Opinion mining using unsupervised machine learning algorithms

Vitalii Duk
Student No. S13002730

A dissertation submitted to
Glyndwr University
in accordance with the requirements of the degree of
M.Sc. in Computer Science

Glyndwr University, Wrexham

August 2014

Abstract

Opinion mining became a significant problem due to extreme development of the E-commerce market. Today, a lot of on-line web platforms are trying to use sentiment analysis of reviews and comments that their customers and users leave. Usage of sentiment analysis algorithms may provide a possibility to classify and organize reviews into more comfortable for end-users form. Background research of the existing work in sentiment analysis area shows that there are still a lot of unsolved problems. One of the main problem is that there are less possible unsupervised methods for opinion mining.

The main purpose of this dissertation is to investigate the usage of clustering algorithms in opinion mining tasks. Dataset with 800 positive and 800 negative previously labeled hotel reviews was used in this work. During the background literature review it was discovered that the problem of sentiment analysis was not studied in pair with unsupervised machine learning approaches, such as clustering. Current research can be divided into two stages: data processing and data analysis. First stage includes text preprocessing and Natural Language Processing phases. In this part software on Python programming language was created. The final results of this subsystem are matrices with text features, which will be used in data analysis. Data analysis is the second stage, which was carried using MATLAB software. This part of work includes performing data standardization, running different variations of clustering algorithms and calculating accuracy for each. Three types of clustering algorithms was investigated during this research, it was Hierarchical clustering, standard K-means clustering and K-means clustering with intelligent initialization. Different distance metrics and methods was studied for each clustering algorithm. The highest accuracy of 85.94% in opinion mining detection was achieved using standard K-means algorithm with correlation as a distance metric and frequency of features >= 10 on standardized dataset.

Finally, there are a list of enhancements that can be applied in future work.

Author's Declaration

I, Vitalii Duk, declare that the work in this dissertation with the title 'Opinion mining using unsupervised machine learning algorithms' was carried out in accordance with the Regulations of Glyndwr University. The work is original except where indicated by special reference in the text and no part of the dissertation has been submitted for any other degree.

Any views expressed in the dissertation are those of the author and in no way represent those of Glyndwr University.

The dissertation has not been presented to any other University for examination either in the United Kingdom or overseas.

Signed: ______________________________

Date: ______________________________

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Opinion mining

Sentiment analysis or opinion mining is the class of methods of content processing, that attempts to detect emotionally colored lexicon and authors emotional opinion in relation to the objects that was determined in the text. Opinion mining is a new research area which is concerned not with the document's topic, but with what opinion it expresses.
The opinion of others has always been an important piece of information, especially when there is a need to make a decision (e.g. buy some goods or etc.). Long time ago, when there was no Internet people asked their friends for advice on better PC or repair center. The World Wide Web *(WWW)* made the process of sharing opinions about everything easier. It also easy to find reviews, recommendations and comments about the dish-washing machine that one may want to buy, or restaurant that somebody is planning to visit this weekend. The process of leaving reviews became more advanced during the last few years. For example, Google Play gives a higher display priority for those reviews that were left on the same to visitor's country and language. *eBay* strongly recommends customers to leaver reviews for each product they have purchased, or at least evaluate the customer's satisfaction using 'stars'. Some websites have a starring system for each review or comment (example: `YouTube.com`).

At the present time opinions mining can be applied for different types of web-platforms.

1. Social networks (Facebook.com, Twitter.com)
2. Customer reviews (TripAdvisor.com, Booking.com, eBay.com, Amazon.com)
3. News (Reuters, BBC)
4. Comments (YouTube.com, Vimeo.com)

Information about the sentiments in the text can be used for the different purposes, for example:

1. Public opinion tracking.
2. Marketing research.
3. Customers opinion analysis.

1.2 Deep opinion mining

Opinion mining for the whole text or document not every time is useful and appropriate. More advanced approaches may analyze each sentence and calculate the average (or other) polarity for entire document or text.

Other part of deeper sentiment analysis is **subjectivity and objectivity identification**.

Sometimes there is a need to detect the opinion or sentiment that is related to some object that was defined in text. This task is the part of **aspect-based sentiment analysis**.

1.3 Aims and objectives

The process of the background research, that was done in Chapter 2, showed that usage of unsupervised algorithms in sentiment analysis is not very popular and is still not investigated enough. How the clustering algorithms and unsupervised approach can be used in sentiment analysis? This dissertation is taking in account such unsupervised machine learning approach as clustering.

Main work will be concentrated on the detection of polarity for whole text review, without subjectivity/objectivity identification. Two possible labels will be used in the review identification process: *'positive'* and *'negative'*.

1.4 Problems

There are a lot of problems that must be solved during the developing of the algorithm that will extract opinions and sentiments from the text.

First of all, most of the machine learning algorithms works with numerical (sometimes binary) data. In this case each text review must be transformed into the appropriate numerical vector. Such vectors will compose a table (matrix), that finally can be used for classification or clustering. There is no standardized way to create a numerical table representation for the text. This research is trying to investigate possible variations of creating numerical representation of the text reviews, using Natural Language Processing *(NLP)*. This subject has many common things with **numerical taxonomy**.

As it was stated earlier, NLP is used in this dissertation for text features extraction. Word order topology is one of the important thing that must be taken in account in the developing process. Each language belongs to one of next word topologies, which represents the positions of subject *(S)*, verb *(V)* and object *(O)* in the text. Possible word topologies with examples are presented in Table 1.1 [?].

#	Variation	Example	Prevalence	Languages
1	**SOV**	She him loves	45%	Ukrainian, Latin, Japanese
2	**SVO**	"She loves him	42%	English, Russian
3	**VSO**	Loves she him	9%	Hebrew, Irish
4	**VOS**	Loves him she	3%	Malagasy, Baure
5	**OVS**	Him loves she	1%	-
6	**OSV**	Him she loves	0%	-

Table 1.1: Possible word topologies

Words that are used to identify sentiment in the text can be domain specific. Those words that express positive opinion in one area, can mean completely different in other area. For example:

1.5 Structure of this dissertation

The aim of this dissertation is to introduce a new, and complete, method that will give a possibility to discern between reviews with positive and negative polarity, without using labeled data.

This method, described in Chapter 3, transforms a raw text review into a data matrix allowing. This approach allows using machine learning algorithms under the unsupervised approach.

The following objectives must be met in order to achieve the above aims:

1. Acquire a dataset of reviews that has been previously used in scientific research, for easy comparison with other standard methods.
2. Run dataset text preprocessing to transform each review to the unified format.
3. Retrieve numerical features from each text review for future use as an input data in machine learning algorithms.
4. Run clustering algorithm with different input parameters: distance metric, feature frequency, data standardization option.
5. Calculate accuracy for each obtained result.
6. Compare final results.

This dissertation is organized in the following way:
Chapter 2 includes description of previous studies in the same and similar area. There is also a description of all algorithms, which will be used during the research.
Chapter 3 describes the algorithm of each step in this research, .
Chapter 4 compares the results, which was obtained during the experiments, including the information about software development process. There is also

a comparison between the unsupervised and supervised machine learning approaches. As there was no previous researches, which do opinion mining with the same dataset, using supervised learning, it was made a decision to conduct them. K-nearest neighbors classification algorithm was selected for comparison.

Chapter 5 makes a summarization of all dissertation and makes a proposal about possibilities for future work.

Chapter 2

Literature review

2.1 Background research

Sentiment analysis became a very popular area of study in last few years. At a present time there are more than 16 000 documents that Google Scholar will return while searching by key phrases like ”sentiment analysis” or ”opinion mining”. Most of the researches are using Natural Language Processing (NLP) in conjunction with Machine Learning (ML) in different variations. Most in opinion mining area has been done using algorithms under the supervised learning framework. Such algorithms tend to give good results, however, they require labeled data to learn from. The quantity of learning data has to be considerable since most research is validated using five or ten fold cross-validation, which means that they have been validated using 80 to 90 percent of labeled data. Whether such algorithms can generalize to the real-world, where it is impossible to obtain 80%, remains to be seen.

The earliest work that describes unsupervised sentimental analysis was written by Turney [22]. It describes basic approaches that can be used for comments classification into two domains: ”recommended” and ”not recommended”. The author gets 74% of accuracy with using adjectives and adverbs as features, on this particular dataset. Using the Part-of-Speech tagger developed algorithm extracts two-word phrases (adjective + noun, verb + adjective, etc.) from the reviews. The next step calculates semantic orientation for each extracted word bi-gram. Semantic orientation *(SO)* is calculated using Pointwise Mutual Information *(PMI)* algorithm:

$$SO\left(phrase\right) = \left(\frac{hits\left(phrase\ NEAR\ \text{“}excellent\text{”}\right)\ hits\left(\text{“}poor\text{”}\right)}{hits\left(phrase\ NEAR\ \text{“}poor\text{”}\right)\ hits\left(\text{“}excellent\text{”}\right)}\right) \tag{2.1}$$

Semantic orientation in this research was calculated based on queries to search engine and counting number of matched results. AltaVista Advanced Search Engine (now part of Yahoo) was used for this purpose. When semantic orientation was calculated for each bi-gram, then algorithm is estimating the average value for whole review. Accuracy in this research is variating from 64% to 84%, depending on domain of review. For example the highest accuracy was achieved with automobile reviews and the lowest with movie reviews. Total size of dataset was 410 reviews, where 170 of them was previously marked as *'recommended'* and the other 240 as *'not recommended'*. But actually author do not use any machine learning approaches in his work, just some NLP functions as Part of Speech *(POS)* tagger.

Brody [8] presents an unsupervised model for on-line reviews that takes attempt to detect aspect (domain) and polarity of the review. This paper mainly takes in account noun-adjective phrases, as the most common way of expressing emotions and sentiment about an aspect.

The fundamental research in opinion mining area was conducted by Pang in 2008 [20]. His article describes the basic principles and definitions of sentiment analysis. There is a good review of all challenges while extracting opinions from the text. The author states that opinion minings is a domain-specific task, where the same words can express different emotions. For example in movie review, phrase *"go read the book!"* indicates negative meaning, but for the book review it seems to be a positive opinion. Based on previous work in this area, Pang suggests that term presence is more relevant than term frequency in opinion mining process. So, the features vector will contain only *"1"* and *"0"* values for each term in entity, rather than including incremented values of its frequency in the text. In machine learning and pattern recognition filed, a *feature vector* is the n-dimensional vector that contains numerical values, which are used to describe some object. As it was discovered, frequency is more important in topic detection, but not in sentimental analysis. Another main point of this research is that not only adjectives are good in sentiment expression, but nouns (e.g., "love") and verbs (e.g., "gem") showed better results in movie-reviews classification. Author also touches the problem of negations and their detection in the text. As an example two sentences with almost the same features can show different sentimental information: "I like this book." and "I don't like this book.". This problem really does not exist in topic/document classification task, where negation are not very significant, only in situation that occurs very infrequently, in cases like "this document is about cars" and "this document is not about cars".

Research that was done by Benamara [6] is trying to declare that usage of adjectives and adverbs combinations in opinion mining is better that usage of adjectives alone. This research is mainly concentrated on adverbs of degree (like "extremely", "hardly", "absolytely") that can describe an intensity of something, other types of adverbs was not investigated and was left for the future work. This type of adverbs was divided into five classes: adverbs of affirmation, adverbs of doubt, strong intensifying adverbs, weak intensifying adverbs, negation and minimizers. Researchers manually created an adverbs score table with 100 most used adverbs, which contains values from 0 to 1 for each adverb. In this table 1 - implies that this adverb completely affirms the adjective to which it belongs, and 0 means that this adverb has no significant impact on the nearest adjective. Also there is a table with adjectives scores, which contains values form -1 to 1 for each adjective. Here -1 represent maximally negative adjectives, and +1 is assigned to maximally positive adjectives. Instead of manual scoring, author use automated OASYS system for adjectives classification and scoring. Main equation of scoring method for each pair "adverb - adjective":

$$f_{VS}(adv,\ adj) \ = \ sc(adj) \ \pm \ (1 \ - \ sc(adj)) \ \times \ sc(adv) \tag{2.2}$$

This method also takes in account variations when there are two related adverbs to the adjective. For example in next context: "hotel was very very good". Usage of three or more adverbs that are related to one adjective was not discovered by author, but if they exist, they can not impact seriously on the final results. By comparing the final results that was based on the dataset with 200 news and 400 blog posts, author states that: 1. Adjectives are more important than adverbs. 2. When one is identifying the strength of the text opinion, score weight of the adverbs must be in range from 30% to 35%.

In [4, 12] authors present an approach for assigning three types of sentiment labels (positive, negative and objective) to each synset (set of synonyms) of WordNet. Term "objective" means that entire word do not have a strong sentiment polarity, actually it is neutral (e.g., "short", "alone"). WordNet is the semantic English dictionary, which includes words that are grouped into sets of synonyms (synsets). All words are distributed between four main groups: adverbs, adjectives, nouns and verbs. Synsets in this dataset are connected with each other using different semantic relations, such as hypernym (breakfast →meal), hyponym (meal →lunch), meronym (window →building), holonym (car →wheel), etc. Semantic relations depend on which part of speech entire word belongs. WordNet is widely used for many researches in computational linguistics and Natural Language Processing [23]. SentiWordNet database was designed to make the process of sentimental analysis easier. To build this database author used semi-supervised machine learning approach. For dataset training purposes Rocchio algorithm and Support Vector Machine learners were used.

