ABSTRACT

Opinion mining is the task to identify the user opinion about a particular object. So, the opinion mining tool processes the reviews collected from different reviewers by generating a list of object/product features and performing aggregation of the opinions about each feature. With the advent of Web 2.0 user got freedom to interact with other users and can create their own web contents. The use of social networking sites (such as Facebook, Twitter, Orkut) and SMS for text messaging has increased a lot recently, which lead to emergence of new language aimed for compactness and time saving. As a result, thousands of abbreviations and shortcuts are used regularly.

Now a days, Social networking sites and online shopping sites like Amazon.com are used by the users to express their opinion about products, events, people etc. As the reviews available on these sites may contain noise such as misspellings, typos, non-standard abbreviations, stylish writing, there is need to make the data noise free so that it can be used for opinion mining.

In this work a framework has been proposed to perform opinion analysis of noisy reviews using techniques such as Term similarity and Document frequency. The reviews for different products have been tested by this framework and the corresponding result is displayed in Negative (-ve) and Positive (+ve) form. The results are found to be satisfactory for all the tested products.

TABLE OF CONTENT

LIST OF FIGURE

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Objective:

The objective of this thesis is to fetch reviews of a product of various companies and selecting the best product for the consumer by analyzing the reviews. A product is launched by various companies who provide different features for the same product in this work software is developed, such that it would find out the best product from various types by checking out the reviews available on the various social networking sites [9, 10].

Opinion mining is the task to identify the user opinion about a particular object. So, the Opinion mining tool processes the Reviews collected from different reviewers by generating a list of object/product features and performing aggregation of the opinions about each feature.

Social networking sites and product selling sites are used by the user to express their opinion about products, events, people etc.

1.2 Problem Statement:

The data available on social networking sites may contain noise, there is need to make the data noise free so that it can be used for Opinion mining.

The Noise we are talking about is the misspellings, typos, non-standard abbreviations, stylish writing – collectively called as Orthographic Errors. These errors are classified into three categories-Misspellings, Typos and Intentional deviations. Typos are introduced due to the incorrect key press by unskilled person. Intentional deviation involves deliberate deviation from correct spelling. Stylish writing like leetspeak is example of Noisy Review [11 - 13].

Table 1 Example of standard and noisy review

Standard Review	Noisy Review
1>Sensor and Image Quality All three cameras share a very similar sensor and 18 megapixels, and so their image quality will be virtually the same. All are capable of taking professional quality images. 2>Power is excellent and you rarely realize when you touch 100 with enough power to surge ahead to overtake. Excellent suspension too. Seating is very comfortable, front or back. Climate control.	1>Sensor nd emz qalty all 3 camrs shar a vry smlr sensr n 18 MP, n so thr emz qalty ll b vrtualy d sam. Al r capbl of takng profsnl qalty emzs. 2> Power is xcellnt n u rarly relese ven u tch 100 wid enough power 2 surge ahed 2 ovrtak. Xcellent suspenson too.Seting is vry com4table, frnt or back. Climate cntrl.

1.3 Motivating Factor:

In early time people use to buy a product by a watching a advertisements or by communication with closed one's (Friends and Family members).

Sometimes due to lack of communication or knowledge a customer buy is an inefficient product. Now a day so many reviews available for a product on social networking sites.

The advanced technology has provided so many options for the user, Early a product was launch by not more than two or three companies but now a company's it's self launch so many products(models) an a single product is available in the market with various features. So a user could select a product according to his/her requirement as many options are available for him.

So by analyzing the reviews one could find out different properties of that product which helps his/her to buy an efficient product.

1.4 Related work:

Opinion mining is studied by various researchers; [14, 15] provides a survey of various techniques used for the opinion mining. [14] also describes various available resources and benchmark datasets.

Opinions can be classified into two categories –Document level, feature based [15]. Document level opinion mining categorizes a single review document as positive or negative. There are some drawbacks of document level opinion mining – a review marked as positive does not mean that the user liked all features of the product. Feature based opinion mining solved this problem, first product features described by user in review are found out and then opinion about each feature is determined. So the feature based opinion mining can be treated as task of finding all quintuple (Object, Feature, Opinion, Opinion Holder and Time) from the review document [15].

In [16] author suggested noise removal techniques such as identifying sentence boundaries, upper/lower case correction, correcting spelling errors using spell suggested. After doing the preprocessing, opinion expressions are found by POS tagging. Our system is different as it uses similarity measure based on edit distance and performs feature based opinion analysis.

1.5 Organization of the thesis:

This thesis work is organized as follows.

Chapter 1: Deals with providing the objective, problem statement, motivation and related work of undertaking this frame work as well as organization of this dissertation.

Chapter2: Deals with the concepts of data mining, data mining techniques and algorithms. It provides basic knowledge of 'sentiment mining' in reviews and how to find out feature of particular product from given reviews.

Chapter3: In this chapter we describe/present the proposed framework for handling noisy reviews.

chapter4: presents the performance study conducted on the proposed algorithm. Each conducted experiment is discussed and detailed comments on the results are given.

Finally, Chapter 5 concludes the thesis and gives some suggestions for future work.

Publications: A paper describing a frame work for handling noisy reviews was presented at the conference mentioned below –

- Brahmraj Singh Rawat, Anwar Shaikh and Rajni Jindal: 2012. Handling Erroneous Reviews in Opinion Mining. In Proceeding of International Conference on Recent Trends of Computer Technology in Academia, Udaipur (India). 21-23 April 2012.

CHAPTER 2 LITERATURE REVIEW

2.1 Data Mining Basic Concepts:

2.1.1 What is Data Mining?

In current world, the increasing technology power and advanced technologies has increased the business need, and now people expect more from systems. Now a day, the computer systems are not only used for storing data but also for providing information and forecasting. In normal terms, data mining refers to extracting or "mining" knowledge from large amounts of data. We know that the mining of gold from rocks or sand is referred to as gold mining rather than rock or sand mining. Thus, data mining should have been more appropriately named "knowledge mining from data," which is unfortunately somewhat lengthy. "Knowledge mining," a shorter term may not show the emphasis on mining from large amounts of data. Nevertheless, mining is a term which characterizes a process that finds small set of precious nuggets from a great deal of raw material.

The data mining uses complex data structure, Artificial Intelligence and Algorithms. It use full for learning from previous knowledge and recognizing hidden data pattern and providing the realistic results along with rationalization. Knowledge Discovery in Database also known as KDD is a synonym of Data Mining, please refer to [7, 8, 10].

2.1.2 Data Mining Techniques:

The most important data mining techniques are as follows:

1 Clustering:

Clustering is a process of partitioning a set of data (or objects) in a set of meaningful sub-classes, called cluster. Cluster is a collection of data objects that are similar to one another and thus can be treated collectively as one group but as a collection, they are sufficiently different from other groups. Clustering is an unsupervised classification which means we do not know the class labels and may not know the number of classes. Clustering has wide applications in

Pattern Recognition, Spatial Data Analysis, Image Processing, Market Research, Information Retrieval, Web Mining etc.

2 Classifications:

Classification is the task of mapping a data item into one of several predefined classes. In the Web domain, one is interested in developing a profile of users belonging to a particular class or category. This requires extraction and selection of features that best describe the properties of a given class or category. It's can be done by using supervised inductive learning algorithms such as decision tree classifiers, naive Bayesian classifiers, k-nearest neighbor classifiers, Support Vector Machines etc.

3 Association Rule:

Association rule discovery techniques are generally applied to databases of transactions where each transaction consists of a set of items. In such a framework the problem is to discover all associations and correlations among data items where the presence of one set of items in a transaction implies (with a certain degree of confidence) the presence of other items.

4 Sequential Patterns:

Sequential patterns are another form of a commonly used application in data mining. Sequential pattern mining functions are quite powerful and can be used to detect the set of customers associated with some frequent buying pattern. A sequential pattern function tries to detect frequently occurring patterns in the data.

2.1.3 Algorithms for finding the frequent patterns in Data mining:

1 Apriori Algorithm:

The Apriori algorithm was appeared first in 1994 [1, 2] and remain the standard reference of all algorithms for finding association rules. The Apriori was the much less candidate set of itemsets it generates for testing in every database pass. Here comes the time saving. The algorithm benefited of the fact that for an itemset to be frequent, all its subsets must be frequent. The advantage of this algorithm is uses large item set property and easily parallelize.

2 FP Growths:

An FP-tree [3] is an extended prefix-tree structure for storing compressed and crucial information about frequent patterns, while the FP-growth algorithm uses the FP-tree structure to find the complete set of frequent patterns. After we build the FP-tree, the FP-growth algorithm recursively builds “conditional pattern base” and “conditional FP-tree” for each frequent item from the FP-tree and then it uses them to generate all frequent itemsets.

3 CATS tree:

The CATS tree [3, 4] contains all transactions of a database in a descending order of local frequency. Unlike the construction of FP-tree, new transactions are added at the root level. If the new transaction contains the same items that also appear in child nodes, the transaction is merged with the node with the highest frequency. The frequency of node is increased. The remainder of the transaction is added to the last merged node as a branch and the process is repeated recursively until all common items are found.

4 Can Tree:

The Can Tree [5] captures the contents of a transaction database with only one scan. The items are arranged according to some canonical order, which can be determined by the user prior to the mining process or at runtime during the mining process such as lexicographic order or fixed frequency-related order. The order will not be affected by the node frequency. It follows a divide and- conquer approach to mine frequent patterns, which is similar to FP-growth.

5 IP Tree:

The IP-tree [6] that contains all nodes appearing in the original database following a frequency-descending order. Like the tree structure mentioned in earlier algorithms, transactions are firstly inserted into IP-tree according to a lexicographic or alphabetic order. The actual support of each item is recorded into F-list. After one scan of the database, we rearrange *F*-list in frequency descending order and restructure the tree nodes according to this newly ordered *F*-list through sorting each path. The IP-tree improves the possibility of prefix-sharing among all the patterns in database with one scan and thus enhances the compactness of the tree.

2.2 Sentiment Mining/Opinion mining:

Sentiment analysis or opinion mining is the computational study of opinions, sentiments and emotions expressed in text. It is the area of research that attempts to make automatic system to determine human opinion from text written in natural language. It seeks to identify the view point underlying a text span.

In general, opinions can be expressed on anything, e.g., a product, a service, an individual, an organization, an event, or a topic. We use the term object to denote the target entity that has been commented on. An object can have a set of components (or parts) and a set of attributes (or properties).Each component may have its own sub-components and its set of attributes, and so on [14, 15].

Example Review:

“I bought a Samsung Galaxy a few days ago. It was such a good phone. The touch screen was really cool. The voice quality was clear too. Although the battery life was not long, that is ok for me. However, my friend was mad with me as I did not tell him before I bought the phone. He also thought the phone was too expensive, and wanted me to return it to the shop. “

The question is: What we want to mine or extract from this review?

- The first thing that we may notice is that there are several opinions in this review.
- Also notice that the opinions have some targets or objects on which the opinions are expressed.
- Finally, we may also notice the sources or holders of opinions.

Opinion mining tries to find out all these things from a given review text.

