May 2020

Evaluation of Text Document Clustering Using K-Means

Lisa Beumer
University of Wisconsin-Milwaukee

Evaluation of Text Document Clustering using k-Means

by

Lisa Beumer

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Mathematics

at

The University of Wisconsin–Milwaukee

May 2020

ABSTRACT

Evaluation of Text Document Clustering using k-Means

by

Lisa Beumer

The University of Wisconsin-Milwaukee, 2020
Under the Supervision of Professor Istvan Lauko

The fundamentals of human communication are language and written texts. Social media is an essential source of data on the Internet, but email and text messages are also considered to be one of the main sources of textual data. The processing and analysis of text data is conducted using text mining methods. Text Mining is the extension of Data Mining to text files to extract relevant information from large amounts of text data and to recognize patterns. Cluster analysis is one of the most important text mining methods. Its goal is the automatic partitioning of a number of objects into a finite set of homogeneous groups (clusters). The objects should be as similar as possible within a group. Objects from different groups, however, should have different characteristics. The starting-point of cluster analysis is a precise definition of the task and the selection of representative data objects. A challenge regarding text documents is their unstructured form, which requires extensive pre-processing. For the automated processing of natural language Natural Language Processing (NLP) is used. The conversion of text files into a numerical form can be performed using the Bag-of-Words (BoW) approach or neural networks. Each data object can finally be represented as a point in a finite-dimensional space, where the dimension corresponds to the number of unique tokens, here words. Prior to the actual cluster analysis, a measure must also be defined to determine the similarity or dissimilarity between the objects. To measure dissimilarity, metrics such as Euclidean distance, for example, are used. Then clustering methods are applied. The cluster methods can be divided into different categories. On the one hand,

there are methods that form a hierarchical system, which are also called hierarchical cluster methods. On the other hand, there are techniques that provide a division into groups by determining a grouping on the basis of an optimal homogeneity measure, whereby the number of groups is predetermined. The procedures of this class are called partitioning methods. An important representative is the k-Means method which is used in this thesis. The results are finally evaluated and interpreted. In this thesis, the different methods used in the individual cluster analysis steps are introduced. In order to make a statement about which method seems to be the most suitable for clustering documents, a practical investigation was carried out on the basis of three different data sets.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence.

BoW Bag-of-Words.

CBOW Continuous Bag-of-Words.

CEO Chief Executive Officer.

CPU Central Processing Unit.

DF Document Frequency.

GloVe Global Vector.

HCA Hierarchical Cluster Analysis.

IDC International Data Corporation.

IDF Inverse Document Frequency.

JSON JavaScript Object Notation.

KDD Knowledge Discovery in Databases.

LM Language Model.

LSA Latent Semantic Analysis.

MLP Multi Layer Perceptron.

MSE Mean Squared Error.

NER Named Entity Recognition.

NLG Natural Language Generation.

NLP Natural Language Processing.

NLU Natural Language Understanding.

NNLM Neural Network Language Model.

PCA Principle Component Analysis.

POS Part of speech.

PPMCC Pearson Product-Moment Correlation Coefficient.

RI Rand Index.

SLP Single Layer Perceptron.

SMC Single Matching Coefficient.

SSE Summation of Squared Errors.

SVD Singular Value Decomposition.

TF Term Frequency.

TF-IDF Term Frequency-Inverse Document Frequency.

TM Text Mining.

TSVD Truncated Singular Value Decomposition.

UML Unified Modeling Language.

VSM Vector Space Model.

ZB Zetabyte.

Acknowledgments

At this point I would like to thank all those who supported me during my thesis. First and foremost, I would like to thank my principal supervisor Professor Istvan Lauko for his expertise, assistance and patience throughout the process of this thesis. I would also like to thank my committee members, Professor Gabriella Pinter and Professor Vincent Larson, for their support and encouragement.

My sincere thanks go to Professor Gerhard Dikta at Fachhochschule Aachen for his support applying for the Dual Degree Program and the UWM for giving me the opportunity to study in Milwaukee.

Finally, a special thanks to my family for their unconditional support throughout my studies.

Lisa Beumer

CHAPTER 1

Introduction

1.1 Motivation

Nowadays we are surrounded by a large amount of data of different types and the Internet has become a central part of the daily life for many people. Due to the development in technology and digitization the amount of data is rapidly increasing. In 2003 the former CEO of Google, Eric Schmidt, emphasized that *"Every two days now we create as much information as we did from the dawn of civilization up until 2003. That's something like five exabytes of data."* [44]. The ***I**nternational **D**ata **C**ooperation* (IDC) estimates in their last released report (November 2018) that in 2025 the total amount of data will reach 175 ZB [40, p. 3]. As Figure 1.1 shows this would be an increase from 33 ZB in 2018 to 175 ZB in 2025.

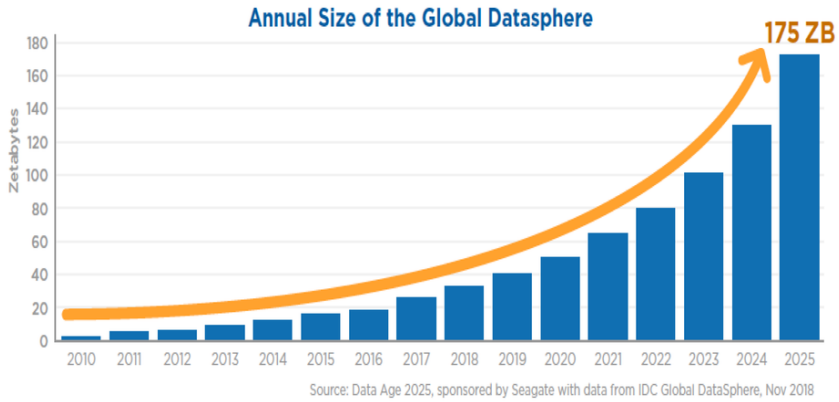


Figure 1.1: IDC study about the volume of data/information created from 2010 until 2025. Image source [40, p. 6].

Referring to [40, p. 5] a connected person will interact with technological devices every 18 seconds, so nearly 4800 times per day and everything we express no matter if verbally or textual carries huge amounts of information. Whether the data is produced from Alexa, Siri, WhatsApp, the Google search engine, online news or social media, they have one thing in common: Natural Language. This is the native speech of people used for communication, for example English, French or German. A big amount of textual data is generated in social media. Twitter users, for example, sent $511,200$ tweets and users conducted $4,497,420$ Google searches every minute last year [15]. Furthermore, there were $18,100,000$ texts and $188,000,000$ emails sent every minute [15]. An overview of the most popular platforms on which data is generated is shown in the Figure A.1 in the appendix on page 132. Due to the increase in available data, it is increasingly difficult for a user to find relevant information to his or her needs. In order to bundle, filter and extract knowledge from the huge number of digital texts, ***T**ext **M**ining* (TM) algorithms are used. Text Mining is an extension of the ***K**nowledge **D**iscovery in **D**atabases* (KDD) process and has several applications such as classification of news stories according to their content, Email filtering, clustering of documents or web pages. Further use cases are Customer Support Issue Analysis and Language Translation.

1.2 Research Objective

In this thesis one of the most fundamental text mining tasks, the text document clustering application is described. Clustering can be useful for information retrieval, topic extraction, organization of documents or support browsing. It can be described as *"Given a set of data points, partition them into a set of groups which are as similar as possible."* [3, p. 2]. So, the goal is to group text documents into categories (clusters) based on their similarity such that texts within a cluster are more similar than texts between different clusters. The techniques can be applied to different text granularities, such as document, paragraph, sentence or term level. This thesis discusses text document clustering. Since data generated from human language is unstructured because it doesn't follow specific rules, it doesn't fit directly into the row and column structure of databases as structured data does. In order to enable computers to work with this kind of data ***N****atural* ***L****anguage* ***P****rocessing* (NLP) was introduced in the 1950s [30, p. 5]. The clustering process consists of multiple steps which are illustrated in Figure 1.2. First, a suitable data set depending on the task has to be collected and pre-processed to improve the data quality and thus the accuracy and efficiency of the text mining process. To map the pre-processed textual data to a numerical representation several approaches can be used. One of the simplest techniques is the Bag-of-Words model (BoW). Based on these numerical representations the data similarity can be determined, which is needed to obtains a relatively small number of clusters compared to the large amount of data during the clustering process.

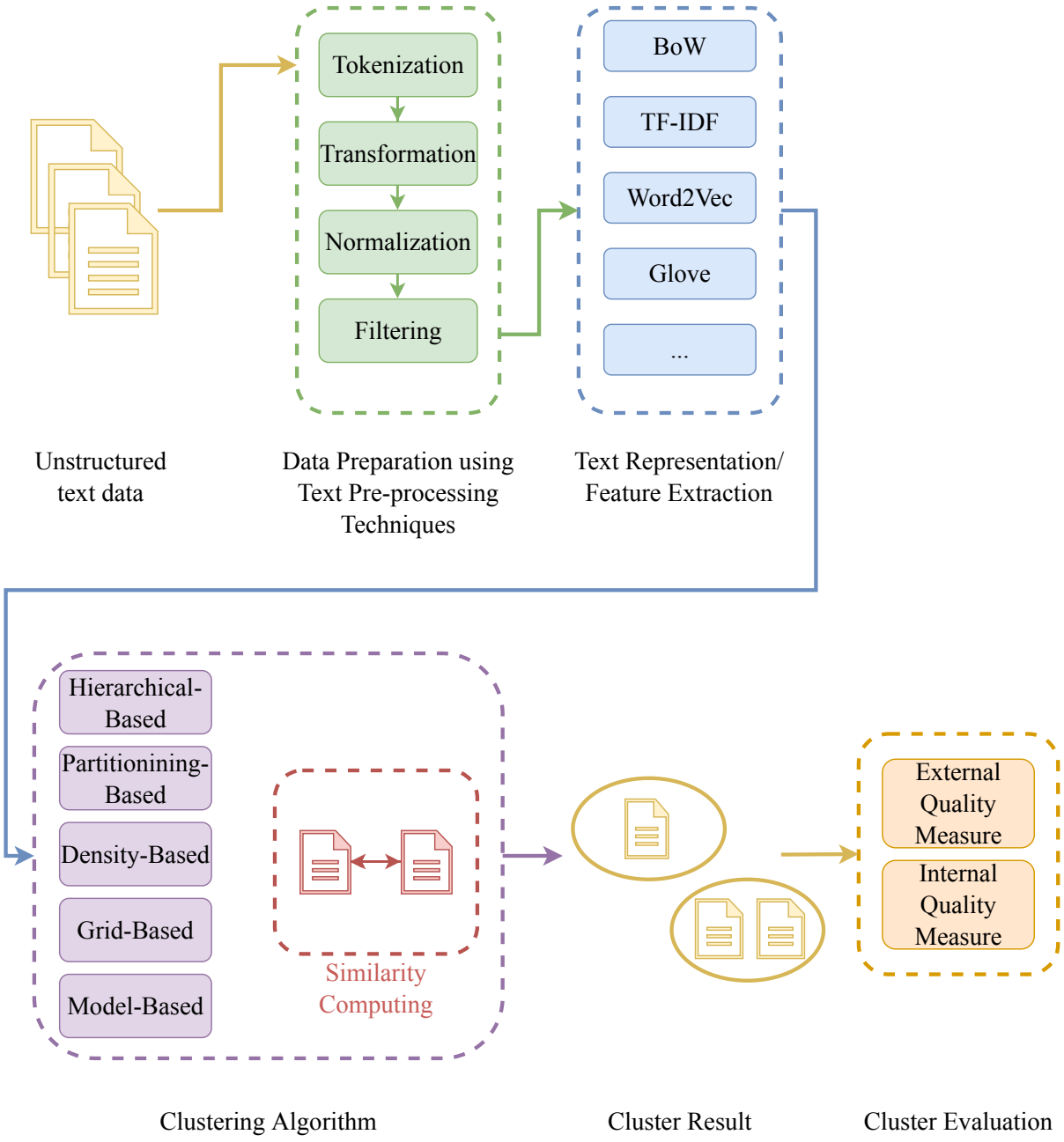


Figure 1.2: Text clustering steps.

The main motivation is to compare the effectiveness of these methods by finding out their potential inefficiencies. Since all of these steps described in Figure 1.2 effect the accuracy of the resulting cluster distribution, the techniques used to pre-process and to analyze the data as well as the different similarity measures and clustering algorithms are introduced and evaluated in this thesis. Before analyzing a large number of already proposed similarity measures the pre-processing steps have to be applied to the dataset. Therefore, various NLP techniques such as stemming or lemmatization in conjunction with various text representations are examined. Furthermore, the key challenge of clustering text data and some methods used to achieve this goal as well as their relative advantages are presented. There are many clustering algorithms proposed in the literature. Here, the focus is on the iterative partitional clustering algorithm called K-Means.

So the following challenges and research questions of finding the best cluster result for document clustering arise:

- Select appropriate features of the documents. What kind of pre-processing techniques need to be applied and what kind of word representation strategies result in suitable similarity results?

- Select a similarity measure between documents. What text similarity approach performs best on the given data sets?

- Select an appropriate clustering method.

- Efficient implementation in terms of required memory, CPU resources and execution time.

- Select a quality measure to check the cluster result.

1.3 Thesis Outline

The entire thesis contains eight chapters. Chapter 1 forms the motivational introduction and sets out the overall goal of this thesis. The following chapter 2 focuses on the main principle of automated data analysis, called KDD, and the extension to unstructured textual data, known as TM. Since TM involves techniques from NLP, this research discipline is briefly introduced in section 2.2. Generally, it is used to analyze and evaluate natural language to enable the computer to get an accurate interpretation of the text content by cleaning the data (see section 2.3) and extracting valuable information from it (see section 2.4). Chapter 3 introduces the main idea of text clustering, related techniques and evaluation methods. The main objective of chapter 4 is the analysis of text similarity measures based on the text representation approaches and the resulting features. The data sets used in this thesis are described in chapter 5. The practical part of the thesis is covered in chapter 6 where all acquired knowledge is used to determine the best similarity measure in combination with a suitable text representation method used to cluster different data sets using the k-Means cluster algorithm. The clustering results are evaluated in chapter 7. Finally, chapter 8 provides a conclusion and outlook according to the thesis' objectives.

CHAPTER 2

Introduction to Knowledge Discovery

The automated analysis and modeling of large data sets is called ***K**nowledge **D**iscovery in **D**atabases* (KDD). KDD is defined by Fayyad [18, 40f] as the *"nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data"*. Referring to [18, p. 39] the following disciplines are used for this purpose: databases, machine learning, statistics, artificial intelligence, high performance computing and data visualization. Figure 2.1 provides a summarization of the basic iterative KDD steps.

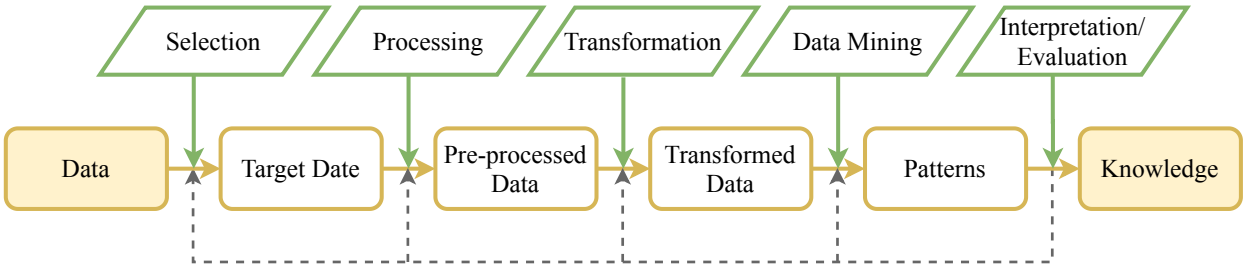


Figure 2.1: Main steps of the KDD process based on [18, p. 41].

The basic workflow of the KDD process consists of the following five phases:

Data selection: Select data according to the objective of the investigation.

Pre-Processing: This phase includes data cleaning and handling of missing values or errors and is a fundamental step for data analysis. Since there is no formal definition of data cleaning the procedure depends on the area the knowledge extraction is applied to.

Transformation:	The data is converted into the appropriate data format required by the analysis method. This includes feature extraction and dimensionality reduction to reduce the storage space and computation time.
Data mining:	Choose and apply data mining algorithm to extract data patterns.
Interpretation/Evaluation:	Evaluate and interpret the obtained results.

These five steps can be extended to a total number of nine [18, p. 42]. The more detailed KDD process lists the understanding of the application and the overall goal of the process as a separate step. Furthermore, the matching of the process goals to data mining methods and the selection of data mining parameters are listed separately. Finally, the usage of the newly gained knowledge for further action is added as the ninth phase.

According to [18, p. 37] and [34, p. 1], data mining is the core of KDD and includes after [18, p. 39] the extraction of new patterns using algorithms from the fields of machine learning, pattern recognition and statistics. Depending on the particular application, different data mining methods are applied. Figure 2.2 gives an overview of data mining applications. Clustering belongs to the category of discovery methods which automatically identify pattern in data. More precisely to the description branch which focuses on data understanding and interpretation.

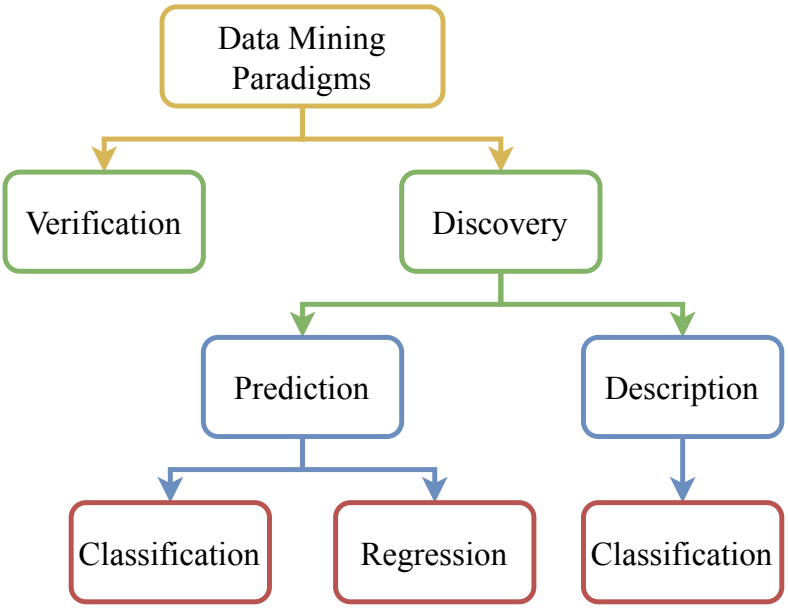


Figure 2.2: Data Mining taxonomy based on [34, p. 6].

2.1 Text Mining

An extension of KDD to unstructured textual data is called ***T**ext **M**ining* (TM) [24, p. 1]. Referring to [24, p. 1], text mining is defined as *"Text mining, also known as text data mining or knowledge discovery from textual databases, refers generally to the process of extracting interesting and non-trivial patterns or knowledge from unstructured text documents. [...]Text mining is a multidisciplinary field, involving information retrieval, text analysis, information extraction, clustering, categorization, visualization, database technology, machine learning, and data mining."*. Besides the same analytical methods of data mining, text mining also applies techniques from NLP (cf. [34, p. 809]).

2.2 Natural Language Processing

As stated in [13, 5f.], natural language processing as well as machine learning and deep learning are subfields of artificial intelligence. Figure 2.3 illustrates the coherence of these research areas as a Venn diagram.

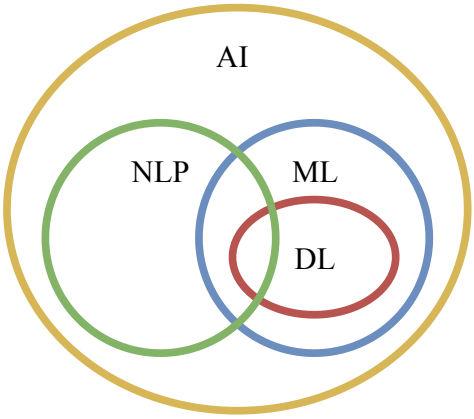


Figure 2.3: Venn diagram for natural language processing based on [13, pp. 5, 69]. The abbreviations are as follows: AI represents artificial intelligence, NLP stands for natural language processing, ML for machine learning and DL deep learning.

The research discipline ***A****rtificial* ***I****ntelligence* (AI) was initiated in 1956 [38, p. 87] and has become very popular in the last few years due to the availability of larger amounts of data, better developed algorithms, enhanced computing power and compact data storage methods. Through AI, machines are able to performs complex tasks comparable to the level of human performance and learn from experiences.

NLP, one sub field of AI, attempts to capture and process natural language using computer-based rules and algorithms. Processing human language is very complex because *"Human language is highly ambigious [...]. It is also ever changing and evolving. People are great at producing language and understanding language, and are capable of expressing, perceiving, and interpreting very elaborate and nuanced meanings. At the same time, while we humans are great users of language, we are also very poor at formally understanding and describing the rules that govern language."* [20, p. 1]. NLP applications must be able to handle for example ambiguity, slang, social context and syntactic variations. Therefore various methods and results from linguistics are combined with artificial intelligence. Because NLP is so complex, it can be divided into two major sub-disciplines: ***N****atural* ***L****anguage* ***U****nderstanding* (NLU) and ***N****atural* ***L****anguage* ***G****eneration* (NLG).

2.2.1 Natural Language Understanding

NLU which is about extracting the meaning of natural language is considered as the integral part of NLP and is also a very complex task [30, p. 91]. For natural language processing, it is not only necessary to understand individual words and sentences, but also to take the complete text into consideration to get an idea about the context. This is one of the reasons why there is much more behind natural language processing than just a dictionary in form of a large database. According to [30, p. 15], natural language understanding is based on different language domains as displayed in Figure 2.4.

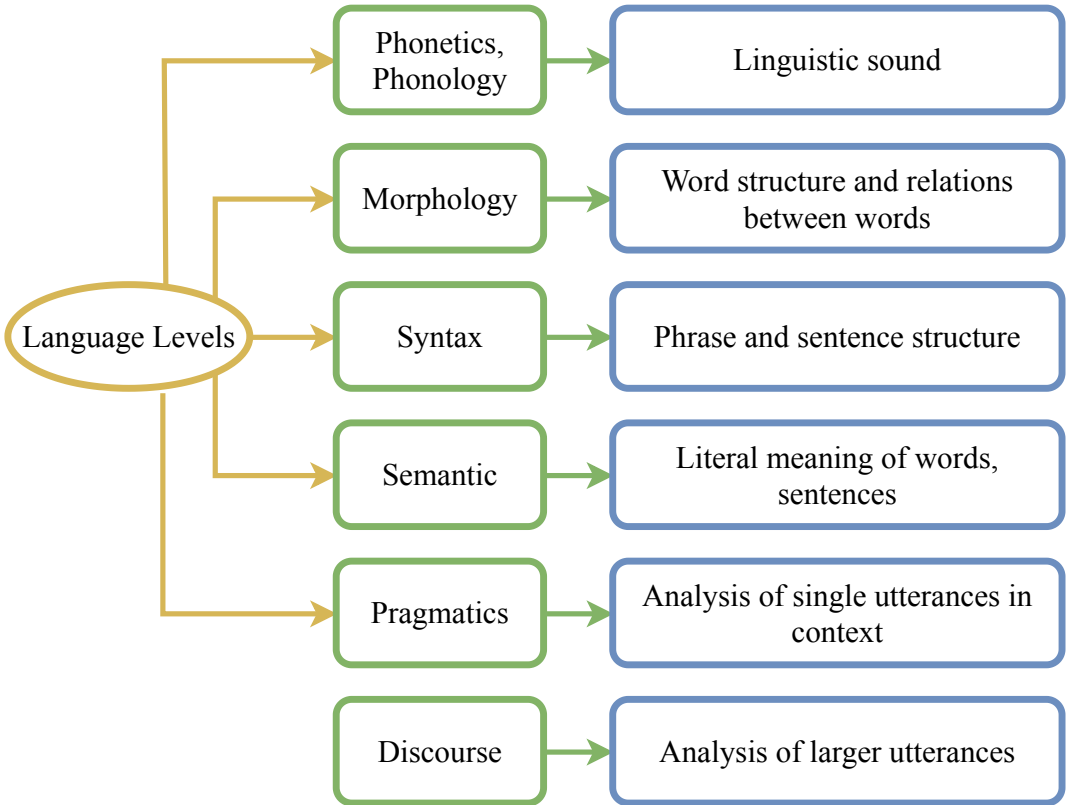


Figure 2.4: This diagram outlines the categories of linguistics. Image based on [30, p. 15].

The analysis of natural language consists of phonological, morphological, lexical, syntactic, semantic, pragmatic and discourse analysis (cf. [30, p. 17]). Phonological analysis includes the study of the pronunciation of words (phonetics) and the structure of sounds (phonology). The second component is the most elementary one and deals with the structure and formation

of words from morphemes (cf. [30, p. 17]). The word *played*, for example, is composed of *play-ed* and the word *unhappiness* of *un-happi-ness*. Breaking a word down into its morphemes is called morphological analysis. Morphemes can be divided into stems and affixes which can be further subdivided into prefix and suffix. There is often more than one affix added to a word, which particularly influences the grammatical meaning (cf. [26, p. 60]). The grammatical process of word formation used to express, for example, the number, case, gender or mood of a word is called inflection (cf. [30, p. 30]). The canonical form of an inflected form is called lemma. A lemma in combination with its inflection form is known as lexeme which thus can be represented as a set of forms with the same meaning. A collection of all lexemes is called dictionary. There exist two different ways to reduce inflection forms: (1) stemming and (2) lemmatization. They share the same idea but use different ways to achieve the result. With the simpler option stemming, words are reduced to their stem using heuristics that cut off the end of words to achieve the correct base. The problem is that a stemming algorithm may cut off too much (overstemming) because it doesn't take the context of the word into account. Taking the words *operate*, *operation*, *operational* or *university*, *universal*, *universe* the algorithm may produce *oper* and *univers* as stems. The lemmatization algorithm, however, relates different forms of the same word to their dictionary form (lemma). Therefore it determines the part-of-speech (POS) which is essential to identify the grammatical context. Taking the sentence

$$\textit{The quick brown fox jumped over the lazy dog.} \tag{2.1}$$

the following part-of-speech tags are determined:

$$\underbrace{\textit{The}}_{DET}\ \underbrace{\textit{quick}}_{ADJ}\ \underbrace{\textit{brown}}_{ADJ}\ \underbrace{\textit{fox}}_{NOUN}\ \underbrace{\textit{jumped}}_{VERB}\ \underbrace{\textit{over}}_{ADP}\ \underbrace{\textit{the}}_{DET}\ \underbrace{\textit{lazy}}_{ADJ}\ \underbrace{\textit{dog}}_{NOUN}\ \underbrace{.}_{PUNCT},$$

The tag `DET` stands for *determiner*, `ADJ` for *adjective*, `NOUN` for *noun, singular or mass*, `VERB` for *verb, modal auxiliary* and *ADP* for *adverb*. If, for instance, the word *saw* is given,

lemmatization would return *see* or *saw* whether the original word is a noun or a verb in the context of the sentence.

The syntax and the semantic component of Figure 2.2 provide information about the lexical meaning of a sentence (cf. [30, p. 46]). This process is visualized in figure 2.5.

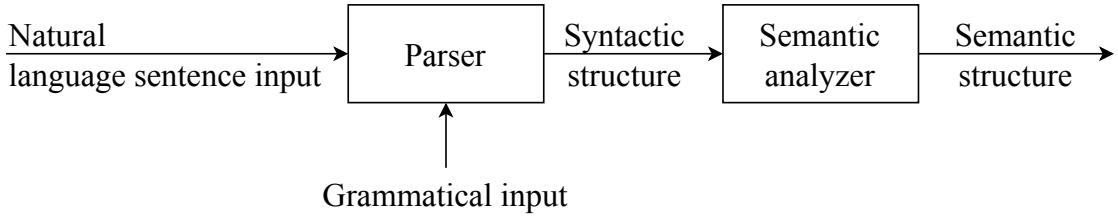


Figure 2.5: Two phase process of natural language understanding. Image taken from [30, p. 46]).

To transform a sentence into a syntactic structure a parser is used which evaluates each sentence compared to formal grammar rules to provide the sentence structure. The resulting structure is called parse tree. The word references of the example sentence 2.1 are displayed in figure 2.6. The linguistic analysis steps applied to get this visualization are implemented in Python and are provided in the appendix in section D.2.

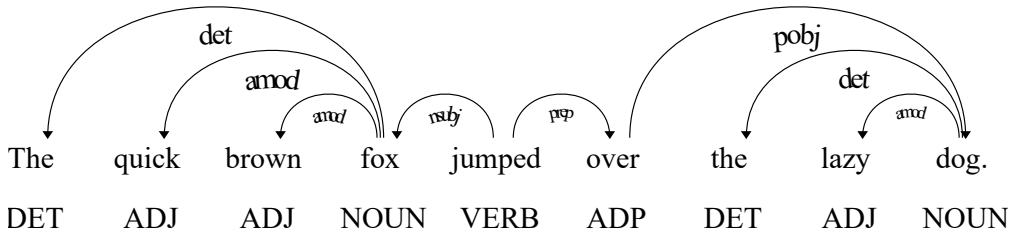


Figure 2.6: Dependency Structure for the example sentence.

The syntactic dependency label `det` is a representation for *determiner*, `amod` for *adjectival modifier*, `nsubj` for *nominal subject*, `prep` for *prepositional modifier* and `pobj` for *object of preposition*. The syntactic structure is then used by a semantic analyzer to establish a correct logic between words and sentences. This is done by determining the basic dependencies of a word related to other words. One common technique used for this purpose is called ***N****amed* ***E****ntity* ***R****ecognition*, NER for short.

To fully understand the natural language, the intended message of the whole text needs to be taken into account. The pragmatic and discourse analysis (step 5 and 6, Fig. 2.2) accomplish this. Pragmatics include the intention and context of the whole text whereas discourse analysis considers the immediately preceding sentences to interpret the actual sentence.

The previous explanations indicate that the analysis of natural language has a complex structure. Each of the introduced categories of linguistics relies on certain models, rules and algorithms. The next section provides an overview of the pre-processing steps applied to textual data in order to extract the meaning which are based on the categories presented above.

2.3 Handling Unstructured Data - Text Pre-Processing

The discrete symbols such as letters and words cannot be processed directly by a computer and must be converted into a structured system that enables automatic processing and evaluation. As stated in [34, p. 19], approximately 40% of the collected data is noisy. In order to improve the efficiency of algorithms working on textual data the data is pre-processed before transforming it into a form computer can work with (cf. [5]). Pre-processing consists of several steps, which are applied depending on the application and the data. Different kinds of data, such as images, text or videos require different pre-processing methods. There exist several studies about pre-processing techniques which are recommended for text document

clustering. According to [5], [27] and [29], pre-processing usually involves tokenization, filtering, normalization (lemmatization or stemming). These techniques are briefly described in the following.

2.3.1 Tokenization

Before information can be extracted from a sentence, it is tokenized which means that it is broken down into its individual parts. One approach is to split up a sentence by spaces [43, p. 264]. Doing so, all punctuation marks and brackets are not recognized as independent tokens. Thus, the words *"dog"* and *"dog."* for example are captured as separate tokens even though they express the same thing. Another approach is to consider punctuation as word boundaries (cf. [43, p. 264]). This however is not suitable depending on the application case, since contractions occurring in the text are separated and in the example of *can't* the resulting division *can* and *t* can no longer be used to determine the negative meaning. Consequently, tokenization is a very complex process that has to be adapted to the problem and the used language.

2.3.2 Filtering

Data filtering is applied to remove frequently used words which do not contain much information and usually just have a grammatical purpose [5]. Those words are called *stop words*. Some examples are *a*, *an* or *the*. Filtering consists of various other steps, for example, removing numbers, symbols, whitespaces or punctuation and general noise removal steps like removing text file headers and footers as well as HTML or metadata.

2.3.3 Normalization

Since words that share the same representation can exist as individual tokens because they are for example conjugated verbs, a lexical normalization is performed which transforms a word

into its canonical form. This can be achieved by stemming or lemmatization. Furthermore, as mentioned in [29], it could be useful to convert upper case into lower case to avoid the same tokens. This technique is called case folding [43, p. 264]. However, it is important to apply the techniques carefully so that the meaning of the text is not destroyed.

