2014

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

AN APPLICATION OF KNOWLEDGE DISCOVERY IN TEXTUAL DATABASES TO IDENTIFY SENTIMENTS IN PRODUCT REVIEWS

Vincent Tran

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2014

Approved by

Advisory Committee

Douglas Kline _____

Devon Simmonds _____

Curry Guinn
_____
Chair

Accepted By

_____
Dean, Graduate School

TABLES OF CONTENTS

ABSTRACT

An application of knowledge discovery in textual databases to identify sentiments in product reviews. Tran, Vincent, 2014. Master's Thesis, University of North Carolina Wilmington.

With the massive amount of textual data available on the web, the ability to automate the extraction of patterns and meaning proves to be important for business decision makers. The Amazon e-commerce site is particularly interesting because of their massive amount of readily available textual data in the form of reviews. These reviews are often loaded with sentiments that have already been tagged by the reviewers via the 5-stars rating system. This rating system defines 5-star reviews as having the most positive sentiment and 1-star reviews as having the most negative sentiment. However, the reviewers' ratings of a product lack the granularity required by manufacturers and business owners to answer the question: what do customers like and dislike about their products? This study explores the feasibility of two knowledge discovery tasks: topic identification and sentiment analysis in the domain of product reviews. Particularly, the study leverages supervised (Naïve Bayes and Support Vector Machine) and unsupervised machine learning techniques (Pointwise Mutual Information and Frequent Itemsets) to detect topics being talked about in the reviews and the overall sentiment towards those topics. The resulting data from the study's experiments suggest that we can answer the question "what do customers like and dislike about the products?" with reasonable accuracy using these particular supervised and unsupervised approaches.

CHAPTER 1: INTRODUCTION

1.1 Product Review

Customer reviews are collections of consumer feedback of products or services by consumers who had purchased the said product or services [1]. Prior to the advent of e-commerce, customer feedback often took the form of notes on a napkin, suggestion board or comment box [2]. This highly limited their visibility to the prospective customers and to the providers of the product or service. With the rapid expansion of e-commerce, the number of products sold online started to grow. In order to enhance customer experience, e-commerce services started to provide a facility for customers to write and share reviews of the product they had purchased [3]. The transparency of the internet combined with its ease of access through the personal computers placed the importance of customer reviews into the focus of good and service providers [3]. Customer reviews become more important when online supermarket sites like Amazon actively facilitate and encourage their users to write feedback on their purchased products. Amazon sends their customers email reminders to rate and review previously purchased products [4]. As a result, the number of reviews grows rapidly. Some popular Amazon products receive thousands of reviews. The length of a single review can surpass 250 words. The sheer volume and size of the available reviews actually become a hindrance. Customers simply will not read 2000 reviews on a product before they buy. A camera manufacturing company's executive cannot read thousands of customer reviews on hundreds of their products to find out which features are being talked about the most and why [3]. To be able to utilize the massive amount of review data out there, we must be able to identify the topics of each review and the sentiment of reviews and the topics. Ultimately, the goal is to build a system that can automate product reviews analysis to discover topics that are being talked about and the customer's sentiment towards those topics. When such

a system can operate reliably in the big data environment, on one hand, product manufacturer will gain a powerful tool to better understanding their customers' demands and feedbacks. With that understanding, manufacturers can go on and improve their products and ultimately increase revenue and customer satisfaction. On another hand, customers will have the ability to make informed buying decision more quickly without having to read hundreds if not thousands of product reviews. The system will ensure that all customer feedbacks are more effectively reaching the manufacturers and businesses.

1.2 Topic Identification

Topic identification (sometimes referred to as topic extraction or aspect extraction) is a key task of the aspect-based sentiment analysis framework. It is the process of assigning one or more label to a token of text (i.e. paragraphs, sentences). These labels can be chosen from a predefined list of topics or one is that is generated on the fly by clustering algorithms [5].

1.3 Sentiment Analysis

Sentiment analysis is the task of analyzing people's opinion, sentiments, evaluation, appraisals, attitudes, and emotions towards entities (i.e. products, services). Opinions are central to almost all human activities. In the business world, decision makers always want to know consumer or public opinions about their products and services [2] [6]. The application of sentiment analysis on product reviews seeks to fulfill this need. In the context of this study, *sentiment* is defined as the evaluation of the reviewer with respect to a sub-topic derived from the product reviews.

1.4 Hypothesis

To build a system for automated review analysis and do so tractably, it is important that the system is unsupervised. Unsupervised systems are systems built based on unsupervised machine learning techniques. Their supervised counterparts are build based on supervised machine learning techniques. Supervised systems are labor-expensive. In order for a supervised system to work, it needs training data. Training data are instances of reviews that have already been sentiment-classified and topic-identified by a human. Such supervised system needs a large amount of training data to achieve acceptable accuracy. One additional drawback of such system is that it is domain dependent. That means once the system has been trained on a dataset within certain domain, it can only correctly classify sentiment and detect topic on data within that domain. An unsupervised system does not require training data to classify sentiment and identifying topic. Unsupervised system can be used across different domains and therefore more tractable for the application of product review analysis. However, generally speaking, supervised systems perform better than unsupervised systems. This study will aim to investigate the current techniques and look ways to improve on this drawback.

This study explores the practicality of applying knowledge discovery techniques to identify feature specific sentiments inside of large databases of product reviews. This explorative study involves four sub-tasks: (1) First, we use the supervised learning algorithms Naïve Bayes and Support Vector Machines to accurately classify the sentiment of review sentences using domain-specific training data; (2) We use supervised learning algorithms Naïve Bayes and Support Vector Machines to accurately sort sentences into categories corresponding to pre-identified topics; (3) we use the unsupervised learning algorithm Pointwise Mutual Information with Semantic Orientation to accurately identify sentiment of review sentences with no training data. A word has

positive Semantic Orientation when it has stronger association to words with generally positive sentiment. A word has negative Semantic Orientation when it has stronger association to words with generally negative sentiment; (4) we use the unsupervised learning algorithm Apriori Frequent Itemset to accurately discover topics of the reviews and tag the review sentences with these topics; (5) we use supervised sentiment analysis approaches to classify sentiment of review sentences using out-of-domain training data.

To measure the performance of our hypotheses, we define classification accuracy as the ratio of true classification or prediction to the number of total instances. For a data set of *n* instances, the classification accuracy is defined as:

Classification accuracy = number of correctly classified instances / n

We define the precision rate as the proportion of classified instances that are true:

Precision = number of truly relevant instances that are classified as relevant / number of instances classified as relevant

Recall rate is defined as the proportion of true instances that are classified correctly:

Recall = number of truly relevant instances that are classified as relevant / number of truly relevant instances

To summarize, our hypotheses are as following:

(1) It is possible to classify the sentiment of review sentences at 50% accuracy or greater using supervised machine learning techniques
(2) It is possible to detect topics discussed in review sentences at 50% or greater precision and recall rate using supervised machine learning techniques

(3) It is possible to classify the sentiment of review sentences at 50% accuracy or greater using unsupervised machine learning techniques without training data

(4) It is possible to detect topics discussed in review sentences at 50% or greater precision and recall rate using unsupervised machine learning techniques without training data

(5) It is possible to classify sentiment of review sentence at 50% accuracy or greater using supervised machine learning techniques with out-of-domain training data

CHAPTER 2: REVIEW OF LITERATURE REVIEW AND ANALYSIS

This chapter discusses several studies with supervised and unsupervised approaches to topic identification and sentiment analysis. These studies are inspiration to this paper.

2.1 Product Review Analysis

In the recent years, the explosion of online supermarkets and review sites provide a massive amount of raw information. These goldmines of information are often untapped or under-utilized [7]. Due the massive quantity of the data, the traditional approach of reading the reviews and manually analyzing the content in it is tedious, unfeasible and sometimes even unreliable [8]. Here we introduce automated product review analysis.

The application of automated product review analysis goes well beyond consumer feedback to other consumers when the business and manufacturers start to use the reviews to improve or amend the designs of their products [8]. This transformation of raw data into useful information for business decision-making purpose is Business Intelligence [8] [9]. Automatic sentiment analysis of the textual reviews of a product may prove to be a useful form of Business Intelligence for manufacturers. Due to this new interest, there has been extensive academic and business research on automatic text analysis for sentiment [10]. The proposed methods however only try to extract the overall sentiment of the document [10]. While overall sentiment of a review is still useful, it does not provide enough granularity to for the Business Intelligence purposes. Product Manufactures need to know the consumer's sentiment on specific product features in order to amend and improve the design of their product.

In the context of Business Intelligence, product review analysis involves extracting the product's features or topics from the textual reviews, then analyzing and classifying the sentiment of those features in the context of the products [10].

2.2 Topic Identification

Topic identification involves the extracting of the product features from the reviews [11]. When reviewers comment on a product, they often target particular product aspects or features. Product features are attributes, or aspects of the products [12]. Although there are several proposed methods on how to approach this task, there are limited studies and experimentation on how effective these methods are [11]. One of these proposed methods involves clustering or grouping of features that are closely related [11]. Ideally, we want to accomplish feature clustering by unsupervised learning approaches [12]. That means given uncategorized reviews; we must cluster them by some measure of similarity. However, studies found that unsupervised learning approaches often do not perform well [12].

According to Mukherjee and Liu, 2012 feature identification task consists of two different sub-tasks [12]. First, they extract the feature terms from the raw text. Then, they group the synonymous feature terms into categories according to statistical distributional similarity [12]. Feature term extraction can often be as trivial as performing a word count on all nouns and selecting the most useful nouns in the corpus. To give an example, in [10] Yi simply defines useful nouns as ones preceded by the definite article "the."

The second sub-task, grouping synonym feature terms into appropriate categories, is a massively more complex problem [11]. The subjectivity of the task creates many great difficulties. Because the synonymy of the feature terms is often domain dependent, it is difficult to devise an

unsupervised learning algorithm to categorize feature terms [11]. For example, in the domain of movie reviews, the term ‘movie ‘and ‘picture’ are synonyms, however, in the domain of camera reviews, the term ‘picture’ is more likely to be synonymous to ‘photo’ while ‘movie’ would be a more appropriate synonym of ‘video’.

The key to feature terms grouping is the measure of similarity. There are two mainstream approaches. One relies on pre-existing knowledge resources such as WordNet and the other one relies on distributional properties of words in corpora [11].

As an example of the first approach, Liu et al proposed a method to group synonymous feature terms by using WordNet synsets. This approach does not take in account the word distribution information. This weakens the cohesiveness of the grouping [11]. Dictionaries like WordNet do not contain the domain specific knowledge required to group domain specific feature terms [11].

The second approach exploits the hypothesis that words with similar word senses tend to appear in similar positional contexts [11]. For example, to analyze the context surrounding the feature terms, Lee and Kim proposed and used collocation-based similarity measures such as Jaccard, Dice and Cosine [13] [14].

2.3 Sentiment Analysis

In general, most studies on sentiment analysis fall into one of the following three categories:

Document level analysis: At this level, we only care about the sentiment of the whole document. Given a particular product, the aim is to know the overall attitude of the reviewer toward this product [6].

Sentence level analysis: This level concerns with the sentiment of particular sentences inside a review. This level of analysis moves closer to the feature/entity level analysis. However, a sentence does not necessarily constitute a feature [6].

Feature/Aspect Level: Document level and sentence level analysis do not necessarily reveal what exactly people liked and disliked about the product. Feature level analysis provides the necessary granularity that makes this application of product review analysis useful for Business Intelligence purposes [6].

To determine the sentiment expressed on each feature of a product, we will review two main approaches: the supervised learning approach and the lexicon approach.

First, we explore the supervised learning approach. There are many mainstream literatures out there on how to go about tackling this task. Wei and Gulla proposed a hierarchical classification model [15]. One of which seems to be the gold standard: Naïve Bayes classifier [6].

By using the statistical Bayesian theorem, we can build a supervised learning classifier to detect the sentiment in a given sentence. Combined with the topic identification, we can identify the main topic of the sentence and thus, the sentiment of the topic or topics. Supervised learning is domain dependent. A model or classifier trained from labeled data of one domain will often perform poorer in another domain [6]. When the task requires a less domain-dependent approach (i.e. cross-domain classification), a lexicon-based approach may be proven more effective [2]

The lexicon-based approach is typically unsupervised. It uses sentiment lexicon, composite expressions, rules of opinions and sentence parse tree to determine sentiment orientation for each feature [6]. For the purpose of this study, we will not go in-depth into the lexicon-based approach.

2.4 Classification Algorithms and Tools

Dues to the subjective nature of the problem, to test our hypotheses we chose several machine learning algorithms from both the supervised learning and unsupervised learning camps.

2.4.1 Supervised Learning

In the supervised learning scenarios, the learner receives a set of labeled data as training data. This strategy allows the learner to infer unseen points, classifying unknown data according to models built by 'learning' the training data [16].

In supervised learning, otherwise known as "learning class by examples," the ultimate aim is to learn a mapping from the input to an output [17, p. 11]. The mapping is realized by the classifier by "learning" the training data provided by the "supervisor [17, p. 11]."

Supervised learning algorithms are most often associated with classification, regression, and ranking tasks [16]. For the purpose of this study, we will explore and compare the performance of two different supervised learning algorithms in performing classification tasks:

2.4.1.1 Naïve Bayes classifier

Despite numerous significantly creative and more elegant solutions in machine learning and classification algorithms, the Naïve Bayes classifier has retained its popularity in the classification class of problems. This popularity is mostly due to its comparable performance and accuracy and higher computational tractability when compared to other methods [18]. Additionally

comparing to other more elegant approaches, the learning process of Naïve Bayes is fast [18]. Unlike the classical Bayesian logic, the Naïve Bayes approach simply waives all dependencies. It assumes that the presence or absence of any conditions in the Bayesian network does not affect any other conditions [19]. For example, consider "cloud formation" and "wet sidewalk" as the two nodes, Naïve Bayes classifier approach considers these two events independent of each other even though they both have a probabilistic relationship with "raining." That is the "cloud formation" does not affect the probability of a "wet sidewalk."

This assumption simplifies the estimation of the class-conditional probabilities. Studies have shown the Naive Bayes classifiers' respectable performance in many domains, despite its simplicity and its apparent weakness in the independency assumption [19].

2.4.1.2 Support Vector Machine (SVM)

The Support Vector Machine is a relatively new supervised machine learning method used primarily for binary classification [20]. The training algorithm's goal is to learn classification and regression rules from data to form a classifier [21]. The algorithm builds a linear hyper-plane, which separate data into two classes. Using this model, the classifier predicts and assigns classes to new data points according to their positions on the plane [21]. Figure 1 retrieved from the scikit-learn Python machine learning package's documentation illustrates a linear-separable data set on a Support Vector Machine feature space [22]. To perform classification, a SVM places new data points in the feature space and classified according to the learned rules.