2.2.1 Representation of Opinion:

Opinion can be represented as a tuple - (O, F, SO, H, T)
Where

- O - is a target object.

- F - is a feature of the object O.
- H - is an opinion holder.
- T - is the time when the opinion is expressed.
- SO - is the sentiment value of the opinion of the opinion holder H on feature F of Object O at time T. SO is positive, negative, or neutral, or a more granular rating.

2.2.2 Types of Opinion:

- Direct Opinions: sentiment expressions on some objects, e.g., products, events, topics, persons.
    - E.g., “the picture quality of this camera is great”
    - Subjective sentences.
- Comparisons: relations expressing similarities or differences of more than one object. Usually expressing an ordering.
    - E.g., “car x is cheaper than car y.”
    - Objective or subjective sentences.

2.2.3 Objectives of Sentiment mining:

Given an opinionated document,

- Discover all quintuples (O, F, SO, H, T)
- So, that Unstructured Text is converted to Structured Data.

2.2.4 Sentiment Mining Techniques:

1. Sentence Level
2. Document level
3. Feature Based

2.2.4.1 Sentence Level Sentiment Classification:

Sentence-level sentiment analysis has two tasks:

a. Subjectivity classification: Find out sentence is Subjective or objective.

   Objective: e.g., I bought a Samsung Galaxy a few days ago.

   Subjective: e.g., It is such a nice phone.

b. Sentiment classification: For subjective sentences or clauses, classify positive or negative. Positive: It is such a nice phone.

2.2.4.2 Document Level Sentiment Classification:

Supervised learning:

Classification is done using predefined data.

Supervision: Data is labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).

Unsupervised learning (clustering):

Class labels of the data are unknown.

Given a set of data, the task is to establish the existence of classes in the data

2.2.4.3 Feature Based:

Identify the sentiment of the opinion holder based on the features of a particular object. Because a person may like some features and some other may not like it.

It gives better and in depth analysis of the product than the sentence level and document level sentiment classification.

2.3 Unsupervised Review Classification:

Most of the times opinion words and phrases are the dominating indicators for sentiment classification. Thus, using unsupervised learning based on such words and phrases would be quite useful [8, 15].

This technique performs classification based on some fixed syntactic phrases that are likely to be used to express opinions.

The algorithm consists of three steps:

Input: Written review

Output: Classification (i.e. positive or negative)

Step 1: Use part-of-speech tagger to identify opinion phrases.

Step 2: Estimate the semantic orientation of extracted phrase.

Step 3: Assign the given review to a class (either recommended or not recommended)

All these steps are described in details –

2.3.1 Extract the Opinion phrases:

It extracts phrases containing adjectives or adverbs. The reason for doing this is that research has shown that adjectives and adverbs are good indicators of subjectivity and opinions. Therefore, the algorithm extracts two consecutive words, where one member of the pair is an adjective/adverb and the other is a context word.

- Two consecutive words are extracted from the review if they match with patterns in the table.
- Reason – **Adjectives & Adverbs** are good indicators of **Opinion**.

Table 2 Patterns of POS tags for extracting two-word phrases

	First word	Second word	Third word(Not Extracted)
1	**JJ**	**NN or NNS**	**anything**
2	**RB, RBR or RBS**	**JJ**	**not NN nor NNS**
3	**JJ**	**JJ**	**not NNN nor NNS**
4	**NN or NNS**	**JJ**	**not NN nor NNS**
5	**RB, RBR or RBS**	**VB, VBD, VBN or VBG**	**anything**

Where

JJ - Adjective

NN - Noun, singular

NNS- Noun, plural

RB - Adverb

RBR- Adverb, comparative

RBS - Adverb, superlative

VB- Verb, base form

VBD- Verb, past tense

VBG - Verb, present participle

VBN - Verb, past participle

- **Stanford POS Tagger** is used for tagging the text
  - **http://nlp.stanford.edu/software/tagger.shtml**
  - Class *MaxentTagger* is used to for tagging the text file.
  - The output is in the form of WORD/TAG eg. Nice/JJ – means "nice" is Adjective.
- Once the tagged text is available, find out the two consecutive words which have the tags mentioned in the above table.
- Below diagram describes the POS tagging process.

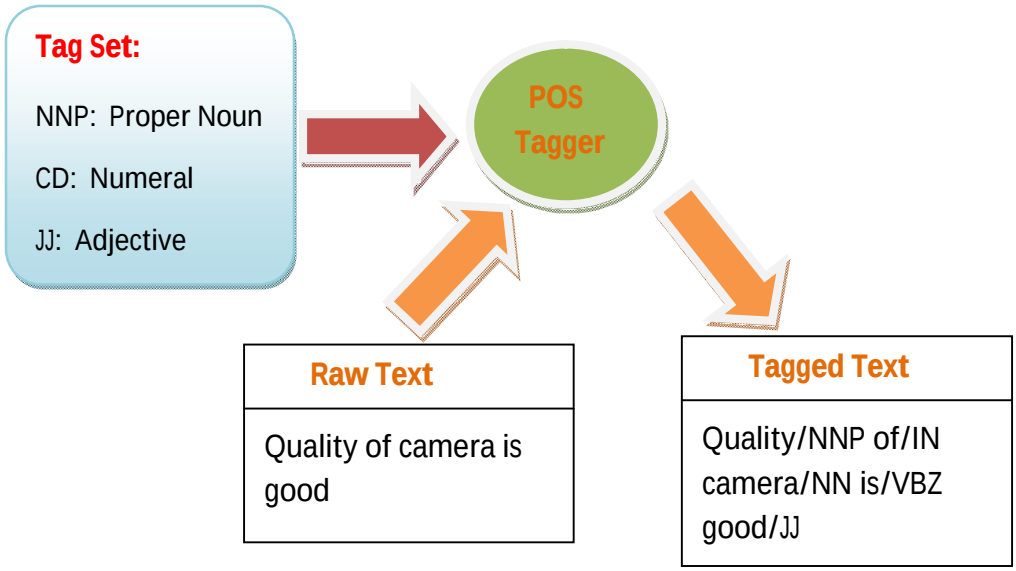


Figure 1 POS Tagging

2.3.2 Estimate the semantic orientation of extracted phrase:

- Estimate the orientation of the extracted phrases using the Point wise Mutual Information (PMI) measure [30].
- PMI between 2 words, $word_1$ and $word_2$ can be defined as :

$$PMI(word_1, word_2) = \log_2 \left(\frac{P(word_1) \,\&\, word_2)}{P(word_1)\, P(word_2)} \right)$$

Here

1. P(word-1 & word-2) = Probability that both words occurs together.
2. P(word-1)*P(word-2) = Probability of co-occurrence of word1 and word 2, If both words are independent.
3. $\frac{P(word-1 \,\&\, word-2)}{P(word-1)*P(word-2)}$ = Degree of statistical dependence between words.
4. log = Gives information of presence of one word when we observe other.

- The semantic orientation of a given phrase is calculated by comparing its similarity to a positive reference word ("excellent") with its similarity to a negative reference word ("poor").
- More specifically, phrase is assigned a numerical rating by taking the mutual information between the given phrases and the word "excellent" and subtracting the mutual information between the given phrase and the word "poor".
- In addition to determining the direction of the phrase's semantic orientation (positive or negative, based on the sign of the rating), this numerical rating also indicates the strength of the semantic orientation (based on the magnitude of the number).
- The **Semantic Orientation** (SO) of a phrase is calculated as :

  SO(*phrase)* = **PMI**(*phrase,* "**excellent**") – **PMI**(*phrase*, "**poor**")

  When, SO is **+ve**: phrase is strongly associated with **excellent**.

  SO is **–ve**: phrase is strongly associated with **poor.**

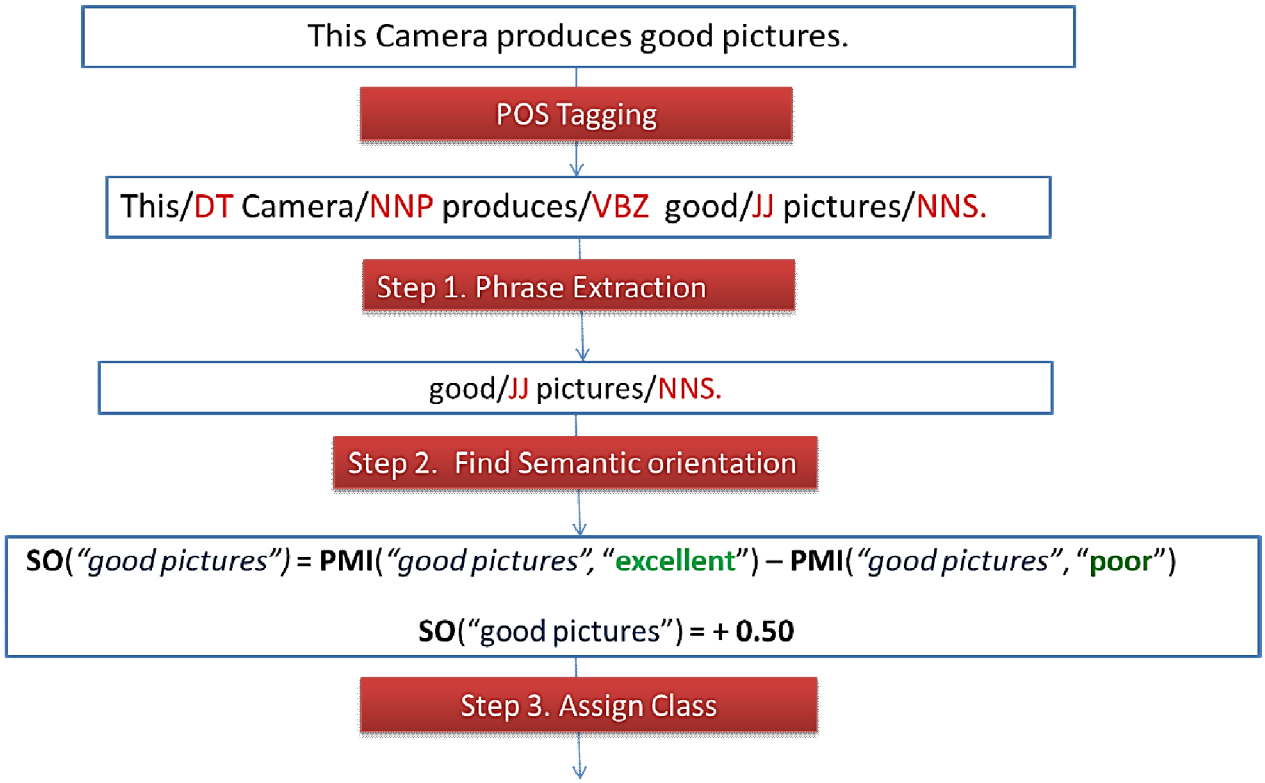
- The probabilities are calculated by issuing queries to a search engine and collecting the number of hits.

