The Pennsylvania State University

The Graduate School

College of Engineering

AMAZON FINE FOOD REVIEWS – DESIGN AND IMPLEMENTATION OF AN AUTOMATED CLASSIFICATION SYSTEM

A Thesis in

Industrial Engineering

by

Rutvik Sharedalal

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2019

The thesis of Rutvik Sharedalal was reviewed and approved* by the following:

Soundar R.T. Kumara
Allen E. Pearce/Allen M. Pearce Professor of Industrial Engineering
Thesis Advisor

Saurabh Basu
Assistant Professor of Industrial and Manufacturing Engineering

Janis Terpenny
Peter and Angela Dal Pezzo Professor
Head of the Department of Industrial and Manufacturing Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Social media has given ample opportunity to the consumer in terms of gauging the quality of the products by reading and examining the reviews posted by the users of online shopping platforms. Moreover, online platforms such as Amazon.com provides an option to the users to label a review as 'Helpful' if they find the content of the review valuable. This helps both consumers and manufacturers to evaluate general preferences in an efficient manner by focusing mainly on the selected helpful reviews. However, the recently posted reviews get comparatively fewer votes and the higher voted reviews get into the users' radars first. This study deals with these issues by building an automated text classification system to predict the helpfulness of online reviews irrespective of the time they are posted. The study is conducted on the data collected from Amazon.com consisting of the reviews on fine food. The focus of previous research has mostly remained on finding a correlation between the review helpfulness measure and review content-based features. In addition to finding significant content-based features, this study uses three different approaches to predict the review helpfulness which includes vectorized features, review and summary centric features, and word embedding-based features. Moreover, the conventional classifiers used for text classification such as Support vector machine, Logistic regression, and Multinomial naïve Bayes are compared with a decision tree-based ensemble classifier, namely Extremely randomized trees. It is found that the Extremely randomized trees classifier outperforms the conventional classifiers except in the case of vectorized features with unigrams and bigrams. Among the features, vectorized features perform much better compared to other features. This study also found that the content-based features such as review polarity, review subjectivity, review character and word count, review average word length, and summary character count are significant predictors of the review helpfulness.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Dr. Soundar Kumara, Professor of Industrial Engineering, for his valuable support and exemplary guidance throughout the course of my thesis. I am also grateful to him as his teaching has helped me acquire great knowledge and skills.

I would also like to express my gratitude to Dr. Saurabh Basu, Assistant Professor of Industrial and Manufacturing Engineering, for taking his time to review my thesis and giving constructive feedback.

I would also like to thank Dr. Janis Terpenny, Department Head of Industrial and Manufacturing Engineering, for being a part of my thesis committee.

I take this opportunity to thank my colleague Visanu Chumongkhon for sharing his knowledge and giving some insightful inputs.

Finally, I express my profound gratitude to my family and friends for their continuous support and encouragement throughout my graduate studies. This would not have been possible without all of you.

Chapter 1
Introduction

Business firms are now using online platforms and social media to market their products and services. The increasing use of social media and the internet has significantly changed the way consumers shop for products. Social media also helps customers receive better accessibility to product information. Studies infer that customers show more trust towards the online consumer reviews than the information provided by the vendors (Salehan et al., 2016). Moreover, online consumer reviews are more user-oriented and describe the products from the user's perspective.

Consumers can judge the quality of the products by reading and examining numerous online reviews available on online public platforms such as Amazon.com. However, if these reviews are not organized or presented properly, can lead the customers to an ambiguous situation. The overwhelming quantity of the reviews can affect the unbiased decision making of the consumers mainly because of the uneven quality or untruthfulness (spam) of the reviews (Salehan et al., 2016; Liu et al., 2008). To overcome this problem, Amazon.com provides 'helpful' field along with the review text which lets the user vote whether the information provided in the review was valuable or not. Consumers can use this as a reference for making a rational purchasing decision.

1.1 Motivation

It has been estimated that the 'Helpfulness Voting' system brings in Amazon.com about $2.7 billion in additional revenue (Cao et al., 2011). It is imperative for any business organization to perceive which factors determine the helpfulness of the online reviews. This can help the online business managers increase the revenue by designing and implementing an automated system for classification of the vast quantity of the online consumer reviews data.

Although this voting process is an improvement in building the automated classification system, there are some detrimental issues that need to be taken care of before proceeding further. First, reviews which are posted most recently might have very few votes or no votes at all and therefore, identifying the helpfulness of these reviews becomes very difficult. Moreover, the highest voted reviews get viewed first before the newly published reviews which are yet to be voted (Liu et al., 2008). For example, Kim et al. (2006) found in their study that, out of 20919 Amazon reviews for all MP3 player products, 38% received 3 or lesser helpfulness votes. It is also known that there is always a considerable bias in the human analysis of textual information as users like to believe what matches with their dispositions (Liu et al., 2012). Studies have also shown that the ratings that the reviews get do not convey profound information as they are either extremely high or extremely low (Ghose et al., 2007).

In light of all the above factors, it is important to build an automated text-based classification system that predicts the helpfulness of online consumer reviews irrespective of when the reviews get posted. If the newly posted reviews are instantaneously deemed ‘Helpful’ or ‘Not helpful’ by the system built on machine learning classification algorithms, it gives consumers a reasonable choice to make their purchasing decisions rather than relying solely on older reviews. Text mining and Natural Language Processing (NLP) have been used in the past to build a text classification system to identify spam reviews (Crawford et al., 2015; Shojaee et al., 2013; Lilleberg et al., 2015). The same can be applied to detect whether the reviews are helpful or not.

1.2 Problem definition

The goal of this paper is to develop an automated text-based classification system that can accurately predict the helpfulness of Amazon online consumer reviews. The problem considered is to perform a binary classification using the combination of text-based features and machine learning classification algorithms. The binary classes will be defined as - '1' being 'Helpful' and '0' being 'Not helpful'. The helpfulness measure will be determined based on the number of users who voted the review as 'helpful'. Multiple types of text-based features will be used for this study which includes matrix based vectorized features, word embedding based features, and features extracted from review and summary text such as structural, syntactic, semantic features and meta-data. There will also be an attempt to combine some of the above features with reference to the previous work. The prediction model will be obtained based on the available training data which consists of the product review text, review summary, review rating, details on the helpfulness votes and other product and user related information. The algorithms that will be used for classification purpose are- Support Vector Machine, Multinomial naïve Bayes, Logistic regression, and Extremely randomized trees. This thesis also discusses the required text pre-processing methods as they help avoid overfitting and reduce computational complexity by removing noise from the text such as common words with no or less contribution.

1.3 Organization of the thesis

The rest of the thesis is structured as follows. Chapter 2 discusses the previous work and provides a detailed explanation of the features and algorithms. Chapter 3 provides the overview of the data. Methodology is discussed extensively in chapter 4. Chapter 5 includes a detailed description of the analysis and summarizes the results. Conclusions of the study and the scope of potential future work are discussed in chapter 6.

Chapter 2
Literature survey and Background

2.1 Literature survey

2.1.1 Research in review helpfulness evaluation

The focus of previous research on the review helpfulness has mostly remained on automatizing the prediction of review helpfulness to tackle the issues regarding the enormous quantity and uneven quality of online consumer reviews. A considerable amount of work has been done in finding correlations between review helpfulness and text-based features with the use of text mining and statistical modeling approaches.

The study conducted by Salehan et al. (2016), investigated the effect of sentiment on the helpfulness of online consumer reviews. The study provided insights into the performance of online reviews by investigating the effect of sentiment polarity on the helpfulness. Their research model included both the title related and review related measures. Apart from polarity and sentiment of the reviews, the other main factors included in the model were the length and longevity of the reviews. As websites like Amazon.com sort reviews by 'most helpful' rather than 'most recent', it was justified to include longevity as one of the factors in the study. It was found that the reviews with neutral polarity affected helpfulness significantly compared to positive and negative ones. Also, the length of a review was found to be another significant factor as a longer review is expected to contain more information.

In another study, Cao et al. (2011) emphasized the question regarding why some reviews get more helpfulness votes compared to other reviews. For this, they studied the impact of basic, stylistic

and semantic characteristics of online reviews on the number of helpfulness votes they receive using the data collected from CNET Download.com. The study was conducted using Ordinal Logistic Regression model with the accuracy being determined by 3 criteria namely – Misclassification rate, AIC (Akaike's Information Criterion) and Lift ratio. It was concluded that the semantic characteristics were more influential compared to the other characteristics and reviews with extreme opinions received more helpfulness votes.

Liu et al. (2008), built a non-linear regression model using the features extracted from the IMDB movie reviews dataset. The factors considered by them were – reviewer expertise, writing style, timeliness of the reviews, length of the review, polarity of the review, and the average rating of all the reviews. The helpfulness measure was approximated as the ratio of the number of users who found the review helpful to the total number of users who voted whether the review was helpful or not. Out of all the factors considered, the factors that were deemed the most important were – reviewer expertise, writing style, and timeliness. These factors were quantified and used in the model to check their individual and combined effectiveness with the help of MSE score. The best result was obtained when the features were combined.

Kim et al. (2006) focused on performing an instantaneous assessment of the review helpfulness to provide rapid feedback to the review authors. In their study, the effect of various text-based features on the review helpfulness was studied using SVM regression modeling. The feature list included – structural features, lexical features, syntactic features, semantic features, and meta-data. It was found that the most useful features were the length of the review, unigrams of the review and the rating that the reviewer gave to the product.

To deal with the problem of assessing a large number of reviews on online market platforms, Ghose et al. (2007) built ranking mechanisms to quickly locate underlying useful reviews. This

ranking mechanism contained both user-oriented and manufacturer-oriented approaches. The review usefulness of the user-oriented approach was determined by the helpfulness votes a review obtained, whereas manufacturer-oriented approach used sales of the product to gauge the usefulness. Econometric and subjectivity analysis was performed using regression modeling on the panel data collected from Amazon.com. It was found in the study that the reviews which contained a mixture of subjective and objective elements got more helpfulness votes.

2.1.2 Research in text classification

This section sheds some light on the work done related to text mining and NLP in the domain other than that of the review helpfulness. The various text-based features and machine learning algorithms used in the domain of spam review detection have been discussed here. This section also emphasizes the use of Support Vector Machine in the text categorization problems.

Crawford et al. (2015) conducted a survey of spam detection in which they discussed features extracted from text using Natural Language Processing and compared different approaches for classifying the reviews as ‘spam’ or ‘not spam’. The features used in the study included Bag of words, Term Frequency, POS tagging, and other lexical and semantic features. The classification task was performed using algorithms such as SVM (Support Vector Machine), Naïve Bayes, and Logistic Regression. Among the algorithms used, SVM was found to perform better than the Naïve Bayes and Logistic Regression while occasionally beaten by them. Moreover, the performance was shown to be increased when multiple features were combined.

In their study of detecting deceptive opinions, Shojaee et al. (2013) came up with stylometric features to distinguish the spammer writing style. It was stated in their study that the spammers try to change their writing style and verbal usage by using simpler, short and fewer average syllables

per word. The features were categorized into 2 types, i.e. lexical and syntactic features. Example of some of these features includes the total number of tokens, average sentence length, average token length, occurrences of uppercase letters. The machine learning classification algorithms used were Support Vector Machine and Naïve Bayes. Features were analyzed separately as well as combined and compared using F-score measure. Highest accuracy was achieved when the features were combined irrespective of the algorithm applied. Moreover, SVM performed better overall compared to Naïve Bayes in all combinations of the features.

Previous research has also shown that combining the semantic features and word frequency-based features can outperform either of them used alone. In their study of text classification, Lilleberg et al. (2015) combined the word vectors obtained from the Word2vec algorithm with the TF-IDF measure of the words. The assumption was that the TF-IDF matrix does not represent the semantic relationships between the tokens which Word2vec provides. Besides this, various combinations were tried in terms of the inclusion of stop words. The ’20 newsgroup’ text data was used for the study and Linear SVC was used as the classification algorithm. Combining Word2vec with TF-IDF without the stop words rendered the highest accuracy among all the different combinations except in some rare cases where it could not outperform the TF-IDF without the stop words.

As far as machine learning classification algorithms are concerned, Naïve Bayes and Support Vector Machine perform better for the text classification problems (Vinodhini et al., 2012). Joachims (1998) has given theoretical evidence on why SVM performs well for text categorization. He tried to strengthen his argument by providing the following justifications:

1) SVM has the potential to handle the large feature spaces related to the text categorization problems as its functionality does not depend upon the dimensionality of the problem.
2) SVM is well suited for problems with sparse instances and dense concepts (High dimensionality with few non-zero entries) such as text categorization.
3) The tendency of most of the text classification problems is to be linearly separable and the idea behind the SVM is to find such linear separators.

The literature denotes that various text-based features have been used in the studies of text mining. As far as the research in review helpfulness is concerned, the focus of the studies has mostly remained on exploring review and reviewer centric characteristics such as review length, review polarity, review subjectivity, reviewer's writing style, and so on. The idea behind this was to come up with the significant features which are highly correlated to the review helpfulness measure in order to rank reviews in terms of their perceived helpfulness. Even though the word embedding based features have been used in other domains of text classification, it has remained highly unexplored in review helpfulness domain. To the best of my knowledge, the performances of vectorized features and word embedding based features are yet to be compared with the review content-based features in predicting the review helpfulness. Moreover, this study also compares the performance of a decision tree-based ensemble classifier - Extremely randomized trees - with the conventional algorithms used for text classification such as Support vector machine, Multinomial naïve Bayes and, Logistic regression. The further sections discuss the functionality of the features and algorithms that are employed in this study.

2.2 Vectorized features

This set of features uses the vectorization process to transform a collection of text documents into numerical feature vectors. We discuss the 2 variants of this representation below.

2.2.1 Count vectorizer

This is a matrix-based feature which converts the collection of text documents into a matrix of word counts (Pedregosa et al., 2011).

