Review Digest

An "all-you-want-to-know" snippet, in an easy to understand format, for an item you want to buy.

Final Report

Team Members:

Richa Prajapati **Keshav Potluri** **Ankur Kumar**

Applied Natural Language Processing
Fall 2015

INTRODUCTION

Online shopping is becoming more and more popular in recent times. And with increased importance and popularity of e-commerce websites, reviews have become a very important part of the complete online experience. Not only are they widely available, but they also play a crucial part in shaping our opinion before we buy a product. However, the amount of effort required to form an opinion about different aspects / features associated with any product is huge, and repeating the same effort when trying to compare consumer opinions of different products adds to the complexity, and tediousness of the task.

In this project we aim to simplify and automate the task of identifying the various aspects being talked about a particular product in a review, extract these aspects and identify the sentiment associated with each of these aspects. In the end of this process, the user will have a summary of all the reviews listing the pros and cons of the specified product. We will be limiting the scope of our project to deal with cellphones only. We will be using Amazon reviews as the source of our analysis data.

PROBLEM STATEMENT

Given the nature and challenges of our data, the goal of this project was threefold:

1. To automatically extract product attributes from a given set of reviews.
2. To extract the phrases which are associated with each of the product attributes.
3. To perform sentiment analysis on these extracted attributes.

PRODUCT ATTRIBUTE EXTRACTION

Attributes of a product are features, components and other aspects of the product like "battery life", "picture quality" etc. In reviews, different people use different words and phrases to describe the same product attribute. We call the actual words or phrases that express the same attribute as "attribute expressions". For example, "battery" and "charge" are attribute expressions referring to the same attribute of phone battery and batter life. Similar attribute expressions, which are domain synonyms, need to be grouped under the same attribute group.

But, clubbing words together just because they are synonyms is neither sufficient nor straightforward. This is because words or phrases that are not synonymous might refer to the same product attribute. For example, "appearance" and "design" are not synonymous but they indicate the same product attribute - design. On the other hand, words or phrases that are synonymous might not be talking about the same feature. For example, "movie" and "picture" are synonymous but "movie" might be more related to video and "picture" to photo. Some words or phrases that are extracted might not be related to attribute features at all. For example, "new

phone", "best smartphone", "previous phone" etc. need to be filtered out to ensure that only those review sentences are analyzed which actually contain an attribute.

EXTRACTING REVIEW PHRASES ASSOCIATED WITH EACH PRODUCT ATTRIBUTE

Almost all the reviews provided by users constitute of multiple sentences. In turn, each sentence constitutes of different phrases, which are associated with different product attributes, and have corresponding adjectives and other descriptors for each of the attributes. For example, "The battery sucks, but the camera is good" - here "sucks" describes the attribute "battery" in the phrase" the battery sucks". In some cases, multiple descriptors are also associated with a single product attribute, and vice-versa. For example, "I love the camera, display and the overall feel of the phone" - here the word "love" describes all three attributes of the phone: "camera", "display" and "feel".

But identifying the semantic relationship between the product attributes, and the descriptors is not straightforward. It is further worsened by presence of sarcastic comments and the need to resolve pronoun association between descriptors and attributes present in different parts of sentences, or even across multiple sentences. An algorithm is needed which can work with different writing styles and churn out ‘single attribute phrases’, i.e. phrases constituting of a single product attribute, and its associated set of descriptors.

SENTIMENT ANALYSIS

The problem that plagued attribute extraction and phrase association arises in sentiment analysis as well. People can convey the same sentiment about a particular attribute in different ways. For example, one person might say that the "battery is not so good", second that "the battery is bad". Here, though both the reviews give a negative sentiment about the battery, the presence of word "good" might throw a simple algorithm off.

The degree of sentiment involved also varies from review to review. Continuing with the last example, one person might say that the "battery is not so good", second that "the battery is bad" and third that "the battery is the worst!". Here though, all the three sentiments are negative, the degree of negativity increases with first being a mild negative to third sentiment being extreme negative. In the case where some reviews are extreme negative, and most are mild positive, one would naturally assume the overall sentiment to be negative. However, for this project, we will be restricting our scope to identifying a sentiment as positive or negative only and not to identify the degree of sentiment.

Nevertheless, we will need to ensure that the training data is big and comprehensive enough for the classifier to understand these intricacies without overfitting to the training data.

PLOTTING RESULTS

The last challenge in this project is to display the algorithm's results in a quick and easily digestible format. Not only will we need to show the whether the general opinion around a particular attribute is positive or negative, but also the number of people who feel that way. To make the results, more objective, we need to ensure that the difference in the number of people with varying opinions is evident in a crisp and concise way. Finally, we would like to show the overall sentiment about the product as a whole as objectively as possible.

DATA SOURCE

Since the objective of the project was to extract product attributes from online reviews and evaluate the user sentiment about these product attributes from the reviews. For the scope of our project, we were specifically focusing on the category "Cellphones" and hence we needed a dataset of reviews about a single cellphone. Also, since our aim was to perform sentiment analysis, we needed a tagged dataset for the purpose of training our classifier.

