Automatic Labelling of Customer Reviews

By

Salomey Afua Addo (salomey@aims.edu.gh)
African Institute for Mathematical Sciences (AIMS) - Ghana
Supervised by: Professor Axel Ngonga
Paderborn University, Germany

June 2018

AN ESSAY PRESENTED TO AIMS-GHANA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE IN MATHEMATICAL SCIENCES

DECLARATION

This work was carried out at AIMS-Ghana in partial fulfilment of the requirements for a Master of Science Degree.

I hereby declare that except where due acknowledgement is made, this work has never been presented wholly or in part for the award of a degree at AIMS-Ghana or any other University.

Scan your signature

Student: Salomey Afua Addo

Scan your signature

Supervisor: Prof. Axel Ngonga

ACKNOWLEDGEMENTS

I thank God almighty for giving me life and wisdom to pursue mathematics. Much appreciation goes to Prof. Axel Ngonga, my supervisor, Dr. Prince Koree Osei, my mentor and AIMS-Ghana president and Ms. Abla Azalekor, my tutor.

DEDICATION

I dedicate this work to all machine learning lovers. We hope to see Artificial Intelligence gain root in Africa.

Abstract

The growth of Information, Communication and Technology (ICT) coupled with the emergence of Internet as a medium of communication has led to the proliferation of Electronic Commerce. Customers are given the opportunity to express their shopping experiences in the form of reviews or star ratings after shopping online. These feedback from customers might help improve products to suit market demands. However, mining customer feedback is an arduous task because there exists a large amount of them and they are unstructured in nature (i.e., natural language). Therefore, in this work we employed Natural Language Processing (NLP) tools such as word embeddings and unsupervised learning techniques such as clustering algorithms to build a pipeline that helps obtain summary words (labels) for customer reviews. Labels obtained for customer reviews would help online market owners like Amazon to uncover the products that customers like or dislike, producers would improve their product quality and customers would make informed purchasing decisions.

keywords: ICT, Electronic Commerce, NLP, word embeddings

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Electronic Commerce, commonly known as e-commerce, involves buying and selling of products or rendering services over computer networks, such as the Internet [1]. Online customer reviews may be defined as product evaluations posted by customers on a company's or a retailer's website [2]. For instance, on Amazon[1], the online retailer and cloud computing corporation's website, customers are offered the opportunity to express their shopping experiences in the form of numerical star ratings ranging from 1 to 5 stars and open-ended customer authored comments about the products. Consequently, there exists a wealth of customer reviews online. The Amazon corporation has a collection of more than 130 million customer reviews [3]. These reviews contain important information that could be useful to stakeholders such as producers and customers. For instance, a producer could improve upon his product design based on the comments customers make about the product. Also customers make informed purchasing decisions based on the reviews made by other customers about the product they intend buying. Below are examples of customer reviews on Amazon Marketplace.

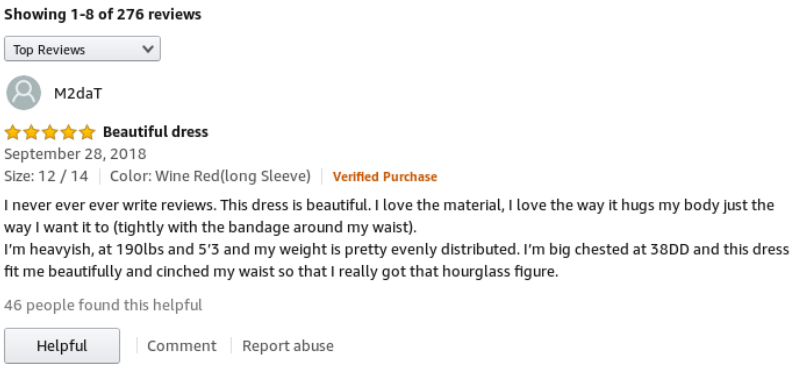

Figure 1.1: A customer review on a dress on Amazon

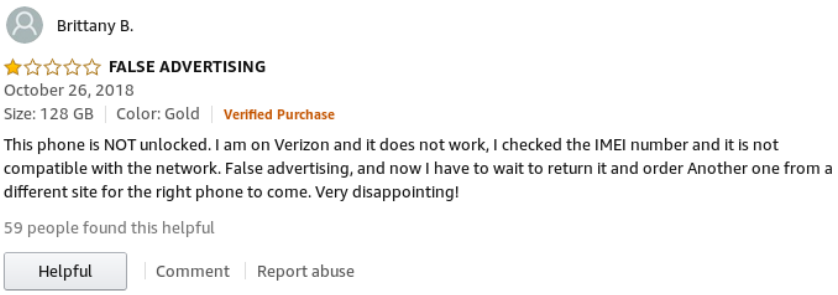

Figure 1.2: A customer review on a phone on Amazon

Given the enormous amount of customer reviews available, it is tedious and time consuming to

[1] https://www.amazon.com/

manually inspect and analyse these reviews. Due to these constraints, producers and customers are unable to accrue the benefits from the enormous body of customer reviews available online.

To address this issue, in this thesis we present an approach that can automatically assign labels to clusters of customer reviews. This approach helps the producers and consumers to have a summary or overview of the customers feedback without manually analysing the vast pool of reviews online.

1.2 Approach

Our goal is to build a pipeline that automatically labels customer reviews. Firstly, we partition the customer reviews into two groups based on the star rating given for each review. Here star rating less than or equal to 2 is considered as negative, while any review with star rating greater than 2 is considered as a positive review. We assume that the first group contains all the negative reviews and the second group contains all the positive reviews. We present an example of negative and positive reviews below.

```
  star_rating      review_body
2    Was soon looking forward to it...
2      Honestly I didn't care for it...
1      Movie is great - problem was...
2      Poor quality video...
1      A financial loss for me ...
1      I hate powerpuff girls because ...
1      Perhaps the saddest thing is...
```

```
  star_rating      review_body
5      Fast shipping. Pleasure to deal...
5      Kevin Kline is the most versatile,
4      great movie
3      Y'know what this reminded me of?
5      Wonderfully accurate portrayal...
4      In great shape for how old it is!
```

Secondly, we pre-process the words in each review by tokenizing the words in the corpus, removing non-alphabetic characters and punctuation marks such as "?",",", and "!". Also, we remove frequently occurring words which carry low information (stop words) such as "and", "or" and "the".

Thirdly, we use Embeddings from language model (ELMo) to convert reviews into vectors. We will provide further details and a definition for Embeddings from Language Model in the next chapter.

Fourthly, we perform thematic clustering on each set of review vectors (vectors for positive and negative reviews) using Density Based Spatial Clustering of Applications with Noise (DBSCAN).

Lastly, we find the Pointwise Mutual Information (PMI) scores between words and the clusters obtained. This approach helps us to determine the most informative words in a cluster. The most informative words are used as labels for each cluster.

We present the algorithm for our pipeline below:

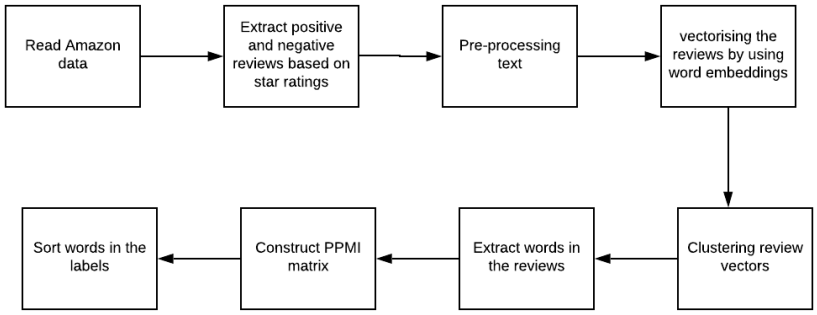

Figure 1.3: The pipeline for our algorithm

1. read the tsv file containing the Amazon reviews,
2. extract both positive and negative reviews based on the star ratings given by the customers. Put each review category into a list,
3. write a function that pre-processes each review by removing non-alphabetic characters and stop words, and returns a list containing lists of tokenised reviews,
4. create a for loop that picks each list of tokenised reviews from the bigger list, and converts each review into a vector using embeddings. Particularly, we used ELMo in this thesis,
5. run a clustering algorithm on the vectors. We used DBSCAN for our clustering,
6. extract the words in each cluster,
7. compute Positive Pointwise Mutual Information (PPMI) between words and clusters to determine the most informative words in a cluster.
8. Sort the labels based on their PPMI scores.

This thesis is organized as follows: Section 2 provides the details of the state-of-art approaches for converting natural language in the form of words into vectors. Section 3 presents clustering analysis and PPMI matrix. Section 4 presents a detailed implementation of our pipeline. Section 5 presents the discussion of the results obtained for the work.

2. Preliminaries

Language modelling is a task pivotal to the study of Natural Language Processing (NLP). It helps carry out tasks such as question answering [4], textual entailment [5] and semantic role labelling [6] among others. The objective of Language modelling is to train language models to learn the likelihood of occurrence of words based on previous sequences of words used in text. Language models can be trained on character level, word level, sentence level or paragraph level [7]. Computers do not have the ability to understand human natural language, hence to be able to create a platform for computers to explore the knowledge in human natural language, we encode words in a numerical or vectorial form. This form of representing words is called word representations or word embeddings.

Before we proceed to how word representations have evolved over time, let us clarify two ways of referring to words in a text. The first way of referring to words in a text, is by the term "word token" (see e.g. 2.0.0.1) and the second way is by the term "word type" (see e.g. 2.0.0.1). A word token refers to the total number of word units in a text or corpus irrespective of how often they may be repeated. While word type refers to the number of distinct words in a text or corpus. For instance, consider the following example of a text:

Example 2.0.0.1. As shades of colors tell the stories of life; we tell the african stories from an african perspective.

In Example 2.0.0.1, the word "tell" appeared twice, and the same thing happens for the words "african" and "stories". Hence, there are 18 word tokens and 13 word types in this particular example of text.

Now let us proceed to how word representation has evolved over time.