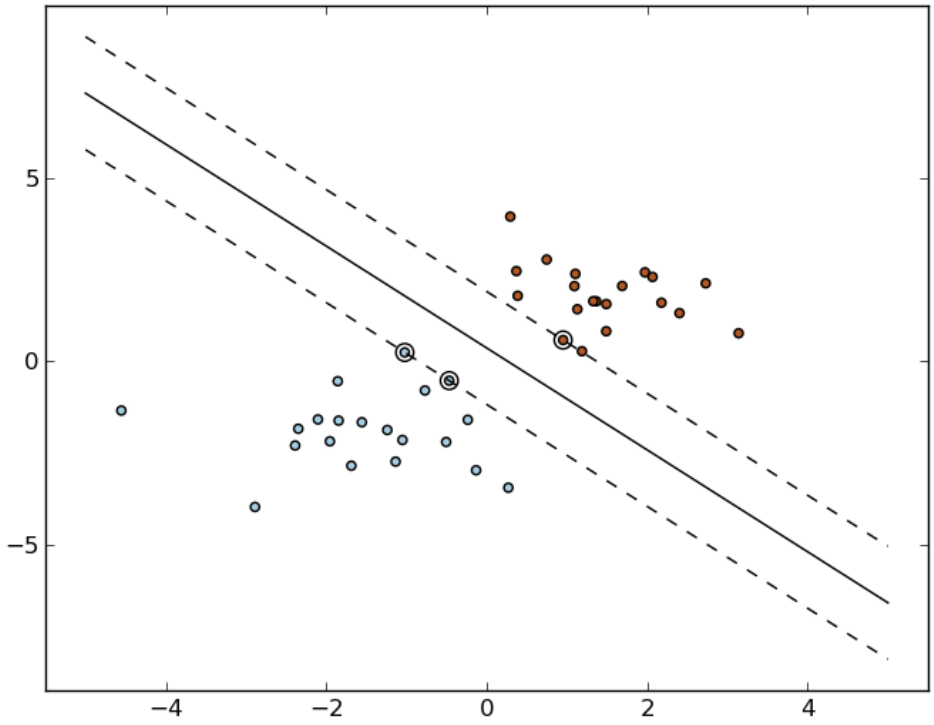

FIGURE 1 - LINEARLY SEPARABLE SVM PLOT WITH HYPER-PLANE [22]

Unfortunately, not all data set are linearly separable. Creating a hyper-plane in a non-linearly separable feature space is not always computationally possible. Because of this problem, researchers proposed the use of a kernel function to transform a low dimensional feature space into a higher dimensional feature space [16]. Figure 2 illustrates the transformation of a finite dimensional feature space into a higher dimensional feature space [23].

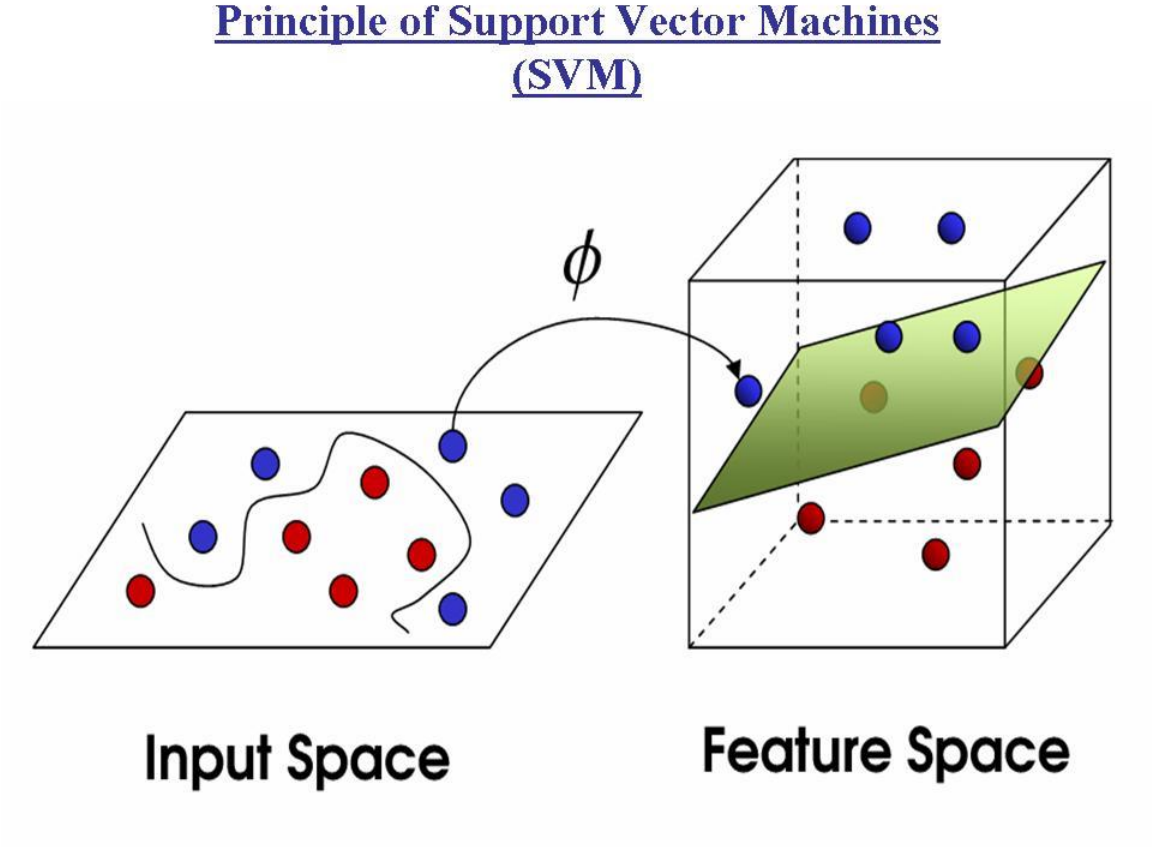

FIGURE 2 - FEATURE SPACE TRANSFORMATION THROUGH KERNEL TRICK [23]

2.4.2 Unsupervised Learning

In unsupervised learning approaches, the learners exclusively receive unlabeled data and infer unseen points. Since there is no labeled training data, the performance of the classifiers is generally difficult to quantitatively evaluated [16].

2.4.2.1 Frequent itemset and association rules

Association rule learning aims to discover interesting relations between variables in a data set such as association rules, correlations, sequences, episodes, classifiers and clusters [24]. There are two objective measurements often used to quantify these rules: support and confidence.

Support

The rule X ➔Y holds with support s if s is the ratio of number of instances (be it a sentence or a whole review) where X U Y to the total number of instances. Support indicates the relevance of the itemset to the data set. It answers the question: how many times X U Y occurs within the dataset.

Confidence

The rule X➔Y holds with confidence c if c is the ratio of number of instances that contains X U Y to the number of instances that contains X. Confidence indicates the degree of association between X and Y. It answers the question: if X occurs, what is the likelihood that Y also occurs.

We will leverage this technique to discover interesting topics that are frequently talked about within our product reviews.

2.4.2.2 Semantic Orientation and Pointwise Mutual Information

In the classic 2002 paper, Peter Turney described a simple unsupervised trained classification method that utilizes the Pointwise Mutual Information (PMI) algorithm to estimate the Sentiment Orientation (SO) of the review sentence [25]. The PMI measure the similarity or association between two words. Refer to Equation 1 for Turney's original PMI equation.

$$\text{PMI}\ (\text{word1}, \text{word2})\ =\ \log_2\left(\frac{\text{p}(\text{word1}\ \&\ \text{word2})}{\text{p}(\text{word1})\text{p}(\text{word2})}\right)$$

EQUATION 1 - TURNEY'S PMI EQUATION

The ratio between *p(word1 & word 2)* and *p(word1)p(word2)* measures the degree of statistical dependence between the words. The log of this ratio is the relevance of the presence of one of the words when we observe the other, known as the PMI

A word has positive Semantic Orientation when it has stronger association to words with generally positive sentiment. A word has negative Semantic Orientation when it has stronger association to words with generally negative sentiment.

Refer to Equation 2 for the Turney's original equation to calculate a word's SO

SO (word) = PMI (word, positive word)

- PMI (word, negative word)

EQUATION 2 - TURNEY'S SENTIMENT ORIENTATION EQUATION

2.4.3 Orange Data Mining and Visualization Toolkit

Orange Data Mining and Visualization Toolkit is an open source component-based data mining and machine learning software suite.

Originally developed at the Bioinformatics Laboratory at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia, Orange features easy to use visual programming user interface (UI), and a Python-based back-end. The product comes with a wide array of powerful explorative data analysis and visualization tools coupled with a scripting interface that allows users to insert custom Python code [26].

2.4.4 Natural Language Toolkit (NLTK)

The Natural Language Toolkit is a well-known package for building natural language processing (NLP) programs in Python. It features over 50 corpora and lexical resources such as

WordNet. On top of this, it comes with a suite of text processing libraries for classification, tokenization, stemming, lemmatization, part-of-speech tagging, and parsing [27].

CHAPTER 3: METHODOLOGY

This chapter discusses the study's methodology for corpus development, both unsupervised and supervised approach for topic identification and sentiment analysis.

3.1 Corpus Development

Statistical natural language techniques require a large corpus to generate reliable statistics. Fortunately, the World Wide Web has ample data for the purposes of our experiments. Dues to its availability and popularity, we selected the Amazon product reviews as the source for our corpus.

3.1.1 Corpus Selection

For this study, we study a dataset of Amazon product reviews and a data set of movie reviews. The primary corpus of this study consists of reviews from the Cisco-Linksys WRT160N Wireless-N Broadband Router. The movie reviews are used to demonstrate accuracy of a cross-domain trained classifier. Some classification techniques have bias tendency towards a certain class when the data distribution is unbalanced. To avoid this, we used the following criteria during the selection of product whose reviews make up the corpus:

- The product needs to have at least 500 reviews.
- The average review rating of the product must be close to three.
- The distribution of reviews in all five classes (1-star, 2-star, 3-star, 4-star, and 5-star) must be uniform or bipolar (U-shaped).

Once we have selected the candidate products, we modified and ran two Perl scripts written by Andrea Esuli to download reviews and parse the relevant information into Unicode text files [28].

3.1.2 Corpus Annotation

This section discusses the methodology used to annotate or tag the corpus. Understanding the annotations is important in interpreting the results in later sections.

3.1.2.1 Cisco-Linksys WRNT160N Wireless Router Set

Custom Python scripts further process and convert the reviews into raw text format. The following columns are extracted from the raw text to form the primary corpus:

- Rating (numerical 1-5)
- Review (string)

The output file is the Comma-Separated Value (CSV) file standard. We used this as the primary corpus, called Corpus A, for the supervised sentiment classification task. Corpus A consists of 1109 reviews.

Since all experiments are performed at sentence analysis level, a dataset of sentence-tokenized reviews must be generated from our review corpus. A secondary dataset was generated, called dataset A1, based on Corpus A. In dataset A1, we further tokenized the reviews into individual sentences. We accomplished this by using a custom Python script. The tokenization script splits each review at standard punctuation characters and line breaks. Additional text preprocessing is deemed necessary after spot checks reveal uncaught cases and improper English syntax (such as '…' and '!?!?'). After tokenization and further data preprocessing, dataset A1 contains 5929 unique sentences. Of those, we selected 2042 sentences to be tagged by a human reader. After reviewing the sentences, relevant topics and a sentiment were appended to the data. We ended up with a corpus that has the following attributes:

- Sentence (string)
- Sentiment (tertiary type: 0 for no sentiment, -1/1 for negative and positive sentiment)
- Support (binary)
- Usability (binary)
- Performance (binary)
- Installation (binary)
- Firmware (binary)

These attributes represent the strongest features of the sentences. The researchers detect the features manually by reading 2042 sentences and categorizing them. For each category/sentence, we assigned the following values:

- 1: the feature exists in the sentence
- 0: the feature doesn't exist in the sentence

The 2042 tagged records make up the training set, called dataset A2, for the supervised topic identification task.

3.1.2.2 Other corpus for cross-domain classification

Additionally we also utilized the Cornell Movie Review sentence polarity dataset v1.0 introduced in Pang/Lee ACL 2002 [29]. It consists of 5331 positive and 5331 negative processed sentences.

3.2 Supervised learning approaches

3.2.1 Supervised learning: Sentiment Analysis

First, we used the supervised learning technique, Naïve Bayes to classify the reviews as positive or negative. This task is simply a binomial classification task whose result's accuracy we used as the baseline. Dataset A1 (the corpus tokenized by sentences that have been tagged with sentiment) is used for review level classification where we evaluate the accuracy of our classifier against the rating specified by the original reviewers. Because we have a fully labeled dataset A1, we can measure our accuracy easily after the experiment. We used Orange as our analysis tool. The feature space consists of word unigrams. The class attribute in dataset A1 has two possible values: 0 (negative) and 1 (positive).

We imported the processed corpus into Orange as a binomial dataset. Using the following pipeline inside of Orange, we trained a Naïve Bayes classifier and evaluated its prediction against the tagged data:

File Import → Preprocess→ Bag of Words processing → Feature Selection →Naïve Bayes classifier → Test Learner →Confusion Matrix

The file import module, accepts a wide range of data source files (.tab, .txt, .data, .dat, .rda, .rdo, .basket, .arff, .xml, .svm). Once imported the data are converted to table data type, which is native to Orange. All values are default. For an illustration of the exact dialog box used in Orange to set the file import module, refer to Appendix B: Figure 6.

The preprocessing module, allows the user to easily perform common data preprocessing tasks such as lower case, remove stop words and lemmatization. All values are defaults. For an illustration of the exact dialog box used in Orange to set the preprocessing module, refer to Appendix B: Figure 7.

The "Bag-of-Words" processing module, presents several options to represent an instance of a string attribute as a multiset of its words. All values are default. For an illustration of the exact dialog box used in Orange to set the "Bag-of-Words" module, refer to Appendix B: Figure 8.

As exhaustive search is impractical, the feature selection module, allows user to evaluate the data instances using a selected subset of the features. All values are default. For an illustration of the exact dialog box used in Orange to set the feature selection module, refer to Appendix B: Figure 9.

Naïve Bayes and Support Vector Machine classifiers are the two main supervised learning algorithm examined in this study. Invoking and parameterizing these classifiers also prove to be simple in Orange. Values for the Naïve Bayes classifier are default. Values for the SVM classifiers are optimally selected by "Automatic parameter search." For an illustration of the exact dialog box used in Orange to set these classifiers, refer to Appendix B: Figure 10 and 11.

The Test Learner (Figure 12) and the Confusing Matrix (Figure 13) modules display the performance of each of the classifiers on the dataset in a simple to understand format. For an illustration of the exact dialog box used in Orange to set the feature selection module, refer to Appendix B: Figure 12 and 13.

To validate our dataset (which was tagged in-house), we ran the same classification pipeline against the independently tagged Cornell Movie reviews (also in sentence format).

To evaluate the performance of the classifiers we utilized k-fold cross validation testing (k=10) as displayed Appendix B: Figure 2.

3.2.2 Supervised Learning: Topic Identification

Dataset A2 serves as the main set for supervised topic identification. First, we import Dataset A2 as a tab delimited text file into Orange. Once the training data are successfully imported, we ran it through our Preprocess →Bag of Words processing → Feature Selection pipeline. Then using a Naïve Bayes classifier and an SVM classifier, the system predicts whether each sentence's topics contain the five topics originally tagged by a human coder: Firmware, Support, Installation, Performance, and Usability. The feature space consists of word unigrams.

To evaluate our learners we used k-fold cross-validation where k = 10. The recall and precision rates are then displayed for each category as Confusion Matrices. Please refer to the result section for these confusion matrices.

To summarize, we used the following pipeline in Orange to detect known or tagged topics in these sentences:

File Import →Preprocess→Bag of Words processing→Feature Selection→Naïve Bayes/SVM classifier → Test learners → Confusion Matrices

3.3 Unsupervised learning approaches

3.3.1 Unsupervised learning: Sentiment Analysis

To achieve this task, we utilized two external word lists from a previous 2004 study by Hu and Liu [3]. One contains generally positive words. The other contains negative words. For this task, we used dataset A1.

By modifying the Pointwise Mutual Information (PMI) algorithm from Peter Turney's paper, we are able to achieve higher performance than Turney's original experiment [25]:

$$\mathbf{PMI}\,(\mathbf{word}, [\mathbf{words\ with\ known\ SO}]) = \mathbf{log_2}\left(\frac{\mathbf{p}(\mathbf{word}\,\&\,[\mathbf{words}])}{\mathbf{p}(\mathbf{word})\mathbf{p}([\mathbf{words}])}\right)$$

EQUATION 3 - MODIFIED VERSION OF THE PMI EQUATION TO CALCULATE WORD'S PMI AGAINST A SET OF WORDS

Turney calculated the PMI of the word against a common word (from his paper "excellent" or "poor"). For this experiment, we will calculate a word's PMI against list of words instead of a single word. The feature (or word) here is a unigram. Thus, the Pointwise Mutual Information value of the word would be to a common theme. In our case, the themes are positive and negative sentiment. Using the two word lists with known sentiment, we compare each list to a list of all words from the corpus, and take the intersection of the two sets. This leaves us with two lists of words that actually exist in the corpus with known sentiment. We calculate the PMI as if the set of words is a single unit. So if the word evaluated co-occurs, at a statistically significant degree, with any word in the list, its PMI against that list of words increases.