We came across many datasets, but most of them were not a good fit for the goal of our project. There was a dataset matching our requirements available with Prof. Bing Liu at the Computer Science Department of University of Illinois at Chicago. The dataset was tagged for sentiment analysis as well as product attributes. However had only 600 sentences for a single cellphone and was tagged for more than 100 product features, averaging around 5-6 sentences per product attribute. We thought it was insufficient for training and testing our classifier, and hence we ruled this dataset out.

As a result we created our own dataset for the project. We created automated scripts that could take a URL for a product on Amazon as an input and return the reviews for the product in a tangible format. We used the scripts to scrape 1141 reviews for the product "I-Phone 5s" from Amazon, split it into individual phrases containing product attribute and words describing the attribute (in step 2 of the project), and tagged it manually for sentiment analysis. This was our training/dev/initial test dataset.

Creating the scripts was extremely useful as we were also able to create more test data for our project. We scraped reviews about two more products, "Samsung Galaxy S5" (2880 reviews) and "LG Nexus 5" (226 reviews). This helped us in evaluating the performance of our algorithm on untagged data about a product that had more reviews (Samsung Galaxy S5) as well as a product that had lesser number of reviews (LG Nexus 5).

Hence our overall dataset consisted of 1141 reviews (8000 product attribute phrases) for "I-Phone 5s", tagged for sentiment analysis, as our training/dev/initial test data, 2880 reviews for "Samsung Galaxy S5" and 226 reviews for "LG Nexus 5" as our test data.

THEORY/ALGORITHM USED

As described above, the whole process involved 3 major steps namely:

1. Product attribute extraction
2. Attribute phrase extraction
3. Opinion sentiment prediction

A workflow describing each step and its underlying steps is shown below:

PRODUCT ATTRIBUTE EXTRACTION

We started with tokenizing the reviews into different sentences and tagging individual words in it. A manual inspection of different review sentences showed that most of the product attributes were either nouns, adjectives, adverbs or a combination of them. We then extracted only those word phrases which used the patterns mentioned in Figure 1 and occurred a certain number of time (100 in this case).

```
P1:{<JJ.*|NN.*><NN.*>+}  # battery charging system
P2:{<RB|RBR|RBS><JJ.*|NN.*>+}  # Shows noun doing action
P3:{<NN.*>+}  # battery life, lcd screen
```

Figure 1: *Shows the Regex grammar used for chunking to extract noun phrases from Amazon reviews*

Many of these extracted phrases were synonyms of each other. In order to club them into similar groups, we used the following 3 properties[1]:

1. ***Common Words***

   Attribute expressions sharing some common words are likely to belong to the same group. For example, "battery life", "battery", "battery charger" etc.

2. ***Lexical Similarity***

   Attribute expressions whose words are synonymous in WordNet [11,16,23,33,36] are likely to belong in the same to the same group. For example, "battery" and "charger"

3. ***Domain Filtering***

   Attribute expressions whose words are related with the application domain or product are likely to be most relevant product attributes. For example, "screen", "internet", "camera" are essential features of a cellphone and hence relevant.

Keeping these constraints in mind, we proceeded with the following:

1. We formed a dictionary (called "word_dict") with words as keys and a list of all extracted attribute phrases containing that word as values
2. From these dictionary, we related every word key with every other word key if their corresponding phrase list had noun-based lexical similarity in WordNet above a particular threshold (0.8 in this case). Following a psuedo code to achieve this:

```
// Calculating pairwise similarity
For word_key1 in word_dict:
        For every word_key2 in word_dict:
                sim(word_key1,word_key2) =
                        Avg(PhraseSim(Phrase1word_dict[word_key1],Phrase2word_dict[word_key2]))
                if sim(word_key1,word_key2) > 0.8:
                        Club(word_key1,word_key2)

//Sub-function for calculating similarity between phrases
PhraseSim(Phrase1,Phrase2):
        return Max(word1Phrase1, word2Phrase2)
```

3. We now had a list of tuples of words which were similar to each other due to the phrases in which they were contained. From this list, we wanted to group together all the word tuples which had a common word. Thus, we formed another dictionary (called "topics") with key as the common word and value as set of all the words in the tuples containing that common word.
4. The dictionary "topics" contained many words unrelated to phone attributes such as "hour", "work", "good", "simple", "expectation", "fine" etc. Thus, we filtered out those words keys which did not have a noun-based lexical similarity with phones in WordNet. Following a psuedo code to achieve this:

```
// Calculating similarity with phone
ProductType = ['cell', 'phone', 'telephone', 'mobile']
For word_key in topic:
        sim(word_key,ProductType) = Avg(WordSim(word_key,ProductProductType))
        if sim(word_key1,word_key2) > 0.7:
                Club(word_key1,word_key2)

//Sub-function for calculating similarity with phone
WordSim(word1,Product):
        return Max(word1, Product)
```

RESULTS FOR ATTRIBUTE EXTRACTION

In order to test our algorithm's precision, we manually inspected all the reviews in test data and drew out the most common product attributes being talked about. We considered these as baseline attributes as they were common attributes across reviews of other cellphones as well. We then drew out a list of attributes given by our proposed algorithm and compared them as shown in the Table below:

Manual Inspection	Iphone 5s (test data)	Galaxy S5	Nexus 5
battery	battery	battery	battery
screen	display	monitor, display	display
charger	charger	charger	
speakers	headphone		
camera	camera	camera	camera
keyboard	button, home	button, home	home
software	system	system	
size	light ,lighter		light ,lighter
connection	connection	wireless	
hardware	hardware	resistance, sensor, port, scanner	processor, hardware
body	back	back	
price			
experience			
speed			
	fan		fan

Figure 2: *Expected Features vs Actual Features obtained for various products*

Our algorithm does a pretty good job of identifying the various product attributes with no manual intervention. The precision is about 78.57% on test data set and 64.28% on Galaxy S5 data set. It has surprisingly good precision on Nexus data set which has very few reviews (226 in total). This is comparable to the baseline precision range of 69-75% given by "Hu and Liu's" work[2]. (http://sentic.net/aspect-parser.pdf)

The comparatively low precision in Galaxy S5 and Nexus 5 can be due to the fact that our algorithm was unable to capture indirect/derived product attributes such as "price", "speed", "user experience" etc. This is because in our algorithm, the product attributes are purely derived form the words present in the reviews and any other word is not used. So in the case of "speed", reviews will contain words like "fast", "faster" or "slow" and in the case of "price", reviews will contain words like "cheap", "expensive", "rip off" etc. Because the algorithm is not able to club them under one group, the frequency of such words (like "cheap", "expensive", "rip off") is not high enough to be captured.

ATTRIBUTE PHRASE EXTRACTION

Initially, we tried to associate the entire sentences to the product attributes they contained, under the assumption that usually customer reviews are written by regular people and are hence, simple in structure - reviews describe one product attribute per sentence. But on analyzing the training / development data set, we found that this was not the case. Each sentence in the review were

mostly complex, with multiple product attributes and multiple set of descriptors. We classified the complex sentences into the following five categories:

1. Sentences with a single product attribute, and a single set of descriptors.

   E.g. 'The battery life is very bad."

2. Sentences with multiple product attributes, and a single set of descriptors.

   E.g. "The camera, as well as the processor is very good."

3. Sentences with a single product attribute, and multiple sets of descriptors.

   E.g. The screen is not so good, but it is enough for general usage.

4. Sentences with multiple product attributes, and multiple set of descriptors.

   E.g. "I hate the camera, the screen is also bad, however the battery is good."

5. Sentences with pronouns, and associated set of descriptors.

   E.g. "It has a 13 Megapixel camera. It takes really good pictures."

In order to overcome these challenges, and correctly extract single attribute phrases from the above 5 type of sentences, we tried using different algorithms to check which one provided the best results.

The implementation can be divided into two stages:

1. Tagging the sentences
2. Chunking / Tokenizing the sentences into phrases
3. Associating phrases with the respective product attribute

Following are the algorithms which we used, along with the detailed explanations of their specific implementations:

TAGGING THE SENTENCES

We initially tried using "nltk.pos_tag" to tag the review sentences, but it was not tagging certain words correctly. For example, words like "not" were getting tagged as conjunction instead of adverb and was interfering with attribute phrase extraction. So, we implemented a trigram back-off tagger, which was trained on brown corpus categories of 'news', 'editorial', and 'reviews' since they were similar to our product reviews.

In order to align the tags of some phrases / terms (like "not") to our liking, we added some additional training data sets.By doing so, we were able to appropriately tag the set of adverbs and conjunctions present in the text, which was not being done by either the "nltk.pos_tag" or by the trigram back-off tagger with brown corpus training data set.

CHUNKING / TOKENIZING THE SENTENCES

To correctly chunk the sentences into single attribute phrases, we decided to use grammar rules to identify the associated descriptors for each product attribute in the sentences, and tokenize them appropriately. Following are the algorithms which we used:

1. ***Context Free Grammar Algorithm***:

   This algorithm utilizes "grammar", which specifies which trees can represent the structure of a given text. These can then be used to find possible syntactic structures for the sentences. An example code for this algorithm is given in Figure 3.

```
grammar1 = nltk.CFG.fromstring("""
      S -> NP VP
      PP -> P NP
      V -> "saw" | "ate" | "walked"
      NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
      N -> "man" | "dog" | "cat" | "telescope" | "park"
      P -> "in" | "on" | "by" | "with"
      """)
sentence = "Mary saw Bob".split()
rd_parser = nltk.RecursiveDescentParser(grammar1)
for tree in rd_parser.parse(sent):
      print(tree)
Output:(S (NP Mary) (VP (V saw) (NP Bob)))
```

   Figure 3: *Code snippet from Chapter 3- NLTK Book*[3]

2. ***Using Regex Parsers to tokenize sentences:***

   In this we tried to find patterns in the tags of the phrases, and then define regex patterns to try and chunk them. A sample code for this algorithm is given in Figure 4.

```
P1:{<JJ.*|NN.*><NN.*>+}  # battery charging system

P2:{<RB|RBR|RBS><JJ.*|NN.*>+}  # Shows noun doing action
P3:{<NN.*>+}  # battery life, lcd screen
```