Table 2-1: Snapshot of count vectorizer matrix

	T1	**T2**	**T3**	**T4**
D1	F11	F12	F13	F14
D2	F21	F22	F23	F24

Table 2-1 shows the snapshot of the count vectorizer matrix where D1, D2 denotes unique text documents and; T1, T2, T3, T4 denotes the unique terms that the document contains. For example, if there are M unique documents and N unique words present in a corpus of text documents, the matrix obtained will be of the size M x N. The value in the cell denotes the number of times a specific word has appeared in a particular document. For instance, F11 denotes the frequency with which Term-1 appears in the Document-1. The columns of this matrix work as unique features when the matrix is used as an input to any machine learning classification algorithm. The limitation of this feature set is that it does not consider the semantic relation between the terms as it counts the frequency of each word separately. In this case, a slight modification called ‘n-gram’ can be used which considers multiple terms as a single entity and then calculates the frequency of its

appearance in the document. For instance, if bi-gram is used, it calculates the frequency with which any 2 terms appear together in a particular document.

2.2.2 TF-IDF vectorizer

This feature works in the same way with the only difference being that it calculates TF-IDF value of a term instead of the frequency with which it appears in a particular document. TF-IDF is basically a multiplication of two terms – Term Frequency and Inverse Document Frequency. The mathematical expressions of these terms are given below (Pedregosa et al., 2011):

$$\boldsymbol{TF(t,d) = f_{t,d}}$$

$$\boldsymbol{IDF(t,d) = \log \frac{N}{|d \in D: t \in T|}}$$

Hence, TF is denoted by the number of times term t appears in document d. Whereas, IDF is the logarithmic ratio of the total number of documents in the corpus and the number of documents in which the term t appears (Pedregosa et al., 2011). The justification for multiplying the term frequency value with the inverse document frequency is to give less weight to the commonly occurring words which do not convey much information. IDF provides a low value for frequently occurring terms as the number of documents in which the term occurs is in the denominator. Like the counter vectorizer matrix, the modifications such as ‘n-grams’ can also be applied to this feature set. While calculating the IDF value, 1 is added in the denominator as a smoothing parameter to avoid divisions by zero in the case of an unseen word.

2.3 Latent semantic analysis

Latent Semantic Analysis is a technique which explores the hidden concepts in text data by converting each document and word in a vector form with vector elements representing the concepts (Thomo et al., 2009). The idea behind the technique is to find the semantic relationships between the document-document, term-term, and term-document with the assumption that the words and documents with similar context or similar meaning will have a smaller distance between their respective vectors.

The vector representation of the terms and documents is obtained by applying Singular Value Decomposition to a bag of words matrix such as Count Vectorizer or TF-IDF Vectorizer discussed above. Let A be the count vectorizer matrix of the dimension MxN, where M is the total number of unique documents and N is the total number of unique terms in the text corpus. This matrix can be decomposed using SVD (Singular Value Decomposition) into 3 separate matrices as shown below:

$$\boldsymbol{A = D \cdot S \cdot W^T}$$

Where, D = Document matrix, S = Singular value matrix, W = Word matrix.

S is a diagonal matrix consisting of singular values along the diagonal. If the smaller singular values are ignored and replaced by zero, a new matrix with reduced dimensionality is obtained. This matrix, if multiplied with the original document and term matrices D and W, a new matrix is obtained with the same dimensions as those of matrix A but with the different elements. This new matrix has been shown to perform better than the original matrix in terms of uncovering the underlying semantic relationships (Landauer et al., 1998). The rows of this matrix are used as word vectors and columns are used as document vectors in a text mining problem.

2.4 Word embedding based features

2.4.1 Word2vec

This concept was first created by Mikolov et al. (2013) at Google. This method uses 2-layered shallow neural networks to represent words in a vector space consisting of several hundred dimensions. This algorithm has 2 variants namely, CBOW (Continuous Bag of Words) and Skip-gram.

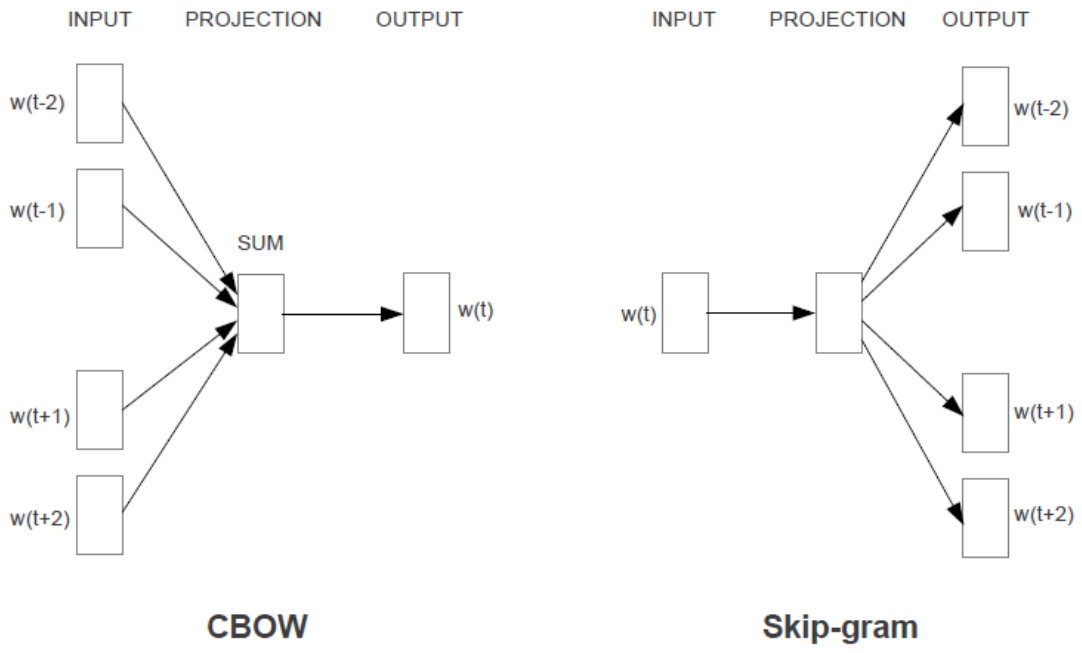


Figure 2-1: CBOW and Skip-gram model architectures (from Mikolov et al., 2013)

As it can be seen from Figure 2-1, the CBOW model predicts a word based on the surrounding context words, whereas the Skip-gram model uses the current word to predict the context words. Before the words are inserted into the model, they are vectorized using the one-hot encoder. For example, the first word in the corpus of vocabulary V will have a vector of elements $\{X_1, X_2, \ldots\ldots, X_v\}$ where X_1 is 1 and all other elements are zero. Word vectors are trained using the back-

propagation method and the Softmax function is used at the output layer to calculate the probability of the outcome (Rong X., 2014). Once the model is trained, the weights between the projection layer and the output layer represent the elements of word vectors for the words which are supposed to be predicted. As far as performance is concerned, Skip-gram is deemed better when the analysis involves capturing the semantic relationship between the words (Mikolov et al., 2013). In a text classification problem, a document or a text needs to be represented in the form of the words it includes. One possible approach is to calculate the element-wise arithmetic mean of the word vectors which yields a single vector for the document. Each of the elements of this document vector can then work as a single feature.

2.4.2 TF-IDF weighted Word2vec

This approach was used by Lilleberg et al. (2015) in their study of text classification. A simple Word2vec approach only provides the vector or the position of a word in an n-dimensional vector space but it does not tell how important it is or how frequently it appears in a particular document. The idea behind using this approach is to consider word importance along with its vector representation. In a text classification problem, the word vectors of a particular document can be multiplied with the corresponding TF-IDF values and then the average can be taken to come up with a single document vector. This can be presented mathematically as below:

$$\boldsymbol{V_D = \frac{1}{w} \times \sum_{w \in D} V_w \cdot TI_{w,D}}$$

Where, V_D = Document vector, V_w = Word vector, $TI_{w,D}$ = TF-IDF value of the word for that document

2.5 Review centric features

The review related features that have been used in this study include structural, syntactic and semantic features of the review text. The structural features of the text consist of the word count of the review text, character count of the review text, and average length of the words used in the review text, whereas the syntactic features include number of punctuation marks used in the review text, the number of proper case words used in the review text, and the number of uppercase words used in the review text.

The semantic features of the text include the polarity and subjectivity of the review text. Polarity denotes whether the text is positive, negative or neutral in the sentiment, whereas subjectivity denotes whether the text is subjective or objective in nature.

2.6 Summary centric features

Along with the review text, the text included in the summaries of the reviews has also been analyzed in this study. The analysis of the summary text involves structural and semantic features. However, the syntactic features are not considered for the summary due to the lesser content compared to that of the review text. Semantic features consist of the polarity and subjectivity of the text.

2.7 Meta-data

Meta-data features are the features which are not directly related to the linguistic features of the text. The meta-data feature that has been included in this study is the rating of the product given by the reviewer which ranges from 1 to 5.

2.8 Machine learning algorithms for classification

2.8.1 Multinomial naïve Bayes

This model comes from the family of probabilistic classifiers which are based on applying the Bayes' theorem with the naïve assumption of independence between the features. In this model, the document is represented as a bag of words and then the document in each class is modeled as samples drawn from a multinomial word distribution (Aggarwal et al., 2012). The conditional probability of a document given a class is then calculated by multiplying the probability of each observed word in the corresponding class. Finally, the Bayes' rule is applied to calculate the posterior probability of a class given a document and the class with the highest posterior probability is assigned to the document. The posterior probability can be mathematically formulated as shown below, assuming there are m unique classes and a document is represented by n-dimensional term vector:

$$\boldsymbol{P(C_i|D) \propto P(C_i) \cdot \prod_{t=1}^{n} P(x_t|C_i)}$$

Where, i = 1,2,3, …., m and t=1,2,3, …., n.

2.8.2 Logistic regression

Logistic regression is a regression-based linear classifier which is used to model discrete response variables (Aggarwal et al., 2012). The coefficients of the regression model are obtained by optimizing a conditional likelihood function. As in the case of Multinomial Naïve Bayes model, the class is assigned to a document which has a higher posterior probability. In this model, the posterior probability of a class given a document is mathematically formulated as shown below given the set of coefficients A and b:

$$P(C_i|X_j) = \frac{e^{(A \cdot X_i + b)}}{1 + e^{(A \cdot X_i + b)}}$$

The above posterior probability can be transformed using a logistic function and represented by the linear combination of the features. This helps in conceiving the problem in terms of simple linear regression.

2.8.3 Support vector machine

The primary principle behind the Support Vector Machine is to sample and find a separator in a feature space which can optimally separate the different classes (Aggarwal et al., 2012). The problem of finding the best hyperplane is an optimization problem and SVM uses quadratic programming to solve this problem.

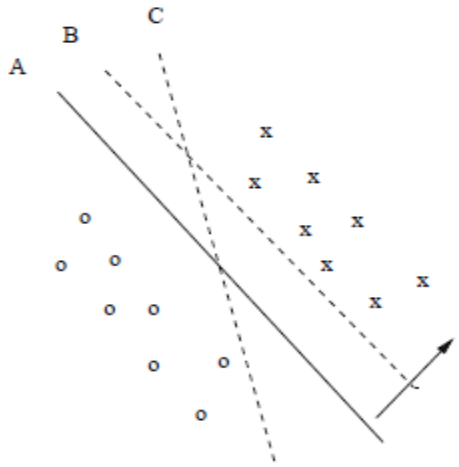


Figure 2-2: Maximum margin of separation for SVM (from Aggarwal et al., 2012)

As it can be seen in the Figure 2-2, hyperplane A provides the maximum margin of separation as the normal distance of any of the data points irrespective of the class, is the largest from this hyperplane. Moreover, the normal vector to this hyperplane is a direction in which the maximum

discrimination is obtained in the feature space. Since SVM utilizes an appropriate combination of the features to come up with the optimal direction of separation in the feature space, it is considered highly robust to the high dimensionality of the feature set (Aggarwal et al., 2012).

2.8.4 Extremely randomized trees

This algorithm was first proposed by Geurts et al. (2006). It belongs to the family of decision trees and is a variant of the random forest algorithm. Like the random forest, it is an ensemble of trees but it differs in a way that it uses the whole learning sample for a single tree rather than a bootstrap sample. Moreover, it splits the nodes by choosing cut-off points randomly, whereas random forest splits the nodes optimally by using the Gini index or Information gain criterion (Geurts et al., 2006). The rationale behind this algorithm is to reduce the variance of the decision trees by introducing randomization in both the features and cut-off point selection during node splitting. For each node, the cut-off points are randomly generated for each of the sampled features and the point with the best score is considered as a cut-off point for that particular node. The score is calculated by the normalization of information gain. Geurts et al. (2006) have also pointed out in their paper that extremely randomized trees are computationally efficient than the random forest as it decides the cut-off points randomly rather than optimally.

Chapter 3
Overview of the data

3.1 Dataset description

The dataset is provided by Kaggle website and was collected and published first by McAuley et al. (2013) in their research related to online reviews. The dataset consists of reviews on fine food posted on Amazon.com. It has a total of 568,454 reviews on 74,258 products. The attributes and their descriptions are discussed in Table 3-1 below:

Table 3-1: Feature description

Review attribute	Description	Variable type
Product ID	Unique identifier for the product	Categorical
User ID	Unique identifier for the user	Categorical
Profile Name	Profile of the user	Text
Helpfulness Numerator	Number of users who found the review helpful	Numerical
Helpfulness Denominator	Number of users who voted whether the review was helpful or not	Numerical
Score	Rating between 1 and 5	Ordinal
Time	Timestamp of the review	Numerical
Summary	Brief summary of the review	Text
Text	Text of the review	Text

3.2 Overview of the review and summary text

The review text contains a detailed description of the product from the user's perspective. Moreover, it also describes the overall sentiment of the user toward the product apart from the description of the product itself. Some of the examples of the review text are shown below:

1) *"This is great stuff. Made some really tasty banana bread. Good quality and lowest price in town."*
2) *This coffee is great because it's all organic ingredients! No pesticides to worry about plus it tastes good, and you have the healing effects of Ganoderma.*
3) *These condiments are overpriced and terrible. The classic is disgustingly sweet. The spiced tastes like a bad spicy marinara sauce from a chain restaurant.*

On the other hand, the summary text represents a user's sentiment in a very confined manner. The information is conveyed in few words in this case. Some of the examples of the summary text are shown below:

1) *"Best deal ever!"*
2) *"Waste of money"*
3) *"Great beans!!!"*
4) *"Big disappointment"*

In this study, one of the approaches uses the features extracted from the review and summary text in order to come up with the features that contribute significantly in predicting the review helpfulness.