- **Proximity Search** –
    - Search the words such that they are located within 'n' words of one another.
    - YINDEX is search engine which allows Proximity search [15] using "**/n**" or "**NEAR**" operator.
    - Query such as "Good Camera **/10** Excellent" will find out the occurrence of "Good Camera" with the word "Excellent"
    - Based on the number of this returned by the search query, the PMI can be calculated as –

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

2.3.3 Assign the given review to a class:

- Calculate the average Semantic Orientation (SO) of the phrases present in the review text.
- Classify them as recommended or not recommended.
- If the average SO is greater than zero then it is Recommended or Positive review.
- If average SO is less than zero then it is not Recommended or Negative review.



Extracted Phrases	Part of speech tag	Semantic orientation	Class
Good Pictures	JJ NN	2.253	Recommended

Figure 2 Example of POS steps and find semantic orientation

2.2.3.1 Example of Review Classification:

Camera works great, takes nice photos even under dark conditions (using the flash).

It connects seamlessly to the web by wifi or 3G, and to my computer using the Bluetooth radio.

I set up my gmail, hotmail and yahoo accounts with no problem at all.

Nice screen to watch movies and get into facebook.

Figure 3 Input to Review Classification Engine

Table 3 Output of the classification for review

	Phrase	Semantic Orientation	Classification
Sentence:1	nice/JJ photos/NNS	0.7828604	(+) Positive phrase
	dark/JJ conditions/NNS	1.3126315	(+) Positive phrase
Sentence:2	bluetooth/JJ radio/NN	-0.50447255	(-) Negative phrase
Sentence:4	Nice/JJ screen/NN	0.905206	(+) Positive phrase

Average Semantic Orientation = 0.624056339263916

(+) POSITIVE Review!!!

2.4 Feature-based Sentiment Analysis:

Tasks involved in the Feature Based sentiment analysis [17]-

1. Identify Object Features.
2. Identify Opinion Words.

3. Handle Negation.
4. Handle But-Clauses.
5. Aggregation of Opinion.

2.4.1 Identification Object Feature:

a. Get the training data.
b. Create the rules based on the format of the sentence.
c. Apply the rule on training data set and calculate confidence.
d. If confidence is acceptable, use this rule on actual data set.

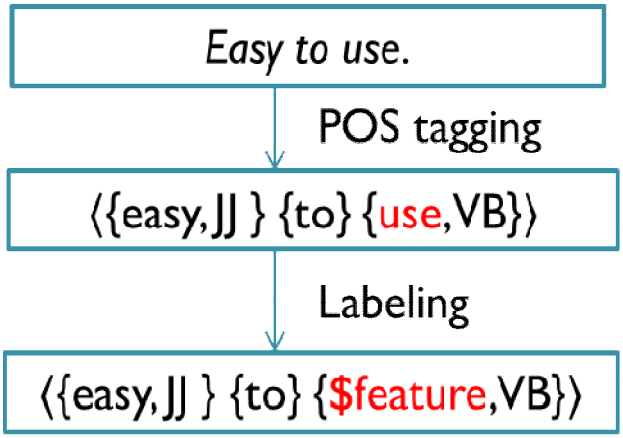


Figure 4 Feature Identification Process

2.4.2 Identify Opinion words:

e. Each positive opinion word is given Opinion Score = +1
f. Each negative opinion word is given Opinion Score= -1
g. Each context dependent word given, Opinion Score = 0

2.4.3 Handling Negation:

If negation is present in the sentence, then reverse the Opinion score of the sentence after the negative clause.

2.4.4 Handling But-Clause:

Opinion before “but” and after “but” are opposite. So adjust the scores of the opinion such that before and after the but clause, the opinion values are opposite.

2.4.5 Opinion Aggregation:

Based on the opinion about each feature in the review, determine the final orientation of the opinion on each object feature in the sentence.

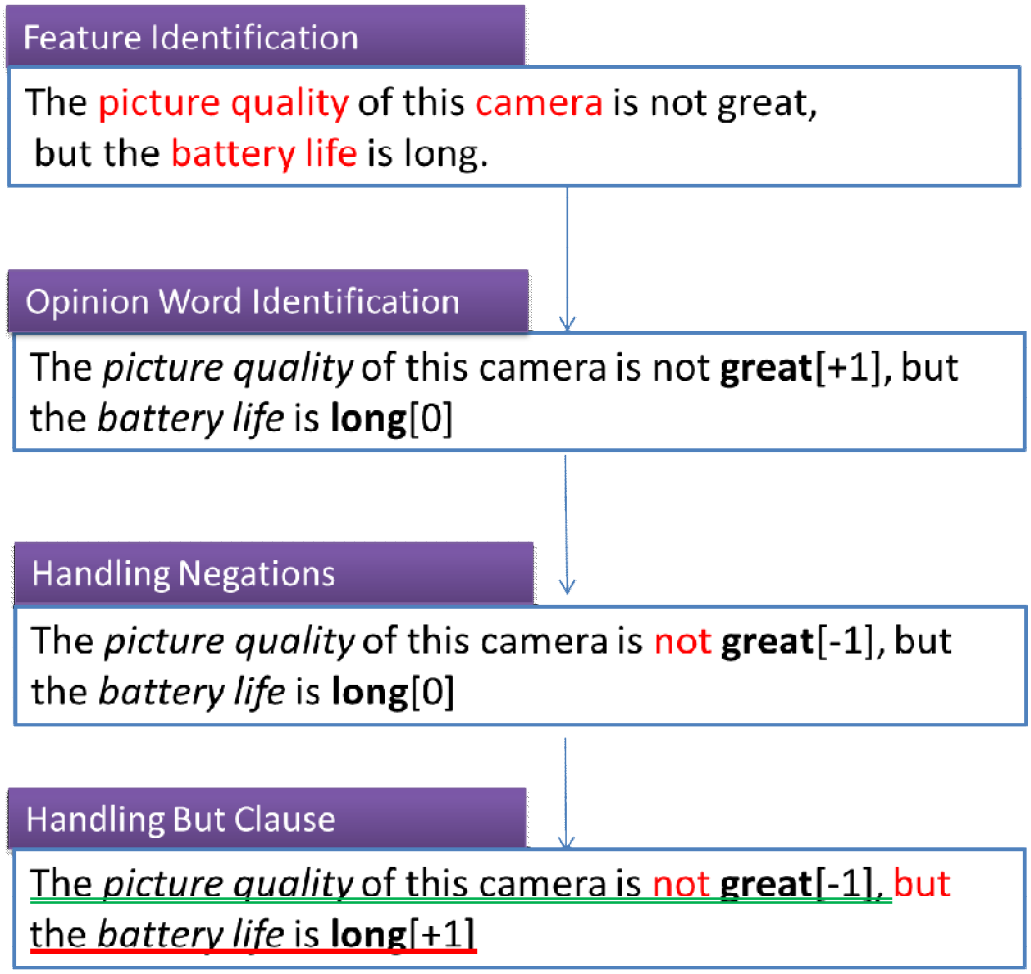


Figure 5 Example of Feature Based Sentiment Analysis

CHAPTER 3 FRAMEWORK FOR HANDLING NOISY REVIEWS

In the previous chapter opinion mining of the error free reviews is described. But, what if there are errors present in the reviews, in such cases we cannot identify nouns and adjectives easily, POS tagging would not work because the erroneous words are treated as Noun, for example –

Correct Sentence:

Quality of camera is good.

After POS tagging:

Quality/**NNP** of/IN camera/**NN** is/VBZ good/**JJ** ./.

Here the words {Quality, Camera} are tagged as Noun and {Good} is tagged as Adjective.

But if we consider the erroneous version of above text –

Erroneous Sentence:

Qlty of cam is gud.

After POS tagging:

Qlty/**NN** of/IN cam/**NN** is/VBZ gud/**NN** ./.

Observe that the words {Qlty, Cam} are tagged as Nouns because the tagger is not able to find these words in normal English words list, so they are treated as nouns. Also the word {gud} is treated as Noun, so we cannot identify the Features (Nouns) and Opinions (Adjective) directly in the given text.

So, there is a need of a mechanism by which we can map these words correctly to features and opinions to make opinion mining possible on such erroneous reviews.

In this section we proposed a framework for mining the erroneous reviews which could handle the spelling mistakes and use of shortcuts in the review text [16, 31, 32].

3.1 Preprocessing:

Opinion mining from erroneous reviews needs prior knowledge about the features and opinion words. To gather this information we need error free reviews, these reviews can be collected from experts and some trusted sites which are guaranteed to provide error free reviews. These reviews are then processed to extract features and opinion words. Preprocessing can be considered as normal opinion mining task performed on good dataset. Various steps involved in the opinion mining of error free reviews are described below

A. Indentify Feature Words

Generally features are nouns e.g. Screen, Battery. These features are specific for a particular product. So a list of features is maintained for every product. This task can be performed manually or automatically by finding the frequently occurring nouns in various reviews.

B. POS Tagging

Words in the review are tagged using Part of Speech (POS) tagger. POS tagger identifies nouns, adjectives, verbs, adverbs, co etc. in a given sentence. Once the review is tagged, the nouns are picked up and are searched in the list of features, if match is found then the sentence are marked as opinion sentence, meaning that the user had expressed some opinion about this feature in current sentence.

C. Opinion Word Extraction

It is observed that the adjectives represent opinion about a noun. In the POS tagged sentence, the adjectives are found out and for every feature present in the sentence, the nearest adjective is assigned as the opinion about that feature.

D. Determinig Opinion Orientation

Finding opinion orientation can be done using the Wordnet dictionary. Using Wordnet the orientation (positive or negative) is assigned to the opinion words). The process starts by using a seed set of opinion words and their orientation e.g. Good (opinion) and Positive (orientation), then the list of opinion is expanded by finding all synonyms of the opinion word, and all these

synonyms are assigned the orientation same as that of the original word, similarly the antonyms of the opinion word are found using Wordnet dictionary and are assigned opposite orientation. E.g. Synonyms of good are assigned Positive orientation and the Antonyms are assigned Negative orientation. This process is repeated for each new opinion word added in the opinion list, and it stops when no additional opinion word could be added.

When an adjective is found in the review, first it is searched in the opinion list, if its present then opinion same orientation is assigned, else the all synonyms of the adjective are searched in the opinion list and if found then same orientation is assigned. Otherwise the antonyms of the adjectives are searched in the opinion list and if found then the opposite orientation is assigned to the adjective.

E. Aggregation of Opinion

Once all the features, opinions and their orientation mentioned in the review are found out. Then for each feature number of positive and negative opinions is counted and if number of positive opinions is more than negative, then the opinion about the feature is considered as positive else it's marked as negative.

All these steps are applied over a dataset which is known to have no error. The reviews, opinion words, features are indexed for further use. A opinion list is created which stores the opinion word and opinion orientation, also a features list indicating various features is maintained.

The reviews are indexed for calculation of the document frequency as a part of similarity measure.

3.2 Prerequisite Data:

Before performing the opinion mining of erroneous reviews it is required to have below information ready, which could be obtained by process described above –

List of Features

Opinion Words and their orientation

Error free Reviews

Once this information is ready, the task is to parse the erroneous review and identify the nouns and adjectives and map them to features and opinions respectively.