   Figure 4: *Shows the Regex grammar used we tried to extract single attribute phrases from Amazon reviews*

3. ***Utilizing conjunctions, and other punctuations in the sentences to perform tokenization:***

   In this, we tried using the conjunctions and the punctuations present in the sentences to determine distinct phrases. A pseudo-code for this is shown below:

```
def chunk_Data_On_CC(tagged_review_Data):
  for each sentence in tagged sentences:
        Split the sentence at each Conjunction
        Create a list of sentences
        for each item in the list of sentences:
              Split them on punctuations like commas, semi-colons, and colons
              Store the list of phrases in the list
  return Dictionary_of_list_of_phrases_for_all_reviews
```

Of all the methods that we tried, the final method i.e. using conjunctions, and punctuations to tokenize sentences worked the best on the review data. This method could not tokenize sentences with multiple product attributes and single set of descriptors appropriately. However, analysis of the data set showed that the percentage of such review sentences were very less, and could be safely used without much repercussions on the overall re-call of the algorithm.

ASSOCIATING PHRASES WITH PRODUCT ATTRIBUTES

Now that we had extracted the "most talked about" product attributes and split the review sentences in phrases containing a single product attribute, we are left we the job of associating each phrase with the corresponding product attribute it contains. We need to build this association so that the sentiments predicted per phrase can be summed-up for the corresponding product attribute. Phrases not containing the extracted product attributes will be discarded.

To do this, we first create a dataframe for every extracted product attribute. We then use a simple algorithm to traverse through each phrase and find whether it contains a product attribute or its synonyms. If yes, then the phrase is added to corresponding attribute's dataframe. If a phrase contains multiple product attributes then the phrase is added to the dataframes of all the contained attributes. Following is a psuedo code showing the alogrithm:

```
for each phrase in a review sentence:
      set1 = all the words in that Phrase
      for every ProductAttribute in Topics:          #Topics is a dictionary containing all the product
            set2 =synonyms for that product attribute      #attributes and its synonyms
            If there are any common terms in set1 and set2 then:
                  Associate that Phrase to that Product Attribute
```

RESULTS OF THE ATTRIBUTE PHRASE EXTRACTION

The method to split sentences on conjunctions and punctuations, though simple, gave surprisingly good results on the product reviews. Most of the phrases contain a single attribute

with a single set of descriptors. In other words, the resulting phrases are complete in themselves and sufficient to extract sentiments for each of the product attribute they contain.

Reviews	Single attribute Phrases	Associated Product Attribute
I was very excited when I received the two phones. Fast shipping and everything, however after about two months of using the phones white lines appeared on the screen of both phones which then cost me additional money to replace the screens. Very disappointed since I selected new phones and not used ones for the purpose of avoiding additional cost.	I was very excited when I received the two phones	Phone
	Fast shipping	
	Everything	
	After about two months of using the phones white lines appeared on the screen of both phones which then cost me additional money to replace the screens	Screen
	Very disappointed since I selected new phones	Phone
	Not used ones for the purpose of avoiding additional cost	
The phone I purchased was suppose to be unlocked but it was not. There was a small crack on the screen that was not shown on the picture of the phone. The charger they gave me only lasted for 2 weeks and the worst part was the battery life of the phone. I can only last about 2 hours when the phone is fully charged. I've had an I Phone before purchasing one from them and it would last me the whole day. I recommend you don't buy a phone from this company, or an I Phone at least.	The phone I purchased was suppose to be unlocked	Phone
	It was not	
	There was a small crack on the screen that was not shown on the picture of the phone	Screen
	The charger they gave me only lasted for 2 weeks	Charger
	The worst part was the battery life of the phone	Battery
	I can only last about 2 hours when the phone is fully charged	Phone

	I've had an I phone before purchasing one from them	Phone
	It would last me the whole day	
	I recommend you don't buy a phone from this company	Phone
	An I phone at least	Phone

Figure 5: *Table showing original reviews and the phrases extracted for product attributes.*

SENTIMENT PREDICTION

In order to extract sentiments from the phrases derived in the Product Phrase Extraction and Association step, we decided to use various machine learning techniques. We used vectorizers to extract and learn syntactic patterns accompanying different sentiments in the reviews. We then trinaed classifiers on manually labelled training dataset to predict the sentiment contained in each of the phrases.

Thus, this step can be separated into three segments:

1. Manually labelling the training set into positive, negative and neutral sentiments
2. Identifying the best vectorizer to extract characteristic features differentiating one sentiment from another
3. Identifying the best classifier which will used these characteristic features to predict sentiments

This involved finding which combination of vectorizer and classifiers works the best for the dataset, without over-fitting to the training dataset.

LABELING TRAINING SET

Using the procedures explained in the Data pre-processing section, we split the Amazon review sentences into single attribute phrases. Then we manually labelled the data, classifying them into 'Negative mentions' (represented as -1), Neutral Mentions (represented as 0), and Positive mentions (represented as 1).

VECTORIZER

We tried various vectorizers found in ScikitLearn python machine-learning library with different parameters values before finalizing which vectorizer will be the best fit for our dataset. Following are the vectorizers we considered, along with the variations in parameters we did.