3.3 Overview of the review helpfulness measures

As can be seen in Table 3-1, the helpfulness related measures included in the data are ‘Helpfulness numerator’ and ‘Helpfulness denominator’. ‘Helpfulness numerator’ denotes the total number of users who found the review helpful and ‘Helpfulness denominator’ denotes the total number of people who voted whether the review was helpful or not. Earlier Amazon.com used to provide both the ‘Yes’ and ‘No’ options to its users in order for them to vote for the helpfulness. However, it now provides only the ‘Yes’ option to its users. Hence, if a user finds a review helpful, he votes for it otherwise he does not do anything. In order to make this study relevant, the process of response variable generation will solely be based on the ‘Helpfulness numerator’ measure which denotes the total number of users who voted in the favor of helpfulness. However, ‘Helpfulness denominator’ measure will be used as a reference to discard the reviews for which the users did not vote at all. Review which has ‘Helpfulness numerator’ value more than a specific threshold will be considered as ‘Helpful’ otherwise it will be considered as ‘Not helpful’. The process of response variable generation is discussed broadly in the next chapter.

3.4 Overview of the text semantics

In the approach consisting of the review and summary centric features, there is an attempt to find a correlation between the semantics of the text and the review helpfulness measure through the polarity and subjectivity of the review and summary text. The polarity of the text denotes how positive or negative the sentiment of the text is, whereas subjectivity denotes how subjective or objective the review statement is. In this study, the polarity is quantified in the range [-1,1] where ‘-1’ denotes extremely negative sentiment and ‘1’ denotes extremely positive sentiment. On the other hand, subjectivity is quantified in the range [0,1] where ‘0’ denotes ‘highly objective’ and ‘1’ denotes ‘highly subjective’ statement.

Chapter 4
Methodology

This section discusses in detail the process of building an automated text classification system for predicting the helpfulness of online fine food reviews posted on Amazon.com. It encapsulates the following steps:

1) Performing exploratory data analysis for generating the binary response variable
2) Explaining various text preprocessing steps which are used to remove noisy terms
3) Explaining the model training procedure and the approaches used for it
4) Explaining the model validation and model selection procedures

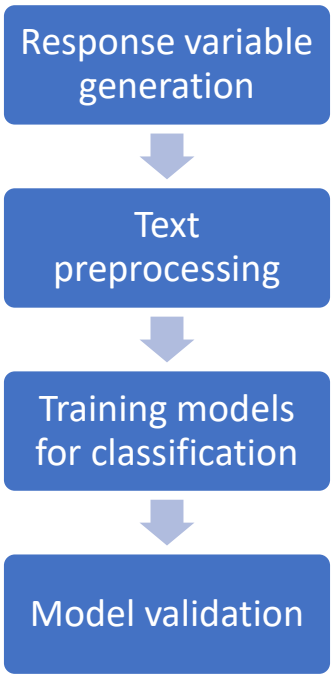


Figure 4-1: Methodology flowchart

4.1 Exploratory data analysis

This section discusses the data manipulation performed before training the model and also describes the process of binary response variable generation.

As it can be seen in Table 3-1, the attribute 'Helpfulness Denominator' denotes the number of users who voted whether they found the review helpful or not. Since the goal of this study is to build a model which predicts whether the review is helpful or not, the reviews which did not get any vote - either helpful or not helpful, will not be considered in the study. This reduces the number of observations from the original 568,454 to 298,402. Now, the next step is to examine the distribution of the attributes 'Helpfulness Numerator' and 'Score' by looking at their descriptive statistics.

Table 4-1: Descriptive statistics of 'Helpfulness Numerator' and 'Review Score' attributes

	Helpfulness Numerator	**Review Score**
Mean	3.321962	3.979310
Standard Deviation	10.288337	1.461233
Min.	0	1
1st quartile	1	3
Median	1	5
3rd quartile	3	5
Max.	866	5

The attribute 'Helpfulness Numerator' denotes the number of users who indicated that the review was helpful. In this study, this attribute will be considered as a reference for generating the binary response variable. As Table 4-1 shows, the median value of this attribute is '1'. Let us now evaluate the frequency distribution of the attribute with its median working as a threshold.

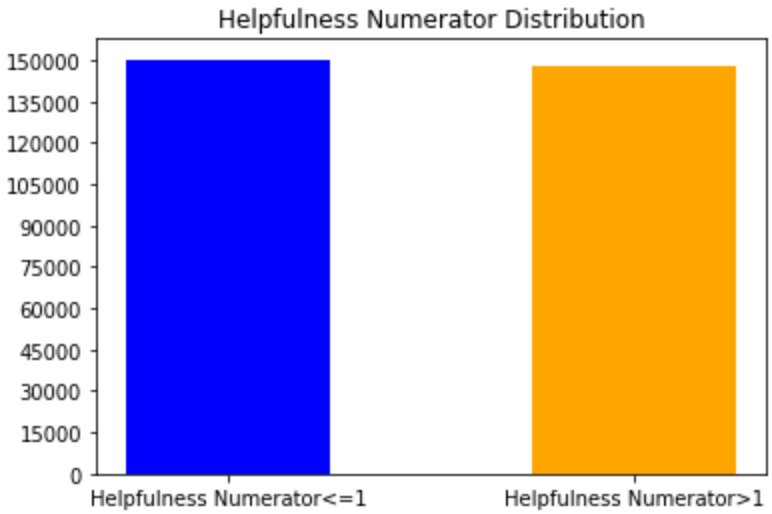


Figure 4-2: Frequency distribution of 'Helpfulness Numerator' attribute

As it can be inferred from Figure 4-2, dividing the 'Helpfulness Numerator' attribute into 2 parts with threshold '1' gives us the set of observations of almost equal size. The specific sample sizes are 150241 and 148161 for values less than or equal to 1 and greater than 1 respectively. Hence, this particular threshold will be used as a reference for generating the binary response variable as it helps avoid the intricacies associated with the class imbalance problem. From now on, the class with the values 'less than or equal to 1' will be referred as class '0' (Not helpful) and the one with the values 'greater than 1' will be referred as class '1' (Helpful).

The next step is to examine the distribution of the binary class attribute in terms of the 'Score' attribute. First of all, let us examine the distribution of the 'Score' attribute and then we will show how the binary classes are distributed for each value of the score attributes. As described in Table 3-1, 'Score' attribute denotes the rating given to a particular product by the reviewer in his review and it consists the integer values from 1 to 5 with '1' being 'extremely low' and '5' being 'extremely high'.

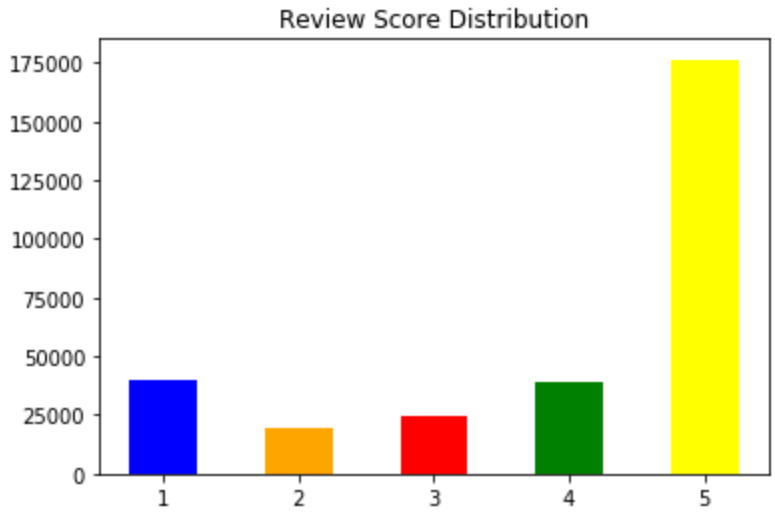


Figure 4-3: Distribution of 'Review Score' attribute

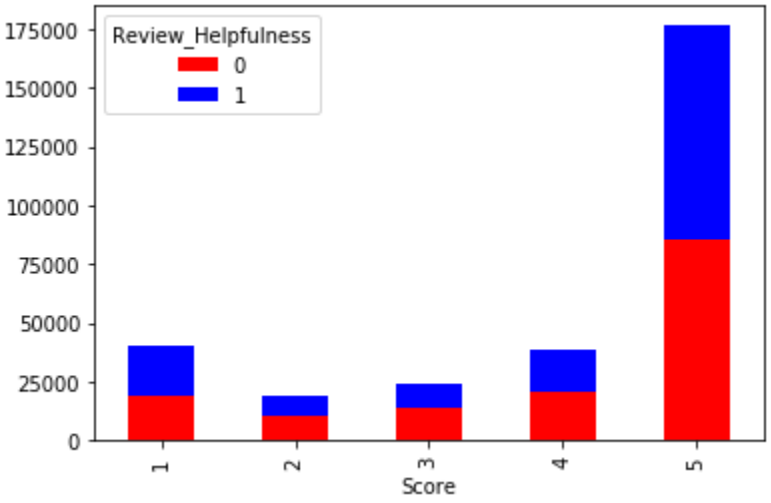


Figure 4-4: Distribution of binary response variable in terms of 'Review Score' attribute

As it can be seen from the Figure 4-3, the distribution of the 'Score' attribute is highly skewed towards the score '5' which means the majority of the reviews have 'extremely high' rating. Moreover, Figure 4-4 shows that the distribution of binary class is the same for each value of the score attribute. Hence, it can be inferred that the 'Score' attribute does not contribute much in predicting the helpfulness of the reviews.

4.2 Text preprocessing

Text preprocessing is highly essential in working with text classification problems. It helps in improving the computational efficiency and avoiding the overfitting problem by eliminating the noisy features. We discuss the various text preprocessing methods that are applied before training the model in the next sections.

4.2.1 Tokenization

Tokenization is basically the process of splitting the sentences into words. Each word is then considered as a separate token. This is the first step in text preprocessing on which all the subsequent steps are built (Hotho et al., 2005).

4.2.2 Converting uppercase tokens into lowercase

Since the semantics of a word or a phrase do not depend upon which case the word is written in, the uppercase words are converted to lowercase to avoid potential duplication of the words. For example, the words ‘DOG’ and ‘dog’ convey the same meaning. This conversion helps in reducing the dimensionality of the feature set.

4.2.3 Removal of punctuation marks

In this step, the punctuation marks are removed from the text as they do not provide any extra information while extracting the semantics from the text (Hotho et al., 2005).

4.2.4 Stop words removal

In natural language processing, stop words are the most commonly used words which do not convey much meaning (Hotho et al., 2005). Some of the examples of the stop words include short

function words such as ‘a’, ‘an’, ‘the’, ‘is’, ‘are’, ‘which’, ‘at’ and, ‘on’. ‘NLTK’ library of python is used in this study to remove stop words.

4.2.5 Lemmatization

Lemmatization is a text normalization process of reducing the inflectional form of words into base or dictionary form which is called ‘lemma’. For example, lemmatization converts the word ‘ran’ into its base form ‘run’. In lemmatization, complete morphological analysis of words is done to ensure that the base word belongs to the dictionary (Manning et al., 2008). In this way, lemmatization has a slight edge over its counterpart ‘stemming’ method which usually removes the prefix or suffix associated with the word. The root word in stemming is not required to be a valid word from the language. Keeping this in mind, lemmatization will be used in this study to reduce the inflectional form of words. This study uses ‘TextBlob’ library of python programming language for performing lemmatization.

4.3 Model training

In this study, 3 different approaches are used to perform the review classification task with each of them combined with 4 different classification algorithms. This will provide us with 12 combinations of the result. Moreover, each approach in itself contains multiple sets of features which are also compared in terms of accuracy. This section discusses each approach in detail.

In the first approach, the vectorized features will be used to perform classification. As described in the chapter-2 of this thesis, the corpus of documents is represented in the matrix format with row denoting the documents and column denoting the unique words of the corpus. The values of matrix elements can be term frequency or the combination of term frequency and inverse document frequency, depending upon the approach. Moreover, applying dimensionality reduction technique

such as Latent Semantic Analysis (LSA) has proved to capture the underlying semantic relationship between words better than the original matrix (Landauer et al., 1998). Table 4-2 outlines the different features used in this approach[1].

Table 4-2: Description of vectorized features

Feature	Description
Uni-gram count vectorizer	Matrix with word frequency as an element and single word as a feature
Uni-gram + bi-gram count vectorizer	Matrix with word frequency as an element and both single and paired words as features
Uni-gram TF-IDF vectorizer	Matrix with a combination of word frequency and inverse document frequency as an element and single word as a feature
Uni-gram + bi-gram TF-IDF vectorizer	Matrix with combination of word frequency and inverse document frequency as an element and both single and paired words as features
Uni-gram TF-IDF vectorizer with LSA	Uni-gram TF-IDF vectorizer with dimensionality reduction
Uni-gram + bi-gram TF-IDF vectorizer with LSA	Uni-gram + bi-gram TF-IDF vectorizer with dimensionality reduction

In the second approach, the set of features include text-based features such as structural features, syntactic features, features quantifying sentiment behind the text, and meta-data. Both the review content and review summary content are considered for the feature extraction process. The idea in this approach is to find the set of significant features in addition to performing review classification. Text preprocessing is not performed while extracting the structural and syntactic features as it defies the whole purpose of extracting these features. For example, syntactic features

[1] Refer chapter-2 of this paper for detailed explanation of vectorized features and LSA

include the count of punctuation marks and uppercase words. However, text preprocessing is performed while extracting semantic features as the idea is to remove noisy words which do not contribute much to the meaning of the text. Table 4-3 describes the features used in this approach.

Table 4-3: Description of review and summary centric features

Type of feature	Feature example
Structural features	Character count of the review text
	Word count of the review text
	Average length of the words used in the review text
	Character count of the summary text
	Word count of the summary text
Syntactic features	Count of punctuation marks in the review text
	Count of proper case words in the review text
	Count of uppercase words in the review text
Features quantifying text sentiment	Polarity of the review text
	Subjectivity of the review text
	Polarity of the summary text
	Subjectivity of the summary text
Meta-data	Review score rating

Here, polarity denotes whether the text is negative, positive or neutral, whereas subjectivity denotes whether the text is subjective or objective.

