ABSTRACT

We analyzed the text data in user reviews of cell phones and accessories posted on Amazon over a 14-year period. In this paper, we present the results of our analysis, showing review distributions, ratings distributions, and review length distributions. We performed part-of-speech tagging on a sample of reviews to see how the `NLTK` tagger handles bad grammar in reviews. We also present a method of extracting keyphrases and sentences to summarize the reviews of a product. Lastly, we present a sentiment classification system using keyphrases as features and the Random Forest algorithm as the classifier.

KEYWORDS

text mining, product reviews, extractive summarization, sentiment classification

1 INTRODUCTION

In this paper, we present the results from our analysis of the product review dataset from UC San Diego (http://jmcauley.ucsd.edu/data/amazon). The dataset is a collection of 190,919 user reviews of cell phones and accessories, posted on Amazon from 2001 to 2014. We also present our development of two applications — an extractive summarizer and a sentiment classification system. Given a set of reviews of a particular product, our summarizer extracts keyphrases and sentences to summarize all the reviews. Our sentiment classification system classifies a review as positive or negative given just the review text (excluding any user rating). We used Python libraries such as `NLTK` and `pandas` for all pre-processing and analysis.

2 DATASET ANALYSIS

2.1 Data

In the dataset, each review has the following components:

- `reviewerID`, the ID of a reviewer.
- `asin`, Amazon standard identification number unique to each product.
- `reviewText`, the full review text by the reviewer.
- `overall`, the rating (1.0, 2.0, 3.0, 4.0, 5.0) of the product.
- `summary`, a short summary of the review.
- `unixReviewTime`, the Unix timestamp
- `reviewTime`, the timestamp in MM DD, YYYY format.

The data was provided as a JSON (JavaScript Object Notation) file. **Table 1** shows some statistics for the data.

Table 1: Dataset Statistics

Total number of reviews	190,919
Total number of unique products	10,420
Total number of unique reviewers	27,874
Range of years	2001–2014

2.2 Python Libraries

We pre-processed the data using Python's internal `JSON` [4] library and `pandas` [1] - a popular data analysis and manipulation library. The JSON data is converted into a `pandas.DataFrame` object — a data table. `pandas` uses the `Matplotlib`[5] library to plot results. For some analysis, we use `NumPy`[3] and `NLTK`[8] – a natural language processing library. **Table 2** shows an example of how the data is represented as a `pandas.DataFrame`.

2.3 Review and Rating Distributions

In the first part of our analysis, we:

- plotted the review distribution by number of products and by number of reviewers
- defined the criteria for categorizing reviewers into five activity levels — most active, active, average, not very active, not active — and categorized the reviewers into these five groups
- plotted the rating distribution for each group of reviewers
- investigated whether active reviewers are more or less likely to give low ratings compared to less active reviewers

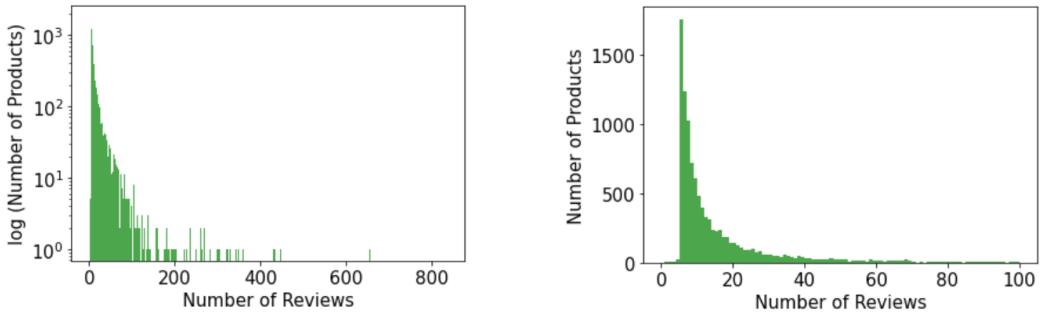
2.3.1 Review Distribution by Number of Products. We analyzed the review distribution by number of products, i.e. how many products have x number of reviews. We present the results as follows:

- Summary statistics. (**Table 3**)
- Histogram (in log scale) showing the review distribution for the entire range of review counts (**Figure 1a**)
- Histogram (in linear scale) showing the review distribution for 0 to 100 reviews (**Figure 1b**)

Most of the products have less than 50 reviews.

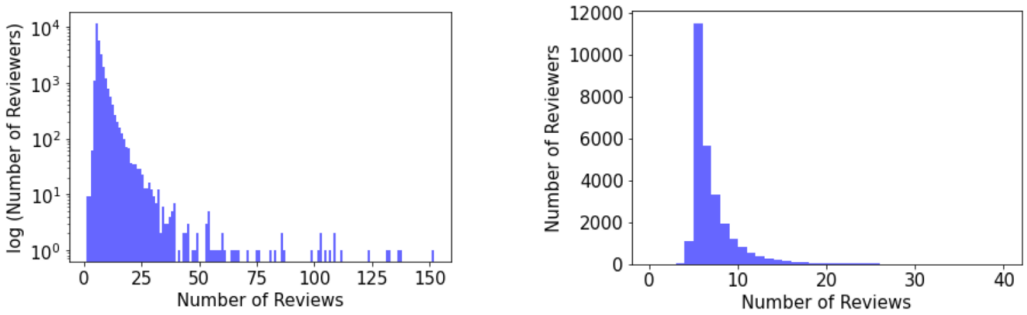
Table 2: A Review as a `pandas.DataFrame`

	reviewerID	asin	reviewText	overall	summary	unixReviewTime	reviewTime
0	ASY55RVNIL0UD	120401325X	These stickers work...	5.0	Really great product.	1389657600	01 14, 2014



(a) 0 to 836 Reviews (log Scale) **(b) 0 to 100 Reviews (Linear Scale)**

Figure 1: Review Distribution (by Number of Products)



(a) 0 to 152 Reviews (log Scale) **(b) 0 to 40 Reviews (Linear Scale)**

Figure 2: Review Distribution (by Number of Reviewers)

Table 3: Number of Reviews Per Product

Statistic	Reviews Per Product
Maximum	836
Minimum	1
Mean	18.32
Median	9.0

Table 4: Number of Reviews Per Reviewer

Statistic	Reviews Per Reviewer
Maximum	152
Minimum	1
Mean	6.85
Median	6.0

2.3.2 Review Distribution by Number of Reviewers. We analyzed the review distribution by number of reviewers, i.e. how many reviewers posted x number of reviews. We present the results as follows:

- Summary statistics. (**Table 4**)
- Histogram (in log scale) showing the review distribution for the entire range of review counts. (**Figure 2a**)
- Histogram (in linear scale) showing the review distribution for 0 to 40 reviews. (**Figure 2b**)

Most of the reviewers posted less than 20 reviews.