Wanzeele in his work [24] is using SentiWordNet dataset for extracting emotions from Twitter micro-blog entries. This research is also comes up with an algorithm that allows calculating emotion distances between two 'tweets' and cluster them based on their emotional value.

Chaovalit and Zhou in their research [9] provide a comparison between supervised and unsupervised classification approaches, based on movie reviews dataset. Dataset that was used in this research was already previously studied in [21]. It includes 1700 movie reviews from IMDB.com website, where 700 of them are positive and 700 are negative. N-gram classifier was used as the supervised approach. On the other side, unsupervised approach in this research is based on Turney's algorithm [22], with the only one difference, where AltaVista Search Engine was replaced by Google Search Engine. Final results of this study shows that supervised approach gave 85% of accuracy and unsupervised approach was at 77%. For future work this paper makes and advise using 'term frequency–inverse document frequency' (TF-IDF) weighting to reduce the number of features.

The work in similar to sentiment analysis area was done by Ott in 2011 [18, 19]. Main aim of this research is to detect deceptive (spam) hotel reviews.

Fundamental research in unsupervised aspect detection was done by Bagheri in 2013 [5]. The main advantage of developed model is that it don't need labeled training data or any additional information, only initial set of aspects. Initial aspects are detecting automatically from the dataset following the next algorithm:

1. Analyze each review in dataset using POS-tagger.
2. Find bigram to four-gram of nouns and adjectives, bigram of determiners and adjectives, bigram to trigram nouns and verb gerunds, unigram to four-gram of nouns.
3. Apply stemming algorithm to select only single form of word.
4. Remove aspects that have stop words ("is", "which", "who").
5.

When the initial aspects are detected, author calculates a metric for each of those aspects. Metric is calculated using authors own A-Score algorithm, which can be defined as:

$$A - Score\left(a\right) = f\left(a\right) * \sum_{i} \log_2 \left(\frac{f\left(a, b_i\right)}{f\left(a\right) * f\left(b_i\right)} * N + 1 \right) \tag{2.3}$$

In this equation a is the current aspect, f_a

2.2 Machine learning

Machine learning is the sub-area of artificial intelligence, which main aim is to create systems that can automatically learn from data, recognize complex patterns and make decision based on this information.
There are few types of machine learning algorithms:

1. **Supervised learning**. Such algorithms are using previously labeled data (examples) as a training datasets.

2. **Unsupervised learning**. The input data for these type of algorithms is not labeled. Main problem is that it is very hard to extract the semantic meaning from the learned model.

3. **Semi-supervised learning**. Is the combination of both supervised and unsupervised approaches.

4. **Active learning**. In these algorithms user can make significant impact on the learning process.

2.3 Natural Language Processing

Natural Language Processing *(NLP)* is used in this research to tokenize review on word basis and assign a part of speech tag to each work (POS-tagger). Word-tokenized text is used in future POS-tagging methods and for negated adjectives lookup. NLP can be divided into two main parts: understanding of natural language and generation (synthesis) of intelligent text. In artificial intelligence field understanding of natural language often is classified as AI-complete task. Basically it means that to solve this problem computers must be on the same level of intelligent development as humans. Computer vision, problems solving or similar can be also categorized as a AI-complete or AI-hard tasks. Part-of-speech tagging is one of the most important and basic tasks in NLP. There are two main groups of tagging algorithms: rule-based and stochastic.

2.4 Data standardization

The main purpose of data standardization (also normalization or feature scaling) is to rescale each feature (column) dimension of the observation set. In some machine learning algorithms (K-means, Hierarchical clustering, Support Vector Machine) where there is a need to calculate a distance between observations, raw data may be not applicable, because features usually have broad range of values. Standardization can be represented by the next equation [17]:

$$y_{iv} = \frac{x_{iv} - a_v}{b_v}, \; i \in I, \; v \in V \tag{2.4}$$

Here a_v is the grand mean or average value (sum of each entity element, which is divided by the total number of elements in this entity). Scale value b_v can be the standard deviation or range. Standard deviation can be calculated as:

$$SD(x) = \sqrt{\frac{\sum\limits_{i=1}^{n}(x_i - \overline{x})}{n}} \tag{2.5}$$

where n is the total number of elements and $\overline{x}$ is mean (average) value: $\overline{x} = \frac{\sum\limits_{i=1}^{n} x_i}{n}$.

In our research we calculate b_v value, using range:

$$b_v = max\,(x_i) - min\,(x_i) \tag{2.6}$$

Usage of range value as the rescaling factor is better than standard deviation. Standard deviation may differ, from the minimum at the peak of unimodal distribution to the maximum at the peak of bimodal distribution, with the same range. While standardizing data with the standard deviation, it will be biased mostly to the shape of unimodal distribution, instead of bimodal, which contribute to clustering most [17]. Basically it means that features will lose their significant values, and it will be harder to indicate clusters.

Standardization is carried out on per-feature basis.

2.5 K-means clustering

K-means is one of the most popular clustering algorithms. Its main aim is to minimize the error (deviation) between each element in the cluster and the center of this cluster (centroid). The main minimization criteria of K-means clustering algorithm has next representation [14]:

$$W\,(S, C) = \sum_{k=1}^{K} \sum_{i \in S_k} d\,(y_i, c_k) = \sum_{k=1}^{K} \sum_{i \in S_k} d\,(y_i, c_k) \tag{2.7}$$

where K is the total number of clusters, S_k is entire obtained cluster, c_k is the centroid (usually, mean value) of each cluster S_k and $d(y_i, c_k)$ is the distance (usually, squared Euclidean distance) between the entity element y_i and its cluster centroid c_k.

The main aim of this clustering alghoritm is to spread I observations into K clusters, minimizing the distance between each element and the center of its cluster.

Algorithm of standard K-means clustering algorithm:

1. Choose initial centroids c_K for each cluster randomly. Create a empty subset of elements in each cluster S_k.
2. Using the distance metric determine clusters S'_k.
3. Check if $S' = S$. If this clusters are equal, stop clustering, in other case, replace S by S'.
4. Calculate new centroids for each cluster, using the mean value.
5. Return to step 2.

In each iteration of this algorithm central elements (centroids) of each cluster are recalculating. It stops when the clusters (and their centroids, respectively) are not changing. Number of clusters must be specified manually. The speed and accuracy of this algorithm highly depend on initial centroids. The main problem with using *K-means* is that the result are not deterministic due to random selection of initial centroids. To get better results one must run *K-means* algorithm several times (iterations).

Basically, K-means was designed for using the squared Euclidean distance as a metric, but other distances can also be used, some of which are explained in the next section.
The standard K-means algorithm is NP-hard task. Its complexity for determined number of clusters (k), dimensions (d) and number of observations (n) is $O(n^{dk+1} \log n)$ [3].

2.6 Hierarchical clustering

Hierarchical clustering is the set of clustering algorithms, visualization of which are based on graphs. The main point in hierarchical clustering is that some

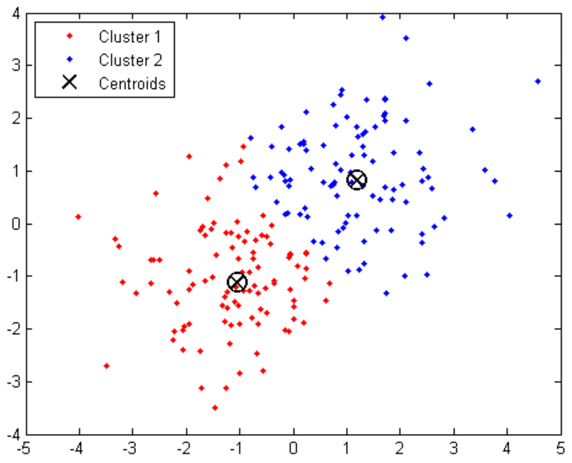


FIGURE 2.1: Example of two clusters and their centroids

set of entities can be described by different level of connectivity. There are two main types of hierarchical clustering algorithms:

1. **Agglomerative**. At initial stage, each entity (observation) is located in its own cluster. After that each pair of clusters will be merged.
2. **Divisive**. At initial stage, all entities (observation) are located in one cluster. They will be split recursively.

Clusters in hierarchical approach can be visualized as dendrogram. There are different methods that allow creating a dendrogram:

1. **Single method**. This method suggests that the distance between two clusters is the *minimum* distance between vector elements: $d(\vec{x}, \vec{y}) = \min \parallel \vec{x} - \vec{y} \parallel$.
2. **Complete method**. This method is similar to *single*, the difference is only that distance between clusters will be the maximum value of an element distances: $d(\vec{x}, \vec{y}) = \max \parallel \vec{x} - \vec{y} \parallel$.
3. **Weighted Pair-Group Method using Centroids** or *WPGMC*. In this method distance between two clusters will be calculated as the Euclidean distance between their weighted centroids.
4. **Unweighted Pair-Group Method using Centroids** or *UPGMC*.

5. **Unweighted Pair-Group Method using Arithmetic averages** or *UPGMA*.

6. **Weighted Pair-Group Method using Arithmetic averages** or *WPGMA*.

7. **Ward's Method**.

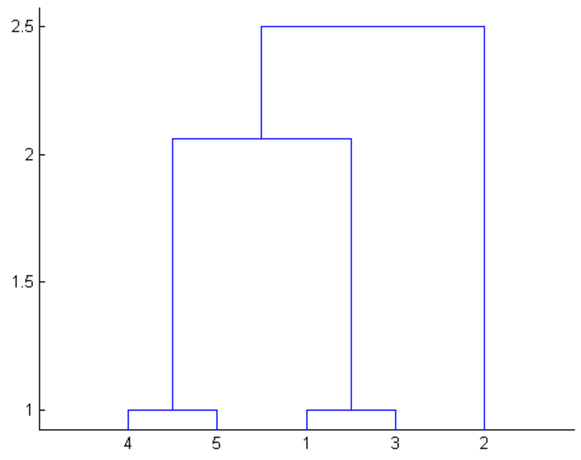


Figure 2.2: Hierarchical clustering visualization

Figure 2.2 shows the dendrogram, where indices on the horizontal axis represents entities from the original dataset, and the height of link between each object is the distance value.

Complexity of agglomerative clustering algorithm is $O\left(n^3\right)$, where n is the number of total entities. On the other hand, divisive approach in the worst case will give $O\left(2^n\right)$ complexity. Each method in Hierarchical clustering can also use different distance metrics, as well as K-means.

2.6.1 Ward's method

For the agglomerative approach Ward distance between two clusters can be defined as [17]:

$$dw(S_{w1}, S_{w2}) = \frac{N_{w1}N_{w2}}{N_{w1} + N_{w2}} d(c_{w1}, c_{w2}) \tag{2.8}$$

where $d(c_{w1}, c_{w2})$ is the Squared Euclidean distance between centroids c_{w1} and c_{w2}. S_{w1} and S_{w2} are two clusters, and N_{w1}, N_{w2} are the cardinalities of those clusters.

Ward's algorithm for the agglomerative case contains next steps:

1. Initialization. Each element must be places in separate cluster, where the centroid is itself.

2. Updating clusters. Find two closest to each other clusters S_{w1}, S_{w2} and merge them into one new cluster $S_{w1 \cup w2}$.

3. Updating distance. Put new cluster $S_{w1 \cup w2}$ into a list of clusters, and remove two old clusters S_{w1} and S_{w2}. Recalculate Ward distances between cluster $S_{w1 \cup w2}$ and other clusters.

4. Repeat iteration. If the number of maximum clusters is $>$ than 1, then go to step 2. In other case, finish clustering algorithm and output the cluster merger.

2.7 Distance metrics

This section is describing distance metrics that can be used in K-means or Hierarchical clustering to measure distance between observation entities. Distance metric is very important and can significantly impact on the final results of clustering.

2.7.1 Euclidean distance

Euclidean distance or metric that can be ordinary measured using the ruler in metric space.

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.9}$$

Here, x and y are two vectors in Euclidean space. Squared Euclidean distance is using basically to give more weight for those objects for which the distance is calculating. Equation that describes Squared Euclidean distance is almost the same to previous one, just without the square root:

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{2.10}$$

2.7.2 City block distance

In different literature this metric can also be referred as **Manhattan distance** or length. Basically, city block distance can be calculated using the sum of differences between line segment projections onto the coordinate axes.

$$d(\vec{x},\vec{y}) = \sum_{i=1}^{n} \mid x_i - y_i \mid \tag{2.11}$$

Figure 2.3 shows how Euclidean and City block (Manhattan) distance between two elements are look like in a Cartesian coordinate system.