2.4 Handling Unstructured Data - Text Representation

As can be seen in Figure 1.2, the pre-processing stage is followed by the conversion of the generated, possibly filtered and normalized, tokens into a numerical representation that makes the unstructured text mathematically computable and manageable by text mining algorithms. A popular model used in this context is called ***V**ector **S**pace **M**odel* (VSM). Before going into detail some new terminology has to be introduced (cf. [26, Sec. 23.1, p. 4]):

1. Document : The unit of text such as paragraphs, articles or sentences.
2. Collection or corpus: A set of documents.
3. Term : Lexical item of a document.

2.4.1 Vector Space Model

The VSM is an algebraic model based on similarity and represents each text document of a collection C as a vector of weighted features in an N-dimensional vector space, where N is the total number of unique terms occurring in the corpus which is also called vocabulary. Let n be the number of documents in a collection, then each document d_j can be represented as a vector

$$d_j = (w_1, w_2, ..., w_N), \tag{2.2}$$

where $j \in 1...n, 1 \leq i \leq N$ and w_i is the weight of the term i in document j (cf. [26, Sec 23. p.5]). Joining these vectors leads to a term-document-matrix [26, Sec. 23.1 p. 7]. Since

a text collection can contain many terms, it has to be first determined which of them are used as features and how to compute their weights. A common feature extraction technique for textual data is the ***B**ag-**o**f-**W**ords* model, or BoW for short. BoW generates a text representation using its words (1-gram). An n-gram is a probabilistic language model based on word count. ***L**anguage **M**odeling* (LM) was proposed in the 1980s (cf. [31]) and is defined as *"[...] the task of assigning a probability to sentences in a language. [...] Besides assigning a probability to each sequence of words, the language models also assign a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words"* [20, p. 105]. So, LM learns to predict the probability distribution of a sequence s of n words (cf. [31])

$$P(s) = P(w_1, w_2, ..., w_n). \tag{2.3}$$

The probability of a sequence s can be expressed as a product of conditional probabilities.

Definition 1 (Conditional Probability)

Given two events A, B. The conditional probability of A given B with $p(B) > 0$ is defined as

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A|B)P(B)}{P(B)}. \tag{2.4}$$

It follows that

$$\begin{aligned} P(s) &= P(w_1, w_2, ..., w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w1w2)\cdots P(w_n|w1w2\cdots w_{n-1}) \\ &= \prod_i P(w_i|w_1 w_2 \cdots w_{i-1}). \end{aligned} \tag{2.5}$$

Because this model depends on many parameters, it is assumed as a simplification that it is sufficient to consider only a maximum of k preceding words [31]

$$P(w_i|w_1 \cdots w_{i-1}) \approx P(w_i|w_{i-k} \cdots w_{i-1}). \tag{2.6}$$

This assumption is also known as Markov assumption. Finally, it follows that

$$P(s) = P(w_1, ..., w_n) \approx \prod_{i=1} P(w_i | w_{i-k} \cdots w_{i-1}). \tag{2.7}$$

Since n-gram models are based on word count, the individual conditional probabilities calculated in 2.7 are based on frequencies, by which the corresponding n-grams appear in the text. Hence

$$P(w_i | w_1 \cdots w_{i-1}) = \frac{count(w_{i-k}, \cdots, w_{i-1}, w_i)}{count(w_{i-k}, \cdots, w_{i-1})}. \tag{2.8}$$

The main idea of BoW is that the meaning of a document is only comprised in its terms and that similar documents have similar content. So each word is considered as a feature making the assumption that the word order and the grammatical structure do not matter (cf. [43, p. 265], [20, p. 69]), which is why this technique is referred to as "bag". This results in the simplest form of 2.7 with $n = 1$, called unigram model or 1-gram model

$$\begin{aligned} P(s) &\approx P(w_1)P(w_2)P(w_3)\cdots P(w_n) \\ &\approx \prod_{i=1}^{n} P(w_i). \end{aligned} \tag{2.9}$$

Let a collection contain two documents:

Document 1:	Document2:
The quick brown fox jumped over the lazy dog.	*The dog is lazy! The fox jumps around.*

(2.10)

Since every unique word is treated as a feature, the generated vocabulary for the non-pre-

processed data contains the following 14 elements which are linked to a vector index:

$$\{0 : \text{The}, 1 : \text{quick}, 2 : \text{brown}, 3 : \text{fox}, 4 : \text{jumped}, 5 : \text{over}, 6 : \text{the},$$
$$7 : \text{lazy}, 8 : \text{dog}, 9 : ., 10 : \text{is}, 11 : !, 12 : \text{jumps}, 13 : \text{around}\} \tag{2.11}$$

So every document will be represented by a feature vector of 14 elements. The following term-document-co-occurrence matrix is obtained for the example provided in 2.10.

$$\begin{array}{c} \\ \textit{the} \\ \textit{quick} \\ \textit{brown} \\ \textit{fox} \\ \textit{jump} \\ \textit{over} \\ \textit{lazy} \\ \textit{dog} \\ \textit{be} \\ \textit{around} \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \left(\begin{array}{cc} 2 & 2 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{array} \right) \end{array} \in \mathbb{R}^{10\times 2}$$

So if the vocabulary is increasing the vector dimension N does. One technique to decrease the vocabulary length is to apply the text pre-processing methods discussed in 2.3. Thereby the vocabulary of 2.11 can be decreased to 10 terms by just ignoring case and punctuation, converting every word to lowercase and lemmatizing terms.

$$\{0 : \text{the}, 1 : \text{quick}, 2 : \text{brown}, 3 : \text{fox}, 4 : \text{jump}, 5 : \text{over}, 6 : \text{lazy},$$
$$7 : \text{dog}, 8 : \text{be}, 9 : \text{around}\} \tag{2.12}$$

However it should be obvious that this approach will not solve all problems. A better idea

is to create a vocabulary based on n grouped tokens (n-grams with $n > 1$). An additional advantage is that they retain some context (e.g. *New York*) and thus capture a little bit more meaning from the document. Nevertheless, it suffers from the fact that language has long distance dependencies but depending on the task *"[...] a bag-of-bigrams representation is much more powerful that bag-of-words [...]"* [20, p. 75]. The vocabulary of a bigram model for the example (2.10) above is:

$$\{0 : \text{around brown}, 1 : \text{brown dog}, 2 : \text{dog fox}, 3 : \text{fox is}, 4 : \text{is jumped},$$
$$5 : \text{jumped lazy}, 6 : \text{lazy over}, 7 : \text{over quick}, 8 : \text{quick the}\} \tag{2.13}$$

A more powerful dimensionality reduction technique is discussed in section 2.4.3. Once the vocabulary has been chosen, the occurrence of terms has to be scored. There exist three methods for weighting the BoW obtained features: (1) one-hot encoding, (2) frequency vectors, also called count vectors and (3) term frequency/inverse document frequency.

One-hot encoding

One-hot encoding is a binary weighting approach where words which are not included in the vocabulary get marked as 0 and present ones as 1. The following feature vector is assigned to the content of the first document presented in 2.10 based on the vocabulary described in 2.12:

	Vocabulary									
	the	quick	brown	fox	jump	over	lazy	dog	be	around
The quick brown fox jumped over the lazy dog.	1	1	1	1	1	1	1	1	0	0

Using boolean values makes it difficult to extract sentence similarity, since the generated vectors are orthogonal. Furthermore the computation of the resulting sparse vectors with lot of 0-values is inefficient because of the huge computation time and the need of more memory.

Frequency Vectors

Another weighting method is the counting approach based on the ***t**erm **f**requency* (TF). The term frequency of a term t within a document $d = \{t_1, t_2, ..., t_m\}$ containing m terms is the amount of times it appears in d and can be defined as (cf. [43, p. 269])

$$tf(t,d) = \sum_{i=1}^{m} f(t,t_i) \text{ with } t_i \in d, |d| = m \text{ and } f(t,t') = \begin{cases} 1, & \text{if t=t'} \\ 0, & \text{otherwise.} \end{cases}, \tag{2.14}$$

The frequency vector for document one 2.10 using the vocabulary 2.12 is:

	Vocabulary									
	the	quick	brown	fox	jump	over	lazy	dog	be	around
The quick brown fox jumped over the lazy dog.	2	1	1	1	1	1	1	1	0	0

This approach implies that terms occurring frequently within a document are more important than less frequently appearing terms and should therefore receive a higher weight. However, frequent words may not contain as much information about the content as rarer do. To alleviate this problem the word frequency can be re-scaled using the ***i**nverse **d**ocument **f**requency* (IDF). This measure is called ***t**erm **f**requency/**i**nverse **d**ocument **f**requency* (TF-IDF).

Term Frequent/Inverse Document Frequency

This approach re-scales the term frequency 2.14 with an inverse document frequency which scores how rare a term is across documents, whereby the frequency of documents containing

a term is defined as ***d**ocument **f**requency* (DF). Let $C = \{d_1, d_2, ..., d_n\}$ be a collection of n documents d, then DF is defined as (cf. [43, p. 271])

$$df(t) = \sum_{i=1}^{n} f'(t, d_i) \text{ with } d_i \in C, |C| = n \text{ and } f'(t, d') = \begin{cases} 1, & \text{if t=d'} \\ 0, & \text{otherwise.} \end{cases}, \tag{2.15}$$

The IDF inversely corresponds to the DF and is defined as (cf. [43, p. 272])

$$idf(t) = log \frac{n}{df(t)}, \tag{2.16}$$

where t is the term and n the total number of documents within a collection.
Equation 2.14 and 2.16 in combination result in the TF-IDF weight (cf. [43, p. 272])

$$tfidf(t, d) = tf(t, d) \times idf(t). \tag{2.17}$$

The TF-IDF value increases if the number of occurrences of a given term in the document increases and with an increase in a rarity of the word across the corpus documents. The TF-IDF vector of the first document presented in 2.10 is:

Vocabulary	TF Document 1	TF Document 2	IDF	TF*IDF Document 2
the	2/9	2/8	$log(2/4)$	-0.067
quick	1/9	0/8	$log(2/1)$	0.033
brown	1/9	0/8	$log(2/1)$	0.033
fox	1/9	1/8	$log(2/2)$	0
jump	1/9	1/8	$log(2/2)$	0
over	1/9	0/8	$log(2/1)$	0.033
lazy	1/9	1/8	$log(2/2)$	0
dog	1/9	1/8	$log(2/2)$	0
be	0/9	1/8	$log(2/1)$	0
around	0/9	1/8	$log(2/1)$	0

Besides the already mentioned disadvantages, e.g. the vector dimensionality or the computational time complexity due to the obtaining sparse vectors, the BoW model is, as a literature search showed, a common model for text mining (cf. [43, p. 265], [33]) and provides good results in text clustering applications (cf. [5], [27], [42], [7]). However, this approach has some other limitations not discussed yet. The assumptions made in the BoW model work for many tasks but not for meaning-related tasks including senses or synonyms. Documents containing similar content but different term vocabulary will not be marked as similar. Furthermore the word order plays an important role. As the following example shows it is not sufficient to use only lexical (surface) similarity[1]. Let the first document contain *The fox jumped over the dog* and the second one *The dog jumped over the fox*. The words are an exact overlap but the context is totally different. Thus it is important to take semantic and syntactic similarity into account. This can also be shown by looking at the terms *New* and *York* which mean a completely different thing when occurring together as *New York*.

Some of these problems can be overcome by using dense vectors instead of sparse ones for word representation. According to [20, p. 92], *"One of the benefits of dense and low-dimensional vectors is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors."*. Moreover, dense vectors represent the words' context and capture dependency structures. The representation of words by dense vectors is known as word embedding.

2.4.2 Word Embedding

Word embeddings were developed to overcome the problems of the traditional bag-of-words model. According to [6] word embeddings are *"dense, distributed, fixed-length word vectors"* which can express the meaning of a word mathematically. A popular example is shown in Figure 2.7.

[1]Lexical similarity measures if the sets of terms of two texts are similar. A lexical similarity of 1 implies a full overlapping of words, while 0 means that there are no common words in both sets.

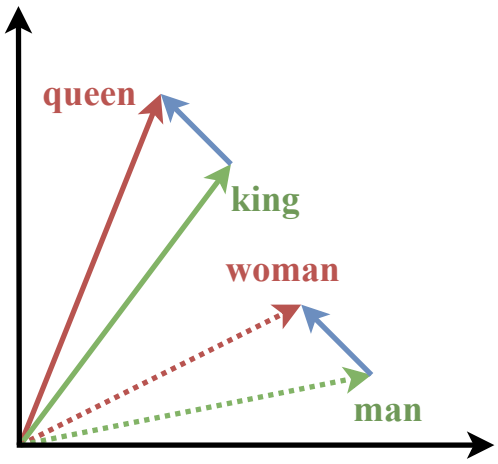


Figure 2.7: Word Embeddings make it possible to solve analogies by solving simple vector arithmetic. The closest embedding to the resulting vector from *king* − *man* + *woman* is *queen*.

The semantic relationships between the word *queen* and *king* can be determined by simple algebraic operations [6]. For example, $\overrightarrow{vec}_{king} - \overrightarrow{vec}_{man} + \overrightarrow{vec}_{woman} \approx \overrightarrow{vec}_{queen}$. So, a feature vector projects each word into a relatively low dimensional vector space where vectors of words with similar meaning are close together. Each word vector is *"built using word co-occurence*[2] *statistics as per distributional hypothesis"* [6]. The distributional hypothesis has been developed by Harris in 1954 [6] and is based on the famous statement by the linguist Firth *"You shall know a word by a company it keeps!"* [19, p. 11] or as [41] suggested *"words which are very similar in meaning will indeed be very similar in contextual distribution"*. In general, word embedding techniques can be divided into two categories: (1) count-based and (2) predictive-based methods (cf. [6]). Count-based vector space models rely on the word frequency and co-occurrence matrices which count how often term co-occur in some environment and are described in 2.4.3. In addition, neural networks which are able to learn low-dimensional word representations are introduced in 2.4.4.

[2]A co-occurence matrix contains the number of times each entity in a row appears in same context as each entity in a columns.

2.4.3 Count-based methods

Count-based methods map high-dimensional count vectors to a lower-dimensional representation, called latent semantic space, by preserving the semantic relationship. They use a co-occurrence matrix to determine how often a word occurs together with its neighboring words in a context window which is specified by a number and the direction. The co-occurrence matrix of 2.10 based on the vocabulary 2.12 and a context window length of 2 is

	the	*quick*	*brown*	*fox*	*jump*	*over*	*lazy*	*dog*	*be*	*around*
the	0	1	1	1	2	1	4	2	2	0
quick	1	0	1	1	0	0	0	0	0	0
brown	1	1	0	1	1	0	0	0	0	0
fox	1	1	1	0	2	1	1	0	0	1
jump	2	0	1	2	0	1	0	0	0	1
over	1	0	0	1	1	0	1	0	0	0
lazy	4	0	0	1	0	1	0	1	1	0
dog	2	0	0	0	0	0	1	0	1	0
be	2	0	0	0	0	0	1	1	0	0
around	0	0	0	1	1	0	0	0	0	0

$\in \mathbb{R}^{10\times 10}$,

where the rows contain the focus words and the columns the context words. The matrix entries are generated as follows:

the	quick	brown	fox	jump	over	the	...
the	quick	brown	fox	jump	over	the	...
the	quick	brown	fox	jump	over	the	...
the	quick	brown	fox	jumps	over	the	...
the	quick	brown	fox	jump	over	the	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

The words highlighted in red are the so-called focus words. The words marked in green are called context words and are counted for the co-occurrence matrix. Since the context window has been set to 2, each red word is surrounded by two green words in each direction. The co-occurrence of the word *quick*, for example, is 1 0 1 1 0 0 0 0 0 0. Since the word *quick*, for example, occurs only once within the corpus 2.10, the co-occurrence values can be explained using the shown co-occurrence matrix above. As shown, the context words of *quick* are *the*, *brown* and *fox*. Thus every other word in the vocabulary is assigned a 0. Each of these three words occurs only once, so a 1 is assigned in each case. One commonly used count-based word embedding method was developed in 1990 [16] and is called ***L**atent **S**emantic **A**nalysis* (LSA). According to [16], it overcomes two fundamental problems: synonymy (variability of word choice) and polysemy(words can have multiple meanings). The algorithm takes as input a term-document matrix C and represents each document as a vector after modeling term-document relationships by extracting latent features using a low-rank approximation for the column and row space computed by a ***s**ingular **v**alue **d**ecomposition* (SVD) of the input matrix. The procedure is as follows ([35, p. 411]). Let C be given, derive $C = U\Sigma V^T$ using singular value decomposition.

Definition 2 (Singular Value Decomposition)

Let t be the number of terms and d the number of documents. Given a rectangular term-document matrix C of size $t \times d$ and $rank(C) = r \leq min\{t, d\}$ the factorization of C, denoted by $SVD(C)$, into the following three matrices is defined as [35, p. 408]

$$C = U\Sigma V^T, \tag{2.18}$$

where the orthonormal matrices $U \in \mathbb{R}^{t\times t}$ and $V \in \mathbb{R}^{d\times d}$ contain the eigenvectors of C^TC and the eigenvectors of CC^T. The columns of U and V are also called left and right singular vectors. The eigenvalues of CC^T are the same as the eigenvalues of C^TC. CC^T is a square matrix which row and column correspond to each of the t terms. So, each entry (i, j) represents the overlap between the i-th and j-th terms, based on their co-occurrence in

documents. In other words, the entry is the number of documents in which both term i and term j occur. The diagonal matrix $\Sigma \in \mathbb{R}^{t \times d}$ *contains the singular values* $\sigma_i = \sqrt{\lambda_i}$*, with* $\lambda_i \geq \lambda_{i+1}, 1 \leq i \leq r$ *and is defined as*

$$\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_d), \sigma_i > 0 \text{ for } 1 \leq i \leq r \text{ and zero otherwise.} \tag{2.19}$$

The matrix Σ can be represented as a $r \times r$ matrix because all eigenvalues $\sigma_i, i > r$ are 0. By removing the rightmost $t - r$ columns of U and the rightmost $d - r$ columns of V, the dimension of these matrices can be reduced to $\mathbb{R}^{t \times r}$ and $\mathbb{R}^{r \times d}$. This reduced form of SVD is also called ***t****runcated singular value decomposition*, TSVD for short. Define a positive number $k \leq r$ which is usually far smaller than r and replace the $r - k$ smallest singular values of Σ. The idea is to find a C_k of rank at most k which minimizes the Frobenius norm of $X = C - C_k$ defined as (cf. [35, p. 410])

$$||X||_F = \sqrt{\sum_{i=1}^{t} \sum_{j=1}^{N} X_{ij}^2}. \tag{2.20}$$

The following theorem due to Eckard and Young states that the low-rank approximation can be described as a minimization problem (cf. [35, p. 411])

Theorem 1

$$\min_{Z|rank(Z)=k} ||C - Z||_F = ||C - C_k||_F = \sigma_{k+1}. \tag{2.21}$$

It follows that the matrix C_k constructed this way has the lowest possible Frobenius error and thus is the best rank k least-squares approximation of C incurring an error equal to 2.21 [35, p. 411]. For a larger k the error gets smaller and for $k = r$ the error is 0. The last step is to compute $C_k = U \Sigma_k V^T$. Figure 2.8 shows a schematic diagram of the SVD for a $t \times d$ term-document matrix.

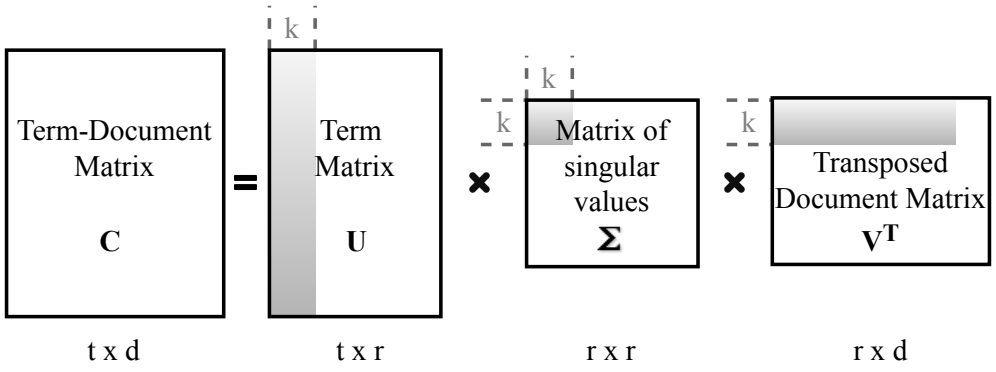


Figure 2.8: SVD decomposition of a term-document matrix C where t is the amount of terms in all documents and d the number of documents within a corpus. The parameter r is the rank of C and k is a chosen positive value usually far smaller than r. The image is based on Fig. 2 of [16, p. 398] and [35, pp. 407–412].

This approach scales quadratically with $\mathcal{O}(dt^2)$ for a $t \times d$ matrix. This computational complexity can be overcome by using a different count-based model developed in 2014, which is called ***G**lobal **V**ector* (GloVe) model. It aims to predict surrounding words instead of determining the co-occurrence matrix directly. This approach is not discussed in this thesis.

2.4.4 Predictive-based methods

Predictive-based word embedding methods derive from ***N**eural **N**etwork **L**anguage **M**odels* (NNLMs) which learn the factors of the product 2.7 using neural networks instead of a count-based approach as presented in section 2.4. As part of supervised learning, neural networks are a relevant part of machine learning. A neural network is based on a model of neural connectivity within the human brain and can learn from data through training to recognize patterns, classify data and predict future events. It is defined as follows [28, p. 34]

Definition 3 (Neural Network)

A neural network is a sorted triple (N, V, w), with two sets N, V and a function w, where N is the set of neurons and V a set $\{(i,j)|i,j \in \mathbb{N}\}$ whose elements are called connections between neuron i and neuron j. The function $w : V \to \mathbb{R}$ defines the weights, where $w((i,j))$,

the weight of the connection between neuron i and neuron j, is shortened to $w_{i,j}$.

Each artificial neuron can be described by four elements:(1) weights, (2) bias, (3) propagation function and (4) activation function. Each weight represents the importance of the input value. The weight is a learnable parameter. The propagation function converts the vector input to a scalar value. According to [28, p. 35], this function is defined as

Definition 4 (Propagation Function)

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of neurons, such that $\forall z \in \{1, ..., n\} : \exists w_{i_z,j}$. Then the network input of j, called net_j is calculated by the propagation function f_{prop} as follows:

$$net_j = f_{prop}(o_{i1}, ..., o_{i_n}, w_{i1}, ..., w_{i_n,j}) \tag{2.22}$$

net_j can, for example, be calculated as the weighted sum of the input signals

$$net_j = \sum_{i \in I} (o_i \cdot w_{i,j}) + \Theta_j. \tag{2.23}$$

The bias Θ, also referred to as offset or threshold, is an extra input to neurons that stores the value of 1. It quantifies the output of a neuron and is uniquely assigned to each neuron. The activation function defines if the node will be activated ("fired") or not and determines the output of the node. It can be defined as [28, p. 36]

Definition 5 (Activation Function)

Let j be a neuron. The activaiton function is defined as

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \Theta_j). \tag{2.24}$$

It transforms the network input net_j, as well as the previous activation state $a_j(t)$, with the treshold value playing an important role. The activation value a_j is the output of the neuron j.

There exist different types of activation functions, for example binary step functions, linear

and non-linear functions. The sigmoid function is a common non-linear activation function and is defined as (cf. [10, p. 228])

$$\frac{1}{1 + exp(-x)}, \tag{2.25}$$

which maps to the range of values of $(0, 1)$.

The simplest form of an artificial neural network is called perceptron and was developed by Frank Rosenblatt in 1958. Originally, a perceptron consists of only one artificial neuron. The abstract model of this simple perceptron, also called ***s**ingle **l**ayer **p**erceptron* (SLP), is shown in Figure 2.9.

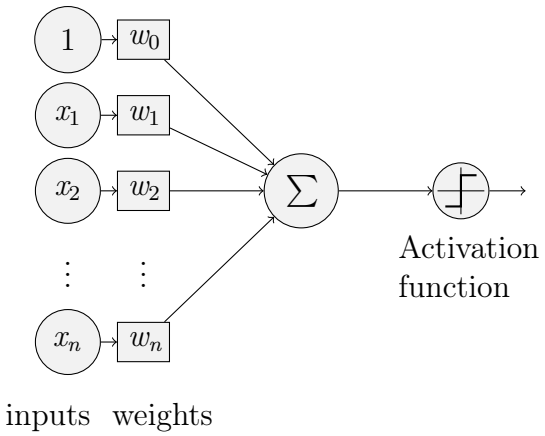


Figure 2.9: Single Layer Perceptron architecture. It has n inputs $x_1, ..., x_n$ with weighting $w_1, ..., w_n$ for $n \in \mathbb{R}$ and one output.

As described in Figure 2.9 a neuron first calculates its output as a weighted sum of the input signals x. The output is then passed to an activation function which calculates the output of a neuron. A single layer perceptron only represents a linear classification, but often more complex classifications must be performed, so several perceptrons can be combined to a network. The perceptrons are first grouped into layers, which are then connected to the perceptrons of the following layer. The result is a so-called ***m**ultilayer **p**erceptron* (MLP) (cf. [10, p. 229]) which is illustrated in Figure 2.10.

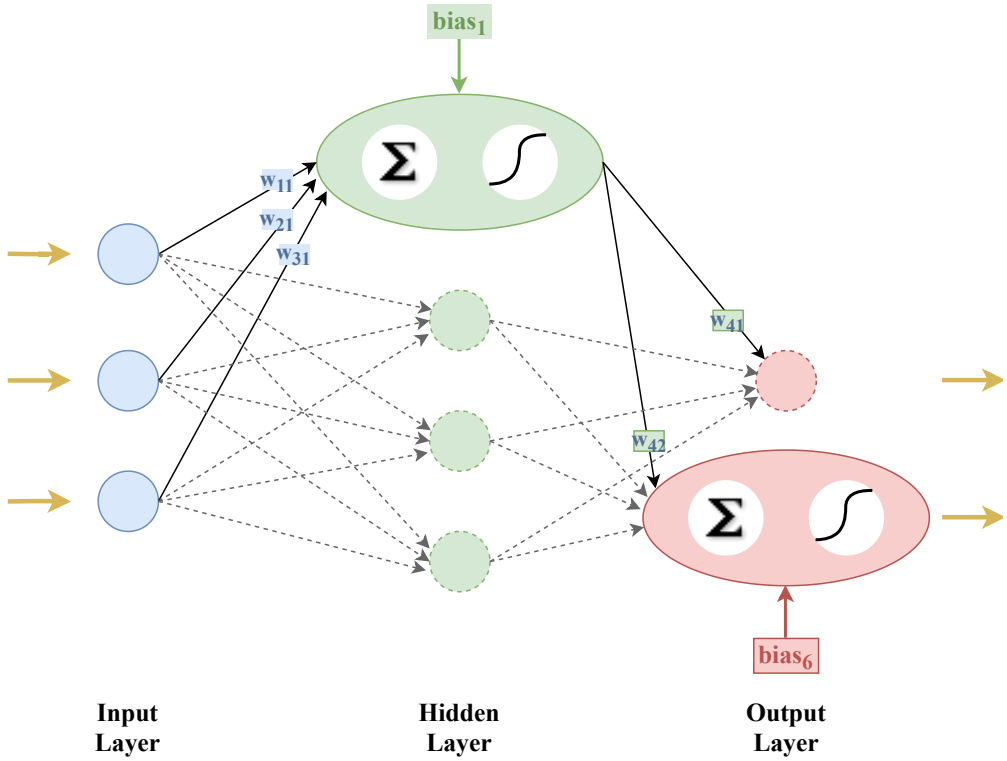


Figure 2.10: Multi Layer Perceptron or feed forward network. The information flow is forward directed from the input neurons via the hidden neurons to the output neurons.

There exist three different types of nodes: (1) input, (2) hidden and (3) output nodes. Input nodes receive initial data for the neural network and output nodes produce the corresponding result. Hidden neurons are located between the input and output neurons and reflect internal computations. If a neural network has only one hidden layer it is also referred to as a shallow neural network. The different artifical neurons are connected to each other by edges having particular weights which represent the impact to the next layer and thus the knowledge itself. The number of neurons, layers and connections determine the complexity, also called depth, of the neural network. So, if these numbers increase the computing power required for training and operation does, too. Neural networks can have a variety of different structures. Besides the already presented options there also exist recurrent networks where information can pass through certain neuron connections of the network backwards and then again forwards. The word embeddings used in this thesis are trained using feed forward shallow neural networks.

Neural networks learn by iteratively comparing the model predictions with the actual observed data and adjusting the weighting factors and the biases in the network so that in each iteration the error between model prediction and actual data is reduced. The model error is quantified by a loss function, for example ***M**ean **S**quared **E**rror* (MSE). The minimization can be performed by the gradient descent optimization algorithm or the back propagation method. Further information about the training process can be found in [10, 232ff] and [28, 58ff]. The network is trained using a training set until a minimum error value ϵ is reached. The size of ϵ depends on the type of problem and the desired accuracy. A training cycle is called epoch.

The neural network model which learns both the text representation and the probabilistic model can be described as follows, taken from [9]:

1. Associate with each word in the vocabulary a distributed word feature vector.

2. Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence.

3. Learn simultaneously the word feature vectors and the parameters of that probability function 2.7.

The preceding words are provided to the network as input. The model attempts to update its parameters so that the probability of predicting the word w_i at each time step is maximized (maximum likelihood method). The probabilities are logarithmized so that the log-probability (log-likelihood) of the corpus is maximized [9].

Word2Vec, Doc2Vec and FastText are popular word embedding methods. The word2vec model was developed by Minkolov et al. [37] in 2013 at Google and represents words as vectors in a latent space of N dimensions. Doc2Vec was developed in 2014 as an extension of word2vec for documents [32]. This model takes into account the entire document structure in addition to the word level and evaluates it accordingly. In contrast, the FastText model

published by Facebook in 2016 works on the character level and generates vectors based on pairs or sequences of letters, so-called n-grams [12].

2.4.5 Word2Vec

There exist two different flavors of word2vec: (1) ***c**ontinuous **b**ag-**o**f-**w**ords* (CBOW) and (2) skip-gram. Figure 2.11 shows the embedding idea and Figure 2.12 their architecture.