Table 5: Number of Reviews Per Year

Year	Total Reviews
2001	1
2002	1
2003	2
2004	44
2005	129
2006	240
2007	389
2008	676
2009	1,304
2010	3,414
2011	9,516
2012	29,612
2013	91,261
2014	54,330

2.3.3 Categorizing Reviewers by Activity Levels. We categorized all the reviewers into five groups — *most active*, *active*, *average*, *not very active*, and *not active*. We simply consider the average number of reviews posted by a reviewer over 10 years. We choose a 10-year time frame (2005 to 2014) because there were very few reviews from 2001 to 2004. We disregard the fact that some reviewers could have started posting reviews a number of years after 2005; others could have stopped posting after a few years.

We present the results as follows:

- The number of reviews in each year. (**Table 5**)
- Classification criteria. (**Table 6**)
- Rating distribution showing, in each group, the number of reviews for each rating and the proportion of all reviews. (**Table 7**)
- Bar plots showing the rating distributions. (**Figure 3**)

From **Table 7** and **Figure 3**, we see that the percentages of low ratings (1.0 or 2.0) are much smaller for the *most active* and *active* reviewers (1.4% and 5.7% respectively) compared to the corresponding percentages for the other three groups (9.3%, 11.2%, 13.4%). We conclude that active reviewers are less likely to give low ratings compared to less active reviewers. One possible reason for this could be that active reviewers tend to shop more online and are more informed about good deals and sellers, and hence resulting in more positive reviews.

2.4 Sentence Segmentation

We performed sentence segmentation on each review using `NLTK`. We present the results as follows:

- Summary statistics. (**Table 8**)
- Histogram (in log scale) showing the review length distribution for the entire range of sentence counts. (**Figure 4a**)
- Histogram (in linear scale) showing the review length distribution for 0 to 50 sentences (**Figure 4b**)

There are reviews with no text (minimum number of sentences = 0 from **Table 8**). Most of the reviews have less than 20 sentences.

2.5 Tokenization and Stemming

We performed tokenization on each review and counted the unique whole tokens in each review. Then we stemmed the tokens and counted the unique stemmed tokens in each review. We converted all text to lower case before tokenization and only included alphabetic tokens. Stemming was done using `NLTK`'s `PorterStemmer` class which is an implementation of Martin Porter's stemming algorithm [9]. We present the results as follows:

- Summary statistics. (**Table 9**)
- Histogram (in log scale) showing the review length distribution for the entire range of whole tokens. (**Figure 5a**)
- Histogram (in linear scale) showing the review length distribution for 0 to 400 whole tokens. (**Figure 5b**)
- Histogram (in log scale) showing the review length distribution for the entire range of stemmed tokens. (**Figure 5c**)
- Histogram (in linear scale) showing the review length distribution for 0 to 400 stemmed tokens. (**Figure 5d**)

Stemming reduced the token counts by 1 to 2 tokens per review on average (from Table **9**). The shapes of the histograms are similar, and most of the reviews have less than 200 unique tokens.

2.6 POS Tagging

In the last part of our data analysis, we performed part-of-speech (POS) tagging on a random sample of five sentences. We used `NLTK`'s POS tagger, which uses the *Penn Treebank* tagset [10]. In online reviews, wrong grammar and punctuation are very common. There are also many misspelled words and informal lingo which will become out-of-vocabulary words for the tagger. We wanted to find out how the tagger handles these cases.

The five selected sentences are the first sentences from five randomly sampled reviews. They are:

(1) `This case protects this phone from MANY a fall I've had.`
(2) `The real thing..and did data and power.`
(3) `great is's gentle on my ipod and the price was a steal and i wouldbuy it again and again`
(4) `I usually have to buy multiple stylus pens because with a family of 4, they get lost, broken, and worn down.`
(5) `Words can NOT express how powerful this phone is.`

(1), (4) and (5) have decent grammar. The only issue is the word `'can NOT'` in (5). The other sentences have bad grammar and punctuation. **Listing 1** shows the tagging results.

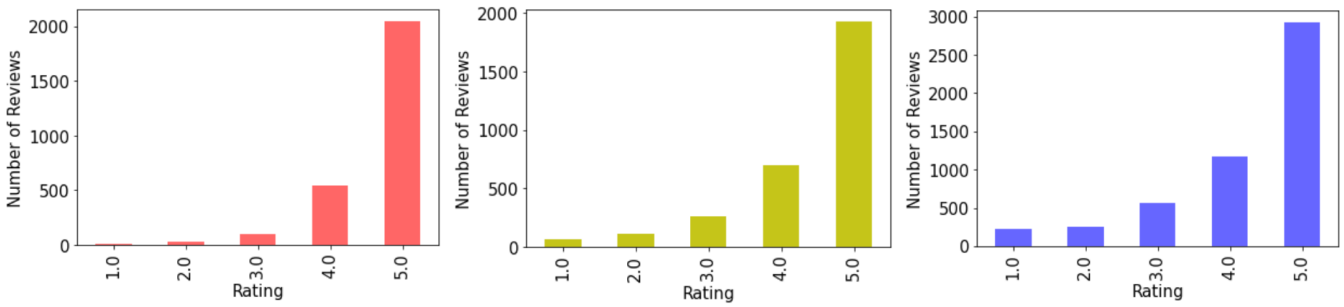
After tokenization, the problematic tokens are `'thing..and'` and `'wouldbuy'` in sentences 2 and 3 respectively. `'thing..and'` was tagged as NN (singular noun), which is the correct tag for `'thing'`. `'wouldbuy'` was tagged as VBP (verb, non-3rd person singular, present), probably because the token comes after `'I'`. In sentences 1 and 5, `'NOT'` and `'MANY'` were mistaken for acronyms and tagged as NNP (proper noun). In sentence 3, `'gentle'` was mistaken for a noun because the preceding token is `"'s"`, a possessive ending. Lastly, `'I'` was tagged as NN instead of PRP (personal pronoun) in sentence 3. **Table 10** shows the correct tags for these words.