2.1 Word Representations

Hitherto, word types in a text were represented by integers, thus this approach of representing word types is referred to as discrete representations [7]. In the discrete representation approach, non-negative integers are used to represent word types in a text. One advantage of discrete representations is that the set of non-negative integers has an infinite number of elements and hence the body of words used in a particular language could be continuously expanded as we encounter new word types [7]. The assignment of integers to word types could be in an alphabetical order or based on the order of occurrence of the word type [7].

However, one limitation of discrete representation is that for instance, two word types with similar meanings might be assigned integers whose difference is huge. When these two integers are plotted on a number line, they may appear to be far apart although the words may be semantically related.

To perform complex NLP tasks, the computer should be able to determine the sense of a word and the similarity among words in a text corpus [7]. Therefore, words are transformed into vectors in

order to capture word sense in each of the dimensions of the vector [7]. For example, a collection of word types can be referred to as a class, (e.g., colours) and we can assign 1 to a specific dimension of the vector if the word type belongs to the word class, or 0 otherwise.

Moreover, we can use dimensions to capture the sense of magnitude. For example, we may assign 1,000,000 to the word "sun" and 1 to the word "atom". Each of the dimensions of a vector associated to a word type is called a feature. Features can be derived from a word's vector by use of automated algorithms or they could be designed by experts [7].

One way of representing word types as vectors is by the one-hot encoding vectors. This method involves assigning a binary value (0 or 1) to each of the dimensions of word vectors. In the one-hot encoding approach each word type has its own dimension. We assign 1 to the dimension of the word type and assign 0 to the other remaining dimensions of the one-hot encoding vector [7]. For the sake of illustration let us consider the following text comprised of two sentences:

Example 2.1.0.1. Ama is a beautiful girl. Ama is a pretty girl.

The vocabulary set denoted as V for this text is $V = \{$Ama, is, a, beautiful, pretty, girl.$\}$ Now let us create a one-hot encoded vector for each word type in the vocabulary, V.

Example 2.1.0.2. **Ama**=[1,0,0,0,0,0], **is**=[0,1,0,0,0,0], **a**=[0,0,1,0,0,0], **beautiful**=[0,0,0,1,0,0], **pretty**=[0,0,0,0,1,0], **girl**=[0,0,0,0,0,1].

The length of the one-hot encoded vector in Example 2.1.0.2 is equal to the size of the vocabulary V. Hence the vectors we obtained in Example 2.1.0.2 lie in a 6 dimensional vector space where each word type lies in each dimension. When we take the dot product of any two distinct vectors of the basis, we obtain zero, e.g.

$$\textbf{beautiful} \cdot \textbf{pretty} = 0 \tag{2.1.1}$$

and

$$\textbf{Ama} \cdot \textbf{is} = 0. \tag{2.1.2}$$

According to one-hot encoding technique, the Equation 2.1.1 implies that the word types "beautiful" and "pretty" have nothing in common just as in Equation 2.1.2 the word types "Ama" and "is" have nothing in common. In reality, this is false.

Another limitation is that, one-hot encoded vectors are highly dimensional, sparse and they could be computationally expensive especially when the vocabulary size increases [7]. Therefore, one-hot encoded vectors are analogous to integer-based representation-they don't convey semantic relationship among words [7]. Hence, they are not useful in performing NLP tasks. In view of this we consider some state-of-the-art approach of producing word embeddings that carry semantic meanings and are greatly useful in various NLP tasks.

2.2 State-of-the-art Word Representations

Currently, the state-of-the-art approach to producing word embeddings is based on neural networks. According to Patterson and Gibson [8], "neural network is a computational model that

shares some properties with the animal brain in which many simple units are working in parallel with no centralized control unit". Mathematically, neural networks are mathematical models defining a function $f : X \mapsto Y$. The state-of-the-art approach to producing word embeddings such as Word2Vec [9], GloVe [10] and ELMo [11] use neural networks to find the vector mappings for each word in a given corpus.

2.2.1 Word2Vec. Word2Vec introduced by Mikolov et al. [9] in 2013, is a model used to produce word embeddings. The main idea behind it is that words with similar context (neighbouring words) have similar meaning. This confirms John Firth, an English linguist's quote, "You shall know a word by the company it keeps" [12]. The Word2Vec model is a two layer neural network which takes a corpus of text as its input and produces a high dimensional vector as output, with each word type in the corpus being represented by a vector. Word2Vec produces vectors such that words with similar context in the corpus are close to each other in the vector space [9]. Let us illustrate a bit more on Word2Vec with the next example.

Example 2.2.1.1. Consider the following text: the capital city of Ghana is Accra and the capital city of Togo is Lome.

In the vector space produced by Word2Vec, the difference between "Ghana" and "Accra" is the same as the difference between "Togo" and "Lome". Also, the vectors for "Ghana" and "Togo" would have close proximity whereas the vectors for "Ghana" and "onions" would be far apart in the vector space. Amazingly, mathematical operations could be applied to the word vectors to yield reasonable results [9]. For instance,

$$Ghana - Accra + Togo = Lome. \tag{2.2.1}$$

There are two approaches or algorithms to convert words to vectors using Word2Vec. They are the skip-gram based model approach and the continuous bag of words approach.

To start with, the skip-gram model approach seeks to predict the surrounding words of a target word in a corpus, given a fixed window size (the radius of words surrounding the target word to the right and left). By illustration let us consider the sentence

Example 2.2.1.2. Ghana's chocolate is the **best** in the world.

Let the word "best" be the target word. Suppose we fix a window size of 3, the context words to the left is "the","is" and"Ghana's" while the context words to the right is "in","the" and "world". The continuous bag of words approach is the reverse of the skip-gram based model approach. It predicts the target word given a context.

We would like to proceed by examining the mathematics behind Word2Vec. The skip-gram based model approach predicts the words of each context independently given the target word. The objective of Word2Vec is to maximise the probability of the context given the target word across the corpus [9]. The probability of predicting a context given its target word across the corpus can be mathematically expressed as follows [9]:

$$\prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} p(w_{t+j} \mid w_t; \theta). \tag{2.2.2}$$

Where

- θ denotes the hyper-parameters of the model (word representations of the target word),
- m stands for the window size (radius around the target word),
- T is the size of the corpus,
- j denotes the index of the words that constitute the context of the target word w_t for a given t,
- t denotes the index of a target words,
- w_t denotes the target word given the index t, where $t = 1, \cdots, T$.

After applying the logarithm function to the expression 2.2.2, we obtain

$$\sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} \mid w_t; \theta). \tag{2.2.3}$$

Let us take the average of the terms in the mathematical Expression 2.2.3 so that instead of taking the probability of the whole corpus we could take the average probability, that is probability of each word. Let us minimise the objective function by introducing a negative sign

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} \mid w_t; \theta). \tag{2.2.4}$$

Equation 2.2.4 now becomes the objective function we seek to minimise.

Similarly, for the continuous bag of words approach, the context words are summed in order to predict the target word. In this case, the objective function of the corresponding minimisation problem that underpins Word2Vec is given by:

$$Z(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \log p(w_t \mid \sum_{-m \leq j \leq m, j \neq 0} w_{t+j}; \theta). \tag{2.2.5}$$

In the Word2Vec model, each word type has two vector representations: the context embedding vector and the target embedding vector [9]. The probability distribution is defined by the softmax function [8]. The probability distribution for skip-gram based model is

$$p(w_c \mid w_t) = \frac{\exp(v_{w_c}^T u_{w_t})}{\sum_{w=1}^{W} \exp(v_w^T u_{w_t})}. \tag{2.2.6}$$

Also, the probability distribution for continuous bag of words is

$$p(w_t \mid w_c) = \frac{exp(u_{w_t})^T v_{w_c}}{\sum_{w=1}^{W} \exp(u_{w_t}^T v_w^T)}. \tag{2.2.7}$$

Where u_{w_t} represents the word vectors for the target words and v_{w_c} represents the word vectors for the context words.

The softmax function would always yield a positive real number since the numerator is made up of an exponential function and the denominator is made up of the sum of exponential functions. Moreover, the softmax function would yield a real number between 0 and 1 due to the normalization factor in the denominator.

In a nutshell, Word2Vec is the state-of-the-art approach for building word vectors. It scans each word in the corpus, predicts the context of each word and captures co-occurrence of words one at a time. This approach is quite inefficient supposing our corpus is really large. It is suggested that an optimal way is to scan through the entire corpus once and count the frequency of co-occurring words [10]. Also, instead of making updates on samples one at a time, it is more efficient to make one update step that captures the entire word count. To tackle these limitations, we introduce the next state-of-the-art approach for producing word vectors called Global Vectors for Word Representation (GloVe) [10].

2.2.2 GloVe. There are two main approaches to produce word representations. These are:

1. matrix factorization method,
2. local window context method.

The matrix factorization method seeks to decompose large matrices that contain statistical information about the corpus, (e.g., window based co-occurrence matrix), by utilizing low rank approximations such as the Singular Value Decomposition (SVD) [10].

The rows and the columns of the co-occurrence matrix corresponds to word types in the corpus and the entries corresponds to the frequency of a given word as it occurs in the context of another word.

Let us take an example of a co-occurrence matrix. Given the three sentences below:

1. I like potatoes.
2. I enjoy pizza.
3. I like reggae.

The vocabulary set V becomes

$$V = \{\text{I, like, enjoy, potatoes, pizza, reggae,.}\} \tag{2.2.8}$$

From Table 2.1 we could see that "like" and "enjoy" co-occur with the word "I". This gives a clue about their similarity. Increasing the size of the corpus would enable us appreciate these similarity relationships much better.

The rows of the co-occurrence matrix could be used to form word vectors. However, there are limitations to the use of a co-occurrence matrix. Firstly, the matrix increases in size as we

count	I	like	enjoy	potatoes	pizza	reggae	.
I	0	2	0	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	1	0	0
potatoes	0	1	0	0	0	0	1
pizza	0	0	1	0	0	0	1
reggae	0	1	0	0	0	0	1
.	0	0	0	1	1	1	0

Table 2.1: A co-occurence matrix for the vocabulary V

increase the vocabulary. Hence, the word vectors generated would have many dimensions, thus requiring lots of storage space and computational cost may increase. Also, one may encounter sparsity issues when using these vectors to train machine learning models, such as classification models. To solve this problem we may employ a dimensionality reduction method such as Principal Component Analysis (PCA) [13] .