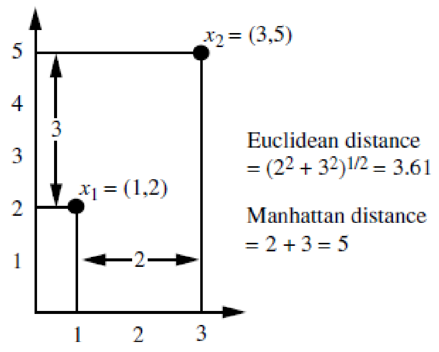


FIGURE 2.3: Visualization of the Euclidean and City block distances

2.7.3 Chebychev distance

Chebychev distance is also called as **maximum metric**, **chessboard distance** or L_∞ **metric**. This distance is calculating as the maximum absolute difference between the elements of two vectors:

$$d(x,y) = \max_{i=1,2\ldots n} \mid x_i - y_i \mid \tag{2.12}$$

2.7.4 Mahalanobis distance

Mahalanobis distance normally is using to measure the distance between the set of observations x and their mean value μ. The dissimilarity between two random vectors can be defined as:

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T S^{-1}(\vec{x}-\vec{y})} \tag{2.13}$$

In this equation, S^{-1} is an invertible covariance matrix, and T refers to the transposed matrix. In case when S is the identity matrix, Mahalanobis distance will be the same as Euclidean distance.

2.7.5 Cosine as a distance

Cosine similarity can be used as a distance metric, to measure how similar are two vectors (observations). Cosine similarity can be described by the next equation [13]:

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\parallel \vec{a} \parallel \parallel \vec{b} \parallel} \tag{2.14}$$

In this equation $\vec{a}\cdot\vec{b}$ is a dot product of two vectors. Dot product can be represented in next form:

$$\vec{a}\cdot\vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n \tag{2.15}$$

where n is the dimension of two vectors (must be the same).
$\parallel \vec{a} \parallel$ and $\parallel \vec{b} \parallel$ is the magnitude or length of the entire vector in the Euclidean space:

$$\parallel \vec{a} \parallel = \sqrt{a_1^2 + a_2^2 + ... + a_n^2} \tag{2.16}$$

If the angle between two vectors are 90° (two vectors are orthogonal), cosine function will be equal 0. If the vectors are completely the same (equal), the angle between them will be equal to 0°, in this case cosine value will be 1. Basically in statistical analysis and clustering is used value that is equal to: $1 - \cos(A, B)$, where A and B are two observation vectors, that are comparing.

2.7.6 Correlation as a distance

Covariance coefficient [10]:

$$cov(x, y) = \frac{1}{n}\sum_{i\in I}(x_i - \overline{x})(y_i - \overline{y}) \tag{2.17}$$

where $\overline{x}$ and $\overline{y}$ are the mean values of x and y.

Sometimes in literature the correlation coefficient is also called a *Pearson product moment correlation coefficient.* It can be defined as covariance coefficient divided by the standard deviation:

$$r(x, y) = \frac{cov(x, y)}{SD(x)SD(y)} \tag{2.18}$$

Usually in clustering algorithms the value, which is equal to $1 - r(x, y)$ is used as a distance metric.

2.7.7 Minkowski distance

Minkowski distance or metric is a general representation of Euclidean and City block *(Manhattan)* metrics. Some times it also refereed to as L_p *norm.*

$$d_p(x, y) = \left(\sum_{v=1}^{n} \mid x_v - y_v \mid^p \right)^{\frac{1}{p}} = \sqrt[p]{\sum_{v=1}^{n} \mid x_v - y_v \mid^p} \tag{2.19}$$

If the *p=2* it will be an Euclidean metric, in case when *p=1* this equation will take form of City block metric. Also, when the $p \to \infty$ it is the Chebyshev metric.

2.7.8 Hamming distance

Hamming distance is using to calculate how different are two vectors of binary data. Usually this distance is using to calculate the difference between two words or strings with the same length.

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{n} \mid x_i - y_i \mid \tag{2.20}$$

Example of Hamming distance calculation:

$$d(10111, 11001) = 3 \tag{2.21}$$

Hamming distance is a particular case of Minkowski distance, where $p = 1$. The main problem of this metric, is that it can be applied only for a binary data.

Levenshtein distance is more advanced metric that can be used for similar purposes.

2.8 Improving K-means initialization algorithm

As it was stated earlier, basic K-means algorithm choose initial centroids randomly, so it is not very efficient, because must be run a lot of times to achieve the best results. There is a possibility to use initial centroids that was determined by other, better and more intelligent, algorithm.

Algorithm of intelligent K-means [10]:

1. Select the item c_t from observations T, whose distance is the farthest from the origin, which is equal to 0.
2. Run K-Means algorithm, to create the cluster S with two initial centroids: c_T and 0.
3. If the number of entities in the cluster around c_t is the same or higher to user-defined value, add c_t to the final list of centroids C.
4. Remove those elements, which belongs to cluster c_t from the main array of observations.
5. If the observations array still contains elements, then go to step 1.
6. Run K-Means algorithm with detected centroids C. The number of centroids in the list will be equal to number of clusters.

Usage of initialization centroids can give a sustainability to K-means algorithm, so there will be no possibility to run numerous iteration (replications) to and select the best result. [17]

2.9 Evaluation of clustering

Evaluation or validation of clustering results is the process of finding the similarity between two clusters. Evaluation process can show the quality of clustering approach. There are two types of evaluation [15]:

1. Internal evaluation. In this method two results of clustering by the same algorithm are used.

2. External evaluation. Result of clustering is compared with some gold standard, such as per-classified data, that was created by experts.

2.9.1 Confusion matrix

Usage of confusion matrix is one of the external methods of quality measurements in clustering. Basically, it the matrix with size KxK, where K is the number of clusters that was determined.

Actual values	**Predicted values**	
	Positive	Negative
Positive	354	446
Negative	41	759

TABLE 2.1: Example of confusion matrix with two clusters

Table 2.1 shows how the confusion matrix will look like for two clusters: *'positive'* and *'negative'*. Each of this clusters contain 800 items ($total = 1600$). It shows that clustering algorithm recognized only 354 *'positive'* items as *'positive'* and other as 446 as *'negative'*. But the algorithm is better in *'negative'* items recognition: only 41 *'negative'* items was incorrectly labeled as *'positive'* and other 759 items was identified correctly. Correct label assignments are located on the diagonal of the matrix, and errors will can be visually inspected on both sides of diagonal. So, if the total number of elements is 1600 (100%), and number of correct guesses is $(354 + 446 = 1113)$, accuracy in percents can be easily calculated, using next formula: $(1113 * 100/1600 = 69.56)$. Accuracy of this algorithm is 69.56%.

There are also some other methods of external evaluation, such as ***Rand measure, F-measure, Jaccard index, Fowlkes–Mallows index*** and ***Mutual information***.

2.9.2 Silhouette coefficient

It is the method of internal clustering evaluation. The silhouette coefficient compares the average distance to entities in the same cluster with the average distance to the entities in other cluster. This method is also can be used for determining the number of clusters in dataset. Silhouette coefficient S_i for $i-th$ point can be calculated using next formula:

$$S_i = \frac{(a_i - b_i)}{\max(a_i, b_i)} \tag{2.22}$$

where a_i is the average distance from point i to other point in the same cluster, and b_i is the minimum distance from point i to point is other cluster.
The silhouette value can be in range from -1 to 1. If this value is high for item i, it means that this item was well-recognized in same cluster, but not good in other cluster. If the average value of silhouette coefficients is high, then the clustering algorithm works well.

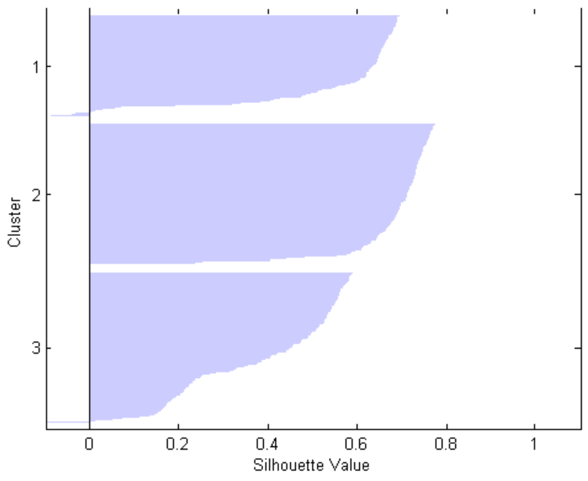


FIGURE 2.4: Example of silhouette plot in MATLAB

Figure 2.4 shows the graphical representation of three clusters. Second cluster has high silhouette value, which basically means that this cluster was determined well. In the third cluster average silhouette value is not very high, and both first and third clusters contain some points with negative values, which means that entities in this cluster was not separated in the efficient way.

Davies–Bouldin index and ***Dunn index*** are also the other methods that can be used for internal clustering evaluation.

2.10 Principal component analysis

Principal component analysis *(PCA)* is one of the main methods that allows reducing the number of dimensions, but at the same time loose the fewest amount of information [1]. The main procedure of PCA, include next steps:

1. Standardize (normalize) input data, to put it into the same range.
2. Calculate k orthonormal vectors (principal components), that provide a basis for the input data.
3. Sort orthonormal vectors in order of decreasing of their significance.
4. After the principal components sorting, the data size can be reduced by removing the weaker components. Using the most significant (stronger) components it will be possible to reconstruct a good approximation of the original data.

2.11 Summary

This chapter provides an overview of previous studies that were done in sentiment analysis area in section 2.1. Following sections gives an explanation of all algorithms and approaches that was used in this research. There are a description of K-means and Hierarchical clustering algorithms in section 2.5 and 2.6. These algorithms are the main elements on which this dissertation is based. There are also an information about the evaluation principles, that will be used in Chapter 4.

Chapter 3

Methodology

During the process of background research it was discovered that there is no existing researches that combines sentiment analysis with the usage of clustering algorithms. Usage of supervised machine learning algorithms can give a higher accuracy, but they require a labeled data for training purposes. Obtaining a labeled data is very hard and sometimes impossible process. As it was stated in Chapter 1, sentiment analysis mostly depends on a specific domain, so supervised algorithms will need a training dataset for each domain. Clustering algorithms, which are completely unsupervised, can be more appropriate for sentiment analysis.

3.1 Programming language and development environment

3.1.1 Python

Python is very progressive and widely used programming language. A lot of scientists are using Python because of several reasons [16]:

1. **Open-source and free**. One do not need to pay for Python usage. It is also possible to commit updates, because Python sources is open, or 'fork' *(create own code repository version based on existing code of the project)* Python for own usage.

2. **Big amount of scientific libraries**. (*SciPy, NumPy, NLTK, Pandas, IPython, etc.*)

3. **Simple to read and understand other code**. Python syntax is highly readable even for people who are not familiar with Python.

4. **Not very complicated**. Python supports Object-Oriented paradigm, but it is not so complicated as Java, where one need to use OOP everywhere and create a class even to print *"Hello, world!"*.

5. **Powerful support of strings and arrays manipulations**. Even in basic functionality there are a lot of methods that can make string manipulation process easier.

6. **Big community of users**. Python users community include people, which works in different areas of science and production. They can always make some advises in solving different problems.

7. **Python is widely used**. Such famous world-wide companies as *Google, Yahoo, NASA, Red Hat, IBM* and others use Python in their projects [11].

Support of such library for Natural Language Processing as *NLTK* is very significant advantage, because for example MATLAB does not have any tools that cn be used for Part-of-Speech tagging. In this research, I have made use of various Python libraries, such as *SciPy, NumPy* and *NLTK*.

SciPy is the scientific library that has a big number of sub packages, like statistical functions, clustering algorithms (Hierarchical, K-means, Vector quantization), input/output module (allows loading and saving data in different formats, e.g., MATLAB .mat files), spatial functions (nearest neighbors, distance functions), etc. The main drawback of this library is that there support of clustering algorithms is currently limited.

NumPy or *Numerical Python* library is the Python extension, that allows easily create N-dimensional arrays (matrices) and manipulate them. Among other, this library also has a lot of functions for linear algebra, Application Programming Interface (API) for integration with C/C++. Arrays in NumPy can have different data types, so it can be easily connected with other databases. NumPy can be used to give Python the functionality, which can be comparable to MATLAB. Part of NumPy library was written in C language, to speed-up the mathematical algorithms.

Natural Language Toolkit *(NLTK)* is the set of Python libraries for symbolic and statistical Natural Language Processing (NLP). It has a big number of tools that can be used for computer linguistic purposes. NLTK provides more than 50 built-in corpora and trained models. Examples of some resources (the full list of available data can be found on-line: `http://www.nltk.org/nltk_data/`):

1. **Brown corpus**. One of the most famous text collection, that contains more than 1 million words and 500 text in English. It was created in 1960s.
2. **Sentiment polarity dataset**. Includes a trained model and raw data of movie reviews, created by Pang [21].
3. **SentiWordNet**. Is an extension of WordNet sysets, that adds sentiment scores. [12]
4. **Stop-words corpus**. A list of words that do not have a significant role, such as: *on, the, is.*
5. **Treebank Part of Speech Tagger**. *HMM* and *Maximum entropy* approaches.