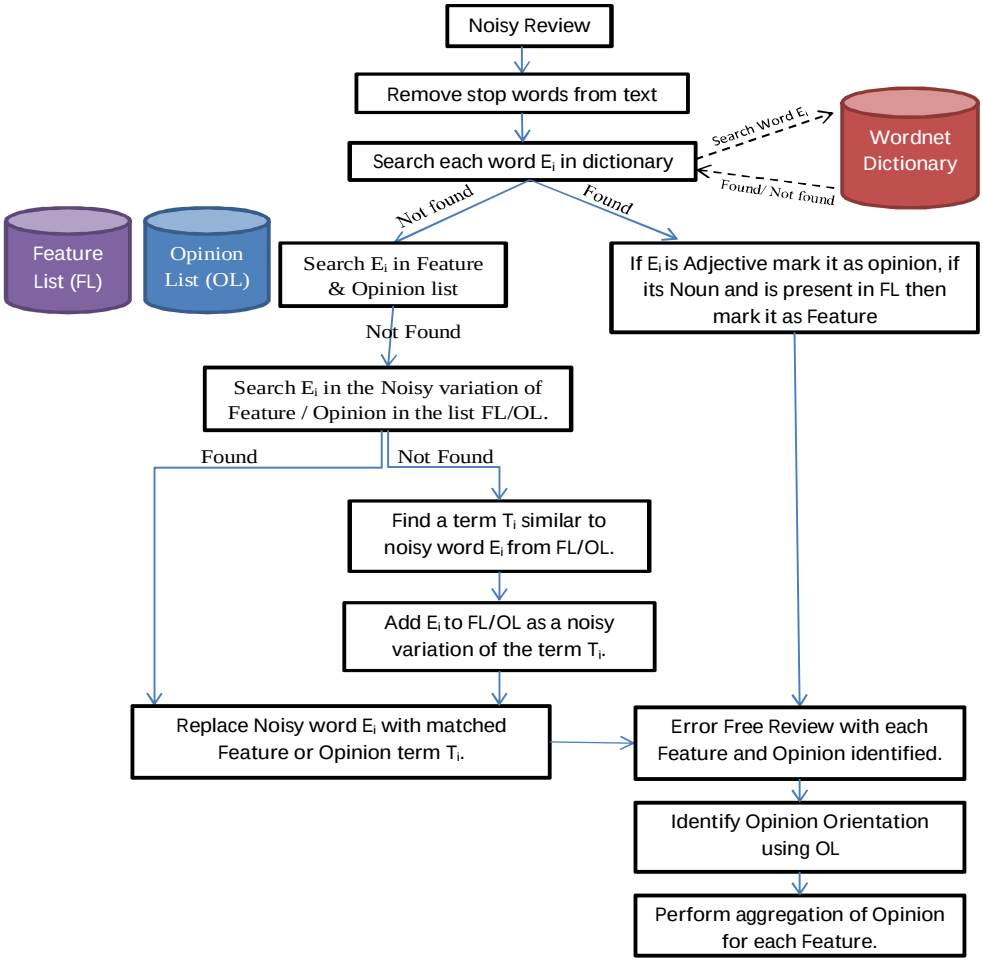


Figure 6 Framework for Handling Erroneous Reviews

3.3 Opinion mining process for Noisy reviews:

Figure 6.depicts the entire process of opinion mining for erroneous reviews. Various steps involved in this process are described below.

3.3.1 Stop word Removal:

In this step the stop words are removed from the review, when using shortcuts the stop words are generally represented by single letters (e.g. “d” is written instead of “the”, “2” is written

instead of “to”). Since we are not interested in the stop words, they are removed from the review.

3.3.2 Dictionary Search:

Each word E_i in the review is searched in the dictionary, to check if the word is in the correct form. If the word is found in the dictionary and it is an adjective then it is marked as opinion word.

3.3.3 Feature List Lookup:

When the word Ei is not found in the dictionary then or if the word is found in dictionary and it is noun then it is searched in the feature list and if exist then it is marked as feature. This step is required even if the word is found in dictionary because the features if a product might not be present in the dictionary.

3.3.4 Opinion Dictionary and Feature Dictionary Lookup:

When there is no direct match for E_i in dictionary or feature list, then we need to find a similar term present in the Feature list or Opinion list. It involves following steps.

3.3.4.1 List Creation:

As the number of opinions /feature terms is more, we create list of opinion/feature terms Ti having first character same as that of erroneous word and the edit distance between Ti and Ei is greater than a threshold. This step reduces the number of terms for which similarity should be calculated.

3.3.4.2 Find Similar Terms:

For each term T_i from above list, similarity is calculated using the formula described below - Similarity (W_1, W_2). If the similarity is above a predefined threshold then the term T_i is added in a set of matched terms.

3.3.4.3 Add noisy term to Opinion/Feature list:

Once similarity of all terms is calculated, the term (T_{max}) with highest similarity value is considered as a match. If T_{max} belongs to Feature list, then the erroneous word E_i is added to feature list as a noisy variation of the Feature.

Similarly if T_{max} belongs to Opinion list, then E_i is added as noisy variation of Opinion term T_{max}. E.g.

Suppose E_i= gud, and after calculating similarity we found that it matches with OPINION term T_{max} ="good", then we would add "gud" as a noisy variation of the Opinion "Good" in the opinion list.

In future these noisy variations are used for lookup. E.g. In future if we encounter the word "gud" then by searching the list containing noisy variations of opinion, we can map "gud" directly to opinion "good" – without calculating similarity.

3.3.4.4 Replace all occurrences of erroneous word:

Once a match T_{max} for E_i is found, all its occurrences in the current review and subsequent reviews are replaced by the T_{max}, to make the processing faster.

3.3.5 Similarity Calculation:

Similarity between two words W_1 (word from feature list / opinion list) and W_2 (word from review) is calculated as –

$$\text{Similarity}(W_1, W_2) = \frac{\text{Length of LCS}(W_1, W_2)}{\text{Length of } W_1} \times \frac{1}{\text{LED}(W_1, W_2)} \times \text{DF}(W_1)$$

Where, LCS = Longest Common Subsequence,

LED (W_1, W2) = Levenshtein Distance between two terms,

DF (W_1) - Document frequency of a term W_1.

The first part of the expression indicates what the percentage of common subsequence between two terms is. As the length of W_1 (Word in Feature List / Opinion List) is always more than the erroneous version W_2, therefore the length of LCS is divided by the length of W_1.
In the second part LED (W_1, W_2) indicates number of substitutions required to convert W_1 into W_2. We are interested in the Words which require fewer substitutions, so inverse of LED indicates that, lesser the edit distance, more similarity.
Finally the third part of expression DF (W_1) – represents the frequency of occurrence of word W_1 in the Error free Review dataset, we have assumed that the users generally use words (opinion and feature) which are more frequently used. E.g. Users generally speak about the features like {Flash, Picture Quality} and opinions {Good, Bad} more frequently.

3.3.6 Determining Opinion Orientation:

Once the opinion words and features are marked in the review, then the nearest opinion word is assigned to a feature in the review. And based on the average positive or negative orientation, we can conclude whether people like or dislike that particular feature.

3.4 Implementation

The feature list and opinion list along with their noisy variations are indexed using Lucene [19], which provides fast access to the search terms. Document frequency of each opinion and feature present in the list is calculated in preprocessing step and is also indexed using Lucene.
POS tagger like Stanford POS tagger [20] is used for the tagging of the review text. Output of tagger is available in XML and plain text format. We have used WordNet [21] dictionary for searching the words and also for the determination of the opinion orientation by finding synonyms and antonyms as described earlier. Implementation of the complete system is in progress.

CHAPTER 4 EXPERIMENTAL RESULTS

4.1 Review Data Set:

4.1.1 Noisy Reviews for Car:

Table 4 Reviews for Car

	Review	Expected			System Detected		
		Features	Opinion	Orientation	Features	Opinion	Orientation
1	pickup, I ffel 1.4 CRDi engn is hvng gud pickup fr Indian rods.	pickup	Good	+	Pickup	Gud	+
		engine	Good	+	Engn	Gud	+
2	Mileage of ths car is gud. I ffel 16/19 Kmpl In city/highway.	Mileage	Good	+	mileage	Gud	+
3	Exclint handlng and braking systm suprb extrior nd clasy intriors,rumy passenger cabin.	handling	excellent	+	handlng	Exclint	+
		braking	excellent	+	Brakng	Exclint	+
4	The acclrates systm is vry gud of i20 car. it caught 100 mrk in just few sec nd d gear system of d car is very smuth.	gear	smooth	+	gear	Smuth	_
		accelerate	Good	+	Not found	Not found	---------
5	D i20 car luks cerinly stylsh with its desgn, siting, boot nd engn vry nois free	engine	Noise free	+	engn	Nois free	+
		looks	Stylish	+	Not found	Not found	--------
6	The sunruf was so stylish, Sinc dis is a new featurs, evrybdy wnts to have a luk onc it is opn. On the higways and in wintr seasn, it is vry plesurble.	sunroof	Good	+	sunruf	Gud	+
7	safty fetures lik airbags,abs,collapsible stering Fetures lik automatic climate cntrl.	Safety	Like	+	Safty	Lik	-
8	I got 16.5 km milage rite frm d vry frst month isn't dat amzing. Gud cntrl, powerful headlites n brks mak ur jurney safe.	Mileage	Very	+	Milage	Vry	+
		brakes	powerful	+	Brks	Powerful	+
9	My dzire gives me gud milage n ts a Powerful vcle in all terms.	Mileage	Good	+	Milage	Gud	+
10	Luk nd Styl, New Model is relly impres2ive, d Desgn for Frnt Mud geard looks gud, and lamps Cmfrt.	Looks	Stylish	+	Luk	Styl	_

4.1.2 Noisy Reviews for mobile:

Table 5 Reviews for Mobile

	Review	Expected			System Detected		
		Features	Opinion	Orientation	Features	Opinion	Orientation
1	5.3 nch supr scrn of d fone , Games n vdios luk brillnt on d 5.3 nch scrn .	Screen	Super	+	Scrn	supr	-
2	ths mobile fone hv not gud otofocus but brigt led flsh.	Autofocus	not good	-	not found	-------	----------
		Flash	Bright	+	Flsh	brigt	-
3	in d fone a dual cor 1.4 ghz procsor vry fst. 16 gb inbult memry a micro SD card slot capbilites.	Processor	Fast	+	Procsor	fst	+
		Memory	Inbuilt	+	Memry	inbult	+
4	Betry lif is brllnt, it goes 3 dys if u stik to non dmandng activtis such as fone cals n web browsing.	Battery	Brilliant	+	Betry	brllnt	+
5	this fone luks styls , 8 mp riar cam nd 2 mp frnt cam r so powerfull.	Looks	Stylish	+	Luks	styls	+
		Camera	Power full	+	Cam	frnt	+
					Cam	riar	+
6	saund qulty is d bast of d fone , when we r listn songs fil gud.	Sound	Best	+	Saund	bast	+
7	Voice recognitn is not bttr dn I xpcted n 4 evrydy wordsis d fstest metod of txt ntry. So my stylus sits unusd.	voice recognitio n	not good	-	not found	--------	--------
8	D fone fit easly in all m pants pockets bcoze d size of d fone cmfrtable . scrn is bg nd butyful.	Size	comforta ble	+	Size	cmfrtabl e	+
		Screen	Beautiful	+	Scrn	butyful	-
9	a fone clck app is gr8 which come bith d fone . d clck allows a pre alrm of natre saond to help wake up.	Clock	Great	+	Clck	gr8	+
10	scrn exllent d prfect size of rading email, web brousing nd especially gr8 if u use it to read wrk relatd attachment.	Screen	Excellent	+	Scrn	exllent	+
11	wtching vdeo simply awsm reslutin nd vibrancy etc. I hv got a fw hd vdeo which I hv wtched on it nd it is awsm.	Video	Awesom e	+	Vdeo	awsm	+
12	D rotation of d scrn is quck, opning pgm is quck also on a side note d drag down bar on d top hs a quck btton for lcking scrn rotation which I thik is gnius.	Screen	Genius	+	Sern	gnius	-