One main limitation of the matrix factorisation approach is that it gives disproportionate importance to frequently occurring words in the corpus, for example words like "the", "is" and "and" among others. This gives the wrong impression that these words(they are also called stop words) share common semantic relationships with the words they co-occur with.

On the other hand, window-based methods seek to learn word representations that enables us to predict words within local context windows [10]. An example of this approach is the skip-gram based model, which we have explained earlier under Word2Vec approach.

Unlike the matrix factorisation approach for learning word vectors, window-based methods are deficient in the information carried by the statistics of co-occurring words of the corpus, therefore it fails to leverage the rich amount of repetitions in the data.

In summary, the matrix factorisation approach complements the window-based approach. The former is fast to train while the latter is slow to train. Also, matrix factorisation makes good use of the statistics of co-occurrence of words in the corpus thereby capturing word similarity well, although it gives undue importance to some frequently occurring words and it performs poorly on specific tasks such as question answering [4]. On the other hand, the window-based approach makes inefficient use of co-occurrence statistics but performs well in specific tasks such as named entity recognition and word analogies. Besides similarity relationships, the window based approach also captures syntactic relationships among words [10].

GloVe, introduced in 2014, was built to take advantage of the efficiencies of the two main approaches for learning word vectors.

The objective function of GloVe is of the form,

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})\left(u_i^T v_j - logP_{ij}\right)^2. \tag{2.2.9}$$

Where

- θ denotes all the parameters of the model
- P_{ij} denotes the co-occurrence matrix of the words in the corpus
- W denotes the size of the vocabulary.

GloVe seeks to minimize the distance between the inner product $(u_i^T v_j)$ and the log count of these words, P_{ij}. Also, f allows us to weight the frequent counts of co-occurrences, that is we try to suppress the count of frequent words (stop words) in the objective function.

2.2.3 Embeddings from Language Models (ELMo). So far we have examined two major state-of-the-art approaches for generating word vectors- Word2Vec and GloVe. These word embeddings represent word types using a vector. However, they do not capture the polysemous (words with more than one meaning) nature of some words in natural language.

Let us consider the following sentences:

1. He fans the fire using a book cover.
2. His fans love him although he is polygamous.

In the first sentence "fans" is a verb which describes the act of rekindling a fire by blowing air while in the second sentence "fans" refers to the followers of a celebrity. Although the word "fans" is spelt the same in both sentences, they have different meanings in different contexts. By using Word2Vec or GloVe it is highly likely that the word "fans" would be represented by the same vector in both sentences. Thus context-independent vectors do not sufficiently model natural language.

In view of this, we introduce a novel contextual word representation-Embeddings from Language Models (ELMo), which was introduced by Peters et al. [14] in 2018. ELMo is based on neural networks, specifically it is a function of all the internal layers of a bidirectional language model (biLM) [14], hence it is often called deep contextualised word representation. One difference between ELMo and the other context-independent word embeddings is that it produces word vectors for word tokens instead of word types.

Let us go deeper into language models. Language models seek to find the probability distribution of the next word in a sequence given the sequence of the preceding words. A language model is trained by reading sentences in a corpus in the forward and backward direction. Thus we have forward language model and backward language model.

Given a sequence of N tokens $(t_1, t_2, ..., t_N)$ in a corpus, a forward language model seeks to predict the next word token given the sequence of previous words. While the backward language model is the reverse, given the future context it seeks to find the probability of the previous word token. Mathematically, we express the forward language model as

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, ..., t_{k-1}) \tag{2.2.10}$$

while the backward language model is expressed as

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, ..., t_N). \tag{2.2.11}$$

A bidirectional language model combines both forward and backward language models in predicting word tokens in a corpus.

Our objective in this model is to jointly maximize the log likelihood of both forward and backward language models. Hence we obtain

$$\sum_{k=1}^{N} \left(log\ p(t_k \mid t_1, ..., t_{k-1}, \theta_x, \overrightarrow{\theta}_{\mathsf{LSTM}}, \theta_s) + log\ p(t_k \mid t_{k+1}, ..., t_N, \theta_x, \overleftarrow{\theta}_{\mathsf{LSTM}}, \theta_s) \right). \tag{2.2.12}$$

Where

- θ_x denotes the parameter for token representation
- θ_s denotes the parameter for the softmax layer
- $\overrightarrow{\theta}_{\mathsf{LSTM}}$ denotes the parameter for forward language model
- $\overleftarrow{\theta}_{\mathsf{LSTM}}$ denotes the parameter for backward language model.

Both the forward and backward directions share the same representation parameter and softmax parameter while we keep separate parameters for each of the Long Short Term Memory (LSTM) for each direction.[14]

Given L number of stacked LSTMs, for each token we train $L-$layer LSTM forward and backward language model, thus we generate $2 \times L + 1$ different vector representations. Therefore, the word embedding for input token k is

$$R_k = \left\{ x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} \mid j = 1, ..., L \right\} \tag{2.2.13}$$

$$= \left\{ h_{k,j}^{LM} \mid j = 0, ..., L \right\}. \tag{2.2.14}$$

Here $x_k^{LM} = h_{k,0}^{LM}$ is a context-independent token representation. We pass the context-independent vector through the $L-$ layers of forward and backward LSTMs. Each LSTM layer returns a context-dependent representation $\overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM}$ at each word token position k [14].

The combination of the intermediate layer representations in the biLM is used to perform specific tasks such as sentiment analysis [4], named entity recognition [11], co-reference resolution [4], among others [14].

To obtain an ELMo word representations customised or tailored for a specific task we find the linear combination of the $h_{k,j}^{LM}$ and some parameters. Hence we have

$$ELMo_k^{\mathsf{task}} = \gamma^{\mathsf{task}} \sum_{j=0}^{L} s_j^{\mathsf{task}} h_{k,j}. \tag{2.2.15}$$

Where γ^{task} allows the task model to scale the entire ELMo vector and s_j^{task} is the softmax-normalized weight for each layer [14].

Not only is ELMo a contextual word vector, it models complex characteristics such as syntactic and semantic word use. It is shown that the higher-level LSTM states models the semantic aspects of words whereas the lower level LSTM captures the context-dependent aspect of word structure such as word syntax [14]. Also, ELMo succeeds in modelling polysemous language due to the fact that it returns unique vectors for each word token.

Unlike Word2Vec and GloVe which are more or less like lookup tables, ELMo word vectors include both word type vectors and neural network parameters [7]. This implies that word type vectors are not redundant however they could be concatenated with context-dependent embeddings to perform well on downstream tasks [14].

3. Approach

3.1 Clustering

Clustering involves grouping a set of objects in such a way that objects that belong to the same group tend to be more similar to each other compared to objects from other groups. Unlike classification where objects are grouped based on some pre-existing labels, clustering does not require any class labelling or pre-existing categorization. Clustering partitions data sets according to their intrinsic categorization. Hence it is referred to as an unsupervised technique for grouping data. A group of similar data points is called a cluster. Clusters are formed based on the similarity among data points. A similarity is an estimate of the strength of the relationship among data points [15]. In most clustering algorithms distance functions such as Euclidean distance, Cosine similarity, Minkowski distance among others are often employed as a measure of similarity [15]. Hence, data points that belong to the same cluster has a minimum distance value while data points across different clusters are a certain maximum distance apart.

When our dataset becomes large and the dimension of the data set increases, it gives rise to the following requirements [16] on cluster algorithms:

1. determination of the input parameter,
2. discovery of clusters with arbitrary shapes,
3. efficiency on large data base.

There are two traditional approaches to clustering algorithms [17]. These are,

1. hierarchical clustering algorithm,
2. partitioning clustering algorithm.

We have two types of hierarchical clustering approach, viz., agglomerative hierarchical clustering and divisive hierarchical clustering [17]. The agglomerative type of hierarchical clustering involves considering each data point as an individual cluster. We merge similar clusters at each iteration until we form k number of clusters, $k \geq 1, k \in \mathbb{N}$. However, the divisive hierarchical clustering algorithm involves considering the entire dataset as one mass cluster. At each iteration, non-similar data points are separated from the cluster. Here each separated data point is considered as a cluster on its own. One setback of the hierarchical clustering algorithm is the need to define the termination condition that indicates when to merge or separate clusters [16]. Examples of hierarchical clustering approach are, Balance Iterative Reducing and Clustering (BIRCH) and Cluster Using REpresentatives (CURE) [18].

Unlike hierarchical clustering, partitioning clustering algorithm involves grouping the data points into sets, each of which is a partition of the data and is comprised of k number of clusters. Each

partition or group on its own represents a cluster and must contain at least one object which belongs to exactly one partition [19]. This type of algorithm has no notion of outliers. There are four main types of partitioning clustering methods, viz. K-means Algorithm [19], K-mediods Algorithm [20] and Clustering for Large Applications based on RANdomized Search (CLARANS) [21]. Let us examine the method by K-means.

The K-means clustering algorithm was introduced by James Macqueen in 1967 [19]. The objective of the K-means is to obtain a minimum distance between the centroid or mean of the cluster and its neighbouring data points.. Thus the objective function used for the K-Means is the sum of deviations between a data point and its centroid using an appropriate distance function. Below is the mathematical expression of the objective function.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \|x_{ij} - c_j\|^2 \tag{3.1.1}$$

Where

- J is the objective function,
- k is the number of clusters,
- n is the number of data points,
- x_{ij} is the ith data point in the jth cluster,
- c_j is the mean for data points in cluster j,
- $x \mapsto \|x\|$ is the Euclidean distance function.

For a given set of data points $x_1, x_2, \cdots x_n$, the centroid, which we denote by C , is defined as follows: The centroid C, is computed as

$$C = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{3.1.2}$$

Where C is the average of all the points in the cluster. The coordinates of the centroid of each cluster is the arithmetic mean for each dimension over all the points in the cluster.

Let us consider the algorithmic steps for K-means clustering. Let $x = \{x_1, x_2, ..., x_N\}$ be the set of data points and $C = \{c_1, c_2, ..., c_k\}$ be the set of centroids. We present the procedure for finding the optimal clusters using K-Means approach.