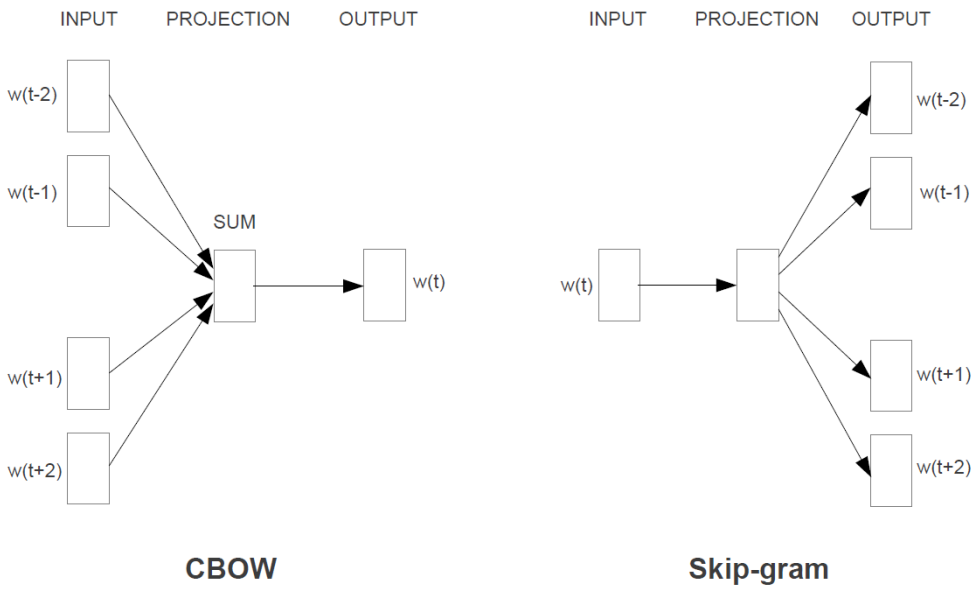
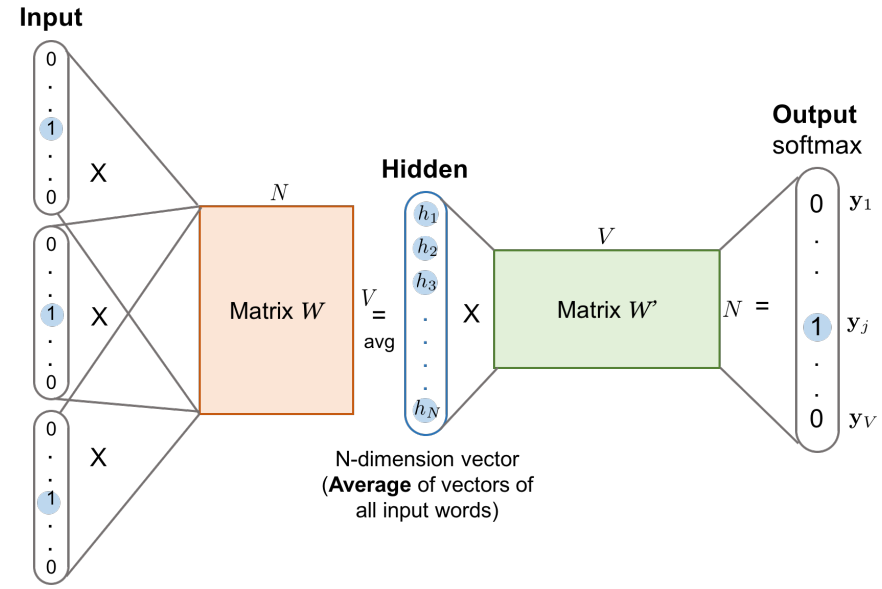


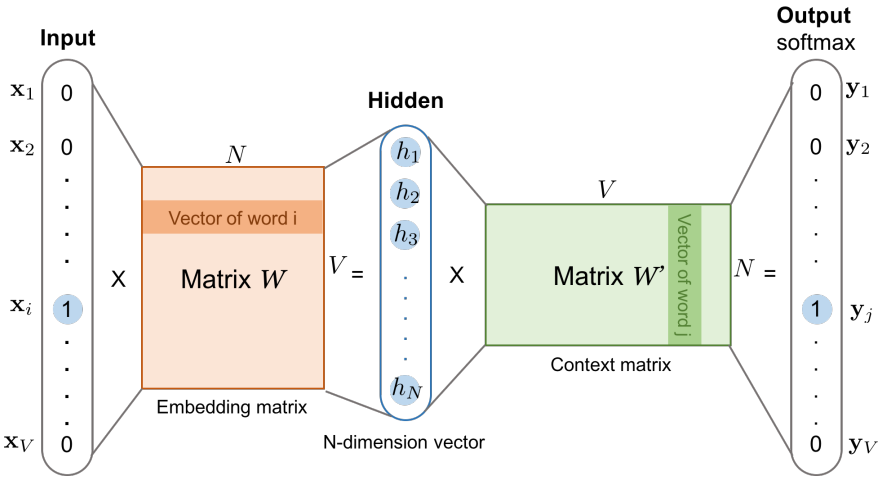
Figure 2.11: Architecture of the CBOW and skip-gram model. Image taken from [37, p. 5].

In CBOW a focus word $w(t)$ is predicted based on the context $w(t \pm 1..n)$. The number of words used as context is determined by the so-called window of size n. For example, if *{'The', 'quick', 'brown', 'fox', 'over', 'the', 'lazy', 'dog'}* is given as context the task is to generate the center word *'jumped'*. In contrast to the CBOW algorithm, the skip-gram algorithm is used to predict the context from a given word. So, given a corpus, it is iterated over every word in the corpus. Let w_t be the word at position t. All words included in a context window of size n around w_t are considered as context. The main question is how likely it is to find w_v in the context of w_t. As can be seen in Figure 2.12, word2vec is a shallow neural network whose input is a text corpus containing V terms and whose output is a set of V feature

vectors.



(a) CBOW



(b) Skip-gram

Figure 2.12: Architecture of the Word2Vec model. Image taken from [46].

The input vector is a one-hot encoded word representations. The number of hidden layer nodes corresponds to the desired length of the resulting vectors N. The input to hidden layer connections can be represented by an unknown word embedding matrix W of size $V \times N$. The i-th row of W represents the word embedding of the word w_i. For the CBOW model, the resulting vectors obtained from multiplying the word vectors of multiple context words with W are averaged. The activation function of the hidden layer is the identity. The connections from hidden layer to output layer can be described by a matrix W' of size $N \times V$ which is unknown, too. Each column of W' matrix represents a word from the vocabulary. The row vectors of this matrix, on the other hand, do not capture the individual words, but the context in which they occur. Therefore W' encodes the meaning of the words. The matrices W and W' are initialized randomly and their optimal values are learned by training the model. The activation function for the output layer is a softmax

$$y = Softmax(W'^T W^T x). \tag{2.26}$$

The ouput of the neural network has to be compared against the pre-defined targets. So the overall loss is computed after each training phase.

CHAPTER 3

Text Document Clustering

In this section, the cluster analysis is explained, which is used to detect hidden patterns in data. Since clustering is an unsupervised learning method, labels are not known a priori and thus have to be learned by the learning algorithm itself (cf. [50, p. 2]). In general, clustering is defined as *"Given a set of data points, partition them into a set of groups which are as similar as possible."* [3, p. 2]. The goal is to partition a set O of n uniformly represented objects into k groups (clusters) $C \subseteq O$ of similar objects. In text document clustering the data objects are usually represented as feature vectors in a N-dimensional vector space $O = \{\vec{o}_1, \ldots, \vec{o}_n\} \subseteq \mathbb{R}^N$. The majority of clustering methods partition a set O into k disjoint clusters $C_1, ..., C_k$ satisfying the following three conditions:

$$\begin{aligned} C_i &\neq 0 \ \forall \ 0 \leq i \leq k \\ C_i \cap C_j &= \{\varnothing\} \ \forall \ i \neq j, 1 \leq i, j \leq k \\ C_1 \cup C_2 \cup ... \cup C_k &= O \end{aligned} \tag{3.1}$$

The conditions ensure that each cluster contains at least one element and that each object must be assigned to a cluster. Each cluster $C_i, 1 \leq i \leq k$ is represented by a cluster center (centroid, prototype) $c_i \in \mathbb{R}^N$ which is usually the mean value with respect to all data

objects contained in C_i

$$c_i = \left(\frac{1}{|C_i|} \sum_{o \in C_i} o_1, ..., \frac{1}{|C_i|} \sum_{o \in C_i} o_n \right), \forall o \in O. \tag{3.2}$$

The basic idea of all clustering methods is that objects within a group are more similar to each other (homogeneity/intra-cluster similarity) while they are less similar to objects in other clusters (heterogeneity/inter-cluster similarity) and is shown in Figure 3.1.

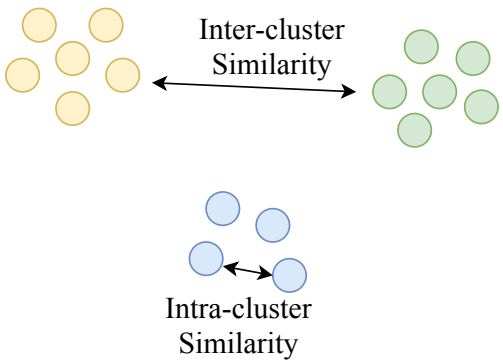


Figure 3.1: Inter-cluster and Intra-cluster similarity of clusters.

Clustering can be applied to objects of different granularities and types, for example documents, paragraphs, sentences or terms in regard to text document clustering. So, the goal overall clustering goal is to determine: (a) the number of clusters, (2) their position, (3) the size of the clusters, (4) their shape and finally (5) their density. The cluster analysis steps are described in figure 1.2.

After pre-processing, all data has been cleaned and transformed to feature vectors. Based on the generated term-document matrix the actual data mining process, in this case clustering, is carried out. For this it is necessary to determine a proximity measure first which quantifies the similarity or difference between two data objects (here documents) and provides a real-valued distance. Different proximity measures can be used for cluster algorithms. All measures refer to the feature values but they consider different properties of the feature vector. There is no optimal proximity measure, since the usage depends on the task [23,

p. 49]. Furthermore, the different methods vary in terms of the scale of measure. In [36] were introduced four scales of measurement: nominal, ordinal, interval, and ratio ([36]). This taxonomy is still widely used to describe the characteristics of a variable. More precisely, the level of measurements indicates if the difference of variables is *"arbitrary, relative, or equidistant"* [36]. The four levels are hierarchical ordered, with the nominal scale as the "lowest" and the ratio scale as the "highest" one. The higher the scale level, the more precise statements can be made about the variables. The following relationship applies:

		Property	Example
Categorical	Nominal	$=, \neq$; Data can be categorized	Gender
	Ordinal	$<, >$; Data can be ranked	Grades
Quantitative	Interval	Compare differences	Temperature
	Ratio	Compare relationship	Revenue

Table 3.1: Level of measurements (Steven's topology) [36]

Section 4 provides further information on a variety of distance and similarity measures depending on the level of scale. According to [11, p. 404] *"It is recommended to apply serveral different similarity measures and check the stability of clustering results."* because different measures lead to different results.

Then a clustering method is selected, which assigns the data objects to clusters based on their similarity values until a termination condition is reached. After the research in cluster analysis started in 1894, many different algorithms were developed (cf. [50, p. 3]). As stated in [47, pp. 307–310] and [25, pp. 274–275], this family of clustering methods can be classified into a wide variety of different types, for example:

- Exclusive vs. non-exclusive

  In exclusive clustering each data object can only be partitioned into one cluster. If the data objects have significant overlap they can be clustered such that they belong to one or more clusters. This is called non-exclusive clustering.

- Fuzzy vs. non-fuzzy

  This aspect relates to the overlapping or non-overlapping characteristic of the resulting cluster distribution. Fuzzy clustering, also referred to as soft clustering is one general type of clustering in which each data object can potentially belong to more than one cluster, depending on an assigned probability value $v_i \in [0, 1], i = 1, ..., k$. On the other hand, in non-fuzzy (hard) clustering each data point belongs to exactly one cluster which leads to non-overlapping clusters. If a data object belongs to a particular cluster, its value is 1, otherwise 0.

- Intrinsic vs. extrinsic

  Intrinsic analysis is intended to uncover unknown structures of the available variables. Before the analysis, both the number of features to be extracted, their significance and thus the cluster labels are unknown. In other words, this is an unsupervised learning method. In contrast, in extrinsic analysis, a concrete idea of cluster labels already exists. Using this type, the predefined structure should be analyzed to determine its consistency with the existing data.

- Constructive vs. iterative improvement clustering

  The clustering methods can be constructive which means that the decision in what cluster each object belongs cannot be change afterwards. Iterative improvement clustering starts with an initial clustering and moves data objects around to improve clustering.

- Monothetic vs. polythetic

  This categorization refers to the cluster membership. In a monothetic scheme the cluster membership is based on the presence or absence of sequentially features. If an approach is polythetic, the clustering decisions are based on the distance between the data objects. Therefore more than one feature is used to divide the collection into groups. Most algorithms are polythetic.

- Hierarchical vs. flat (partitional) clustering

  Hierarchical methods produce a nested sequence of partitions, while partitional approaches produce only a single partition. For a detailed description of the sub-types see section 3.1.

- Type of clustering algorithm

  Every algorithm is based on a different set of rules to determine the similarity between data objects. A distinction is made between: (1) distributed, (2) density, (3) connectivity and (4) centroid based models. The basic idea of distribution based algorithms is that the data generated from the same distribution belongs to the same cluster. The probability that a point belongs to a cluster decreases the further the point is located from the cluster center. Density-based methods assume that the data which is in the region with high density of the data space is considered to belong to the same cluster. So, the more dense the objects the more related they are. Centroid-based, also know as prototype, clustering is an iterative hard clustering algorithm using a similarity function to measure the closeness between a data object and the centroid of the clusters. Commonly, the number of clusters must be defined beforehand. The core idea of connectivity based methods, also known as hierarchical clustering, is that data objects closer to each other are more related than the ones lying farther away. This is similar to centroid-based clustering but instead of pre-defining clusters the clusters are described by the maximum distance needed to connect parts of the cluster.

As mentioned in [25, p. 268] *"There is no clustering technique that is universally applicable in uncovering the variety of structures present in multidimensional data sets."*. Thus the clustering method must be appropriate to the area of application.

3.1 Clustering Algorithms

According to [25, p. 274] the different clustering approaches can be described with the help of a hierarchy as shown in Figure 3.2.

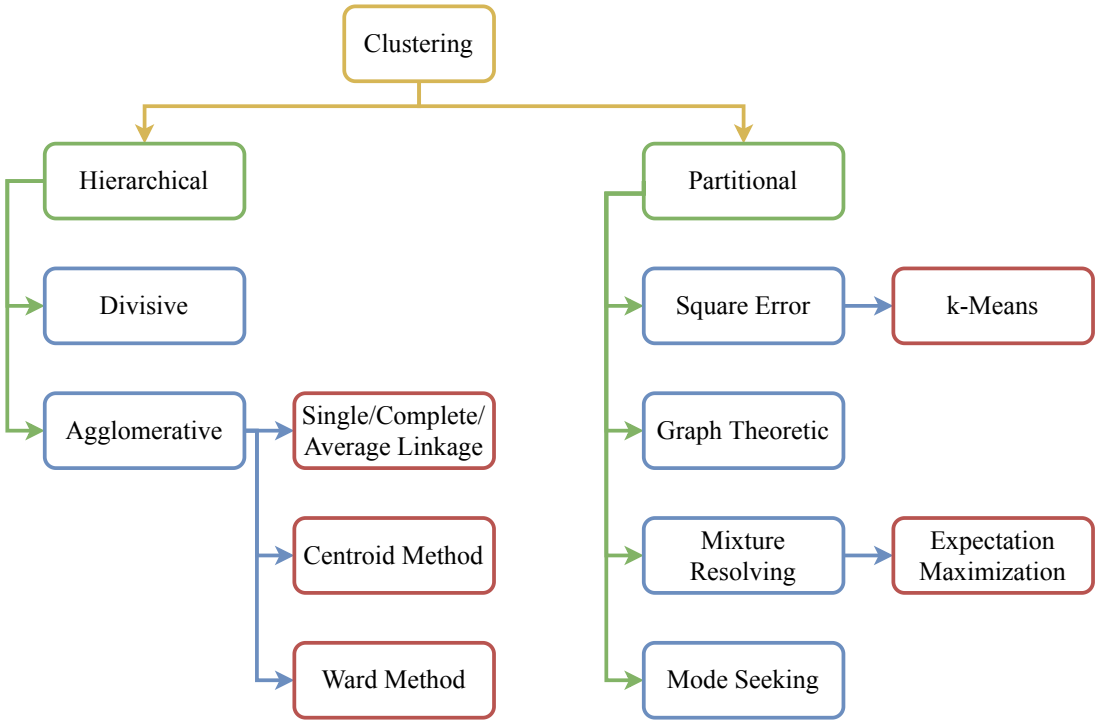


Figure 3.2: Clustering approaches. Image based on [25, p. 275].

As described above there exist two general methodologies: (1) hierarchical algorithms and (2) partitioning-based algorithms (cf. [25, p. 274]). Hierarchical clustering or ***h****ierarchical* ***c****luster* ***a****nalysis* (HCA) seeks to build a nested series of partitions (hierarchy), while partitional clustering creates only one partition. Within each of the types there exist different algorithms for finding the clusters.

3.1.1 Hierarchical clustering

This method creates a hierarchical partition of a given data set without specifying k. A hierarchy of O is a family $H = \{H_1, H_2, ..., H_c\}$ of c clusters. Let C_i and C_j be subsets of H_m and H_l with $m > l$. Then either one subset contains the other one entirely or they are disjoint: $C_i \in C_j$ or $C_i \cap C_j = \varnothing \; \forall i \neq j$ and $m, l = 1, ..., c$. The overall process can be described as:

Parameters	Distance or similarity function
Goal	Determine a tree based hierarchical taxonomy (dendogram).
Types	Divisive, agglomerative

There are two different hierarchical clustering methods, which differ in the proximity measure they use and their algorithmic procedure (top-down/bottom-up). Once clusters have been formed, they can no longer be changed. Applying HCA leads to an iteratively build dendrogram (Figure 3.3) by either merging (bottom-up approach) or splitting (top-down approach) of the data set.

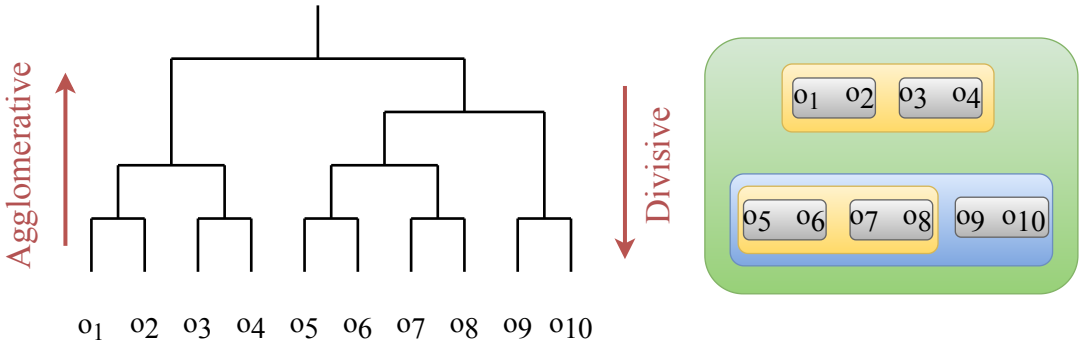


Figure 3.3: Ten data points $(o_1, ..., o_{10})$ on a 2D plane are clustered. The dendrogram on the left side shows the clustering result.

Every merging step is represented by the combination of two vertical lines into one and the division step by the splitting of a vertical line into two vertical lines. The horizontal scale represents the clusters and the height (vertical scale) of the dendrogram indicates the order in which the clusters were joined and the similarity between the two clusters.

Agglomerative clustering algorithm

The agglomerative method, also known as the bottom-up HCA method, is more common in practice and is suitable whenever the data has a hierarchical structure which should be recovered. The algorithm (see 3.4) starts with each data object forming a separate group. Thus, the agglomerative methods have perfect intra-cluster homogeneity at the beginning of the iteration process. Then the distance matrix D is determined using a similarity measure (see chapter 4). D is a symmetric matrix with the diagonal values being 0. In an iteration step two most similar clusters are combined until all data objects are finally merged into a single cluster. After every merging step the distance matrix has to be updated by deleting the row and column corresponding to the combined clusters and adding a row and column for the combined cluster. The inter-cluster distance can be calculated using different linkage criteria. The result is a reverse partition sequence $P_n, P_{n-1}, ..., P_1$.

Algorithm: Agglomerative Clustering	
Init:	
n	{Data objects}
$k = n$	{Clusters}
While $k \neq 1$	
Compute the similarity between each of the k clusters	
Join the two most similar clusters	
$k := k - 1$	
End:	
n	{Data objects}
$k = 1$	{Cluster}

Figure 3.4: Agglomerative clustering approach (cf. [25, p. 277])

The key step is the selection of the clusters that will be merged. The distance between two clusters (inter-cluster distance) can be calculated using different methods. In single link clustering the clusters with the two data points having the smallest distance will be merged. The distance is defined as:

$$d(C_1, C_2) = min_{o_1 \in C_1, o_2 in C_2} d(o_1, o_2) \tag{3.3}$$

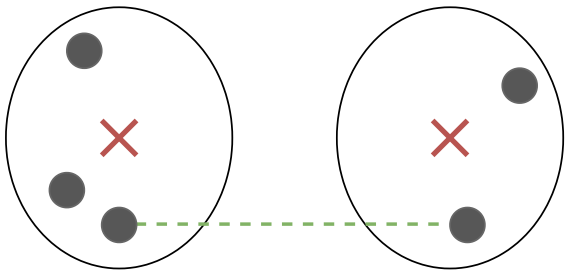


Figure 3.5: Single Linkage

In complete linkage clustering, the distance between two clusters is defined as the maximum

distance between two points in each cluster.

$$d(C_1, C_2) = max_{o_1 \in C_1, o_2 \in C_2} d(o_1, o_2) \tag{3.4}$$

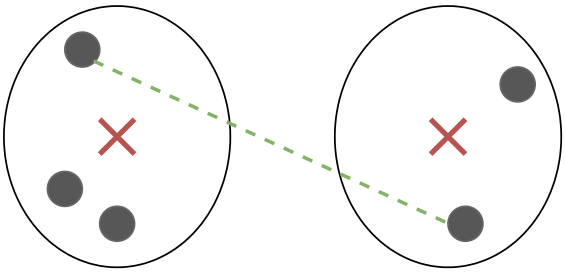


Figure 3.6: Complete Linkage

In average linkage clustering, the average distance between the points from two clusters is defined as:

$$d(C_1, C_2) = \frac{\sum_{o_1 \in C_1, o_2 \in C_2} d(o_1, o_2)}{|n_{C_1}||n_{C_2}|}, \tag{3.5}$$

where n_{C_1}, n_{C_2} represent the number of data objects within C_1, C_2.

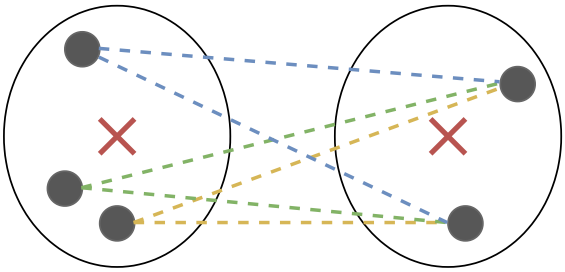


Figure 3.7: Average Linkage

Using the centroid method, the centroids c_i of all clusters are first determined and then the distance between them is calculated in each iteration steps.

$$d(C_1, C_2) = d(c_1, c_2) \tag{3.6}$$

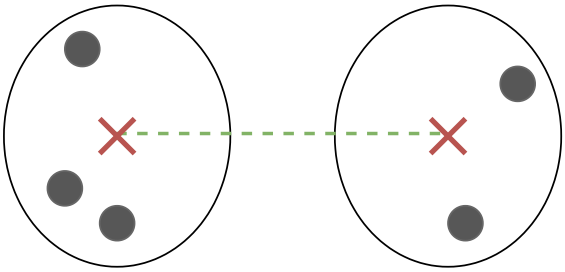


Figure 3.8: Centroid Method

Divisive clustering algorithm

A cluster hierarchy can also be generated top-down. This hierarchical clustering method is called divisive clustering and starts with a single cluster containing all data objects. Then, splits are performed in every iteration until a separate cluster is assigned to each data object. Consequently, the partition sequence $P_1, P_2, ..., P_n$ is produced. Since there exist 2^n ways of splitting each cluster, heuristics such as k-Means are needed. Due to the needed flat clustering method, divisive clustering is more complex.

Hierarchical Clustering Evaluation

Let n be the number of data objects (here text documents). The memory complexity for hierarchical clustering is $\mathcal{O}(n^2)$. Since the similarity matrix has to be updated in every iteration, the time complexity is $\mathcal{O}(n^3)$ in many cases because there are n steps at which the distance matrix must be updated and searched. For some approaches (single and complete linkage) the complexity can be reduced to $\mathcal{O}(n^2 \log n)$ (cf. [25, p. 293]). The space and time complexity is very high for a huge data set. Another limitation is that no objective function is directly minimized. Moreover merges are final and cannot be undone which prevents global optimization and trouble for noisy data. Another disadvantage is that all different approaches to calculate the similarity have their own disadvantages. Single-link algorithms

for example can capture clusters of different sizes and shapes but on the other hand are sensitive to noise. Complete-linkage methods are less susceptible to noise and outliers but tends to break large clusters.

An advantage is that hierarchical clustering is suitable for data with arbitrary shape and attributes of different types. Furthermore, the generated dendrograms are great for visualization. However, it might be difficult to identify the correct number of clusters by the dendrogram. Another benefit is that no a priori information about the number of clusters is needed. By cutting the dendrogram at a proper level a desired number of clusters can be obtained.

3.1.2 Partitioning Algorithms

As opposed to hierarchical cluster methods, partitioning algorithms generate a flat clustering instead of a dendrogram (Figure 3.3) which has only one granularity level and is shown in Figure 3.9.

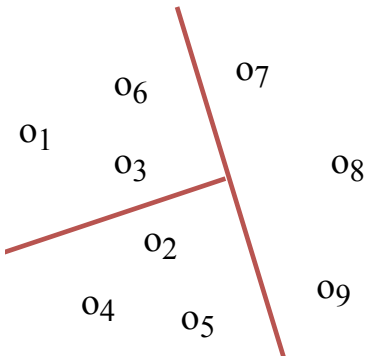


Figure 3.9: Flat clustering is the result of just dividing the data objects into groups without organizing them in a hierarchy.

The k-Means algorithm is a hard cluster algorithm and each data point is deterministically assigned to a specific cluster. The main idea is to define k centroids, one for each cluster. In order to achieve this, the algorithms rearrange an initial non-overlapping cluster setting into k optimal clusters by moving the data objects from one cluster to another such that a certain

homogeneity criterion is optimized. This is usually based on a certain proximity measure and the assumption that k is known or has been estimated in advance. In order to reach a global minimum, an exhaustive search would be necessary, in which all combinations of divisions would have to be considered. Since this is practically impossible, iterative optimization algorithms are used which vary in terms of their criterion function. Referring to [25, p. 18] the most popular one is the ***s**ummation of **s**quared-**e**rrors* (SSE) (cf. [50, p. 8]):

$$\min_{\{c_i\},1\leq i\leq k} \sum_{i=1}^{k} \sum_{o\in C_k} \omega_o \, dist(o, c_i), \tag{3.7}$$

where ω_o is the feature weight of o, k is the number of pre-defined clusters C_i with centroids c_i and $dist()$ is the distance between a data object o and a centroid c_i. The distance function can be optionally chosen. For the squared Euclidean distance (L_2) the so called k-Means clustering algorithm is obtained.

k-Means

The k-Means algorithm is *"one of the oldest and most widley used [...] prototype-based, simple partitional clustering algorithm"* [50, p. 7]. The pseudocode based on [25, 279f] is shown in algorithm 1.

Algorithm 1 k-Means

Input Data set $O = \{o_1, o_2, ..., o_n\}$, total number of clusters k
Output Cluster center dividing O into k clusters

1: Fix a subset of $k << n$ centroids among the given data objects
2: **while** Stopping criteria not fulfilled **do**
3: **for all** Data objects $o \in O$ **do** ▷ *Assignment step:*
4: Find closest center $c_i \in C, 1 \leq i \leq k$ using euclidean distance $||o - c_i||^2$
5: Assign data object o to cluster C_i
6: **end for**
7: **for all** Cluster centers $c_i \in C$ **do** ▷ *Compute Centroids:*
8: Set c_i as defined in 3.2
9: **end for**
10: **end while**

The algorithm has four steps. For an effective k-Means clustering the number of clusters has to be specified in advance. There have been developed different approaches to find the k-value which are explained in section 3.1.2. Then k data objects of the data set are randomly selected and initially defined as centroids. Due to the random choice of initial cluster setting, the algorithm gives varying results on different runs. There exist several algorithms which are described in section 3.1.2 to determine a suitable initialization. Once they have been determined every data object $o_i \in O, 1 \leq i \leq n$ is assigned to a cluster to which it has the greatest similarity. The similarity between o_i and the cluster centers $c_i \in C, 1 \leq i \leq k$ is for the k-Means algorithm implicitly defined as the Euclidean distance

$$|\vec{o_n} - \vec{c_i}| = \sqrt{\sum_{j=1}^{N} (o_{n,j} - c_{i,j})^2}, \tag{3.8}$$

where N is the number of features of the data objects. In text document clustering N is the total number of unique tokens occurring in the document corpus. However, other distance measures can also be used. Afterwards the cluster centers are redetermined. A centroid represents the mean value with regard to all the points contained in $C_i, 1 \leq i \leq k$ (see 3.2). The last two iteration steps are repeated until the mapping process becomes stable, until the defined number of iterations is reached or until a stopping criterion for example if the ratio between the decrease and the objective function is below a threshold is reached. Stability in this context means that the centroid of a cluster in an actual iteration is identical to the centroid of the last iteration.

The advantage of this method is, that it is easy to implement and is able to identify unknown groups of data from complex data sets. According to [50, p. 8] *"[...] K-Means is very simple and robust, highly effcient, and can be used for a variety of data types."*. Another benefit is that it guarantees convergence by trying to minimize the total SSE as a criterion function over a number of iterations. Furthermore it is variable, for example that a created cluster can be altered again. This is not possible for agglomerative approaches because there clusters can

only be extended by objects. Moreover k-Means is linear in iterations i, number of clusters k and number of vectors n and therefore has an overall linear time complexity of $\mathcal{O}(i * k * n)$ ([25, p. 293]). So, k-Means is more sufficient especially for large data sets than hierarchical algorithms. However, there are also disadvantages. One limitation is that the algorithm can end up with a singleton cluster if an outlier is chosen as an initial centroid. Furthermore outliers do not fit well into any cluster. Another reason why outliers have a great influence on the result is because all objects are included in the calculation of the centroid. Nevertheless, *"Some disadvantages of K-Means [...] are dominated by the advantages [...]"* [50, p. 8].

Choosing k

The k-value cannot be calculated using the k-Means algorithm. The user must initially determine this value. The values of k can be determined using statistical measures, visualization or a neigbourhood measure (cf. [39, pp. 104–107]). The most well known technique is called elbow method. The algorithm is ran for different k values and the SSE is calculated and graphed. The point where the function appears to become flat also referred to as "elbow" is chosen as the optimal value of k. Another option to determine the k value is the average silhouette method. The silhouette index is a clustering evaluation method which estimates the average distance between clusters and is described in section 3.1.4. The idea is to calculate the average silhouette coefficient for different values of k (see 3.23), plot the resulting curve and choose the k value which maximizes the curve. Further options are described in [39].

Choosing Initial Centroids

Whenever the initial cluster setting is chosen randomly, different runs of k-Means produce different total SSEs. Therefore, choosing suitable initial centroids is the key step of k-Means. The idea is to place the initial cluster centers optimally, preferably as close as possible to the optimal centroids, in order to minimize the required iterations and thus the required

computation time. One option is to run the algorithm several times with different random initial cluster points and then choose the result with the lowest SSE. Instead of randomly selection, hierarchical clustering can be used to cluster a sample of points. Then k clusters are extracted and used as initial centroids. A commonly used approach is called k-Means++[1]. According to [8, p. 3], the steps involved are:

1. Randomly select the first centroid c_1 from the data points X.

2. Select $x \in X$ as a new center c_i with probability

$$\frac{D(x)^2}{\sum_{x \in X} D(x)^2}, \tag{3.9}$$

   where $D(x)$ denotes the distance each data object to its nearest, previously choosen centroid.

3. Repeat step 2 until k centroids have been chosen.

4. Continue as the standard k-Means algorithm (1) does.

3.1.3 Visualize Cluster Result

In order to evaluate the cluster result, the grouping must be presented in a structured way. There are several ways to do this. A dendrogram for example is a tree-like structure frequently used to illustrate the arrangement of the clusters produced by hierarchical clustering. This method describes the cluster formation process. The distance at which clusters are merged is called the merging level. Another widley used technique is called ***P****rincipal* ***C****omponent* ***A****nalysis* (PCA). PCA is a dimensionality reduction technique primarily used for visualization [10, p. 531]. The goal of PCA is to extract the most important information from the data set $\{x_1, x_2, ..x_N\}$ where each variable has dimension D and express this information

[1]The ++ only refers to the special initialization of the cluster centers. The k-Means algorithm does not change.

in terms of a smaller number of linearly uncorrelated variables, the principal components, having dimensionality $M < D$ preserving as much variance as possible [10, p. 562]. For the visualization of the cluster results, the data is projected into a 2- or 3-dimensional sub-space. If the data should be displayed in two dimensions, the x-axis of the coordinate system normally represents the first principle component and the y-axis the second. The procedure is as follows. First, the origin of the coordinate system is placed in the center of mass of the data points. The next step is to find the direction of the first coordinate such that it points in the direction of the greatest variance of the data points. This direction defines the new axis (first principle component). Then an orthogonal axis to the one just found is defined and rotated around it until the variation along the new axis is maximal. If the data is to be displayed in 3D, an additional axis orthogonal to the second axis must be defined. This axis is rotated around the second axis until it points in the direction of the remaining maximum variation. So, the main components are formed in descending order of importance where the first main component is responsible for the majority of the variations. The result is the rotation of the coordinate system such that the axes have maximal variation along their direction.