The third and final approach involves the use of word embedding based features. In this approach, a text document is presented in a vector form by performing vector arithmetic on words it consists of. The words are converted into vectors by using Word2vec algorithm which employs a shallow neural network for training the word vectors (Mikolov et al., 2013). In this approach, a combination of word embedding matrix and TF-IDF matrix is also tried. The idea behind using this combination is that the Word2vec only gives the position of a word in a vector space but it does not denote how

frequently it appears in different documents. Moreover, 2 variants of the Word2vec algorithm namely, CBOW (Continuous bag of words) and skip-gram are also compared in this study. Table 4-4 outlines the features used in this approach[2].

Table 4-4: Description of word embedding based features

Feature	Feature description
Word2vec with CBOW	Generating word vector using CBOW variant of Word2vec and then combining the vectors
Word2vec with skip-gram	Generating word vector using skip-gram variant of Word2vec and then combining the vectors
TF-IDF weighted Word2vec with CBOW	Generating word vector using CBOW variant of Word2vec, multiplying the vectors with corresponding TF-IDF values and then combining the vectors
TF-IDF weighted Word2vec with skip-gram	Generating word vector using skip-gram variant of Word2vec, multiplying the vectors with corresponding TF-IDF values and then combining the vectors

It is imperative to scale the features once they are extracted. Since SVM is a distance-based matrix, Hsu et al. (2003) recommend scaling features to the range [-1,1] or [0,1] in order to avoid the domination of features with greater numeric ranges over the ones with smaller numeric ranges. Scaling also helps in achieving faster optimization in the case of logistic regression as it uses the gradient descent method for maximizing the likelihood function. In this study, min-max normalization is used to scale the features in the range of [0,1]. Sparse features such as vectorized features are not scaled as the sparsity is lost due to scaling (Pedregosa et al., 2011).

[2] Refer chapter-2 of this thesis for detailed explanation on word embedding based features

Table 4-5 lists the classification algorithms used in this study.

Table 4-5: List of classification algorithms used in the study

Algorithm family	Algorithm
Probabilistic classifiers	Multinomial naïve Bayes
Linear classifiers	Logistic regression, Support vector machine
Ensemble classifiers	Extremely randomized trees

4.4 Model validation

This section discusses model validation using the train and test sets and sheds light on the model performance measurement in order to perform model comparison.

This study uses the hold-out method to split the original dataset into train and test sets. The dataset is shuffled and divided into train and test sets with the ratio 7:3 for all the approaches used in the study. The models are trained using the samples of train set and its performance is measured by applying the trained model on the unseen samples of the test set. The simple hold-out method was preferred over the more sophisticated cross-validation method such as k-fold validation due to computational constraints. The hold-out method was chosen with the assumption that the sample size of the train set is sufficiently large to avoid the overfitting problem.

This study deals with the binary classification of the review texts. Hence, the potential performance measures for this study include accuracy, precision, sensitivity, specificity, and area under ROC (Receiver Operating Characteristics) curve. In a binary classification setting, correction of classification can be evaluated using the confusion matrix which is shown in Table 4-6.

Table 4-6: Confusion matrix

Actual class	Predicted as positive	Predicted as negative
Positive	True positive (tp)	False negative (fn)
Negative	False positive (fp)	True negative (tn)

The parameters of the confusion matrix can be described as below:

1) True positive – sample belonging to positive class predicted as positive
2) True negative – sample belonging to negative class predicted as negative
3) False positive – sample belonging to negative class predicted as positive
4) False negative- sample belonging to positive class predicted as negative

This explanation of the parameters provides a basis for understanding the evaluation purposes of different performance measures used for performing binary classification. Table 4-7 lists and explains these measures.

Since the goal of this study is to extract and label reviews based on their usefulness, it is not affordable to misclassify the review of either class into its opposite class. Moreover, it is also imperative to evaluate the overall performance of the classification models. Hence, the performance measures used in this study are overall accuracy and AUC (area under the curve).

Table 4-7: Performance measures for a binary classifier (from Sokolova et al., 2009)

Measure	Formula	Evaluation focus
Accuracy	$\frac{(tp+tn)}{(tp+tn+fp+fn)}$	Overall effectiveness of a classifier
Precision	$\frac{tp}{(tp+fp)}$	Class agreement of he data labels with the positive labels given by the classifier
Recall (Sensitivity)	$\frac{tp}{(tp+fn)}$	Effectiveness of a classifier to identify positive labels
F-score	$\frac{(\beta^2+1)\cdot tp}{(\beta^2+1)\cdot tp+\beta^2\cdot fn+fp}$	Relation between data's positive labels and those given by a classifier
Specificity	$\frac{tn}{(fp+tn)}$	How effectively classifier identifies negative labels
AUC	$\frac{1}{2}(\frac{tp}{tp+fn}+\frac{tn}{tn+fp})$	Classifier's ability to avoid false classification

As it can be inferred from Table 4-7, the accuracy measure is defined as the ratio of correctly classified samples to the total number of samples. It denotes the overall performance of the classification model irrespective of the individual classes. Talking about the area under the ROC curve, it tries to strike the balance between 2 performance measures namely, true positive rate and false positive rate (James et al., 2013). True positive rate denotes the proportion of correctly identified as positives among all positives, whereas false positive rate denotes the proportion of incorrectly identified as positives among all negatives. The functionality of the ROC curve can be inferred from Figure 4-5.

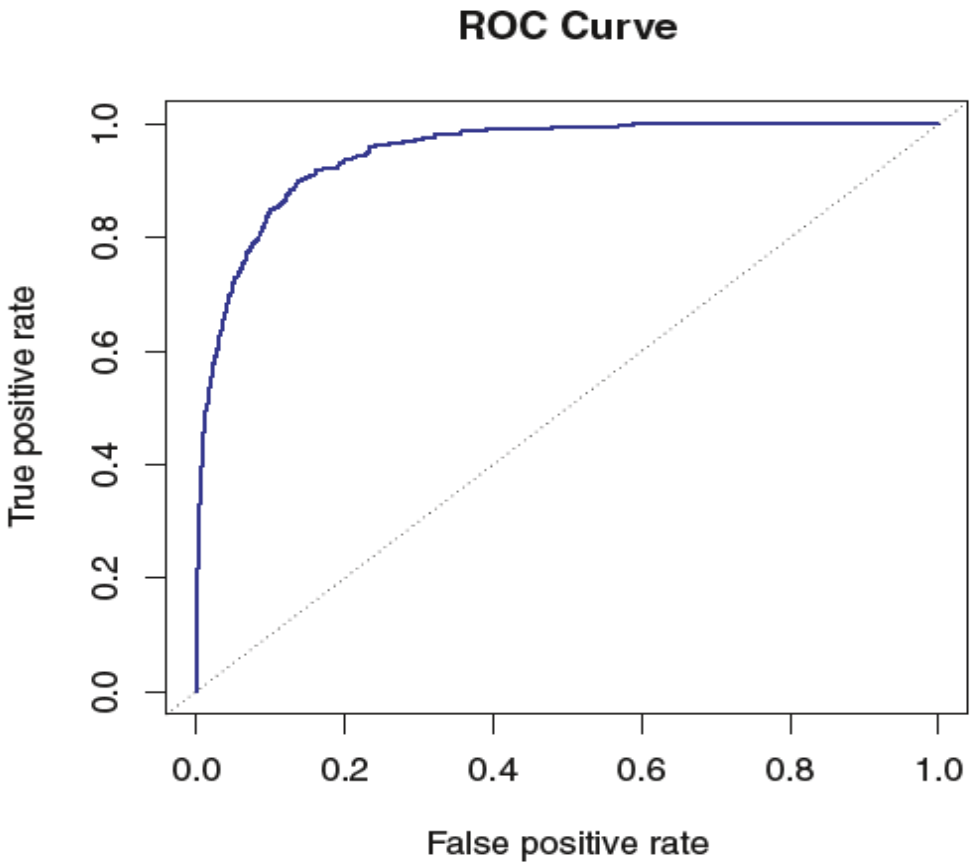


Figure 4-5: ROC (Receiver Operating Characteristics) curve (from James et al., 2013)

In the ROC curve, the performance of a classifier is measured by the area under the curve. As can be seen in Figure 4-5, the false positive rate also increases to increase the true positive rate. Hence, the curve should bend at the top left corner for an ideal classifier (James et al., 2013). This helps in increasing the true positive rate without increasing the false positive rate much. It also means that the more the area under the curve, the better the performance of the classifier is. ROC curve also helps in choosing the optimal threshold as the threshold can be varied along the curve to obtain the true positive rate and false positive rate as per the problem in hand.

Chapter 5
Analysis and Results

5.1 Analysis of vectorized features

This section discusses the analysis performed using the vectorized features. In this approach, the features are obtained by representing the review text into a matrix form where rows denote the unique reviews and columns denote the unique words used in the corpus of review texts. Two variants of this matrix are used to perform the analysis which are – count vectorizer and TFIDF vectorizer. In count vectorizer, matrix element denotes the frequency with which a word appears in a particular review, whereas element of a TFIDF matrix denotes the value obtained by multiplying the term frequency and inverse document frequency values of a word for that particular review. Moreover, this analysis also involves considering a single word(unigram) or a pair of adjacent words(bigrams) as columns of these matrices. Choosing a pair of words helps in considering the context words which could prove beneficial in uncovering the underlying semantic relationships between the words. In this study, both unigrams and a combination of unigrams and bigrams are analyzed and compared in terms of classification performance. Moreover, a TF-IDF matrix obtained with a dimensionality reduction technique such as Latent Semantic Analysis (LSA) is also included in this analysis which in the past has proven to be more efficient compared to the original TF-IDF matrix (Landauer et al., 1998).

The first step in the analysis is to perform text preprocessing in order to reduce dimensions of the matrix by removing noisy words. The next step is to generate the train and test sets using random shuffling with the ratio 7:3. This is followed by generating matrices in which matrix forms of review texts are obtained using ‘sci-kit learn’ library of Python programming language. Moreover,

this library is also used throughout this study to implement classification algorithms along with the hyperparameter tuning. Features are not scaled for sparse matrices as sparsity is lost after scaling. However, it is performed when a reduced dimensional matrix is obtained using Latent Semantic Analysis. Table 5-1 shows the hyperparameters used for the classification algorithms.

Table 5-1: Classifiers and their hyperparameters for vectorized features

Algorithm	Hyperparameters
Multinomial naïve Bayes	Additive smoothing parameter = 1
Logistic regression	Regularization parameter C =1, penalty norm = L^1
Support vector machine	Kernel= linear, regularization parameter C=1, penalty norm = L^1
Extremely randomized trees	Number of trees = 10, features considered while splitting a node = $\sqrt{\text{Total number of features}}$

Multinomial naïve Bayes classifier calculates the probability of a class given a document by multiplying the posterior probabilities of the words appearing in that class. Hence, if it is used to calculate the probability of a word which has never appeared before in that particular class of documents, it assigns zero probability for that word which in turn assigns zero probability for that particular class. To avoid this, a smoothing parameter with the value 1 is added to the word count while calculating the probability. Talking about logistic regression and support vector machine, the regularization parameter used is 1 and the penalty norm used is 'L-1' norm. 'L-1' norm is used as it aids in reducing the dimensionality of the matrix by providing a sparse solution with very few non-zero coefficients (Pedregosa et al., 2011). However, 'L-2' norm is used when applying LSA as the dimensionality reduction, in this case, is done by LSA itself. Classifying with extremely randomized trees classifier involves the use of 10 decision trees and the number of features considered while splitting a node is calculated taking a square root of the total number of features

used in the study. Geurts et al. (2006) found in their study that calculating the number of features in this manner tends to provide near-optimal accuracy and good computational complexity in classification problems. Throughout this study, the hyperparameters for Multinomial naïve Bayes and extremely randomized trees classifier will remain the same, whereas it will change for support vector machine and logistic regression according to the requirements for dimensionality reduction.

Now, let us first discuss the analysis performed using the count vectorizer matrix. Table 5-2 shows the results obtained using count vectorizer matrix.

Table 5-2: Performance evaluation of count vectorizer matrices

	Multinomial naïve Bayes	**Logistic regression**	**Support vector machine**	**Extremely randomized trees**
Uni-gram count vectorizer	Accuracy:62.40%	Accuracy:65.51%	Accuracy:66.86%	**Accuracy:70.38%**
	AUC: 0.6234	AUC: 0.6547	AUC: 0.6683	**AUC: 0.7033**
Uni-gram + bi-gram count vectorizer	**Accuracy:71.09%**	Accuracy:70.31%	Accuracy:70.71%	Accuracy:70.38%
	AUC: 0.7109	AUC: 0.7030	AUC: 0.7070	AUC: 0.7033

As it can be seen from Table 5-2, extremely randomized trees classifier performs better than the conventional classifiers used for text classification in the case of count vectorizer with unigrams. However, it does not perform better than the conventional classifiers in the case of high dimensional count vectorizer containing both unigrams and bigrams. Moreover, the overall accuracy improves when bigrams are used in combination with unigrams.

Table 5-3: Performance evaluation of TF-IDF vectorizer matrices

	Multinomial naïve Bayes	**Logistic regression**	**Support vector machine**	**Extremely randomized trees**
Uni-gram TF-IDF vectorizer	Accuracy:63.99%	Accuracy:66.98%	Accuracy:66.79%	**Accuracy:70.38%**
	AUC: 0.6395	AUC: 0.6696	AUC: 0.6674	**AUC: 0.7033**
Uni-gram + bi-gram TF-IDF vectorizer	Accuracy:70.67%	Accuracy:70.95%	Accuracy:70.76%	**Accuracy:71.27%**
	AUC: 0.7067	AUC: 0.7093	AUC: 0.7074	**AUC: 0.7122**

Table 5-3 evaluates the performance of TF-IDF vectorizers. As in the case of count vectorizer, extremely randomized trees classifier performs better than the conventional classifiers when only unigrams are used but fails to do so when bigrams are combined with unigrams. Moreover, combining bigrams with unigrams improves overall accuracy irrespective of the classifier used.