4.1.3 Noisy Reviews for Camera:

Table 6 Reviews for Camera

	Review	Expected			System Detected		
		Features	Opinion	Orientation	Features	Opinion	Orientation
1	I own 2 d SX40hs d qality of d foto's r mch sharper widtru 2 life clrs, d biggest imprvment I c is in low lite.	Photo	Sharper	+	Not found		----
2	D new Canon DIGIC 5 emage processor n a hi sensitivity, 12.1 Megapxl CMOS sensor dropeddwn 4rm 14 megapixl in d SX30IS.	Image	sensitivity	+	Emage	New	+
		Megapixel	sensory	+	Megapxl	seonsory	_
3	Body construct n is very chep plastic type constructn .	Body	Cheap	+	Body	Chep	_
4	D btery hs gud lify , it's able to tke 400 fhotos at d high resoltion.	Battery	Good	+	Btery	gud	+
5	D smrt-otorealy works gr8 n selects d perfect seting 4 wht u r shuting.	auto really	Great	+	Otorealy	Gr8	+
6	D Zum is spectacular, It wil pick up evry little dtail, like u r rite in frnt of it.	Zoom	Spectacular	+	Zum	spectcular	+
7	d color is gr8 n d 1080p video is clr.	Color	Great	+	Color	gr8	+
		Video	Clear	+	Not found	Not found	--------
8	Bed lenc cover, Der is no tite grip of d cover 2 d camra body.	Lens cover	Bad	-	lenc cover	bed	-
9	LCD D t2i`s lcd scrn is amazng. Vry vibrnt n lots of pixl. Much bttr tn XSI`s. I`d recmnd getng n lcd protctr n cut it out 2 fit d LCD scrn.	Screen	Amazing	+	Scrn	amazng	+
					Scrn	fit	-
10	softwre D main thng I use is DPP 2 proces RAW. One majr fault wid it is t8 u need 1024x768 scren 2 instal run it. As a reslt, it won't run on my notbuk unless I attach a hdtv or extrnl monitr 2 it.	Software	Main	+	Software	main	-
		Screen	Need	-	Screen	need	-
11	gud pic qlity nd cam hs ligt wight.	picture	Good	+	Pic	gud	+
		weight	Light	+	Wight	ligt	-

4.2 Results:

Tables below represent the expected features and system driven output features of different - different brands. Correct Opinion give details of the entire possible outcome based on the matches found in section 4.1 when compared with both, the expected and system generated feature, respectively.

4.2.1 Compared result (Expected & Detected) for feature of Car:

Table 7 Result on Reviews of Car

# Review	Total Expected (Feature)	System Detected (Feature)	Correct Opinion
1	2	2	2
2	1	1	1
3	2	2	2
4	2	1	0
5	2	1	1
6	1	1	1
7	1	1	0
8	2	2	2
9	1	1	1
10	1	1	0
Total	15	13	10

In Table 7, it is clearly mentioned that from 10 reviews, total expected features are 15 and our system identified 13 features. Out of which 10 opinions are correctly identified for features extracted from expected and system.

4.2.2 Compared result (Expected & Detected) for feature of Camera:

Table 8 Result on Reviews of Camera

# Review	Total Expected (Feature)	System Detected (Feature)	Correct Opinion
1	1	0	0
2	2	2	1
3	1	1	0
4	1	1	1
5	1	1	1
6	1	1	1
7	2	1	1
8	1	1	1
9	1	2	1
10	2	2	1
11	2	2	1
Total	15	14	9

In Table 8, it is clearly mentioned that from 11 reviews, total expected features are 15 and our system identified 14 features. Out of which 9 opinions are correctly identified for features extracted from expected and system.

4.2.3 Compared result (Expected & Detected) for feature of Mobile:

Table 9 Result on Reviews of Mobile

# Review	Total Expected (Feature)	System Detected (Feature)	Correct Opinion
1	1	1	0
2	2	1	0
3	2	2	2
4	1	1	1
5	2	3	3
6	1	1	1
7	1	0	0
8	2	2	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	0
Total	16	15	11

In Table 9, it is clearly mentioned that from 12 reviews, total expected features are 16 and our system identified 15 features. Out of which 11 opinions are correctly identified for features extracted from expected and system.

4.3 Overall Result:

4.3.1 Statistical View:

In Table 10, cumulative data of each product has been aggregated in one data set, which will be used for calculating the accuracy and performance of our system based on our proposed (methodology / algorithm / framework) in (next)section 4.3.2.

Table 10 Generic / Overall Result

# Review	Total Expected (Feature)	System Detected (Feature)	Correct Opinion
Car	15	13	10
Camera	15	14	9
Mobile	16	15	11

4.3.2 Graph of overall Reviews:

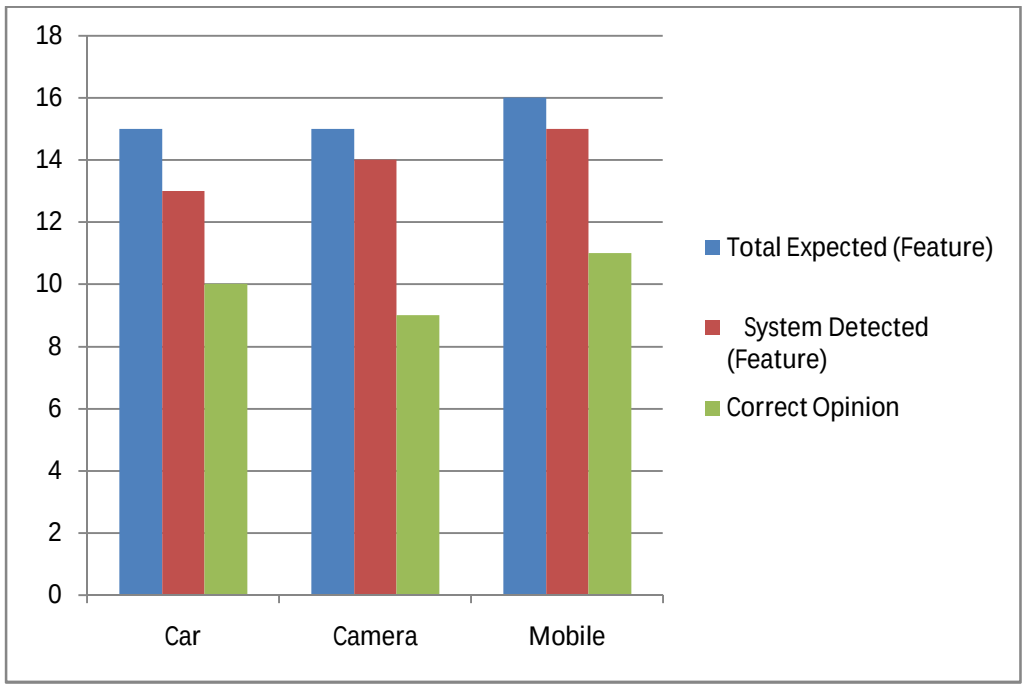


Figure 7 System Result

The above chart depicts the feature on the basis of opinions of each unit or class. From the above results we represent the performance of our proposed system in noisy review scenario.

We analyzed the system accuracy on the basis of opinions of three items which are car, camera and mobile. Along with parameters i.e., the total expected and system detected feature. We have a parameter named correct opinion (which is exactly calculating the overall efficiency based on the accuracy of our system) i.e., the correct matches which our system made to achieve accuracy of feature extraction on expected level, respectively.

CHAPTER 5 FUTURE WORK & CONCLUSION

5.1 Future Work:

In the current framework there is no way to distinguish between noisy/erroneous review and a correct review, so in future we will work on mechanism which would be able to differentiate them. For this purpose, we will append one more segregate function just above our proposed framework. This will help in reducing the time for redundant analysis of correct review. Also, the formula for calculating similarity can also be optimized further to improve the accuracy of the system. Thus as a future prospect, this classification will help in comparing two or more product based on their review submitted, the framework could be proposed for phonetic words (eg. photo - foto, kernel - colonel, seller – cellear, tea – t – tee, sea – c – see) and Neutral opinions, respectively.

5.2 Conclusion:

In this thesis we have represented a solution to take care of noisy reviews and formulated a similarity measure to identify a match between noisy word and the correct word. This framework can handle the opinion mining of reviews from social networking sites, where percentage of noise is more. Firstly, we mentioned our problem statement and also some related work. Secondly, we define concepts of Data Mining and underlying trend. Finally, we proposed our framework for handling noisy data to get correct opinion. Our results show the accuracy of predicting the correct opinion by our proposed system is tangible. Though segregation between correct review and noisy review could not be made out yet the overall performance depicted so far is very much lucrative.

PUBLICATION FROM THE THESIS:

Brahmraj Singh Rawat , Anwar Shaikh and Rajni Jindal. 2012. Handling Erroneous Reviews in Opinion Mining. In Proceeding of International Conference on Recent Trends of Computer Technology in Academia, Udaipur (India). 21-23 April 2012.

REFERENCES:

[1] Agrawal, R. and Srikant, R. "Fast algorithms for mining association rules." VLDB-94, 1994.

[2] Kumar, Santosh, Rukmani, "Implementation of Web Usage Mining Using APRIORI and FP Growth Algorithms". International Journal of Advanced Networking and Applications.1 (6). 2010 Pp. 400-404.

[3] J.Han, J.Pei, Y.Yin. "Mining frequent patterns without candidate generation" In: Proc. the 2000 ACM SIGMOD International Conference on Management of Data, pp. 1-12, (2000)

[4] Cheung W, Zaïane "Incremental mining of frequent patterns without candidate generation or support constraint[C]". In: *Proc IDEAS* 2003. Hongkong, China, 2003. 111-116.

[5] Leung, C. Khan and Hoque, T. CanTree (2005). "A Tree Structure for Efficient Incremental Mining of Frequent Patterns Proceedings" in the Fifth IEEE International Conference on Data Mining.