1. ***Count Vectorizer***

   This vectorizer converts text 'documents' into a matrix of token counts, specifically, a sparse matrix representation of the counts using the *scipy.sparse.coo_matrix.*

   We specified the analyzer to 'words', allowed lowercasing of tokens, and also asked the vectorizer to consider Unigrams, Bigrams, and Trigrams as features. We limited the number of features to 1000. A sample code of the vectorizer is given below:

```
vectorizer = CountVectorizer (
   analyzer = 'word', # Assigning analyzer as word
   tokenizer = tokenize,
   lowercase = True, # Lowercasing the words
   stop_words = 'english', # For removing stop words from being considered as features
   ngram_range = (1,3), # Allowing unigrams, bigrams, and trigrams to be considered as features
   max_features = 1000 # Using the top 1000 features
)
```

2. ***Tf-Idf Vectorizer***

   This vectorizer converts the collection of raw 'documents' into a matrix of TF-IDF features. It is equivalent to applying CountVectorizer on the set of 'documents' and then applying Tf-Idf transformation on it.

   "Tf–Idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus."– *Wikipedia*

   In this vectorizer as well, we tried different combinations of various parameters, to output the best results. A sample code for this vectorizer:

```
vectorizer = TfidfVectorizer (
   analyzer = 'word',
   min_df=1,
   tokenizer = tokenize,
   lowercase = True,
   stop_words = 'english',
   ngram_range = (1,3),
   max_features = 1000
)
```

CLASSIFIER

To perform sentiment analysis on the phrases associated with different product attributes, in conjunction with the vectorizers, we tried three different classifier algorithms to analyze which combination provided the best precision, and re-call, without over-fitting to the training data set. Following are the classifiers which we considered, explained with reasons for why they were or weren't chosen.

1. ***Logistic Regression Classifier***

   This model is a discriminative model, which can be used to directly estimate the probability of occurrence of y given occurrence of x (p(y|x)). For this model to work correctly, there is no restriction on the features to be co-related. It can be used to provide multi-category classification in cases where the categories are exhaustive, and mutually exclusive, i.e. every instance belongs to one, and only one category. A sample code for the LR Classifier :

```
log_model = LogisticRegression ()
log_model = log_model.fit (X=X_train, y=y_train)
y_log_pred = log_model.predict (X_dev)
```

2. ***SVM Classifier***

   This is also a discriminative classifier, which is formally defined by a hyper-plane, i.e. provided labelled training data set, this classifier outputs an optimal hyper-plane, which can then we utilized to classify new examples. A sample code for SVM classifier :

```
svm_model = nltk.svm.SVC (kernel='linear', C = 1.0)
svm_model = svm_model.fit (X=X_train, y=y_train)
y_svm_pred = svm_model.predict (X_dev)
```

3. ***Random Forest Classifier***
   A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).A sample code for RF classifier :

```
rf_model = RandomForestClassifier(n_estimators=1000)
rf_model = rf_model.fit (X=X_train, y=y_train)
y_rf_pred = rf_model.predict (X_dev)
```

RESULTS OF THE OPINION SENTIMENT PREDICTIONS

On the training dataset, the combination of Count Vectorizer and Logistic Regression Classifier gave a precision of 81%. However, running this combination on the test dataset made us realize that Count Vectorizer and Logistic Regression Classifier were over-fitting on the training data set and not producing as good results as expected.
On the other hand, though the combination of Tf-Idf Vectorizer and Random Forest Classifier was giving only 76% precision on the training data set, it was giving much better results on the test dataset.

```
# Classifier report for the Random Forest Model
print(classification_report(y_dev, y_rf_pred))
```

```
             precision    recall  f1-score   support

         -1       0.76      0.76      0.76        17
          0       0.65      0.75      0.70        20
          1       0.85      0.74      0.79        23

avg / total       0.76      0.75      0.75        60
```

Figure 6: *Shows the precision and recall for the final combination of Tf-Idf Vectorizer and Random Forest Classifier*

Hence, we decided to trade-off on lower accuracy on training set for better results on the test data set and finalized on using Tf-Idf Vectorizer in combination with Random Forest Classifier to predict sentiments. The precision and recall for Tf-Idf Vectorizer and Random Forest Classifier combination is shown in Figure 6. Also, some of the results of sentiment prediction on the extracted attribute phrases are shown in the table below:

Phrases from Reviews	Sentiment
I was very excited when I received the two phones	1
Fast shipping	1
Everything	0
After about two months of using the phones white lines appeared on the screen of both phones which then cost me additional money to replace the screens	-1
Very disappointed since I selected new phones	-1
Not used ones for the purpose of avoiding additional cost	0
The phone I purchased was suppose to be unlocked	-1
It was not	-1
There was a small crack on the screen that was not shown on the picture of the phone	-1
The charger they gave me only lasted for 2 weeks	-1
The worst part was the battery life of the phone	-1
I can only last about 2 hours when the phone is fully charged	-1
I've had an i phone before purchasing one from them	0
It would last me the whole day	1
I recommend you don't buy a phone from this company	-1
An I phone at least	0

Figure 7: *Table showing sentiments predicted by Random Forest Classifier*

RESULTS

We ran our algorithm on three datasets: Training/Dev dataset of I-Phone 5s (1141 reviews), large test dataset of Samsung Galaxy 5s (2880 reviews) and a smaller dataset of LG Nexus 5 (226 reviews).