Table 5-4: Performance evaluation of TF-IDF vectorizer matrices with LSA

	Multinomial naïve Bayes	**Logistic regression**	**Support vector machine**	**Extremely randomized trees**
Uni-gram TF-IDF vectorizer with LSA	Accuracy:51.42%	Accuracy:57.81%	Accuracy:57.66%	**Accuracy:69.80%**
	AUC: 0.5119	AUC: 0.5772	AUC: 0.5758	**AUC: 0.6976**
Uni-gram + bigram TF-IDF vectorizer with LSA	Accuracy:51.37%	Accuracy:57.42%	Accuracy:57.59%	**Accuracy:69.98%**
	AUC: 0.5111	AUC: 0.5733	AUC: 0.5751	**AUC: 0.6994**

As it can be inferred from Table 5-4, extremely randomized trees classifier outperforms the conventional classifiers when LSA is applied on TF-IDF vectorizer. Moreover, accuracy does not improve when TF-IDF vectorizer is used along with LSA. There is also not any difference between the performances of TF-IDF vectorizer containing only unigrams and the one that contains both unigrams and bigrams.

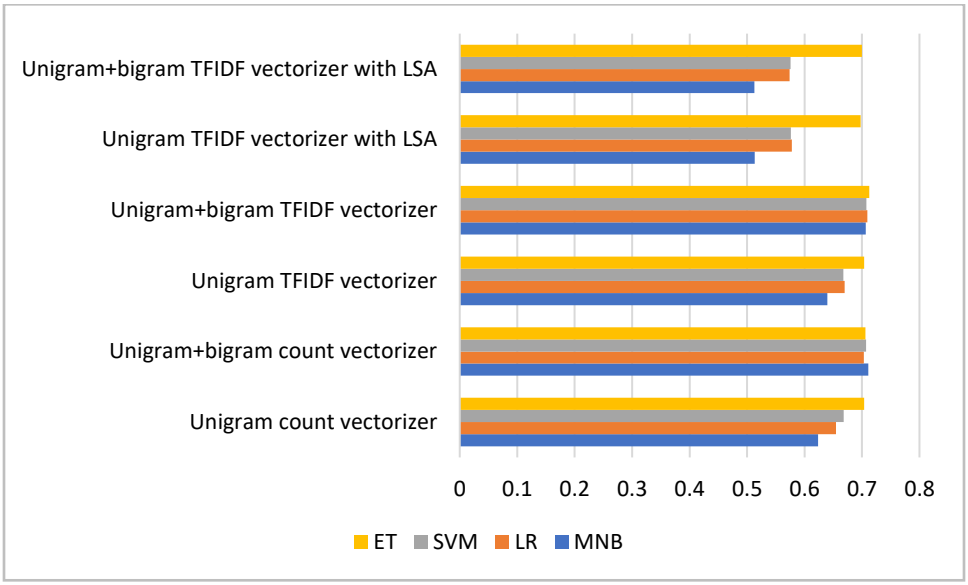


Figure 5-1: Performance comparison of vectorized features

Figure 5-1 compares the performances of vectorized features. The score is calculated by taking the average of overall accuracy and AUC measures. As it can be inferred, extremely randomized trees classifier consistently performs well irrespective of the vectorizer used. Moreover, it outperforms the conventional classifiers used for text classification when applied to matrices with comparatively lower dimensions such as vectorizers with LSA and unigram vectorizers. However, conventional classifiers perform as well as the extremely randomized trees classifier when it comes to matrices with larger dimensions such as matrices with a combination of unigrams and bigrams.

5.2 Analysis of review centric and summary centric features

This section outlines the analysis performed using the review and summary centric features which include structural features, syntactic features, semantic features, and meta-data. The idea behind performing this analysis is to come up with the list of features that contribute significantly towards predicting the helpfulness of the reviews.

Talking about the analysis procedure, the features are extracted for both the review text and summary text. For this purpose, 26 observations which do not contain any review summary text are not taken into consideration. There are also some important points discussed below that need to be considered while performing the analysis:

1) Text preprocessing is not performed while extracting the structural and syntactic features as it defies the whole purpose of including these features in the study.
2) Text preprocessing is however performed while extracting the semantic features as the idea is to discard words that do not contribute much to understand the semantics of the review and summary texts.
3) Syntactic features are not extracted for the review summary as there is very less scope due to the smaller content of review summaries.

Python programming language's 'NLTK' and 'TextBlob' libraries are used to perform stop words removal and lemmatization respectively. Moreover, 'TextBlob' library is also used to quantify the semantic features extracted from both the review and review summary texts. The semantic features used in this study are polarity and subjectivity of the review and summary texts. 'TextBlob' quantifies polarity in the range [-1,1] where '-1' being 'extremely negative', '0' being neutral, and '1' being 'extremely positive'. Subjectivity is quantified in the range [0,1] where '0' being 'highly

objective' and '1' being 'highly subjective'. The next step is to generate train and test sets with the ratio 7:3 with random shuffling. Features are then scaled using min-max scaling for all algorithms except extremely randomized trees.

Table 5-5 outlines the classification algorithms and their hyperparameters used to perform this analysis.

Table 5-5: Classifiers and their hyperparameters for review and summary centric features

Classifier	Hyperparameters
Multinomial naïve Bayes	Additive smoothing parameter = 1
Logistic regression	Regularization parameter C =0.01, penalty norm = L^1
Support vector machine	Kernel= linear, regularization parameter C=0.01, penalty norm = L^1
Extremely randomized trees	Number of trees = 10, features considered while splitting a node = $\sqrt{\text{Total number of features}}$

In this analysis, the feature selection for support vector machine and logistic regression is performed using 'SelectFromModel' transformer of 'sci-kit learn' library of Python. This transformer uses model's characteristics such as regularization parameter and penalty function to select significant features. To aid with the feature selection process, regularization parameter is set to be 0.01 in order to avoid overfitting by reducing the penalty of misclassification and L-1 norm is used as it helps in reducing the dimensionality of the features by producing sparse estimated coefficients (Pedregosa et al., 2011). The feature selection is performed by discarding the features which have their coefficients below a threshold value. The threshold values used in this analysis is 1e-5.

For extremely randomized trees classifier, the feature importance is calculated by combining 2 measures - the fraction of samples a feature splits across all the trees and its contribution to decrease in class impurity across all trees (Pedregosa et al., 2011).

Table 5-6 compares the performances of the classification algorithms and also shows the important features chosen by the classifiers.

Table 5-6: Performance evaluation of classifiers and feature selection

Classifier	Performance measure	Important features
Multinomial naïve Bayes	Accuracy: 53.88% AUC: 0.5373	No feature selection apart from text preprocessing
Logistic regression	Accuracy: 55.51% AUC: 0.5543	Review score rating, review word count, summary character count, review polarity, review subjectivity, summary polarity
Support vector machine	Accuracy: 55.81% AUC: 0.5574	Review score rating, review character count, average word length, summary word count, summary character count, review polarity, review subjectivity, summary polarity
Extremely randomized trees	**Accuracy: 70.70%** **AUC: 0.7069**	Review character count, average word length, review word count, review polarity, review subjectivity, summary character count

As it can be inferred from Table 5-6, extremely randomized trees classifier outperforms the conventional classifiers used for text classification. Talking about feature importance, the common important features for all algorithms are review polarity, review subjectivity and summary character count. Moreover, extremely randomized trees classifier also identifies review word count, review character count, and average word length of the review as important features. Even though SVM and logistic regression count review rating as an important feature, it is ranked last by extremely randomized trees[3].

5.3 Analysis of word embedding based features

This section discusses the analysis of the classification performed with the help of word-embedding based features. This analysis includes 2 variants of Word2vec algorithm namely, CBOW (Continuous bag of words) and skip-gram. Moreover, Word2vec is also combined with the TF-IDF vectorizer to check whether it performs more efficiently than the Word2vec or TF-IDF vectorizer alone. 'Gensim' library of the Python programming language is used for converting words into vectorized form.

According to Mikolov et al. (2013a), selection of training algorithm and hyperparameters is a problem specific decision. However, it was claimed that increasing vector dimensionality improves performance until a point at which improvement is diminishing (Mikolov et al., 2013b). Moreover, there is often some computational complexity associated with training the Word2vec algorithm. In order to update the output word vector, one needs to iterate over every word in the vocabulary for each training instance (Rong X., 2014). To deal with this problem, Mikolov et al. (2013) introduced the negative sampling technique to choose a limited number of samples during

[3] Even though extremely randomized trees classifier quantifies importance for every feature, only top 6 are shown here.

each iteration to update the output word vector. It has been proven by experiments that the number of these chosen samples can be 2 to 5 for large datasets and, 5 to 20 for small datasets. In their study of analogical reasoning, Mikolov et al. (2013a) found that the process of selecting 15 samples performed slightly better than the one with 5 samples. Moreover, the negative sampling technique as a whole was found to perform better than its counterpart – hierarchical softmax.

The selection of hyperparameters for this study is based on the factors discussed above. Table 5-7 provides details on the hyperparameters used to build the Word2vec model.

Table 5-7: Hyperparameters used for Word2vec model

Hyperparameter	Value
Dimensionality of word vector	300
Number of context words considered	5
Minimum word frequency required	5
Sample size used for negative sampling	15

The very first step of the analysis procedure involves performing text preprocessing, followed by generation of the train and test set with the split ratio 7:3. In order to train the model, the train set containing the review texts is converted into the list of lists format where each list includes tokens of an individual review text. The next step is to build the model using the hyperparameters shown in Table 5-7. As it can be inferred from Table 5-7, the word vectors will contain 300 elements after the model is trained. Moreover, 5 context words are considered in predicting the current word in CBOW variant of the model and, predicting the context words using the current word in the skip-gram variant. To deal with the computational complexity problem, words with less than 5 counts are discarded and only 15 words are considered while updating the weights of the output vector.

Two approaches are used in this study to come up with the vectors for individual review text. In the first approach, the word vectors of a review text are added and then their average is calculated to obtain a vector for the whole review text. The second approach multiplies the vector of a word to the corresponding TF-IDF value and then takes an average to determine the vector for the review text. This procedure is repeated for all the review texts of the training set and a matrix is obtained with rows denoting the number of reviews and columns denoting the elements of the review vector.

This matrix along with the response variable provides a basis for performing review helpfulness prediction by applying machine learning classification algorithms. The columns of this matrix are scaled using min-max scaling for all algorithms except for extremely randomized trees. Table 5-8 outlines the classification algorithms used and their hyperparameters.

Table 5-8: Classifiers and their hyperparameters for word embedding based features

Classifier	Hyperparameters
Multinomial naïve Bayes	Additive smoothing parameter = 1
Logistic regression	Regularization parameter C = 1, penalty norm = L^2
Support vector machine	Kernel= linear, regularization parameter C=1, penalty norm = L^2
Extremely randomized trees	Number of trees = 10, features considered while splitting a node = $\sqrt{\text{Total number of features}}$

Table 5-9 outlines the performance of the first approach. As it can be inferred from Table 5-9, extremely randomized trees algorithm outperforms the conventional algorithms used for the text classification purpose. The conventional algorithms perform slightly better than what would have been achieved with a random guess. However, there is not any significant difference as far as the performances of the features are considered. Word2vec with CBOW variant performs the same as the skip-gram variant.

Table 5-9: Performance evaluation of Word2vec approach

	Multinomial naïve Bayes	**Logistic regression**	**Support vector machine**	**Extremely randomized trees**
Word2vec with CBOW	Accuracy:51.46%	Accuracy:53.42%	Accuracy:53.46%	**Accuracy:69.73%**
	AUC: 0.5178	AUC: 0.5341	AUC: 0.5345	**AUC: 0.6974**
Word2vec with skip-gram	Accuracy:51.63%	Accuracy:53.47%	Accuracy:53.47%	**Accuracy:70.17%**
	AUC: 0.5208	AUC: 0.5347	AUC:0.5347	**AUC: 0.6932**

Now, let us discuss the performance of the second approach. Table 5-10 outlines the performance of the TF-IDF weighted Word2vec approach.

Table 5-10: Performance evaluation of TF-IDF weighted Word2vec approach

	Multinomial naïve Bayes	**Logistic regression**	**Support vector machine**	**Extremely randomized trees**
TF-IDF weighted Word2vec with CBOW	Accuracy:50.92%	Accuracy:53.45%	Accuracy:53.46%	**Accuracy:68.89%**
	AUC: 0.5211	AUC: 0.5344	AUC: 0.5346	**AUC: 0.6929**
TF-IDF weighted Word2vec with skip-gram	Accuracy:50.60%	Accuracy:53.47%	Accuracy:53.47%	**Accuracy:68.90%**
	AUC: 0.5131	AUC: 0.5347	AUC: 0.5347	**AUC: 0.6925**

As it can be inferred from Table 5-10, extremely randomized trees algorithm again outperforms the conventional algorithms. Moreover, there is not much difference in term of performance between CBOW and skip-gram variants. It should also be noted that combining word vectors with TF-IDF values does not help much in improving the overall prediction performance.

5.4 Comparison of features and classifiers used in the study

This section compares the performances of all the features and classifiers used throughout the study. This comparison provides a basis for determining the most feasible combination of feature set and classifier that can be used in building an automated text classification system for predicting the review helpfulness.