NLTK include not only datasets for English language, there are also Spanish, German, Indian and other word-lists and grammar files.

As the Integrated Developing Environment *(IDE)* for Python was used *PyCharm*, it is free, open-source and powerful tool for programming in Python.

3.1.2 MATLAB

MATLAB was used in this work for performing data standardization, running clustering algorithms and clusters evaluation. It has huge number of toolboxes that can be used in different ares, from aerospace to statistics. In this research only Statistic Toolbox is required. Statistics toolbox provide statistical and machine learning algorithms for data processing. MATLAB was used because

K-means and Hierarchical clustering algorithms are fully implemented there. For data standardization and obtaining initial centroids for K-means algorithm with intelligent initialization was used own functions written in MATLAB.

3.2 Dataset

The dataset, which was used in this work, was originally obtained by Ott [18, 19], and was used in his research for deceptive and spam reviews detection. It contains 800 positive reviews and the same amount of negative reviews, so the total size number of entities is 1600. Also, each set of positive and negative reviews contains 400 truthful and 400 deceptive reviews, but this information will be not included in this research. Always guessing approach will give an accuracy in 50%. Due to probability theory: $P(A) = \frac{m}{n} = \frac{1}{2} = 0.5$, where m = total number of possible events, and n = total number of successful events (clusters), which in this work is equal 2 (”positive” and ”negative”).

3.3 Features

In this research three types of features were used: *adjectives, adverbs and negated adjectives.*

Adjective is the part of speech that can give more information about the object. It has a significant role in every language, because basically they 'describe the word'. Examples of adjectives are: *beautiful, large, wealthy, new, excellent, poor.* Adjectives can determine positive and negative emotions.

An adverb is a word that defines the state or action of the object and indicates its quality. Adverbs can answers such questions as *'when?', 'where?', 'how?', 'in what way?'.* Examples of adverbs: *often, usually, incredibly, extremely, very, quiet.*

Some times in text adjectives can be negated by such words as *'not'* or similar. In this case the opinion can be significantly different. For example in the sentence *”The service in the hotel was not good!”*, there is one adjective - *”good”*, but the general opinion in the whole review is negative, because negation phrase *”was not”* is changing this adjective meaning. After the process of POS-tagging and defining adjectives, next step is to detect those adjectives *(JJ)* that were negated.

1. Detect *"wasn't", "isn't", "aren't", "don't", "not", "won't"*, just before the adjective *(JJ)*. Example: *"The room wasn't clean!", "This hotel isn't bad.", "Waitress aren't good in their job!"*.

2. Detect when the word *"couldn't", "shouldn't", "doesn't", "didn't", "wouldn't"* stands before the adjective on position *"adjective - 2"*. Example: *"This trip couldn't be awesome."*.

3. Detect when the first word in phrase (bi-gram) *"could not", "should not", "does not", "did not", "would not", "is not", "was not"* stands before adjective on position *"adjective - 3"* and the second word *("not")* stands on position *"adjective - 2"*. Example: *"The light was not bright!"*.

There is no reason to check negative words that can stand after the adjective, because they can not impact on the word behind them. For sure this approach does not cover all possible variations, only the basic ones.

As the result we get three matrices:

1. Matrix with adjectives features for each review. Size $M * Aj$, where $M =$ number of reviews, $Aj =$ total number of adjectives that was found in whole dataset.

2. Matrix with adverbs features for each review. Size $M * Av$, where $M =$ total number of reviews in dataset, $Av =$ total number of adverbs that was found in whole dataset.

3. Matrix with negated adjectives. Size of this matrix is $M * Ajn$, where $M =$ total number of reviews in dataset, $Ajn =$ total number of negated adjectives that was found in whole dataset.

On the final step all three matrices (arrays) will be horizontally concatenated. The final array with all features has size $M * N$, where $N = Aj + Av + Ajn$, and M is the constant number of reviews. The sum of each column in the final array is $>= 1$.

3.4 Clustering algorithms

Three types of clustering algorithms will be studied in this dissertation:

1. Standard K-means clustering.
2. K-means clustering with intelligent initialization algorithm.
3. Hierarchical clustering.

Both K-means (intelligent and standard) and Hierarchical clustering algorithms will be examined with next distance metrics:

1. Euclidean.
2. Cosine.
3. City block (Manhattan).
4. Correlation.

But, Hierarchical clustering will include examining next extra distance metrics:

1. Squared Euclidean.
2. Chebychev.
3. Mahalanobis.
4. Minkowski.
5. Jaccard.
6. Spearman.
7. Hamming.

All distance metrics were explained in Chapter 2.

3.5 Summary

This chapter covers the methodology that was used in this dissertation. It describes the numerical feature, which will be used as the main data for in clustering algorithms. There is also a brief description of the dataset, which include hotel reviews. This chapter also include an information about programming languages that will be used during the research and their benefits. Finally, parameters, which will be used to run clustering algorithms are described at the end.

Chapter 4

Implementation and Findings

4.1 Implementation

In the current work next software/versions was used:

1. MATLAB version 8.3.0.532 (R2014a) was used in this research.
2. PyCharm Community Edition version 3.4.1.
3. Python version 2.7.6 (x64).

Algorithm to retrieve text features, written in Python:

1. Create empty arrays which will contain adjectives features, adverbs features, and negated adjectives features separately.
2. Read each file with a review. Remove white spaces from the left and right side of each element.
3. Tokenize each review on word basis.
4. Using word-tokenized comments find adjectives and adverbs with POS-tagger.
5. For each adjective and adverb count the frequency of its appearance in the review.
6. Write the frequencies into the arrays.

7. Using word-tokenized comments find negated adjectives, count their frequencies and save it into the array.

8. Save all obtained information into compatible *.mat* files, using *SciPy* library.

Algorithm of MATLAB part of program:

1. Read data performed by Python program.

2. Create sub-datasets with different feature frequencies (<= 1, 10, 20, 50, 100).

3. Create standardized datasets for each feature frequency, in addition to existing datasets.

4. Run standard K-means clustering algorithm with different distance metrics (Squared Euclidean, City block, Cosine, Correlation) and number of replications = 100.

5. Run K-means clustering with intelligent centroids selection and different distance metrics (Squared Euclidean, City block, Cosine, Correlation).

6. Run Hierarchical clustering algorithm with different methods (Average, Centroid, Complete, Median, Single, Ward, Weighted) and distance metrics for each method (Euclidean, Squared Euclidean, City block, Minkowski, Chebychev, Mahalanobis, Cosine, Correlation, Spearman, Hamming, Jaccard).

7. Save all gathered data into *.mat* files.

Final data array with features for each review, that will be used in clustering algorithms, is shown in table 4.1.

	adj_1	...	adj_k	adv_1	...	adv_v	neg_adj_1	...	neg_adj_m
Review ID#1	2	...	0	3	...	1	0	...	1
...	...	...	...	...	...	...	...	...	...
Review ID#n	1	...	2	0	...	2	1	...	0

TABLE 4.1: Final array with features

4.2 Code review

4.2.1 Python code

According to the algorithm, which was described in previous section, Python program for text features extraction was developed. This subsection will include brief description of the valuable parts of source code.

Next part of code works with raw review text. It uses Python NLTK library to tokenize text and recognize adjectives and adverbs using the POS-tagger, as it was described in previous section.

```python
text = file_pointer.read().rstrip()
...
text_tokens = word_tokenize(text)
text_pos = pos_tag(text_tokens)
for (tx, tk) in text_pos:
  tx = str.lower(tx).strip("'")
    if tk[0:2] == 'RB':
      add_pos(0, tx, review_index, text_tokens)
    elif tk[0:2] == 'JJ':
      add_pos(1, tx, review_index, text_tokens)
```

Here, ***pos_tag()*** is the method, which returns bi-grams in form: *word/part-of-speech*. **RB** tag refers to **adverbs**, and **JJ** identifies **adjectives**. Other labels are not used in this program. ***pos_tag()*** function use **Treebank text corpus** and *Maximum Entropy Classifier* for POS-tagging [7].

If the adjective of adverb was found function ***add_pos()*** is called. This function is used to build the feature matrices for each review.

Internal structure of ***add_pos()*** can be described using next algorithm:

1. Check if entire *adjective, adverb or negated adjective* (feature) is in list of previously detected features.
2. If this feature was found, get its index and increment the frequency of this feature for the entire review.
3. In case when the feature appears first time in this dataset, add it to the list and go to step 2.

```
def add_pos(pos, value, review_id, tokens):
if pos == 0:
  try:
    txi = adverbs.index(value)
  except ValueError:
    txi = -1
  if txi < 0:
    adverbs.append(value)
    adverbs_result = append(adverbs_result,
    zeros([len(adverbs_result), 1]), axis=1)
    adverbs_result[review_id][adverbs.index(value)] = 1
  else:
    ...
    adverbs_result[review_id][adverbs.index(value)] += 1
  ...
else:
  try:
    txi = adjectives.index(value)
  except ValueError:
    txi = -1
  if txi < 0:
    adjectives.append(value)
    ...
    adjectives_result = append(adjectives_result,
    zeros([len(adjectives_result), 1]),
      axis=1)
    adjectives_result[review_id][adjectives.index(value)] = 1
    negative_adjectives_result =             append(
   negative_adjectives_result,
      zeros([len(negative_adjectives_result),
      1]), axis=1)
    negative_adjectives_result[review_id][adjectives.index(value)]
    = \
      find_negative(tokens, value)
  else:
    ...
    adjectives_result[review_id][adjectives.index(value)] += 1
    negative_adjectives_result[review_id][adjectives.index(value)]
    = \
      find_negative(tokens, value)
```

Function that detects negated adjectives is called from ***add_pos()*** method. It takes two parameters: tokenized by-word text of the review and the adjective, negated form of which will be searched.

```
def find_negative(tokens, adj):
  negatives = 0
  for (i, v) in enumerate(tokens):
    if str.lower(v) == str.lower(adj):
      if i >= 1:
        word = str.lower(tokens[i - 1])
        if word in reject[0]:
          negatives += 1
        continue
      if i >= 2:
        word = str.lower(tokens[i - 2])
        if word in reject[1]:
          negatives += 1
        continue
      if i >= 3:
        word = str.lower(tokens[i - 3])
        word += ' ' + str.lower(tokens[i - 2])
        if word in reject[2]:
          negatives += 1
        continue
  return negatives
```

The algorithm of this function was described in Chapter 3, section 3.3. Basically, this method looks for three patterns in which adjective can appear. This function returns the number of entire adjective negations in the review.

4.2.2 MATLAB code

As it was stated before, MATLAB was used for running clustering algorithms. MATLAB build-in functions from the Statistical toolbox was used during this process.

Automation scripts were written to make the process of gathering results easier.

This scripts for running standard K-means has the next structure:

```
load('datasets\accuracy.mat');
load('datasets\dataset.mat');
Datasets = [0, 10, 20, 50, 100];
```

```
Distances = ['sqEuclidean'; 'cityblock  '; 'cosine     '; 'correlation'];
Distances = cellstr(Distances);
Total = size(Datasets, 2) * size(Distances, 1) * 2;
A_kmeans = cell(1, Total);
Errors = cell(1, Total);
I = 1;
for dataset = 1:1:size(Datasets, 2)
  s_dataset = Datasets(dataset);
  for distance = 1:1:size(Distances, 1)
    s_distance = char(Distances(distance));
    try
      eval(sprintf('U = kmeans(D_F%i_Std, 2, ''replicates'', 100, ''distance'', ''%s'');', ...
        s_dataset, s_distance));
      A_kmeans{I} = {U, 'Std', s_dataset, s_distance, ...
        CheckLabels(U, accuracy)};
      catch err
        Errors{I} = {ex};
      end
      try
        eval(sprintf('U = kmeans(D_F%i_NoStd, 2, ''replicates'', 100, ''distance'', ''%s'');', ...
          s_dataset, s_distance));
        A_kmeans{I} = {U, 'noStd', s_dataset, s_distance, ...
          CheckLabels(U, accuracy)};
      catch err
        Errors{I} = {err};
      end
      I = I + 2;
    end
end
save('results/kmeans_all_nostd_all.mat', 'A_kmeans', 'Errors');
```

This script includes next steps:

1. Load datasets and matrix of labels for accuracy check.
2. Create nested ***for*** loops to work with every dataset (different feature frequency) and distance metric. Script for Hierarchical clustering includes the third *for-loop* to work with different *methods* (e.g Ward, centroid, etc.).

3. Use ***try/catch*** construction for better errors handling. When the error occurs it will be written into special array for future examination. There are two ***try/catch*** combinations, for standardized and not standardized datasets.

4. Using the combination of ***eval()*** and ***sprintf()*** functions to run K-means (or other) algorithm. This functions were used to dynamically change the name of dataset (features frequency and standardization option) matrix and such parameters as *'distance'* and method (for Hierarchical clustering).