[6] Ming, Z. Taiyong, W. "Improved Pattern Tree for Incremental Frequent-Pattern Mining" Springer-Verlag Berlin Heidelberg. 2010 pp. 129-134

[7] Danish khan "CAKE – Classifying, Associating & Knowledge Discovery an Approach for Distributed Data Mining (DDM) Using Parallel Data Mining Agents (PADMAs)", IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. 2008

[8] Yan Chen1, Ming Yang2, Lin Zhang3, "General Data Mining Model System Based on Sample Data Division", Second International Symposium on Knowledge Acquisition and Modeling.2009

[9] Web 2.0 - http://en.wikipedia.org/wiki/Web_2.0

[10] http://en.wikipedia.org/wiki/List_of_social_networking_websites

[11] http://www.webopedia.com/quick_ref/textmessageabbreviations.asp

[12] M. Tavosanis, "A Causal Classification of Orthography Errors in Web Texts," AND 2007, pp. 99-106.

[13] http://en.wikipedia.org/wiki/Leet

[14] B. Pang, and L. Lee, "Opinion Mining and Sentiment Analysis," Foundations and Trends in Information Retrieval, vol. 2, pp. 1-135, 2008.

[15] B. Liu, "Sentiment Analysis and Subjectivity," Handbook of Natural Language Processing, Second Edition, 2010.

[16] L. Dey, and M. Haque, "Opinion mining from noisy text data," Springer IJDAR (2009) 12, pp. 205-226.

[17] M. Hu, and B. Liu, "Mining and Summarizing Customer Reviews", KDD' 04. USA, August 2004.

[18] G.Kothari, S. Negi, T. A. Faruquie, V. T. Chakaravarthy, and L. V. Subramaniam, "SMS based Interface for FAQ Retrieval," 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pp. 852–860, August-2009.

[19] Apache Lucene - lucene.apache.org

[20] Stanford POS tagger – http://nlp.stanford.edu/software/tagger.shtml

[21] Princeton Wordnet Dictionary - http://wordnet.princeton.edu/

[22] M. Gamon, "Sentiment classification on customer feedback data: Noisy data, large feature vectors, and the role of linguistic analysis," Proceedings of the International Conference on Computational Linguistics (COLING), 2004.

[23] Pavel, S, Using WordNet for opinion mining. In: Proceedings of the Third InternationalWordNet Conference, GWC2006, pp. 333–335. Brno, CZ (2006).

[24] Huifeng Tang, Songbo Tan, Xueqi Cheng, "A survey on sentiment detection of reviews", Chinese Academy of Sciences, Beijing 100080, PR China, 2009 Published by Elsevier Ltd.

[25] Jaap Kamps, Maarten Marx, Robert J. Mokken, Maarten de Rijke, "Using WordNet to Measure Semantic Orientations of Adjectives", this research was supported by the Netherlands Organization for Scientific Research (NWO) under projects 400-20-036, 612-13-001, 365-20-005, 612.069.006, 612.000.106, 220-80-001.

[26] E. Riloff, S. Patwardhan, and J. Wiebe, "Feature subsumption for opinion analysis," Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2006.

[27] B. Liu, Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer, 2006.

[28] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," Proceedings of ICML, pp. 282–289, 2001.

[29] B. Liu, M. Hu, and J. Cheng, "Opinion observer: Analyzing and comparing opinions on the web," by the International World Wide Web Conference Committee (IW3C2)., May 10-14, 2005.

[30] G. Qiu, B. Liu, J. Bu and C. Chen. Expanding Domain Sentiment Lexicon through Double Propagation, International Joint Conference on Artificial Intelligence (IJCAI-09), 2009.

[31] http://en.wikipedia.org/wiki/Noisy_text_analytics.

[32] Phani gadde, L.V. subramaniam, Tanveer A. Faruquie, "Adapting a WSJ trained Part-of-Speech tagger to Noisy Text" J-MOCR-AND '11 Beijing, China Copyright 2011 ACM 978-1-4503-0685.

Appendix – A

Code for standard reviews

```
package opinionMining;
import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import Indexer.IndexSearch;
import Indexer.LuceneIndexer;
import edu.stanford.nlp.ling.HasWord;
import edu.stanford.nlp.ling.Sentence;
import edu.stanford.nlp.ling.TaggedWord;
import edu.stanford.nlp.tagger.maxent.MaxentTagger;
public class StandardReview {
	String reviewID,heading,text,pros,cons,source,date;
	MaxentTagger tagger;
	IndexSearch searchingFeature;
	int OP_DIST_THRESHOLD;
	LuceneIndexer tempResultIndexer;
	IndexSearch searchingOrientation=new
IndexSearch(System.getProperty("ORIENTATION_INDEX"));
	private boolean debug=true;
	public StandardReview() throws IOException, ClassNotFoundException
	{	//create tagger object
	 tagger = new MaxentTagger(System.getProperty("TAGGER_FILE"));
	 searchingFeature=new IndexSearch(System.getProperty("FEATURE_INDEX"));
	OP_DIST_THRESHOLD=Integer.parseInt(System.getProperty("OPINION_DISTANCE_THRE
SHOLD"));
		//create index for feature
		tempResultIndexer=new LuceneIndexer(System.getProperty("TEMP_INDEX"))


	}
```

```
public void storeReview(String reviewID2,String heading2,String text2,String pros2, String
cons2, String source2, String date2)
        {       reviewID=reviewID2;
                heading=heading2;
                text=text2;
                pros=pros2;
                cons=cons2;
                source=source2;
                date=date2;
        }       //pos tagging of the text
        public void processReview() throws ParseException, IOException
        {       //separate into sentences
  List<List<HasWord>> sentences = MaxentTagger.tokenizeText(new StringReader(text));
           for (List<HasWord> sentence : sentences)
                   {    extractOpinionWords(sentence);
                }        //end of for loop - outer
        }               //method to close index- called by XML reader
                        public void opimizeAndCloseIndex()
                {
                        try {
                        this.tempResultIndexer.OptimizeAndClose();
                }       catch (Exception e) {

                        e.printStackTrace();
                } }
        void extractOpinionWords(List<HasWord> sentence) throws ParseException,
IOException
        {

                 ArrayList<TaggedWord> tSentence = tagger.tagSentence(sentence);
                 ArrayList<Integer> featureLocation=new ArrayList<Integer>();
                 ArrayList<Integer> opinionLocation=new ArrayList<Integer>();
                 ArrayList<Boolean> opinionNegation=new ArrayList<Boolean>();
                 ArrayList<Integer> IN_CC_Location=new ArrayList<Integer>();
                 ArrayList<Integer> CC_Location=new ArrayList<Integer>();
                 ArrayList<Integer> NN_Location=new ArrayList<Integer>();
                 ArrayList<Integer> nounOpinion=new ArrayList<Integer>();
                 ArrayList<Integer> featureOpinion=new ArrayList<Integer>();
                 //add the lowest starting point of sentence
```

```
                IN_CC_Location.add(-1);
System.out.println("\n\n------------------------------POS TAGGED TEXT------------------------------");

        System.out.println(Sentence.listToString(tSentence, false));
        int i;
        Boolean negation=false;
        for(i=0;i<tSentence.size();i++)
        {       //remember the position of NOT
if(tSentence.get(i).value().equalsIgnoreCase("NOT")||tSentence.get(i).value().equalsIgnoreCase("YET")|
|tSentence.get(i).value().equalsIgnoreCase("NO"))

                {
                        negation=true;
                }
  //if verb/adverb(JJ) follows not then also the NOT is valid, otherwise reset the position of NOT
  //else if(! tSentence.get(i).tag().contains("RB") && !tSentence.get(i).tag().contains("JJ"))

{               else if( !tSentence.get(i).tag().contains("JJ")){
                        negation=false;
                }       //search the NOUN
                        if(tSentence.get(i).tag().contains("NN"))
                {       NN_Location.add(i);
                        String singular=tSentence.get(i).value();
                        if(tSentence.get(i).tag().contains("NNS"))
                        {
                                //convert to singular
                                String plural=tSentence.get(i).value();
                                singular=plural.substring(0,plural.length()-1);
                                tSentence.set(i, new TaggedWord(singular,"NNS"));
                        }       //search the NOUN in feature list
                       if(searchingFeature.searchFeature("Feature:"+singular))
                       {        //feature found
                                featureLocation.add(i);
                       }
                     }          //search Adjective
                                else if(tSentence.get(i).tag().contains("JJ"))
                    {           //store adjective as opinion
                                opinionLocation.add(i);
                       //store the opinion negation flag
```

```
                opinionNegation.add(negation);

            }

            else if(tSentence.get(i).tag().equals("CC") )

            {       //store Coordinate conjunction position

                IN_CC_Location.add(i);
                CC_Location.add(i);
            }
            else if(tSentence.get(i).tag().equals("IN"))

            {       //store Subordinate conjunction position
                IN_CC_Location.add(i);
            }
            else if(tSentence.get(i).tag().equals(",")||tSentence.get(i).tag().equals(":"))
            {
                IN_CC_Location.add(i);
            }
        }//end of for loop-sentence
        //add the end point of sentence
            IN_CC_Location.add(i);
        //reprint the sentence with feature
        if(debug)
        {
            if(featureLocation.size()>0)
            {


System.out.println("\n              **** FEATURES AND OPINION  ***              ");

                        for(i=0;i<tSentence.size();i++)
                    {
                        System.out.print(" "+i+":"+tSentence.get(i).value());
                        if(featureLocation.contains(i))
                        System.out.print("#FEATURE# ");
                        if(opinionLocation.contains(i))
                        System.out.print(" #OPINION# ");
                        if(IN_CC_Location.contains(i))
```

```
                        System.out.print(" #CONJ# ");
                    }
                    System.out.println("\nFeature:"+featureLocation);
                    System.out.println("Noun   :"+NN_Location);
                    System.out.println("Opinion:"+opinionLocation);
                    System.out.println("IN CC  :"+IN_CC_Location);
                    System.out.println("CC     :"+CC_Location);
                } //else

// System.out.println("\n                ****  NO FEATURES FOUND  ***                ");


}       //check if there are more than one features

        if(NN_Location.size()>0 && opinionLocation.size()>0)
        {       //for each noun
                int ccIndex=1;
                for(int nnIndex=0;nnIndex<NN_Location.size();nnIndex++)

                {       //proceed till we get the windows maximum point

                        while(NN_Location.get(nnIndex)>IN_CC_Location.get(ccIndex))
                       ccIndex++;
                        //set the window limits
                        int winMax=IN_CC_Location.get(ccIndex);
                        int winMin=IN_CC_Location.get(ccIndex-1);
                        int opIndex=0;
                        //find the opinion window start
  while(opIndex<opinionLocation.size() && opinionLocation.get(opIndex)<winMin)
                                        opIndex++;
        if(opIndex<opinionLocation.size() && opinionLocation.get(opIndex) < winMax)
                        {
                                int opAtMinDist=opIndex;
                int minDist= Math.abs(NN_Location.get(nnIndex)-opinionLocation.get(opIndex));
                                opIndex++;
                                //find the opinion at nearest distance
while(opIndex<opinionLocation.size() && opinionLocation.get(opIndex)<winMax)
                                {
 int dist = Math.abs(NN_Location.get(nnIndex)-opinionLocation.get(opIndex));
```

```
                                        if(dist<minDist)
                                        {
                                                opAtMinDist=opIndex;
                                                minDist=dist;
                                        }
                                                opIndex++;
                                }
         //assign the opinion if its within threshold
                if(minDist<=OP_DIST_THRESHOLD)
                nounOpinion.add(opAtMinDist);
                else
                nounOpinion.add(-1);
        }
        else
        { //no opinions in this window
                nounOpinion.add(-1);            }
                 }//end for loop

          //consider only features (not all nouns)
                if(System.getProperty("ALL_NOUNS").equalsIgnoreCase("FALSE"))
                {
                        System.err.println("Only frequent features");
                        for(int k=0;k<NN_Location.size();k++)
                        { //if noun is a feature
                                if( featureLocation.contains(NN_Location.get(k)))
                                {
                                featureOpinion.add(nounOpinion.get(k));
          } } }                 //print the noun - opinion pair
                                if(System.getProperty("ALL_NOUNS").equalsIgnoreCase("TRUE"))
                        {       for(i=0;i<nounOpinion.size();i++)
                 {
                                if(nounOpinion.get(i)>-1)
                                {
        String opinion=tSentence.get(opinionLocation.get(nounOpinion.get(i))).value();
        String orientation=searchingOrientation.searchOpinionOrientation("opinion:"+opinion);
//indexing each sentence with the opinions words and orientation
        tempResultIndexer.indexTempFeatureAndOpinionWords

        (reviewID,
```

```
 tSentence.get(NN_Location.get(i)).value(),
            opinion,
          opinionNegation.get(nounOpinion.get(i)).toString(),
            orientation              );
             if(debug)
          {
          System.out.print("\n"+tSentence.get(NN_Location.get(i))+"-->");
          System.out.print(opinion);
          //opinion is to be negated
          if(opinionNegation.get(nounOpinion.get(i)))
          System.out.print(" #NOT# ");
          System.out.print(orientation);
                                                  } }
                                    }  }

                   else
                   {          for(i=0;i<featureLocation.size();i++)
                   {         if(featureOpinion.get(i)>-1)
                   {
String opinion=tSentence.get(opinionLocation.get(featureOpinion.get(i))).value();
String orientation=searchingOrientation.searchOpinionOrientation("opinion:"+opinion);

 //indexing each sentence with the opinions words and orientation


tempResultIndexer.indexTempFeatureAndOpinionWords
                                                                  (reviewID,


tSentence.get(featureLocation.get(i)).value(),
                                                                   opinion,

opinionNegation.get(featureOpinion.get(i)).toString(),
                                                                   orientation
 );        }//end if condition size>0
            else
            {
            System.out.println("@@@@@ NO OPINION or FEATURES @@@@@");
            }}//end of function extract opinion
}
```