1. Randomly select "k" data points as cluster centroids,
2. compute the distance between each data point and the clusters' centroids,
3. assign each data point to the nearest centroid, i.e., data point whose distance from the cluster center is minimum as compared to all other centroids,

4. recompute the new cluster centroid using the formula

$$c_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_i \tag{3.1.3}$$

where n_j is the number of data points in the jth cluster and $x_1, \cdots x_{n_j}$ are the data points in this cluster.

5. recompute the distance between each data point and the newly obtained cluster centroids.

6. stop if no data point is reassigned to a new cluster. Otherwise, repeat 3, 4, and 5.

K-means is easy to understand and implement, computationally faster than hierarchical clustering methods and yields the best results when the data set is well separated from each other. However, its limitations are that they require a priori specification of the k number of centroids and it performs poorly when the data set is noisy. Besides, the algorithm fails when the data set is non-linear.

We introduce K-medoids which is another approach for partitioning clustering approach. We have already stated that K-means is sensitive towards outliers hence we introduce another partitioning clustering approach which is less sensitive to outliers. The differences between K-medoids and K-means is that while K-means takes the means of the data points as the representative of each cluster, K-medoids takes the medoid-which is the most centrally located object in the cluster, as the representative of the cluster. Also, in K-means the representative of a cluster may not necessarily belong to the data set, however each medoid used in k-medoids algorithm belongs to the data set. The objective for k-medoids is similar to that of K-means, in that we seek to minimise the distance between the points and medoids in each cluster. The strategy is to find k clusters given n data points by first randomly selecting k number of medoids for each cluster. The remaining data points are assigned to each cluster based on their closeness to the medoid of each cluster.

The objective function is expressed mathematically as

$$E = \sum_{j=1}^{k} \sum_{i=1}^{n} \|x_{ij} - m_j\|^2. \tag{3.1.4}$$

Where

- E is the objective function,
- k is the number of clusters,
- n is the number of data points,
- x_{ij} is the ith data point in the jth cluster,
- m_j is the medoid for cluster j,

- $x \mapsto \|x\|$ is the Euclidean distance function.

Let us consider the algorithmic steps for k-Medoids clustering. Let $x = \{x_1, x_2, ..., x_N\}$ be the set of data points and $M = \{m_1, m_2, ..., m_k\}$ be the set of medoids. We present the procedure for finding the optimal clusters using k-Medoids approach.

1. Randomly select "k" cluster medoids,
2. compute the distance between each data point and cluster centroids,
3. assign each data point to the nearest medoid, i.e., data point whose distance from the cluster medoid is minimum as compared to all other medoids,
4. compute the cost, that is the sum of distances of points to their mediods,
5. for each medoid m , for each non-medoid data point o, swap m and o, re-assign each data point to the closest medoid,
6. recompute the sum of the distances of points to their medoids,
7. if the total cost increases in the previous step undo the swap,
8. if the total cost decreases, swap the medoid m with o to form the new set of $k-$Mediod,
9. repeat 5,6,7 and 8 until there is no change in the sum of the distances of points to their medoids.

The two main approaches to clustering algorithm-hierarchical and partitioning clustering methods fail to meet all the requirements when the database is large. Hence, we consider the density based approach which was explored by Jain in 1988 [22]. The density based notion of constructing clusters involves partitioning a datavset into non-overlapping cells and constructing a histogram based on the cells created. Here cluster centers corresponds to the cells with high frequency counts (high density) whereas areas of low density may be classified as "noise". This is a density based approach to distinguishing between clusters and noise.

3.1.1 Density Based Spatial Clustering of Applications with Noise (DBSCAN). This algorithm was introduced by Ester et al. [16]. It relies on the density based notion of clustering to discover the clusters and noise in a dataset. The main idea behind DBSCAN is that every point in the cluster must contain a minimum number of points in its neighbourhood determined by a specified radius. DBSCAN has two input parameters, performs better than other algorithms on a large dataset and discovers arbitrary cluster shapes in any dimensional space [16]. Before we explore how DBSCAN is implemented let us consider some definitions and lemmata pertaining to density based notion of clustering.

Definition 3.1.1.1 (Eps-neighbourhood of a point). The Eps-neighbourhood of a point p denoted by $N_{\mathsf{Eps}(p)}$, is defined by $N_{Eps}(p) = \{q \in D : \mathsf{dist}(p, q) \leq \mathsf{Eps}\}$[16].

Where p and q are points in the dataset D, and Eps is the radius of the neighbourhood with respect to the point p and $dist()$ is an arbitrary distance measure.

There are two types of points in a cluster. These are;

- core points,
- border points.

A core point in a cluster has at least a minimum number of points denoted as $MinPts$ in its Eps-neighbourhood while the border point has less than the minimum number of points in its Eps-neighbourhood. All neighbours in the Eps-strip of a core point is considered to belong to the core point's cluster. This notion is termed as direct density reachability. Let us take a formal definition.

Definition 3.1.1.2 (directly density-reachable). A point p is directly density-reachable from a point q with respect to Eps, $MinPts$ if

- p$\in N_{Eps}$(q) and
- $\mid N_{Eps(q)} \mid \geq MinPts$ (core point condition) [16].

If any of the points, say q in the neighbourhood of a core point, say p, is a core point, then the neighbourhood of q is included in the cluster formed by p. This notion is termed as density-reachability.

Definition 3.1.1.3 (density-reachable). A point p is density reachable from a point q with respect to Eps and$MinPts$ if there is a chain of points $p_1, \cdots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i [16].

All points within a cluster is said to be density-connected. Let us take a formal definition of density-connectedness.

Definition 3.1.1.4 (density-connected). A point p is density connected to a point q with respect to Eps and $MinPts$ if there is a point o such that both, p and q are density-reachable from o [16].

Let us consider a formal definition of a cluster.

Definition 3.1.1.5 (cluster). Let D be a dataset of points. A cluster C with respect to Eps and $Minpts$ is a non-empty subset of D satisfying the following conditions:

- $\forall p,\ q$: if $p \in C$ and q is density reachable from p with respect to Eps and $MinPts$, then $q \in C$.
- $\forall p,\ q$: p is density-connected to q [16].

Any point that does not belong to a cluster is considered as noise. Let us take a formal definition of a noise

Definition 3.1.1.6 (noise). Let $C_1, \cdots, C_k$ be the clusters of the dataset D with respect to parameters Eps_i and $Minpts_i$, $i = 1, \cdots, k$. Then we define the noise as the set of points in the dataset D not belonging to any cluster C_i, i.e. noise $= \{p \in D : \forall i : p \notin C_i\}$ [16].

3.2 Pointwise Mutual Information (PMI)

According to Bouma [23], "Pointwise Mutual Information is a measure of how much the actual probability of a particular co-occurrence of events $p(x, y)$ differs from what we would expect it to be on the basis of the probabilities of the individual events and the assumption of independence $p(x)p(y)$". The formula for computing PMI for the random variables X,Y given a pair of outcomes x and y is defined as:

$$PMI(x, y) = log_2 \frac{p(x, y)}{p(x)p(y)} \tag{3.2.1}$$

where $p(x)$ and $p(y)$ are marginal probabilities and $p(x, y)$ is the joint probability. The numerator of Equation (3.2.1) expresses how often x co-occurs with y and the denominator expresses the probability of x and y occurring together, given they occurred independently [24]. Pointwise mutual information is one of the efficient ways to estimate the association between words [25]. It estimates how much more two words co-occur in our word corpus than we would have expected them to occur a priori by chance. [24].

The formula for PMI between a target word w and a context word c is given as [25]

$$PMI(w, c) = \log_2 \frac{p(w, c)}{p(w)p(c)}.$$

The values of PMI are elements of $\mathbb{R}$, that is it assumes both positive and negative values. If PMI is negative it implies that there is a weak association between the target and context words. Also, if PMI value is positive it implies that there is a strong association between words. PMI value being 0 implies that the target word and context word are independent, that is, $p(w, c) = p(w)p(c),\ \log_2(1) = 0$.

Negative PMI values could be misleading if the size of our corpus is not enormous [24]. In view of this, let us replace all negative PMI values with 0. This approach is called Positive Pointwise Mutual Information (PPMI) [25]. The formula for PPMI is given as

$$PPMI(w, c) = max\left(log_2 \frac{p(w, c)}{p(w)p(c)}, 0\right). \tag{3.2.2}$$

In our approach, to find the PPMI of each word in a cluster we need to first compute a word-cluster matrix.

3.3 Word-Cluster Matrix

The frequency of the occurrence of t words in the entire document is represented by a word-cluster matrix. The word-cluster matrix constitutes n clusters and t rows. Each column represents a cluster of reviews obtained from DBSCAN and each row represents the unique words in the entire document. Each entry a_{ij} of the matrix, A, represents the frequency count of the ith word in the jth cluster. The PPMI of each entry in the matrix can be computed as:

$$ppmi_{ij} = \begin{cases} P_{ij} & \text{if } P_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3.1}$$

where

$$P_{ij} = \log\left(\frac{p_w}{p_u p_s}\right), \tag{3.3.2}$$

$$p_w = \frac{a_{ij}}{\sum_{i=1}^{t}\sum_{j=1}^{n} a_{ij}}, \tag{3.3.3}$$

$$p_u = \frac{\sum_{j=1}^{n} a_{ij}}{\sum_{i=1}^{t}\sum_{j=1}^{n} a_{ij}}, \tag{3.3.4}$$

$$p_s = \frac{\sum_{i=1}^{t} a_{ij}}{\sum_{i=1}^{t}\sum_{j=1}^{n} a_{ij}}. \tag{3.3.5}$$

Here, p_w is the probability that the word w_i occurs in the cluster j with respect to the entire document, p_u is the probability of the word w_i occuring in the entire document and p_s is the probability of a cluster s_j in the entire document.

4. Implementation

We present the implementation of our approach for obtaining labels automatically given any customer reviews.