Mathematically, the dispersion of the variables can be describes by the variance σ^2 which is defined as follows for one-dimensional data sets X containing N data objects:

$$Var(X) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2, \tag{3.10}$$

where $\bar{x}$ is the expected value, i.e. (cf. [10, p. 562])

$$\bar{x} = \sum_{i=1}^{N} x_i. \tag{3.11}$$

The following applies to multidimensional data objects $x \in \mathbb{R}^{\mathbb{D}}$ in a data set X given as

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,D} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,D} \end{bmatrix}, \tag{3.12}$$

$$Var(X) = \frac{1}{N} \sum_{i=1}^{N} \begin{pmatrix} (x_{i,1} - \bar{x_1})^2 \\ (x_{i,2} - \bar{x_2})^2 \\ \vdots \\ (x_{i,N} - \bar{x_N})^2 \end{pmatrix}. \tag{3.13}$$

The relationship between two data objects is expressed by the covariance. This is a generalization of the variance, with

$$Var(x) = Cov(x, x). \tag{3.14}$$

The covariance matrix CM of a data set X (3.12) is constructed using the following matrix

$$X' = \begin{bmatrix} x_{1,1} - \bar{x_1} & \cdots & x_{1,D} - \bar{x_D} \\ \vdots & \ddots & \vdots \\ x_{N,1} - \bar{x_1} & \cdots & x_{N,D} - \bar{x_D} \end{bmatrix}. \tag{3.15}$$

It follows

$$CM = \frac{XX'^T}{N} = \begin{bmatrix} C_{1,1} & \cdots & C_{1,D} \\ \vdots & \ddots & \vdots \\ C_{N,1} & \cdots & C_{N,D} \end{bmatrix}, \tag{3.16}$$

with

$$C_{i,j} = \frac{1}{N}((x_i - \bar{x_i})(x_j - \bar{x_j})), \forall 1 \leq i, j \leq D. \tag{3.17}$$

The goal of PCA is to find an orthonormal basis B that transforms the coordinate system $Y = BX$ such that the covariance matrix become a diagonal matrix $CM_Y = \frac{1}{N} YY^T$

while keeping the maximum variation. The eigendecomposition of the covariance matrix CM provides the basis vectors of B whose row vectors represent the principle components. The eigenvalues of the covariance matrix will be the diagonal elements of the resulting matrix CM_Y. In the two-dimensional case, the two eigenvectors of CM corresponding to the two largest eigenvalues must be calculated. The computational cost of computing the full eigendecomposition of a $D \times D$ matrix is $\mathcal{O}(D^3)$ ([10, p. 563]).

3.1.4 Cluster Validation

As already described at the beginning, the cluster analysis is mainly influenced by the feature extraction method, the number of clusters, the distance measure and the clustering algorithm. To evaluate the resulting structure, validation procedures are used. As described in [25, p. 268] and [22, p. 123], there exist three types of validation approaches to evaluate the goodness of the clustering result: (1) internal, (2) external and (3) relative procedures. External techniques compare the clustering result with previously defined or determined structures. Internal validation methods focus on the heterogeneity between the clusters and the homogeneity within the clusters and evaluate the results without an external reference. The relative criteria compare the results of different cluster schemes using the same algorithm with different parameter values.

External evaluation criteria

A predefined cluster partition $P = \{P_1, ..., P_m\}$ (ground truth value) will be compared with a division $C = \{C_1, ..., C_n\}$ of the same objects generated by a clustering algorithm. The

proposed evaluation techniques use the following notation, based on [22, p. 126]:

TP :Data objects belong to the same cluster of P and C.

TN :Data objects belong to the same cluster of C and to different ones of P.

FP :Data objects belong to different clusters of C and to the same cluster of P

FN :Data objects belong to different cluster of C and to different ones of P.

The ***R**and **I**ndex* (RI) and the Jaccard coefficient are common evaluation technique and measures similarity between C and P. The RI is defined as

$$RI = \frac{TP + TN}{TP + TN + FP + FN}. \tag{3.18}$$

For the Jaccard coefficient the following applies

$$J = \frac{TP}{TP + TN + FP}. \tag{3.19}$$

The range of these indexes is between 0 and 1. They are maximized if $n = m$ (cf. [22, p. 126]).

Internal evaluation criteria

The cluster division $C = \{C_1, ..., C_k\}$ is evaluated without external cluster information. Two commonly used indices are the Dunn index and the silhouette coefficient. The Dunn index defines the ratio between the minimal inter-cluster distance to maximal intra-cluster distance. The index is defined as [22, p. 130]

$$D = min_{i=1,...,k} \left\{ min_{j=i+1,...,k} \left(\frac{d(C_i, C_j)}{max_{l=1,...,k} diam(C_l)} \right) \right\}, \tag{3.20}$$

where k is the number of clusters contained in the data set, $d(C_i, C_j)$ defines the inter-cluster distance between two clusters C_i and C_j by

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \tag{3.21}$$

and $diam(\cdot)$ denotes the diameter (intra-cluster distance) of a cluster

$$diam(C) = \max_{x,y \in C} d(x, y). \tag{3.22}$$

The Dunn index has a value between 0 and ∞. The clustering result is optimal if the Dunn index is maximized. This indicate the presence of compact and well-separated clusters. The silhouette index measures the average distance between clusters and is defined by

$$S = \frac{1}{n} \sum_{i=1}^{n} S(i), \tag{3.23}$$

where

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{3.24}$$

and

$$\begin{aligned} a(i) &= \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j), \\ b(i) &= min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j). \end{aligned} \tag{3.25}$$

Furthermore, $d(i, j)$ is the distance between two data objects i and j, $a(i)$ describes the average distance between a data object i and all other objects in the same cluster and $b(i)$ the average distance between i and all objects in any other cluster. $S(i)$ is not defined for only one cluster. The range of the values is $[-1, 1]$, whereas 1 represents well-clustered objects. Thus the silhouette index has to be maximizes.

CHAPTER 4

Text Similarity

A key factor for any clustering approach is the proximity measure which characterizes the similarity or dissimilarity (distance) between objects. Those terms are opposite to each other which means that the similarity measure decreases as the distance measure increases. Let a collection containing n documents $O = o_1, ..., o_n$ be given. A value can be assigned to each object pair using a proximity dimension function p: $(o_i, o_j) \rightarrow p(o_i, o_j) \in \mathbb{R}$. A similarity measure is a function which assigns a real number $s_{i,j}$ between 0 and 1 to the text where a score of 1 indicates maximum similarity and can be described as

$$0 \leq s_{i,j} \leq 1 \tag{4.1}$$

$$s_{ii} = 1 \tag{4.2}$$

$$s_{i,j} = s_{j,i}, \tag{4.3}$$

for all $1 \leq i \leq n$ and $1 \leq j \leq N$. The function values $s(i, j)$ can be combined to a symmetrical $N \times N$-matrix which is called similarity matrix.

Similarity Measure

There exist different similarity measures for the scale of measurements described in [36]. The similarity of metric variables can be determined using Cosine similarity or Pearson's correlation coefficient. The Cosine $cos(x, y)$ measures the similarity by calculating the cosine

of the angle between two feature vectors x and y containing N features

$$s(x,y) = cos(x,y) = \frac{\langle x,y\rangle}{||x||||y||} = \frac{\sum_{i=1}^{N} x_i y_i}{\sqrt{\sum_{i=1}^{N} x_i^2}\sqrt{\sum_{i=1}^{N} y_i^2}}. \tag{4.4}$$

The similarity ranges from -1 (opposite) over 0 (orthogonality) to 1 (exactly the same). Cosine similarity is not invariant to shifts whereas the Pearson's correlation coefficient is. The correlation coefficient, also known as ***P**earson **P**roduct-**M**oment **C**orrelation **C**oefficient* (PPMCC), measures the linear correlation between two finite and positive variables that are at least interval-scaled and is defined as

$$Corr(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} = \frac{\langle x - \bar{x}, y - \bar{y}\rangle}{||x - \bar{x}||||y - \bar{y}||} = cos(x - \bar{x}, y - \bar{y}). \tag{4.5}$$

The value of a correlation coefficient ranges between -1 (perfectly negatively linearly related) and 1 (perfectly positively linearly related). A correlation of 0 indicates that x and y don't have any linear relation.

Common similarity coefficients for non-binary data are Simple Matching coefficient, Jaccard coefficient and Sørensen-Dice coefficient. For binary data these coefficients are based on a contingency (matching) table. Let x and y be two binary variables containing N features which can have two states (0:absent, 1: present). The corresponding matching table is shown in 4.1.

		Object y		
		0	1	*sum*
Object x	0	$a = \sum_{j=1}^{N} I(x_k = 0, y_k = 0)$	$b = \sum_{j=1}^{N} I(x_k = 0, y_k = 1)$	$a + b$
	1	$c = \sum_{j=1}^{N} I(x_k = 1, y_k = 0)$	$d = \sum_{j=1}^{N} I(x_k = 1, y_k = 1)$	$c + d$
	sum	$a + c$	$b + d$	p

Table 4.1: Contingency table for binary data

The ***s**imple **m**atching **c**oefficient* (SMC) is defined as

$$s(x, y) = \frac{\text{number of matching attributes}}{\text{number of attributes}} = \frac{a + d}{a + b + c + d}. \tag{4.6}$$

The simple matching coefficient is invariant if the binary value is symmetric. A binary attribute is symmetric if its states have equal importance. The SMC is similar to the Jaccard coefficient which is defined as the number of features where both variables are equal to 1 divided by the number of attributes where either of the two is equal to 1,

$$s(x, y) = \frac{|x \cap y|}{|x \cup y|} = \frac{d}{b + c + d}. \tag{4.7}$$

The closer the Jaccard coefficient is to 1, the greater the similarity of the quantities. The minimum value of the Jaccard coefficient is 0. The Jaccard coefficient is non-invariant if the binary variable is asymmetric. The Dice's coefficient is defined as

$$s(x, y) = \frac{2|x \cap y|}{|x| + |y|} = \frac{2d}{2d + b + c}. \tag{4.8}$$

If the two variables are identical, the coefficient is equal to 1, while if x and y have no attributes in common, it is equal to 0.

Distance Measure

Proximities measure the distance d between two objects x, y in a set by a numerical value. A distance measure is considered a metric if the following four conditions are fulfilled (cf, [23, p. 50]).

Definition 6 (Metric)

Let d be a distance measure between two objects $x, y \in X$. d is a metric on X if and only if

d is always positive:	$\forall x, y \in X, d(x, y) \geq 0$
d meets the identity criterion:	$d(x, y) = 0 \leftrightarrow x = y$
d is symmetrical:	$d(x, y) = d(y, x)$
d satisfies the triangle inequality:	$d(x, z) \leq d(x, y) + d(y, z) \forall x, y, z \in X$

A common distance function is the Minkowski distance or L_p-norm which compares the objects N features and is defined as:

$$d(x, y) = L_q(x, y) = ||x - y||_p = \sqrt[p]{\sum_{i=1}^{N} |x_i - y_i|^p}. \tag{4.9}$$

For $p = 1$ the respective distance function is known as Manhattan (city block) distance

$$d(x, y) = ||x - y||_1 = \sum_{i=1}^{N} |x_i - y_i|. \tag{4.10}$$

The so called Euclidean distance derives from setting $p = 2$

$$d(x, y) = ||x - y||_2 = \sqrt{\sum_{i=1}^{N} |x_i - y_i|^2}. \tag{4.11}$$

The square Euclidean distance follows from the Euclidean distance:

$$d(x, y) = \sum_{i=1}^{N} |x_i - y_i|^2. \tag{4.12}$$

As already mentioned in 3, a precise definition of the proximity between two objects is a precondition for an accurate cluster analysis. The presented similarity and distance measures

are typically applied in clustering and have proven to be useful depending on the task. The question now is what effect these measures have when applied to text documents. The similarity of a pair of documents is not always clear and usually varies depending on the problem. For example, as [23, p. 49] points out, two research papers are considered similar if they are similar in terms of their topics, whereas web pages are considered similar if they have almost the same meaning and conveying the same information. So the similarity can be defined in terms of surface (lexical) and meaning (semantic) closeness. As already discussed in section 2.4 it is depending on the task useful to take the context into account to capture more of the semantics. According to [21, p. 13] string-based algorithms are used to measure lexical similarity and corpus- and knowledge-based algorithms are based on semantic similarity. According to [21, p. 13] string-based algorithms operate on strings and character chains. Character-based measures include Jaro-Winkler and Damerau-Levenshtein measures, for example. The simple matching, jaccard and dice coefficient as well as the cosine similarity and the manhattan and euclidean distance are term based string measurements. Corpus-based similarity determines the semantic similarity according to information gained from a large corpora. A widley used technique is LSA (see section 2.4.3) (cf. [21, p. 14]) which captures the most descriptive features of the document meaning and thus reduces the dimensionality of the document vector space representation. Knowledge-based similarity uses information derived from semantic networks, for example WordNet which is a large lexical database. This approach will not be discussed further in this thesis.

CHAPTER 5

Data sets

The different methods are evaluated on three data sets:

1. *20 Newsgroups*
2. *Jeopardy!* questions
3. *Reddit* comments

All data sets have pre-assigned category labels. This allows external cluster validation techniques to be used. The data sets differ in terms of document size, number of categories and average category size. The smallest contains $18,846$ (*20 newsgroups*) documents in total, the second smallest $39,999$ (*Reddit*) and the largest $349,641$ (*Jeopardy!*). The number of categories are 5 (*Reddit*), 20 (*20 newsgroups*) and $43,369$ (*Jeopardy!*). The characteristics and sources of these data sets are described in the following subsections.

5.1 *20 Newsgroups*

The *20 Newsgroups* data set is a widely used data set in text mining. It can be downloaded from [1] or loaded directly from the *Python* library *scikit-learn* using the following command:

```
from sklearn.datasets import fetch_20newsgroups
# Fetch dataset
# Strip metadata using remove statement
df = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))
```

The data set contains 18,846 non-null newsgroup posts out of 20 different categories. 380 of them are empty and 180 are duplicated posts. The resulting data set thus contains 18,286 entries. Table 5.1 provides an overview of the categories partitioned according to their topic.

comp.graphics (1) comp.os.ms.windows.misc (2) comp.sys.ibm.pc.hardware (3) comp.sys.mac.hardware (4) comp.windows.x (5)	rec.autos (7) rec.motorcycles (8) rec.sport.baseball (9) rec.sport.hockey (10)	sci.crypt (11) sci.electronics (12) sci.med (13) sci.space (14)
misc.forsale (6)	talk.politics.misc (18) talk.politics.guns (16) talk.politics.mideast (17)	talk.religion.misc (19) alt.atheism (0) soc.religion.christian (15)

Table 5.1: *20 Newsgroups* categories sorted according to topics.

For the further processing the corresponding targets (see 5.1) are used. As can be seen in Figures 5.1 and 5.2, the data set is balanced in terms of cluster size.

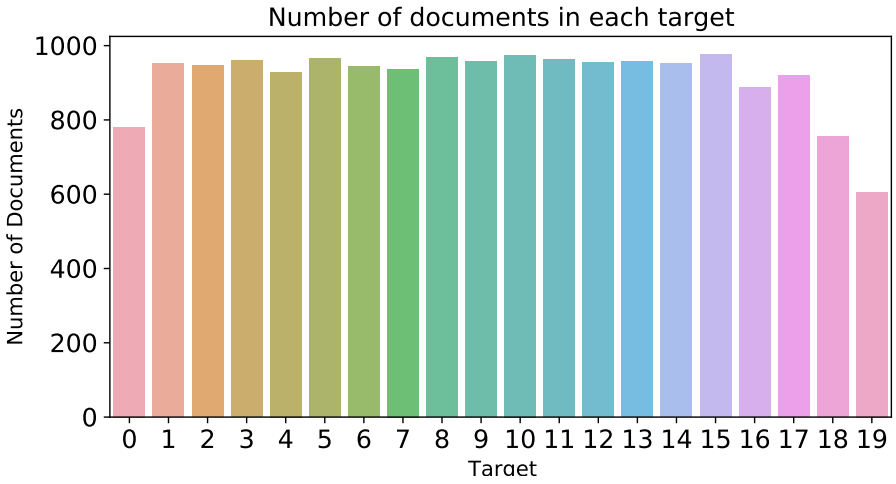


Figure 5.1: *20 Newsgroups* categories.

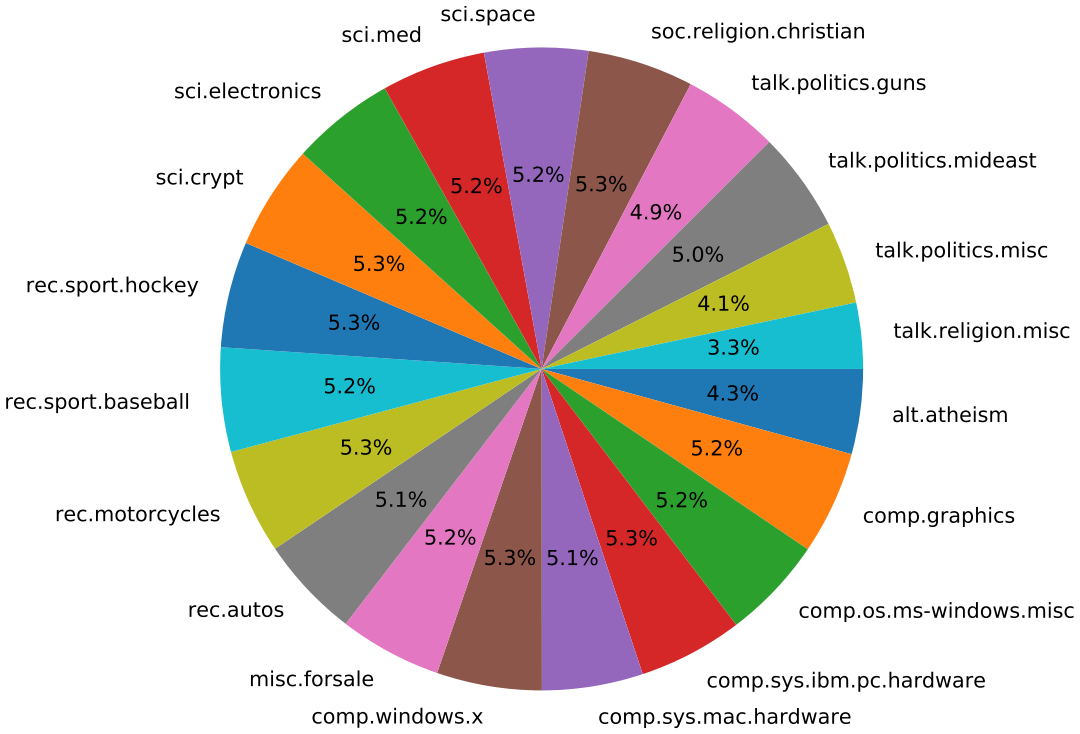


Figure 5.2: Pie chart *20 newsgroups* categories.

The frequency of the categories can also be displayed as a word cloud (see Figure 5.3). The frequency ranges from 606 to 976.

Figure 5.3: Word cloud of *20 newsgroups* categories. The size of each word indicates its frequency or importance.

For the cluster analysis, 5 and 10 categories were clustered. If the specified number of

clusters `n_categ=20` differs from the maximum number (in this case 20), the data objects of the `n_categ` most frequently occurring categories are selected. For 5 categories 4,847 data objects are clustered and for 10 categories 9,629.

5.2 *Jeopardy!*

The *Jeopardy!* data set, taken from [2], includes information about every question asked in season 1 through 35 of the American quiz show *Jeopardy!*, among other things. The data set stored in JSON format contains 348,220 non-null and 1421 duplicate entries out of 43,369 categories. Each entry contains the following information

1. `category`
2. `value`
3. `question`
4. `answer`
5. `round`
6. `show_number`
7. `air_date`

For the unsupervised clustering approach only the questions and their categories are extracted from the JSON file. Some of the categories occur only once, while others occur 855 times. The 10 most frequent ones are shown in Figure 5.4.
For the cluster analysis, the data of 5, 10 and 20 categories is used. These categories and their target values are listed in table 5.2.

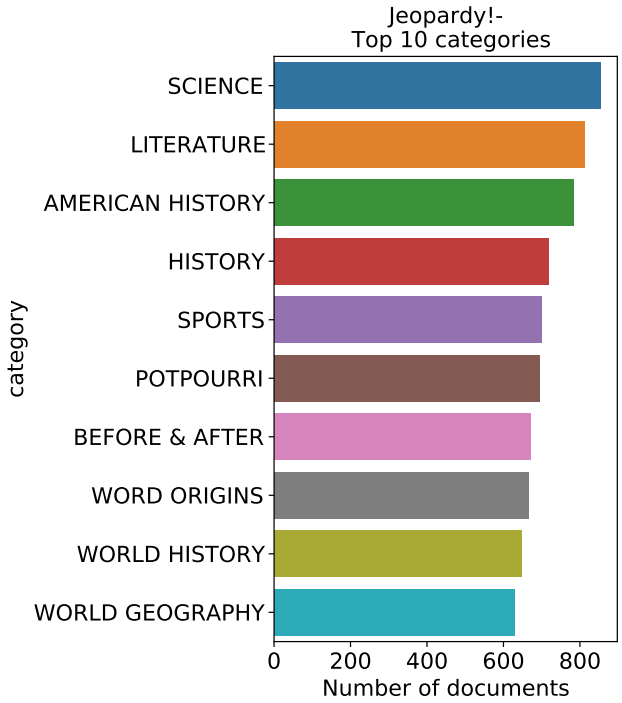


Figure 5.4: 10 most frequent *Jeopardy!* categories.

SCIENCE (0) TRANSPORTATION (12)	LITERATURE (1) POTPOURRI (5) ART (15) WORD ORIGINS (7)	AMERICAN HISTORY (2) HISTORY (3) WORLD HISTORY (8) U.S. HISTORY (19)
SPORTS (4)	BEFORE & AFTER (6)	WORLD GEOGRAPHY (9) U.S. CITIES (11) U.S. GEOGRAPHY (18) WORLD CAPITALS (16)
BUSINESS & INDUSTRY (10) COLLEGES & UNIVERSITIES (13)	RELIGION (14)	ANIMALS (17)

Table 5.2: Category names and their target values for `n_categ=20` of the *Jeopardy!* data set.

The resulting data set for 5 categories contains 3,869 entries, 7,175 for 10 categories and 12,543 objects for 20. As can be seen in Figures 5.5 and 5.6, the data set is balanced in terms of cluster size.

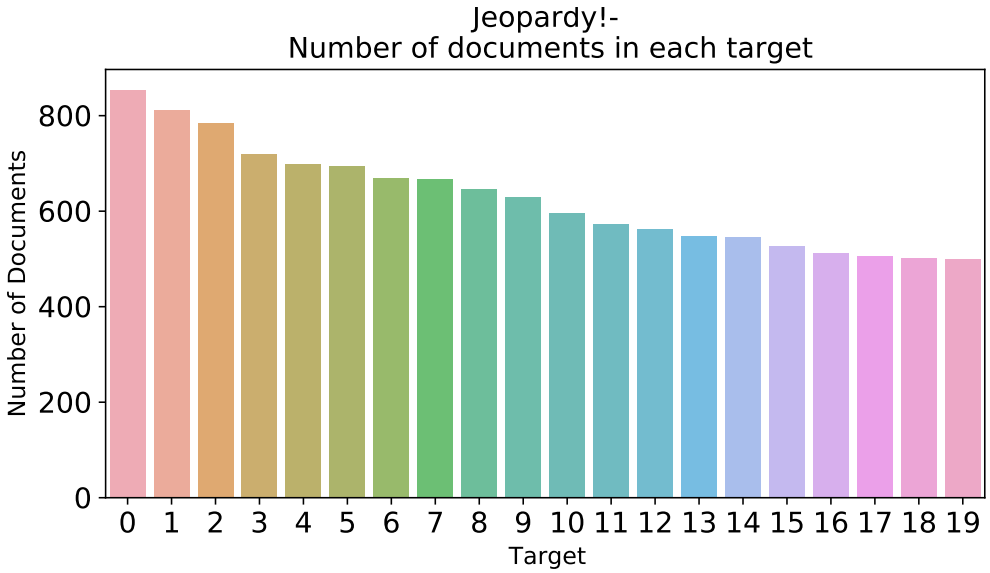


Figure 5.5: Number of documents in each *Jeopardy!* category for `n_categ=20`.

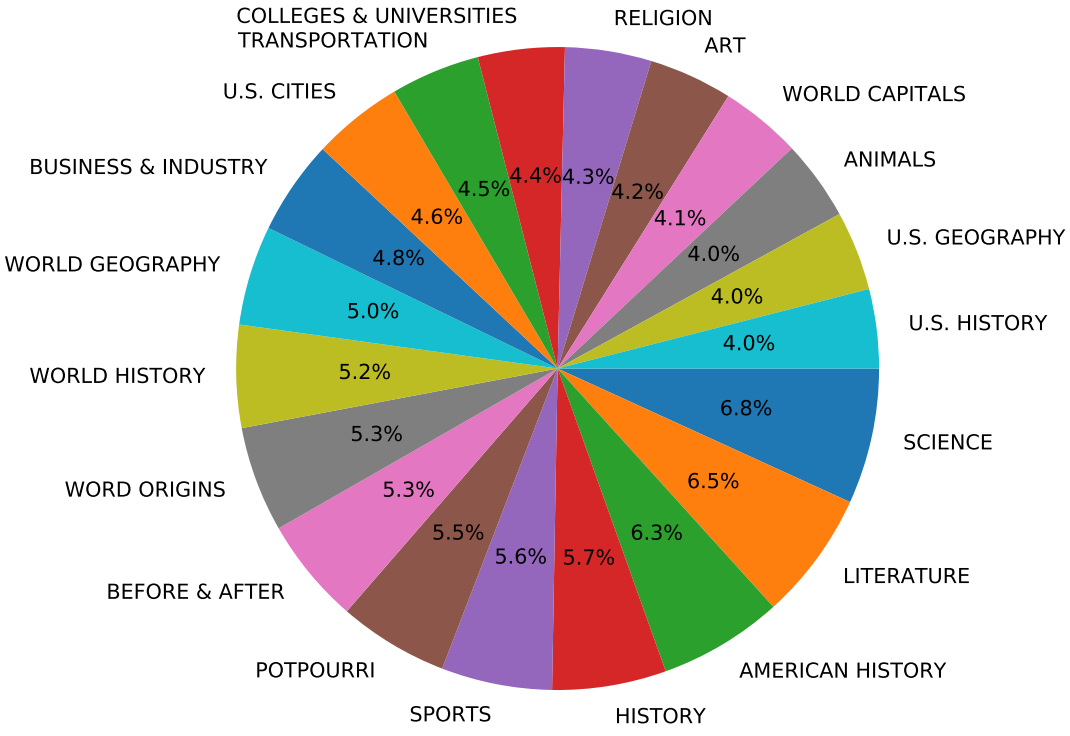


Figure 5.6: Percentage distribution of *Jeopardy!* categories for `n_categ=20`.

5.3 *Reddit* comments

This data set was downloaded from [14] and contains 39,999 parent comments from May 2015 out of 5 subreddit pages. Since 1,753 text entries are duplicates, only 38,245 entries are used for further processing. The file includes the following information:

1. `parent_id`
2. `text`
3. `topic`
4. `length`
5. `size_range`

For this thesis only the `question` and the `topic` information are extracted. The categories and their corresponding target values are `nfl` (0), `pcmasterrace` (1), `news` (2), `movies` (3) and `relationships` (4). The proportion of objects contained in each category is visualized in Figure 5.7 and 5.8.

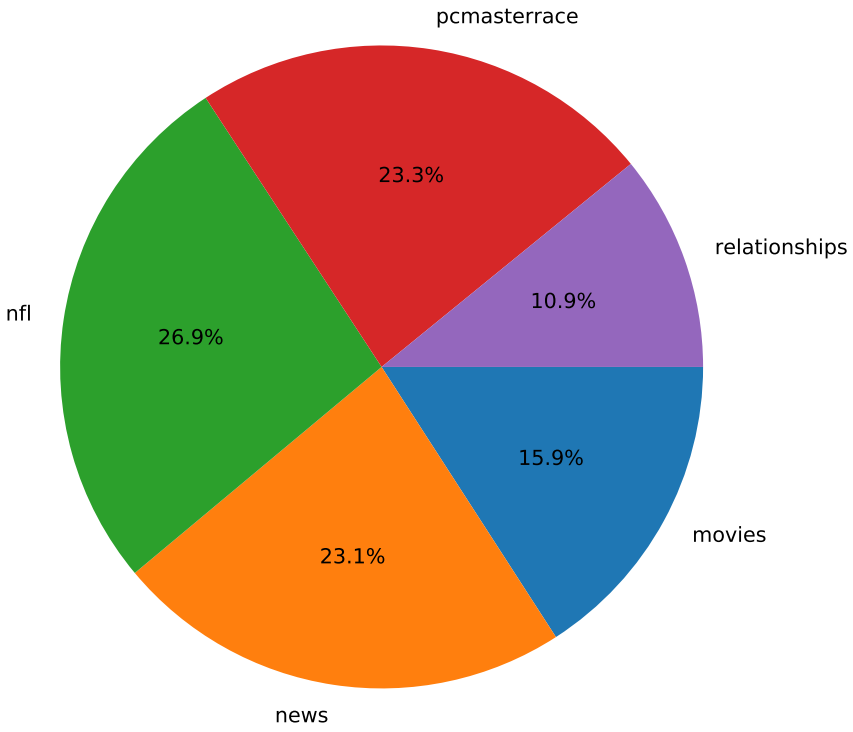


Figure 5.7: Percentage distribution of categories contained in the *Reddit* data set. The frequency of categories ranges from $4,161$ to $10,270$.

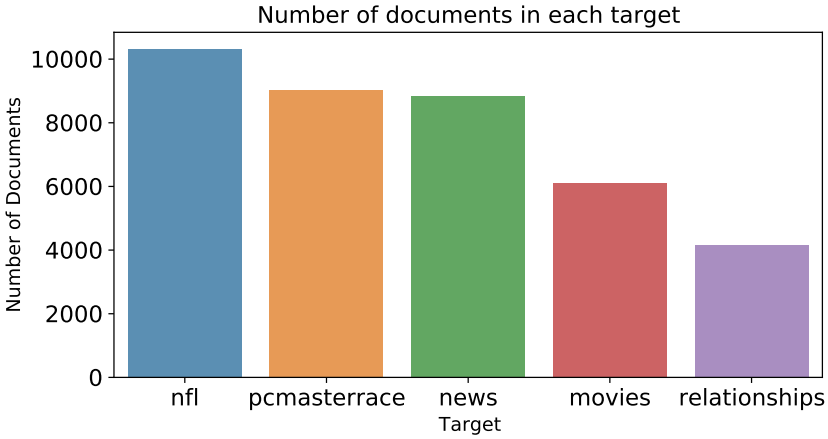


Figure 5.8: Number of documents in each *Reddit* category.

CHAPTER 6

Implementation and Results

In order to execute the KDD process introduced in chapter 2 a *Python3* program was developed using *Jupyter Notebook*. The basic software and libraries used in this thesis are:

Basic Software:	*Anaconda*, *Jupyter Notebook*
Data Pre-Processing:	*pandas*, *spaCy*, *numpy*
Data Transformation/ Data Mining:	*gensim*, *scikit-learn*
Visualization:	*matplotlib*, *seaborn*, *wordcloud*

To unpack the program provided as a zip-file a corresponding program must be available. All relevant files are then unpacked and stored in the following directory structure. In order to run the *Jupyter notebook* files it is necessary to ensure that all required packages are installed. A list of all packages is contained in the file *requirements.txt*. An overview is provided in the appendix (see D.1). In order to install all packages run:

```
1 pip install -r requirements.txt
```

The text document clustering was implemented for the three data sets introduced in 5. The file *clusterAnalysis.ipynb* contains the actual cluster analysis. As can be seen in Figure 6.1 the user has to specify global parameters, first.