Appendix – B

Code for Similarity of reviews

```
package opinionMining;
 public class similarity {
        static boolean debug=false;
          public static double similarityMeasure(String s, String t) {
                s=s.toLowerCase();
                t=t.toLowerCase();
                if(s.length()==0||t.length()==0)
                return 0;
                 if(s.charAt(0) != t.charAt(0))
                        return 0.0;
                 else
                        return LCSRatio(s, t) / editDistanceSMS(s,t);
       }
              public static double LCSRatio(String s, String t) {
              int lcs_len = lcs(s,t);
              int tok_len = 1;
              if(s.length() > t.length())
              tok_len = s.length();
              else
                      tok_len = t.length();
                 double lcsRatio = (double) lcs_len / (double) tok_len;
                 if(debug)
                 System.out.println("The LCSRatio is " + lcsRatio);
                 return lcsRatio;                        }
          public static int editDistanceSMS(String s, String t) {
          int levDist = getLevenshteinDistance(constantSkelton(s), constantSkelton(t)) + 1;
          if(debug)
          System.out.println("The Edit Distance is " + levDist);
          return levDist;
           }      public static String constantSkelton(String s)
                   {  int i=0, j=0;
                    char[] cArray = s.toCharArray();
                    char[] tArray = new char[s.length()+1];
```

```
                for (j = 1; j < s.length(); j++) {
                 if (s.charAt(j-1) == s.charAt(j))
                 cArray[j-1] = (char) 27;

              }
for (j = 0; j < s.length(); j++) {
if ((s.charAt(j) == 'a') || (s.charAt(j) == 'e') || (s.charAt(j) == 'i') || (s.charAt(j) == 'o') ||
(s.charAt(j) == 'u')
|| (s.charAt(j) == 'A') || (s.charAt(j) == 'E') || (s.charAt(j) == 'I') || (s.charAt(j) == 'O') ||
(s.charAt(j) == 'U'))
cArray[j] = (char) 27;
}               i =0;

                for (j = 0; j < s.length(); j++) {
                 if (cArray[j] == (char) 27)
                 continue;
                 else  {
                        tArray[i] = cArray[j];
                        i++;
                } }
                tArray[i] = '\0';
                String t = String.copyValueOf(tArray, 0, i);
                return t;
        }
        public static int getLevenshteinDistance (String s, String t) {
                if (s == null || t == null) {
                throw new IllegalArgumentException("Strings must not be null");
                }
                int n = s.length(); // length of s
                int m = t.length(); // length of t
                if (n == 0) {
                return m;
                } else if (m == 0) {
                 return n;
                }       int p[] = new int[n+1]; //'previous' cost array, horizontally
                         int d[] = new int[n+1]; // cost array, horizontally
                         int _d[]; //placeholder to assist in swapping p and d
                         // indexes into strings s and t
                         int i; // iterates through s
                         int j; // iterates through t
```

```
        char t_j; // jth character of t
        int cost; // cost
        for (i = 0; i<=n; i++) {
         p[i] = i;
        }
        for (j = 1; j<=m; j++) {
        t_j = t.charAt(j-1);
        d[0] = j;
         for (i=1; i<=n; i++) {
          cost = s.charAt(i-1)==t_j ? 0 : 1;
 // minimum of cell to the left+1, to the top+1, diagonally left and up +cost
d[i] = Math.min(Math.min(d[i-1]+1, p[i]+1),  p[i-1]+cost);
          }
          // copy current distance counts to 'previous row' distance counts
          _d = p;    p = d;    d = _d;
        }
        // our last action in the above loop was to switch d and p, so p now
        // actually has the most recent cost counts
        return p[n];
      }          public static int lcs(String X, String Y)
                {         int i,j;
                          int n = X.length();
                                    int m = Y.length();
    /* initialize the n x m matrix B and C for dynamic programming

     * B[i][j] stores the directions, C[i][j] stores the length of LCS of
     * X[0..i-1] and Y[0..j-1]          */
    int[][] C = new int[n+1][m+1];
    int[][] B = new int[n+1][m+1];
    /* C[i][0] = 0 for 0<=i<=n */
    for (i = 0; i <= n; i++) {
       C[i][0] = 0;
    }
    /* C[0][j] = 0 for  0<=j<=m */
    for (j = 0; j <= m; j++) {
       C[0][j] = 0;     }
```

```
/* dynamic programming */

            for (i = 1; i <= n; i++) {
               for (j = 1; j <= m; j++) {
                  if (X.charAt(i-1) == Y.charAt(j-1)) {
                     C[i][j]=C[i-1][j-1]+1;
                    B[i][j]=1;  /* diagonal */

                  }
                  else if (C[i-1][j]>=C[i][j-1]) {
                     C[i][j]=C[i-1][j];
                    B[i][j] = 2;  /* down */

                  }

                  else {
                     C[i][j]=C[i][j-1];
                     B[i][j]=3;   /* forword */
                  }   }     }

            /* print out the result */
                if(debug){
                System.out.println("String X is " + X);
                System.out.println("String Y is " + Y);
                System.out.println("The length of LCS is " + C[n][m]);
                System.out.println("The LCS is " + lcs);
            }     return C[n][m];
          }
 /*  private static double getIDF(String domainTerm) {
// TODO Auto-generated method stub

          return 1;   }     */
          public static void main(String[] args) {
       String X = "estb";  /* String X */
       String Y = "establishment";    /* String Y */
       System.out.println("The Similarity Measure between "+ X+" and "+ Y+" is " +
       similarityMeasure(X,Y));
  } }
```