The Sentiment Orientation (SO) of the word is calculated using the same formula provided by Peter Turney. The SO of a sentence is the sum of its words. A negative SO score is interpreted as negative sentiment; a positive SO score as positive

3.3.2 Unsupervised learning: Topic Identification

To accomplish the unsupervised machine learning, topic identification task, we explored the use of frequent itemsets to discover common utterances by measuring the confidence and support of these frequent itemsets [24].

We used the Python-based Orange Data Mining and Visualization Toolkit to accomplish this. Dataset A1 with 5929 sentences.

Since this task is unsupervised, the only attribute needed for this data set is:

- Sentence (string)

This piece involves two distinct tasks. First, we need to tell the machine to determine the topic.

First, we imported the dataset A1 into Orange as a tab-delimited text file. Then then we applied the following pipeline to discover the frequent item set:

Preprocess → Bag of Words processing → Feature Selection → Discretize → Python script

Inside of the Python script module, the following code was invoked:

```
import Orange

data = Orange.data.Table(in_data)
print "Association rules:"
rules = Orange.associate.AssociationRulesInducer(data, support=0.005, confidence=0.5)
for r in rules:
    print "%5.3f %5.3f %s" % (r.support, r.confidence, r)

out_object = rules
```

The resulting frequent item sets output at the command line.

Next, we examined the resulting itemsets and manually grouped them into likely categories. It should be noted that this is a step that requires human intervention. This category grouping is one part of the pipeline that is ‘supervised’. With the categories determined, Python

codes leveraging NLTK modules go through the untagged corpus and tag them according to the frequent item sets using the following logic:

If itemset in sentence:

tag sentence with itemset's topic = 1

else:

tag sentence with itemset's topic = 0

3.4 Cross-domain training

To examine the feasibility of cross-domain trained, supervised sentiment analysis, a SVM classifier and a Naïve Bayes classifier were trained on the sentiments of the Linksys review dataset. The trained classifiers then were used to classify the sentiment of movie reviews from the Cornell movie review dataset.

The resulting data are discussed further in Chapter 4.

CHAPTER 4: EXPERIMENTS AND RESULTS

This chapter presents the raw data and results of all the experiments performed in this study: supervised sentiment analysis, supervised topic identification, unsupervised sentiment analysis, unsupervised topic identification, and cross-domain train, supervised sentiment analysis.

4.1 Supervised Learning: Sentiment Analysis

The recall and precision rates of the SVM classifier are slightly lower than of the Naïve Bayes classifier in this particular task. Overall, the performance is comparable. Naïve Bayes achieved 75% precision and 82% recall in the negative sentiment class. The classifier achieved 72% precision and 63% recall in the positive sentiment class.

SVM achieved 72% precision and 84% recall with the negative sentiment class. It performed slightly worse than Naïve Bayes in the positive sentiment class with 72% precision and 56% recall rate.

	Negative Sentiment		Positive Sentiment		Classification Accuracy
	Precision	Recall	Precision	Recall	
Naïve Bayes (Supervised)	75%	82%	72%	63%	72%
SVM (Supervised)	72%	84%	72%	56%	74%

TABLE 1 - PRECISION AND RECALL OF NAIVE BAYES AND SVM CLASSIFIERS IN SENTIMENT ANALYSIS TASK

The results in Table 1 are used as our baseline metrics, against which we will compare the performance of our unsupervised approach to sentiment analysis.

For validation purpose, we also applied the same classification pipeline to a movie reviews data set widely used in many well-known studies. The resulting performance of indicate that the baseline for a supervised approach for sentiment analysis on a more well established, independently tagged, and larger data set at sentence level is actually lower than of our in-house tagged data set. More importantly, the performance of this data set is different from that of ours even when we are using the same classifiers and parameters:

Positive precision and recall: 61.6% and 60.7%

Negative precision and recall: 63.3% and 64%

4.2 Supervised learning: Topic identification

Human coders tagged each review with the following topics:

Firmware, Support, Installation, Performance, Usability

Using Naive Bayes and Support Vector Machine, we have achieved the performance illustrated in Table 2:

	Firmware		Support		Installation		Performance		Usability	
	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec
NB (Supervised)	87%	75%	75%	66%	76%	69%	74%	64%	75%	53%
SVM (Supervised)	96%	85%	94%	69%	97%	80%	99%	70%	95%	54%

TABLE 2 - THE PRECISION AND RECALL RATE FOR NAÏVE BAYES AND SUPPORT VECTOR MACHINE AGAINST THE FIVE TOPICS: FIRMWARE, SUPPORT, INSTALLATION, PERFORMANCE, AND USABILITY

Naive Bayes detects the topic Firmware in 75% of the sentences that actually talks about firmware. For all the sentences that Naive Bayes classifier tagged as talking about firmware, 87% are correct.

The Naive Bayes classifier detects the topic Support in 65% of the true Support sentences. Out of the sentences tagged as Support by the classifier, 75% are correctly identified.

Support vector machine classifier detects 85% of the Firmware sentences. For all the sentences that the SVM classifier tagged as Firmware, 95% of the predictions are correct.

4.3 Unsupervised learning: Sentiment Analysis

We have achieved the following performance using the unsupervised learning approach based on Peter Turney's 2001 paper. Using the tagged corpus of 642 sentences, we compare the system's prediction to the actual sentiment. The performance of our unsupervised approach can be best illustrated in Table 3 below:

	Positive Sentiment		Negative Sentiment		Overall Accuracy
	Precision	Recall	Precision	Recall	
PMI (Unsupervised)	70%	71%	79%	77%	75%
Naïve Bayes (Supervised)	75%	82%	72%	63%	72%
SVM (Supervised)	72%	84%	72%	56%	74%

TABLE 3 - SIDE-BY-SIDE PERFORMANCE COMPARISON OF AN UNSUPERVISED APPROACH AGAINST NAÏVE BAYES AND SVM

Overall, our unsupervised approach achieved higher accuracy rate than both Naïve Bayes and SVM in the sentiment analysis task.

4.4 Unsupervised learning: Topic Identification

Using an unsupervised approach to detect topics in the 2000 sentences, we have detected the following topics and achieved the following performance against the 2000 tagged sentences from our Linksys wireless router dataset.

Table 4 displays the discovered frequent itemsets and their corresponding topics:

Topic	Frequent Itemsets
Firmware	['firmware', 'upgrade'], ['update', 'firmware'], ['ddwrt','firmware'], ['late','firmware']
Installation	['setup','easy']
Performance	['speed', 'connection'], ['far', 'work'], ['signal','wireless'], ['internet','connection'], ['mpb','speed'], ['download','speed'], ['slow','speed'], ['house','range']
Usability	['could','connection'], ['laptop', 'connect'], ['device','connect'], ['work', 'fine'], ['laptop ','wireless'], ['device','wireless'], ['seem','work'],['drop','wireless'], ['great','work'], ['internet', 'work'], ['internet','wireless'], ['drop','connection'], ['power','cycle']
Support	['tech','call'], ['customer','call'], ['tech', 'support'], ['support','call'], ['service','call'], ['customer','support'], ['customer','service'],['warranty','call']

TABLE 4 - TOPICS AND THEIR CORRESPONDED DISCOVERED FREQUENT ITEMSETS

The itemsets are contained inside of Python list objects. Using the rules above the system's prediction performance is as follows illustrated in Table 5. The table displays the performance of our approach side by side with Naïve Bayes and SVM performance:

	Firmware		Support		Installation		Performance		Usability	
	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
Naïve Bayes (Supervised)	87%	75%	75%	66%	76%	69%	74%	64%	75%	53%
SVM (Supervised)	96%	85%	94%	69%	97%	80%	99%	70%	95%	54%
Frequent Itemset (Unsupervised)	93%	95%	82%	58%	85%	30%	29%	68%	35%	70%

TABLE 5 - SIDE-BY-SIDE COMPARISON BETWEEN FREQUENT ITEMSET APPROACH VS. SVM AND NAÏVE BAYES OVA APPROACH.

SVM's performance overtakes Naïve Bayes' on all categories. Frequent itemset approach performs slightly better than Naïve Bayes performs on several categories, but still performs worse than SVM.

4.5 Supervised learning: Sentiment analysis, a cross-domain experiment

To demonstrate an application of cross-domain trained supervised approach for sentiment analysis, we used the Linksys amazon review trained classifiers to classify sentiment on Cornell movie reviews. Figure 12 shows a less than impressive result from the cross-domain trained classifiers.

Negative Sentiment Class			
	Precision	Recall	Classification Acuracy
SVM	51%	0%	51%
Naïve Bayes	51%	0%	51%
Positive Sentiment Class			
	Precision	Recall	Classification Acuracy
SVM	n/a	0%	51%
Naïve Bayes	n/a	0%	51%

TABLE 6 SUPERVISED LEARNERS (SVM AND NAÏVE BAYES) PERFORMANCE WHEN TRAINED ON LINKSYS PRODUCT REVIEW AND TESTED ON CORNELL MOVIE REVIEWS

When trained in a cross-domain manner, the classifiers failed to detect features that they were trained on in the other dataset. The resulting classification placed all instances of the Cornell movie reviews in the target class 0: negative sentiment. Hence, 100% recall on the negative sentiment class and 0% recall on the positive sentiment class.

4.6 Sentiment per Topic

In addition to performing the sentiment analysis and topic identification experiments as separate tasks, the study also combined the results using a SVM classifier to detect topics. The reason being that SVM classifier achieved the highest performance in detecting topics in the tagged dataset. To classify customer sentiment towards these topics, the study used SVM, Naïve Bayes and Pointwise Mutual Information algorithms. The table below displays the resulting data:

Topics are detected using an SVM classifier (supervised)		Firmware	Installation	Performance	Usability	Support
SVM (Supervised)	Positive	29.89%	56.01%	43.70%	51.28%	24.52%
	Negative	70.11%	43.99%	56.30%	48.72%	75.48%
	Net Sentiment	Negative	Positive	Negative	Positive	Negative
Naïve Bayes (Supervised)	Positive	8.70%	40.44%	68.49%	34.03%	18.57%
	Negative	91.30%	59.56%	31.51%	65.97%	81.43%
	Net Sentiment	Negative	Negative	Positive	Negative	Negative
PMI (Unsupervised)	Positive	20.65%	45.82%	60.71%	34.35%	36.67%
	Negative	79.35%	54.18%	39.29%	65.65%	63.33%
	Net Sentiment	Negative	Negative	Positive	Negative	Negative

TABLE 7 SENTIMENT IS CLASSIFIED USING SVM, NAIVE BAYES AND PMI ALGORITHMS FOR EACH TOPIC

Refer to the Appendix for breakdown of the data above in pie chart format. On three of four topics, there are some disagreements between the Support Vector Machine classifier and the other two techniques. Generally, this can be addressed by a majority vote amongst an odd number of different classifiers.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

In Chapter 1, we established the following hypotheses:

(1) It is possible to classify the sentiment of review sentences at 50% accuracy or greater using supervised machine learning techniques

(2) It is possible to detect topics discussed in review sentences at 50% or greater precision and recall rate using supervised machine learning techniques

(3) It is possible to classify the sentiment of review sentences at 50% accuracy or greater using unsupervised machine learning techniques without training data

(4) It is possible to detect topics discussed in review sentences at 50% or greater precision and recall rate using unsupervised machine learning techniques without training data

(5) It is possible to classify sentiment of review sentence at 50% accuracy or greater using supervised machine learning techniques with out-of-domain training data

5.1 Supervised learning approach

5.1.1 Sentiment analysis

Utilizing Naïve Bayes and SVM classifier, we established the performance baseline for the main dataset, Linksys router reviews. The overall classification accuracy was about 72-74%.

In the 2002 paper, Pang and Lee achieved 78.7% accuracy with Naïve Bayes and 72.8% accuracy with SVM using 16165 unigram features on the Cornell movie review corpus [29]. By comparison, our Naïve Bayes and SVM classifiers only used 266 features and yielded comparable performance. The slight difference in performance could be attributed to the smaller feature space and the data dependent nature of any supervised learning approach.

This data dependency is demonstrated when we trained our classifier on the Linksys product review and tested the same classifiers on a different dataset, Cornell Movie reviews. The same classifiers that performed so well in the same-domain trained supervised sentiment analysis task, essentially identified every instance of the Cornell movie reviews as negative as seen in Figure 12.

In the limited domain of Linksys router reviews, the accuracy of our system to tell sentiment is better than of a random assignment, and thus is acceptable and supports part (1) of our hypothesis: supervised learning algorithms can be used to accurately classify sentiments of review sentences.

5.1.2 Topic Identification

Our supervised approach to topic identification is essentially a multiclass classification task using the One versus All (OVA) approach. The results were surprising. Unlike the results from our binary classification task (sentiment analysis), in topic identification, SVM actually performed consistently better than Naïve Bayes. On all five of the topic classes, both precision and recall metrics of the SVM classifier improved over Naïve Bayes.

The performance of both the Support Vector Machine classifier and Naïve Bayes classifier are superior to random assignment. It should be noted that for this particular task in this domain, support vector machine is clearly a more superior classifier. The results support part (2) of our hypothesis: supervised learning algorithms can be used to accurately identify topics of review sentences.

5.2 Unsupervised learning approach

5.2.1 Sentiment Analysis

Using an unsupervised approach, the overall accuracy of our system in the task of classifying sentiment is 75%. This suggests that an unsupervised approach to sentiment analysis is not only feasible but in some cases can actually perform better than a supervised approach, inside this domain.

These data support part (3) of our hypothesis. It is feasible to build a completely unsupervised system to perform sentiment analysis. Other applications may also include spam filtering, which has been primarily dominated by naïve Bayes based systems.

5.2.2 Topic Identification

The degree of un-supervision of our approach to topic identification is arguable. As one of the crucial tasks: assigning meanings and labels to clusters of frequent itemsets, still needs the supervision of a human reader. As the system is not completely unsupervised, we cannot call it as such. Our approach should be correctly referred to as semi-unsupervised.

The results from these experiments do not support part (4) of our hypothesis. It is not yet feasible to build a completely unsupervised system to perform the task of topic identification. It is, however, possible to build a semi-unsupervised learning system to do so. This shortcoming is discussed further in the Future works section.

The performance of our semi-unsupervised system is overall better than of a Naïve Bayes classifier. However, it is worse than of a Support Vector Machine classifier in 8 out of 10 metrics. Refer to Table 5 in Chapter 4.

5.3 Cross-domain experiment

In part (5) of the hypothesis, the study set out to explore the feasibility of cross-domain trained classifiers for the sentiment analysis tasks. The cross-domain experiment yielded some less than impressive results. The Naïve Bayes and SVM classifiers when trained on Linksys reviews failed to accurately classify the sentiments in the Cornell movie reviews.

Part (5) of our hypothesis is not supported by these results. As supervised learner cannot perform well when trained on one domain and tested on a different domain.

5.4 Future works

The results of this study are optimistic. A system that can automate product review analysis is plausible given the results achieved in this exploratory study. Furthermore, it is entirely conceivable that there are more complex algorithms out there that can analyze these reviews more effectively. Based on the results of this study, an end-to-end system can be built using these same techniques to achieve similar performance today.

The results suggests that the perceived performance drawback of unsupervised technique is not always true. A relatively simple unsupervised technique such as Pointwise Mutual Information has surpassed the performance of Support Vector Machine and Naïve Bayes classifier in this case. This will give rise to the possibility that an entirely unsupervised, end-to-end system can one day surpass the performance of a domain-dependent supervised system. This system will be able to discover the customer sentiment toward relevant features of any product. Product manufacturers such as Linksys will utilize this information to improve their products and services. Consumers will be able to use this information to make informed buying decisions more quickly and provide effective feedbacks via the system. Amazon as an e-marketplace provider would be more interested in the cross-domain capability of this system. If they develop

and own this system, it would potentially be marketed to product manufacturers like Linksys as a Software-as-a-Service.