Table 6: Classification of Reviewers

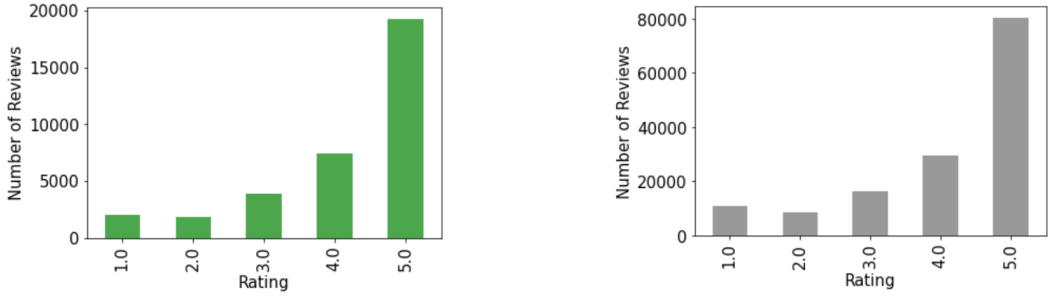
Activity Level	Reviews Per Year	Reviews in 10 Years	Total Reviewers	Total Reviews
Most active	> 6	61–152	28	2,733
Active	3–6	31–60	76	3,072
Average	2–3	21–30	212	5,151
Not very active	1–2	10–20	2,773	34,420
Not active	< 1	1–9	24,785	145,543

Table 7: Rating Distributions

	Number of Reviews (% of Total Within Group)				
Activity Level	Rating 1.0	Rating 2.0	Rating 3.0	Rating 4.0	Rating 5.0
Most active	11 (0.4)	28 (1.0)	100 (3.6)	546 (20.0)	2,048 (74.9)
Active	62 (2.0)	114 (3.7)	265 (8.6)	700 (22.8)	1,931 (62.9)
Average	221 (4.3)	260 (5.0)	571 (11.1)	1171 (22.7)	2,928 (56.8)
Not very active	2,024 (5.9)	1,814 (5.3)	3,860 (11.2)	7,450 (21.6)	19,272 (56.0)
Not active	10,760 (7.4)	8,673 (6.0)	16,279 (11.2)	29,437 (20.2)	80,394 (55.2)



(a) Most Active Reviewers **(b) Active Reviewers** **(c) Average Reviewers**

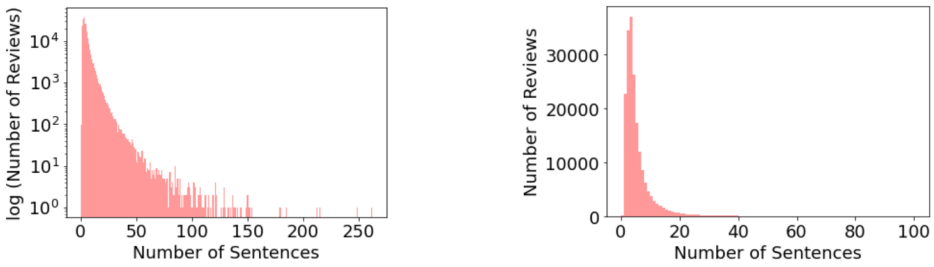


(d) Not Very Active Reviewers **(e) Not Active Reviewers**

Figure 3: Rating Distribution for the Five Groups of Reviewers

Table 8: Number of Sentences Per Review

Statistic	Sentences Per Review
Maximum	262
Minimum	0
Mean	5.16
Median	4.0



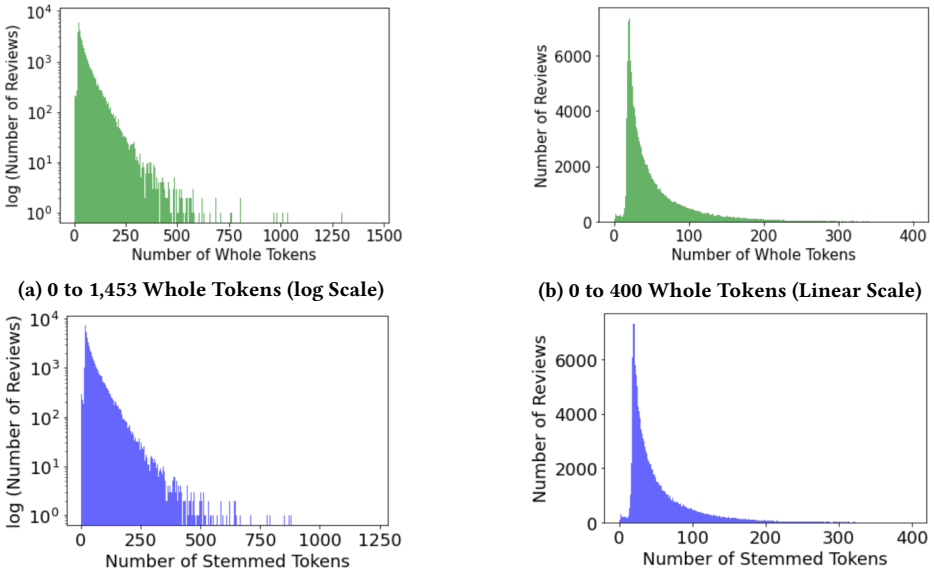
(a) 0 to 262 Sentences (log Scale)

(b) 0 to 50 Sentences (Linear Scale)

Figure 4: Review Length Distribution (by Number of Sentences)