Global Parameters

→Please specify a data set:

0: *20 newsgroups*

1: *Jeopardy!* questions

2: *Reddit* comments

```
datasets = 0
```

→Please specify the number of data objects:

The **20 newsgroups** data set contains 18,286 not null and not duplicated data objects, the **Jeopardy** data set 348,220 and the **Reddit** data set 38,245. If n_comp_reduced is set to *False* the total number of data objects of each data set is used. If it is set to *True* n_comp objects will be selected.

```
n_comp_reduced = True
n_comp = 100
```

→Please specify the number of categories:

The **20 newsgroups** data set contains 20 categries, the **Jeopardy** data set 43,369 and the **Reddit** data set 5. If n_categories_reduced is set to *False* the total number of categories is used. If it is set to *True* n_categ categories will be selected.

```
n_categories_reduced = False
n_categ = 4
```

→Please decide if the data set should be cleaned or not:

```
clean = True
if clean:
    clean_annot = "activated"
else:
    clean_annot = "deactivated"
```

→Please specify the metric and the word embedding:

The following metrics are implemented: - manhattan - euclidean - cosine - jaccard - matching - overlap

The following BoW representations are implemented: - Frequency Vectors - One-Hot-Encoding - TF-IDF - LSA

The following word embeddings are implemented: - Word2Vec CBOW - Word2Vec skip-gram - Word2Vec pre-trained - Doc2Vec pre-trained - FastText pre-trained - GloVe pre-trained

```
# Metric options
# metrices = [ "manhattan", "euclidean", "cosine", "jaccard", "matching",␣
 ↪"overlap", "ppmcc"]
metrices = ["manhattan", "euclidean", "cosine", "jaccard","matching", "ppmcc"]

# BoW options
# bow_embedding_types = dict({0: "Frequency Vectors", 1: "One-Hot-Encoding", 2:␣
 ↪"TF-IDF", 3: "LSA"})
bow_embedding_types = dict(
    {0: "Frequency Vectors", 1: "One-Hot-Encoding", 2: "TF-IDF", 3: "LSA"}
)

# Word embedding options
# dh_embedding_types = dict({0: "Word2Vec pre-trained", 1: "Doc2Vec␣
 ↪pre-trained", 2: "FastText pre-trained"})
dh_embedding_types = dict(
    {
        0: "Word2Vec pre-trained",
        1: "Doc2Vec pre-trained",
        2: "FastText pre-trained"
    }
)
```

Figure 6.1: Code snippet - Global Parameter Settings

Two of the datasets are stored in the directory *data*, the other one is loaded from the *scikit-learn* library. The file *helperFunctions.ipynb* contains all shared functions. The images created during the program execution are automatically saved in the *images* directory. The directory *embeddings* contains pre-trained word embeddings which are described in section 6.2.

6.1 Pre-Processing

After loading the three data sets, they are stored in a *pandas* data frame. In order to make them more manageable, the rows with null values and empty texts are dropped. Moreover HTML content is removed. Before each data set is converted into a machine readable format,

they are cleaned-up and pre-processed. The conversion into tokens and the pre-processing is mainly done using the open source libraries *spaCy* and *regular expressions (re)*. *spaCy* is written in *Python* and *Cython*. Besides *spaCy* there is a wide range of libraries for NLP. Depending on the application, some of them are more suitable than others (cf. [4]). For this thesis *spaCy* was mainly chosen because of its high performance and claimed accuracy for syntactic analysis in [4]. The command sequence of the tokenization process is as follows:

Listing 6.1: Data Tokenization

```
# Load language model instance (available pretrained statistical model for English
    language)
# en_core_web_sm: English multi-task CNN trained on OntoNotes. Assigns context-specific
    token vectors, POS tags, dependency parse and named entities.
# This pipeline is by convention stored in a variable called 'nlp'.
nlp = spacy.load("en_core_web_sm")
nlp.add_pip(merge_entities)
if clean:
        df["cleaned"] = df.text.apply(lambda x:cleaning(x))
        df["tokens"] = df.cleaned.apply(lambda x: nlp(x))
else:
        df["tokens"] = df.text.apply(lambda x: nlp(x))
# Number of tokens after tokenization
df["numTokens"] = [len(token) for token in df.tokens]
# Save index of non empty token lists
idx_token = df[df.numTokens != 0].index.tolist()
# Remove column entries where numTokens is 0
print(
  "The dataset contains {} texts which result in an empty token list.".format(
  len(df[df.numTokens == 0]))
)
# Drop columns of dataset where numTokens = 0
df = df.drop(df[df.numTokens == 0].index)
df = df.reset_index(drop=True)
```

First, a language model containing language specific rules has to be loaded for the tagging, parsing and entity recognition process. The library *spaCy* provides different pre-trained language models. The `en_core_web_sm` model used in this thesis is a small-sized English model trained on written web text (blogs, news, comments) that includes vocabulary, syntax,

entities and word vectors [17]. It can be downloaded using the following command

```
python -m spacy download en_core_web_sm
```

After loading the English model, a `nlp` object is received which contains the processing pipeline (`tagger`, `parser`, `ner`). In order to tokenize named entities, i.e. *New York*, the function *merge_entities* is added to the *spaCy* pipeline. If the user activated text cleaning (*clean=True*), which is recommended, the data is cleaned before processing it with the loaded English model. For example, URLs and email addresses are removed, as well as whitespaces, newlines and tabs. A detailed overview is given in the corresponding *Python* function, listed in the appendix (see B.1). During processing *spaCy* first generates tokens. Therefore, it separates words by whitespace characters and applies exception rules and prefix or suffix rules. The official example provided on [45] is the tokenization of the sentence *Let's go to N.Y.!* and is shown in Figure 6.2.

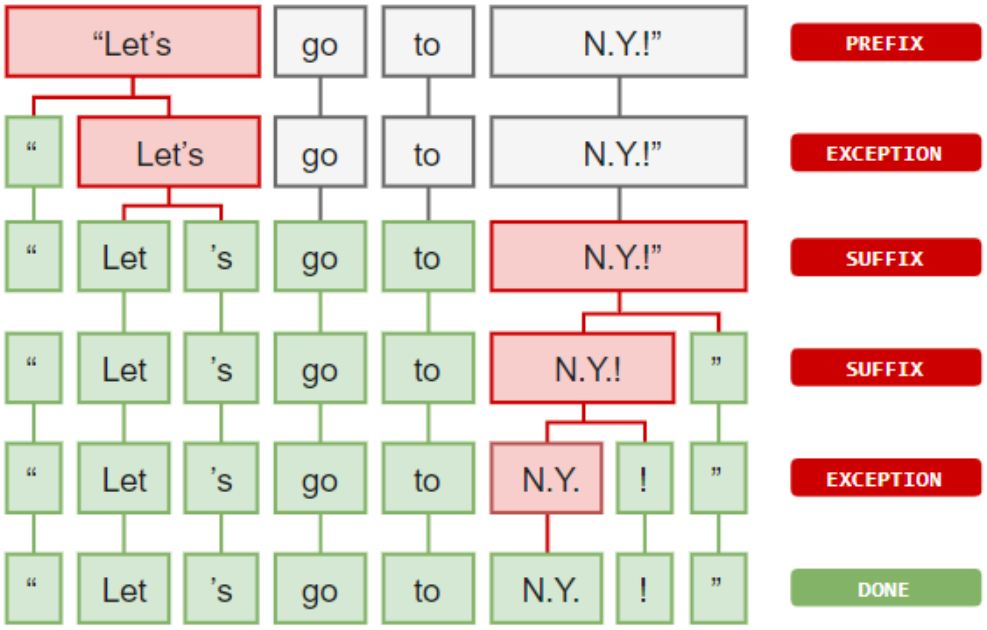


Figure 6.2: *spaCy* tokenization process. Image taken from [45]

The processed text is saved in a `doc` object, stored in `df['tokens']`. Each token is represented as a `token` object including several attributes. For example `.text` returns the token text and `.is_punct` indicates whether the token is punctuation or not. Since the

tokenization process can result in an empty list, these are subsequently removed.

As described in 2.3, the data is pre-processed in order to remove *stop words*, for example, and therefore to reduce the vector space dimension. Furthermore, the cluster effectiveness can be improved at the same time. An outline of the implemented pre-processing process can be obtained from Listing 6.2.

Listing 6.2: Data Pre-processing

```
def dataNormalization(doc, stopwords):
  """
  This function normalizes tokens.
  :param text: tokenized text
  :return normalized: normalized tokens
  """
  normalized = []
  for token in text:
    # Filter out symbols, punctuations and whitespaces
    if token.pos_ not in ('SYM', 'PUNCT', 'SPACE'):
      lemma = ""
      if token.lemma_ != "-PRON-": # -PRON- stands for personal symbol
        lemma = token.lemma_.lower().strip()
      else:
        lemma = token.lower_
      # Remove stopwords
      if lemma not in stopwords:
        lemma = re.sub(r"[^a-zA-Z]+",'',lemma)
        if lemma != '':
          if len(lemma)>3:
            normalized.append(lemma)
  if len(normalized) == 1 and len(normalized[0])==1:
    return []
  return normalized

# Load 326 stopwords
stopwords = spacy.lang.en.stop_words.STOP_WORDS
# Normalize tokens
df['tokensPreProc'] = df.tokens.apply(lambda x: dataNormalization(x, stopwords))
```

The first step is the removal of all white spaces as well as punctuations and symbols. Then

each inflectional form is reduced to its lower case base form using the lemmatization tool provided in *spaCy*. Since *stop words* do not contain important meaning they are removed in the next step. The first 10 out of 326 *stop words* are *'m*, *myself*, *himself*, *enough.*, *this*, *may*, *because*, *above*, *see* and *say*. Since there usually is some noise present in the corpus after performing, the words having a very short length, here 3, are removed. If an empty token list is assigned to a document after normalization, it is removed.

6.2 Data Representation

After pre-processing the text data, the features are generated. The user can choose the following representation types to be used for the cluster analysis.

Listing 6.3: Data Representation Options

```
1 bow_embedding_types = dict({0: 'Frequency Vectors', 1: 'One-Hot-Encoding', 2: 'TF-IDF',
      3: 'LSA'}) # Bag of word representation methods
2 dh_embedding_types = dict({0: 'Word2Vec pre-trained', 1: 'Doc2Vec pre-trained', 2: '
      FastText pre-trained'})
```

The open source natural language processing library *gensim*, which is implemented in *Python* and *Cython* is used to build the document vectors. Before converting the corpus which contains all documents into vectors, a mapping (dictionary) between each word in a document and an unique id must be generated. Therefore the `gensim.corpora.dictionary.Dictionary` class is used, as shown in Listing 6.4.

Listing 6.4: Data Representation: Dictionary construction

```
1 dictionary_raw = Dictionary(df.tokensString)    # Create dictionaries (map tokens to id
2 dictionary_norm = Dictionary(df.tokensPreProc) # based on observed order)
3 features_raw = len(dictionary_raw.token2id)    # Extract dictinary feature
4 features_norm = len(dictionary_norm.token2id)
```

First, the BoW model is implemented. To convert the tokenized documents to numerical vectors using this approach, the function shown in Listing 6.5 is used.

Listing 6.5: Conversion from document tokens to vectors

```
def bow_document_term_matrix(model_type, dictionary, corpus, features):
  model = None
  if model_type == 0: # Term Frequency
        model = [dictionary.doc2bow(doc) for doc in Documents(corpus)]
        matrix = matutils.corpus2dense(model, features).transpose()
  elif model_type == 1: # One Hot
        model = [[(token[0], 1) for token in dictionary.doc2bow(doc)] for doc in
    Documents(corpus)]
        matrix = matutils.corpus2dense(model, features).transpose()
  elif model_type == 2: # TF-IDF
        model = TfidfModel(dictionary=dictionary, normalize=True)
        vec = [model[dictionary.doc2bow(doc)] for doc in corpus]
        matrix = matutils.corpus2dense(vec, features).transpose()
  else:
    return []
  del model
  return matrix
```

The function distinguishes between the desired BoW types. For the representation using term frequency encoding the function `doc2bow()` is called which counts the number of occurrences of each word, converts it to its integer id stored in the dictionary and returns it as a sparse vector. Since *gensim* doesn't provide a one-hot encoder, the `doc2bow()` is used again, with the difference that the list of tuples returned from this function is converted into a list containing the returned `token_id` and a 1 as a tuple. In order to generate TF-IDF encoded tokens a corresponding model is build using `models.TfidfModel()`. As already described in chapter 2.4.2, the word embeddings are separated into count based and predictive based methods. In the context of count based methods, LSA is implemented. Before the actual model can be created the optimal number of components (k_tsvd) must be identified. Therefore a TSVD is applied to each corpus using `sklearn.decomposition.TruncatedSVD`. Then the explained variance stored in the attribute `explained_variance_ratio` is extracted and the cumulative explained variance is calculated. If this value exceeds a threshold (here 80%), the corresponding number of components is set as the optimal

number. This number is then passed to the constructor of the LSA model, implemented in `gensim.models.LsiModel`, as described in Listing 6.6.

Listing 6.6: Creation of an LSA model using *gensim*

```
# train model
lsamodel = LsiModel( tfidf_norm, num_topics=optimal_k_tsvd, id2word=dictionary_norm)
tfidf_lsa = lsamodel[tfidf_norm]
# Convert corpus into dense numpy array
tfidf_lsa_mat = matutils.corpus2dense(tfidf_lsa, optimal_k_tsvd).transpose()
```

Since it is beneficial to clean and pre-process the data sets and that TF-IDF is the only BoW method which measures the importance of a word in regard to a document in a corpus, LSA is only applied to the vector representation of this kind of data (`tfidf_norm`).

The last step is the representation of the corpus based on the distributed hypothesis which is not just a mapping from each token to a score. The library *gensim* provides already pre-trained word embeddings as well as the functionality to train models. In this thesis only pre-trained models were used. For word2vec a pre-trained model trained on Google News was downloaded from [48]. The file is called `GoogleNews-vectors-negative300.bin.gz` and contains 300-dimensional vectors for 3 million words and phrases. For the extension of word2vec, called doc2vec, a pre-trained model containing vectors of the same length, was downloaded from [49]. Since a FastText model can not be loaded into *gensim*, it has to be downloaded using `gensim.downloader.api.load()` each time the complete program is executed. The data set is called `fasttext-wiki-news-subwords-300` and contains 1 million vectors whose dimension is 300, too.

The chosen word embedding models are iteratively scanned and the corresponding pre-trained model containing the word representations is loaded for raw as well as pre-processed data. The word2vc model is completely loaded after 183.5 seconds, the doc2vec one after 8.07 seconds and the fastText model takes 325 seconds to load. To create the respective term document matrix, out-of-vocabulary words are first removed from the documents. Docu-

ments that have no word vectors in word2vec are completely removed. For the remaining ones the procedure is as follows: For each word the corresponding trained word vector of the model is used. Each document is then represented as the average of its word vectors.

Once word vectors of a representation type are generated, the k-Means algorithm is applied to generate the corresponding clusters.

6.3 k-Means Algorithm

The clustering procedure is shown in Figure 6.3 as an example of vector representation using BoW. It is carried out analogously for LSA and word embeddings.

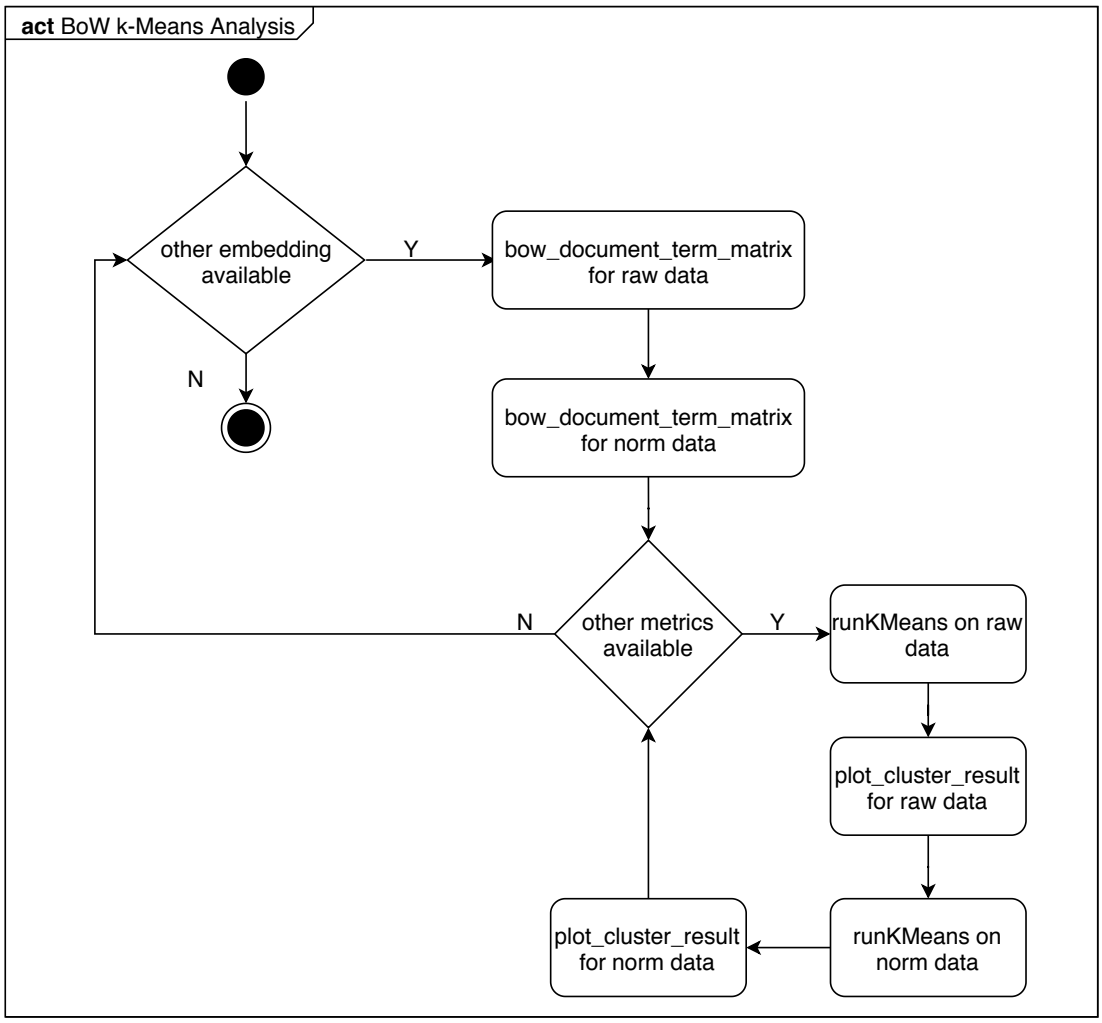


Figure 6.3: Activity Diagram of implemented clustering analysis using BoW text representation approach.

As discussed in 6.2 the feature matrix for each embedding is first generated from the dictionary and the corpus. The `runKMEans()` function is then called for each metric that is to be tested. Figure 6.4 shows the corresponding UML activity diagram.

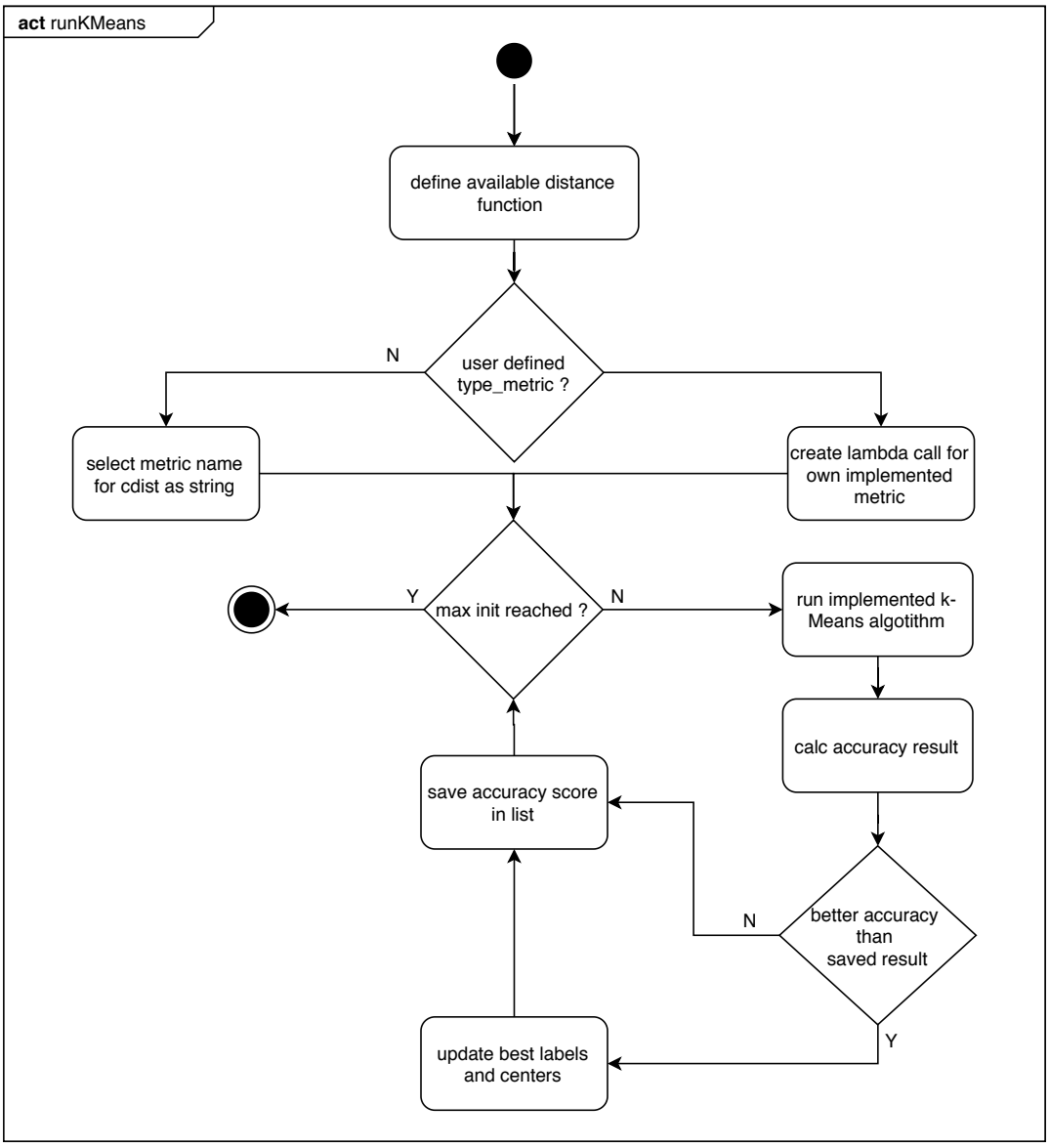


Figure 6.4: Activity Diagram of implemented `runKMeans` function.

A code snippet of this function is provided in the following Listing:

Listing 6.7: Run k-Means

```
1 def runKMeans(df_mat, labels_gt, optimal_cluster_no, metric, n_init):
2   term_based_metrics = {'manhattan':'cityblock', 'euclidean':'euclidean', 'cosine':'
      cosine',
3                         'dice':'dice', 'jaccard':"jaccard", 'matching':"matching",
4                         'overlap':type_metric.USER_DEFINED}
5   user_metric_func = {'overlap':my_overlap}
```

```
6     # Compute cluster distance function
7     dist_type = term_based_metrics[metric]
8     dist = None
9     if dist_type == type_metric.USER_DEFINED:
10      if metric == "overlap":
11        dist = lambda p1,p2: user_metric_func[metric](p1, p2)
12    else:
13      dist = dist_type
```

The function *runKMeans* receives the feature vectors, the correct targets and the pre-defined number of clusters, the distance metric and the number of times the algorithm will be run with different initial centroid settings (`n_init`). The value of `n_init` is set to 3. For each of the above data sets the number of clusters is set equal to the number of pre-assigned labels. The distance metrics shown in 6.7 are implemented. For all metrics except overlap and PPMCC, the provided functions of `scipy.spatial.distance.cdist` are called. Alternatively, a user defined metric whose name is stored in the `term_based_metric` dictionary as a `USER_DEFINED` metric is passed to the function. After specifying the appropriate metric inside of `runKMeans`, a customized k-Means algorithm according to algorithm 1 is called as often as specified with the passed `n_init` variable. The k-Means algorithm provided by *scikit-learn* could not be used because it is only implemented for the Euclidean distance and no other distance metric can be specified. An alternative is offered by a library called *pyclustering*. However, this library has a disadvantage, too. Depending on the parameter setting the k-Means algorithm returns less clusters than specified. This contradicts the conditions defined in 3.1. The ***N****atural* ***L****anguage* ***T****oolkit*, which is also a commonly used library for NLP can also be used for clustering using the k-Means algorithm. Unfortunately, the computation time for large data sets is a considerable problem here. So a custom implemented k-Means algorithm (`k_means_own`) based on algorithm 1 is used in this thesis. The control and object flow of this function represented as a UML diagram in Figure 6.5.

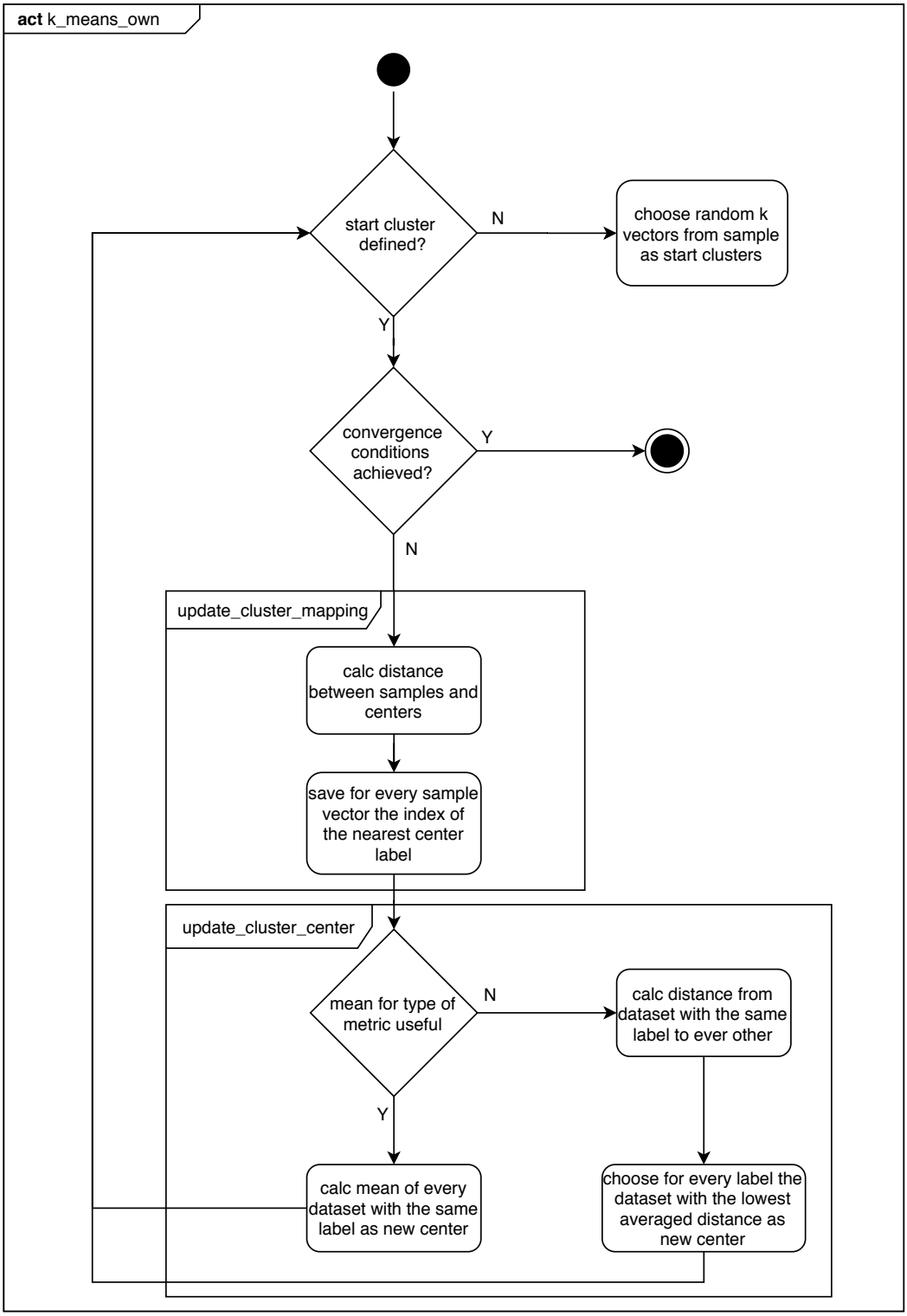


Figure 6.5: Activity Diagram of implemented `k_means_own` function.

As can be seen in Listing 6.8, the function receives the feature vectors, the numbers of clusters k and the specified distance function. Optionally a start cluster setting, a convergence conditions and a flag which indicates how the cluster centers are calculated in each update step can be passed. If no cluster division is passed, k cluster centers are randomly chosen from the data set.

Listing 6.8: Implemented k-Means Algorithm

```
def kmeans_own(sample,k,distance_metric,initial_centers=None, converg_dist=0.01,mean_f=
    True):
  # List contains the corresponding labels
  nearest_cluster_label = np.zeros(len(sample))
  if initial_centers == None:
    random_ind = choice(len(sample), size=k, replace=False)
    initial_centers = sample[random_ind,:]
  temp_dist = 1.0
  max_iter = 100
  iter_count = 0
  while temp_dist > converg_dist and iter_count < max_iter:
    nearest_cluster_label = update_cluster_mapping(sample,initial_centers,
    distance_metric)
    new_centers = update_cluster_center(sample,nearest_cluster_label,initial_centers,
    distance_metric,mean_f)
    temp_dist = 0
    for i in range(len(initial_centers)):
     temp_dist = temp_dist + np.sum(distance.cdist([initial_centers[i]],[new_centers[i
    ]],distance_metric))
    initial_centers = new_centers
    iter_count = iter_count + 1
  temp_dist = calc_wce_count(sample,nearest_cluster_label,initial_centers,
    distance_metric)
  return temp_dist,new_centers, nearest_cluster_label
```

Subsequently, an iterative process takes place within the while loop, which is repeated until a maximum number of iterations is reached or the calculated cluster centers have moved less than `converg_dist`. First, the data objects are assigned to the cluster centers using the `update_cluster_mapping` function. This function requires the feature vectors, the current cluster centers and a distance function. The output are the new labels. The implementation

of this function shown in 6.9 is based on the k-Means algorithm implemented in *sklearn*.

Listing 6.9: Assign data objects to clusters

```
def update_cluster_mapping(sample,centers,distance_metric):
  num_cores = multiprocessing.cpu_count()
  X,Y,dtype = _return_float_dtype(sample,centers)

  fd = delayed(_dist_calcer)
  ret = np.empty((sample.shape[0], centers.shape[0]), dtype=dtype, order='F')

  Parallel(n_jobs=num_cores,backend="threading")(
           fd(distance_metric,ret, s , X[s], Y)
           for s in gen_even_slices(_num_samples(X), effective_n_jobs(num_cores)))

  # determine new cluster centers
  min_dist_ind = np.argmin(ret,axis=1)
  return min_dist_ind
```

For each feature vector the function calculates the distance to each cluster center using the passed metric. This is one of the most computationally intensive steps of the algorithm, so the calculations are performed simultaneously. The feature vector is divided into as many equal parts as there are CPU cores on the computer. Every thread calls the function `_dist_calcer` which calculates the distance using the function `scipy.spatial.distance.cdist` in combination with the specified metric. Each calculated portion is then stored in the correct position of the entire distance matrix. The advantage of the `cdits` function is that already implemented distance metrics like euclidean or cosine similarity, for example, are based on fast C implementations. The parallel implementation allows a relatively fast calculation of distance matrices even for large data sets. As can be seen in Listing 6.9, the nearest cluster center is then determined for each feature vector using the `numpy.argmin` function. The resulting cluster labels are returned to the `kMeans_own` function. Once the cluster assignment has been determined, the cluster centers are updated. This is done within the `update_cluster_center` function, which calculates the average value of all data objects belonging to a cluster and defines it as the centroid. However, this only works for metrics that

allow the calculation of an average. For metrics, like Jaccard, there is no average of points. So, the centroid is defined as the point closest to other points. Closest in this case indicates smallest maximum distance to the other points, smallest average distance to other points or smallest sum of squares of distances to other points. In this thesis the data points whose average distance to all other points within the group is minimal is assigned as the new cluster center. So, the k-Means algorithm tends to become a k-Medoids algorithm. The determined centroids are returned to the calling function `kMeans_own`. The convergence criteria is updated by calculating the distance between the old and new cluster centers (within-cluster error). If the termination criterion of the loop is not met, the next iteration of the while loop starts. Otherwise, the current cluster centers and labels are returned as the result of the algorithm. This is then compared with the known targets to evaluate the clustering efficiency. As mentioned before, the k-Means algorithm is ran several times (`n_init`), so that the center points and labels that have achieved the best result are returned at the end of the runs. Afterwards the results are graphically presented using PCA and the images are automatically saved.