Appendix - C

Code for noisy reviews

```
package opinionMining;
import global.PROJECTParameters;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import Indexer.IndexSearch;
import Indexer.LuceneIndexer;
import wordnet.DICTIONARY;
public class NoisyReview {
	String reviewID,heading,text,pros,cons,source,date;
	IndexSearch searchingFeature,noisySearchingFeature;
	DICTIONARY dictionary;
	IndexSearch searchingOrientation=new
	IndexSearch(System.getProperty("ORIENTATION_INDEX"));
	IndexSearch noisySearchingOrientation=new
	IndexSearch(System.getProperty("NOISY_OPINION_INDEX"));
	int
OP_DIST_THRESHOLD=Integer.parseInt(System.getProperty("OPINION_DISTANCE_THRESHOLD
"));
LuceneIndexer noisyfeature=new
LuceneIndexer(System.getProperty("NOISY_FEATURE_INDEX"));
LuceneIndexer noisyOpinion=new
LuceneIndexer(System.getProperty("NOISY_OPINION_INDEX"));
	boolean debug=false;
	public NoisyReview() throws CorruptIndexException, IOException, ParseException
	{
		searchingFeature=new IndexSearch(System.getProperty("FEATURE_INDEX"));
		noisySearchingFeature=new
		IndexSearch(System.getProperty("NOISY_FEATURE_INDEX"));
		dictionary=new DICTIONARY();
	}
```

```
public void storeReview(String reviewID2,String heading2,String text2,String pros2, String
cons2, String source2, String date2)
        {       reviewID=reviewID2;
                heading=heading2;
                text=text2;
                pros=pros2;
                cons=cons2;
                source=source2;
                date=date2;
        }       //process complte noisy review
                public void processReview() throws ParseException, IOException
        {
                 ArrayList<Integer> featureLocation=new ArrayList<Integer>();
                 ArrayList<Integer> opinionLocation=new ArrayList<Integer>();
                 ArrayList<Boolean> opinionNegation=new ArrayList<Boolean>();
                 ArrayList<Integer> IN_CC_Location=new ArrayList<Integer>();
                 ArrayList<Integer> nounOpinion=new ArrayList<Integer>();
                 ArrayList<Integer> featureOpinion=new ArrayList<Integer>();
                        text=text.replaceAll("\\.", " FULLSTOP ");
                        text=text.replaceAll(",", " comma ");
                        text=text.replaceAll("!", " FULLSTOP ");
                //remove stop words
                String nonStopText=removeEnglishStopWords(text);

                nonStopText=nonStopText.replaceAll("  ", " ");

                if(debug)
                System.out.println("Stop words removed:"+nonStopText);
                String noisyWords[]=nonStopText.trim().split(" ");
                //System.out.println("Stop words removed:"+nonStopText.split(" "));
                String standardWords[]=noisyWords;
                boolean negation = false;
                 //add the lowest starting point of sentence
                 IN_CC_Location.add(-1);
                for(int i=0;i<noisyWords.length;i++)
                {       String word=noisyWords[i];
                        if(debug)

                        System.out.println("Processing - "+word+"#");
```

```
                String temp;

                 if(word.length()>1)

                //check if its negation

                if(isNegation(word))

                {
                        negation=true;
                }
                else if (isIN_CC (word))
                {
                        IN_CC_Location.add(i);
                        negation=false;
                }
                /* temporarily commented
                //search each word in dictionary

                //else if( ! dictionary.isInDictionary(word))
                //not in dictionary,

                */

        //Search Feature list
                        else if(searchingFeature.searchFeature("Feature:"+word))
                        {
        //found in feature list
                                featureLocation.add(i);
                                if(debug)
                                System.out.println("Found in FEATURE list");
                        }else if( !
searchingOrientation.searchOpinionOrientation("opinion:"+word).equals("NONE"))
                {       //found in opinion list
                        opinionLocation.add(i);
                        opinionNegation.add(negation);
                        negation=false;
                        if(debug)
                        System.out.println("Found in OPINION list");
                        }
```

```
                else if(
(temp=noisySearchingFeature.searchNoisyFeature("NoisyFeature:"+word))!=null)
                {       //found in noisy variation of feature
                        standardWords[i]=temp;
                        featureLocation.add(i);
                        if(debug)
                        System.out.println("Found in NOISY FEATURE list");
                }else if( !
(temp=searchingOrientation.searchNoisyOpinion("NoisyOpinion:"+word) ).equals("NONE"))
                {       //found as noisy variation of opinion
                        standardWords[i]=temp;
                        opinionLocation.add(i);
                        opinionNegation.add(negation);
                        negation=false;
                        if(debug)
                        System.out.println("Found in NOISY OPINION list");
                }else
                {       //find similar terms from the Feature list and opinion list
                        similarWord sword=getSimilarWord(word);
                        if(sword.score>0)
                        {//MATCH IS FOUND
                        standardWords[i]=sword.word;
                        if(sword.isFeature)
                        {//store it as feature
                        featureLocation.add(i);
                         //add to noisy Feature dictionary
                        noisyfeature.indexNoisyFeature(word, sword.word);

                        if(debug)
                        System.out.println("Found in SIMILAR FEATURE list");
                }else
                        {//store it as opinion
                        opinionLocation.add(i);
                        opinionNegation.add(negation);
                        negation=false;
                //add noisy opinion to list
                noisyOpinion.indexNoisyOpinion(word, sword.word);
                         If(debug)
```

```
        System.out.println("Found in SIMILAR OPINION list");
        }
} else
        {    if(debug)
             System.out.println("-- NOT found --");
                        }
                }//end if
                }//end for
         ArrayList<Integer> NN_Location = featureLocation;
         if(debug)
{        System.out.println("\nFeature:"+featureLocation);
         System.out.println("Noun   :"+NN_Location);
         System.out.println("Opinion:"+opinionLocation);
         System.out.println("Negation:"+opinionNegation);
         System.out.println("IN CC  :"+IN_CC_Location);
         }
/*****************************/

IN_CC_Location.add(noisyWords.length);
// Locate the features and opinion words
 //check if there are more than one features

 if(NN_Location.size()>0 && opinionLocation.size()>0)
 {       //for each noun
         int ccIndex=1;
         for(int nnIndex=0;nnIndex<NN_Location.size();nnIndex++)
         {       //proceed till we get the windows maximum point
                 while(NN_Location.get(nnIndex)>IN_CC_Location.get(ccIndex))
                                 ccIndex++;
                 //set the window limits

                 int winMax=IN_CC_Location.get(ccIndex);
                 int winMin=IN_CC_Location.get(ccIndex-1);
                 int opIndex=0;
                 //find the opinion window start

while(opIndex<opinionLocation.size() && opinionLocation.get(opIndex)<winMin)
                                 opIndex++;
 if(opIndex<opinionLocation.size() && opinionLocation.get(opIndex) < winMax)
```

```
                {
                        int opAtMinDist=opIndex;
            int minDist= Math.abs(NN_Location.get(nnIndex)-
           opinionLocation.get(opIndex));
            opIndex++;

                        //find the opinion at nearest distance

      while(opIndex<opinionLocation.size() && opinionLocation.get(opIndex)<winMax)
    {
      int dist = Math.abs(NN_Location.get(nnIndex)-opinionLocation.get(opIndex));
      if(dist<minDist)
    {  opAtMinDist=opIndex;
       minDist=dist;
                        }
       opIndex++;

                }

//no opinions in this window
                        nounOpinion.add(-1);
                                }
                    }//end for loop
            for(int k=0;k<NN_Location.size();k++)
    {       //if noun is a feature

                if( featureLocation.contains(NN_Location.get(k)))
            {
                featureOpinion.add(nounOpinion.get(k));
                }
            }

//print the noun - opinion pair
                 for(int i=0;i<featureLocation.size();i++)
                 {
                    if(featureOpinion.get(i)>-1)
                    {
                    System.out.print("\nReview# "+reviewID+"\t");
```

```
        String opinion=noisyWords[opinionLocation.get(featureOpinion.get(i))];
        String orientation=searchingOrientation.searchOpinionOrientation("opinion:"+opinion);

//indexing each sentence with the opinions words and orientation

        //if(debug)
                        {   //opinion is to be negated

        System.out.print(noisyWords[featureLocation.get(i)]+"\t");
        if(opinionNegation.get(featureOpinion.get(i)))
                {
                        System.out.print("NOT\t");
                         if(orientation.equals("+"))
                         orientation="-";
                         else
                         orientation="+";
                }else
                        System.out.print("\t");
                        System.out.print(opinion+"\t");
                        System.out.print(orientation);
                }
        }   }
         }//end if condition size>0
         else
         {
           System.out.println("@@@@@ NO OPINION or FEATURES @@@@@");
         }
        }//end function
        private similarWord getSimilarWord(String noisyWord) throws ParseException,
IOException {
                //get Features starting with same letter

                similarWord simFeature= searchingFeature.getSimilarFeature(noisyWord);
                similarWord simOpinion= searchingOrientation.getSimilarOpinion(noisyWord);
                if(simOpinion.score>simFeature.score)
                {       return simOpinion;
                }else
                {
```

```
                return simFeature;
        }
}
private boolean isIN_CC (String word) {
        // TODO Auto-generated method stub
        word=word.toLowerCase();
        String[] conjunctions={"and",//cc
                        "or",//cc
                        "but",//cc
                        "either",//cc
                        //IN------
                        "for",
                        "after",
                        "that",
                        "with",
                        "on",
                        "though",
                        "if",
                        "of",
                        ",",
                        "fullstop",
                        "comma"};
                for(int i=0;i<conjunctions.length;i++)
        {
                if(conjunctions[i].equals(word))
                        return true;
        };      return false;
}

private boolean isNegation(String word) {
        // TODO Auto-generated method stub
        word=word.toLowerCase();
                String[] negations={"not",
                        "yet",
                        "no",
                        "nor"};
```

```
		for(int i=0;i<negations.length;i++)
		{
			if(negations[i].equals(word))
				return true;
		}; return false;
}
public String removeEnglishStopWords(String text)
{
			// TODO Auto-generated method stub
			String[] stopWords={".a.",
					".a.",
					".b.",
					".c.",
					".d.",
					".e.",
					".f.",
					".g.",
					".h.",
					".i.",
					".j.",
					".k.",
					".l.",
					".m.",
					".n.",
					".o.",
					".p.",
					".q.",
					".r.",
					".s.",
					".t.",
					".u.",
					".v.",
					".w.",
					".x.",
					".y.",
					".z.",
					".0.",
					".1.",
```

```
".2.",
".3.",

".4.",
".5.",
".6.",
".7.",
".8.",
".9.",
".want.",
".about.",
".above.",
".after.",
".again.",
".against.",
".all.",
".am.",
".an.",
".and.",
".any.",
".are.",
".arent.",
".as.",
".at.",
".be.",
".because.",
".been.",
//".before.",
".being.",
".below.",
".between.",
".both.",
".but.",
".by.",
".cant.",
".cannot.",
".could.",
".couldn't.",
".couldnt.",
```

".did.",
".didn't.",
".didn.",
".do.",
".does.",
".doesn't.",
".doesnt.",
".doing.",
".dont.",
".down.",
".during.",
".each.",
".few.",
".for.",
".doesn't.",
".doesnt.",
".from.",
".further.",
".had.",
".hadn't.",
".hadnt.",
".has.",
".hasn't.",
".hasnt.",
".have.",
".haven't.",
".havent.",
".having.",
".he.",
".hed.",
".hell.",
".hes.",
".her.",
".here.",
".heres.",
".hers.",
".herself.",
".him.",

".himself.",
".his.",
".how.",
".hows.",
".i.",
".id.",
".ill.",
".im.",
".ive.",
".if.",
".in.",
".into.",
".is.",
".isn't.",
".isnt.",
".it.",
".its.",
".its.",
".itself.",
".lets.",
".me.",
".more.",
".much.",
".most.",
".mustn't.",
".mustnt.",
".my.",
".myself.",
//".no.",
".nor.",
//".not.",
".of.",
".off.",
".on.",
".once.",
".only.",
".or.",
".other.",

".ought.",
".our.",
".ours.",
".ourselves.",
".out.",
".over.",
".own.",
".same.",
".shan't.",
".shant.",
".she.",
".shed.",
".shell.",
".shes.",
".should.",
".shouldn't.",
".shouldnt.",
".so.",
".some.",
".such.",
".than.",
".that.",
".thats.",
".the.",
".their.",
".theirs.",
".them.",
".themselves.",
".then.",
".there.",
".theres.",
".these.",
".they.",
".theyd.",
".they'll.",
".theyll.",
".theyre.",
".theyve.",

".this.",
".those.",
".through.",
".to.",
".too.",
".under.",
".until.",
".up.",
".very.",
".was.",
".wasn't.",
".wasnt.",
".we.",
".wed.",
".well.",
".were.",

".weve.",
".were.",
".weren't.",
".werent.",
".what.",
".whats.",
".when.",
".whens.",
".where.",
".wheres.",
".which.",
".while.",
".who.",
".whos.",
".whom.",
".why.",
".did.",
".may.",
".will.",
".are.",
".be.",
".have.",

```
                                        ".do.",
                                        ".can.",
                                        ".might.",
                                        ".should."};

                        text="."+text+".";
                        text=text.replace(" ", ".");
                        text=text.toLowerCase();
                        for(int i=0;i<stopWords.length;i++)
                {       if(text.contains(stopWords[i]))
                        text = text.replace(stopWords[i], ".");
                        };
                        text=text.replace(".", " ");
                        //System.out.println("Removed stop words :"+faq);
                        return text;
                }
        public static void main(String args[]) throws CorruptIndexException, IOException,
ParseException
        {       PROJECTParameters.load();
                NoisyReview nrev=new NoisyReview();
                nrev.storeReview("1", "", "NO USER REPLACEABLE BATTERY, , Unless you buy the
extended warranty for $65 .", "", "", "", "");
                nrev.processReview();

                nrev.storeReview("2", "", "d dsgn is clvr.", "", "", "", "");
                nrev.processReview();


        }
}
```