Table 9: Number of Unique Tokens Per Review

Statistic	Whole Tokens Per Review	Stemmed Tokens Per Review
Maximum	1,453	1,224
Minimum	0	0
Mean	55.65	53.87
Median	37.0	36.0



(a) 0 to 1,453 Whole Tokens (log Scale)

(b) 0 to 400 Whole Tokens (Linear Scale)

(c) 0 to 1,224 Stemmed Tokens (log Scale)

(d) 0 to 400 Stemmed Tokens (Linear Scale)

Figure 5: Review Length Distribution (by Number of Whole and Stemmed Tokens)

Listing 1: Tagging Results for Five Random Sentences

```
[('This', 'DT'), ('case', 'NN'), ('protects', 'VBZ'), ('this',
     'DT'), ('phone', 'NN'), ('from', 'IN'), ('MANY', 'NNP'),
     ('a', 'DT'), ('fall', 'NN'), ('I', 'PRP'), ("'ve", 'VBP'),
     ('had', 'VBN'), ('.', '.')]

[('The', 'DT'), ('real', 'JJ'), ('thing..and', 'NN'), ('did',
     'VBD'), ('data', 'NNS'), ('and', 'CC'), ('power', 'NN'),
     ('.', '.')]

[('great', 'JJ'), ('is', 'VBZ'), ("'s", 'POS'), ('gentle', 'NN'),
     ('on', 'IN'), ('my', 'PRP\$'), ('ipod', 'NN'), ('and',
     'CC'), ('the', 'DT'), ('price', 'NN'), ('was', 'VBD'), ('a',
     'DT'), ('steal', 'NN'), ('and', 'CC'), ('i', 'NN'),
     ('wouldbuy', 'VBP'), ('it', 'PRP'), ('again', 'RB'), ('and',
     'CC'), ('again', 'RB')]

[('I', 'PRP'), ('usually', 'RB'), ('have', 'VBP'), ('to', 'TO'),
     ('buy', 'VB'), ('multiple', 'JJ'), ('stylus', 'NN'),
     ('pens', 'NNS'), ('because', 'IN'), ('with', 'IN'), ('a',
     'DT'), ('family', 'NN'), ('of', 'IN'), ('4', 'CD'), (',',
     ','), ('they', 'PRP'), ('get', 'VBP'), ('lost', 'VBN'),
     (',', ','), ('broken', 'VBN'), (',', ','), ('and', 'CC'),
     ('worn', 'VBD'), ('down', 'RB'), ('.', '.')]

[('Words', 'NNS'), ('can', 'MD'), ('NOT', 'NNP'), ('express',
     'VB'), ('how', 'WRB'), ('powerful', 'JJ'), ('this', 'DT'),
     ('phone', 'NN'), ('is', 'VBZ'), ('.', '.')]
```

Table 10: Wrongly Tagged Tokens

Sentence	Token	Given Tag	Correct Tag
1	'MANY'	NNP	JJ
3	'gentle'	NN	JJ
3	'I'	NN	PRP
5	'NOT'	NNP	RB

3 REVIEW SUMMARIZER

3.1 Objectives

We set out to develop an automatic extractive summarizer to achieve two goals. First, the summary should inform us about the product, including the brand, its features, and the quality of the product as rated by the reviewers. For instance, if the product is a headphone, we would like to know if it is a wired or bluetooth headphone. We would also like to know what the reviews say about its sound quality. Second, the summary should inform us about the general sentiments of the reviews.

3.2 Previous Work

To fulfill the two goals, the system needs to be able to recognize named entities (brand and model of the product), words describing features and quality, and sentiment words. Although we have a large dataset in which every record contains a full review and a summary of the review, we will use unsupervised methods to develop a summarizer. Among unsupervised methods, graph-based methods originating from the TextRank algorithm [6] have been well-studied [7]. We initially experimented with TextRank to score sentences. We used Glove word vectors to represent sentences as vectors, and then created a matrix of cosine similarities between sentences before applying the TextRank algorithm.

However we are more interested in methods based on term frequencies (`tf`) and inverse document frequencies (`idf`), so that we have full control over how we score candidate phrases and sentences. The method which we present in this paper was inspired by the keyphrase extraction system YAKE! [2] and the method proposed by [11].

3.3 Methodology

It is difficult to achieve our goals with a summary consisting of only keyphrases or only sentences. Keyphrases alone cannot convey the general sentiment of a set of reviews. A handful of sentences may not capture all the important words relevant to the product. Thus our summarizer extracts a set of keyphrases (unigrams, bigrams and trigrams) *and* a set of sentences. We describe our keyphrase extraction method as follows:

(1) **Prepare the data.** For the summarizer, we need the product ID, the number of reviews for each product, and the complete review text.
(2) **Prepare the stopword list.** We add more words to `NLTK`'s stopword list, by including contractions that were left out of the list, such as `i'm`. We also expect some reviewers to omit the apostrophe. So we included contractions without the apostrophe, such as `isnt`. We also add informal lingo such as *lol*. Lastly, we remove the words `not` and `very` from `NLTK`'s list as these convey sentiments.
(3) **Prepare a punctuation list for punctuation removal.** We exclude `-` (hyphen), `$` and `%` from our punctuation list. We include hyphens in tokens because we do not wish to break up hyphenated words. `$` and `%` are included because they convey information which may be relevant to a review.
(4) **Design two different tokenization functions.** `text_preprocess_clean()` replaces all the punctuation in the punctuation list with a space, performs case-folding, removes numbers, and tokenize the text, excluding stopwords. `text_preprocess()` performs case-folding and tokenizes the text, including stopwords and punctuation. Spaces are added to both sides of a period so that text such as `xxx.yyy` are not regarded as a single token `'xxx.yyy'`. In both functions, we replace the plural forms of all cell phones-related words to their singular forms. For example, we replace `cases` with `case` and `batteries` with `battery`. These are very common words in reviews and we want to ensure that the singular and plural forms are not counted separately. Note that we do not stem or lemmatize the text as we want to preserve the complete spelling of all words.
(5) **Calculate `idf`.** For all words (excluding stopwords) in the corpus of reviews, we create a dictionary which maps each word to its `idf` value, defined as:
`idf = log10[(document count)/(document frequency)]`,

where a document is defined as the set of all reviews for a product and document frequency is the number of documents containing the word. That is, the document count is the number of unique products.