4.1 Read Amazon data

We first read the tab separated text file, without quote or escape characters containing customers' details on Amazon. This data set contains columns for marketplace, customer_id, review_id, product_id, product_parent, product_title, product_category, star_rating, helpful_votes, total_votes, vine, verified_purchase, review_headline, review_body and review_date [1]. Find below an example of how to read a tsv file in Python.

```
obs=[]
filename = "amazon_reviews_us_Video_v1_00.tsv"
file = open(filename, mode='r')
for line in file.readlines():
    obs.append(line.split('\t'))
```

4.2 Extracting positive and negative reviews based on star ratings

Next, we created a data frame for the dataset using Pandas [2]. We extracted the negative review_body (the review text) and positive review_body based on the star ratings given by the customers for each product purchased. Customer reviews with star ratings less than or equal to two are grouped as negative reviews while reviews with star ratings greater than two are grouped as positive reviews. The reason why we first grouped the positive and negative reviews is to help stakeholders such as producers to easily identify products that are successful on Amazon or products that do not meet the customers' satisfaction. This approach helps the producers to improve on their products in order to meet market demands. We present a sample of negative reviews extracted from Amazon customer reviews on musical instrument.

```
star_rating          review_body
2                    Note- Does not Fit Epiphone SG Special! Mine i...

2                    Bridge pickup was broken. I replace d the pick...
1                    I was hoping it would work well, but tried a s...
1                    It works well while it can. Mine failed when I...
```

[1] https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt
[2] https://pandas.pydata.org/pandas-docs/stable/

```
1                    Really bad. Bought as a midi trigger kit but t...
1                    sorry  but  wasnt made  right
2                    Novelty item. Fun. The delay on it is bad enou...
2                    did not fit the Asus
1                    The company accidentally sent me an alto inste...
2                    Makes lots of Noise while unit has nice feature...
```

We also present 10 samples of positive reviews extracted from the Amazon reviews on musical instrument.

```
star_rating        review_body
3        Works very good, but induces ALOT of noise.
5        Nice headphones at a reasonable price.
3        removes dust. does not clean
5        I purchase these for a friend in return for pl...
5        This is an awesome mic!
5        Used to cool equipment inside credenzas.  Work...
5        Well built Ukulele! My daughter loves it!
5        Had to replace a new light after a lightning s...
5        I've owned multiple fixed boards over the year...
5        Consistent quality, no strong odours, works we...
```

4.3 Pre-processing of text

To be able to perform any NLP task on the negative and positive reviews we first had to pre-process the reviews. We tokenised sentences in the documents into samples $S_i, i = 1, \cdots, n$, where n is the total number of reviews in either the positive or negative reviews. Next, we tokenise each word in S_i and converted all words in S_i into lower case. We also removed all non-alphabetic characters such as "#*&" and punctuation marks such as "?!,." from our document by using python module regex. Also, we removed words which are commonly used such as "and", "of", "in", "the","a" and "an". These words are known as stop words. Our goal is to use statistical analysis to extract words that best describes customers' reviews, for this reason we remove words and characters that are statistically insignificant for our work. We present below the reviews before text pre-processing.

```
["Sounds okay, but I had numerous problems with this microphone.
<br /><br />First, sections were just loose to begin with. Fortunately,
I found I could tighten the bottom section and that improved.
<br /><br />Next, both controls (the On/Off switch and the Volume control)
fell off within in the first few minutes of working with them.
To clarify, the caps or knobs fell off. The switch itself was still intact.
It felt very cheap.
<br /><br />Next, while it sounded good at first, I suddenly started
getting some loud 60Hz hum. This was isolated to where the USB cord
```

```
connected to the bottom of the mic.
Reconnecting everything firmly
reduced the hum but I don't know for how long.]
```

Let us consider the text after pre-processing.

```
[['sounds','okay','numerous','problems',
  'microphone','br','br', 'first',
  'sections','loose','begin','fortunately',
  'found','could', 'tighten','bottom',
  'section','improved','br','br',
  'next','controls','switch','volume',
  'control','fell','within','first',
  'minutes','working','clarify','caps',
  'knobs','fell','switch','still',
  'intact','caps','knobs','tiny',
  'rubber','like','pieces','could',
  'easily','lost','felt','cheap',
  'br','br','next','sounded',
  'good','first','suddenly','started',
  'getting','loud','hz','hum',
  'isolated','usb','cord','connected',
  'bottom','mic','reconnecting','everything',
  'firmly','reduced','hum','know','long']]
```

4.4 Vectorising the reviews using Embeddings from Language Modelling (ELMo)

After pre-processing our text we vectorised each review in the document by using ELMo. ELMo generates a three layered vector representation of each review with a fixed number of features, that is 1024 features. We extracted the top layer of the ELMo vector for our work because according to Peters et al. [14] semantic information is captured in the top layer of ELMo vectors. Below is an example of a review vector for the first negative review.

4.5 Clustering review vectors using DBSCAN

We perform DBSCAN clustering from the vector array produced by ELMo. The DBSCAN algorithm groups together review vectors that are close to each other based on a Euclidean distance and a minimum number of points. Review vectors in regions with points less than the minimum number of points are marked as outliers. The DBSCAN algorithm has two main parameters, these are eps and $minpts$. The eps specifies how close points should be in a cluster. A small eps

value would render most points in the data set to be considered as noise, however a large eps will cause clusters to merge, hence unsimilar points would be in the same cluster. Ester et al. [16] suggested we choose an arbitrary point p and find the distance to the fourth nearest neighbour in the data set. This distance will be our eps value. Also, we may fix eps based on domain knowledge [26], that is the task or application of clustering can inform how one could select the eps value. In this case it is advisable to vary the $minpts$ to improve clustering results.

The second parameter $minpts$ can be derived from the dimension of the data set. Sander et al. [27] proposed that we set $minpts$ to twice the dimension of the data set. However, it is also advisable to increase $minpts$ for improved clustering results when the data set is noisy or large [26]. Figure 5.2 is an image of our clustering result for 20 positive reviews. We employed a dimensionality reduction tool PCA to present a two dimensional distribution of the review vectors.

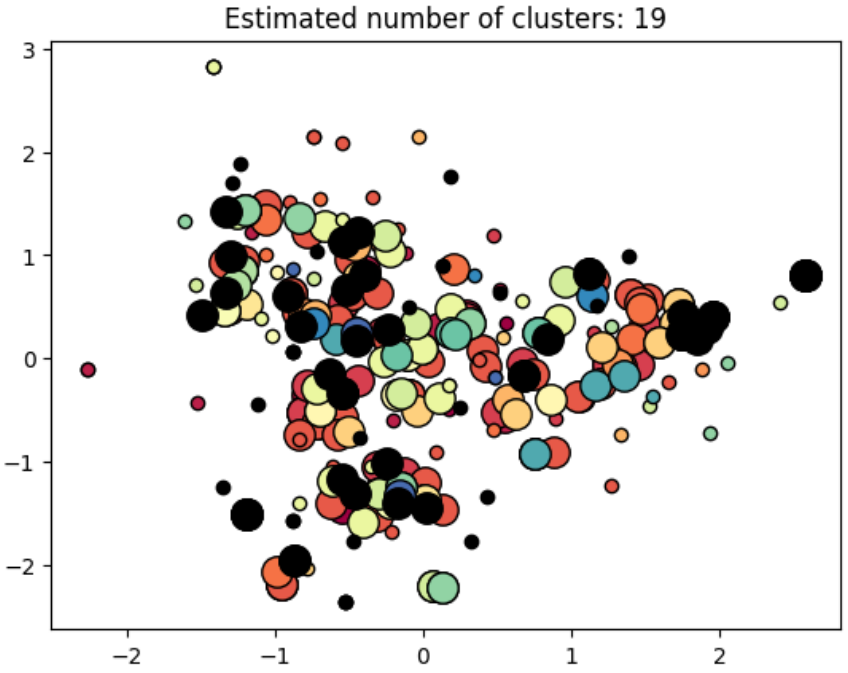

Figure 4.1: Clustering results for 20 positive reviews

Figure 4.2 is an image of our clustering result for 20 negative reviews.

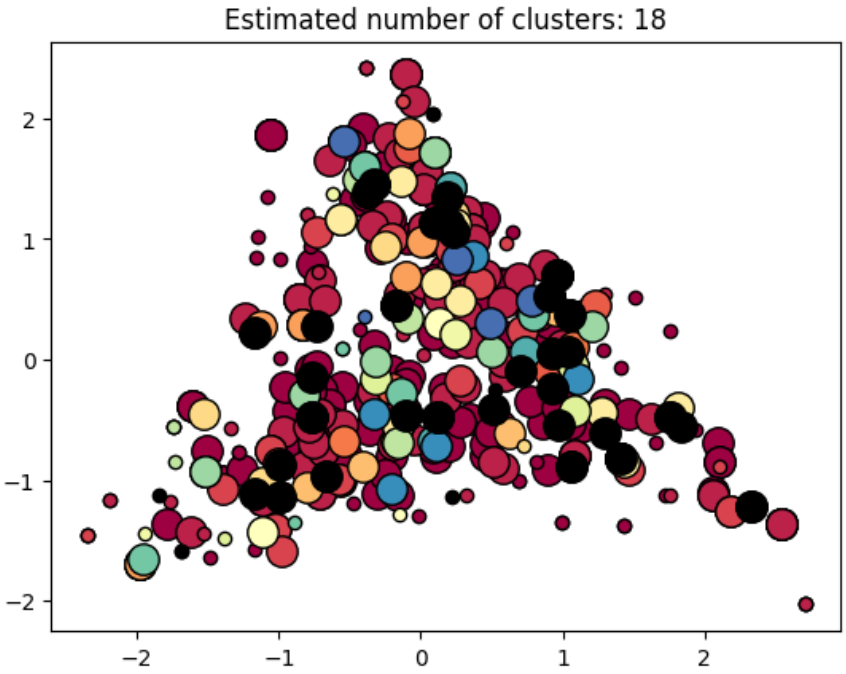

Figure 4.2: Clustering results for 20 negative reviews

4.6 Extracting the words in the reviews in the clusters

We extract the reviews in each cluster by using the indices of the cluster labels. DBSCAN clustering algorithm on scikit-learn [28] has attribute called labels. Cluster label of value -1 represents a noise point and all other non-negative cluster label value represents a point (either core point or border point) in the cluster. We extracted the indices of the core points and border points from the list of label values, thereafter we extracted the reviews that correspond with the indices of the core pints and border points in the cluster. We present the two clusters for our positive reviews.

```
Cluster 0:
['love lights bright vibrant colors remote worked change colors flash
come v adapter either',
'son drummer band bought surprise loves says fits drum gear room spare
thanks time purchase chalk one mom',
'works great happy product thanks',
'fantastic piano keyboard fair price got incredible sound quality
effects high quality really fun mess around keys feel
realistic well weighted touch screen icing cake']

Cluster 5:
['leans little stand straight',
'excellent product',
'nice quality product',
```

```
'tried loads tuners however reviewers said best bother anything else get',
'guitar excellent finish nice finish frets ruff edges nice would gave
stars action strings way high tried lower hit pickup could lowed enough
would come undone pickup still high still hit strings use pickup mounting
plate one vintage witch matched binding great much lower height black
one came bridge slammed bottom still strings high looking lower
bridge playable great looking guitar much better looking fender
squire vintage standard wise
cheep bridge hardware bridge design bad hardware used pickup
great sound cheep changing well
tele dreams']
```