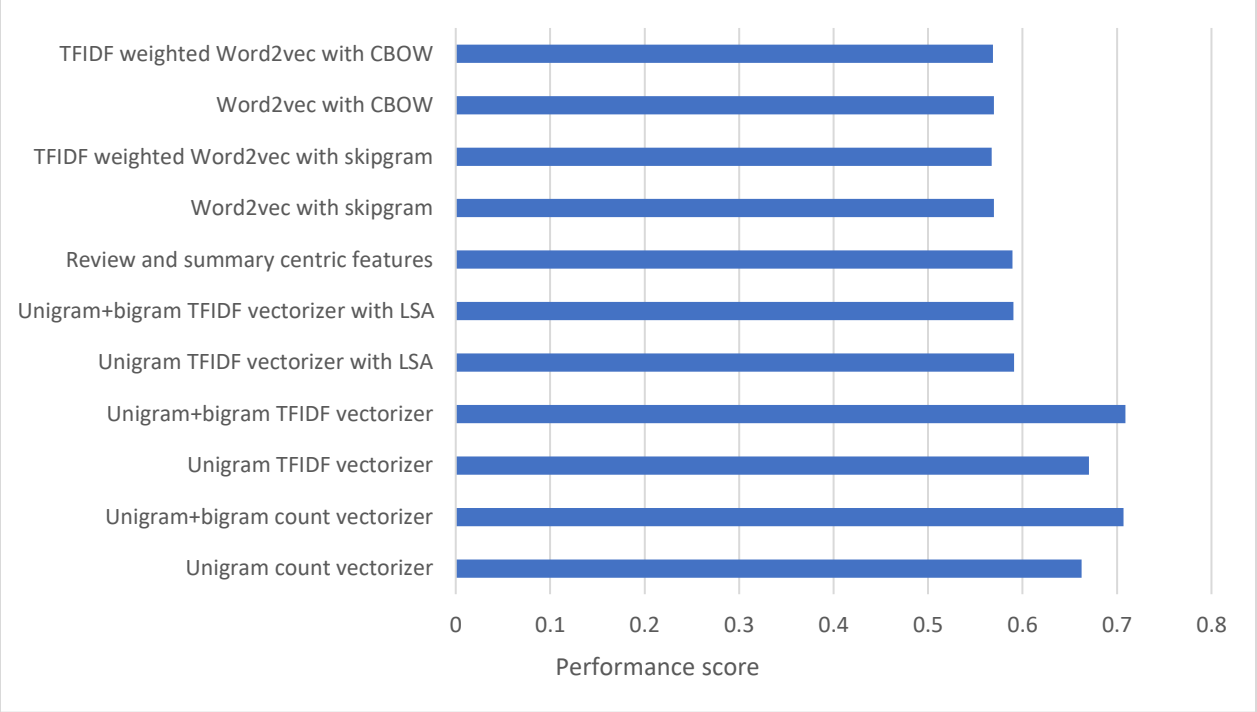
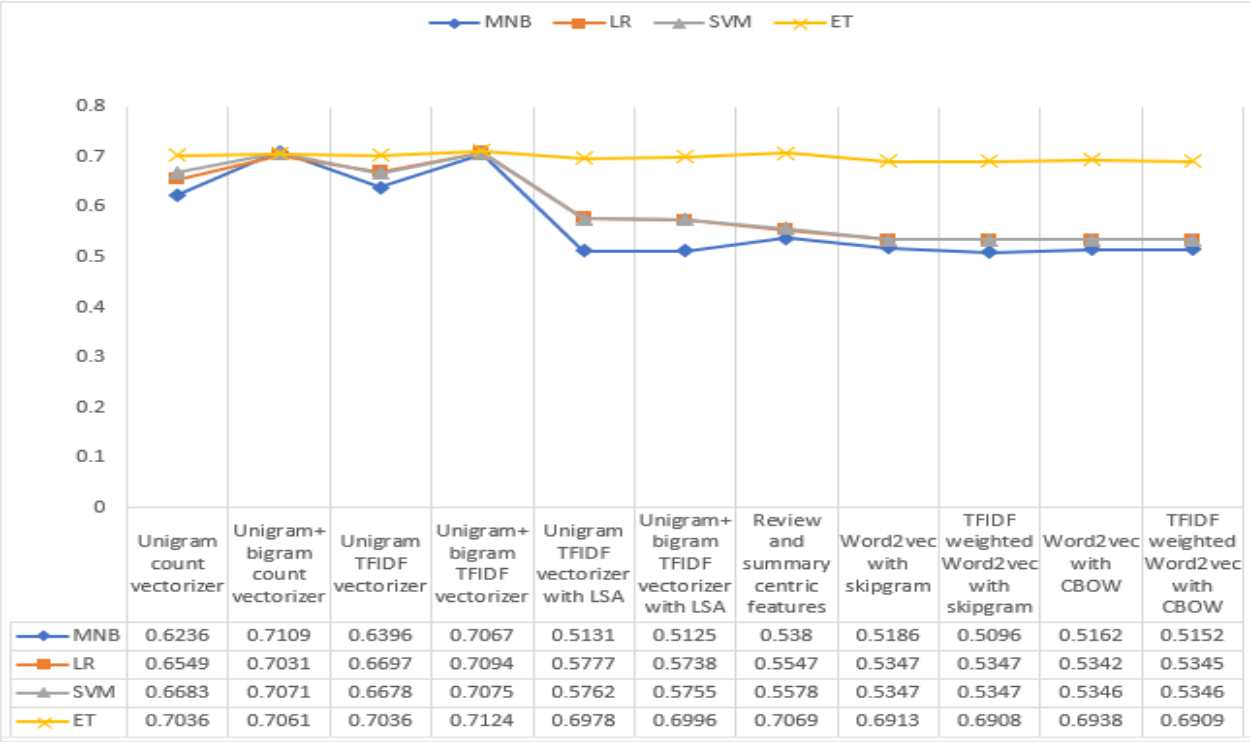


Figure 5-2: Performance comparison of features used in the study

Figure 5-2 shows the performance comparison of all the feature sets used throughout this study. The X-axis denotes the performance score of an algorithm which is calculated by taking the average of overall accuracy and AUC (area under the ROC curve) measures across all the 4 classifiers used in the study. The formula for calculating the score is shown below:

$$Score = \frac{\sum_{Algorithms} Overall\ accuracy + \sum_{Algorithms} AUC}{2}$$

As it can be inferred from Figure 5-2, the overall best performance is achieved by vectorized features which incorporate both unigrams and bigrams, followed by vectorized features with only unigrams. Other feature sets perform slightly better than what would have been obtained by random guessing with LSA based vectorizers and review and summary centric features performing marginally better than the word embedding based features. Now, let us discuss the performance comparison of the classifiers used in the study.



	Unigram count vectorizer	Unigram+ bigram count vectorizer	Unigram TFIDF vectorizer	Unigram+ bigram TFIDF vectorizer	Unigram TFIDF vectorizer with LSA	Unigram+ bigram TFIDF vectorizer with LSA	Review and summary centric features	Word2vec with skipgram	TFIDF weighted Word2vec with skipgram	Word2vec with CBOW	TFIDF weighted Word2vec with CBOW
MNB	0.6236	0.7109	0.6396	0.7067	0.5131	0.5125	0.538	0.5186	0.5096	0.5162	0.5152
LR	0.6549	0.7031	0.6697	0.7094	0.5777	0.5738	0.5547	0.5347	0.5347	0.5342	0.5345
SVM	0.6683	0.7071	0.6678	0.7075	0.5762	0.5755	0.5578	0.5347	0.5347	0.5346	0.5346
ET	0.7036	0.7061	0.7036	0.7124	0.6978	0.6996	0.7069	0.6913	0.6908	0.6938	0.6909

Figure 5-3: Performance comparison of classifiers used in the study

Figure 5-3 shows the performance comparison of all the classifiers used in the study. The performance score is calculated taking the average of overall accuracy and AUC measures. The formula is shown below:

$$Score = \frac{Overall\ accuracy + AUC}{2}$$

As it can be inferred from Figure 5-3, extremely randomized trees classifier performs consistently well irrespective of the feature used. In most of the cases, it outperforms the conventional classifiers used for text classification except in the case of vectorized features which have both unigrams and bigrams as features. This justifies the popularity of using conventional classifiers in high dimensional setup. The difference between the performances of extremely randomized trees and the conventional classifiers is also marginal for the vectorized features with only unigrams.

Chapter 6
Conclusions and Future work

In this study, the focus was on building an automated text classification system which can predict the helpfulness measure of an online review irrespective of the time of posting. The purpose of this was to provide both consumers and manufacturers a wide variety of reviews to choose from by including the most recent yet unvoted reviews in addition to higher voted old dated reviews. The first step was to conduct a literature survey in order to get familiar with the work that had been done in the domain of text classification including the work relevant to the review helpfulness measurement. It was found that the work done regarding the review helpfulness focused mostly on finding a correlation between review content-centric features and the helpfulness measure of the review. It was also noticed that the word embedding based features had been used in the domain of text classification other than that of review helpfulness. In this study, the comparison was made between the vectorized features, review and review summary centric features, and word embedding based features, which to the best of my knowledge, has never been attempted before. Moreover, the conventional classifiers used for text classification such as Support vector machine, Multinomial naïve Bayes, and Logistic regression were compared with Extremely randomized trees - a decision tree-based ensemble classifier. In order to perform the prediction task using the above discussed features and classifiers, the binary response variable was generated on the basis of helpfulness votes obtained by a review. It was followed by text preprocessing and finally, the models were trained and tested using hold-out validation along with accuracy and AUC scores.

The key findings of this study are discussed below:

- Extremely randomized trees classifier performed consistently well irrespective of the feature set used. In most of the instances, it outperformed the conventional classifiers used for text classification except in the case of vectorized features with unigrams and bigrams.
- Conventional classifiers performed better with sparse features such as count vectorizer and TF-IDF vectorizer with unigrams and bigrams compared to the features with lesser dimensionality. Their performances matched with that of Extremely randomized trees classifier which justifies the claims made by Joachims (1998) and Aggarwal et al. (2012) regarding high suitability of conventional classifiers such as SVM for sparse text classification problems.
- Among the features used, the overall best performance was obtained when the classifiers were implemented using the vectorized features with unigrams and bigrams, followed by vectorized features with only unigrams. Performing dimensionality reduction on vectorized features using LSA did not help much. Moreover, the results obtained using other features were slightly better than those obtained by random guessing with the summary and review centric features performing marginally better than word embedding based features. Also, there was no difference between the performances of CBOW and skip-gram variants of Word2vec model.
- Talking about the review and summary-centric features, the common significant features identified by all classifiers were review polarity, review subjectivity and summary character count. Even though the review score rating was identified as a significant feature by SVM and Logistic regression, it was ranked last in feature importance by more accurate Extremely randomized trees classifier. This complements the result obtained while

performing exploratory data analysis. The other significant features obtained by the Extremely randomized trees classifiers were review character count, review word count, and average word length of the review. This features along with review polarity and review subjectivity complement the results obtained in the studies conducted by Kim et al., (2006); Cao et al., (2011); and Salehan et al., (2016).

This research provides ample scope for potential future work and can be moved forward in multiple directions. First of all, the validation procedure can be made more robust. Even though there was enough data to train in the case of the hold-out validation method, the likelihood of overfitting can further be reduced by using k-fold cross-validation. The overall best performance was achieved by the vectorized features with unigrams and bigrams. It can be checked whether the performance improves when tri-grams are also added to the combination of unigrams and bigrams, especially in the case of conventional classifiers which are usually suitable for high dimensional sparse text classification problems. Moreover, the overall best performing classifier was Extremely randomized trees which is an ensemble of decision tree classifiers. It can be compared with other decision tree-based ensemble classifiers such as Random forest and Gradient tree boosting, and examine whether the performance can be enhanced. Word embedding based features performed marginally better than random guessing. Hence, there is a need to implement a more sophisticated technique which can convert review text into a vector form. The Paragraph Vector algorithm proposed by Le et al., (2014) directly provides a document vector without manipulating the document's word vectors. This algorithm can be considered for future study as it has been shown to perform better than the vectorized features in text classification problems.

Appendix

Python code

Analysis of vectorized features

```
import pandas as pd
import numpy as np
reviews=pd.read_csv("Reviews.csv")
#Removing reviews which have 'helpfulness denominator' value 0
 rs=reviews[reviews.HelpfulnessDenominator!=0]
#Generating binary response variable
rs["Review_Helpfulness"]= np.where(rs['HelpfulnessNumerator']>1,1,0)
#Converting uppercase into lowercase
rs['Text'] = rs['Text'].apply(lambda x: " ".join(x.lower() for x in
x.split()))
#Removing punctuation marks
rs['Text'] = rs['Text'].str.replace('[^\w\s]','')
#Removing stopwords
import nltk
from nltk.corpus import stopwords
stop = stopwords.words('english')
stop += ['.','/','<','>','``','"','-','--']
rs['Text'] = rs['Text'].apply(lambda x: " ".join(x for x in x.split() if x
not in stop))
#Performing lemmatization
import textblob
from textblob import TextBlob
from textblob import Word
rs['Text'] = rs['Text'].apply(lambda x: " ".join([Word(word).lemmatize()
for word in x.split()]))
#Generating train and test sets
from sklearn.model_selection import train_test_split
train_x, valid_x, train_y, valid_y = train_test_split(rs['Text'],
rs['Review_Helpfulness'],test_size=0.3,shuffle=True,random_state=42)

#Computations for uni-gram count vectorizer
#Support vector machine
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC
clfsvc=make_pipeline(CountVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,1)),LinearSVC(C=1,dual=False,penalty='l1',random_state=42))
clfsvc.fit(train_x,train_y)
predicted=clfsvc.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Multinomial naive Bayes
```

```
from sklearn.naive_bayes import MultinomialNB
clfmnb=make_pipeline(CountVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,1)),MultinomialNB())
clfmnb.fit(train_x,train_y)
predicted=clfmnb.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Logistic regression
from sklearn.linear_model import LogisticRegression
clflr=make_pipeline(CountVectorizer(stop_words='english',analyzer='word',n
gram_range=(1,1)),LogisticRegression(C=1,dual=False,penalty='l1',random_st
ate=42))
clflr.fit(train_x,train_y)
predicted=clflr.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
# Extremely randomized trees
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline
clfetree=make_pipeline(CountVectorizer(stop_words='english',analyzer='word
',ngram_range=(1,1)),ExtraTreesClassifier(n_estimators=10,random_state=0))
clfetree.fit(train_x,train_y)
predicted=clfetree.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))

#Computation for uni-gram+bi-gram count vectorizer

#Support vector machine
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC
clfsvc=make_pipeline(CountVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,2)),LinearSVC(C=1,dual=False,penalty='l1',random_state=42))
clfsvc.fit(train_x,train_y)
predicted=clfsvc.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Multinomial naive Bayes
from sklearn.naive_bayes import MultinomialNB
clfmnb=make_pipeline(CountVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,2)),MultinomialNB())
clfmnb.fit(train_x,train_y)
predicted=clfmnb.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Logistic Regression
from sklearn.linear_model import LogisticRegression
```

```
clflr=make_pipeline(CountVectorizer(stop_words='english',analyzer='word',n
gram_range=(1,2)),LogisticRegression(C=1,dual=False,penalty='l1',random_st
ate=42))
clflr.fit(train_x,train_y)
predicted=clflr.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
# Extremely randomized trees
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import make_pipeline
clfetree=make_pipeline(CountVectorizer(stop_words='english',analyzer='word
',ngram_range=(1,2)),ExtraTreesClassifier(n_estimators=10,random_state=42)
)
clfetree.fit(train_x,train_y)
predicted=clfetree.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))

#Computation for unigram TF-IDF vectorizer
#Support vector machine
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC
clfsvc=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,1),norm=None),LinearSVC(C=1,dual=False,penalty='l1',random_
state=42))
clfsvc.fit(train_x,train_y)
predicted=clfsvc.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Multinomial naive Bayes
from sklearn.naive_bayes import MultinomialNB
clfmnb=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,1),norm=None),MultinomialNB())
clfmnb.fit(train_x,train_y)
predicted=clfmnb.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Logistic regression
from sklearn.linear_model import LogisticRegression
clflr=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word',n
gram_range=(1,1),norm=None),LogisticRegression(C=1,dual=False,penalty='l1'
,random_state=42))
clflr.fit(train_x,train_y)
predicted=clflr.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
# Extremely randomized trees
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
clfetree_tfidf=make_pipeline(TfidfVectorizer(stop_words='english',analyzer
='word',ngram_range=(1,1),norm=None),ExtraTreesClassifier(n_estimators=10,
random_state=0))
clfetree_tfidf.fit(train_x,train_y)
predicted=clfetree_tfidf.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))

#Computation for uni-gram+bi-gram TF-IDF vectorizer
#Support vector machine
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC
clfsvc=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,2),norm=None),LinearSVC(C=1,dual=False,penalty='l1',random_
state=42))
clfsvc.fit(train_x,train_y)
predicted=clfsvc.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Multinomial naive Bayes
from sklearn.naive_bayes import MultinomialNB
clfmnb=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word',
ngram_range=(1,2),norm=None),MultinomialNB())
clfmnb.fit(train_x,train_y)
predicted=clfmnb.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
#Logistic regression
from sklearn.linear_model import LogisticRegression
clflr=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word',n
gram_range=(1,2),norm=None),LogisticRegression(C=1,dual=False,penalty='l1'
))
clflr.fit(train_x,train_y)
predicted=clflr.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
# Extremely randomized trees
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
clfetree=make_pipeline(TfidfVectorizer(stop_words='english',analyzer='word
',ngram_range=(1,2)),ExtraTreesClassifier(n_estimators=10,random_state=42)
)
clfetree.fit(train_x,train_y)
predicted=clfetree.predict(valid_x)
print(accuracy_score(valid_y,predicted))
print(roc_auc_score(valid_y,predicted))
```