(6) **Calculate `tf` and `tf-idf`.** For a particular product, we tokenize (excluding stopwords) all its reviews. Then we create two dictionaries. The first dictionary maps each token in the set of tokens to its `tf` value, defined as:

$$\texttt{tf} = 1 + \texttt{log10(token frequency)},$$

where token frequency is the number of times the token occurs in the product's reviews. The second dictionary maps each token in the set of tokens to its `tf-idf` value, by multiplying `tf` and `idf`:

$$\texttt{tf-idf} = \texttt{tf} \times \texttt{idf},$$

(7) **Select candidate unigrams, bigrams and trigrams.** We create three separate sets of candidates, for unigrams, bigrams and trigrams. For a particular product, we first tokenize all its reviews into unigrams, bigrams and trigrams. Unigram tokens exclude stopwords. For bigrams and trigrams tokens, we include stopwords and punctuation during tokenization and concatenate contiguous tokens to form separate sets of bigrams and trigrams. From the set of bigrams (or trigrams), we select bigrams (or trigrams) which do not contain any stopwords or punctuation as candidates. The following example shows this process:
REVIEW TEXT:
`Cool phone! Love its large screen and colors.`
TOKENIZATION:
`['cool', 'phone', '!', 'love', 'its', 'large', 'screen', 'and', 'colors', '.']`
BIGRAMS:
`[('cool', 'phone'), ('phone', '!'), ('!', 'love'), ('love', 'its'), ('its', 'large'), ('large', 'screen'), ('screen', 'and'), ('and', 'colors'), ('colors', '.')]`
CANDIDATE BIGRAMS:
`[('cool', 'phone'), ('large', 'screen')]`
If we had excluded stopwords and punctuation during tokenization, we would get more candidate bigrams, including nonsensical bigrams `('phone', 'love')` and `('love', 'large')`:
TOKENIZATION:
`['cool', 'phone', 'love', 'large', 'screen', 'colors']`
CANDIDATE BIGRAMS:
`[('cool', 'phone'), ('phone', 'love'), ('love', 'large'), ('large', 'screen'), ('screen', 'colors')]`

(8) **Select final candidates by parts-of-speech.** We POS-tag the sets of candidate unigrams, bigrams and trigrams and use the tags to select final candidates. For unigrams, we only select nouns (tags NN, NNS, NNP, NNPS) as final candidates. For bigrams, we select those with these POS tag patterns: all nouns (NN, NNS, NNP, NNPS) (e.g. *battery charger*), adjective (JJ, JJR, JJS, CD) followed by noun (e.g. *great phone*), and adverb (RB, RBR, RBS) followed by adjective (e.g. *not great*). For trigrams, we select those with these POS tag patterns: all nouns (e.g. *car phone holder*), adjective followed by two nouns (e.g. *great battery charger*), and adverb followed by adjective followed by noun (e.g. *very good charger*).
Specific phone brands and models are useful information in a summary. Therefore, we include as candidates those phrases for which the first word (for bigrams and trigrams) or second word (for trigrams) is a popular phone brand. For such phrases, it is possible that the second or third word in the phrase is the model name or number. We create a list of popular phone brands (such as `apple` and `samsung`) and models (such as `iphone` and `galaxy`) to match with the word.

(9) **Score the candidates.** The score for each candidate consists of two components — the token frequency `tf` and the review frequency `rf` which we define as follows:
Token frequency `tf`: `tf` is the number of times the token occurs in the set of all reviews of the product. This value is stored in the `tf` dictionary created in Step (6). For unigrams, the score is simply the `tf` value. For bigrams and trigrams, we add the `tf` values of the component tokens to get the `tf` score of the bigram or trigram. This will no doubt give bigrams and trigrams a higher score than unigrams, which is fair because we believe bigrams and trigrams convey more information. However, we will see later that `rf` offsets this advantage.
Review frequency `rf`: `rf` is the number of reviews of the product containing the unigram, bigram or trigram. We calculate this score using two different formulas, one for unigrams and bigrams, and one for trigrams. For a unigram or bigram `x`, the formula is:

$$\texttt{rf} = \texttt{RF\_WEIGHT} \times \texttt{log10(rf[x])} + \texttt{rf[x]/n},$$

and for a trigram `x`, the formula is:

$$\texttt{rf} = \texttt{log10(rf[x])} + \texttt{rf[x]/n},$$

where RF_WEIGHT is a parameter which allows us to adjust the influence of unigrams and bigrams, `rf[x]` is the dictionary mapping a unigram, bigram or trigram (`x`) to its `rf` value, and `n` is the number of reviews the product has. The justification for this formula is as follows: The `log` term is 0 if `x` occurs in only one review. However the value of `rf` depends on how many reviews `n` a product has. If there are very few reviews, a small `rf` value does not mean that `x` is not important. Therefore we add `rf[x]/n` to give more weight to small values of `rf` and `n`. From our experiments, we observe that `RF_WEIGHT = 2` is a suitable weight to prevent trigrams from always outscoring unigrams and bigrams. Note that unigrams will tend to have much larger `rf` values than bigrams and trigrams. So this score offsets the higher `tf` scores that bigrams and trigrams will have.

(10) **Calculate the final score and rank the candidates.** The final score is the sum of the `tf` and `rf` scores. We rank the candidates and select the top `k` candidates based on the following formula:

$$\texttt{k} = \texttt{ceil(SUMMARY\_SIZE\_FACTOR} \times \texttt{log10(total number of words)},$$

where `total number of words` is the total word count in all the reviews of the product and `ceil` is the ceiling function. We set `SUMMARY_SIZE_FACTOR` = 5, which will give k = 20 if the word count is 10,000.

(11) **Check distance between candidates.** As we did not reduce our tokens to root forms, we include this step to filter out candidates that are very similar (such as singular and plural forms). Using the Levenshtein distance function from the `jellyfish` library, we use the following formula to calcuate the distance between two terms `t1` and `t2`:

```
distance = 1 - jellyfish.levenshtein(t1,t2) /
               max(len(t1),len(t2))
```

For any two candidates in the ranked shortlist, if this distance is >= 8.0, we remove the shorter candidate from the shortlist. The summarizer will finally output the candidate phrases with the highest final scores, in alphabetical order.