We look forward to addressing the problem of true unsupervised topic identification and explore a different experiment setup for cross-domain trained sentiment analysis approach.

First, our semi-unsupervised to topic identification lacks a mechanism for the machine to detect and group the frequent itemsets into distinctive topics. One suggestion for future study would be to utilize WordNet to possibly detect the semantic distance between frequent itemsets. As the WordNet API already has a method to calculate semantic distance between Synsets (equivalence of a unigram), some heuristic process will need to be devised to implement the methods on frequent itemsets.

Secondly, a revisit to the feasibility of cross-domain trained supervised sentiment analysis is need. This study experimented on two domains that are may be too different to yield any meaningful results (Linksys routers vs Movies). An argument could be made that if the classifiers were to be trained on Linksys router reviews and tested on a different brand of routers or a different type of electronic products, the results would be more interesting. As a suggestion for future study, our Linksys-router-trained Naïve Bayes and SVM classifiers should be given another chance to go against a tagged corpus of a different brand of routers.

REFERENCES

[1] A. Mukherjee and B. Liu, "Modeling Review Comments," in *Proceedings of 50th Annual Meeting of Association for Computational Linguistics*, Jeju, Republic of Korea, 2012.

[2] B. Liu, "Sentiment Analysis and Subjectivity," in *Handbook of Natural Language Processing*, Chicago, Illinois, Chapman and Hall/CRC, 2010.

[3] M. Hu and B. Liu, "Mining Opinion Features in Customer Reviews," in *Proceedings of Nineteeth National Conference on Artificial Intellgience*, San Jose, USA, 2004.

[4] B. Liu, M. Hu and J. Cheng, "Opinion Observer: Analyzing and Comparing Opinions on the Web," in *Proceedings of WWW*, Chicago, 2005.

[5] M. Aery, N. Ramamurthy and Y. A. Aslandogan, "Topic Identification of Textual Data," The University of Texas at Arlington, Arlington, Texas, 2003.

[6] B. Liu, Sentiment Analysis and Opinion Mining, Chicago, Illinois: Morgan & Claypool Publishers, 2012.

[7] S. Mukherjee and P. Bhattacharyya, "Feature Specific Sentiment Analysis for Product Reviews," in *Proceedings of the 13th international conference on Computational Linguistics and Intelligent Text Processing*, Berlin, 2012.

[8] A. Pandit and P. A. Bhole, "Automated Intelligence Product Review Analysis," *International Journal of Wisdom Based Computing,* vol. 1, no. 2, pp. 61-62, 2011.

[9] O. Rud, Business Intelligence Success Factors: Tools for Aligning Your Business in the Global Economy, Hoboken: Wiley & Sons, 2009.

[10] J. Yi, T. Nasukawa, R. Bunescu and W. Niblack, "Sentiment analyzer: extracting sentiments about a given topic using natural language processing techniques," in *Third IEEE International Conference on Data Mining*, San Jose, 2003.

[11] Z. Zhai, B. Liu, H. Xu and P. Jia, "Clustering Product Features for Opinion Mining," in *Proceedings of ACM International Conference on Web Search and Data Mining*, Chicago, 2011.

[12] A. Mukherjee and B. Liu, "Aspect Extraction through Semi-Supervised Modeling," in *Proceedings of 50th Annual Meeting of Association for Computational Linguistics*, Jeju, Republic of Korea, 2012.

[13] L. Lee, "Measures of distributional similarity," in *Proceedings of the ACL*, Chicago, 1999.

[14] M.-C. Kim and K.-S. Choi, "A comparison of collocation-based similarity measures in query expansion," *Information Processing and Management: an International Journal,* vol. 35, no. 1, pp. 19-30 , 1999.

[15] W. Wei and J. A. Gulla, "Sentiment learning on product reviews via sentiment ontology tree," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, PA, 2010.

[16] M. Mohri, A. Rostamizadeh and A. Talwalkar, Foundations of Machine Learning, Cambridge, MA: MIT Press, 2012.

[17] E. Alpaydin, Introduction to Machine Learning, Cambridge: MIT Press, 2010.

[18] Y. Song, A. Kołcz and C. L. Giles, "Better Naive bayes classification for high-precision spam detection," *Software-Practice and Experience,* vol. 39, no. 1, pp. 1003-1024, 2009.

[19] D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," in *Proceedings of ECML-98, 10th European Conference on Machine Learning*, Chemnitz, 1998.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995.

[21] R. Burbidge and B. Buxton, "An introduction to support vector machine for data mining," University College London, London, 2001.

[22] D. Cournapeau, "SVM: Maximum margin separating hyperplane," 2010. [Online]. Available: http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#example-svm-plot-separating-hyperplane-py. [Accessed 2013].

[23] A. Niissalo, "Principles of Support Vector Machines (SVM).," 2010. [Online]. Available: http://www.imtech.res.in/raghava/rbpred/svm.jpg. [Accessed 2013].

[24] J. Han, M. Kamber and J. Pei, "Frequent Item set Mining Methods," in *Data Mining: Concepts and Techniques*, Burlington, MA, Morgan Kaufmann, 2011, pp. 227-232.

[25] P. Turney, "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews," *ACL,* no. Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pp. 417-424, 2002.

[26] J. Demšar, T. Curk and A. Erjavec, "Orange: Data Mining Toolbox in Python," *Journal of Machine Learning Research,* vol. 14, p. 2349−2353, 2013.

[27] E. Loper and S. Bird, "NLTK: The Natural Language Toolkit," in *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, Philadelphia, 2002.

[28] A. Esuli, "Amazon reviews downloader and parser," 2013. [Online]. Available: http://www.esuli.it/fossil/repo/amazonReviewsDownloader/index. [Accessed 2013].

[29] P. Pang and L. Lee, "Thumbs up?: sentiment classification using machine learning techniques," *ACL,* vol. 10, no. Proceedings of the ACL-02 conference on Empirical methods in natural language processing, pp. 79-86 , 2002.

[30] C.-C. Hsu, Y.-P. Huang and K.-W. Chang, "Extended Naive Bayes classifier for mixed data," *expert systems with application,* vol. 35, no. 1, pp. 1080-1083, 2008.

[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations,* vol. 11, no. 1, 2009.

[32] P. Reutemann, B. Pfahringer and E. Frank, "Proper: A Toolbox for Learning from Relational Data with Propositional and Multi-Instance Learners," in *Proceedings of the 17th Australian joint conference on Advances in Artificial Intelligence*, Berlin, 2004.

[33] G. Tsoumakas, I. Katakis and I. Vlahavas, "Mining Multi-label Data," in *Data Mining and Knowledge Discovery Handbook*, New York, Springer, 2010.

[34] J. Read and P. Reutemann, "MEKA: A Multi-label Extension to WEKA," 2012. [Online]. Available: http://meka.sourceforge.net/. [Accessed 2013].

APPENDIX A: PYTHON CODES

```
#!/usr/bin/python
__author__ = 'Vincent'

import nltk, Orange, math, replacers, numpy as np
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import RegexpTokenizer
from nltk.collocations import *
from nltk.stem import WordNetLemmatizer, PorterStemmer, RegexpStemmer



corpus = Orange.data.Table('C:\XO.tab')

#corpus = [['My linksys works good have a great security wireless and the
speed needed for computer \,games console a smartphone  this product is not
great.','5'],
#           ['Since I am not technically smart\, I had a week\'s worth of
agonizing over this installation\, Had to make 2 calls to Linksys-two calls
to TWC -My ISP #\'s weren\'t the same\, etc. etc.  Now I have to reboot
almost everyday--since I lose the connection. Dont know if that\'s the fault
of Linksys or TWC or ME!!  Not happy\, still looking for a better solution.',
'1'],
#           ['The router does not work.  Since the Amazon return policy does
not provide a refund\, I will throw away the router and purchase a new one at
a local electronics store.  This was a very disappointing process and I do
NOT recommend this router or buying anything off of amazon.','1'],
#           ['The router performed perfectly out of the box.  I changed
several settings to match my network and security settings and all works
well.I use this in my home office.  I work from home and depend heavily on
it.  It just works great!I would highly recommend this router.','5']]
english_stops = set(stopwords.words('english'))
regReplacer = replacers.RegexReplacer()
repReplacer = replacers.RepeatReplacer()
splReplacer = replacers.SpellingReplacer()
antReplacer = replacers.AntonymReplacer()
tokenizer = RegexpTokenizer(r'\w+')
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
regexSt = RegexpStemmer('ing$|ly$|ivity$', min=4)
wcount = 0
uAdj =[]      #unique adj
corpAdj = [] #corpus with only adjectives
seedP = ['good', 'excellent', 'great']
seedN = ['bad', 'terrible', 'poor']
allWords =[] #flattened array of all words in corpus
uWords = []
uCount = {}
aCount={}
c = corpus
corpNegation = [] #corpus of adjective and negation
```

```
bigram_measures = nltk.collocations.BigramAssocMeasures()
aSO = {}

with open('negative-words.txt') as f:
    seedN = f.read().splitlines()
with open('positive-words.txt') as p:
    seedP = p.read().splitlines()

for thing in c: #make list of unique adjective
    line = str(thing[0])
    line = line.lower()
    words = tokenizer.tokenize(line)
    words = [word for word in words if word not in english_stops]
    freshWords = []
    #words = antReplacer.replace_negation(words)
    #print words
    for word in words:
        word = regReplacer.replace(word)
        word = repReplacer.replace(word)
        word = splReplacer.replace(word)
#prettifying words
        word = word.decode('utf-8')
        freshWords.append(word)
    #print freshWords
    allWords.extend(freshWords)
    tWords = nltk.pos_tag(freshWords)
    newtWords =[]
    for w in tWords:
        newtWords.append(w)
#         if w[1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS']:
#             newtWords.append(w)

    newtWords = nltk.untag(newtWords)
    #print newtWords
    trueAdj = []
    for wd in newtWords:
        trueAdj.append(wd)
#         if wordnet.synsets(wd, pos='a') or wordnet.synsets(wd, pos='r'):
#             w = wd
#             trueAdj.append(w)
    corpAdj.append(trueAdj)
    uAdj.extend((list(set(trueAdj).difference(uAdj))))
    uWords.extend((list(set(freshWords).difference(uWords))))
#print uAdj

for a in uAdj:
    count = allWords.count(a)
    aCount[a] = count


for a in uWords:
    count = allWords.count(a)
    uCount[a] = count

for thing in c: # transform corpus into adjectives and their negation
```

```
    line = str(thing[0])
    line = line.lower()
    words = tokenizer.tokenize(line)
    freshWords = []
    for word in words:
        word = regReplacer.replace(word)
        word = repReplacer.replace(word)
        word = splReplacer.replace(word)
#prettifying words
        word = word.decode('utf-8')
        freshWords.append(word)
    tWords = nltk.pos_tag(freshWords)
    newtWords =[]
    for i in range(0, len(tWords)):
        newtWords.append(tWords[i])
#         if tWords[i][1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS']:
#             if i-1 > -1 and tWords[i-1][0] == 'not':    #check 1 character
before adj for negation
#                 newtWords.append(tWords[i-1])
#                 newtWords.append(tWords[i])
#             elif i-2 > -1 and tWords[i-2][0] == 'not':  #check 2 characters
before adj for negation : (e.g., 'not very good')
#                 newtWords.append(tWords[i-2])
#                 newtWords.append(tWords[i])
#             else:
#                 newtWords.append(tWords[i])
    newtWords = nltk.untag(newtWords)
    corpNegation.append(newtWords)



wcount= len(allWords) #set total word count
seedP = list(set(seedP).intersection(uAdj)) #recheck to make sure initial
seeds are valid
seedN = list(set(seedN).intersection(uAdj))

#usage- getP(word, str:N/P)
# N for negative
# P for positive
def getC (w1):

    global seedP, seedN, uAdj, corpNegation
    cor = corpNegation
    cP = 0
    cN = 0
    for th in cor:
        #thing = tokenizer.tokenize(th)
        if w1 in th:
            for i in range(0, len(th)):
                th[i] = th[i].encode('ascii')
                if i-1 > -1 and th[i-1] == 'not':
                    if th[i] in seedP:
                        #print words[i]
                        cN += 1
                    elif th[i] in seedN:
```

```
                        cP +=1
                else:
                    if th[i] in seedP:
                        cP += 1
                    elif th[i] in seedN:
                        cN += 1

    return float(cP), float(cN)




def getSO(w1):
    global wcount, seedP, seedN, aCount
    c = getC(w1)
    CountP = c[0]
    CountN = c[1]
    spCount = 0
    snCount = 0
    #print wcount
    for p in range(0, len(seedP)):
        spCount += aCount[seedP[p]]
    for n in range(0, len(seedN)):
        snCount += aCount[seedN[n]]



    pmiP = bigram_measures.pmi((.1 + CountP),(aCount[w1], spCount), wcount)
    pmiN = bigram_measures.pmi((.1+ CountN),(aCount[w1], snCount), wcount)

    SO = pmiP - pmiN
#    if SO > 3:                  #if strong SO, add word as seed
#        #seedP.append(list(w1))
#        seedP.extend(list(set([w1]).difference(seedP)))
#    elif SO < -4:
#        seedN.extend(list(set([w1]).difference(seedN)))


    return SO




def corpAnalyze():
    global corpNegation,aSO

    cor = corpNegation
    for t in range(0,len(cor)):
        line = str(thing)
        line = line.lower()
        words = tokenizer.tokenize(line)
        lineSO = 0
```

```
        for i in range(0, len(words)):
            if words[i] in aSO:
                if i-1 > -1 and words[i-1] == 'not':
                    lineSO -= aSO[words[i]]
                else:
                    lineSO += aSO[words[i]]
        cor[t].append(lineSO)
    return cor

for a in range (0, len(uAdj)):
    aSO[str(uAdj[a]).encode('ascii')] = getSO(uAdj[a].encode('ascii'))



def getRegSO(w1):
    global wcount, seedP, seedN, uCount
    c = getC(w1)
    CountP = c[0]
    CountN = c[1]
    spCount = 0
    snCount = 0

    for p in range(0, len(seedP)):
        spCount += aCount[seedP[p]]
    for n in range(0, len(seedN)):
        snCount += aCount[seedN[n]]


    pmiP = bigram_measures.pmi((.1 + CountP),(uCount[w1], spCount), wcount)
    pmiN = bigram_measures.pmi((.1+ CountN),(uCount[w1], snCount), wcount)

    SO = pmiP - pmiN
    return SO


print aSO
print 'Negative seeds: ' + str(seedN)
print 'Positive seeds: ' + str(seedP)
outC =[[]]
for i,thing in enumerate(c): #make list of unique adjective
    freshWords2=[]
    line = str(thing[0])
    line = line.lower()
    words = tokenizer.tokenize(line)
    words = [word for word in words if word not in english_stops]
    for word in words:
        word = regReplacer.replace(word)
        word = repReplacer.replace(word)
        word = splReplacer.replace(word)
#prettifying words
        word = word.decode('utf-8')
        freshWords2.append(word)
    tWords2 = nltk.pos_tag(freshWords2)
    # newtWords2 =[]
    # for w in tWords2:
```

```
    #     if w[1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS']:
    #         newtWords2.append(w)
    #
    # newtWords2 = nltk.untag(newtWords2)
    # #print newtWords
    trueAdj2 = []
    lineSO = 0
    for wd in freshWords2:
        if wd.encode('ascii') in aSO:
            lineSO += aSO[wd.encode('ascii')]
        elif wordnet.synsets(wd, pos='a') or wordnet.synsets(wd, pos='r'):
            lineSO += getRegSO(wd)
    outC.append([thing[1], lineSO])



  #  for w in freshWords2:
  #      lineSO += getRegSO(w)
  #  outC.append([thing[1], lineSO])
correct = 0
incorrect = 0
rightDog = 0
rightCat = 0
total = len(outC)
threshold = 0
print outC
for i in range (1,len(outC)):
    rating1 = outC[i][0]
    if rating1 == '?':
        rating1 = outC[i-1][0]
    r1 = int(rating1.native())
    if r1 == 1 and outC[i][1] > threshold:
        correct+= 1
        rightDog +=1
    elif r1 == 0 and outC[i][1]< threshold:
        correct+=1
        rightCat +=1
    else:
        incorrect+=1


precision = 0
recall = 0
guessDog = 0
trueDog = 0
guessCat = 0
trueCat = 0
for k in range (1,len(outC)):
    rating = outC[k][0]
    if rating == '?':
        rating = outC[k-1][0]
    r = int(rating.native())
    if r in [1]:
        trueDog +=1
```

```
    if outC[k][1] > threshold:
        guessDog +=1
    if r in [0]:
        trueCat +=1
    if outC[k][1] < threshold:
        guessCat +=1
precisionDog = float(rightDog)/float(guessDog)
recallDog = float(rightDog)/float(trueDog)
precisionCat = float(rightCat)/float(guessCat)
recallCat = float(rightCat)/float(trueCat)



print 'Correct: ' + str(correct)
print 'Incorrect: '+ str(incorrect)
print 'n = ' + str(total)
print str((float(correct)/float(total))*100) +'% correct'
print 'Precision (POS): '+ str(precisionDog*100) + '%'
print 'Recall (POS): '+ str(recallDog*100) + '%'
print 'Precision (NEG): '+ str(precisionCat*100) + '%'
print 'Recall (NEG): '+ str(recallCat*100) + '%'
print 'Threshold:' + str(threshold)
#print 'Seed cycles: '+ str(cycles)



# print seedP
# print seedN
# print aCount
# print corpAdj
# print corpNegation
```