CHAPTER 7

Results

In this chapter the implemented KDD process is evaluated based on the different data sets. The aim is to answer the research questions and to evaluate the cluster challenges which were already presented in chapter 1. These were:

- Select appropriate features of the documents. What kind of pre-processing techniques need to be applied and what kind of word representation strategies result in suitable similarity results?

- Select a similarity measure between documents. What text similarity approach performs best on the given data sets?

- Select an appropriate clustering method.

- Efficient implementation in terms of required memory, CPU resources and execution time.

- Select a quality measure to check the cluster result.

7.1 Pre-processing

The runtimes required for the cleaning and tokenization process of all three data sets as described in 6.2 are shown in Table C.3. As can be seen in Figures 7.1 and 7.2 the cleaned

data sets, *20 newsgroups* and the *Reddit*, are unbalanced in terms of document size which is measured by computing the number of tokens in each document.

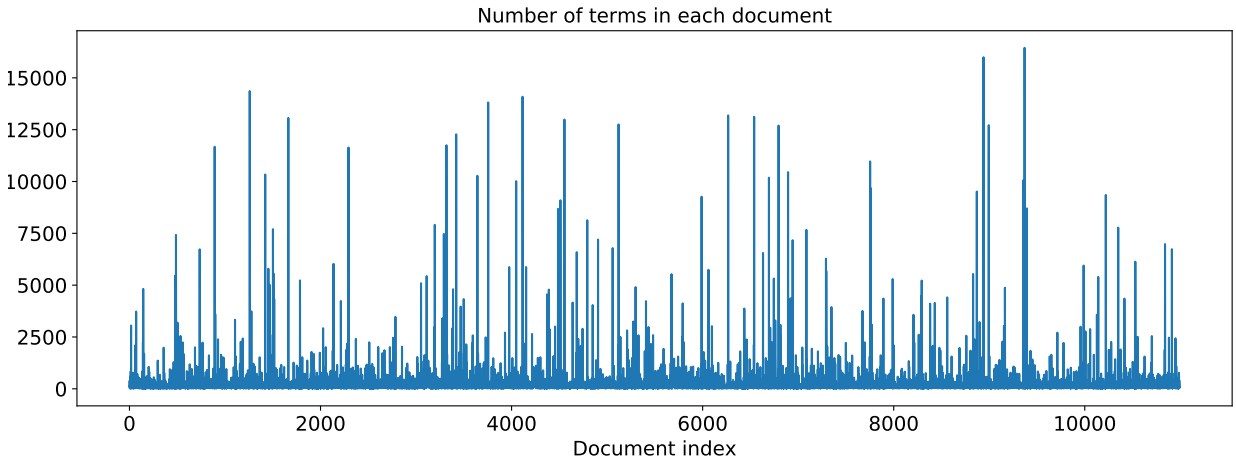


Figure 7.1: *20 Newsgroups* data set (`n_categ=20`) is unbalanced in terms of document size. Some documents contain only one term and others 37, 424. The average term frequency is 238.

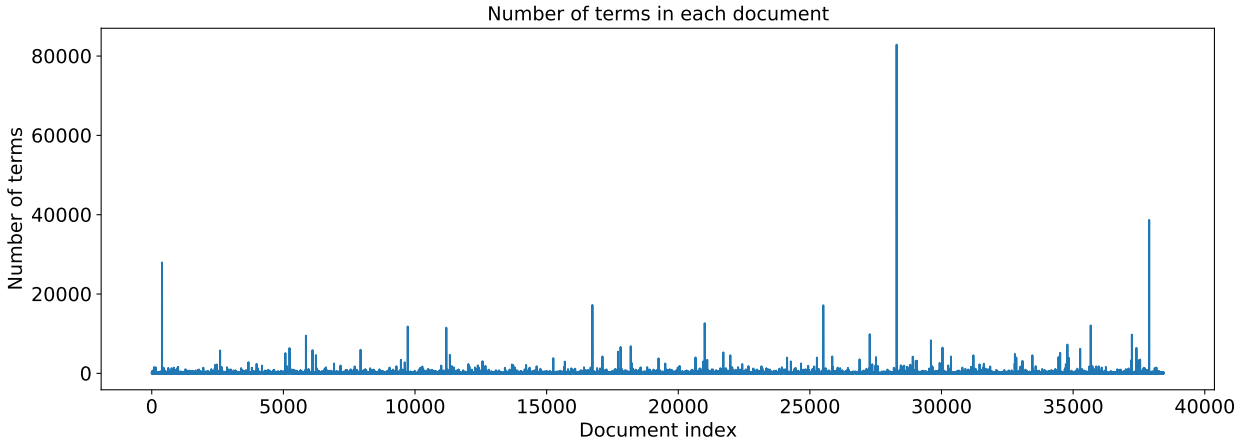


Figure 7.2: *Reddit* data set (`n_categ=5`) is unbalanced in terms of document size. One document with many tokens is particularly prominent.

As illustrated in Figure 7.3, the number of words contained in each document varies for the cleaned *Jeopardy!* data set, too. However, there are not as many "outlier documents" as in the *20 newsgroups* and *Reddit* data set. The number of words for `n_categ=20` ranges from 2 to 47 per document. The average number of tokens in a document is 17.

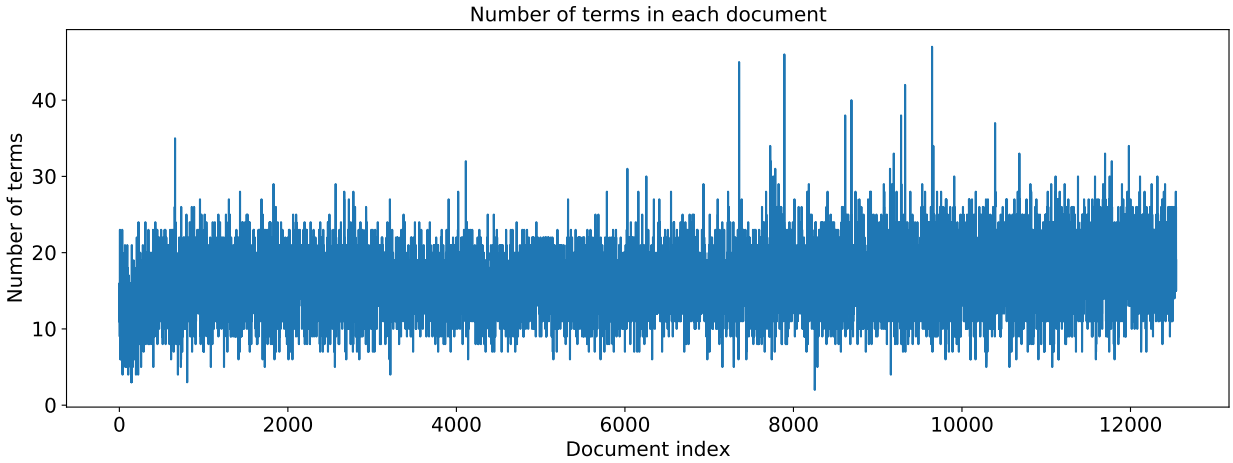


Figure 7.3: Document size of *Jeopardy!* data set for `n_categ=20`.

Table C.2 provides an overview of the token numbers for cleaned and uncleaned data. These were generated within the times specified in Table C.3. It can be seen that the larger the data sets, the longer the time for cleaning and tokenization. Furthermore, cleaning together with tokenization does not take as long as just tokenization for data sets containing documents involving a large amount of tokens. This, along with the fact that cleaning improves the final data mining result, suggests that it should be applied in advance. Depending on the data set, cleaning the tokens without further pre-processing reduces the data set by $177,327$ to $309,021$ for *20 newsgroups* or 38 to 198 tokens for *Jeopardy!*. An overview of the minimum, maximum and average number of tokens per category is provided in Figures A.2, A.3 and A.4 in the appendix. As visualized in Figures 7.4, 7.5 and 7.6, the huge number of features is mainly due to the large number of *stop words*, for example *the*, *this*, *of* and *and*.

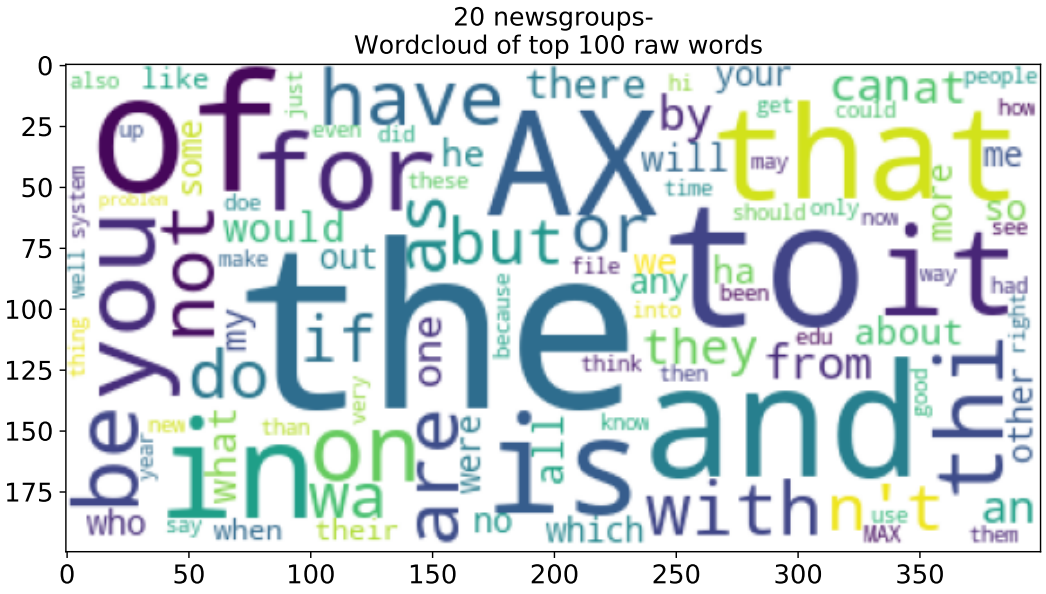


Figure 7.4: Top 100 tokens contained in *20 newsgroups* corpus (`n_categ=20`).

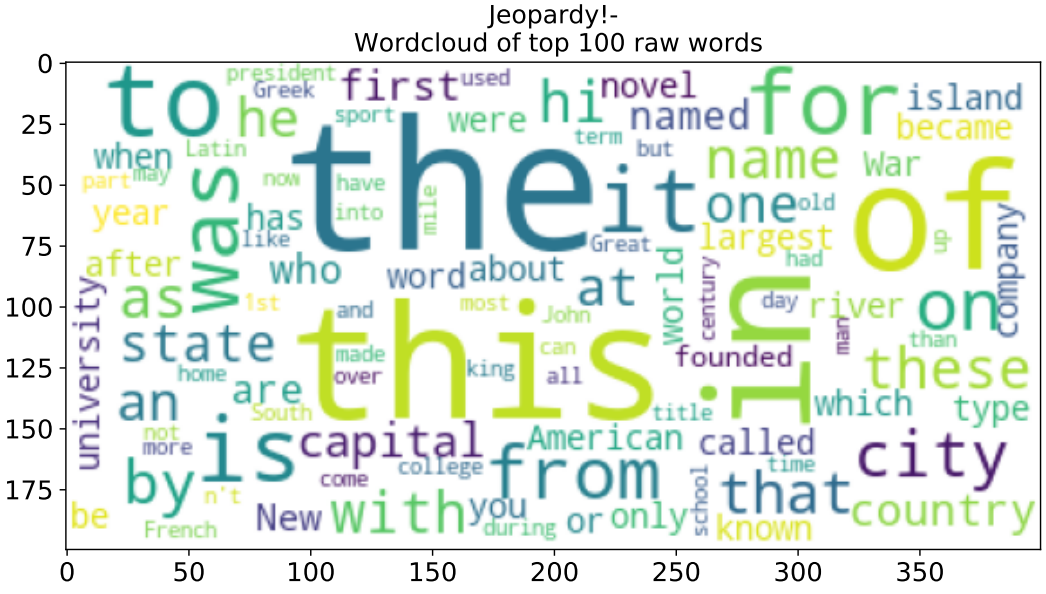


Figure 7.5: Top 100 tokens contained in *Jeopardy!* corpus (`n_categ=20`).

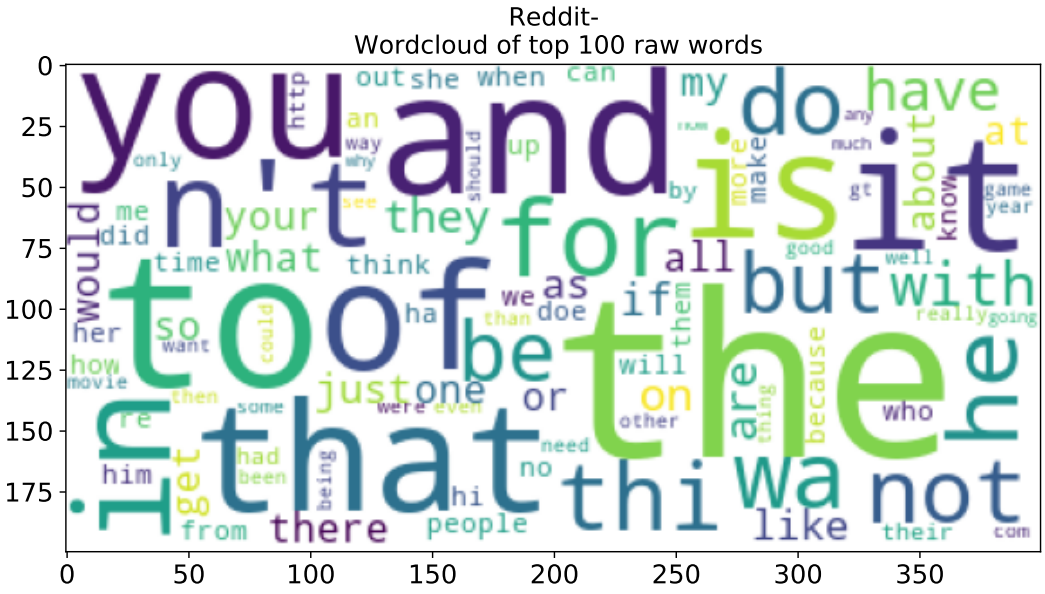


Figure 7.6: Top 100 tokens contained in *Reddit* corpus (`n_categ=5`).

The resulting word cloud is similar for different numbers of categories. Since cleaning the data does not remove *stop words*, there is no significance difference between the uncleaned and the cleaned word cloud. One thing that is noticeable, for example, is that the token *n't* is listed in the uncleaned cloud but not in the cleaned one. Figures A.2, A.3 and A.4 show the maximum, minimum and average number of tokens after pre-processing the data. The resulting top 100 words are visualized as a word cloud in Figures 7.7, 7.8 and 7.9.

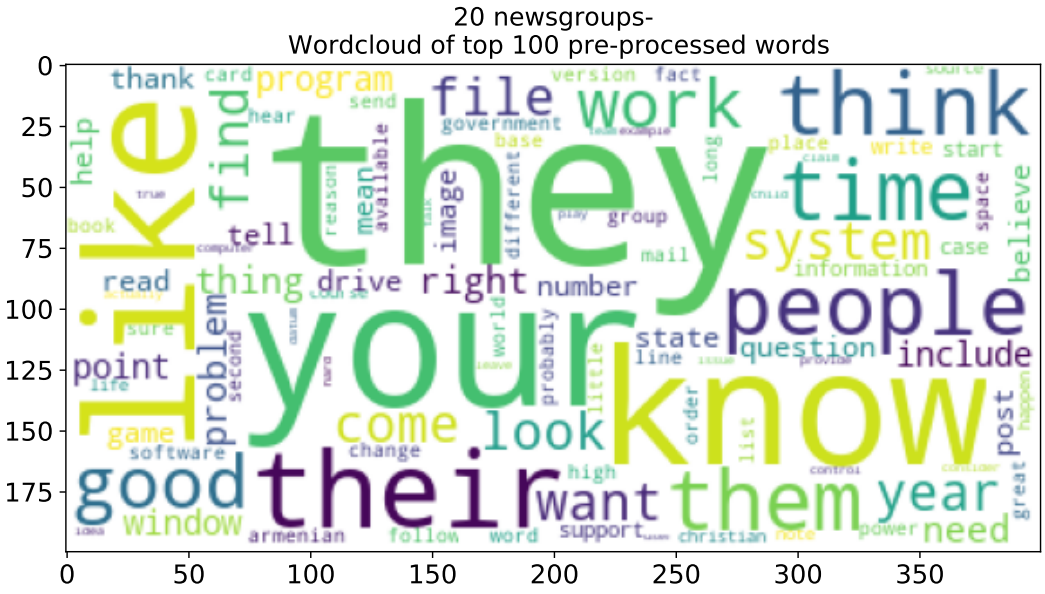


Figure 7.7: Top 100 pre-processed tokens contained in *20 newsgroups* corpus (`n_categ=20`).

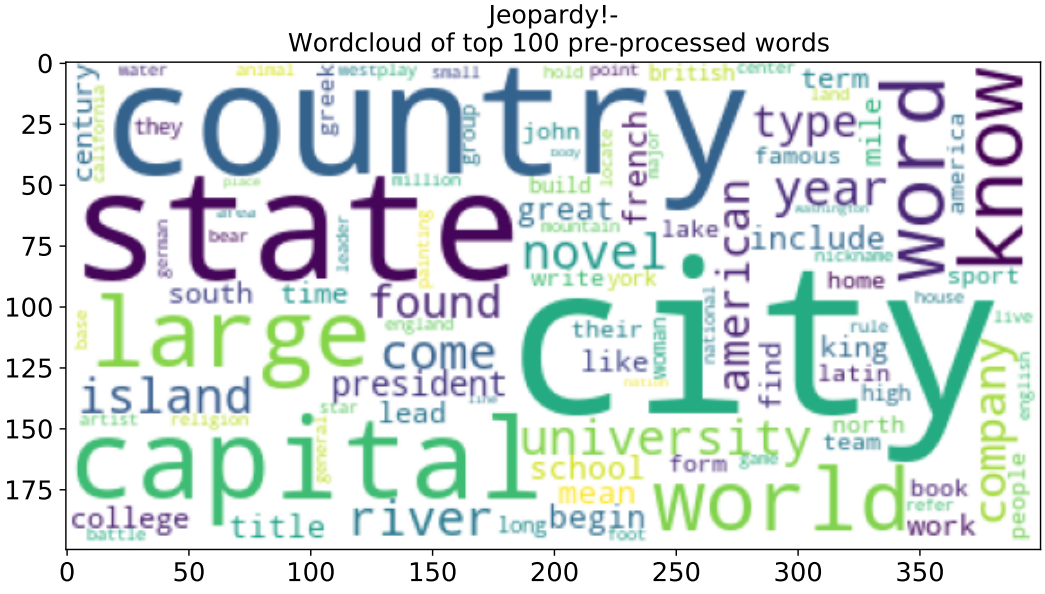


Figure 7.8: Top 100 pre-processed tokens contained in *Jeopardy!* corpus (`n_categ=20`).

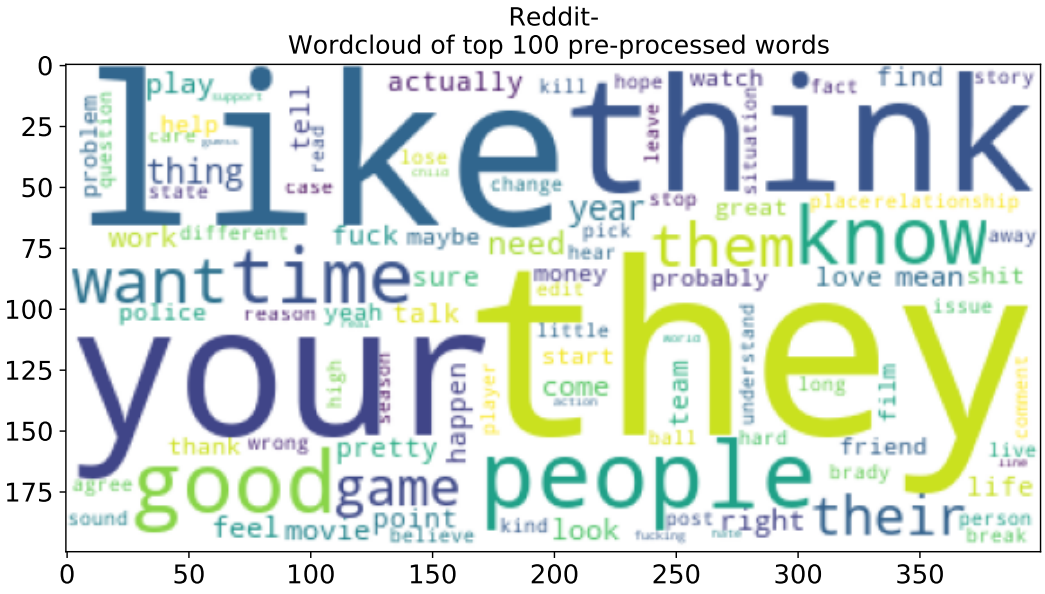


Figure 7.9: Top 100 pre-processed tokens contained in *Reddit* corpus (`n_categ=5`).

Table C.1 provides an overview of the token numbers for cleaned and uncleaned pre-processed data. The corresponding computation times are listed in Table C.3. Depending on the data set, cleaning alone reduces the data set by $4,852$ to $7,639$ for *20 newsgroups*, for example. Furthermore, it is also shown that the normalization of the data, both for non-cleaned and cleaned data, reduces the total number of tokens. Looking at the *20 newsgroups* data set, on average 30% of the raw tokens remain after normalization. The average for the *Jeopardy!* data set is about 40%. The cluster analysis considers raw and pre-processed data for both uncleaned and cleaned data. To compare these techniques, all records that are removed during pre-processing because they are assigned an empty token list are removed from the raw object list as well, so that the raw and pre-processed data objects match. Each data object is finally represented as a point in a finite-dimensional space, where the dimension corresponds to the number of unique tokens.

7.2 Data Representation

After already established that pre-processing plays a fundamental role not only for the reduction of features, the extracted tokens can be represented numerically.

7.2.1 Bag-of-Words

For the BoW approach the dictionary containing a word and id mapping needs to be determined. The dictionary characteristics for the cleaned *20 newsgroups* data set containing all categories are listed as an example in 7.1. As can be seen in this Listing, the number of features can be significantly reduced by pre-processing the data. For the *20 newsgroups* data set containing 5 categories this is a reduction of about 37%. In addition, the difference between raw and pre-processed data is apparent from the corpus. For example, the raw data set contains many *stop words*, symbols and punctuation which are removed from the pre-processed data sets.

Listing 7.1: Dictionary details for raw and pre-processed *20 newsgroups* data set ($n_c ateg = 5$). Cleaning deactivated.

```
+---------------+-------------------------------------------------------------------------+
| RAW DATA      |                                                                         |
+===============+=========================================================================+
| Corpus        | 0    [\n\n, I, am, sure, some, bashers, of, Pens, f...                 |
|               | 1    [\n, [, stuff, deleted, ], \n\n, Ok, ,, here, ...                 |
|               | 2    [\n\n\n, Yeah, ,, it, 's, the, second, one, .,...                 |
+---------------+-------------------------------------------------------------------------+
| Corpus Length | 4833                                                                    |
+---------------+-------------------------------------------------------------------------+
| Dictionary    | {'\n': 0, '\n\n': 1, ' ': 2, '         ': 3, '!': 4, "'": 5, ',': 6} |
+---------------+-------------------------------------------------------------------------+
| Features      | 62979                                                                   |
+---------------+-------------------------------------------------------------------------+


+-------------------+-----------------------------------------------------------------------------------------------------+
| NORMALIZED DATA   |                                                                                                     |
+===================+=====================================================================================================+
| Corpus            | 0    [sure, basher, pens, pretty, confused, lack, k...                                              |
|                   | 1    [stuff, delete, solution, your, problem, canad...                                              |
|                   | 2    [yeah, second, believe, price, good, look, bru...                                              |
+-------------------+-----------------------------------------------------------------------------------------------------+
| Corpus Length     | 4833                                                                                                |
+-------------------+-----------------------------------------------------------------------------------------------------+
| Dictionary        | {'actually': 0, 'basher': 1, 'beat': 2, 'bowman': 3, 'confused': 4, 'couple': 5, 'devil': 6} |
+-------------------+-----------------------------------------------------------------------------------------------------+
| Features          | 39961                                                                                               |
+-------------------+-----------------------------------------------------------------------------------------------------+
```

Table C.4 lists the matrix dimensions and their computation times corresponding to the three different BoW encoding types. It can be seen that the computation time increases with the size of the term document matrix. It is also obvious that the term frequency and the one-hot encoding are calculated faster because they are based on a simpler calculation rules. Despite the large amount of matrix entries, computation times of 0.35 up to 14.69 seconds are within reasonable limits. An overview of the 10 most frequently occurring words for all three data sets, in terms of raw and pre-processed data, using the different BoW methods, is given in the appendix on page 136 - 144.

7.2.2 Latent Semantic Analysis

By using LSA the terms occurring often in the same document and documents sharing some of the same terms are extracted in order to determine hidden (latent) global information within the corpus. The high dimensional term-document matrix can then be projected into a subspace of smaller dimension. Table C.5 shows the achieved dimensionality reductions using LSA for the three data sets introduced above. The size of the cleaned and pre-processed term-document-matrix of the *20 newsgroups* data set (`n_categ=5`), for example, could be reduced from $4830 \times 38,635$ to $4830 \times 2,220$. To reach a threshold of 80% for this data only 5.7% of the original term-document matrix columns are needed. The curve of the explained variance is shown in Figure 7.10.

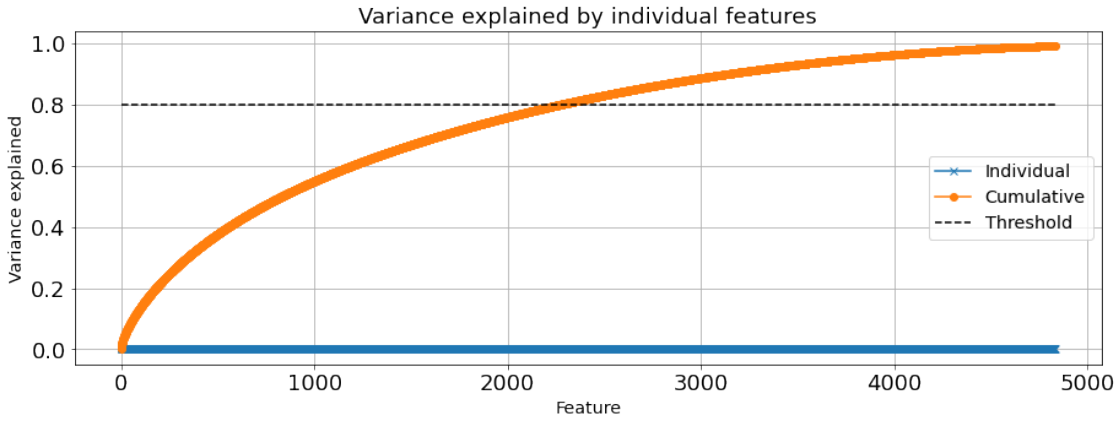


Figure 7.10: Explained Variance of *20 newsgroups* data set with `n_categ=5`.

The corresponding singular values are displayed in Figure 7.11.

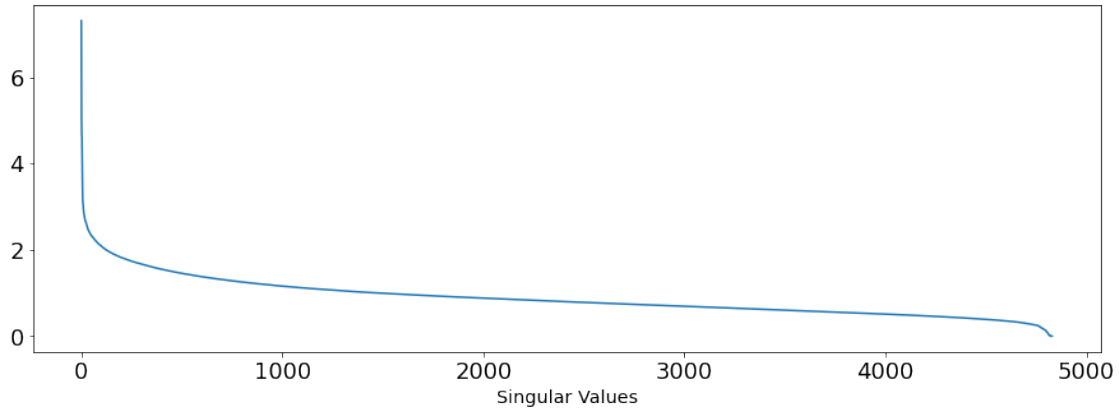


Figure 7.11: Singular Values of *20 newsgroups* data set with `n_categ=5`.

This implies that the majority of the columns of V[1] have a very small effect on the reconstruction of the term-document matrix. This in turn explains why just a few components are needed to approximate the matrix. However, the creation of the matrix using this method takes longer, because a TSVD is executed at the beginning. The execution times of LSA are listed in Table C.5.

[1] A SVD factorizes a matrix into $U\Sigma V^T$ where V contains the right-singular vectors.

7.2.3 Word Embeddings

Since all pre-trained vectors have a length of 300, the features are also mapped to this dimension. For example, considering the *Jeopardy!* data set with 20 categories, the matrix of the cleaned and pre-processed data each document is represented by 49,150 features. After applying LSA the number of columns is reduced by 74%. Using the pre-trained methods, the number of columns is 300 which is only 1.57% of the original feature size. Table C.6 lists the dimensions of the word embedding matrix using word2vec, doc2vec and fastText and their computation times.

7.3 Cluster Evaluation

Finally, the techniques applied in the cluster analysis steps are compared. The cluster analysis idea is to run the k-Means algorithm for four data settings: (1) raw, (2) pre-processed, (3) uncleaned and (4) cleaned and different text representations as well as similarity measures. Some disadvantages of methods used in the clustering steps have already been presented in the chapters about theoretical basics. For example, that the clustering result of raw data is usually more inaccurate than the one of pre-processed data. Furthermore, TF-IDF should provide the best results for clustering. To confirm or disprove these claims, the cluster results for all four data settings and the different representations and similarity measures were generated based on the *20 newsgroups* data set with 5 categories. The results were evaluated for cluster performance and cluster accuracy. The resulting most suitable techniques were then applied to the two other data sets with varying numbers of predefined categories in order to examine how they behave based on them and possibly lead to similarly successful results.

The results of the *20 newsgroups* data set using 5 categories are discussed first, followed by those of LSA and finally those of word embeddings. The techniques that seem to be most

appropriate are then applied to the other two data sets.

7.3.1 *20 newsgroups*

Bag-of-Words approach

The runtime specifications for running the k-Means three times using different metrics and data setting are listed in Table 7.1.

	Term-Frequency Encoding		**One-Hot Encoding**		**TF-IDF**	
	Raw	*Pre-Processed*	*Raw*	*Pre-Processed*	*Raw*	*Pre-Processed*
Manhattan	545	340	618	356	539	380
Euclidean distance	333	63	130	21	131	52
Cosine	70	37	63	48	61	39
Jaccard	196.5	116	183	140	498	311
Matching coefficient	236	140	220	162	252	157
PPMCC	470	353	464	646	448	526

Table 7.1: Runtime of three k-Means repetitions, for the BoW approach on uncleaned data, specified in seconds.

As the table shows, the runtimes of the individual procedures vary greatly with regard to the data settings and similarity metrics used. The runtime is shorter for pre-processing because the number of features is reduced. The variation due to different metrics used is mainly due to two reasons. Firstly, user-defined distance measures are slower because they do not have a fast C implementation in *cdist* and are instead calculated vector-wise using a loop in *Python*. In addition, the metrics for which the mean cannot be used to calculate the new cluster centers are slower, because in order to determine the medoid, a pairwise distance matrix must always be calculated additionally in each iteration step.