5. After finishing all loops, save the final variables into **.mat** file. Cell arrays was used to store the results. Each row in such array contains next cells: U (final labels), standardization option (Yes/No), distance metric (e.g. Euclidean), feature frequency, accuracy.

Accuracy was calculated using the confusion matrices, that was described in Chapter 2, section 2.9. Automation scripts for running intelligent K-means and Hierarchical clustering have the same structure.

4.3 Summary

The research was successfully conducted according to the algorithm that was described in previous section.

Maximum number of features was 2656, in the full dataset. Results in Table 4.3 shows that there are not a big number of adjectives, adverbs and negated adjectives that has a high frequency. For example, number of features decreased by 7 times, between full features matrix (frequency >= 1) and matrix with features frequency >= 10.

The other thing that was discovered (Table 4.2), is that there are not a lot of negated adjectives in the whole dataset (only 119), but the number of adjectives is very big (1930).

#	**Group of features**	**Number of features**
1	Adjectives	1930
2	Adverbs	607
3	Negated adjectives	119
	Total:	**2656**

Table 4.2: Number of features by type

#	**Features frequency**	**Number of features**
1	>= 1	2656
2	>= 10	360
3	>= 20	232
4	>= 50	114
5	>= 100	64

Table 4.3: Number of features in different datasets

Table 4.4 show most frequent adjectives, adverbs and negated adjectives that was extracted from the text, and used in clustering analysis. Negated adjectives was explained in Chapter 3(section 3.3). For example adjective *'worth'* appears in context *'was not worth'* or similar.

#	**Adjectives**		**Adverbs**		**Negated adjectives**	
	word	***count***	***word***	***count***	***word***	***count***
1	great	700	very	1212	worth	41
2	nice	393	just	511	sure	15
3	good	316	even	416	clean	15
4	other	310	again	390	good	13
5	clean	291	here	372	able	9
6	next	254	only	368	bad	9
7	comfortable	236	there	342	available	9
8	best	209	also	336	great	8
9	first	206	really	310	helpful	8
10	more	191	never	286	free	7

Table 4.4: Most frequent features in dataset

As it was stated before, the standard K-means is a non-deterministic algorithm. Experiments using the standard K-means algorithm with 300 and even 1000 replications produced results very similar to those with 100 replications. This shows that the results are indeed stable and relevant.

TABLE 4.5: Results of K-means clustering

#	**Std.**	**Feat. frequency**	**Distance metric**	**Accuracy**
1	No	>= 1	squared Euclidean	69.5625
2	No	>= 1	City block	66.5000
3	No	>= 1	Cosine	75.3125
4	**No**	**>= 1**	**Correlation**	**75.4375**
5	Yes	>= 1	squared Euclidean	69.5625
6	Yes	>= 1	City block	71.6250
7	Yes	>= 1	Cosine	75.2500
8	**Yes**	**>= 1**	**Correlation**	**80.7500**
9	No	>= 10	squared Euclidean	69.6250
10	No	>= 10	City block	66.5000
11	No	>= 10	Cosine	75.3125
12	**No**	**>= 10**	**Correlation**	**76.0000**
13	Yes	>= 10	squared Euclidean	69.3125
14	Yes	>= 10	City block	69.312500
15	Yes	>= 10	Cosine	73.0000
16	**Yes**	**>= 10**	**Correlation**	**85.9375**
17	No	>= 20	squared Euclidean	69.6250
18	No	>= 20	City block	66.5000
19	No	>= 20	Cosine	75.2500
20	**No**	**>= 20**	**Correlation**	**76.2500**
21	Yes	>= 20	squared Euclidean	68.5000
22	Yes	>= 20	City block	71.6250
23	Yes	>= 20	Cosine	72.0000
24	**Yes**	**>= 20**	**Correlation**	**84.9375**
25	No	>= 50	squared Euclidean	69.4375
26	No	>= 50	City block	66.5000
27	No	>= 50	Cosine	-
28	No	>= 50	Correlation	-

Continued on next page

Table 4.5 – *Continued from previous page*

#	Std.	Feat. frequency	Distance metric	Accuracy
29	Yes	>= 50	squared Euclidean	68.8125
30	Yes	>= 50	City block	71.6250
31	Yes	>= 50	Cosine	71.0000
32	**Yes**	**>= 50**	**Correlation**	**82.9375**
33	No	>= 100	squared Euclidean	69.8750
34	No	>= 100	City block	66.5000
35	No	>= 100	Cosine	-
36	No	>= 100	Correlation	-
37	Yes	>= 100	squared Euclidean	66.3125
38	Yes	>= 100	City block	71.6250
39	Yes	>= 100	Cosine	67.1875
40	**Yes**	**>= 100**	**Correlation**	**81.0625**

Results shows (Table 4.5 and 4.6) that using standard and intelligent K-means algorithm the best result was achieved using **correlation** as a distance metric, **standardized** dataset and **features frequency** >= **10 and 20**.
In Table 4.5 with standard K-means results lines 27,28,35,36 are empty due to errors that occurred during the calculation process. These errors occur because in datasets with features frequency 50 and 100 on not standardized data, some points have small relative magnitudes, so the results of distance computation is meaningless (almost zero).
Lines 29,31,37,39 in Table 4.6 are also empty, because the intelligent algorithm of initial centroids selection cannot finish calculation process. This problem occurs also only on not standardized datasets with features frequency 50 and 100.

Table 4.6: Results of Intelligent K-means clustering

#	Std.	Feat. frequency	Distance metric	Accuracy
1	No	>= 0	squared Euclidean	69.5625
2	Yes	>= 0	squared Euclidean	68.3750
3	No	>= 0	City block	50.0625
4	Yes	>= 0	City block	50.1875
5	No	>= 0	Cosine	75.4375

Continued on next page

Table 4.6 – *Continued from previous page*

#	Std.	Feat. frequency	Distance metric	Accuracy
6	Yes	>= 0	Cosine	75.2500
7	No	>= 0	Correlation	75.4375
8	**Yes**	**>= 0**	**Correlation**	**80.1250**
9	No	>= 10	squared Euclidean	69.6250
10	Yes	>= 10	squared Euclidean	69.3125
11	No	>= 10	City block	63.8750
12	Yes	>= 10	City block	50.0625
13	No	>= 10	Cosine	75.3125
14	Yes	>= 10	Cosine	70.6250
15	No	>= 10	Correlation	76.0000
16	**Yes**	**>= 10**	**Correlation**	**85.6250**
17	No	>= 20	squared Euclidean	69.6250
18	Yes	>= 20	squared Euclidean	68.1250
19	No	>= 20	City block	64.5625
20	Yes	>= 20	City block	50.0625
21	No	>= 20	Cosine	75.2500
22	Yes	>= 20	Cosine	70.2500
23	**No**	**>= 20**	**Correlation**	**81.3125**
24	**Yes**	**>= 20**	**Correlation**	**84.8125**
25	No	>= 50	squared Euclidean	69.4375
26	Yes	>= 50	squared Euclidean	67.5625
27	No	>= 50	City block	65.7500
28	Yes	>= 50	City block	52.9375
29	No	>= 50	Cosine	-
30	Yes	>= 50	Cosine	70.7500
31	No	>= 50	Correlation	-
32	**Yes**	**>= 50**	**Correlation**	**82.9375**
33	No	>= 100	squared Euclidean	70.0000
34	Yes	>= 100	squared Euclidean	51.9375
35	No	>= 100	City block	66.5000
36	Yes	>= 100	City block	51.6875
37	No	>= 100	Cosine	-

Continued on next page

Table 4.6 – *Continued from previous page*

#	**Std.**	**Feat. frequency**	**Distance metric**	**Accuracy**
38	Yes	>= 100	Cosine	67.1875
39	No	>= 100	Correlation	-
40	Yes	>= 100	Correlation	52.0625

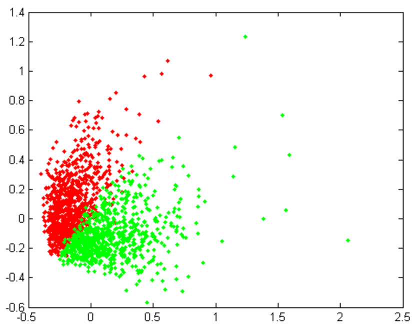
Results of Hierarchical clustering contains 770 rows in total. Only significant results with accuracy that is higher than **55%** was included into the Table 1. **Ward's method** in pair with **City block distance metric** showed the highest accuracy **(79.93%)** in labeling entities into the clusters.
Unsuccessful hierarchical approaches is described in Table 2.

Table 4.7 show that usage of features frequencies >= 10 and 20 can give the best results. All three clustering algorithms gave the highest accuracy with those frequencies. This information can be used in future research, because it can be stated that the best frequency is somewhere in range from 5 to 25.

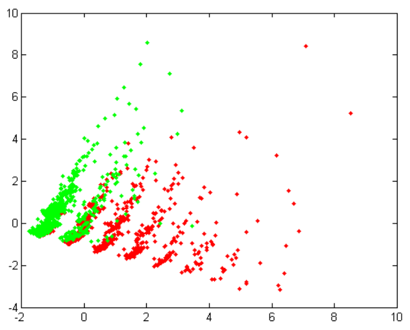
Algorithm	Freq.		Maximum	Minimum	Mean	Mode
K-means	>= *1*	*Std*	80.75%	69.56%	74.30%	69.56%
		Not Std.	80.75%	66.50%	71.70%	66.50%
	>= *10*	*Std*	**85.94%**	69.31%	**74.39%**	69.31%
		Not Std.	76.00%	66.50%	71.86%	66.50%
	>= *20*	*Std*	**84.94%**	68.50%	**74.27%**	68.50%
		Not Std.	76.25%	66.50%	71.91%	66.50%
	>= *50*	*Std*	82.94%	68.81%	73.59%	68.81%
		Not Std.	69.44%	66.50%	67.97%	66.50%
	>= *100*	*Std*	81.06%	66.31%	71.55%	66.31%
		Not Std.	69.88%	66.50%	68.19%	66.50%
iK-means	>= *1*	*Std*	80.13%	50.19%	68.48%	50.19%
		Not Std.	80.13%	50.06%	67.63%	75.44%
	>= *10*	*Std*	**85.63%**	50.06%	**68.91%**	50.06%
		Not Std.	76.00%	63.88%	71.20%	63.88%
	>= *20*	*Std*	**84.81%**	50.06%	**68.31%**	50.06%
		Not Std.	81.31%	64.56%	72.69%	64.56%
	>= *50*	*Std*	82.94%	52.94%	68.55%	52.94%
		Not Std.	-	-	-	-
	>= *100*	*Std*	67.19%	51.69%	55.72%	51.69%
		Not Std.	-	-	-	-
Hierarchical	>= *1*	*Std*	70.75%	55.81%	66.22%	67.44%
		Not Std.	70.75%	58.69%	68.14%	70.00%
	>= *10*	*Std*	72.88%	55.69%	65.88%	66.75%
		Not Std.	**79.94%**	58.38%	**68.13%**	73.75%
	>= *20*	*Std*	73.94%	63.50%	68.28%	64.63%
		Not Std.	**75.25%**	56.19%	**64.80%**	68.69%
	>= *50*	*Std*	72.19%	55.25%	66.09%	66.00%
		Not Std.	70.31%	55.94%	62.87%	55.94%
	>= *100*	*Std*	72.88%	55.38%	61.66%	58.00%
		Not Std.	72.19%	59.06%	65.81%	69.31%

TABLE 4.7: Statistical analysis of accuracy depending on features frequency

Principal component analysis *(PCA)* method, which is described in Chapter 2, was used to reduce number of dimensions (features). It gives a possibility to plot the distribution of entities between clusters. MATLAB code was used for plotting next graphs.