Our Training/Dev dataset was manually tagged for sentiment analysis. Hence, we were able to measure the precision and recall for this dataset. On an average, the precision obtained was 0.76, with precision being as high as 0.85 for positive sentiments and as low as 0.65 for neutral sentiments. The recall obtained was on an average 0.75, with little variance across all the sentiments. The recall obtained was very consistent. The overall f-1 score was 0.75.

In terms of product attribute extraction, while tagging the training data manually, we made a list of attributes we were expecting from the algorithm. The extracted attributes for all three data sets are as shown below.

Manual Inspection	Iphone 5s (test data)	Galaxy S5	Nexus 5
battery	battery	battery	battery
screen	display	monitor, display	display
charger	charger	charger	
speakers	headphone		
camera	camera	camera	camera
keyboard	button, home	button, home	home
software	system	system	
size	light ,lighter		light ,lighter
connection	connection	wireless	
hardware	hardware	resistance, sensor, port, scanner	processor, hardware
body	back	back	
price			
experience			
speed			
	fan		fan

Figure 8: *Table showing actual product features extracted vs expected product features*

Some of the expected attributes were not obtained in the test data. But on closer inspection we found that this was because these attributes were not frequently talked about in the reviews for the test data.

For I-Phone 5s (training data) the overall results were as shown below in the graphs. As seen in the graphs, neutral mentions are included while talking about individual product attributes but

not when talking about the product overall. This is because, most of the neutral mentions about the overall product were not sentiments, but general statements that do not convey sentiments like "I gifted this phone to my wife", "I travel a lot with this phone" etc. Hence we removed the neutral sentiments from the final results for the phone as a whole. Overall, 441 users were happy with the device, while 335 were not as happy, conveying that overall users are not as happy as one would expect. Battery and Charger were the biggest pain points for I-Phone 5s. The camera, the weight and the system was a plus for it.

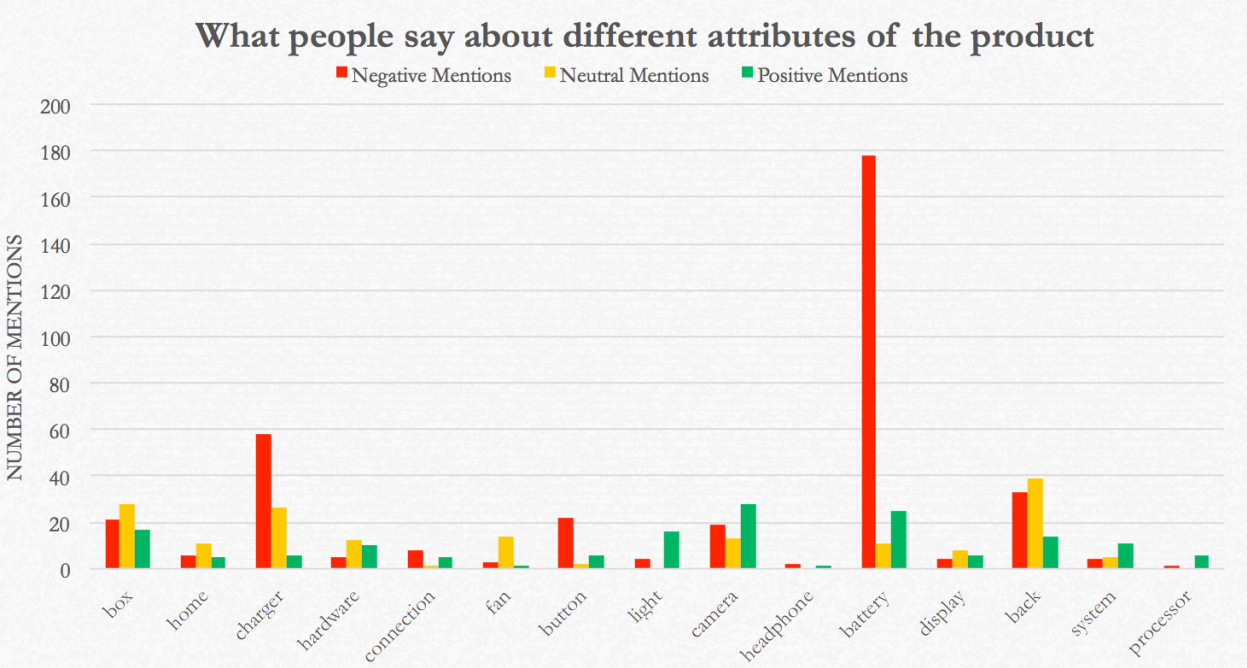

Figure 9: *Figure showing sentiments for each product attribute for I-Phone 5s*

Figure 10: *Overall sentiments for I-Phone 5S*

The results were pretty good in both the test data sets. Samsung Galaxy S5 was evaluated with 2880 reviews. Overall 1937 positive mentions were obtained with 1005 negative mentions, pointing to overall a better sentiment from the users towards the phone. Charger and Battery were again big pain points along with one of its ports getting negative mentions. The camera and the display being the positive attributes of the phone.