By looking at the cluster accuracy results of the three BoW methods, which are visualized in sections A.2.2 and A.2.3 in the appendix, the difference between the raw and pre-processed data becomes also obvious. Due to the pre-processing, a better accuracy can be achieved

because the data sets become more unique to each other, so that the subjects similarities receive a higher weighting. If the unprocessed data are excluded for further consideration, TF-IDF performs best on average for the euclidean (0.61), cosine (0.69) and PPMCC (0.79) metric. The matching matrices displayed in Figures 7.12 and 7.13 visualize the cluster assignment errors of each category based on pre-processed data represented using TF-IDF and cosine and euclidean distance. In the matrix all correct cluster assignments are located in the diagonal. The error such as the assignment of rec.motorcycle data object to targets 1, 2 and 3 using euclidean distance is represented by values outside the diagonal.

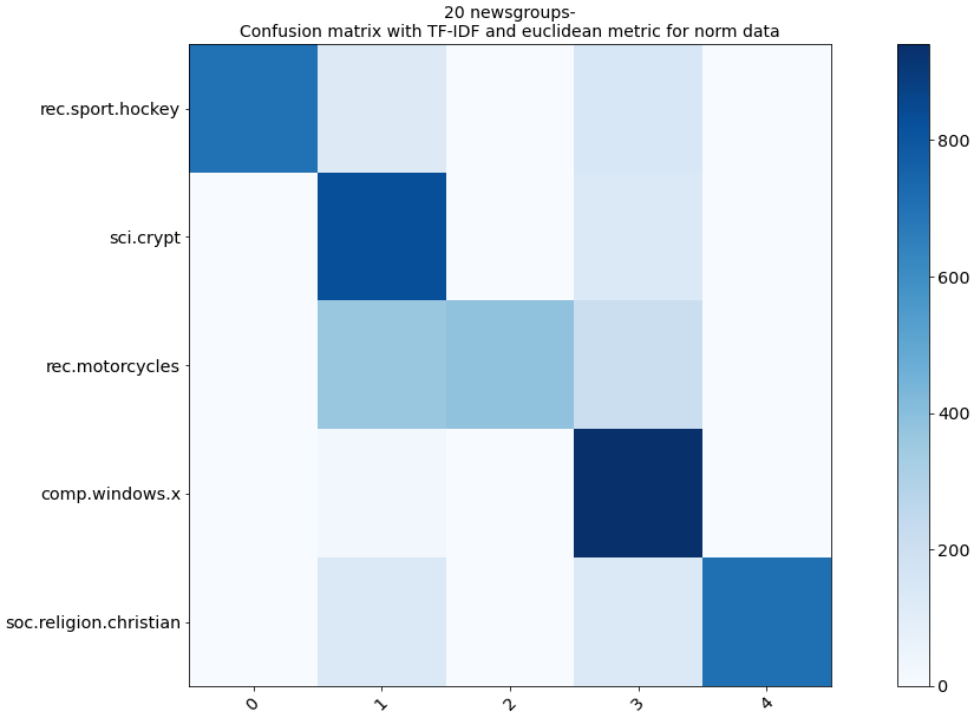


Figure 7.12: *20 newsgroups*: Matching matrix for pre-processed data transformed with TF-IDF with euclidean distance.

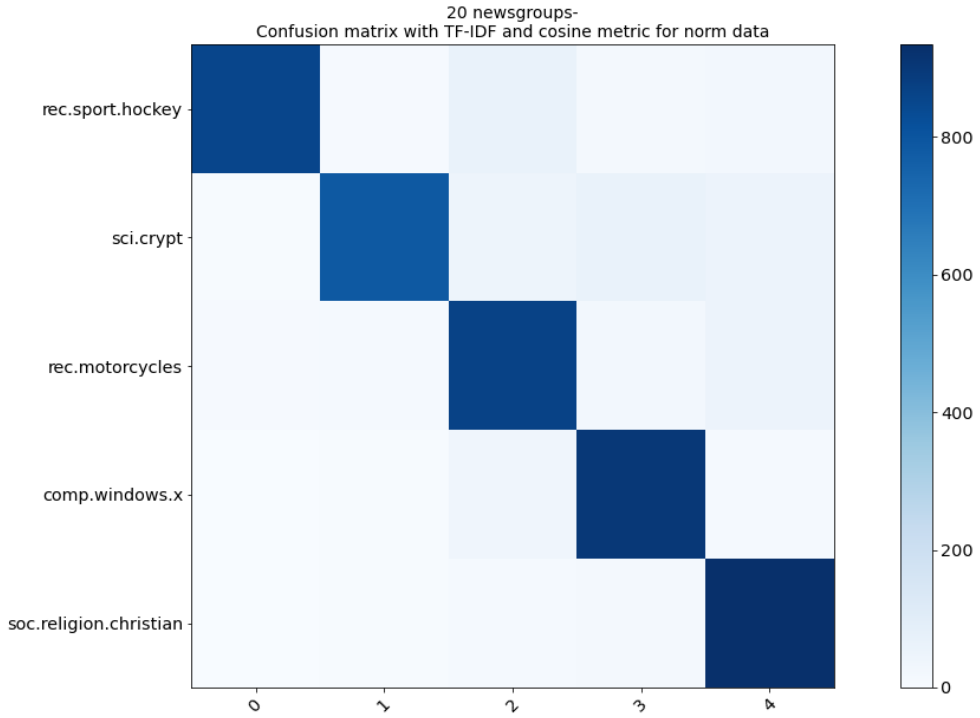


Figure 7.13: *20 newsgroups*: Matching matrix for pre-processed data transformed with TF-IDF with cosine distance.

The cluster result generated for TF-IDF and cosine is shown in the Figure 7.14.

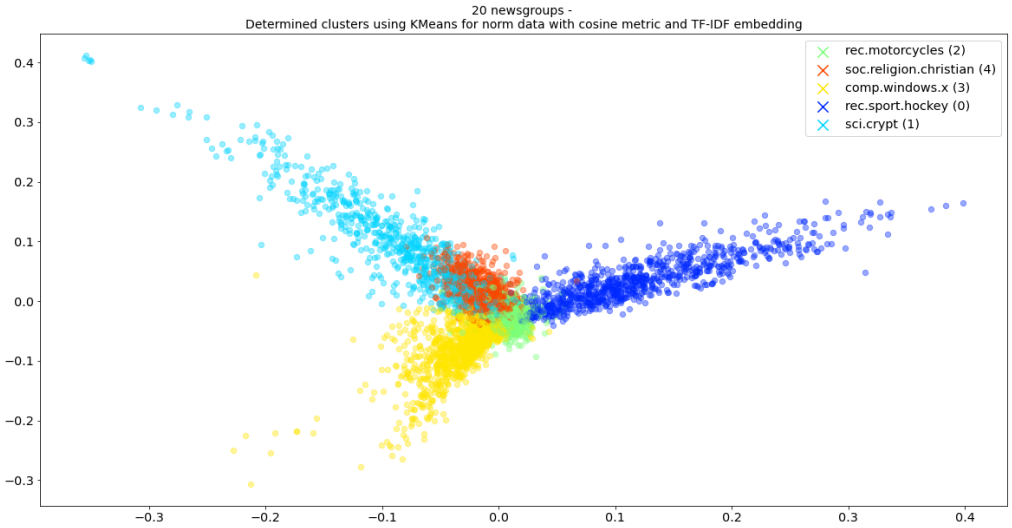


Figure 7.14: *20 newsgroups*: Cluster result for pre-processed data transformed with TF-IDF and cosine similarity.

Comparing it with the original plot (see Figure 7.15), which can be generated because the labels are pre-defined, the clustering errors shown in Figure 7.16 arise, which can also be obtained from the matching matrix.

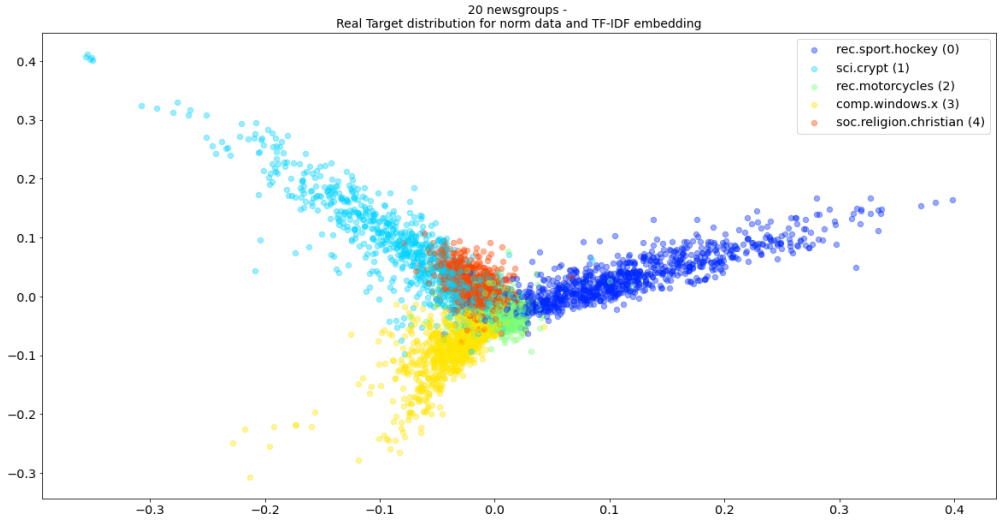


Figure 7.15: *20 newsgroups*: Pre-defined clusters for pre-processed data transformed using TF-IDF (`n_categ=5`).

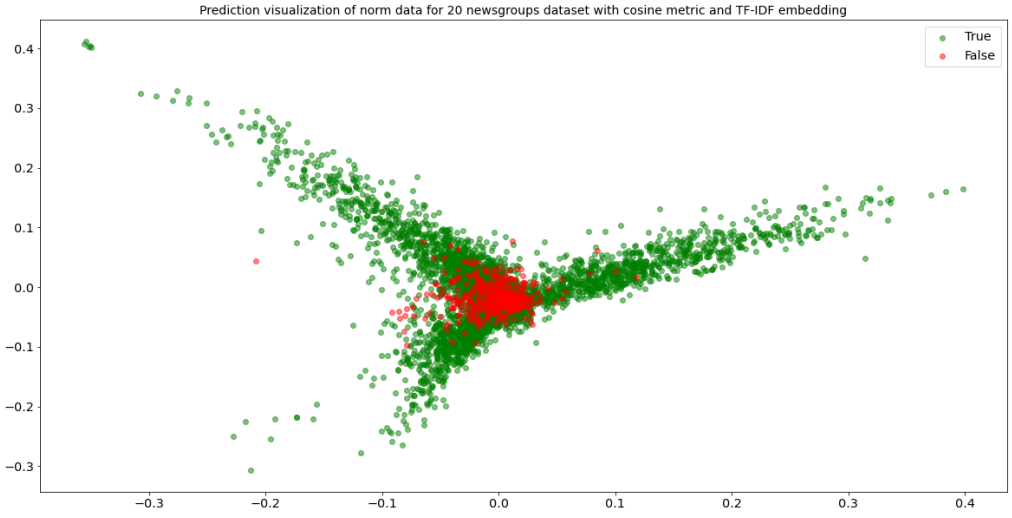


Figure 7.16: *20 newsgroups*: Cluster assignment error for pre-processed data transformed with TF-IDF and cosine similarity (`n_categ=5`).

Jaccard does not perform well on TF-IDF with 0.20, because the real-valued vector entries (no binary entries) differ from each other due to scaling by IDF and there is therefore an empty intersection. So the Jaccard distance is usually 1. The cluster result is shown in Figure 7.17. The corresponding errors are visualized in Figure 7.18.

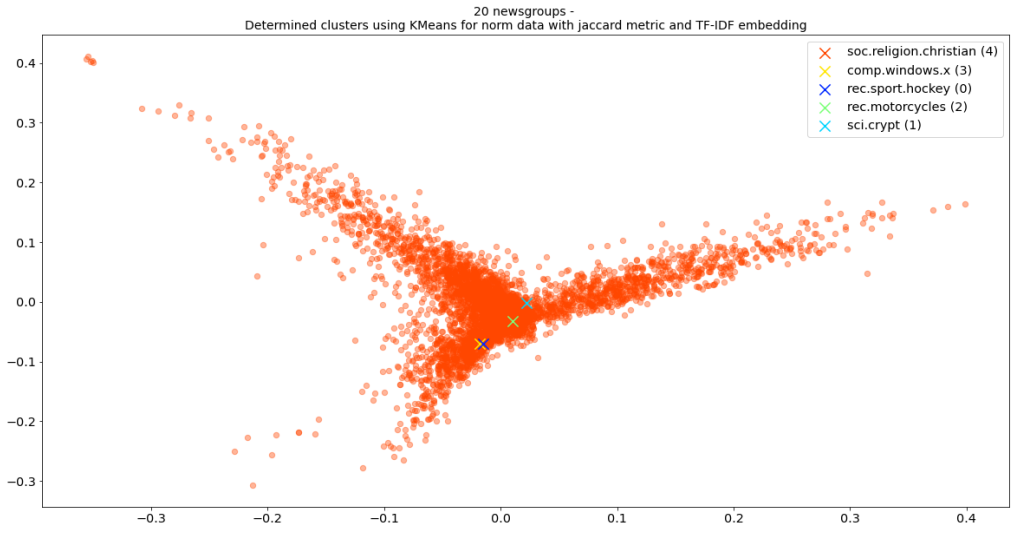


Figure 7.17: *20 newsgroups*: Cluster result for pre-processed data transformed with TF-IDF and Jaccard similarity (`n_categ=5`).

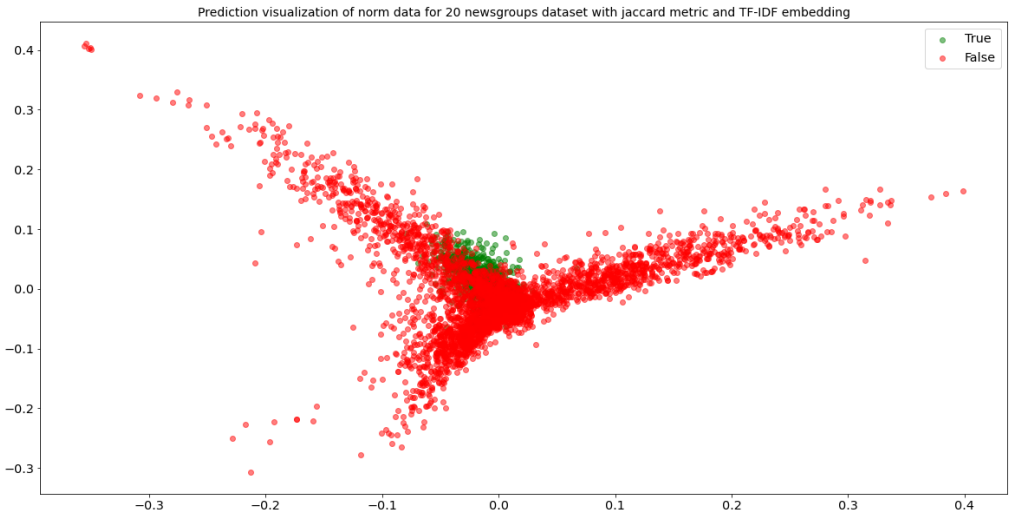


Figure 7.18: *20 newsgroups*: Cluster assignment error for pre-processed data transformed with TF-IDF and Jaccard similarity (`n_categ=5`).

For one-hot encoding and term frequency, however, the clustering result using Jaccard looks better because the probability of matching vector entries increases. Since one-hot encoding is a binary encoding, Jaccard scores better for this representation (41%) than for term frequency encoding (28%). The cluster result is displayed in Figure 7.19.

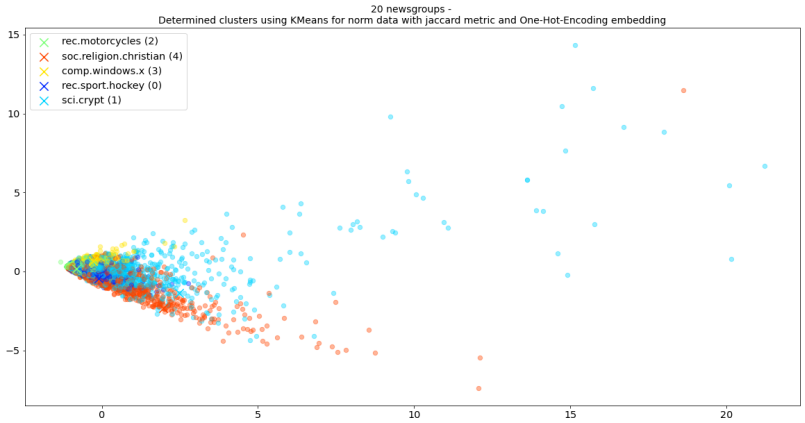


Figure 7.19: *20 newsgroups*: Cluster result for pre-processed data transformed with one-hot encoding and Jaccard similarity (`n_categ=5`).

If compared with the predefined cluster result (Figure 7.20), the following error overview shown in Figure 7.21 is obtained.

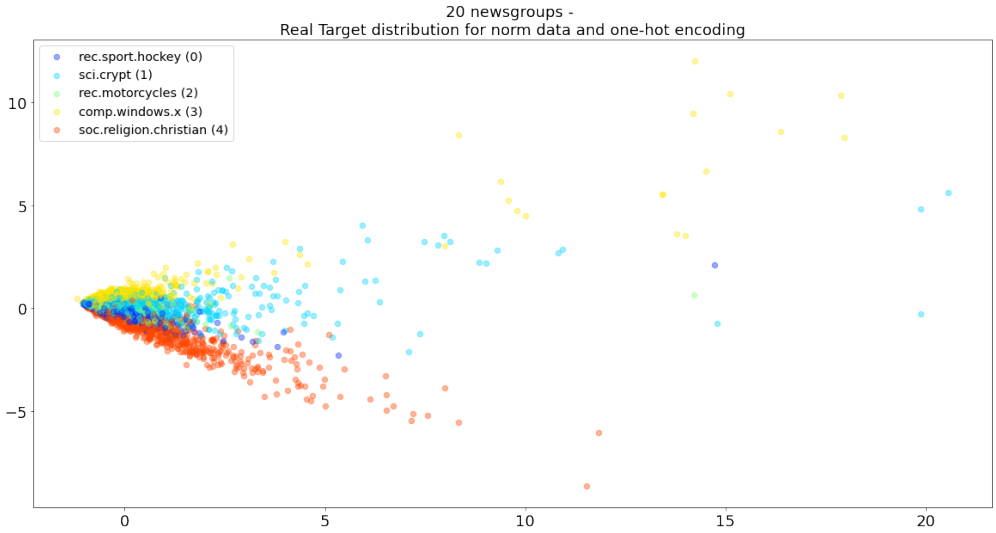


Figure 7.20: *20 newsgroups*: Pre-defined clusters for pre-processed data transformed using one-hot encoding (`n_categ=5`).

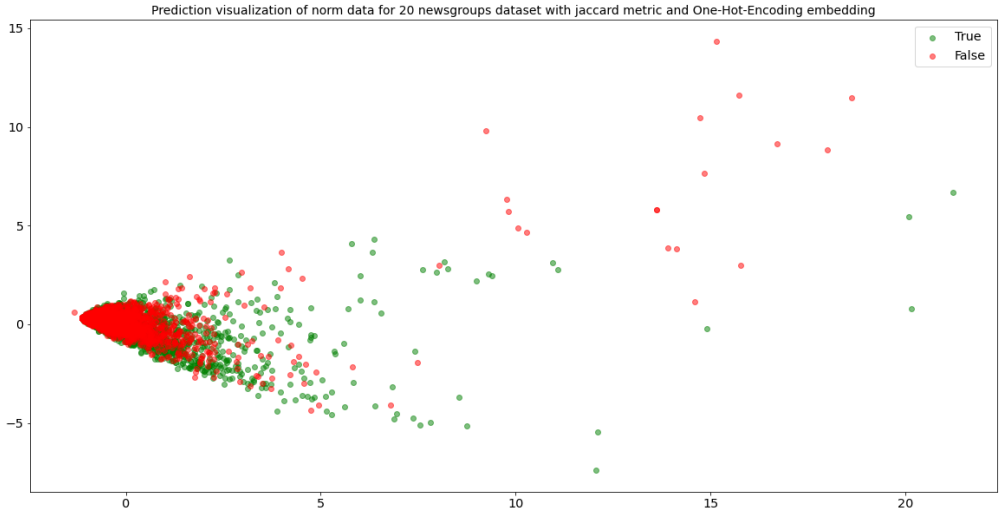
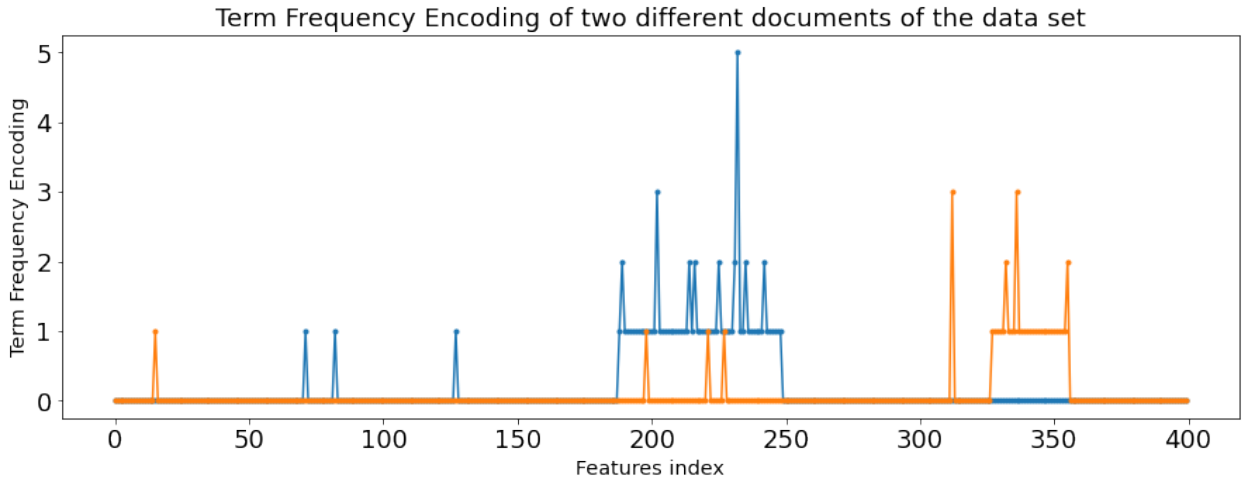


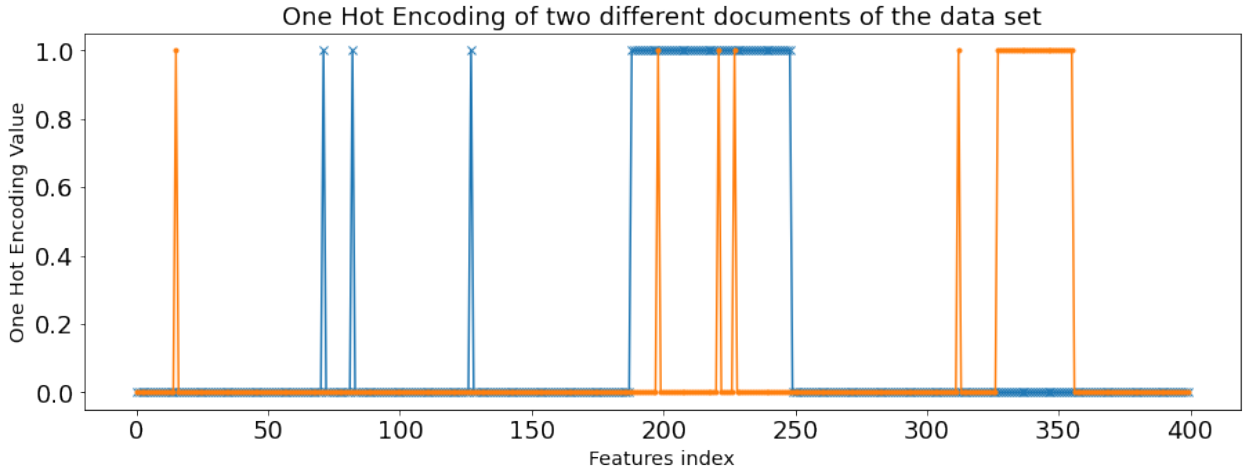
Figure 7.21: *20 newsgroups*: Cluster assignment error for pre-processed data transformed with one-hot encoding and Jaccard similarity (`n_categ=5`).

Surprisingly, cosine performs relatively well for one-hot encoding and term frequency with an accuracy of 0.7. The assumption was that the vectors contain only a few identical features

because of the sparsity and are therefore orthogonal to each other. The distance calculation of the k-Mean would then return maximum inequality. However, if at least one entry of the vectors matches, the distance calculation yields values less than 1. Since the k-Means algorithm assigns the points to the cluster centers with the smallest distance, these values can be used for an assignment. This is illustrated by the example shown in Figure 7.22.



(a) Term Frequency



(b) One-hot encoding

Figure 7.22: *20 newsgroups*: Exemplary vector representation of two data objects. The cosine distance for the term frequency encoded vectors is 0.963 and for the one-hot encoded one 0.936.

The obtained results for one-hot encoding and cosine similarity is shown in Figure 7.23. The corresponding error overview is visualized in Figure 7.24

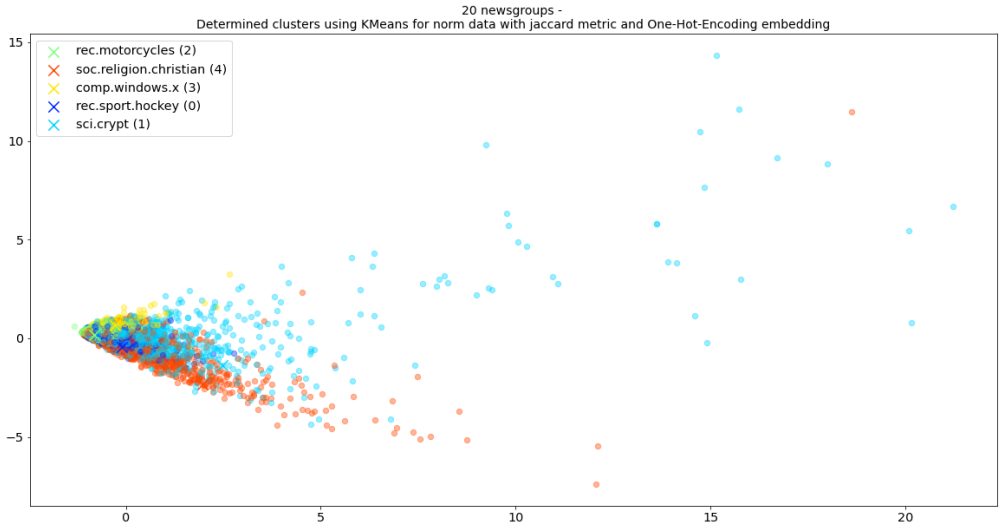


Figure 7.23: *20 newsgroups*: Cluster result for pre-processed data transformed with one-hot encoding and cosine similarity (`n_categ=5`).

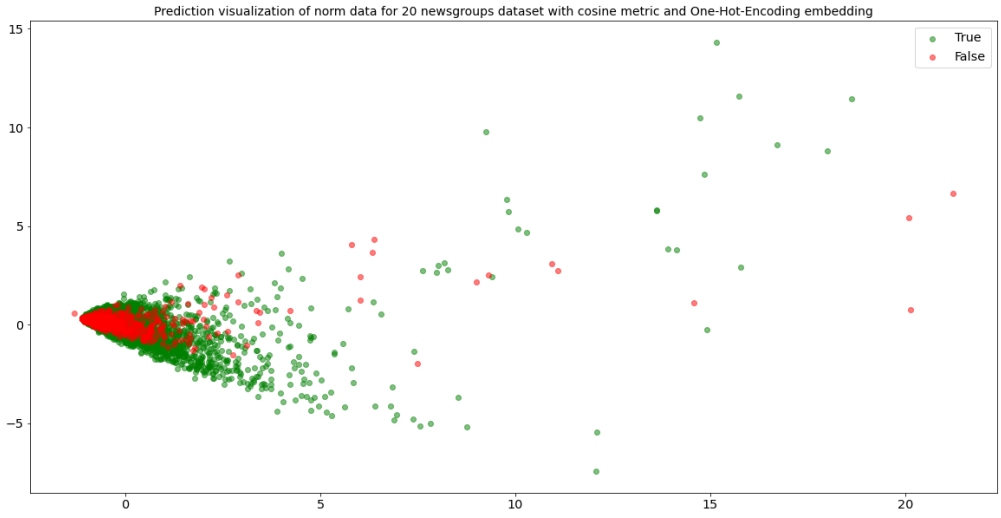


Figure 7.24: *20 newsgroups*: Cluster assignment error for pre-processed data transformed with one-hot encoding and cosine similarity (`n_categ=5`).

Since the documents of the *20 newsgroups* data set are unbalanced in terms of document size, a data subset containing 10 instead of 5 categories will probably contain more outliers. As can be seen from the plot containing the pre-defined labels shown in Figure 7.25, the

PCA clusters can no longer be clearly separated by using five additional categories.

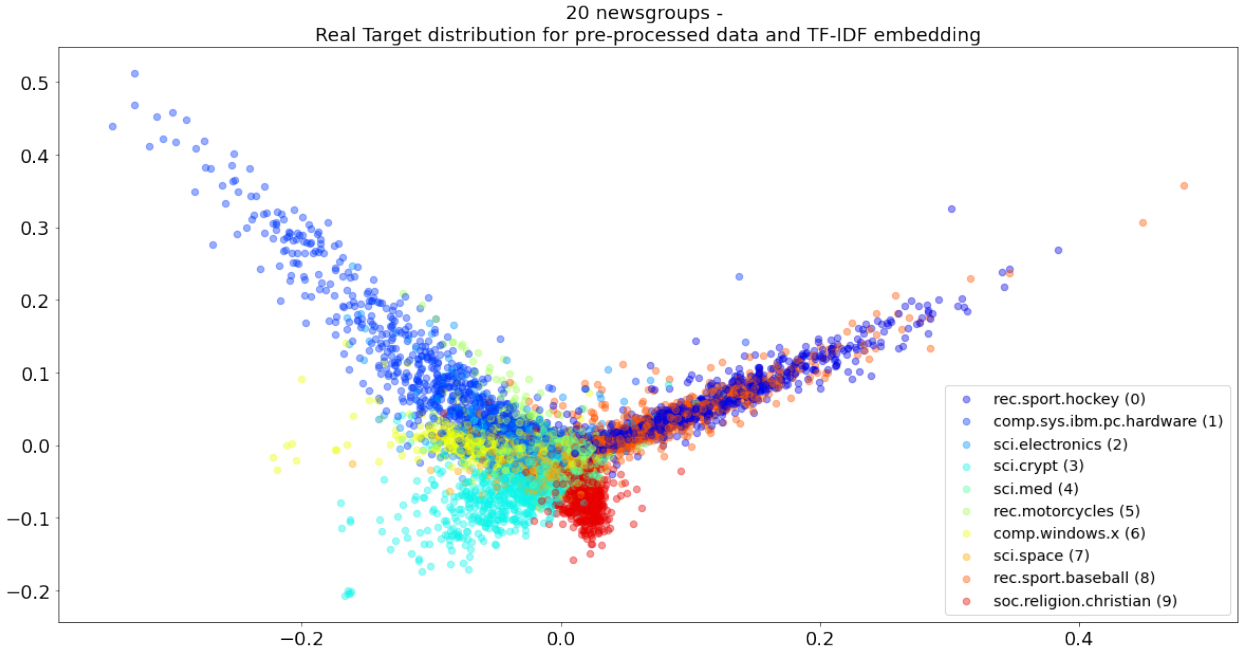


Figure 7.25: *20 newsgroups*: Pre-defined clusters for pre-processed data transformed using one-hot encoding (`n_categ=10`).