The second part of our summarization process is sentence extraction. Our goal is to extract sentences which convey the most information, by selecting more expressive comments containing richer words instead of comments with common words like `'good phone'` or `'i like this product'`. To achieve this, we will use `tf-idf` to score every sentence, and then rank and select the top-scoring sentences. We describe the sentence extraction method as follows:

(1) **Sentence tokenize all the reviews of the product.** We use `NLTK`'s sentence tokenizer.

(2) **Create lists of unigrams, bigrams and trigrams in each sentence.** For each sentence, we tokenize it (excluding stopwords and punctuation) and form sets of unigrams, bigrams and trigrams.

(3) **Calculate `tf-idf` for each unigram, bigram and trigram in each sentence.** For bigrams and trigrams, we calculate the `tf-idf` score the same way we calculated the `tf` score — by adding the `tf-idf` values of the component tokens.

(4) **Calculate the final score for each sentence and rank the sentences.** The final score of a sentence is calculated using the following formula:

```
score = (sum of tf-idf scores)/(length of
                sentence),
```

where the sum of `tf-idf` scores is the sum of the scores of all unigrams, bigrams and trigrams in the sentence, and the length of sentence is the number of words in the sentence. Normalization ensures that longer sentences do not outscore shorter sentences. In our system, we allow the user to select the number of sentences to extract. The summarizer will output the sentences with the highest final scores.

3.4 Discussion: `tf` vs `tf-idf`

We use `tf` instead of `tf-idf` to score the keyphrases. `tf-idf` favours words which occur frequently in the product's reviews but rarely in other products' reviews. This does not help in achieving our goal of extracting phrases which convey the most information about the product. Such phrases will always be the ones that occur most frequently in the product's reviews, regardless of their occurrences in the rest of the corpus. Our experiments show that using `tf-idf` gives undesirable output, such as gibberish and badly misspelled words, foreign language words, and rare and irrelevant words (e.g. *dunkin donuts*) which convey no information about the product. Using `tf` to score phrases give much better results.

Using `tf-idf` to score sentences results in more original and informative comments which will be of interest to users. As the score is normalized, one rare and irrelevant word in a sentence is unlikely have too much influence on the final score.

3.5 Limitations of our Design

There are other features which could help in scoring keyphrases, such as word casing (upper case words more likely to be named entities) and word position (words mentioned at the start of a document could be more important). We did not have time to explore these. Due to our POS filtering, our system tends to extract mostly noun phrases and adjective-noun phrases. Extracting sentiment words is a harder task which we have not explored in great detail (we present a basic extraction of sentiment words in Section 4). Designing a state-of-the-art extractive summarizer is a challenge which we hope to achieve in our future research.

3.6 Evaluation

We briefly describe two ways of evaluating the summary.

(1) **Matching machine results with human results.** This is a common evaluation method used in NLP tasks. If the machine results are comparable to (or better than) human results, we can consider the problem solved. To evaluate extractive summaries, we can find the *precision* and *recall* (which are based on the number of common words between machine and human summary) and then calculate the F1 measure, defined by:

$$\mathtt{F1} = \frac{2 \times precision \times recall}{precision + recall}$$

(2) **Using a human to grade the machine results.** A more subjective method would be to appoint a human to grade the machine summary, or to compare it with summaries from other systems. For our summarizer, we used this method to informally evaluate our results.

3.7 Results

In **Appendix A**, we present the summaries produced for three different products.

4 SENTIMENT CLASSIFICATION

We implemented a sentiment classification system to classify a review as positive or negative, using only keyphrases from the review as features. We experimented with two classification algorithms: Gaussian Naive Bayes and Random Forest.

We present our method in the following sections. The document pre-processing pipeline is described in Section 4.1. Feature extraction, involving scoring and selecting top-scoring sentiment words, is described in Section 4.2. Finally, sentiment classification is performed and the results are presented in Section 4.3.

4.1 Text Pre-processing

The text pre-processing pipeline consists of the following steps:

- Case-folding
- Removing numbers
- Removing punctuation
- Removing leading and trailing spaces
- Tokenization
- Lemmatization
- Removing stop words

We used `NLTK`'s English stopwords list, but we handcrafted it to remove descriptive product review terms such as:

- `not`, e.g. not good
- `wouldn't`, e.g. wouldn't recommend
- `didn't`, e.g. didn't like

4.2 Sentiment Word Extraction

Unigrams might not hold sufficiently meaningful information. We assume that bigrams and trigrams convey more information. We calculate bigram and trigram frequencies in the corpus using modules from the `scikit-learn` library.

After pre-processing and n-gram counting, the top 20 unigrams and bigrams/trigrams in the corpus by term frequency are shown in **Figure 6**. As we can see among the top 20 n-grams, a number of bigrams convey meaningful sentiment information, such as *highly recommend*, *work well* and *really like*.

Using the bigram and trigram frequencies obtained in the previous step, we calculate the `tf-idf` scores of these n-grams in the review for which we want to classify. The `tf-idf` scores are then sorted in descending order to obtained a rank list of sentiment words. Our application allows a user to enter a particular review index (from 0 to 190,918) and the number of phrases required.

The output of the top 5 phrases for the review of index 777 is presented in **Listing 2**.