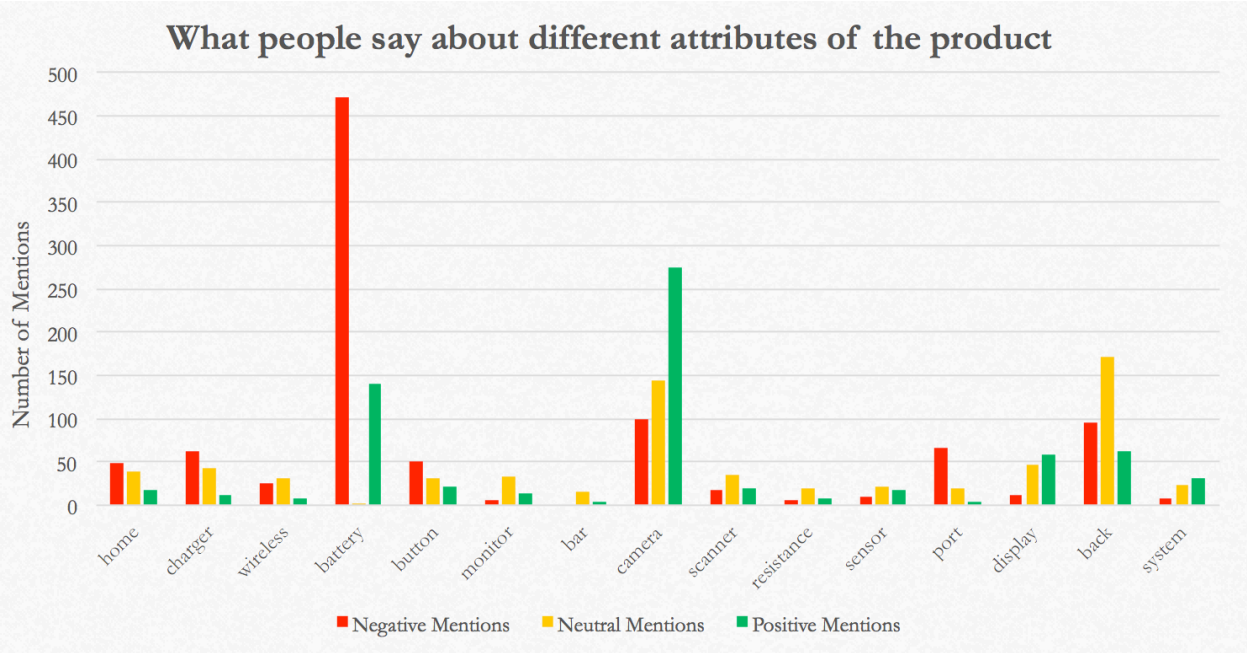

Figure 11: *Figure showing sentiments for each product attribute for Samsung Galaxy S5*

Figure 12: *Overall sentiments for Samsung Galaxy S5*

The results on a test dataset of LG Nexus 5 with relatively less reviews (226 reviews) were also very good. Overall, there were 189 positive mentions along with 99 negative mentions, indicating an overall positive sentiment. Battery and Charger again getting negative reviews (Smartphones!!!). However, LG Nexus got positive reviews in many categories like camera, weight, display, processor and hardware.