We present the two clusters for our negative reviews.

```
 Cluster 3:
['item terrible expecting great quality begin bridge neck warped
tuning pegs
right cost repairs violin would proper luthier get damn thing
playable better ordering guitar center', 'records warped took package
pretty disappointed',
'made seatbelt material almost friction unless hold tightly hands
times strap causes guitar slide place apparently people like like
strap hold guitar nicely balanced waste energy trying keep guitar line',
'bought music studio within first week one broke thought
anomaly broken br br problem pretty simple made cheap plastic turn locking
points necessary mic stand plastic inserts simply crumble unless
purchasing whip small children la catholic nuns waste money inexpensive
pack reason',
'looked great put came weeks temporary',
'waste money piece junk',
'good hope another materials']

Cluster 10:
  ['firs snark purchased works great
  particular one turned
  consistently drained battery
  time go turn dead disappointed',
  'got even stay upright heavy
  support even making proper
  adjustments
  cheap worth excited price
  nope shoulda saved money
  put nicer one',
  'part tang guess called
  snapped playing well really
```

```
    hard guess broke
    playing snap went looks nice
    everything great price guess
    would almost
    recommend anyone experience
    one buy beginner though good
    learn learned next one get
    higher quality',
    'difficult use uncomfortable
    sound quality good returning
    update
    returned headset grand electronics
    july request request result
    review contacted said would
    replace could update review
    response
    new headset nothing update']
```

4.7 PPMI Matrix

To compute the PPMI matrix, we first constructed a word-cluster count matrix by using CountVectorizer, an object of scikit-learn [3] which converts a collection of text documents to a matrix of token counts. Each entry of the word-cluster count matrix specifies the number of times a word appears in a given cluster. We proceed by computing the PPMI matrix by using Equation 3.2.2. Figure 4.3 is an example of word-cluster count matrix for negative reviews.

Figure 4.4 is an example of a PPMI matrix for negative reviews.

4.8 Sort words in the labels

Lastly, we collected all words with non-negative PPMI scores in each cluster. These words are now our label vectors for each cluster. We sort the words in the labels we obtained in descending order based on their PPMI scores. The most informative words that could be used to summarize the clusters have high PPMI scores as compared to the other words in the label vectors. Below is an example of label vectors and their respective PPMI scores.

```
Cluster_labels 11:
  {'german': 5.101337170814387,
   'student': 5.101337170814387,
   'reed': 5.101337170814387,
   'reeds': 5.101337170814387,
```

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
aa	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aaa	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abandon	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
able	4	2	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
absolute	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
absolutely	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
absurd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
abuse	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abysmal	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 4.3: word-cluster count matrix for 20 negative reviews

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
aa	0.000000	1.246067	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0
aaa	0.000000	1.246067	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0
abandon	0.000000	1.246067	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0
able	0.058802	0.000000	0.0	0.0	0.0	0.0	2.487219	0.000000	1.41548	0.0	0.0	2.904113	0.0	0.0	0.0	0.000000	0.0	0.0
absolute	0.000000	1.246067	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0
absolutely	0.000000	0.329776	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	1.479298	0.0	0.0
absurd	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	3.088736	0.0	0.0
abuse	0.869733	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0
abysmal	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	3.471421	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0
ac	0.869733	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.00000	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0

Figure 4.4: PPMI matrix for 20 negative reviews

```
    'clarinets': 5.101337170814387,
    'distance': 5.101337170814387,
    'st': 5.101337170814387,
    'somewhere': 5.101337170814387,
    'unscrew': 5.101337170814387,
    'intermittently': 5.101337170814387,

    'american': 5.101337170814387,
    'counterpart': 5.101337170814387,
    'clarinet': 4.695872062706222,
    'perhaps': 4.408189990254441,
    'excellent': 4.408189990254441,
    'acceptable': 4.408189990254441,
    'clicking': 4.408189990254441,
    'best': 4.002724882146277,
```

```
765     'disappointment': 3.715042809694496,
766     'driver': 3.491899258380286,
767     'screw': 3.155427021759073,
768     'say': 3.0218956291345505,
769     'side': 3.0218956291345505,
770     'size': 3.0218956291345505,
771     'wrong': 2.904112593478167,
772     'brand': 2.904112593478167,
773     'give': 2.904112593478167,
774     'able': 2.904112593478167,
775     'purchase': 2.850045372207891,
776     'though': 2.7987520778203407,
777     'ever': 2.616430521026386,
778     'year': 2.616430521026386,
779     'recommend': 2.616430521026386,
780     'ordered': 2.5363878133528495,
781     'even': 2.462279841199128,
782     'know': 2.393286969712176,
783     'using': 2.393286969712176,
784     'makes': 2.26812382675817,
785     'enough': 2.2109654129182217,
786     'small': 2.2109654129182217,
787     'noise': 2.0102947174560706,
788     'much': 1.8824613459461856,
789     'would': 1.856144037628812,
790     'use': 1.8432406327929043,
791     'fit': 1.8432406327929043,
792     'return': 1.6356012680146599,
793     'used': 1.4904192581701619,
794     'could': 1.4904192581701619,
795     'one': 0.4473768206568631}
```

5. Results and Discussion

5.1 First setting

We implement the algorithm of our approach with Amazon reviews on musical instruments. This data is saved in a tsv file of size 475.2MB and was downloaded from Amazon.com [1]. There are 904,765 reviews in this dataset, after extracting the positive and negative reviews based on star ratings we obtained 106271 negative reviews and 798494 positive reviews. We used an Inspiron 5558 laptop with memory size 8Gib with CPU capacity of 2700 MHz to run our Python codes. Due to the inefficiency of our laptop to run huge dataset, we implemented our algorithm on 70 customer reviews on musical instruments. It took 920.4337 seconds to run the algorithm on 70 positive reviews while it took 164.5895 seconds for 70 negative reviews. We obtained 73 clusters for the negative reviews and 49 clusters for the positive reviews. Each cluster contains an average of 8 reviews. The unique words extracted from the clusters obtained was used in constructing a word-cluster matrix. Each entry of the word-cluster matrix represents the number of occurrence of the ith word in the jth cluster. After computing the PPMI matrix by using Equation 3.2.2, we eliminated words with 0 PPMI scores. The remaining words in the word-cluster matrix were ranked according to their PPMI scores. The PPMI score of a word is a measure of the significance of the word in a cluster. Therefore, by our approach we were able to extract key words from customer reviews on Amazon. See the Appendix for some results.

[1] https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon$_reviews_us_Musical_Instruments_v1_00.tsv.gz$

References

[1] Martin Kütz. *ntroduction to E-Commerce: Combining Business and Information Technology*. 2016.

[2] M MUDAMBI Susan and S David. What makes a helpful online review? a study of customer reviews on amazon. com. *MIS Quarterly*, 34(1):185–200, 2010.

[3] inc. Amazon Web Services. Amazon customer reviews from us customers for the product "watches". Retrieved from https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Watches_v1_00.tsv.gz, Accessed May, 2019.

[4] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. *arXiv preprint arXiv:1707.07045*, 2017.

[5] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

[6] Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 473–483, 2017.

[7] Noah A Smith. Contextual word representations: A contextual introduction. *arXiv preprint arXiv:1902.06006*, 2019.

[8] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[11] Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*, 2017.

[12] W Haas. J.r. firth: Papers in linguistics, 1934–1951. xii, 233 pp., 11 plates. london, etc. *Bulletin of The School of Oriental and African Studies-university of London*, 21, 10 1958. doi: 10.1017/S0041977X00060559.

[13] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[14] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[15] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[17] Leonard Kaufman and Peter J Rousseeuw. Partitioning around medoids (program pam). *Finding groups in data: an introduction to cluster analysis*, pages 68–125, 1990.

[18] Yogita Rani[1] and Harish Rohil. A study of hierarchical clustering algorithm. *ter S & on Te SIT-2*, page 113, 2013.

[19] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[20] Yuzhen Zhao, Xiyu Liu, and Jianhua Qu. The k-medoids clustering algorithm by a class of p system. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, 9(18):5777–5790, 2012.

[21] Raymond T Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge & Data Engineering*, (5):1003–1016, 2002.

[22] Anil K Jain and Richard C Dubes. Algorithms for clustering data. 1988.

[23] Gerlof Bouma. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, pages 31–40, 2009.

[24] James H Martin and Daniel Jurafsky. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson-/Prentice Hall Upper Saddle River, 2009.

[25] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[26] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):19, 2017.

[27] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2(2):169–194, 1998.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct): 2825–2830, 2011.

[29] Robert M Fano and David Hawkins. Transmission of information: A statistical theory of communications. *American Journal of Physics*, 29:793–794, 1961.