UnsupervisedSA.py

```
#!/usr/bin/python
__author__ = 'Vincent'

import nltk, Orange, math, replacers, numpy as np
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import RegexpTokenizer
from nltk.collocations import *
from nltk.stem import WordNetLemmatizer, PorterStemmer, RegexpStemmer


corpus = Orange.data.Table('C:\Users\Vincent\Desktop\XTD.tab')
c = corpus
tokenizer = RegexpTokenizer(r'\w+')
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
firmware = ['firmware']
installation = ['set','setup', 'set up', 'set-up' 'install', 'installation']
performance = ['good range','range','speed','coverage','performance',
'network']
usability = ['ease','easy','problem', 'Internet', 'internet']
support = ['customer','support','service','call']


for thing in c:
    line = str(thing[0]).lower()
    words = tokenizer.tokenize(line)
#    for i in range(0,len(words)):
#        words[i]= stemmer.stem(str(words[i]))
    if list(set(words).intersection(firmware)):
        thing[1]= 1
    if list(set(words).intersection(installation)):
        thing[2]= 1
    if list(set(words).intersection(performance)):
        thing[3]= 1
    if list(set(words).intersection(usability)):
        thing[4]=1
    if list(set(words).intersection(support)):
        thing[5]=1
    print thing
```

UnsupervisedTD.py

```
#!/usr/bin/python
__author__ = 'Vincent'

import nltk, Orange, math, replacers, numpy as np
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import RegexpTokenizer
from nltk.collocations import *
import csv
from nltk.stem import WordNetLemmatizer, PorterStemmer, RegexpStemmer



#corpus = Orange.data.Table('C:\Users\Vincent\Desktop\XTD.tab')
corpus = Orange.data.Table(r'C:\Users\Vincent\Desktop\net2kb.tab')
#corpus = [['My linksys works good have a great security wireless and the
speed needed for computer \,games console a smartphone  this product is not
great.','5'],
#           ['Since I am not technically smart\, I had a week\'s worth of
agonizing over this installation\, Had to make 2 calls to Linksys-two calls
to TWC -My ISP #\'s weren\'t the same\, etc. etc.  Now I have to reboot
almost everyday--since I lose the connection. Dont know if that\'s the fault
of Linksys or TWC or ME!!  Not happy\, still looking for a better solution.',
'1'],
#           ['The router does not work.  Since the Amazon return policy does
not provide a refund\, I will throw away the router and purchase a new one at
a local electronics store.  This was a very disappointing process and I do
NOT recommend this router or buying anything off of amazon.','1'],
#           ['The router performed perfectly out of the box.  I changed
several settings to match my network and security settings and all works
well.I use this in my home office.  I work from home and depend heavily on
it.  It just works great!I would highly recommend this router.','5']]
english_stops = set(stopwords.words('english'))
regReplacer = replacers.RegexReplacer()
repReplacer = replacers.RepeatReplacer()
splReplacer = replacers.SpellingReplacer()
antReplacer = replacers.AntonymReplacer()
tokenizer = RegexpTokenizer(r'\w+')
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
regexSt = RegexpStemmer('ing$|ly$|ivity$', min=4)
wcount = 0
uAdj =[]      #unique adj
corpAdj = [] #corpus with only adjectives
seedP = ['good', 'excellent', 'great']
seedN = ['bad', 'terrible', 'poor']
```

```
# firmware = [['firmware', 'upgrade'], ['update', 'firmware'],
['ddwrt','firmware'], ['late','firmware']]
# installation = [['setup','easy']]
# performance = [['speed', 'connection'], ['far', 'work'],
['signal','wireless'] ,['internet','connection'],['mpb','speed']
#                     ['download','speed'], ['slow','speed'],
['house','range']]
# usability = [['could','connection'], ['laptop', 'connect'], ['device',
'connect'], ['work', 'fine'],
#              ['laptop ','wireless'],
['device','wireless'],['seem','work'],['drop','wireless'], ['great','work'],
#              ['internet', 'work'], ['internet', 'wireless'],
['drop','connection'], ['power','cycle']]
# support = [['tech','call'],['customer','call'],['tech',
'support'],['support','call'],['service','call'],
#             ['customer','support'],
['customer','service'],['warranty','call']]



firmware = [['firmware', 'version'],['late','firmware'],
['update','firmware']]
feature = [['share','usb'],['parental','control'],
['external','drive'],['storage','usb'],['drive','usb']]
performance =
[['netflix','stream'],['video','stream'],['strength','signal'],['strength','r
outer'],
['strong','signal'],
['mbp','router'],['far','router'],['performance','router'],['range','router']
]
usability =[['stop','work'], ['restart','router'],
['reboot','router'],['work','fine'],['problem','router'],
['issue','router'],['die', 'router'], ['good', 'router'], ['drop',
'connection'], ['easy', 'use']]
support =[['forum','netgear'], ['told','support'], ['customer','service'],
['tech','support'],
['problem', 'support','netgear']]
installation =[['configure','router'],
['easy','setup'],['easy','set','router']]

allWords =[] #flattened array of all words in corpus
uWords = []
uCount = {}
aCount={}
c = corpus
corpNegation = [] #corpus of adjective and negation
bigram_measures = nltk.collocations.BigramAssocMeasures()
aSO = {}

with open('negative-words.txt') as f:
    seedN = f.read().splitlines()
with open('positive-words.txt') as p:
    seedP = p.read().splitlines()

for thing in c: #make list of unique adjective
```

```
    line = str(thing[0])
    line = line.lower()
    words = tokenizer.tokenize(line)
    words = [word for word in words if word not in english_stops]
    freshWords = []
    #words = antReplacer.replace_negation(words)
    #print words
    for word in words:
        word = regReplacer.replace(word)
        word = repReplacer.replace(word)
        word = splReplacer.replace(word)
#prettifying words
        word = word.decode('utf-8')
        freshWords.append(word)
    #print freshWords
    allWords.extend(freshWords)
    tWords = nltk.pos_tag(freshWords)
    newtWords =[]
    for w in tWords:
        newtWords.append(w)
#         if w[1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS']:
#             newtWords.append(w)

    newtWords = nltk.untag(newtWords)
    #print newtWords
    trueAdj = []
    for wd in newtWords:
        trueAdj.append(wd)
#         if wordnet.synsets(wd, pos='a') or wordnet.synsets(wd, pos='r'):
#             w = wd
#             trueAdj.append(w)
    corpAdj.append(trueAdj)
    uAdj.extend((list(set(trueAdj).difference(uAdj))))
    uWords.extend((list(set(freshWords).difference(uWords))))
#print uAdj

for a in uAdj:
    count = allWords.count(a)
    aCount[a] = count


for a in uWords:
    count = allWords.count(a)
    uCount[a] = count

for thing in c: # transform corpus into adjectives and their negation
    line = str(thing[0])
    line = line.lower()
    words = tokenizer.tokenize(line)
    freshWords = []
    for word in words:
        word = regReplacer.replace(word)
        word = repReplacer.replace(word)
        word = splReplacer.replace(word)
#prettifying words
```

```
        word = word.decode('utf-8')
        freshWords.append(word)
    tWords = nltk.pos_tag(freshWords)
    newtWords =[]
    for i in range(0, len(tWords)):
        newtWords.append(tWords[i])
#         if tWords[i][1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS']:
#             if i-1 > -1 and tWords[i-1][0] == 'not':    #check 1 character
before adj for negation
#                 newtWords.append(tWords[i-1])
#                 newtWords.append(tWords[i])
#             elif i-2 > -1 and tWords[i-2][0] == 'not':  #check 2 characters
before adj for negation : (e.g., 'not very good')
#                 newtWords.append(tWords[i-2])
#                 newtWords.append(tWords[i])
#             else:
#                 newtWords.append(tWords[i])
    newtWords = nltk.untag(newtWords)
    corpNegation.append(newtWords)



wcount= len(allWords) #set total word count
seedP = list(set(seedP).intersection(uAdj)) #recheck to make sure initial
seeds are valid
seedN = list(set(seedN).intersection(uAdj))

#usage- getP(word, str:N/P)
# N for negative
# P for positive
def getC (w1):

    global seedP, seedN, uAdj, corpNegation
    cor = corpNegation
    cP = 0
    cN = 0
    for th in cor:
        #thing = tokenizer.tokenize(th)
        if w1 in th:
            for i in range(0, len(th)):
                #th[i] = th[i].encode('ascii')
                if i-1 > -1 and th[i-1] == 'not':
                    if th[i] in seedP:
                        #print words[i]
                        cN += 1
                    elif th[i] in seedN:
                        cP +=1
                else:
                    if th[i] in seedP:
                        cP += 1
                    elif th[i] in seedN:
                        cN += 1

    return float(cP), float(cN)
```

```
def getSO(w1):
    global wcount, seedP, seedN, aCount
    c = getC(w1)
    CountP = c[0]
    CountN = c[1]
    spCount = 0
    snCount = 0
    #print wcount
    for p in range(0, len(seedP)):
        spCount += aCount[seedP[p]]
    for n in range(0, len(seedN)):
        snCount += aCount[seedN[n]]



    pmiP = bigram_measures.pmi((.1 + CountP),(aCount[w1], spCount), wcount)
    pmiN = bigram_measures.pmi((.1+ CountN),(aCount[w1], snCount), wcount)

    SO = pmiP - pmiN
#    if SO > 3:                  #if strong SO, add word as seed
#        #seedP.append(list(w1))
#        seedP.extend(list(set([w1]).difference(seedP)))
#    elif SO < -4:
#        seedN.extend(list(set([w1]).difference(seedN)))


    return SO




def corpAnalyze():
    global corpNegation,aSO

    cor = corpNegation
    for t in range(0,len(cor)):
        line = str(thing)
        line = line.lower()
        words = tokenizer.tokenize(line)
        lineSO = 0
        for i in range(0, len(words)):
            if words[i] in aSO:
                if i-1 > -1 and words[i-1] == 'not':
                    lineSO -= aSO[words[i]]
                else:
                    lineSO += aSO[words[i]]
        cor[t].append(lineSO)
    return cor
```

```
# Uncomment for SA
for a in range (0, len(uAdj)):
     #aSO[str(uAdj[a]).encode('ascii')] = getSO(uAdj[a].encode('ascii'))
     aSO[str(uAdj[a])]= getSO(uAdj[a])



def getRegSO(w1):
    global wcount, seedP, seedN, uCount
    c = getC(w1)
    CountP = c[0]
    CountN = c[1]
    spCount = 0
    snCount = 0

    for p in range(0, len(seedP)):
        spCount += aCount[seedP[p]]
    for n in range(0, len(seedN)):
        snCount += aCount[seedN[n]]


    pmiP = bigram_measures.pmi((.1 + CountP),(uCount[w1], spCount), wcount)
    pmiN = bigram_measures.pmi((.1+ CountN),(uCount[w1], snCount), wcount)

    SO = pmiP - pmiN
    return SO


#print aSO
#print 'Negative seeds: ' + str(seedN)
#print 'Positive seeds: ' + str(seedP)
outC =[]
for i,thing in enumerate(c): #make list of unique adjective
    freshWords2=[]
    line = str(thing[0])
    line = line.lower()
    words = tokenizer.tokenize(line)
    words = [word for word in words if word not in english_stops]
    for word in words:
        word = regReplacer.replace(word)
        word = repReplacer.replace(word)
        word = splReplacer.replace(word)
#prettifying words
        word = word.decode('utf-8')
        freshWords2.append(word)
    tWords2 = nltk.pos_tag(freshWords2)
    # newtWords2 =[]
    # for w in tWords2:
    #     if w[1] in ['JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS']:
    #         newtWords2.append(w)
    #
    # newtWords2 = nltk.untag(newtWords2)
    # #print newtWords
    for r in firmware:
```

```
    if list(set(words).intersection(list(r))):
        thing[1]= 1
        break
    else:
        thing[1] = 0


for r in installation:
    if list(set(words).intersection(list(r))):
        thing[2]=1
        break
    else:
        thing[2] = 0

for r in performance:
    if list(set(words).intersection(list(r))):
        thing[3]= 1
        break
    else:
        thing[3] = 0



for r in usability:
    if list(set(words).intersection(list(r))):
        thing[4]=1
        break
    else:
        thing[4] = 0

for r in support:
    if list(set(words).intersection(list(r))):
        thing[5]=1
        break
    else:
        thing[5] = 0

for r in feature:
    if list(set(words).intersection(r)):
        thing[6]= 1
    else:
        thing[6] = 0


trueAdj2 = []
lineSO = 0
for wd in freshWords2:
    #if wd.encode('ascii') in aSO:
    if wd in aSO:
        #lineSO += aSO[wd.encode('ascii')]
        lineSO += aSO[wd]
    elif wordnet.synsets(wd, pos='a') or wordnet.synsets(wd, pos='r'):
        lineSO += getRegSO(wd)
if lineSO < 0:
    thing[7] = 0
```

```
    elif lineSO > 0:
        thing[7] = 1
    outC.append(thing)


#myfile = open('res2k.csv', 'wb')
myfile = open('net2k.csv', 'wb')
wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
#wr.writerow(['Sentence','Firmware','Installation','Performance','Usability',
'Support','Sentiment'])
wr.writerow(['Sentence','Firmware','Installation','Performance','Usability','
Support','Feature','Sentiment'])
for thing in outC:
    wr.writerow([thing[0],thing[1],thing[2],thing[3],thing[4],thing[5],
thing[6], thing[7]])
myfile.close()



  #  for w in freshWords2:
  #      lineSO += getRegSO(w)
  #  outC.append([thing[1], lineSO])
# correct = 0
# incorrect = 0
# rightDog = 0
# rightCat = 0
# total = len(outC)
# threshold = 0

# for i in range (1,len(outC)):
#     rating1 = outC[i][0]
#     if rating1 == '?':
#         rating1 = outC[i-1][0]
#     r1 = int(rating1.native())
#     if r1 == 1 and outC[i][1] > threshold:
#         correct+= 1
#         rightDog +=1
#     elif r1 == 0 and outC[i][1]< threshold:
#         correct+=1
#         rightCat +=1
#     else:
#         incorrect+=1


# precision = 0
# recall = 0
# guessDog = 0
# trueDog = 0
# guessCat = 0
# trueCat = 0
# for k in range (1,len(outC)):
#     rating = outC[k][0]
#     if rating == '?':
#         rating = outC[k-1][0]
```

```
#     r = int(rating.native())
#     if r in [1]:
#         trueDog +=1
#     if outC[k][1] > threshold:
#         guessDog +=1
#     if r in [0]:
#         trueCat +=1
#     if outC[k][1] < threshold:
#         guessCat +=1
# precisionDog = float(rightDog)/float(guessDog)
# recallDog = float(rightDog)/float(trueDog)
# precisionCat = float(rightCat)/float(guessCat)
# recallCat = float(rightCat)/float(trueCat)



# print 'Correct: ' + str(correct)
# print 'Incorrect: '+ str(incorrect)
# print 'n = ' + str(total)
# print str((float(correct)/float(total))*100) +'% correct'
# print 'Precision (POS): '+ str(precisionDog*100) + '%'
# print 'Recall (POS): '+ str(recallDog*100) + '%'
# print 'Precision (NEG): '+ str(precisionCat*100) + '%'
# print 'Recall (NEG): '+ str(recallCat*100) + '%'
# print 'Threshold:' + str(threshold)
#print 'Seed cycles: '+ str(cycles)




# print seedP
# print seedN
# print aCount
# print corpAdj
# print corpNegation
```