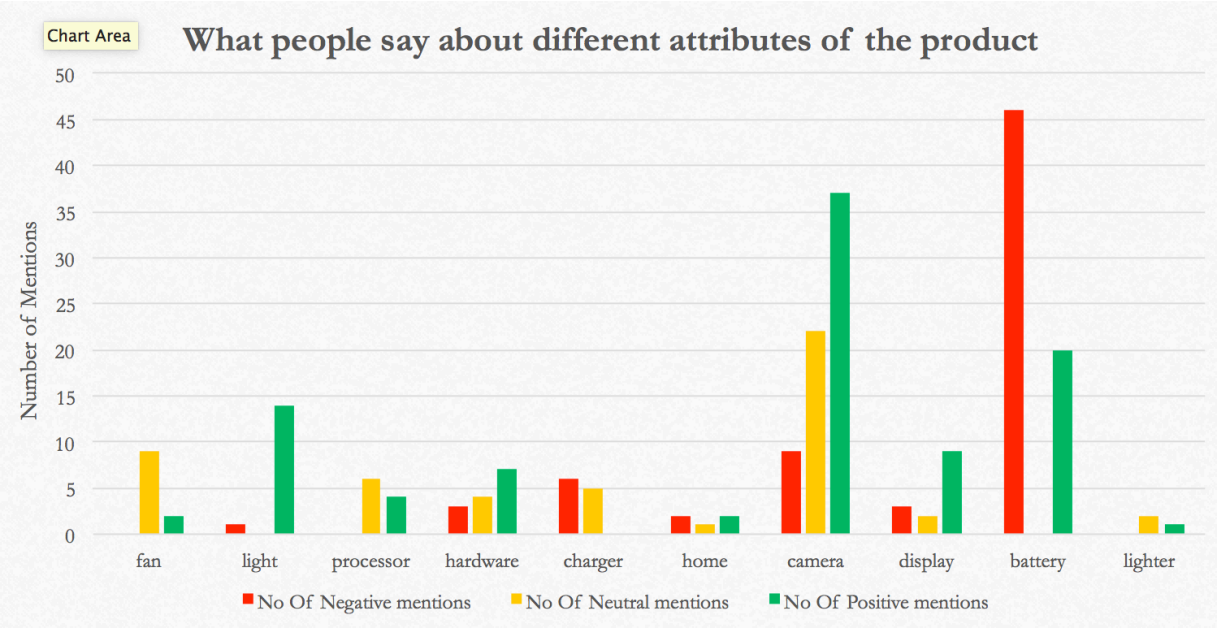

Figure 13: *Figure showing sentiments for each product attribute for Nexus 5*

Figure 14: *Overall sentiments for LG Nexus 5*

CONCLUSION (HOW FAR WE GOT)

The algorithm worked well within the product category of Cellphones. The overall precision and accuracy obtained was good, both in terms of feature extraction as well as sentiment analysis. Every part of the algorithm was automated, and hence the results were obtained with minimum human intervention. The algorithm worked well even when the number of reviews were less. The entire algorithm was written in existing libraries and packages within NLTK and hence is affected by the performance of these libraries and packages. There is a scope for improvement, but given the constraints, the overall goal of the project was achieved successfully.

FUTURE WORK

The overall results for the project were very good, but there is always room for improvement. We were able to extract very good features by selecting features that were lexically more similar to the product category ("cellphones") with the help of WordNet. WordNet however, sometimes does not give an accurate measure of how close words are to each other. Substituting it with another resource that can establish a more accurate lexical similarity can maybe provide better results in terms of feature extraction. Creating an ontology for products and its attributes is another way of achieving a higher accuracy.

Also, the current code takes about 15 minutes to provide output, because feature extraction requires extensive usage of WordNet to identify similarity between the words extracted as "potential product attributes". First the words are grouped together based on their similarity to each other and clubbed together as one product attribute, and then the attributes are filtered based on their similarity to the actual product category ("cellphones"). This process is performance intensive, and hence there is a scope for improvement in terms of performance. Substituting WordNet with some other resource, creating a manual ontology, manually intervening to club potential product attributes into final product attributes etc. are some of the ways in which this can be acheived.

The scope of the project allowed us to work on only one product category, i.e. "Cellphones". The same algorithm can be used to evaluate other product categories like "Television", "Car", "Scented Candles" etc. as well. The project can be incorporated into an online app or a tool of sorts that can help the users evaluate different products of their choice. Also, the project just focused on evaluating single products. The project can also be extended to compare different products on different product attributes.

INDIVIDUAL CONTRIBUTION

The project work was divided equally among all the three group members. The group collaboratively researched and discussed different approaches that we could take, divided the work equally, worked individually on our own parts and then came together again to evaluate the

results and identify future course of action. There were a number of components that individuals worked on, but did not give good results and hence were excluded in the final project. For the purpose of not excluding that effort, we have also included those things as a part of the individual contribution.

Ankur worked on the sentiment analysis. He worked on vectorizing and classifying the sentiments for every product attribute. He worked on Count Vectorizer, tf-idf Vectorizer, Logistic Regression Model, SVM and Random Forest Model. He also worked on extracting phrases about product attributes from the reviews. He created the n-gram tagger and trained it on grammar that was required for the project. He also worked on Context Free Grammar and Regex Parsers to tokenize the reviews for phrase extraction.

Richa worked on feature extraction. She came up with different regex parser grammars for chunking and extracting most common Noun Phrases. She reiterated on this to obtain the best results. She then worked on grouping the extracted words together based on common terms and then leveraging WordNet to improve the grouping of words using lexical similarity. She worked associating the extracted phrases to the individual extracted features.

Keshav worked on scraping the data from the website, cleaning it and converting it into tangible format. He then worked on improving the product attribute extraction algorithm. He improved the extracted keywords by filtering them on their similarity to the product category. He also improved the accuracy of grouping of words. He worked on improving code performance and quality. He worked on data visualization for the results.

All three together spent time on research and coming up with the strategy. All three divided the training data into three parts and tagged manually one part of the data. Richa and Ankur worked on presentation. Keshav created the poster. All three divided the final report into three equal parts and worked on it.

REFERENCES

1 Clustering Product Features for Opinion Mining by Zhongwu Zhai, Bing Liu, Hua Xu, and Peifa Jia

2 A Rule-Based Approach to Aspect Extraction from Product Reviews by Soujanya Pouria, Erik Cambria, Lun-Wei Ku, Chen-Gui, and Alexander Gelbukh

3 Natural Language Processing in Python by Steven Bird, Evan Klein, and Edward Loper

Lastly, we would also like to thank Marti Hearst who gave us access to the variety of papers on Opinion Mining and Sentiment Analysis. It helped us understand the overall process and come up with various strategies to implement the steps involved.