APPENDICES

5.2 Python codes for the entire pipeline for our approach

```
# import the needed package for the tasks

import numpy as np
import pandas as pd
import re
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from allennlp.commands.elmo import ElmoEmbedder
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.preprocessing import StandardScaler



import time
import random

then = time.time() #Time before the operations start

#DO YOUR OPERATIONS HERE



# parameters for the pipeline

num_reviews = 5
eps = 0.3
min_samples = 4

# read the tsv file
obs=[]
filename = "/home/salomey/Salomay/ESSAY_PHASE_A/Data_for_project/amazon_revi
file = open(filename, mode='r')
for line in file.readlines():
    obs.append(line.split('\t'))

```

```
926
927
928 # visualizing the data in a dataframe using pandas
929 data=pd.DataFrame(obs,columns=["marketplace ","customer_id" ,"review_id ","p
930
931
932 # drop the first row
933 data = data.drop([0], axis=0)
934 #print("These are the amazon review data for video",data)
935
936 # extracting positive reviews based on star ratings>2
937
938 positive_review= data[data['star_rating']>'2']
939
940
941
942 # extracting the positive review and converting into a list
943 positives = positive_review.review_body.tolist()
944 #print("These are the positive reviews:",positives)
945
946 # extracting negative reviews based on star ratings<=2
947
948 negative_df=data[data['star_rating']<='2']
949
950
951 # converting negative reviews into a list
952 negatives = negative_df.review_body.tolist()
953 #print("These are the negative reviews:",negatives)
954 # pre-processing reviews(remove stop words and non-alphabetic characters)
955
956 def process(review, remove_stopwords=True):
957
958     review_text = re.sub("[^a-zA-Z]"," ", review)
959
960     words = review_text.lower().split()
961
962     if remove_stopwords:
963         stops = set(stopwords.words("english"))
964         words = [w for w in words if not w in stops]
965
966     cleaned_reviews=[]
967
968     for word in words:
969         cleaned_reviews.append(word)
970
```

```
971
972     return(cleaned_reviews)
973
974 cleaned_review_body = []
975 for review in negatives:# change "negatives" to "positives" if you are inter
976     cleaned_review_body.append( " ".join(process(review)))
977
978 tokkenn=[]
979 for r in cleaned_review_body:
980     tokkenn.append(process(r))
981 #print("these are the pre-processed words", tokkenn)
982 # converting the words into vectors using ELMo
983
984 elmo = ElmoEmbedder()
985 word_embeddings = []
986 for reviews in tokkenn[:num_reviews]:
987     word_embeddings.append(elmo.embed_sentence(reviews)[0]) # select the top
988
989 #print("These are the word embeddings for the reviews",word_embeddings)
990 #print(len(word_embeddings))
991 X=(np.concatenate(word_embeddings))
992 #print(type(X))
993 #print("This is the shape of our vector X",np.shape(X))
994
995 ##############################################################################
996
997 # Now we cluster our vector X using sklearn DBSCAN
998
999 X = StandardScaler().fit_transform(np.asarray(X))
1000 plt.rcParams.update({'figure.max_open_warning': 0})
1001
1002
1003 db = DBSCAN(eps, min_samples,algorithm="kd_tree").fit(X)
1004
1005 core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
1006 core_samples_mask[db.core_sample_indices_] = True
1007 labels = db.labels_
1008 #print("these are labels",labels)
1009
1010 #print("These are the core sample indices",db.core_sample_indices_)
1011
1012 n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
1013 n_noise_ = list(labels).count(-1)
1014 #print('Estimated number of clusters: %d' % n_clusters_)
1015 #print('Estimated number of noise points: %d' % n_noise_)
```

```
1016
1017 # Black removed and is used for noise instead.
1018 unique_labels = set(labels)
1019 fig = plt.figure()
1020 colors = [plt.cm.Spectral(each)
1021           for each in np.linspace(0, 1, len(unique_labels))]
1022 for k, col in zip(unique_labels, colors):
1023     if k == -1:
1024         # Black used for noise.
1025         col = [0, 0, 0, 1]
1026
1027     class_member_mask = (labels == k)
1028
1029     xy = X[class_member_mask & core_samples_mask]
1030     plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
1031              markeredgecolor='k', markersize=14)
1032
1033     xy = X[class_member_mask & ~core_samples_mask]
1034     plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
1035              markeredgecolor='k', markersize=6)
1036
1037
1038 plt.title('Estimated number of clusters: %d' % n_clusters_)
1039
1040 fig.savefig('cluster_image.png', bbox_inches='tight')
1041
1042 # extracting the reviews from the clusters
1043
1044 index2 = []
1045 counter = 0
1046 for i in labels:
1047     if i != -1: # remove the noise labels
1048         index2.append(counter)
1049     counter+=1
1050
1051 label2 = []
1052 for j in index2:
1053     k = labels[j]
1054     label2.append(k) # these are the labels for points in each cluster
1055
1056
1057 d = {"Index":index2, "Labels":label2}
1058
1059 df = pd.DataFrame(d)
1060
```

```
s = pd.Series(["Labels"])


# extracting the indices of the labels in the cluster
index_kk = []

for i in range(len(label2)):
    p= df[df['Labels']==i]

    index_kk.append(p.Index.tolist())
#print("index for each label:",index_kk )
index_labels = [x for x in index_kk if x != []]
#print ("These are the indices for each label",index_labels)

# extracting the reviews in each cluster
empty_lst = []
for i in index_labels:
    empty_lst.append([cleaned_review_body[j] for j in i])

#print ("these are the reviews found in each cluster",empty_lst)

for i in range(n_clusters_):
    #print("Cluster %d:" % i,' %s' % empty_lst[i])
    print("Cluster %d:" % i,' %s' % len(empty_lst[i]))

# all reviews in a cluster is now a single token
clust = []
for i in range(len(empty_lst)):
    t= ' '.join(empty_lst[i])
    clust.append(t)
#print("These are the reviews in a cluster",clust)

##########################################################################


# building word-cluster count matrix

def fn_tdm_df(docs, xColNames = None):
    ''' create a term document matrix as pandas DataFrame
    with **kwargs you can pass arguments of CountVectorizer
    if xColNames is given the dataframe gets columns Names'''

    #initialize the  vectorizer
    vectorizer = CountVectorizer()
    x1 = vectorizer.fit_transform(clust)
```

```
    #create dataFrame
    df = pd.DataFrame(x1.toarray().transpose(), index = vectorizer.get_featu

    if xColNames is not None:
        df.columns =  xColNames

    return df

for i in range(len(clust[:5])):

    count_matrix=fn_tdm_df(docs=clust, xColNames =None)
print("This is the count matrix",count_matrix.head())

# building word-cluster PMI matrix

def pmi(df):
    '''
    Calculate the positive pointwise mutal information score for each entry
    https://en.wikipedia.org/wiki/Pointwise_mutual_information
    We use the log( p(y|x)/p(y) ), y being the column, x being the row
    '''
    # Get numpy array from pandas df
    arr = df.as_matrix()

    # p(y|x) probability of each t1 overlap within the row
    row_totals = arr.sum(axis=1).astype(float)
    prob_cols_given_row = (arr.T / row_totals).T

    # p(y) probability of each t1 in the total set
    col_totals = arr.sum(axis=0).astype(float)
    prob_of_cols = col_totals / sum(col_totals)

    # PMI: log( p(y|x) / p(y) )
    # This is the same data, normalized
    ratio = prob_cols_given_row / prob_of_cols
    ratio[ratio==0] = 0.00001
    _pmi = np.log(ratio)
    _pmi[_pmi < 0] = 0

    return _pmi



# PMI matrix into a data frame
```

```
c=pmi(count_matrix)
vectorizer = CountVectorizer()
x1 = vectorizer.fit_transform(clust)
PMI_df = pd.DataFrame(c, index = vectorizer.get_feature_names())

# extracting the labels of each cluster with a positive pmi score into a lis
List = []
for c in range(len(empty_lst)):
    col=PMI_df[c]
    List.append(col[col>0].sort_values(ascending=False).to_dict())
#print('These are the label vectors for each cluster')

for i in range(n_clusters_):
    print("Cluster_labels %d:\n" % i,' %s' % List[i].keys())



now = time.time() #Time after it finished

print("It took: ", now-then, " seconds")
```