Listing 2: Sentiment Word Output for a Sample Review

```
GETTING 5 TOP KEYWORDS FOR REVIEW #777
PRODUCT ID: B000652QNS
REVIEWER ID: A40P9CE7RCMUK

REVIEW TEXT:
   Just recived my screen protectors and there amazing.
        They came packaged very well. They come with a very
        thivk cleaning cloth and some wet and dry cleaning
        wipes. Once I put on protector on it was really
        clear no problems at all great feeling to the
        protector it was cut really well. Will defenetly
        buy more when I need them wont be using another
        protector but boxwave.. very satisfied (:

PROCESSED REVIEW TEXT:
   recived screen protector amazing came packaged very well
        come very thivk cleaning cloth wet dry cleaning
        wipe put protector really clear problem great
        feeling protector cut really well defenetly buy
        need wont using another protector boxwave very
        satisfied

TOP KEYWORDS AND TFIDF SCORES:
   1.                    come very (0.505)
   2.               cleaning cloth (0.469)
   3.               very satisfied (0.443)
   4.                  really well (0.418)
   5.                    very well (0.304)
```

4.3 Sentiment Classification Results

For the classification task, we first map the ratings into three classes:

- Rating 1.0 to 2.0 : Negative class
- Rating 3.0 : Neutral class
- Rating 4.0 - 5.0 : Positive class

The distribution of the sentiment classes from the review dataset is presented in **Figure 7**. Most of the ratings are in the positive class. To train and test the model, we split the dataset into training and test sets in the ratio 80:20. We used `tf-idf` to rank and select the top 2,000 unigrams and bigrams for training. We experimented with two classifiers: Gaussian Naive Bayes and Random Forest. **Listing 3** and **Listing 4** show the results.

Listing 3: Results for Gaussian Naive Bayes Classifier

```
Confusion Matrix:
[[ 3316 1179  401]
 [ 1436 1957  805]
 [ 4011 6627 18452]]

Result metrics:
              precision   recall f1-score  support

    Negative      0.38     0.68     0.49     4896
     Neutral      0.20     0.47     0.28     4198
    Positive      0.94     0.63     0.76    29090

    accuracy                        0.62    38184
   macro avg      0.51     0.59     0.51    38184
weighted avg      0.79     0.62     0.67    38184

Accuracy score: 0.6213335428451707
```

Listing 4: Results for Random Forest Classifier

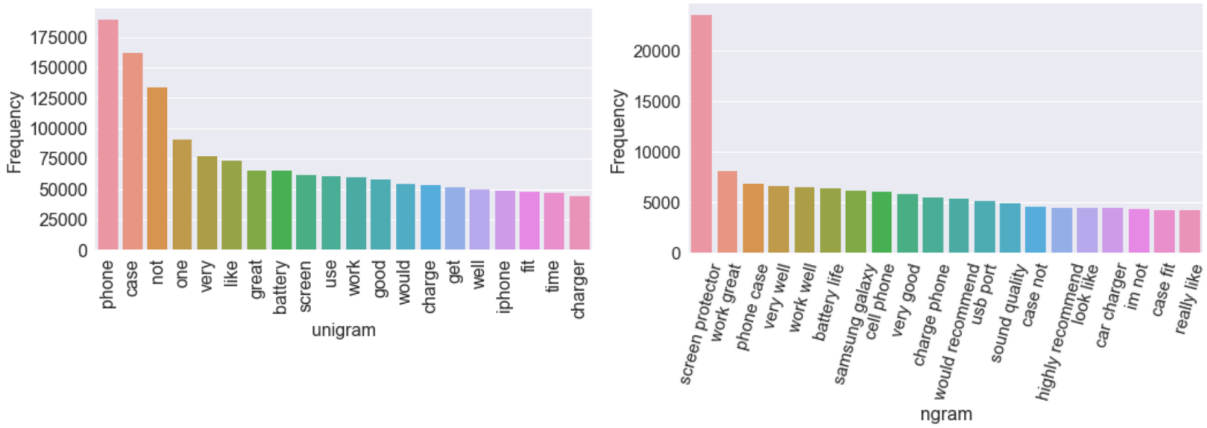
```
Confusion Matrix:
[[ 1978   55  2863]
 [  420  201  3577]
 [  285   84 28721]]

Result metrics:
              precision   recall f1-score  support

    Negative      0.74     0.40     0.52     4896
     Neutral      0.59     0.05     0.09     4198
    Positive      0.82     0.99     0.89    29090

    accuracy                        0.81    38184
   macro avg      0.72     0.48     0.50    38184
weighted avg      0.78     0.81     0.76    38184

Accuracy score: 0.8092394720301697
```



(a) Top 20 Unigrams **(b) Top 20 N-grams**

Figure 6: Top 20 Unigrams and N-grams in the Corpus by Term Frequency

Figure 7: Distribution of Sentiment Classes

4.4 Discussion

Random Forest achieved 80.9% accuracy on the test dataset, which is about 18% better than Naive Bayes. We note that the training data is heavily skewed towards the positive class. If there were more examples of negative reviews, the model would be better at identifying negative reviews. We also did not experiment with tuning the hyperparameters of the classifiers.

5 CONCLUSION

We analyzed the Amazon product review dataset and presented various review and rating distributions. We categorized the reviewers in terms of their activity levels and found that more active reviewers are less likely to give low ratings than less active reviewers. We tagged a few sentences to see if the NLTK POS tagger is able to correctly tag the grammatically incorrect text. Except for a few wrong tags, the tagging results are generally good.

We developed an extractive summarizer to extract informative and meaningful keyphrases and sentences from all the reviews of a product. The summarizer extracts keyphrases using a score based on term frequency and review frequency. The summarizer extracts sentences using a normalized score based on the `tf-idf` values of unigrams, bigrams and trigrams in the sentence. Results evaluated informally show that the summarizer is able to extract keyphrases which describe the product being reviewed, and sentences which convey information about the product and the reviewers' sentiments.

Lastly, we implemented a sentiment classifier to classify a review as positive or negative. We used keyphrases scored by `tf-idf` as features for training. We experimented with the Gaussian Naive Bayes and Random Forest algorithms. The Random Forest classifier achieved an accuracy of 80.9%.

REFERENCES

[1] 2020. *pandas - Python Data Analysis Library.* Retrieved April 5, 2020 from https://pandas.pydata.org/

[2] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Mário Jorge, Célia Nunes, and Adam Jatowt. 2018. YAKE! collection-independent automatic keyword extractor. In *European Conference on Information Retrieval.* Springer, 806–810.