TestPipe.py

```
__author__ = 'Vincent'

import csv
testList = []
trainList = []

with open(r'TID_Perf\res2k.csv', 'rb') as test:
    testData = list(csv.reader(test))
with open(r'TID_Perf\Sent2k.csv', 'rb') as train:
    trainData = list(csv.reader(train))

for row in testData:
    testList.append([row[0], int(row[1]), int(row[2]), int(row[3]),
int(row[4]), int(row[5])])


for r in trainData:
    trainList.append([r[0], int(r[1]), int(r[2]), int(r[3]), int(r[4]),
int(r[5])])




def getPerf(i):
    global testList, trainList
    guessDog = 0
    trueDog = 0
    guessCat = 0
    trueCat = 0
    rightDog = 0
    rightCat = 0
    correct = 0
    incorrect = 0

    for c in range(0, 1004):
        if testList[c][i] == 1:
            guessDog+=1
        if testList[c][i] == 0:
            guessCat+=1
        if trainList[c][i]== 1:
            trueDog+=1
        if trainList[c][i] == 0:
            trueCat+=1
        if testList[c][i] == trainList[c][i] == 1:
            rightDog += 1
            correct += 1
        if testList[c][i] == trainList[c][i] == 0:
            rightCat += 1
            correct += 1
        else:
            incorrect +=1


    precisionDog = float(rightDog)/float(guessDog)
    recallDog = float(rightDog)/float(trueDog)
```

```
    precisionCat = float(rightCat)/float(guessCat)
    recallCat = float(rightCat)/float(trueCat)

    return [precisionDog, recallDog, precisionCat, recallCat]


firmware = getPerf(1)
installation = getPerf(2)
performance = getPerf(3)
usability = getPerf(4)
support = getPerf(5)

print 'Results: Has Topic? Y/N'
print 'Firmware:'
print 'Precision: '+str(firmware[0])+r'/'+ str(firmware[2])
print 'Recall: '+str(firmware[1])+r'/'+ str(firmware[3])
print ''
print 'Installation:'
print 'Precision: '+str(installation[0])+r'/'+ str(installation[2])
print 'Recall: '+str(installation[1])+r'/'+ str(installation[3])
print ''
print 'Performance:'
print 'Precision: '+str(performance[0])+r'/'+ str(performance[2])
print 'Recall: '+str(performance[1])+r'/'+ str(performance[3])
print ''
print 'Usability:'
print 'Precision: '+str(usability[0])+r'/'+ str(usability[2])
print 'Recall: '+str(usability[1])+r'/'+ str(usability[3])
print ''
print 'Support:'
print 'Precision: '+str(support[0])+r'/'+ str(support[2])
            print 'Recall: '+str(support[1])+r'/'+ str(support[3])
```