Analysis of vectorized features with LSA

```
#Computation for unigram TF-IDF vectorizer with LSA


#Performing dimensionality reduction using LSA
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect=TfidfVectorizer(stop_words='english',analyzer='word',ngram_rang
e=(1,1),norm=None)
train_tfidf = tfidf_vect.fit_transform(train_x)
print(train_tfidf.shape)
svd = TruncatedSVD(100,random_state=42)
lsa=make_pipeline(svd,MinMaxScaler(copy=False))
train_lsa = lsa.fit_transform(train_tfidf)
test_tfidf=tfidf_vect.transform(valid_x)
test_lsa = lsa.transform(test_tfidf)
#Support vector machine
from sklearn.svm import LinearSVC
clfsvc=LinearSVC(C=1,dual=False,penalty='l2',random_state=42).fit(train_ls
a, train_y)
predicted = clfsvc.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
#Logistic regression
from sklearn.linear_model import LogisticRegression
clflr=LogisticRegression(C=1,dual=False,penalty='l2',random_state=42).fit(
train_lsa, train_y)
predicted = clflr.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
#Multinomial naive Bayes
from sklearn.naive_bayes import MultinomialNB
clfmnb=MultinomialNB().fit(train_lsa, train_y)
predicted = clfmnb.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
#Extremely randomized tress
from sklearn.ensemble import ExtraTreesClassifier
clfetree=ExtraTreesClassifier(n_estimators=10,random_state=42).fit(train_l
sa,train_y)
predicted=clfetree.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))

# Computation for uni-gram+bi-gram TFIDF vectorizer with LSA
#Perfroming dimensionality reduction using LSA
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import roc_auc_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect=TfidfVectorizer(stop_words='english',analyzer='word',ngram_rang
e=(1,2),norm=None,max_features=1500000)
train_tfidf = tfidf_vect.fit_transform(train_x)
print(train_tfidf.shape)
svd = TruncatedSVD(100,random_state=42)
lsa=make_pipeline(svd,MinMaxScaler(copy=False))
train_lsa = lsa.fit_transform(train_tfidf)
test_tfidf=tfidf_vect.transform(valid_x)
test_lsa = lsa.transform(test_tfidf)
#Support vector machine
from sklearn.svm import LinearSVC
clfsvc=LinearSVC(C=1,dual=False,penalty='l2',random_state=42).fit(train_ls
a, train_y)
predicted = clfsvc.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
#Logistic regression
from sklearn.linear_model import LogisticRegression
clflr=LogisticRegression(C=1,dual=False,penalty='l2',random_state=42).fit(
train_lsa, train_y)
predicted = clflr.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
#Multinomial naive Bayes
from sklearn.naive_bayes import MultinomialNB
clfmnb=MultinomialNB().fit(train_lsa, train_y)
predicted = clfmnb.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
#Extremely randomized tress
from sklearn.ensemble import ExtraTreesClassifier
clfetree=ExtraTreesClassifier(n_estimators=10,random_state=42).fit(train_l
sa,train_y)
predicted=clfetree.predict(test_lsa)
print(accuracy_score(valid_y, predicted))
print(roc_auc_score(valid_y, predicted))
```

Analysis of review and summary centric features

```
import pandas as pd
import numpy as np
reviews=pd.read_csv("Reviews.csv")
#Removing reviews which have 'helpfulness denominator' value 0
rs=reviews[reviews.HelpfulnessDenominator!=0]
#Generating binary response variable
rs["Review_Helpfulness"]=np.where(rs['HelpfulnessNumerator']>1,1,0)
#Checking missing values for each attribute
rs.isna().sum()
#Removing null values of 'Summary' attribute
rsnew=rs.dropna()
```

```
#Extracting structural and syntactic features of review and summary text
#1. Word count of the review text
rsnew['review_word_count'] = rsnew['Text'].apply(lambda x: len(x.split()))
#2. Character count of the review text
rsnew['review_char_count'] = rsnew['Text'].apply(len)
#3. Average length of the words used in the review text
rsnew['avg_word_length']=rsnew['review_char_count']/(rsnew['review_word_co
unt']+1)
#4. Total number of punctuation marks in the review text
from string import punctuation
rsnew['review_punctuation_count']=rsnew['Text'].apply(lambda
x:len("".join(c for c in x if c in punctuation)))
#5. Total number of proper case words in the review text
rsnew['review_propercaseword_count']   =   rsnew['Text'].apply(lambda   x:
len([word for word in x.split() if word.istitle()]))
#6. Total number of upper case words in the review text
rsnew['review_uppercaseword_count']   =   rsnew['Text'].apply(lambda   x:
len([word for word in x.split() if word.isupper()]))
#7. Word count of the summary text
rsnew['summary_word_count']     =     rsnew['Summary'].apply(lambda     x:
len(x.split()))
#8. Character count of the summary text
rsnew['summary_char_count'] = rsnew['Summary'].apply(len)
#9. Meta-data (Review rating)
rsnew['Score']

#Extracting Semantic Features
#1. Polarity and subjectivity of the review text
# Text pre-processing
rsnew['Text'] = rsnew['Text'].apply(lambda x: " ".join(x.lower() for x in
x.split()))
rsnew['Text'] = rsnew['Text'].str.replace('[^\w\s]','') import nltk
from nltk.corpus import stopwords
stop = stopwords.words('english')
stop += ['.','/','<','>','``','"','-','--']
rsnew['Text'] = rsnew['Text'].apply(lambda x: " ".join(x for x in x.split()
if x not in stop))
import textblob
from textblob import TextBlob
from textblob import Word
rsnew['Text']=rsnew['Text'].apply(lambda
x:" ".join([Word(word).lemmatize() for word in x.split()]))
rsnew['polarity'] = rsnew['Text'].apply(lambda x: TextBlob(x).sentiment[0])
rsnew['subjectivity']= rsnew['Text'].apply(
      lambda x: TextBlob(x).sentiment[1])
#Polarity and subjectivity of the summary
rsnew['summary_polarity']=rsnew['Summary'].apply(
      lambda x: TextBlob(x).sentiment[0])
rsnew['summary_subjectivity']=rsnew['Summary'].apply(
      lambda x: TextBlob(x).sentiment[1])
#Creating dataframe with the features extracted
features_df=rsnew[['Score','review_word_count','review_char_count','avg_wo
rd_length','review_punctuation_count','review_propercaseword_count','revie
```

```
w_uppercaseword_count','summary_word_count','summary_char_count','polarity
','subjectivity','summary_polarity','summary_subjectivity']]

#Support vector machine(Feature selection step)
X=features_df y=rsnew['Review_Helpfulness']
from sklearn.model_selection import train_test_split
train_x,valid_x,train_y,valid_y=train_test_split(X,y,test_size=0.3,shuffle
=True,random_state=42)
from sklearn.preprocessing import MinMaxScaler
min_max_scaler= MinMaxScaler(copy=False)
train_scaled= min_max_scaler.fit_transform(train_x)
valid_scaled = min_max_scaler.transform(valid_x)
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import LinearSVC
clfsvc=LinearSVC(C=0.01,penalty='l1',dual=False,random_state=0).fit(train_
scaled,train_y)
model=SelectFromModel(clfsvc,prefit=True)
X_new=model.transform(train_scaled)
selected_feature_names=X.columns[model.get_support()]
print(X_new.shape)
print(model.get_support())
print(selected_feature_names)

#Support vector machine(Building a model with selected features)
from sklearn.metrics import roc_auc_score
SVM_new_features=features_df[['Score','review_char_count','avg_word_length
','summary_word_count','summary_char_count','polarity','subjectivity','sum
mary_polarity']]
train_x,valid_x,train_y,valid_y=train_test_split(SVM_new_features,y,test_s
ize=0.3,shuffle=True,random_state=42)
min_max_scaler= MinMaxScaler(copy=False)
train_scaled= min_max_scaler.fit_transform(train_x)
valid_scaled = min_max_scaler.transform(valid_x)
from sklearn.svm import LinearSVC
clfsvc=LinearSVC(C=0.01,penalty='l1',dual=False,random_state=42).fit(train
_scaled, train_y)
model=SelectFromModel(clfsvc,prefit=True)
X_new=model.transform(train_scaled)
selected_feature_names=SVM_new_features.columns[model.get_support()]
predicted = clfsvc.predict(valid_scaled)
accuracy=np.mean(predicted==valid_y)
print(accuracy)
print(roc_auc_score(valid_y,predicted))
print(clfsvc.coef_)
print(model.get_support())
print(selected_feature_names)

#Logisitc regression(Feature selection step)
X=features_df y=rsnew['Review_Helpfulness']
from sklearn.model_selection import train_test_split
train_x,valid_x,train_y,valid_y=train_test_split(X,y,test_size=0.3,shuffle
=True,random_state=42)
from sklearn.preprocessing import MinMaxScaler
min_max_scaler= MinMaxScaler(copy=False)
```

```
train_scaled= min_max_scaler.fit_transform(train_x)
valid_scaled = min_max_scaler.transform(valid_x)
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
clflr=LogisticRegression(C=0.01,penalty='l1',dual=False,random_state=0).fi
t(train_scaled,train_y)
model=SelectFromModel(clflr,prefit=True)
X_new=model.transform(train_scaled)
selected_feature_names=X.columns[model.get_support()]
print(X_new.shape)
print(model.get_support())
print(selected_feature_names)

#Logistic regression(Building a model with selected features)

LR_new_features=features_df[['Score','review_word_count','summary_char_cou
nt','polarity','subjectivity','summary_polarity']]
train_x,           valid_x,           train_y,           valid_y           =
train_test_split(LR_new_features,y,test_size=0.3,shuffle=True,random_state
=42)
min_max_scaler= MinMaxScaler(copy=False)
train_scaled= min_max_scaler.fit_transform(train_x)
valid_scaled = min_max_scaler.transform(valid_x)
from sklearn.linear_model import LogisticRegression
clflr=LogisticRegression(C=0.01,penalty='l1',dual=False,random_state=42).f
it(train_scaled, train_y)
model=SelectFromModel(clflr,prefit=True)
X_new=model.transform(train_scaled)
selected_feature_names=LR_new_features.columns[model.get_support()]
predicted = clflr.predict(valid_scaled)
accuracy=np.mean(predicted==valid_y)
print(accuracy)
print(roc_auc_score(valid_y,predicted))
print(clflr.coef_)
print(model.get_support())
print(selected_feature_names)

#Extremely randomized trees(Building a model along with feature selection)
from sklearn.ensemble import ExtraTreesClassifier
X=features_df y=rsnew['Review_Helpfulness']
from sklearn.model_selection import train_test_split
train_x,valid_x,train_y,valid_y=train_test_split(X,y,test_size=0.3,shuffle
=True,random_state=42)
clfet=ExtraTreesClassifier(n_estimators=10,random_state=42).fit(train_x,tr
ain_y)
predicted=clfet.predict(valid_x)
accuracy=np.mean(predicted==valid_y)
print(accuracy)
print(roc_auc_score(valid_y,predicted))
print(clfet.feature_importances_)
#Feature importance ranking
featimp=pd.Series(clfet.feature_importances_,index=X.columns).sort_values(
ascending=False)
print(featimp)
```

```
#Multinomial naive Bayes
from sklearn.naive_bayes import MultinomialNB
train_x,valid_x,train_y,valid_y=train_test_split(X,y,test_size=0.3,shuffle
=True,random_state=42) min_max_scaler= MinMaxScaler(copy=False)
train_scaled= min_max_scaler.fit_transform(train_x)
valid_scaled = min_max_scaler.transform(valid_x)
clfmnb=MultinomialNB().fit(train_scaled,train_y)
predicted=clfmnb.predict(valid_scaled)
accuracy=np.mean(predicted==valid_y)
print(accuracy)
print(roc_auc_score(valid_y,predicted))
```

Analysis using word embedding based features[4]

```
import pandas as pd
reviews=pd.read_csv("Reviews.csv")
#Removing reviews which have 'helpfulness denominator' value 0
rs=reviews[reviews.HelpfulnessDenominator!=0]
#Generating binary response variable
rs["Review_Helpfulness"]=np.where(rs['HelpfulnessNumerator']>1,1,0)
#Converting uppercase into lowercase
rs['Text'] = rs['Text'].apply(lambda x: " ".join(x.lower() for x in
x.split()))
#Removing punctuations
rs['Text'] = rs['Text'].str.replace('[^\w\s]','')
#Removing stopwords
import nltk
from nltk.corpus import stopwords
stop = stopwords.words('english')
stop += ['.','/','<','>','``','"','-','--']
rs['Text'] = rs['Text'].apply(lambda x: " ".join(x for x in x.split() if x
not in stop))
#Performing lemmatization
import textblob
from textblob import TextBlob
from textblob import Word
rs['Text'] = rs['Text'].apply(lambda x: " ".join([Word(word).lemmatize()
for word in x.split()]))
#Generating train and test sets
from sklearn.model_selection import train_test_split
train_x, valid_x, train_y, valid_y = train_test_split(rs['Text'],
rs['Review_Helpfulness'],test_size=0.3,shuffle=True,random_state=42)
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from gensim.models.word2vec import Word2Vec
from collections import Counter, defaultdict
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
```

[4] This code is inspired from the work done by Drozd (2016).