[3] NumPy developers. 2020. *Matplotlib: Python plotting — Matplotlib 3.2.1 documentation.* Retrieved April 5, 2020 from https://numpy.org/

[4] Python Software Foundation. 2020. *json — JSON encoder and decoder — Python 3.8.2 documentation.* Retrieved April 5, 2020 from https://docs.python.org/3/library/json.html

[5] Eric Firing Michael Droettboom John Hunter, Darren Dale and the Matplotlib development team. 2020. *Matplotlib: Python plotting — Matplotlib 3.2.1 documentation.* Retrieved April 5, 2020 from https://matplotlib.org/

[6] Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing.* 404–411.

[7] Eirini Papagiannopoulou and Grigorios Tsoumakas. 2020. A review of keyphrase extraction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, 2 (2020), e1339.

[8] NLTK Project. 2020. *Natural Language Toolkit — NLTK 3.5b1 documentation.* Retrieved April 5, 2020 from https://www.nltk.org/

[9] NLTK Project. 2020. *nltk.stem package — NLTK 3.5b1 documentation.* Retrieved April 5, 2020 from https://www.nltk.org/api/nltk.stem.html#nltk.stem.porter.PorterStemmer

[10] NLTK Project. 2020. *nltk.tag package, NLTK 3.5b1 documentation.* Retrieved April 5, 2020 from http://www.nltk.org/api/nltk.tag.html#module-nltk.tag

[11] Miguel Won, Bruno Martins, and Filipa Raimundo. 2019. *Automatic extraction of relevant keyphrases for the study of issue competition.* Technical Report. EasyChair.

A RESULTS FROM THE REVIEW SUMMARIZER

A.1 Example 1

USER INPUT:

```
product_idx = 3000
num_sentences  = 6
```

OUTPUT:

```
**** PRODUCT REVIEW SUMMARIZER ****

Index: 3000
Product ID: B0056IKP10
Number of reviews: 16

Sample reviews:

Review 1:
 I've said it in other reviews, these things can be a crapshoot. I did alright with this one though. No hot pixels or weird build quality. The screen came well packaged and on-time.  I'm very pleased with this product and would recommend it to anyone that's dropped their phone a little too hard!

Review 2:
 Great fit, even though I immediately tore it up upon installation (careful with those thin ribbon cables!!) the part itself was a perfect fit. Looked great, packaged well.

Review 3:
 Cracked my screen and replaced it with this one and now it is crystal clear, no more cracks in the screen


SUMMARY KEY PHRASES:

dozen philips screws
good screen
new screen
one though
original screen
phone
previous screen
ridiculously small screws
screen
screen parts
thin ribbon cable
two dozen philips
wor

SUMMARY SENTENCES:

The repair kit contains the pentalobular driver and other tools.

You really should have a good set of Jewelers screwdrivers on hand, as there are a lot of ridiculously small screws.

The iPhone 4 CDMA requires a Philips #00 driver for all two dozen philips screws once inside the case.

Great fit, even though I immediately tore it up upon installation (careful with those thin ribbon cables!!)

but it looks great.No cloudy images.Recognizes touch 100\%No signs of LCD failingNot thicker or slimmer than the original screen Not happy to say the least.

Useful to fix broken device.
```

A.2 Example 2

USER INPUT:

```
product_idx = 4000
num_sentences  = 5
```

OUTPUT:

```
**** PRODUCT REVIEW SUMMARIZER ****

Index: 4000
Product ID: B0058OILYU
Number of reviews: 12

Sample reviews:

Review 1:
 This case is really good for this phone. It fits the phone perfectly. Its not loose, and its easy to put on and take off. It adds a little bit of bulk but not too much. as far as the bulk it adds, its between an otterbox commuter case and defender series. I havent used the speck candyshell case, so i cant really compare the ageis case to that.

Review 2:
 This was my favorite case when I had my LG. It protected well, but the covers for the charger and headphone jack eventually came off.

Review 3:
 8-30-11Just got this today and it's definitely a quality case on first blush.  Shipping was fast as well.  The hard plastic portion has a soft touch feel to it so it's really very comfy in hand.  It does add some bulk, but that's fine because I find that the thinness of the G2x does cause my hands to cramp a bit when doing some gaming (gaming on the g2x is the awesome).  If you're looking for a
```

nice heavy duty case.. consider this one!

SUMMARY KEY PHRASES:

ageis case
awesome case
case
case fit
commuter case
hard plastic
hard plastic portion
heavy duty case
new phone
outer hard plastic
phone
phone multiple times
speck candyshell case

SUMMARY SENTENCES:

I havent used the speck candyshell case, so i cant really compare the ageis case to that.

It has perfect cut outs as well.

The silicone attraches waaaayyyy too much pocket lint2.

My phone was ran over by an suv and only damage sustained was to the outer hard plastic of the case!!!

This case has just the right amount of impact absorption.

A.3 Example 3

USER INPUT:

product_idx = 7000
num_sentences = 4

OUTPUT:

**** PRODUCT REVIEW SUMMARIZER ****

Index: 7000
Product ID: B005G4GBLW
Number of reviews: 7

Sample reviews:

Review 1:
 good quality, quick ship great price. Perfect fit did not have to trim corners. fits great even with the case. No problems and easy to install without bubbles.

Review 2:
 The cover is good for the price. So far it hasnt given me any problems. Will probably order again. Maybe

Review 3:
 I found these protectors to be inferior to the ones I've used before for the following reasons:- They scratch easily. The protector surface is too sensitive, I get it scratched in the process of application while using the credit card edge to chase the bubbles out;- Partly for the above reason they don't last long;- Get foggy overtime, sometimes only a very short time after the application.I still give them three stars because of the price point and also because they do perform the basic function they were designed for - protect the screen.

SUMMARY KEY PHRASES:

credit card edge
first screen protector
great price
motorola photon perfect
perfect fit
photon perfect
price
protector
screen protector
ship great price
very short time

SUMMARY SENTENCES:

Nice cover, cheap price, easy to install, fit the Motorola Photon perfect.

The protector surface is too sensitive, I get it scratched in the process of application while using the credit card edge to chase the bubbles out;- Partly for the above reason they don't last long;- Get foggy overtime, sometimes only a very short time after the application.I still give them three stars because of the price point and also because they do perform the basic function they were designed for - protect the screen.

Perfect fit did not have to trim corners.

good quality, quick ship great price.