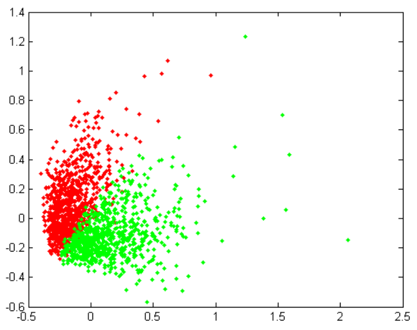


(A) F. frequency >= 10;
Standardized dataset;
Correlation distance;
Accuracy = 85.93%

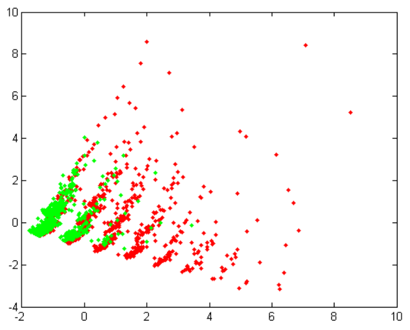


(B) F. frequency >= 20;
Not Standardized dataset;
Correlation distance;
Accuracy = 76.25%

FIGURE 4.1: Visualization of the best K-means clustering results

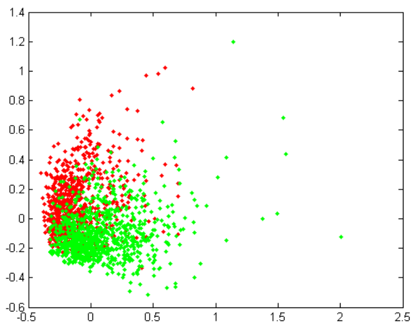


(A) F. frequency >= 10;
Standardized dataset;
Correlation distance;
Accuracy = 85.62%

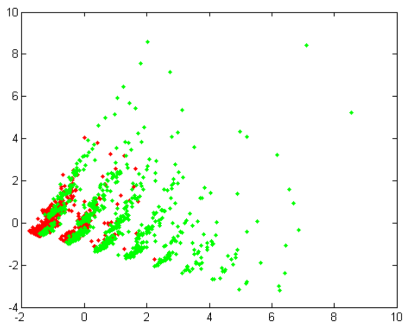


(B) F. frequency >= 20;
Not Standardized dataset;
Correlation distance;
Accuracy = 81.31%

FIGURE 4.2: Visualization of the best intelligent K-means clustering results



(A) F. frequency >= 20;
Standardized dataset;
Ward method;
City block distance;
Accuracy = 73.93%



(B) F. frequency >= 10;
Not Standardized dataset;
Ward method;
City block distance;
Accuracy = 79.93%

FIGURE 4.3: Visualization of the best Hierarchical clustering results

As it can be examined from figures, standardized dataset groups entities in more convenient way for clustering. Elements in not standardized dataset are not clearly spitted into two clusters. Actually, both datasets, standardized and not standardized does not have clearly separated clusters.
The other thing that can be concluded from obtained results is that intelligent K-means (with initialization) can give almost the same results as the standard K-means algorithm. The main advantage of using K-means with intelligent initialization, is that this algorithm will give deterministic results on every iteration (replication), so there is no need to run huge number of replications, as it was in standard K-means.

Table 4.8 shows results of applying statistical functions to obtained accuracy, to find out is it better to use standardized or not standardized datasets for each clustering algorithm. It can be concluded that K-means (standard and intelligent version) clustering can achieve better accuracy with standardized data. On the other hand, Hierarchal clustering showed better results with not standardized data. In this statistical analysis only results of Hierarchical clustering with accuracy >= 55% was included, to give more weight for the values, and exclude not significant methods and distance metrics.

Table 4.8: Statistical analysis of obtained accuracy

Stat. function	**K-means**		**iK-means**		**Hierarchical**	
	Std	*Not Std*	*Std*	*Not Std*	*Std*	*Not Std*
Maximum	**85.93%**	76.25%	**85.62%**	81.31%	73.93%	**79.93%**
Minimum	66.31%	66.50%	50.06%	50.06%	55.25%	55.93%
Mean	73.61%	70.88%	65.99%	70.50%	65.38%	66.0625%
Mode	71.62%	66.50%	50.06%	69.62%	66.75%	55.93%

In Table 4.8 **mode** is the value that appears most frequently in the set of accuracies. **Mean** is the average value of frequencies.

4.4 Results comparing

There are no existing researches, which are using the same dataset and same of different (supervised) machine learning approach. The was made a decision to conduct run some supervised algorithms using the same dataset and features, to compare the accuracy.

K-nearest Neighbors *(KNN)* classification algorithm was used as the mostly used supervised machine learning algorithm.
The KNN algorithm has next steps:

1. Training phase. Sort the features and label vectors.
2. Classification phase. Each entity in testing set will be classified by assigning most frequent label among the k (manually defined by user) closest elements from training set. Basically, distance metric between points is the Euclidean distance, but other distances, such as *correlation* or *cosine* can be applied.

K-nearest neighbors is one of the most simple classification algorithms in machine learning.

Leave-one-out cross-validation *(LOOCV)* algorithm was used to check the generalization of *KNN* classification. It was used because it gives a deterministic result.

One element will be taken from the main dataset. For the training purposes will be used a dataset without this one element. This one element will be classified, based on previously trained model. Such algorithm will be applied for each element in the whole dataset. For example, if the dataset contains 1600 elements, 1600 iterations will be conducted, where 1599 elements will be used for training purposes and only 1 in the classification. Result of each classification iteration will be saved, and on the final stage the average accuracy of the entire classification algorithm will be calculated.

k-fold cross-validation is the other possible cross-validation algorithm. This algorithm is more appropriate for a huge dataset. The main purpose is to separate whole dataset into k smaller subsets (e.g. 5 or 10), train the classifier on $k - 1$ subsets and test the classification algorithm on 1 subset. It is very similar to *leave-one-out* cross-validation.

Table 4.9 shows the final results of *KNN* classification of the hotel reviews. Only standardized datasets was used, because they showed better results in clustering.

MATLAB code that was used for automated run of *KNN* classifiers is presented below. It is very similar to the code, which was used for clustering. There are also three nested **for-loops** to run ***knnclassify()*** function with different distance metrics, datasets (features frequencies) and the number of **K** nearest neighbors (special parameter) for *KNN* classifier.

```
clc;
clear;
load('datasets\accuracy.mat');
load('datasets\dataset.mat');
Datasets = [0, 10, 20, 50, 100];
Ks = [1, 5, 10, 20, 50, 100, 200, 500];
Distances = ['euclidean'; 'correlation'];
Distances = cellstr(Distances);
St = 'Std';
for dataset = 1:1:size(Datasets, 2)
  s_dataset = Datasets(dataset);
  for distance = 1:1:size(Distances, 1)
    s_distance = char(Distances(distance));
    for ks = 1:1:size(Ks, 2)
      s_ks = Ks(ks);
      clear D;
      eval(sprintf('D = D_F%i_%s;', s_dataset, St));
      t = size(D, 1);
      for i = 1:t
      DataLearn = D;
      DataLearn(i,:)=[];
      y=accuracy;
      y(i,:)=[];
      new_y = knnclassify(D(i,:), DataLearn, y, s_ks, s_distance);
      x(i,1) = new_y==accuracy(i,:);
    end
    ac = (sum(x)/t)*100;
    fprintf('%s; Features=%i; Distance=%s; Accuracy=%.2f; KNN=%i;\
   n', St, s_dataset, s_distance, ac, s_ks);
    end
  end
end
```

F. frequency	**Distance**	**Number of the nearest neighbors**				
		1	*10*	*50*	***100***	*500*
>= **1**	*Correlation*	70.38%	79.44%	85.69%	87.13%	86.50%
	Euclidean	62.63%	58.56%	50.50%	50.19%	50.00%
>= **10**	*Correlation*	72.88%	80.44%	86.38%	**88.38%**	86.69%
	Euclidean	66.50%	67.38%	59.69%	56.75%	50.13%
>= **20**	*Correlation*	73.63%	81.50%	85.31%	**87.44%**	86.00%
	Euclidean	69.81%	71.00%	65.50%	66.81%	53.50%
>= **50**	*Correlation*	71.25%	81.31%	82.38%	83.88%	85.81%
	Euclidean	69.50%	75.63%	78.00%	78.63%	67.69%
>= **100**	*Correlation*	71.06%	78.69%	81.38%	81.63%	83.31%
	Euclidean	70.13%	76.56%	80.38%	81.00%	79.56%

Table 4.9: Results of K nearest neighbors classification

Actually algorithm of K-nearest neighbors is not the best possible variation of supervised approach in machine learning. Future work might include investigation of such algorithm as **Support Vector Machine** *(SVM)*, Naive Bayes and others.

Obtained results shows that supervised approach can give better results. The difference between the best result that was achieved using unsupervised (85.94%) and supervised (88.38% accuracy) algorithms are not very significant. On the other hand, one must take in account that almost all dataset in supervised approach was used for model training purposes. Such situation is almost impossible in real world situation.

Chapter 5

Conclusion

5.1 Summary

This dissertation introduces an unsupervised machine learning approach for the sentiment analysis of hotel reviews. Using clustering algorithms during this work it was achieved 85.93% of accuracy while processing hotel reviews. While examining the obtained results it can be concluded that Algorithm of intelligent K-means initialization that was described in Chapter 2 can give a deterministic valuable results. Maximum accuracy that was achieved using this algorithm is 85.62%. Standard K-means algorithm with the same parameters *(distance metric = correlation, features frequency ¿= 10, standardized data)* gave an accuracy 85.93%. Achieved results proves that intelligent K-means algorithm can give almost the same results to standard K-means. The main advantage of intelligent K-means is that there is no need to run a big number of iterations to achieve the best results.
Final results, which was achieved using clustering algorithms were compared with the results of supervised algorithms (K nearest neighbors). Supervised approach showed better results, but there was no significant difference between those two approaches (difference = 3%). For sure for more, in future researches this comparison must be extended by adding more supervised algorithms.

Achieving 100% accuracy in opinion mining is very complicated task. There might be a problem, is even the software that will give nearly 100% will be created, for sure humans will disagree with some percent of this results.

5.2 Future work

This work can be improved in the future. There are some suggestions for other researchers.

5.2.1 Multi-language support

Current research is using dataset with a hotel reviews in English only. Multi-language opinion mining support is very significant for those web-resources that has international audience. On such websites as `TripAdvisor.com`, `Booking.com`, `Play.Google.com` users leave reviews in different languages all over the world about the same hotel or application, and each review, despite the language on which it was written, has the same relevance.

5.2.2 On-line algorithm

To create a scalable working prototype Apache Hadoop can be used as the basic platform. **Apache Hadoop** is an open-source set of libraries that was developed for processing of huge amount of data. It was designed for distributed computing, and can be easily scaled to numerous machines. It is very important to study how the unsupervised opinion mining approach can be used in real software architecture.

5.2.3 Expanding the number of features

The future researches can be conducted by adding more features to the current approach.
Based on research that was conducted by Benamara (2007) [6], which states that using adverb-adjective bigrams is more useful, and can help to detect emotions better, this approach might be used in future work. As an example current feature matrix can be replaced or expanded by adding frequency of adverb-adjective bi-grams. The algorithm could tokenize each review into word unigrams, then do POS tagging and re-tokenize tagged review into bigrams, where adverb-adjective pairs will be matched. To check the best variant, it one

can use both methods: expanding existing feature matrix or use just adjective-adverb bigrams to compare the accuracy.

Adding punctuation marks, such as fullstops, question and exclamation marks cat be useful, because people use them to make their sentences and phrases more expressive. There was no information about using this features in previous studies, so one can prove or disprove usefulness of these features.

It can be also investigated in the future what type of features are more important and which of them are less important. For example clustering can be performed for 'adjectives', 'adverbs' and 'negated adjectives' features separately. It will show the significance of each type. After that it will be possible to use different variations of features frequencies, depending on their weight. For example: one data set will include observations with adjectives frequency = 10, adverbs frequency = 20 and negated adjectives frequency = 1.

5.2.4 Algorithms

Not all possible variations of clustering algorithms were investigated in the current work.

Future work can start with usage of different data standardization algorithms. For example range scaling feature that was used in this work can be compared with standard deviation (was described in Chapter 3).

More distance metrics can be used in standard and intelligent version of K-means clustering. For example, such distance metrics was not studied: Chebychev, Mahalanobis, Euclidean (only Squared Euclidean distance was used in K-means clustering), Spearman, Hamming, Jaccard, Minkowski.

Hidden Markov Model *(HMM)* is another possible unsupervised approach that can be studied in the future. Currently there are no work that uses *HMM* to do opinion mining. But it is widely used in other subareas of machine learning, like part-of-speech taggers.

Gaussian Mixture Models *(GMM)* are often used in data clustering. This algorithm can be included in future work, because some times it can be more appropriate than K-means clustering, especially when the clusters have different sizes and correlations.

5.2.5 Expanding datasets

Current research includes only hotel reviews, so the described algorithm must be tested on other datasets, to prove the results in other domains. For example there is a dataset with IMDB film reviews, that was also classified into two clusters: positive and negative, it is available on-line: that was studied in [9, 21].

5.2.6 Other improvements

As this research works with real-world dataset, which contains reviews that was written by humans, it should be taken into account that each written text, especially on-line, will have a grammar mistakes and misspellings. It was noticed that any previous work do not include information about how significant such mistakes might be. The possible variant for future work is to create or implement existing algorithm for mistakes correction.

5.2.7 Neutral opinion

This work does not take into account possibility of neutral reviews. Future reviews might conduct research with three clusters to include "positive", "negative" and "neutral" opinions.

This report was created in LaTeX.

References

[1] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[2] Shigeo Abe. *Support vector machines for pattern classification.* Springer, 2010.

[3] David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153. ACM, 2006.

[4] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 2200–2204, 2010.

[5] Ayoub Bagheri, Mohamad Saraee, and Franciska de Jong. An unsupervised aspect detection model for sentiment analysis of reviews. In *Natural Language Processing and Information Systems*, pages 140–151. Springer, 2013.

[6] Farah Benamara, Carmine Cesarano, Antonio Picariello, Diego Reforgiato Recupero, and Venkatramana S Subrahmanian. Sentiment analysis: Adjectives and adverbs are better than adjectives alone. In *ICWSM*, 2007.