TestPerformance.py

APPENDIX B: ORANGE SCREENSHOTS

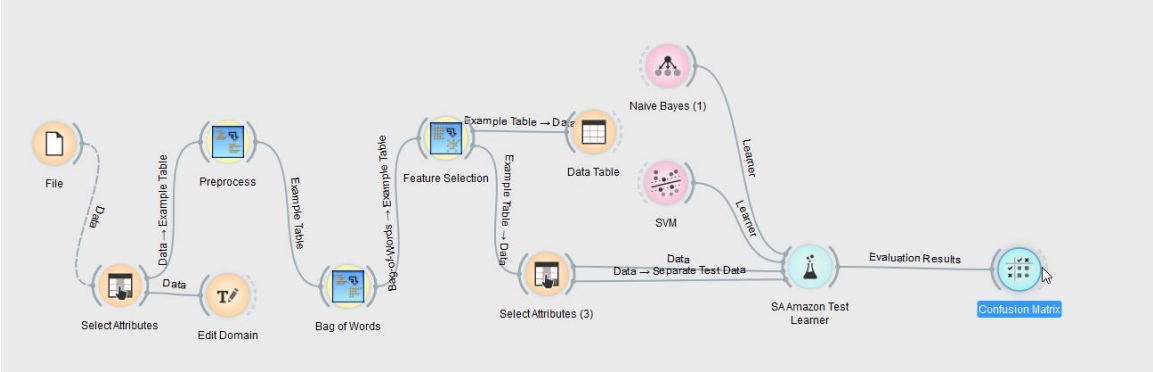

FIGURE 3 SUPERVISED SENTIMENT ANALYSIS PIPELINE IN ORANGE

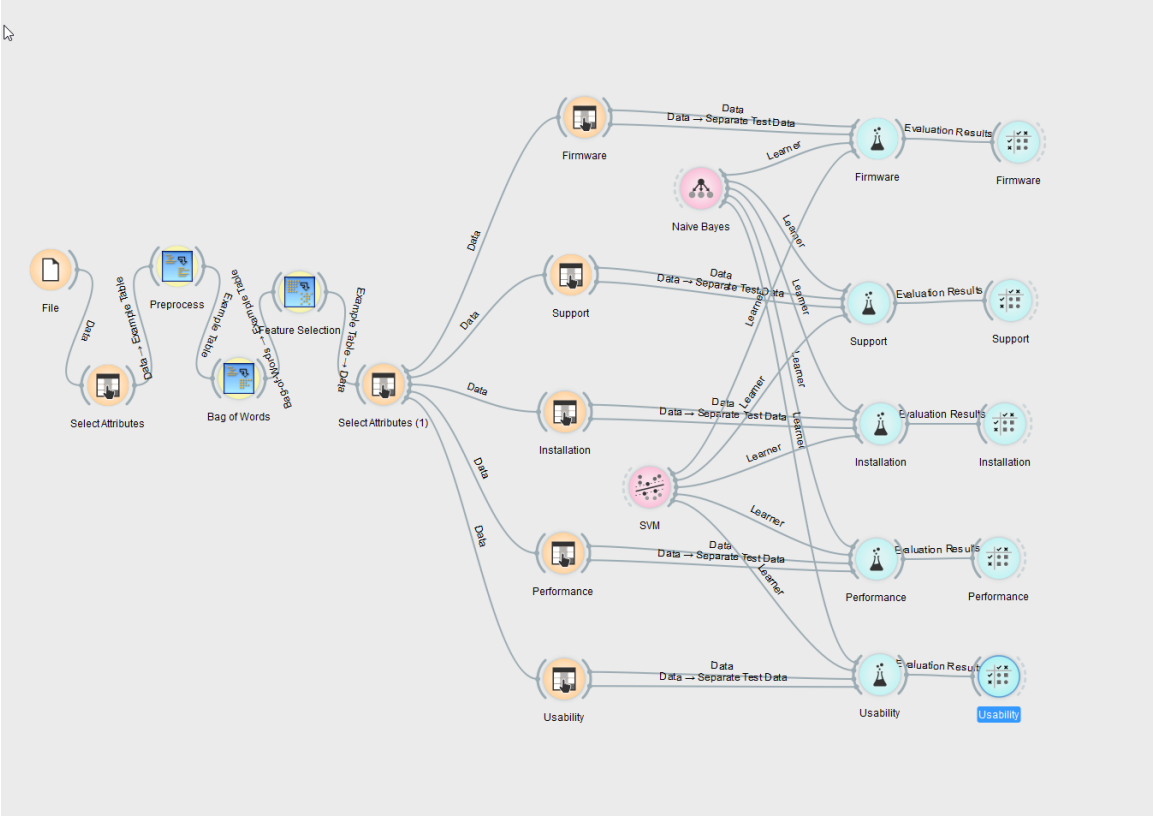

FIGURE 4 SUPERVISED TOPIC IDENTIFICATION PIPELINE IN ORANGE

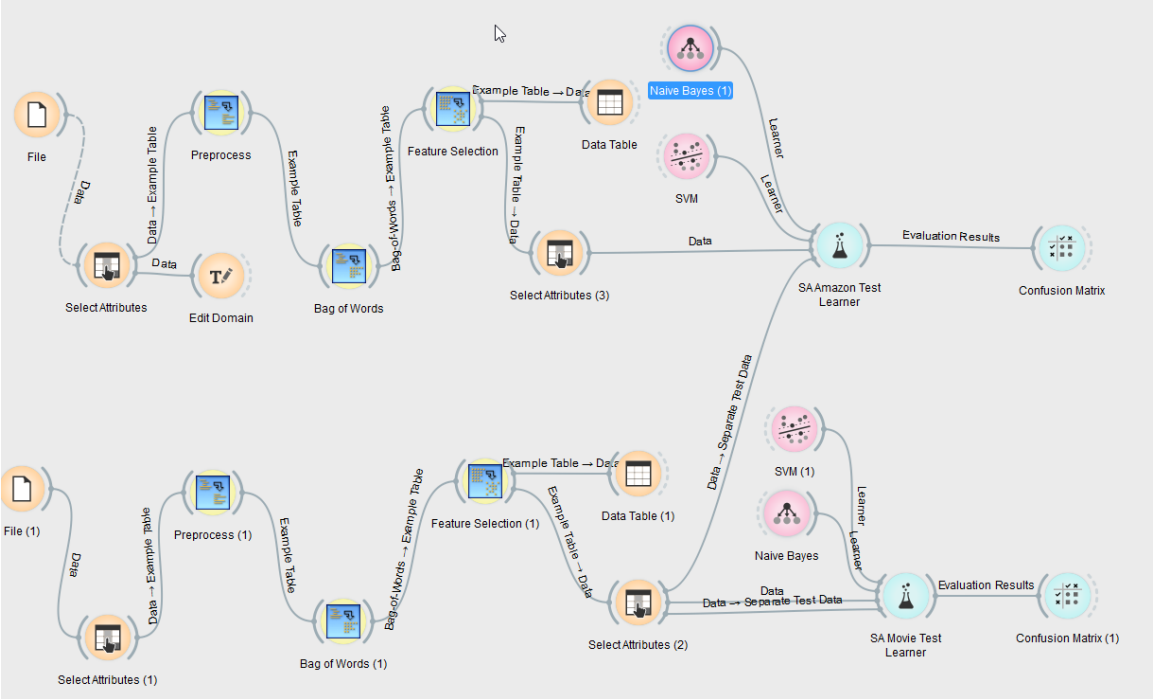

FIGURE 5 SUPERVISED SENTIMENT ANALYSIS CROSS-DOMAIN TRAINED IN ORANGE

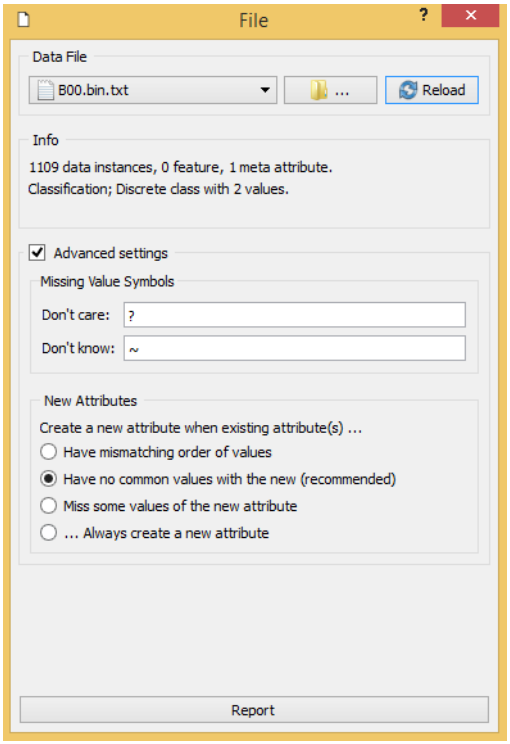

FIGURE 6 - FILE IMPORT MODULE FROM ORANGE

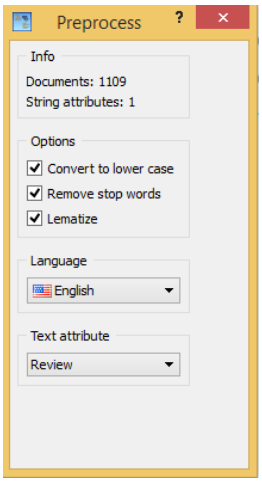

FIGURE 7 - PREPROCESS MODULE

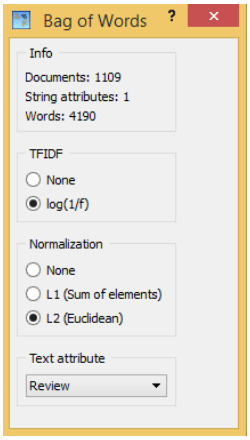

FIGURE 8 - BAG-OF-WORDS MODULE

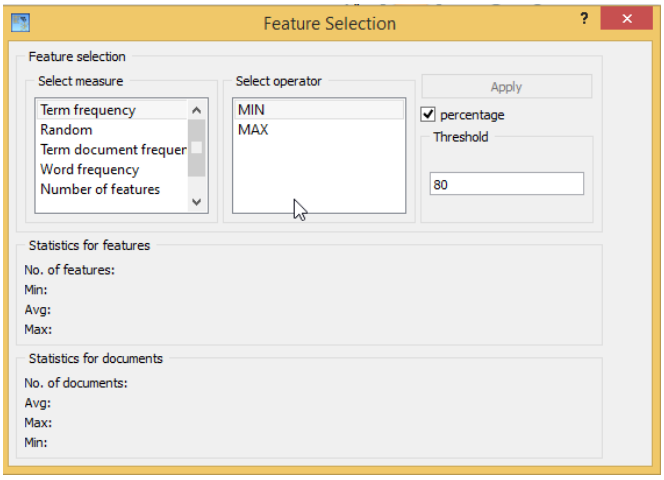

FIGURE 9 - FEATURE SELECTION MODULE

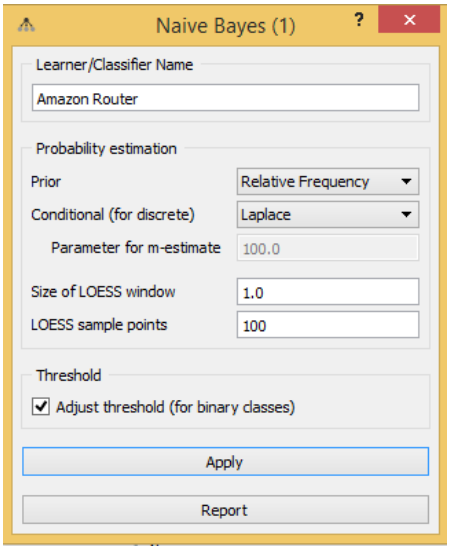

FIGURE 10 - NAIVE BAYES CLASSIFIER

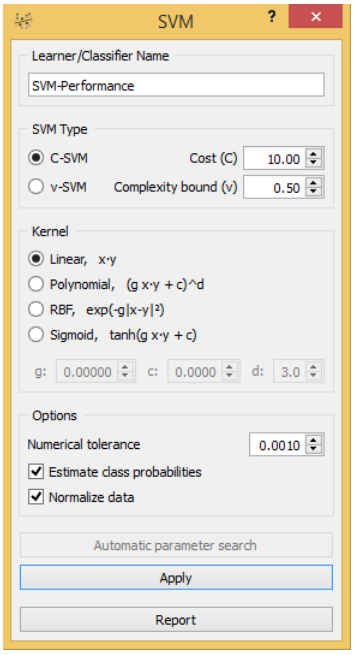

FIGURE 11 - SVM CLASSIFIER

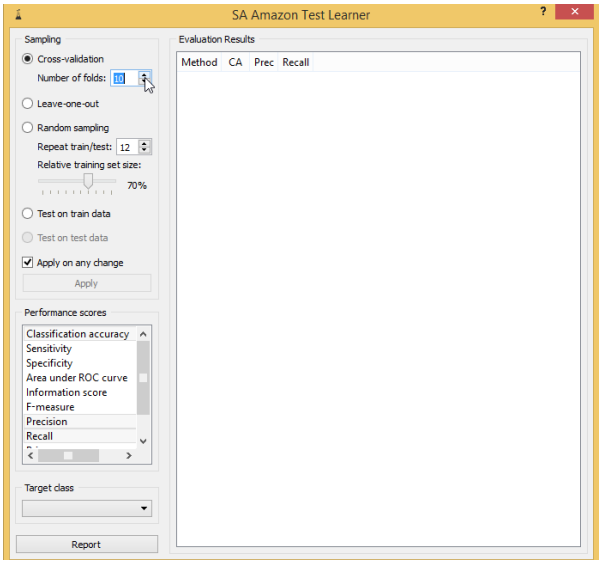

FIGURE 12 - TEST LEARNER MODULE

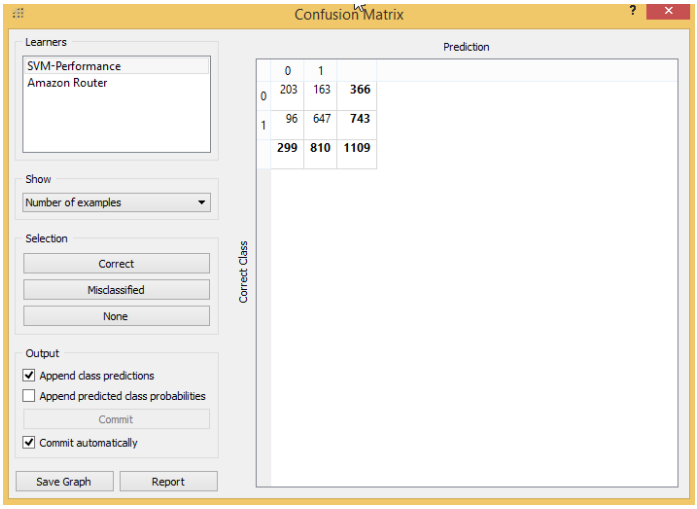

FIGURE 13 - CONFUSION MATRIX MODULE

APPENDIX C: CHARTS OF SENTIMENT BY TOPICS

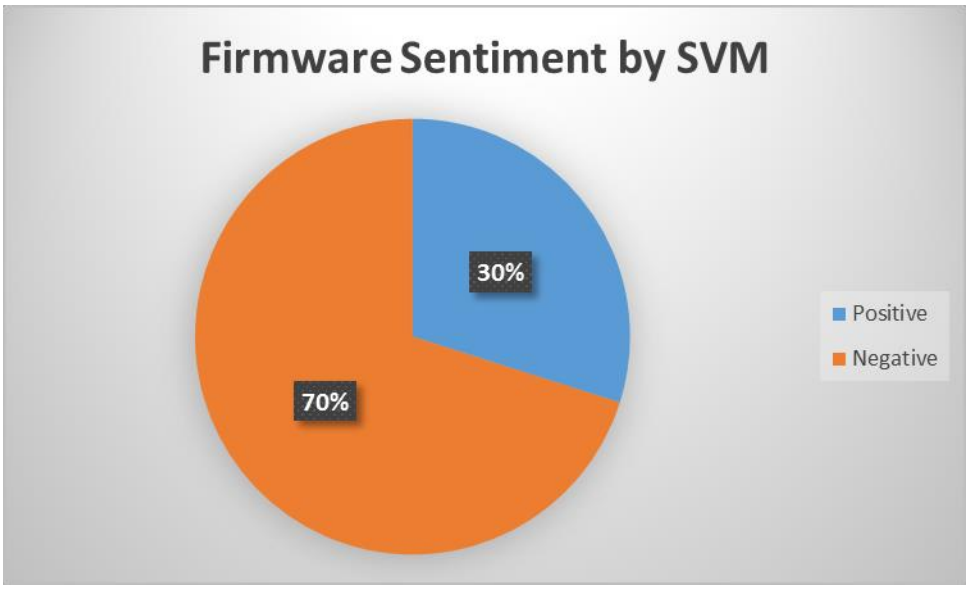

FIGURE 14 SENTIMENT FOR TOPIC FIRMWARE BY SVM

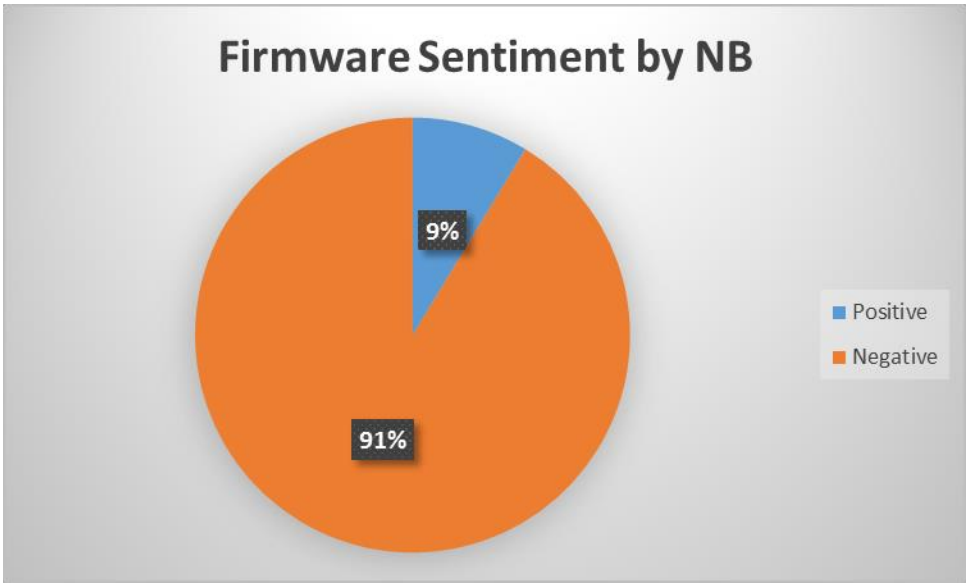

FIGURE 15 SENTIMENT FOR TOPIC FIRMWARE BY NAÏVE BAYES

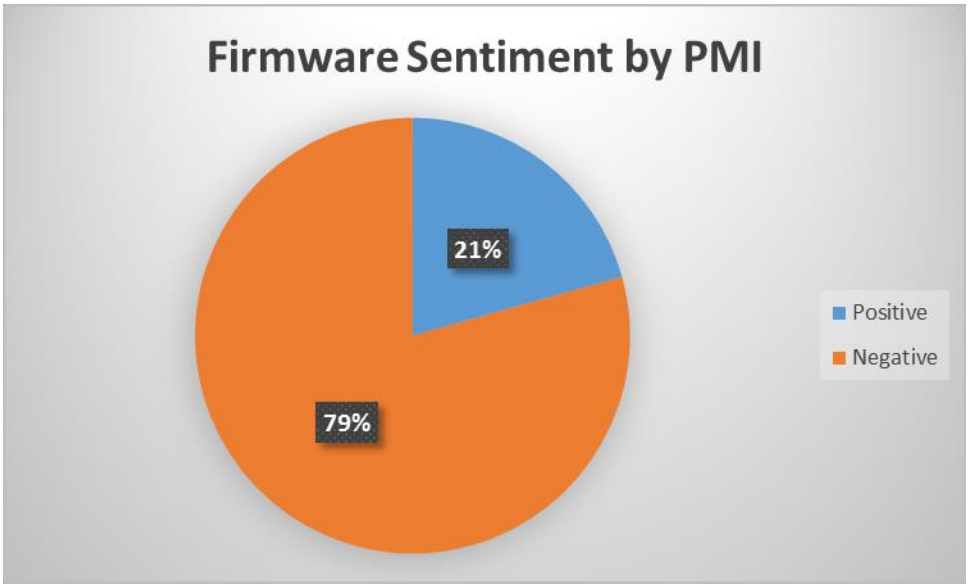