5.3 Cluster images for both positive and negative reviews

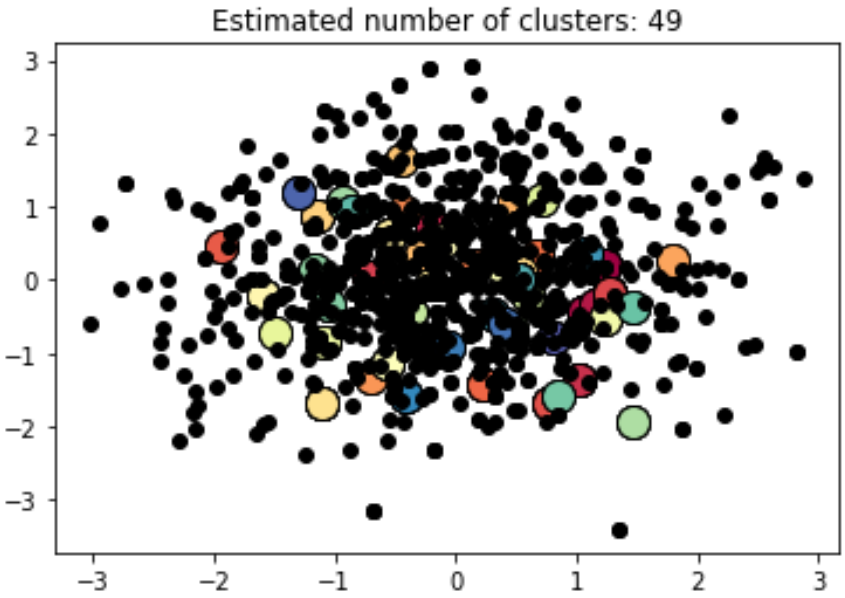

Figure 5.1: Clustering results for 70 positive reviews

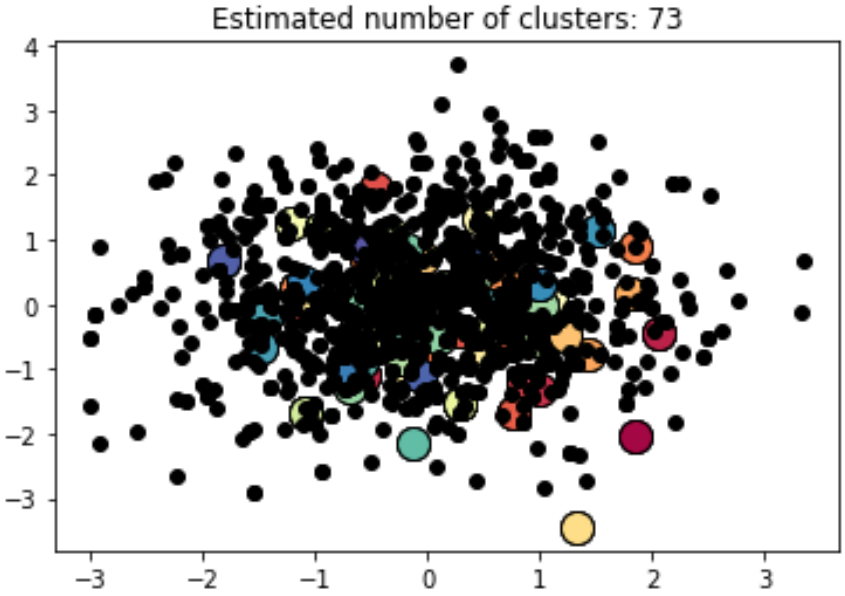

Figure 5.2: Clustering results for 70 negative reviews

5.4 Labels for the first six selected clusters

```
Cluster_labels 0:
  dict_keys(['cast', 'factory', 'phillips', 'originals', 'randomly',
  'rd', 'recorders', 'word', 'speaking', 'substitute', 'ping',
  'grainy', 'backlash', 'possess', 'events', 'welll', 'dean',
  'washed', 'die', 'strips', 'operate', 'turns', 'smoothly', 'channels',
  'pickup', 'mics', 'didnt', 'chinese', 'pain', 'wasting', 'front',
  'dime', 'bother', 'dont', 'mediocre', 'results', 'fell', 'crap',
  'tuners', 'manufacturer', 'hoping', 'built', 'directly', 'almost',
  'returning', 'barely', 'far', 'plastic', 'useless', 'iphone',
  'cheaply', 'cut', 'thin', 'still', 'away', 'stay', 'pick', 'getting',
  'system', 'bridge', 'better', 'violin', 'apart', 'guitars',
  'junk', 'line', 'replace', 'unless', 'properly', 'comes', 'screws',
  'pretty', 'way', 'low', 'think', 'different', 'send', 'cheap',
  'much', 'well', 'buying', 'tune', 'broken', 'hold', 'ok', 'disappointed',
  'made', 'good', 'unit', 'also', 'put', 'work', 'small', 'bad',
  'using', 'mic', 'use', 'used', 'money', 'broke', 'first', 'new',
  'time', 'even', 'two', 'back', 'one', 'guitar', 'got', 'sound', 'quality']
Cluster_labels 1:
  dict_keys(['weights', 'reducer', 'germany', 'steel', 'unplug',
  'designed', 'behringer', 'opened', 'removed', 'plate', 'tuner',
  'produce', 'noise', 'pedal', 'china', 'turned', 'ends', 'inside', 'xlr',
```

```
  'connection', 'install', 'cheaply', 'plug', 'connector', 'last', 'amp', 'v
  'bottom', 'junk', 'happy', 'nothing', 'apart', 'looked', 'line', 'ever', '
  'plastic', 'makes', 'touch', 'less', 'day', 'buying', 'thing', 'recommend'
  'guitar', 'fit', 'bad', 'much', 'mic', 'even', 'made', 'great', 'back', 'd
Cluster_labels 2:
  dict_keys(['advise', 'descriptions', 'starter', 'souvenir',
  'someday', 'solution', 'shoddy', 'seal', 'changed', 'harmonica',
  'chassis', 'promised', 'coolest', 'department', 'distances',
  'astounding', 'nilly', 'drilled', 'msd', 'motu', 'easier', 'mkii',
  'exist', 'frustrate', 'internet', 'industrial', 'graders', 'hindsight',
  'teach', 'pwma', 'harmonicas', 'ticking', 'alvarez', 'arts',
  'willy', 'wireless', 'asking', 'upper', 'air', 'tombo', 'tremolo',
  'thus', 'wt', 'octave', 'tech', 'pyle', 'seriously', 'paper',
  'amplifier', 'ukulele', 'respond', 'neither', 'heats', 'brought',
  'workmanship', 'learning', 'electronic', 'initial', 'sustain',
  'assistance', 'everywhere', 'worst', 'materials', 'student',
  'playable', 'weird', 'night', 'none', 'making', 'external',
  'owned', 'flute', 'three', 'probably', 'unplayable', 'takes',
  'sounding', 'school', 'serious', 'wasted', 'experience', 'difference',
  'products', 'middle', 'beginner', 'drivers', 'musical', 'awful',
  'basically', 'project', 'worthless', 'mm', 'twice', 'hot', 'tell',
  'heard', 'units', 'output', 'amps', 'th', 'manufacturer', 'instrument',
  'band', 'find', 'kit', 'pass', 'pads', 'save', 'returned', 'dollars',
  'ever', 'hard', 'audio', 'days', 'whole', 'note', 'static',
  'sounded', 'instead', 'purchase', 'notes', 'warranty', 'take',
  'full', 'support', 'see', 'metal', 'play', 'away', 'called',
  'ago', 'decent', 'feels', 'years', 'playing', 'maybe', 'sounds',
  'much', 'worked', 'toy', 'speakers', 'brand', 'second', 'played',
  'help', 'jack', 'gave', 'ordered', 'couple', 'junk', 'matter',
  'apart', 'minutes', 'color', 'trying', 'started', 'defective',
  'might', 'pay', 'least', 'like', 'received', 'properly', 'expensive',
  'also', 'sent', 'even', 'lot', 'holes', 'far', 'price', 'stay',
  'year', 'bad', 'looking', 'give', 'go', 'tune', 'money', 'looks',
  'day', 'noise', 'buying', 'thing', 'end', 'never', 'ok', 'disappointed',
  'get', 'bought',
  'something', 'would', 'another', 'really', 'cheap', 'well', 'use',
  'used', 'new', 'good'])
 'waste', 'keep', 'least', 'unit', 'probably', 'expensive', 'properly',
 'however', 'sent', 'problem', 'cable', 'many', 'less', 'high',
 'away', 'different', 'strings', 'amp', 'well', 'working', 'looks',
 'without', 'since', 'guitar', 'first', 'need', 'poor', 'make',
 'recommend', 'two', 'work', 'quality', 'long', 'fine',
 'something',
 'works', 'product', 'also', 'worked', 'amazon', 'sound', 'really',
 'much', 'money', 'would', 'even', 'time', 'great'])
```

```
Cluster_labels 4:
  dict_keys(['ac', 'posting', 'curling', 'smoking', 'glanced',
  'dollar', 'elected', 'prepped', 'needle', 'suppor', 'exit',
  'machines', 'lemon', 'flawlessly', 'knobs', 'fluke', 'standard',
  'samson', 'apparently', 'sink', 'appears', 'arm', 'seconds',
  'guide', 'shipped', 'unplugged', 'saw', 'smoke', 'particular',
  'singing', 'shot', 'shock', 'owned', 'needless', 'nd', 'sad',
  'point', 'start', 'unusable', 'shipping', 'fact', 'items', 'electronics',
  'mine', 'mount', 'turning', 'supposed', 'star', 'machine', 'adapter',
  'tight', 'replacement', 'build', 'device', 'weight', 'update',
  'felt', 'read', 'take', 'things', 'obviously', 'quickly', 'either',
  'help', 'stand', 'seem', 'review', 'anything', 'keep', 'say',
  'turn', 'lot', 'plugged', 'every', 'wrong', 'touch', 'bit',
  'looking', 'sure', 'enough', 'power', 'last', 'give', 'thought',
  'disappointed', 'item', 'something', 'would', 'months', 'problem',
  'worked', 'another', 'fit', 'small', 'bad', 'tried', 'well',
  'one', 'could', 'buy', 'back', 'get', 'quality', 'like', 'good',
  'product'])
Cluster_labels 5:
  dict_keys(['flying', 'picking', 'rattling', 'icicle', 'fit',
  'someting', 'prescreen', 'embarrassed', 'soundtrack', 'special',
  'plig', 'tambourines', 'taping', 'recordings', 'lo', 'pitched',
  'converter', 'warm', 'marimba', 'analog', 'man', 'audition',
  'timpani', 'watching', 'painted', 'buzzy', 'percussionist', 'classical',
  'local', 'instruments', 'recording',
  'lp', 'laptop', 'jingles', 'quite', 'gig', 'meinl', 'everywhere',
  'mid', 'fixed', 'albeit', 'duct', 'foot', 'epiphone', 'chrome',
  'wasting', 'digital', 'tambourine', 'stock', 'regular', 'noisy',
  'provided', 'shop', 'quick', 'sg', 'settings', 'particular',
  'apart', 'lower', 'noticed', 'okay', 'leather', 'awful', 'past',
  'fell', 'project', 'middle', 'device', 'toy', 'higher', 'zero',
  'version', 'song', 'simple', 'large', 'fall', 'show', 'build',
  'guess', 'inside', 'try', 'wish', 'advertised', 'metal', 'needed',
  'things', 'body', 'pieces', 'feel', 'kept', 'buying', 'seems',
  'garbage', 'feedback', 'computer', 'matter', 'couple', 'find',
  'nothing', 'terrible', 'least', 'phone', 'music', 'however',
  'pretty', 'makes', 'plastic', 'like', 'come', 'wrong', 'purchase',
  'old', 'many', 'think', 'bit', 'bad', 'cheap', 'years', 'mic',
  'broken', 'go', 'noise', 'thought', 'first', 'end', 'need',
  'set', 'quality', 'received', 'right', 'something', 'way', 'play',
  'sound', 'fit', 'tried', 'much', 'money', 'product', 'would',
  'could', 'made', 'two', 'bought', 'back', 'guitar', 'got'])
```