[7] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python.* " O'Reilly Media, Inc.", 2009.

[8] Samuel Brody and Noemie Elhadad. An unsupervised aspect-sentiment model for online reviews. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 804–812. Association for Computational Linguistics, 2010.

[9] Pimwadee Chaovalit and Lina Zhou. Movie review mining: A comparison between supervised and unsupervised classification approaches. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 112c–112c. IEEE, 2005.

[10] Mark Ming-Tso Chiang and Boris Mirkin. Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads. *Journal of classification*, 27(1):3–40, 2010.

[11] Python Community. Organizations that use python. https://wiki.python.org/moin/OrganizationsUsingPython, 2013.

[12] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of LREC*, volume 6, pages 417–422, 2006.

[13] Graham L Giller. The statistical properties of random bitstreams and the sampling distribution of cosine similarity. *Available at SSRN 2167044*, 2012.

[14] Jiawei Han and Micheline Kamber. *Data Mining, Southeast Asia Edition: Concepts and Techniques*. Morgan kaufmann, 2012.

[15] Mehmed Kantardzic. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.

[16] Mark Lutz. *Learning python*. " O'Reilly Media, Inc.", 2013.

[17] Boris Mirkin. *Clustering: a data recovery approach*. CRC Press, 2012.

[18] Myle Ott, Claire Cardie, and Jeffrey T Hancock. Negative deceptive opinion spam. In *HLT-NAACL*, pages 497–501, 2013.

[19] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 309–319. Association for Computational Linguistics, 2011.

[20] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[21] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[22] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[23] Princeton University. About wordnet. `http://wordnet.princeton.edu/`, 2010.

[24] Ruben Van Wanzeele, Katja Verbeeck, Annemie Vorstermans, Tom Tourwe, and Elena Tsiporkova. Extracting emotions out of twitter's microblogs. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, 2011.

Appendix A. Source code

LISTING 1: Python code: DataFeatures.py

```python
from numpy import zeros, append, save, column_stack, array
from scipy.io import savemat
from nltk import word_tokenize, pos_tag
import os

# Create arrays
adjectives = []
adverbs = []
adjectives_count = []
adverbs_count = []
adjectives_result = zeros((0, 0), dtype='u4')
adverbs_result = zeros((0, 0), dtype='u4')
negative_adjectives_result = zeros((0, 0), dtype='u4')

# emotions: 0 - negative, 1 - positive (+1 for MATLAB)
# fairness: 0 - deceptive, 1 - truthful (+1 for MATLAB)
# ['review text', positive/negative, truthful/deceptive]
reviews = zeros((0, 3))

# Dictionary with negating words for adjectives
reject = [
    # (?) JJ
    ["wasnt", "isnt", "arent", "dont", "not", "wont", "nt"],
    # (?) (X) JJ
    ["couldnt", "shouldnt", "doesnt", "didnt", "wouldnt"],
    # (?) (not) (X) JJ
    ["could not", "should not", "does not", "did not", "would not"
    ,
     "is not", "was not"]
]

# Read dataset directories recursively
```

```
def read_folders(path):
    emotions = {0: 'negative_polarity', 1: 'positive_polarity'}
    fairness = {0: 'deceptive_from_MTurk', 1: 'truthful_from_Web'}
    for e in range(0, 2, 1):
        for f in range(0, 2, 1):
            temp_path = path + emotions[e] + "\\" + fairness[f] + "\\"
            for p in range(1, 6):
                fin_path = temp_path + "fold" + p.__str__() + "\\"
                read_reviews(fin_path, e + 1, f + 1)


# Read all reviews from entire folder
def read_reviews(folder, em, fair):
    global reviews, adjectives_result, adverbs_result, \
        negative_adjectives_result
    for f in os.listdir(folder):
        full_file = os.path.join(folder, f)
        if os.path.isfile(full_file):
            file_pointer = open(full_file, 'r')
            text = file_pointer.read().rstrip()
            reviews = append(reviews, [[text, em, fair]], axis=0)

            review_index = len(reviews) - 1
            if review_index < 0:
                review_index = 0

            adjectives_result = append(adjectives_result,
                                       [zeros(adjectives_result.shape[1])],
                                       axis=0)
            negative_adjectives_result = append(negative_adjectives_result,
                                                [zeros(negative_adjectives_result.shape[1])],
                                                axis=0)
            adverbs_result = append(adverbs_result, [zeros(adverbs_result.shape[1])],
                                    axis=0)

            text_tokens = word_tokenize(text)
            text_pos = pos_tag(text_tokens)
            for (tx, tk) in text_pos:
```

```python
                tx = str.lower(tx).strip("'")
                if tk[0:2] == 'RB':
                    add_pos(0, tx, review_index, text_tokens)
                elif tk[0:2] == 'JJ':
                    add_pos(1, tx, review_index, text_tokens)


# Add feature to the array
# 0 - adverb; 1 - adjective
def add_pos(pos, value, review_id, tokens):
    global adverbs, adjectives, adverbs_result, adjectives_result, \
        negative_adjectives_result
    if pos == 0:
        try:
            txi = adverbs.index(value)
        except ValueError:
            txi = -1

        if txi < 0:
            adverbs.append(value)
            adverbs_count.append(1)
            adverbs_result = append(adverbs_result,
                                    zeros([len(adverbs_result), 1]),
                                    axis=1)
            adverbs_result[review_id][adverbs.index(value)] = 1
        else:
            adverbs_count[adverbs.index(value)] += 1
            adverbs_result[review_id][adverbs.index(value)] += 1

        return adverbs.index(value)
    else:
        try:
            txi = adjectives.index(value)
        except ValueError:
            txi = -1

        if txi < 0:
            adjectives.append(value)
            adjectives_count.append(1)
            adjectives_result = append(adjectives_result,
                                       zeros([len(adjectives_result), 1]),
                                       axis=1)
            adjectives_result[review_id][adjectives.index(value)] = 1
            negative_adjectives_result = append(negative_adjectives_result,
                                                zeros([len(negative_adjectives_result),
                                                       1]), axis=1)
            negative_adjectives_result[review_id][adjectives.index(value)] = \
                find_negative(tokens, value)
        else:
            adjectives_count[adjectives.index(value)] += 1
            adjectives_result[review_id][adjectives.index(value)] += 1
            negative_adjectives_result[review_id][adjectives.index(value)] = \
                find_negative(tokens, value)


# Find negated adjectives in text
def find_negative(tokens, adj):
    negatives = 0
    for (i, v) in enumerate(tokens):
        if str.lower(v) == str.lower(adj):
            if i >= 1:
                word = str.lower(tokens[i - 1]).strip("'").strip('"').\
                    replace("'", "").replace('"', '')
                if word in reject[0]:
                    negatives += 1
                    continue
            if i >= 2:
                word = str.lower(tokens[i - 2]).strip("'").strip('"').\
                    replace("'", "").replace('"', '')
                if word in reject[1]:
                    negatives += 1
                    continue
            if i >= 3:
                word = str.lower(tokens[i - 3]).strip("'").strip('"').\
                    replace("'", "").replace('"', '')
                word += ' ' + str.lower(tokens[i - 2]).strip("'").\
                    strip('"').replace("'", "").replace('"', '')
                if word in reject[2]:
                    negatives += 1
                    continue
    return negatives


# Save obtained results in different formats
def save_data(temp, arrays_to_file, arrays_to_separate_matfiles):
    if arrays_to_file == 1:
        d = 'npy/' + temp
        save(d + 'reviews.npy', reviews)
        save(d + 'adverbs.npy', adverbs)
        save(d + 'adjectives.npy', adjectives)
        save(d + 'adverbs_result.npy', adverbs_result)
        save(d + 'adjectives_result.npy', adjectives_result)
        save(d + 'negative_adjectives_result.npy',
             negative_adjectives_result)

    if arrays_to_separate_matfiles == 1:
        d = 'mats/' + temp
        savemat(d + 'reviews.mat',
                mdict={'reviews': reviews_modified})
        savemat(d + 'adverbs_result.mat',
                mdict={'adverbs_result': adverbs_result})
        savemat(d + 'adjectives_result.mat',
                mdict={'adjectives_result': adjectives_result})
        savemat(d + 'negative_adjectives_result.mat',
                mdict={'negative_adjectives_result': negative_adjectives_result})

# ## MAIN RUN SECTION

read_folders("E:\\op_spam_v1.4\\")

reviews_modified = zeros((0, 1), dtype='u4')
reviews_modified = reviews[0:, 1:2].astype('u4')

```

```
### SAVE RESULTS
save_data('', 1, 1)
```

Listing 2: MATLAB code: RunKmeans.m

```
clc;
clear;

load('datasets\accuracy.mat');
load('datasets\dataset.mat');


Datasets = [0, 10, 20, 50, 100];


Distances = ['sqEuclidean'; 'cityblock  '; 'cosine     '; '
    correlation'];
Distances = cellstr(Distances);


Total = size(Datasets, 2) * size(Distances, 1) * 2;

A_kmeans = cell(1, Total);
Errors = cell(1, Total);
I = 1;

for dataset = 1:1:size(Datasets, 2)
    s_dataset = Datasets(dataset);

    for distance = 1:1:size(Distances, 1)
        s_distance = char(Distances(distance));

        try
            eval(sprintf('U = kmeans(D_F%i_Std, 2, ''replicates'',
     100, ''distance'', ''%s'');', ...
                s_dataset, s_distance));
            A_kmeans{I} = {U, 'Std', s_dataset, s_distance, ...
                CheckLabels(U, accuracy)};
            display(I);
        catch err
            Errors{I} = {ex};
             display(ex);
        end

        try
            eval(sprintf('U = kmeans(D_F%i_NoStd, 2, ''replicates''
    ', 100, ''distance'', ''%s'');', ...
            s_dataset, s_distance));
            A_kmeans{I} = {U, 'noStd', s_dataset, s_distance, ...
                CheckLabels(U, accuracy)};
            display(I);
        catch err
            Errors{I} = {err};
            display(err);
        end

    I = I + 2;

    end

end

save('results/kmeans_all_nostd_all.mat', 'A_kmeans', 'Errors');

clc;
clear;

% Final results
%   A_kmeans{i} = [ [U], Std, Distance, Accuracy];
```

Listing 3: MATLAB code: i_Kmeans.m

```
function [Centroids, QtdEntitiesInCluster] = i_Kmeans(Data,
    MinEntitiesInCluster, IsDataStandarized, k)
%First Step = Standarize data if needed
InitialSize = size(Data,1);
QtdEntitiesInCluster = [];

if IsDataStandarized == false
    r = Data - repmat(mean(Data), InitialSize ,1);
    Data = r./repmat(max(Data) - min(Data),InitialSize , 1);
end

%Second Step = Sorts Data Accordint to Distance to Zero
[~,index] = sort(sum(Data.^2,2));
Data = Data(index,:);

%Third Step Anomalous Patter
```

```
Centroids = [];
while ~isempty (Data)
    CurrentSize = size(Data,1);
    TentCentroid = Data(CurrentSize,:); % Gets a tentative
    Centroid
    DistanceToCentre = sum(Data.^2,2);
    while true
        BelongsToCentroid = sum((Data-TentCentroid(ones(
    CurrentSize,1),:)).^2,2) < DistanceToCentre;  %faster
        NewCentroid = mean(Data(BelongsToCentroid==1,:),1);
        if isequal(TentCentroid, NewCentroid), break; end
        if (isp > 500), break; end
        TentCentroid = NewCentroid;
    end

    if sum(BelongsToCentroid==1) > MinEntitiesInCluster
        Centroids = [Centroids; NewCentroid]; %#ok<AGROW>
        QtdEntitiesInCluster = [QtdEntitiesInCluster; sum(
    BelongsToCentroid==1)];
    end
    Data(BelongsToCentroid==1,:)=[];
end

if nargin == 4 && size(Centroids,1) > k
    %Gets the k most populated clusters
    [~,ind] = sort(QtdEntitiesInCluster,'descend');
    Centroids = Centroids(ind(1:k),:);
end
```