FIGURE 16 SENTIMENT FOR TOPIC FIRMWARE BY PMI

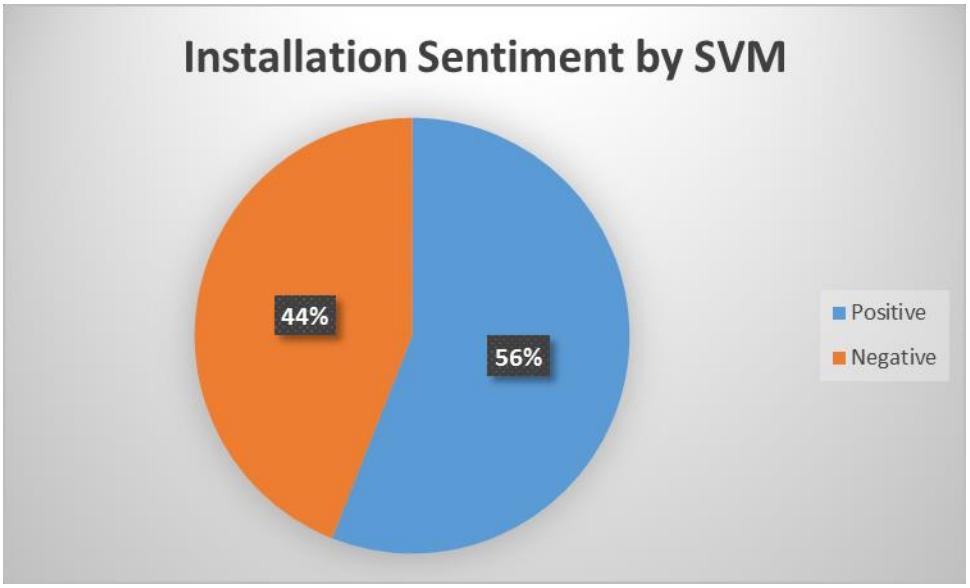

FIGURE 17 SENTIMENT FOR TOPIC INSTALLATION BY SVM

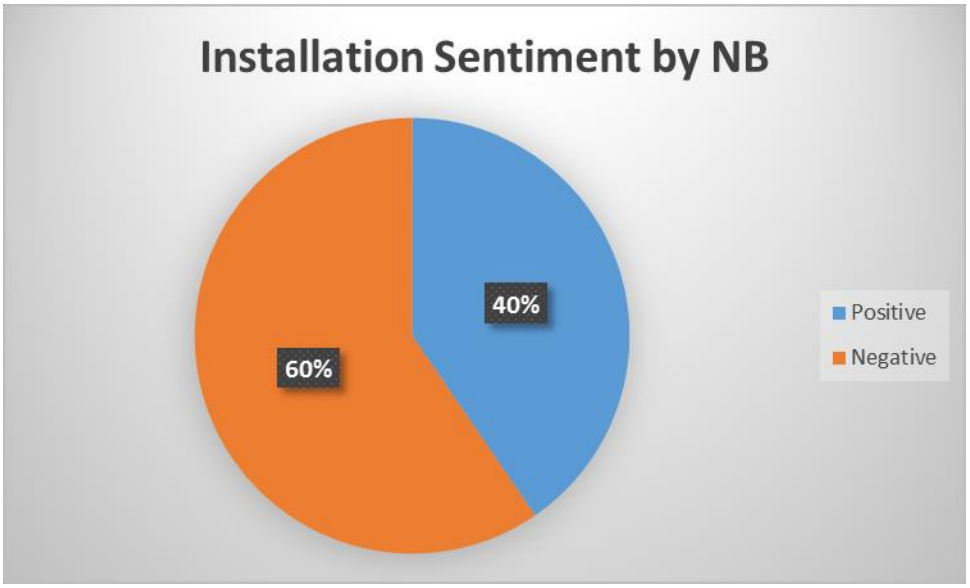

FIGURE 18 SENTIMENT FOR TOPIC INSTALLATION BY NB

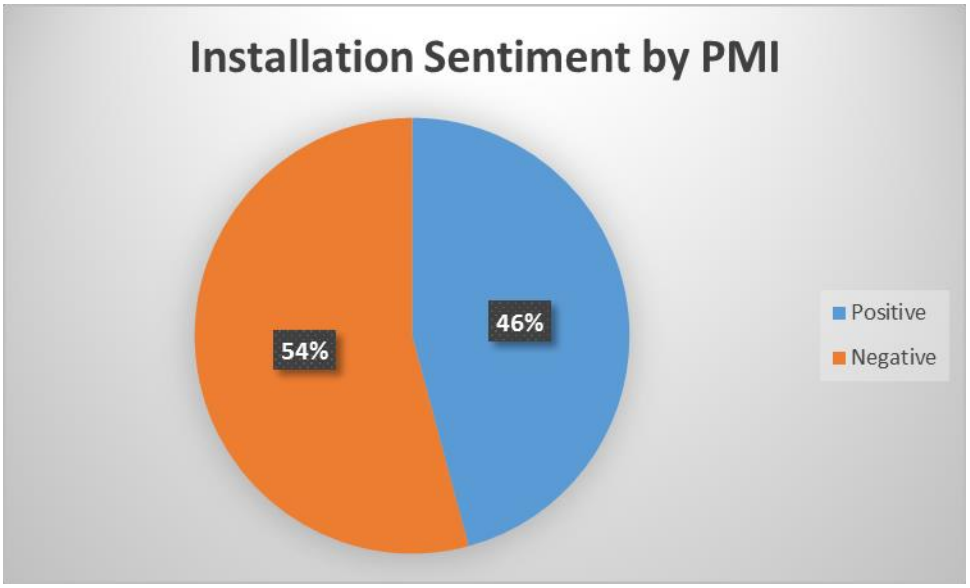

FIGURE 19 SENTIMENT FOR TOPIC INSTALLATION BY PMI

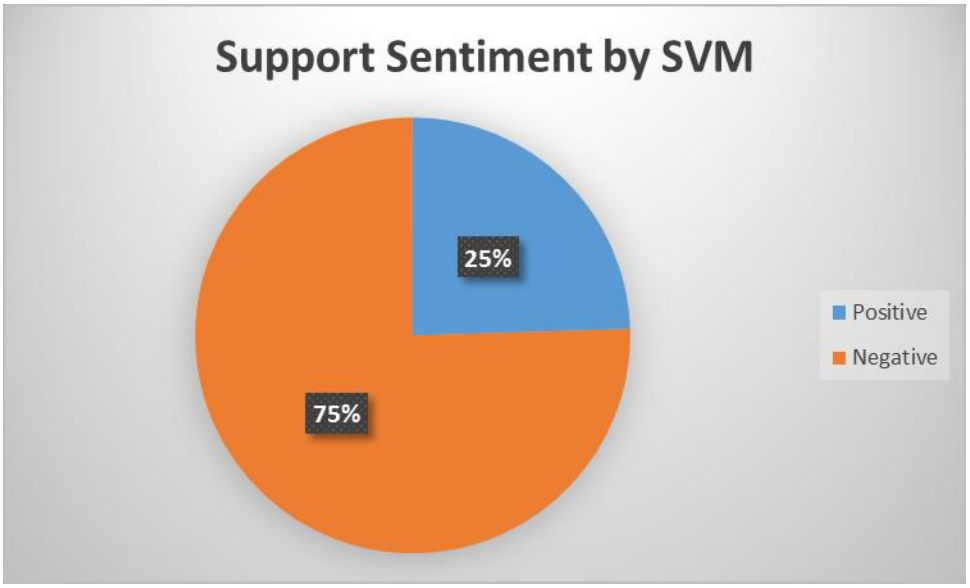

FIGURE 20 SENTIMENT FOR TOPIC SUPPORT BY SVM

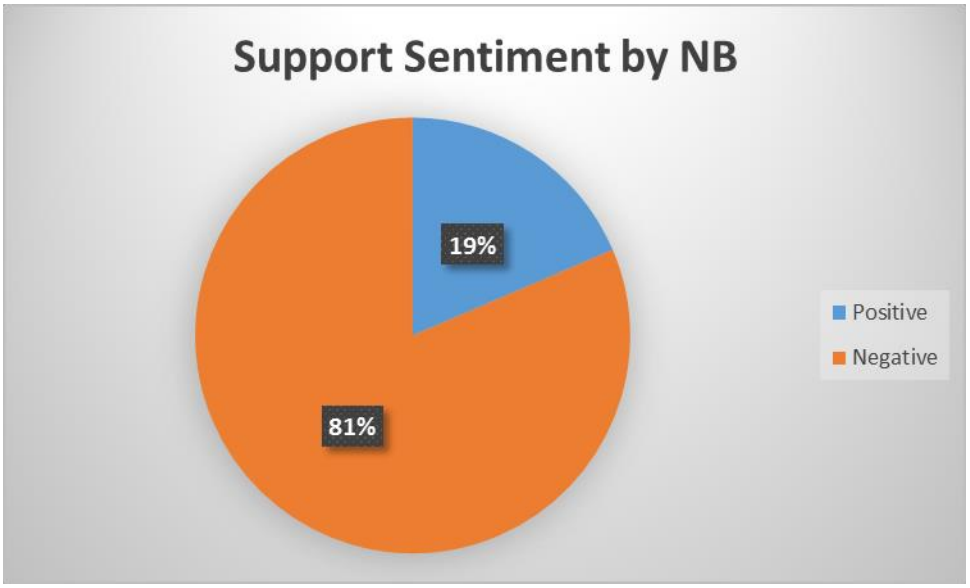

FIGURE 21 SENTIMENT FOR TOPIC SUPPORT BY NB

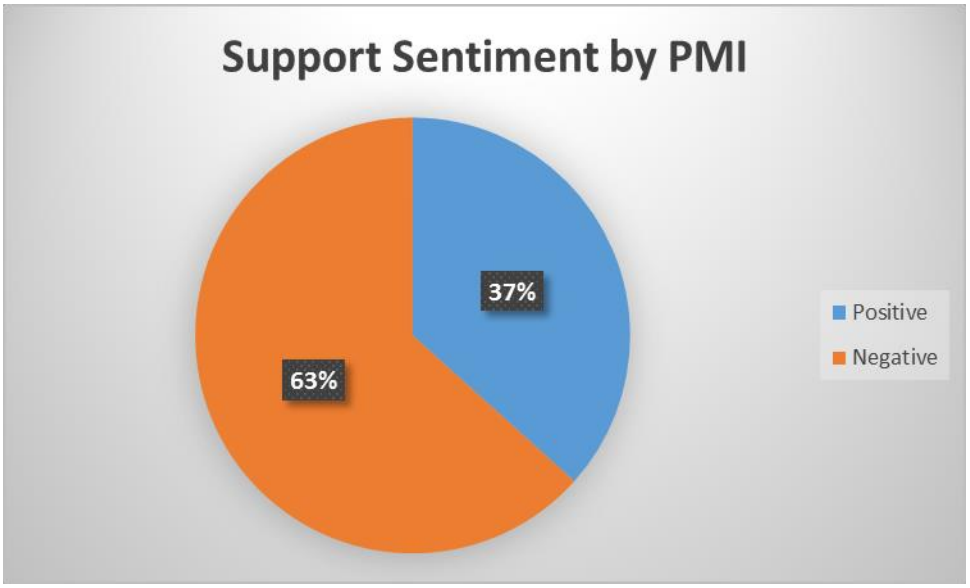

FIGURE 22 SENTIMENT FOR TOPIC SUPPORT BY PMI

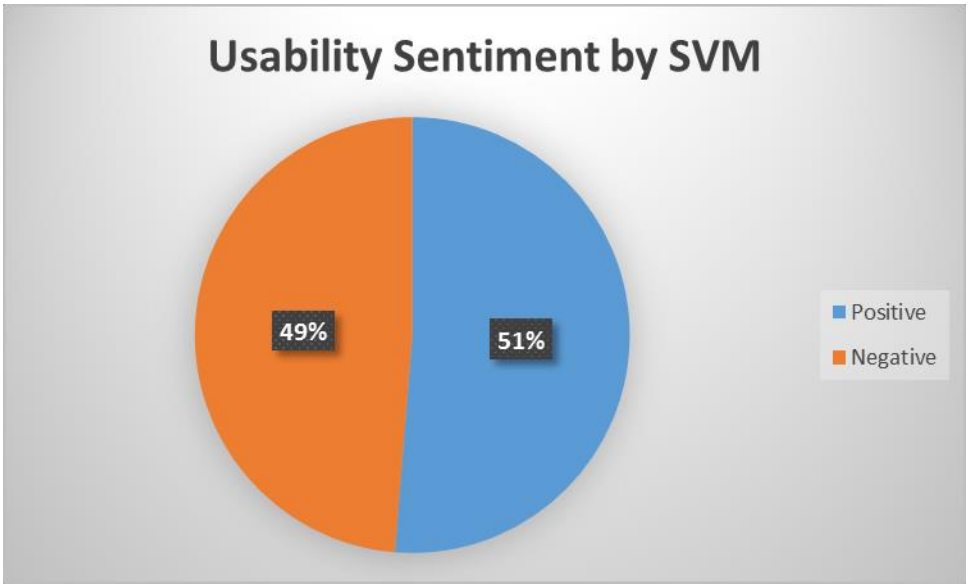

FIGURE 23 SENTIMENT FOR TOPIC USABILITY BY SVM

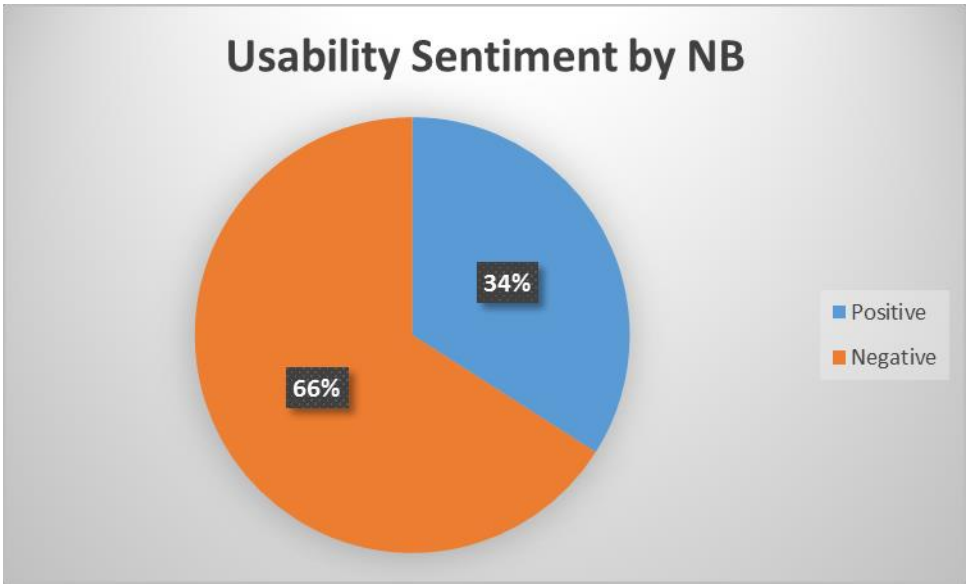

FIGURE 24 SENTIMENT FOR TOPIC USABILITY BY NB

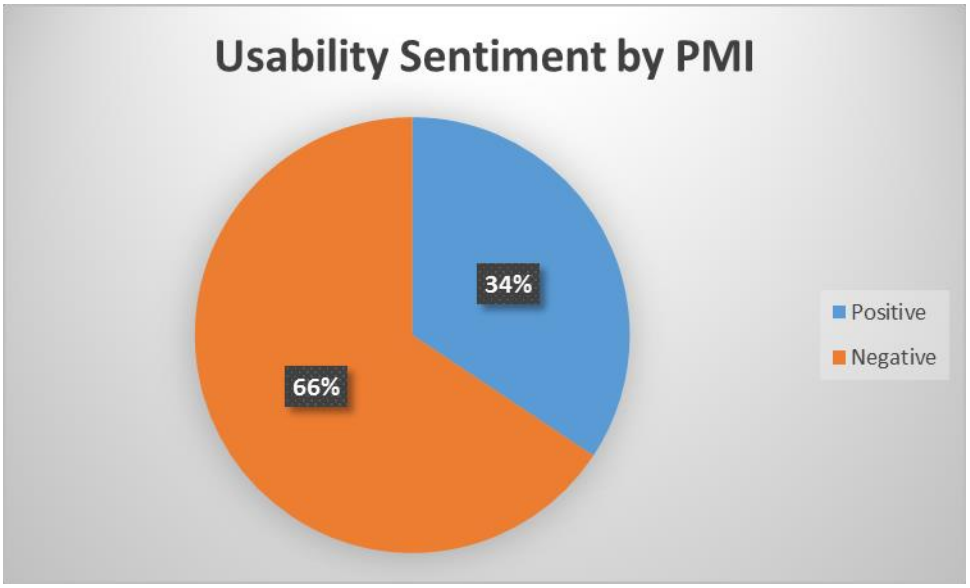

FIGURE 25 SENTIMENT FOR TOPIC USABILITY BY PMI

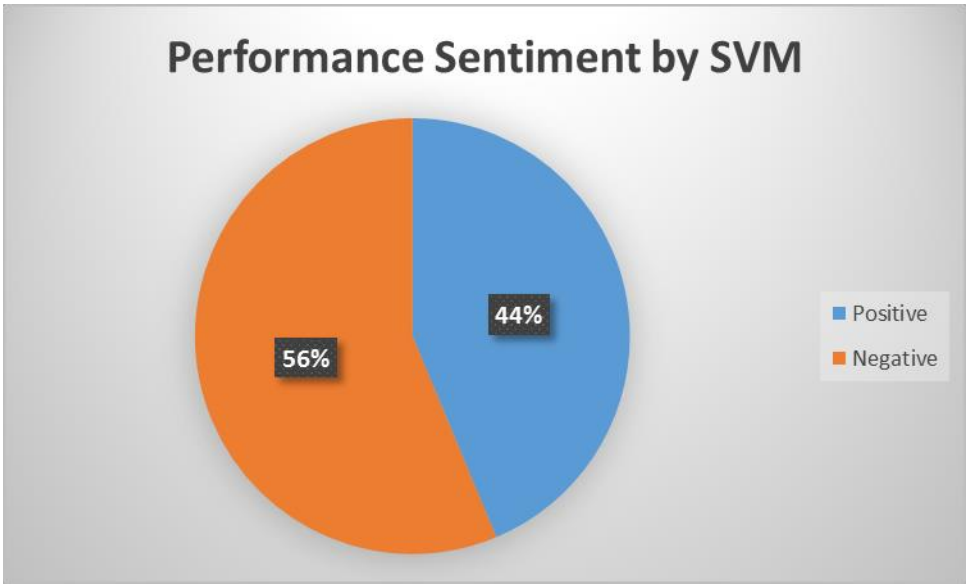

FIGURE 26 SENTIMENT FOR TOPIC PERFORMANCE BY SVM

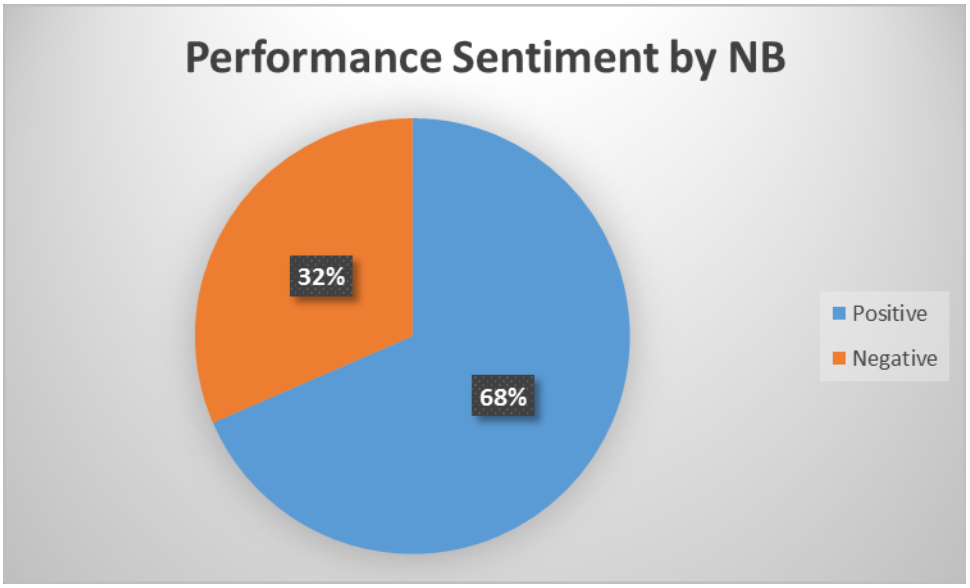

FIGURE 27 SENTIMENT FOR TOPIC PERFORMANCE BY NB

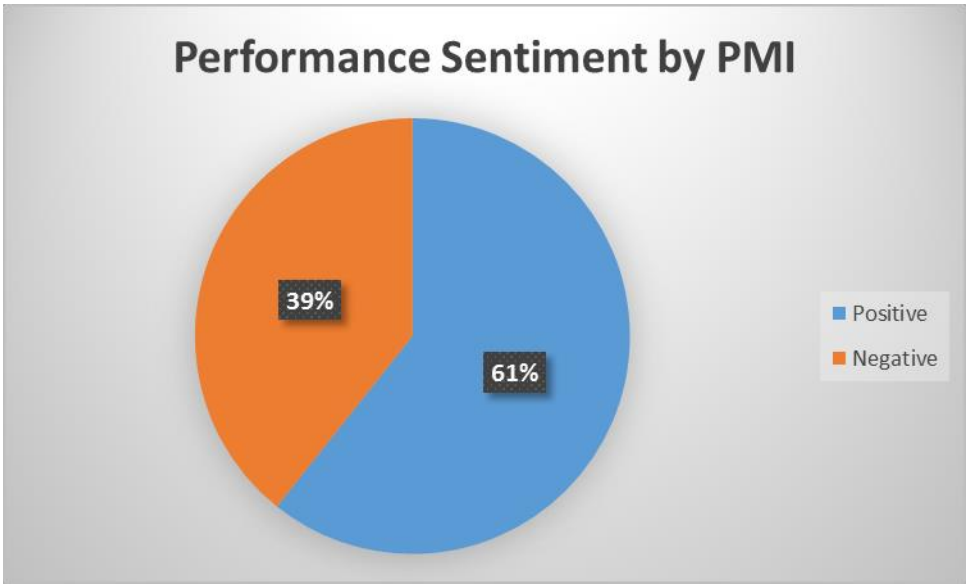

FIGURE 28 SENTIMENT FOR TOPIC PERFORMANCE BY PMI

APPENDIX D: DATA AND ORANGE CONFIGURATION FILES

All files needed to replicate results can be found here:

https://github.com/vttran/OrangeCorpus