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
#Converting review texts into a list of lists form sentences=[] for i in
train_x: sentence=TextBlob(i).words sentences.append(sentence)

#Training Word2vec model with skip-gram variant
model = Word2Vec(sentences, size=300, window=5, min_count=5,
workers=2,sg=1,negative=15)
w2v = {w: vec for w, vec in zip(model.wv.index2word, model.wv.syn0)}

#Generating review text vectors using word vectors(Word2vec approach)

class MeanEmbeddingVectorizer(object):
      def __init__(self, word2vec):
            self.word2vec = word2vec
            if len(word2vec) > 0:
            self.dim = len(word2vec[next(iter(self.word2vec))])
            else:
            self.dim = 0

      def fit(self, X, y):
            return self

      def transform(self, X):

            return np.array([
            np.mean([self.word2vec[w] for w in words if w in self.word2vec]
            or [np.zeros(self.dim)], axis=0)
            for words in X
            ])

#Extremely randomized trees
etree = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("extra trees", ExtraTreesClassifier(n_estimators=10,random_state=42))])
score_acc_et = accuracy_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_et)
score_auc_et = roc_auc_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_et)

#Multinomial naive Bayes
mnb = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("multinomial nb",
MultinomialNB())])
score_acc_nb = accuracy_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_nb)
score_auc_nb = roc_auc_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_nb)

#Support vector machine
```

```
svc = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("supportvec",
LinearSVC(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_svc=accuracy_score(svc.fit(train_x,train_y).predict(valid_x),val
id_y)
print(score_acc_svc)
score_auc_svc=roc_auc_score(svc.fit(train_x,train_y).predict(valid_x),vali
d_y)
print(score_auc_svc)

#Logistic regression
lr = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("LogisticReg",
LogisticRegression(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_lr=accuracy_score(lr.fit(train_x,train_y).predict(valid_x),valid
_y)
print(score_acc_lr)
score_auc_lr=roc_auc_score(lr.fit(train_x,train_y).predict(valid_x),valid_
y)
print(score_auc_lr)

#Generating review text vector using word vectors(TFIDF weighted Word2vec
approach)
from sklearn.feature_extraction.text import TfidfVectorizer

class TfidfEmbeddingVectorizer(object):
      def __init__(self, word2vec):
            self.word2vec = word2vec
            self.word2weight = None
            if len(word2vec) > 0:
            self.dim = len(word2vec[next(iter(self.word2vec))])
            else:
            self.dim = 0

      def fit(self, X, y):
            tfidf = TfidfVectorizer(analyzer=lambda x: x)
            tfidf.fit(X)
            # if a word was never seen - it must be at least as infrequent
            # as any of the known words - so the default idf is the max of
            known idf's max_idf = max(tfidf.idf_)
            self.word2weight = defaultdict(
            lambda: max_idf,
            [(w, tfidf.idf_[i]) for w, i in tfidf.vocabulary_.items()])
            return self

     def transform(self, X):
            return np.array([
            np.mean([self.word2vec[w] * self.word2weight[w]
            for w in words if w in self.word2vec] or
            [np.zeros(self.dim)], axis=0)
            for words in X
            ])

#Extremely randomized trees
```

```
etree = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("extra trees", ExtraTreesClassifier(n_estimators=10,random_state=42))])
score_acc_et = accuracy_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_et)
score_auc_et = roc_auc_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_et)

#Multinomial naive Bayes
mnb = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("multinomial nb",
MultinomialNB())])
score_acc_nb = accuracy_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_nb)
score_auc_nb = roc_auc_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_nb)

#Support vector machine
svc = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("supportvec",
LinearSVC(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_svc=accuracy_score(svc.fit(train_x,train_y).predict(valid_x),val
id_y)
print(score_acc_svc)
score_auc_svc =
roc_auc_score(svc.fit(train_x,train_y).predict(valid_x),valid_y)
print(score_auc_svc)

#Logistic regression
lr = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("LogisticReg",
LogisticRegression(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_lr=accuracy_score(lr.fit(train_x,train_y).predict(valid_x),valid
_y)
print(score_acc_lr)
score_auc_lr=roc_auc_score(lr.fit(train_x,train_y).predict(valid_x),valid_
y)
print(score_auc_lr)

#Training Word2vec model with CBOW variant
model = Word2Vec(sentences, size=300, window=5, min_count=5,
workers=2,sg=0,negative=15)
w2v = {w: vec for w, vec in zip(model.wv.index2word, model.wv.syn0)}

#Generating review text vectors using word vectors(Word2vec approach)

class MeanEmbeddingVectorizer(object):
    def __init__(self, word2vec):
        self.word2vec = word2vec
        if len(word2vec) > 0:
        self.dim = len(word2vec[next(iter(self.word2vec))])
```

```
            else:
            self.dim = 0

      def fit(self, X, y):
            return self

      def transform(self, X):

            return np.array([
            np.mean([self.word2vec[w] for w in words if w in self.word2vec]
            or [np.zeros(self.dim)], axis=0)
            for words in X
            ])

#Extremely randomized trees
etree = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("extra trees", ExtraTreesClassifier(n_estimators=10,random_state=42))])
score_acc_et = accuracy_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_et)
score_auc_et = roc_auc_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_et)

#Multinomial naive Bayes
mnb = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("multinomial nb",
MultinomialNB())])
score_acc_nb = accuracy_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_nb)
score_auc_nb = roc_auc_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_nb)

#Support vector machine
svc = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("supportvec",
LinearSVC(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_svc=accuracy_score(svc.fit(train_x,train_y).predict(valid_x),val
id_y)
print(score_acc_svc)
score_auc_svc=roc_auc_score(svc.fit(train_x,train_y).predict(valid_x),vali
d_y)
print(score_auc_svc)

#Logistic regression
lr = Pipeline([("word2vec vectorizer", MeanEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("LogisticReg",
LogisticRegression(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_lr=accuracy_score(lr.fit(train_x,train_y).predict(valid_x),valid
_y)
print(score_acc_lr)
```

```
score_auc_lr=roc_auc_score(lr.fit(train_x,train_y).predict(valid_x),valid_
y)
print(score_auc_lr)

#Generating review text vector using word vectors(TFIDF weighted Word2vec
approach)
from sklearn.feature_extraction.text import TfidfVectorizer

class TfidfEmbeddingVectorizer(object):
      def __init__(self, word2vec):
            self.word2vec = word2vec
            self.word2weight = None
            if len(word2vec) > 0:
            self.dim = len(word2vec[next(iter(self.word2vec))])
            else:
            self.dim = 0

      def fit(self, X, y):
            tfidf = TfidfVectorizer(analyzer=lambda x: x)
            tfidf.fit(X)
            # if a word was never seen - it must be at least as infrequent
            # as any of the known words - so the default idf is the max of
            known idf's max_idf = max(tfidf.idf_)
            self.word2weight = defaultdict(
            lambda: max_idf,
            [(w, tfidf.idf_[i]) for w, i in tfidf.vocabulary_.items()])
            return self

     def transform(self, X):
            return np.array([
            np.mean([self.word2vec[w] * self.word2weight[w]
            for w in words if w in self.word2vec] or
            [np.zeros(self.dim)], axis=0)
            for words in X
            ])

#Extremely randomized trees
etree = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("extra trees", ExtraTreesClassifier(n_estimators=10,random_state=42))])
score_acc_et = accuracy_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_et)
score_auc_et = roc_auc_score(etree.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_et)

#Multinomial naive Bayes
mnb = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)),       ("multinomial       nb",
MultinomialNB())])
score_acc_nb = accuracy_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_acc_nb)
```

```
score_auc_nb = roc_auc_score(mnb.fit(train_x, train_y).predict(valid_x),
valid_y)
print(score_auc_nb)

#Support vector machine
svc = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("supportvec",
LinearSVC(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_svc=accuracy_score(svc.fit(train_x,train_y).predict(valid_x),val
id_y)
print(score_acc_svc)
score_auc_svc =
roc_auc_score(svc.fit(train_x,train_y).predict(valid_x),valid_y)
print(score_auc_svc)

#Logistic regression
lr = Pipeline([("word2vec vectorizer", TfidfEmbeddingVectorizer(w2v)),
("Min_max_Scaler",MinMaxScaler(copy=False)), ("LogisticReg",
LogisticRegression(C=1,penalty='l2',dual=False,random_state=42))])
score_acc_lr=accuracy_score(lr.fit(train_x,train_y).predict(valid_x),valid
_y)
print(score_acc_lr)
score_auc_lr=roc_auc_score(lr.fit(train_x,train_y).predict(valid_x),valid_
y)
print(score_auc_lr)
```

References

Aggarwal, C.C. and Zhai, C., 2012. A survey of text classification algorithms. In *Mining text data* (pp. 163-222). Springer, Boston, MA.

Cao, Q., Duan, W. and Gan, Q., 2011. Exploring determinants of voting for the "helpfulness" of online user reviews: A text mining approach. *Decision Support Systems*, *50*(2), pp.511-521.

Crawford, M., Khoshgoftaar, T.M., Prusa, J.D., Richter, A.N. and Al Najada, H., 2015. Survey of review spam detection using machine learning techniques. *Journal of Big Data*, *2*(1), p.23.

Drozd, N., 2016. Text Classification with Word2Vec – DS lore. Retrieved from http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/

Geurts, P., Ernst, D. and Wehenkel, L., 2006. Extremely randomized trees. *Machine learning*, *63*(1), pp.3-42.

Ghose, A. and Ipeirotis, P.G., 2007, August. Designing novel review ranking systems: predicting the usefulness and impact of reviews. In *Proceedings of the ninth international conference on Electronic commerce* (pp. 303-310). ACM.

Hotho, A., Nürnberger, A. and Paaß, G., 2005, May. A brief survey of text mining. In *Ldv Forum* (Vol. 20, No. 1, pp. 19-62).

Hsu, C.W., Chang, C.C. and Lin, C.J., 2003. A practical guide to support vector classification.

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An introduction to statistical learning* (Vol. 112, p. 18). New York: springer.

Joachims, T., 1998, April. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (pp. 137-142). Springer, Berlin, Heidelberg.

Kim, S.M., Pantel, P., Chklovski, T. and Pennacchiotti, M., 2006, July. Automatically assessing review helpfulness. In *Proceedings of the 2006 Conference on empirical methods in natural language processing* (pp. 423-430). Association for Computational Linguistics.

Landauer, T.K., Foltz, P.W. and Laham, D., 1998. An introduction to latent semantic analysis. *Discourse processes*, *25*(2-3), pp.259-284.

Le, Q. and Mikolov, T., 2014, January. Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188-1196).

Lilleberg, J., Zhu, Y. and Zhang, Y., 2015, July. Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)* (pp. 136-140). IEEE.

Liu, B. and Zhang, L., 2012. A survey of opinion mining and sentiment analysis. In *Mining text data* (pp. 415-463). Springer, Boston, MA.

Liu, Y., Huang, X., An, A. and Yu, X., 2008, December. Modeling and predicting the helpfulness of online reviews. In *2008 Eighth IEEE international conference on data mining* (pp. 443-452). IEEE.

Manning, C.D., Raghavan P., & Schütze, H., 2008. Introduction to information retrieval. Cambridge University Press.

McAuley, J.J. and Leskovec, J., 2013, May. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web* (pp. 897-908). ACM.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013a. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013b. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), pp.2825-2830.

Rehurek, R. and Sojka, P., 2010. Software framework for topic modelling with large corpora. *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. pp. 45-50. Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

Rong, X., 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Salehan, M. and Kim, D.J., 2016. Predicting the performance of online consumer reviews: A sentiment mining approach to big data analytics. *Decision Support Systems*, *81*, pp.30-40.

Shojaee, S., Murad, M.A.A., Azman, A.B., Sharef, N.M. and Nadali, S., 2013, December. Detecting deceptive reviews using lexical and syntactic features. In *2013 13th International Conference on Intellient Systems Design and Applications* (pp. 53-58). IEEE.

Sokolova, M. and Lapalme, G., 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), pp.427-437.

Thomo, A., 2009. Latent semantic analysis (Tutorial). *Victoria, Canda*, pp.1-7.

Vinodhini, G. and Chandrasekaran, R.M., 2012. Sentiment analysis and opinion mining: a survey. *International Journal*, *2*(6), pp.282-292.