Listing 4: MATLAB code: RuniKmeans.m

```
clc;
clear;

load('datasets\accuracy.mat');
load('datasets\dataset.mat');


Datasets = [0, 10, 20, 50, 100];


Distances = ['sqEuclidean'; 'cityblock  '; 'cosine     '; '
    correlation'];
Distances = cellstr(Distances);
```

```


Total = size(Datasets, 2) * size(Distances, 1) * 2;
A_ikmeans = cell(1, Total);
I = 1;

for dataset = 1:1:size(Datasets, 2)
    s_dataset = Datasets(dataset);

    if(I < 25)
        eval(sprintf('ZNoStd = i_Kmeans(D_F%i_NoStd, 0, true, 2);', s_dataset));
    end
    eval(sprintf('ZStd = i_Kmeans(D_F%i_Std, 0, true, 2);', s_dataset));

    for distance = 1:1:size(Distances, 1)
        s_distance = char(Distances(distance));

        if (I < 25)
            try
                        eval(sprintf('U = kmeans(D_F%i_NoStd, 2, ''distance'', ''%s'', ''start'', ZNoStd);', ...
                            s_dataset, s_distance));
                        A_ikmeans{I} = {U, ZNoStd, 'noStd', s_dataset, ...
                            s_distance, CheckLabels(U, accuracy)};
                        display(I);
                        clear U;
            catch err
            end
        end

        try
            eval(sprintf('U = kmeans(D_F%i_Std, 2, ''distance'', ''%s'', ''start'', ZStd);', ...
                s_dataset, s_distance));
            A_ikmeans{I+1} = {U, ZStd, 'Std', s_dataset, s_distance, ...
                CheckLabels(U, accuracy)};
            display(I+1);
            clear U;
        catch err
```

```
            end

            I = I + 2;

        end

        clear ZStd ZNoStd;
end

save('results/ikmeans_all_std.mat', 'A_ikmeans');

clc;
clear;

% Final results
%   A_ikmeans{i} = [ [U], [Z], Std, Distance, Accuracy];
% U - Final Cluster Labels
% Z - Initial Centroids
```

Listing 5: MATLAB code: RunHierarchical.m

```
clc;
clear;

load('datasets\accuracy.mat');
load('datasets\dataset.mat');

Datasets = [0, 10, 20, 50, 100];

Methods = ['average  '; 'centroid'; 'complete'; 'median  ';
    'single  '; 'ward    '; 'weighted'];
Methods = cellstr(Methods);

Distances = ['euclidean  '; 'seuclidean '; 'cityblock  ';
    'minkowski  '; 'chebychev  '; 'mahalanobis'; 'cosine     ';
    'correlation'; 'spearman   '; 'hamming    '; 'jaccard    '];
Distances = cellstr(Distances);

Total = size(Datasets, 2) * size(Methods, 1) * size(Distances, 1)
    * 2;
A_Hierarchical = cell(1, Total);
Errors = cell(1, Total);
I = 1;

```

```
for dataset = 1:1:size(Datasets, 2)
    s_dataset = Datasets(dataset);

    for method = 1:1:size(Methods, 1)
        s_method = char(Methods(method));

        for distance = 1:1:size(Distances, 1)
            s_distance = char(Distances(distance));

            try
                eval(sprintf('Z = linkage(D_F%i_Std, ''%s'', ''%s'');', ...
                    s_dataset, s_method, s_distance));
                U = cluster(Z, 'maxclust', 2);
                A_Hierarchical{I} = {U, Z, 'Std', s_dataset, s_method, ...
                    s_distance, CheckLabels(U, accuracy)};
                clear U Z;
                disp(I);
            catch ex
                Errors{I} = {ex, 'Std', s_dataset, s_method, s_distance};
                disp(ex);
            end
            try
                eval(sprintf('Z = linkage(D_F%i_NoStd, ''%s'', ''%s'');', ...
                s_dataset, s_method, s_distance));
                U = cluster(Z, 'maxclust', 2);
                A_Hierarchical{I+1} = {U, Z, 'NoStd', s_dataset, s_method, ...
                    s_distance, CheckLabels(U, accuracy)};
                clear U Z;
                disp(I+1);
            catch ex
                Errors{I+1} = {ex, 'NoStd', s_dataset, s_method, s_distance};
                disp(ex);
            end

            I = I + 2;

        end
    end
end

save('results/Hierarchical_all.mat', 'A_Hierarchical', 'Errors');
save('results/Hierarchical_Errors.mat', 'A_Hierarchical', 'Errors'
    );

clear;
clc;
```

Appendix B. Result tables

TABLE 1: Results of Hierarchical clustering

#	Std.	Features frequency	Method	Distance metric	Accuracy
1	Yes	>= 1	Average	Cosine	66.7500
2	Yes	>= 1	Average	Correlation	70.7500
3	Yes	>= 1	Complete	Cosine	55.8125
4	**No**	**>= 1**	**Complete**	**Correlation**	**75.6250**
5	No	>= 1	Complete	Spearman	65.1875
6	No	>= 1	Ward	Euclidean	70.0000
7	No	>= 1	Ward	City block	65.9375
8	No	>= 1	Ward	Minkowski	70.0000
9	No	>= 1	Ward	Chebychev	58.6875
10	Yes	>= 1	Ward	Cosine	69.2500
11	No	>= 1	Ward	Cosine	69.5625
12	Yes	>= 1	Ward	Correlation	64.0000
13	No	>= 1	Ward	Correlation	64.8750
14	Yes	>= 1	Ward	Spearman	66.0625
15	**No**	**>= 1**	**Ward**	**Spearman**	**75.3750**
16	Yes	>= 1	Ward	Hamming	67.4375
17	No	>= 1	Ward	Hamming	67.4375
18	Yes	>= 1	Ward	Jaccard	67.4375
19	No	>= 1	Ward	Jaccard	66.8125
20	Yes	>= 1	Weighted	Cosine	70.6250
21	Yes	>= 1	Weighted	Correlation	64.1250
22	Yes	>= 10	Average	Cosine	70.4375
23	Yes	>= 10	Average	Correlation	72.8750
24	Yes	>= 10	Complete	Cosine	55.6875

Continued on next page

Table 1 – *Continued from previous page*

#	Std.	Features frequency	Method	Distance metric	Accuracy
25	Yes	>= 10	Complete	Correlation	59.3125
26	No	>= 10	Complete	Correlation	65.5625
27	No	>= 10	Complete	Spearman	58.3750
28	Yes	>= 10	Ward	Euclidean	66.7500
29	No	>= 10	Ward	Euclidean	73.7500
30	Yes	>= 10	Ward	City block	61.8125
31	**No**	**>= 10**	**Ward**	**City block**	**79.9375**
32	Yes	>= 10	Ward	Minkowski	66.7500
33	No	>= 10	Ward	Minkowski	73.7500
34	Yes	>= 10	Ward	Chebychev	56.3125
35	No	>= 10	Ward	Chebychev	62.5000
36	Yes	>= 10	Ward	Cosine	69.5000
37	No	>= 10	Ward	Cosine	63.3750
38	Yes	>= 10	Ward	Correlation	70.6875
39	No	>= 10	Ward	Correlation	65.0000
40	Yes	>= 10	Ward	Spearman	66.2500
41	No	>= 10	Ward	Spearman	72.2500
42	Yes	>= 10	Ward	Hamming	70.2500
43	No	>= 10	Ward	Hamming	70.2500
44	Yes	>= 10	Ward	Jaccard	70.2500
45	No	>= 10	Ward	Jaccard	64.6875
46	Yes	>= 10	Weighted	Correlation	65.5000
47	Yes	>= 20	Average	Cosine	67.4375
48	Yes	>= 20	Average	Correlation	67.1875
49	No	>= 20	Complete	Cosine	56.2500
50	No	>= 20	Complete	Correlation	63.2500
51	No	>= 20	Complete	Spearman	57.2500
52	No	>= 20	Ward	Euclidean	68.6875
53	Yes	>= 20	Ward	City block	73.9375
54	**No**	**>= 20**	**Ward**	**City block**	**75.2500**
55	No	>= 20	Ward	Minkowski	68.6875
56	No	>= 20	Ward	Chebychev	56.1875

Continued on next page

Table 1 – *Continued from previous page*

#	Std.	Features frequency	Method	Distance metric	Accuracy
57	Yes	>= 20	Ward	Cosine	70.9375
58	No	>= 20	Ward	Cosine	66.9375
59	Yes	>= 20	Ward	Correlation	71.5625
60	No	>= 20	Ward	Correlation	60.3125
61	Yes	>= 20	Ward	Spearman	70.6875
62	No	>= 20	Ward	Spearman	72.1875
63	Yes	>= 20	Ward	Hamming	64.6250
64	No	>= 20	Ward	Hamming	64.6250
65	Yes	>= 20	Ward	Jaccard	64.6250
66	No	>= 20	Ward	Jaccard	68.0000
67	Yes	>= 20	Weighted	Correlation	63.5000
68	Yes	>= 50	Average	Cosine	66.5625
69	Yes	>= 50	Average	Correlation	67.0000
70	No	>= 50	Complete	Euclidean	55.9375
71	No	>= 50	Complete	Minkowski	55.9375
72	Yes	>= 50	Complete	Cosine	55.6250
73	Yes	>= 50	Complete	Spearman	62.3125
74	Yes	>= 50	Ward	Euclidean	66.0000
75	No	>= 50	Ward	Euclidean	62.8750
76	Yes	>= 50	Ward	Squared Euclidean	69.1875
77	No	>= 50	Ward	Squared Euclidean	69.1875
78	Yes	>= 50	Ward	City block	65.8125
79	No	>= 50	Ward	City block	63.3750
80	Yes	>= 50	Ward	Minkowski	66.0000
81	No	>= 50	Ward	Minkowski	62.8750
82	Yes	>= 50	Ward	Chebychev	55.2500
83	No	>= 50	Ward	Chebychev	60.0000
84	Yes	>= 50	Ward	Cosine	68.1250
85	Yes	>= 50	Ward	Correlation	68.5625
86	Yes	>= 50	Ward	Spearman	72.1875
87	Yes	>= 50	Ward	Hamming	70.3125
88	No	>= 50	Ward	Hamming	70.3125

Continued on next page

Table 1 – *Continued from previous page*

#	Std.	Features frequency	Method	Distance metric	Accuracy
89	Yes	>= 50	Ward	Jaccard	70.3125
90	No	>= 50	Ward	Jaccard	65.3750
91	Yes	>= 50	Weighted	Correlation	68.0625
92	Yes	>= 100	Average	Cosine	63.6250
93	Yes	>= 100	Average	Correlation	68.4375
94	Yes	>= 100	Complete	Cosine	56.0000
95	Yes	>= 100	Ward	Euclidean	58.0000
96	No	>= 100	Ward	Euclidean	69.3125
97	Yes	>= 100	Ward	Squared Euclidean	59.2500
98	No	>= 100	Ward	Squared Euclidean	59.2500
99	Yes	>= 100	Ward	City block	55.3750
100	No	>= 100	Ward	City block	59.0625
101	Yes	>= 100	Ward	Minkowski	58.0000
102	No	>= 100	Ward	Minkowski	69.3125
103	Yes	>= 100	Ward	Chebychev	55.6875
104	No	>= 100	Ward	Chebychev	72.1875
105	Yes	>= 100	Ward	Cosine	58.5625
106	Yes	>= 100	Ward	Spearman	72.8750
107	Yes	>= 100	Ward	Hamming	67.0625
108	No	>= 100	Ward	Hamming	67.0625
109	Yes	>= 100	Ward	Jaccard	67.0625
110	No	>= 100	Ward	Jaccard	64.5000
111	Yes	>= 100	Weighted	Cosine	65.5000
112	Yes	>= 100	Weighted	Correlation	57.7500

Table 2: Hierarchical clustering approaches with small accuracy

Method	Distance metric	Standardization	Features frequency
	Mahalanobis		
	Chebychev		
Average	City block	Yes & No	>= 1, 10, 20, 50, 100

Continued on next page

Table 2 – *Continued from previous page*

Method	**Distance metric**	**Standardization**	**Features frequency**
	Euclidean		
	Hamming		
	Jaccard		
	Minkowski		
	Spearman		
	Squared Euclidean		
	Correlation	No	
	Cosine		
Centroid	Mahalanobis	Yes & No	>= 1, 10, 20, 50, 100
	Hamming		
	Jaccard		
	Minkowski		
	Spearman		
	Squared Euclidean		
	Euclidean	No	>= 20, 50, 100
		Yes	>= 1, 10, 20, 50, 100
Complete	Mahalanobis	Yes & No	>= 1, 10, 20, 50, 100
	Hamming		
	Jaccard		
	Squared Euclidean		
	Chebychev		
	City block		
	Cosine	No	$>= 1, 10, 50, 100$
		Yes	$>= 20$
	Correlation	Yes	$>= 1, 20, 50, 100$
		No	$>= 50, 100$
	Spearman		
Median & Single	Mahalanobis	Yes & No	>= 1, 10, 20, 50, 100
	Hamming		
	Jaccard		
	Squared Euclidean		
	Chebychev		

Continued on next page

Table 2 – *Continued from previous page*

Method	**Distance metric**	**Standardization**	**Features frequency**
	City block		
	Cosine		
	Minkowski		
	Spearman		
	Euclidean		
	Correlation		
Weighted	Mahalanobis	Yes & No	>= 1, 10, 20, 50, 100
	Hamming		
	Jaccard		
	Squared Euclidean		
	Chebychev		
	City block		
	Minkowski		
	Spearman		
	Euclidean		
	Correlation	No	
	Cosine		
		Yes	>= 10, 20, 50
Ward	Mahalanobis	Yes & No	>= 1, 10, 20, 50, 100
	Squared Euclidean		>= 1, 10, 20
	Chebychev	Yes	>= 1, 20
	Euclidean		
	Minkowski		
	Spearman	No	>= 50, 100
	Cosine		
	Correlation		
		Yes	>= 100
	City block		>= 1