The clustering algorithm is sensitive to them because outliers cause a shift of the centers. The k-Means algorithm doesn't recognize the outliers and just assigns them to a cluster as for all other data objects. They are often assigned to individual clusters. Figure 7.26 shows that the accuracy of the best representation settings (TF-IDF) that are yield from the analysis using 5 clusters drops from 0.69 to 0.56.

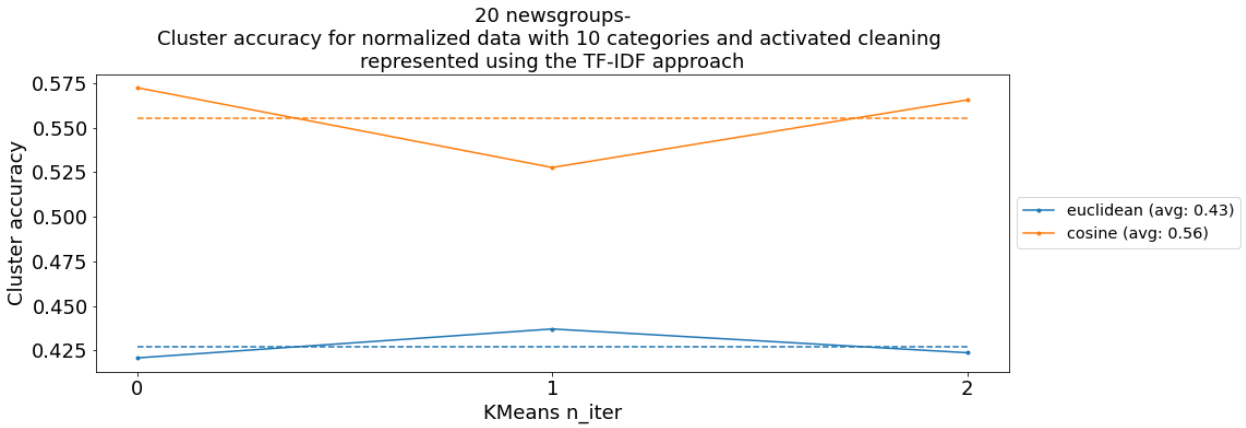


Figure 7.26: *20 newsgroups*: Cluster accuracy for pre-processed data transformed with TF-IDF and cosine similarity.

The corresponding PCA plot is shown in Figure 7.27.

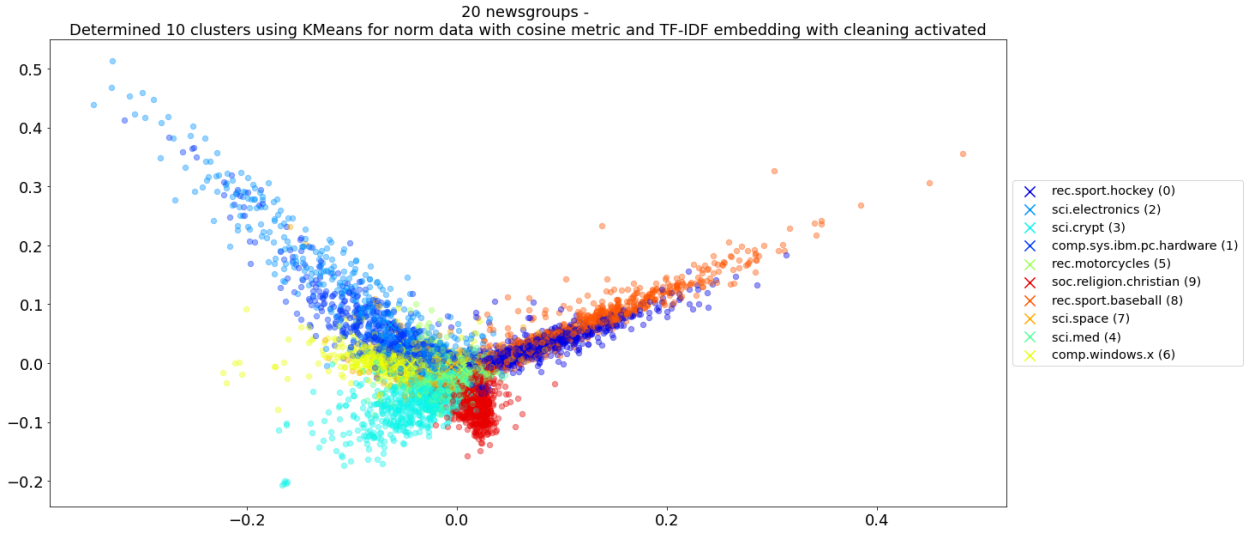


Figure 7.27: *20 newsgroups*: Cluster accuracy for pre-processed data transformed with TF-IDF and cosine similarity (`n_categ=10`).

The fact that especially the topics rec.sport.hockey and rec.sport.basketball are not well clustered can be seen in the matching matrix shown in Figure 7.28.

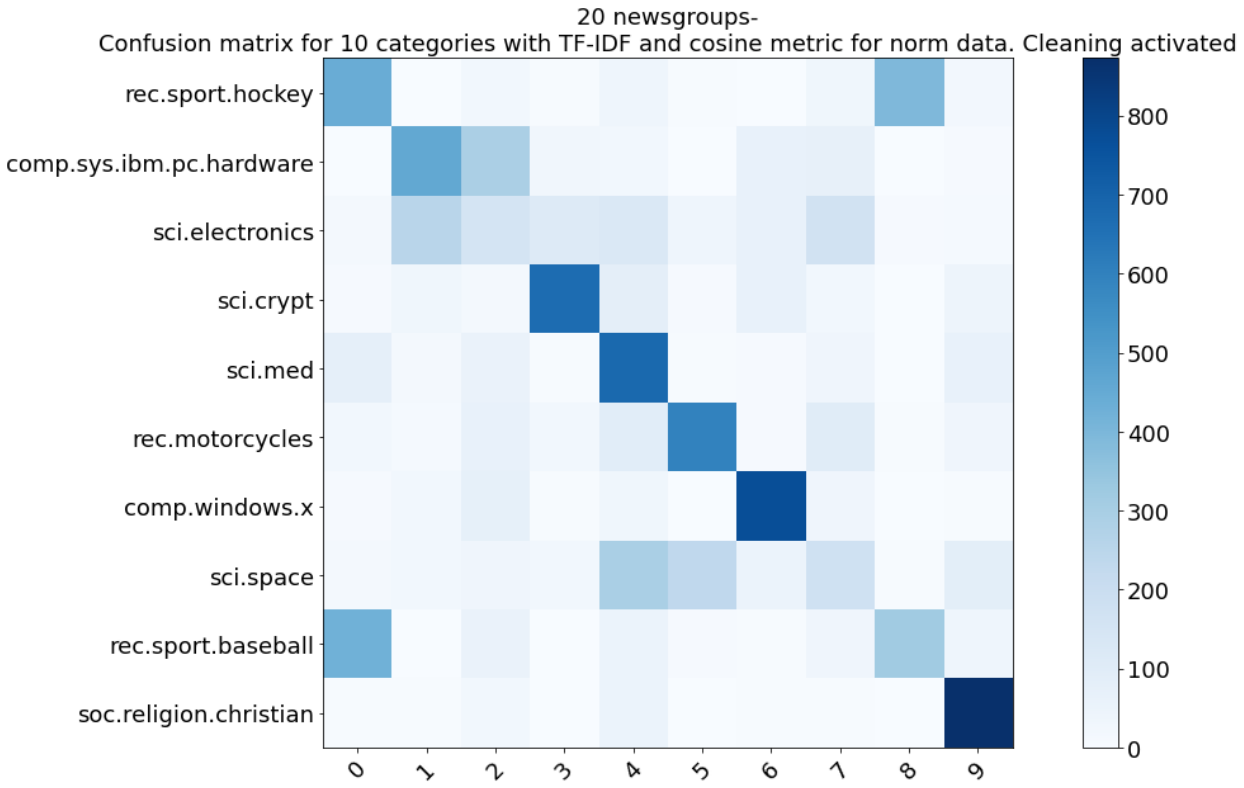


Figure 7.28: *20 newsgroups*: Matching Matrix TF-IDF encoding for pre-processed data out of 5 categories.

So an accurate clustering requires a precise and suitable definition of the document closeness. Overall, observing the cluster result of correctly and incorrectly clustered data objects, it can be seen that data is often incorrectly assigned in areas that overlap when transformed by PCA into a two-dimensional space.

7.3.2 Latent Semantic Analysis

As already described in section 6.2, LSA was only applied for pre-processed data transformed by TF-IDF. Figure 7.29 illustrates that the use of the PPMCC for LSA does lead to an accuracy improvement of 7%. The accuracy for the cosine similarity increases from 69% to 82%.

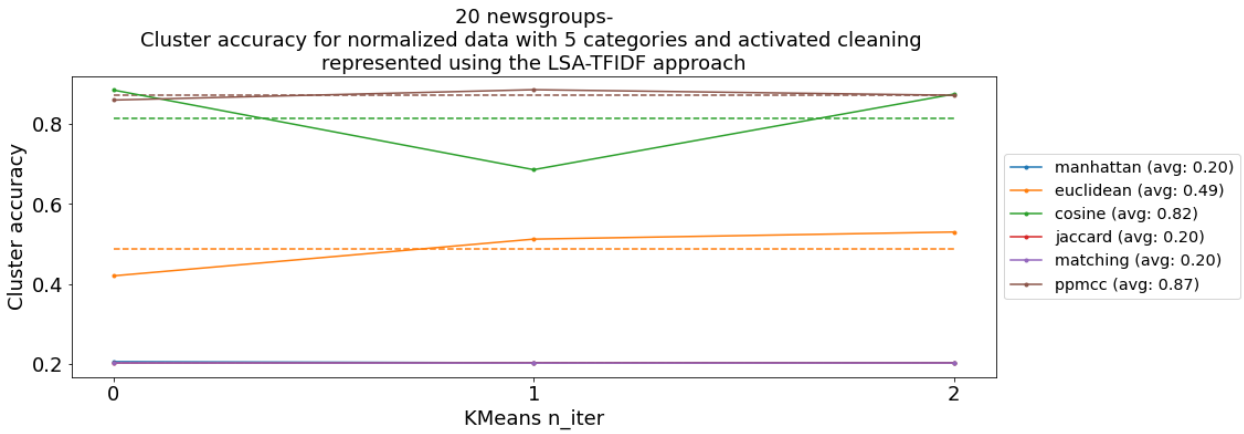


Figure 7.29: *20 newsgroups*: Cluster accuracy for pre-processed data transformed with TF-IDF, cosine similarity and LSA.

On the other hand, the number of features is significantly reduced, which has a positive effect on the runtime of the cluster algorithm without taking the computation of the optimal feature number into account. The runtime, specified in seconds, depending on three executions of the k-Means algorithm for different metrics is listed in table 7.2.

Manhattan	24
Euclidean distance	10
Cosine	6.2
Jaccard	24
Matching Coefficient	24
PPMCC	268

Table 7.2: Runtime of three k-Means repetitions specified in seconds for pre-processed data, LSA and TF-IDF

The cluster result is shown in the Figure 7.30. An overview of the correctly and incorrectly clustered data is visualized in Figure 7.31.

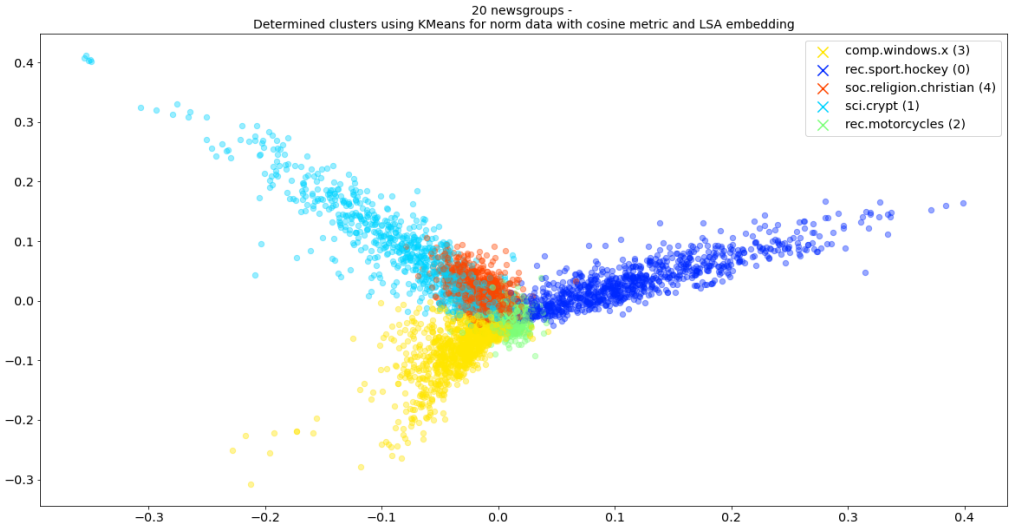


Figure 7.30: *20 newsgroups*: Cluster result for pre-processed data transformed with TF-IDF, cosine similarity and LSA.

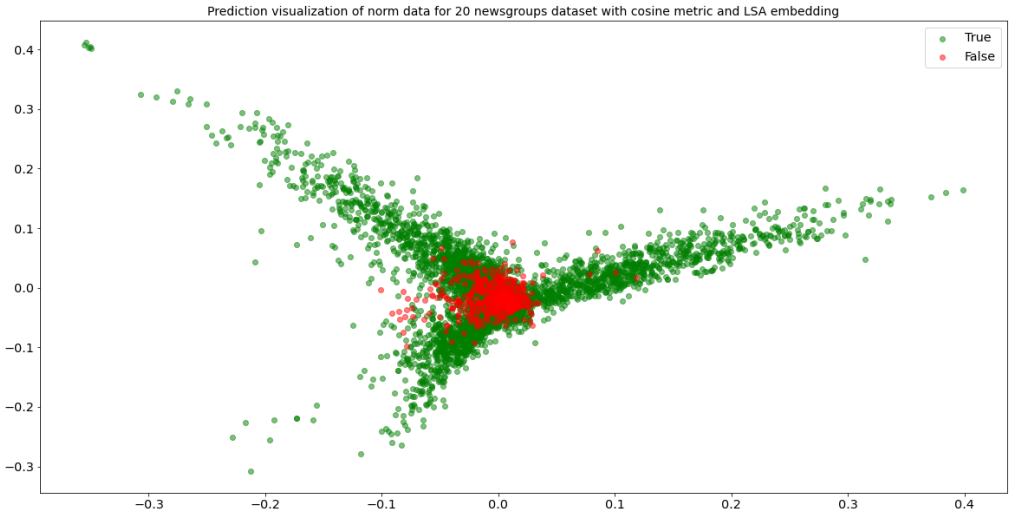


Figure 7.31: *20 newsgroups*: Cluster assignment error for pre-processed data transformed with TF-IDF, cosine similarity and LSA.

7.3.3 Word Embeddings

The accuracies of the cluster result of raw and pre-processed *20 newsgoups* data with respect to word2vec, doc2vec and fastText are shown in Figures A.20, A.21 and A.22 in the appendix. As can be seen from these Figures, fastText performs worse than word2vec on the *20 newsgroups* dataset. Doc2Vec generates even better accuracy. This is probably due to the fact that word2vec averages all the words in a document and therefore does not learn the entire document itself. With doc2vec, a representation of the document is learned in parallel to the words. Moreover it can be recognized that TF-IDF surprisingly performs better on the *20 newsgroups* data set than the word embeddings. So choosing the complex models is not beneficial here. This may be due to the fact that the word embedding has been over-fitting on the training data set. It might also be more sensitive to noise because of the content within the hidden layers. Another problem could be that the data set is too specific and the trained corpus is not suitable for it. But this is rather unlikely, because at the beginning of the embedding process it is checked if there are out-of-vocabulary words and none are recognized. The cluster result of the best word embedding model and the correctly and incorrectly assigned data points can be seen in Figures 7.32 and 7.33. The results for the other two embeddings are shown in Figures A.23 and A.24 in the appendix.

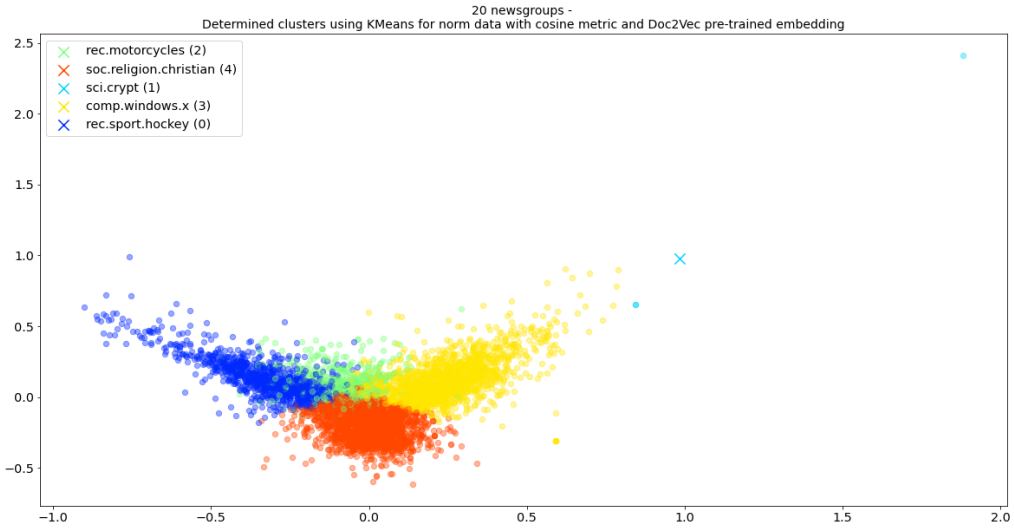


Figure 7.32: *20 newsgroups*: Cluster result for pre-processed data transformed with doc2vec and cosine similarity for clustering.

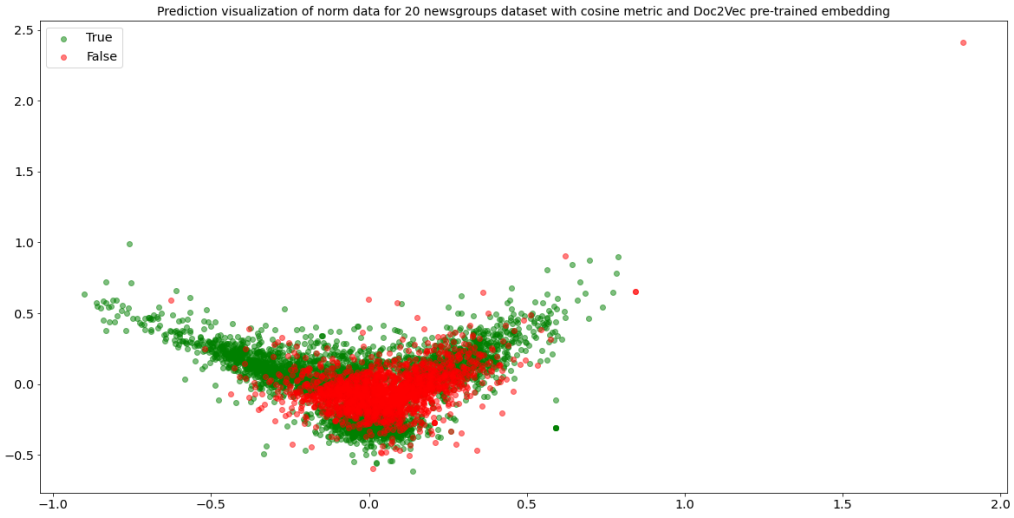
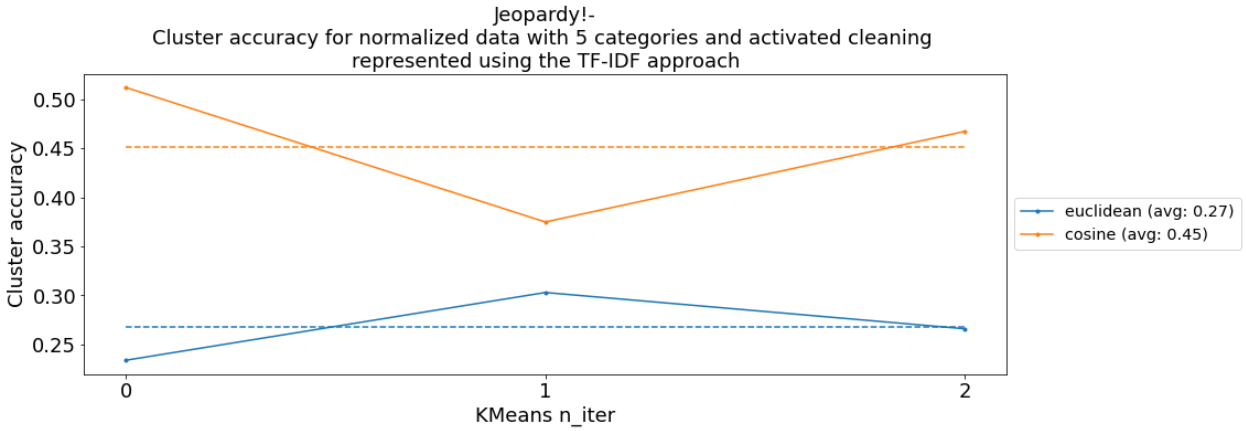


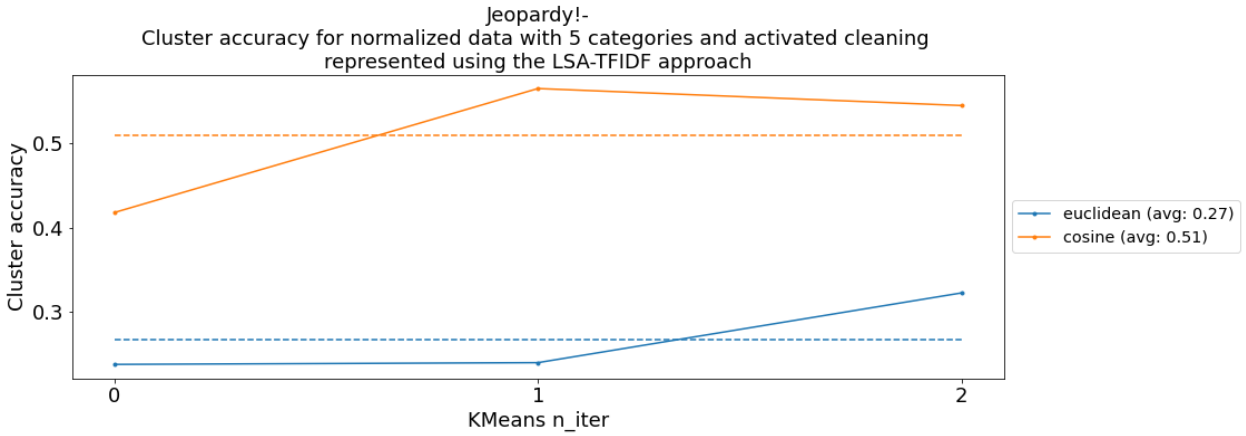
Figure 7.33: *20 newsgroups*: Cluster assignment error for pre-processed data transformed with doc2vec and cosine similarity.

7.4 *Jeopardy!*

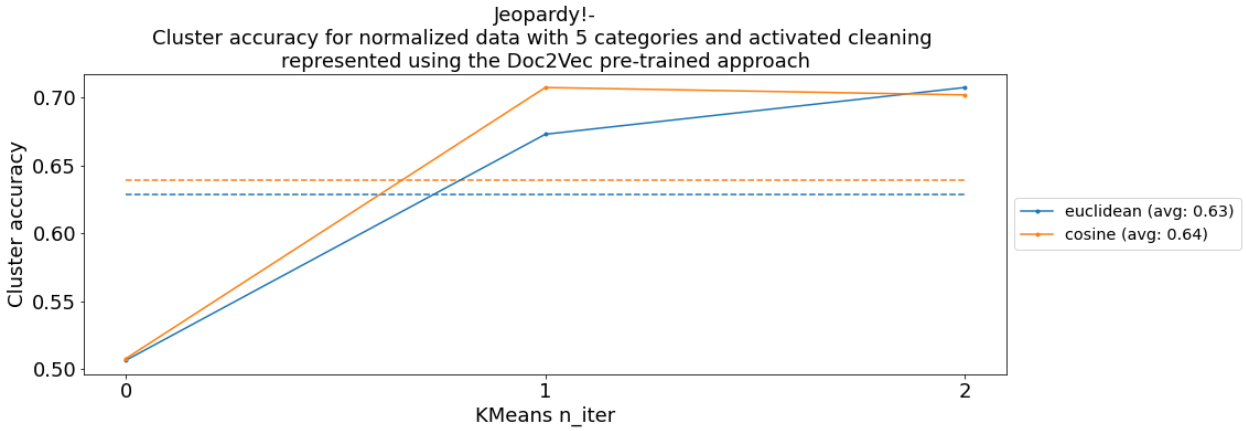
For the second data set, the word embeddings then perform significantly better with a maximum average value of 64% for doc2vec. The TF-IDF encoding using cosine similarity results in an accuracy of 45%. The application of LSA again provides better values (51%) than TF-IDF. The accuracy values are shown in Figure 7.34. This is also shown by the matching table displayed in Figure 7.35. Since the data set contains *History* and *History of the USA* as categories which are very close to each other, as can be seen in Figures 7.37 and 7.36, several targets of these two categories are clustered incorrectly.



(a) TF-IDF

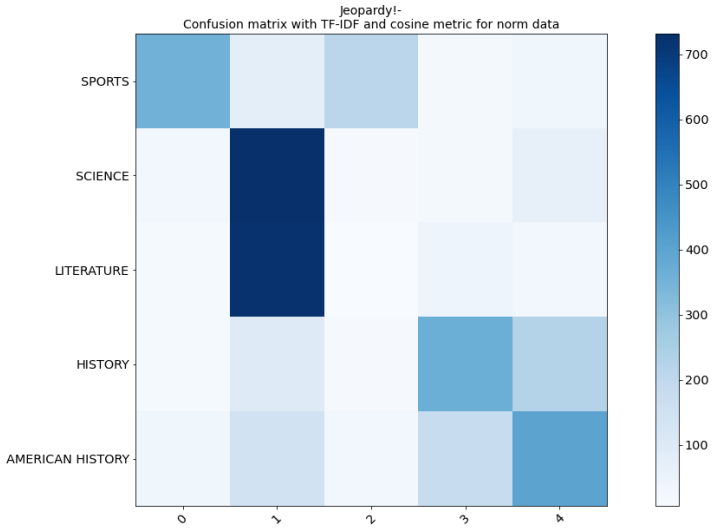


(b) LSA

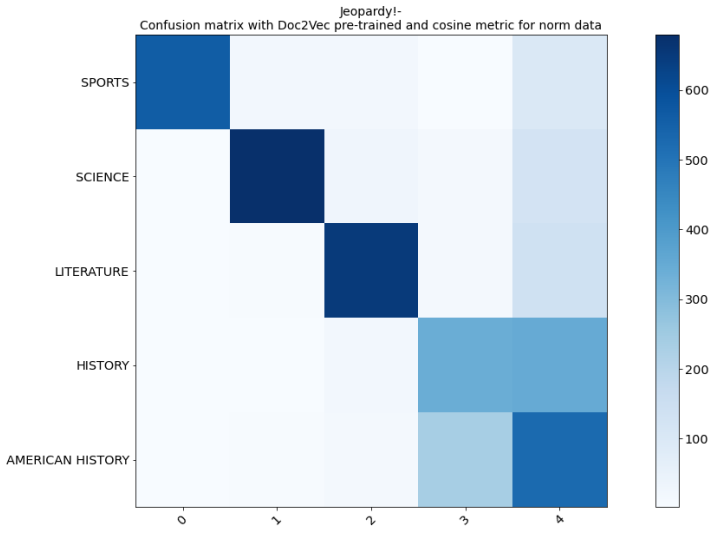


(c) Doc2Vec

Figure 7.34: *Jeopardy!*: Cluster accuracy for pre-processed data using the TF-IDF, LSA and doc2vec approach (`n_categ=5`).

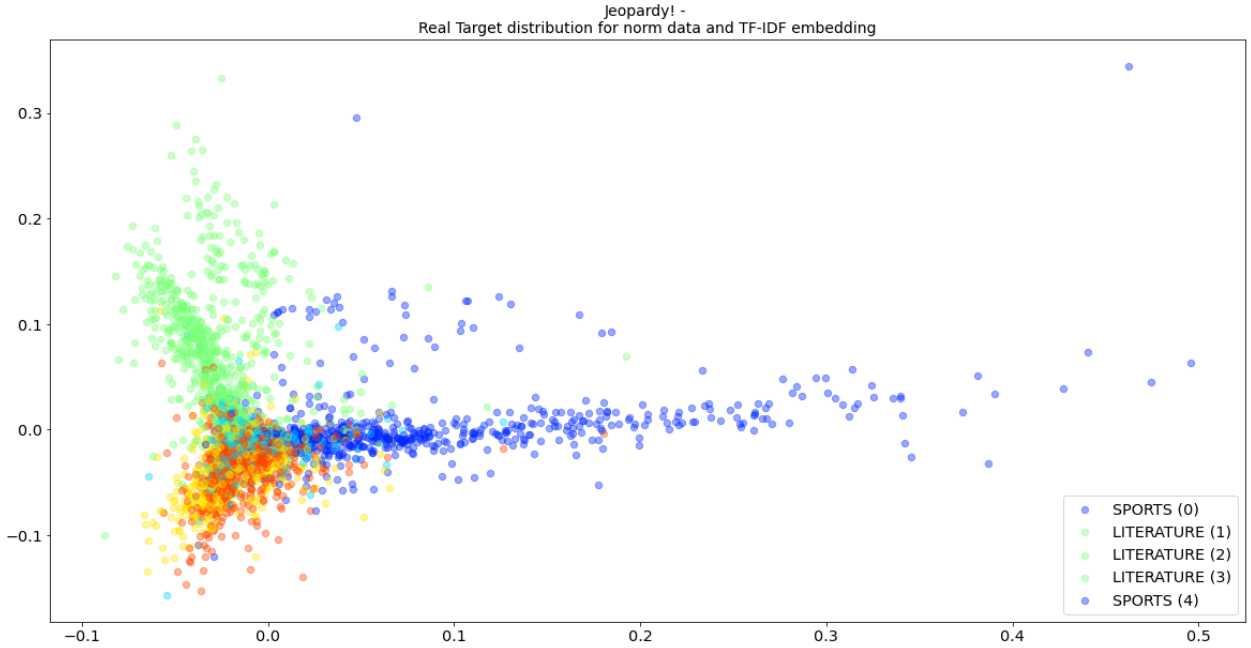


(a) TF-IDF



(b) Doc2Vec

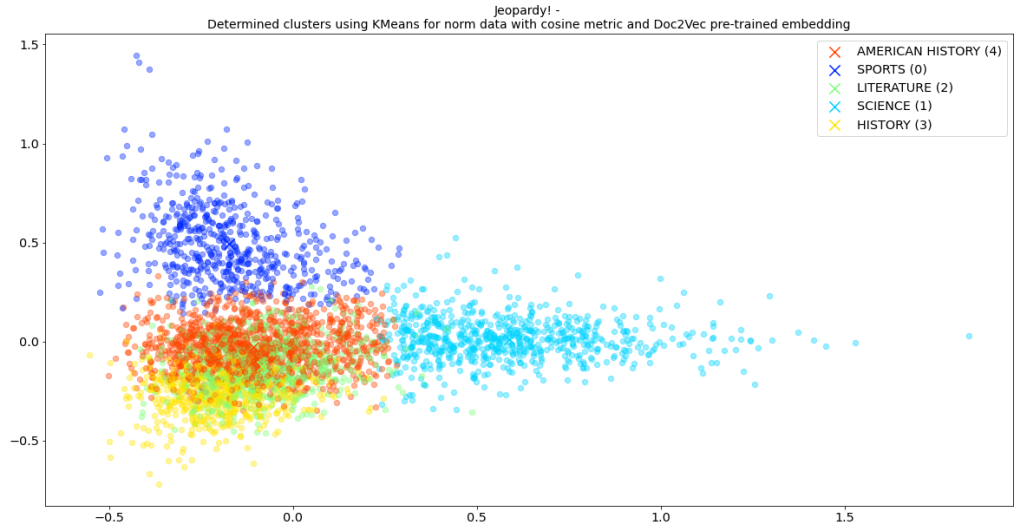
Figure 7.35: *Jeopardy!*: Matching matrix for pre-processed data transformed using TF-IDF and Doc2Vec and cosine similarity.



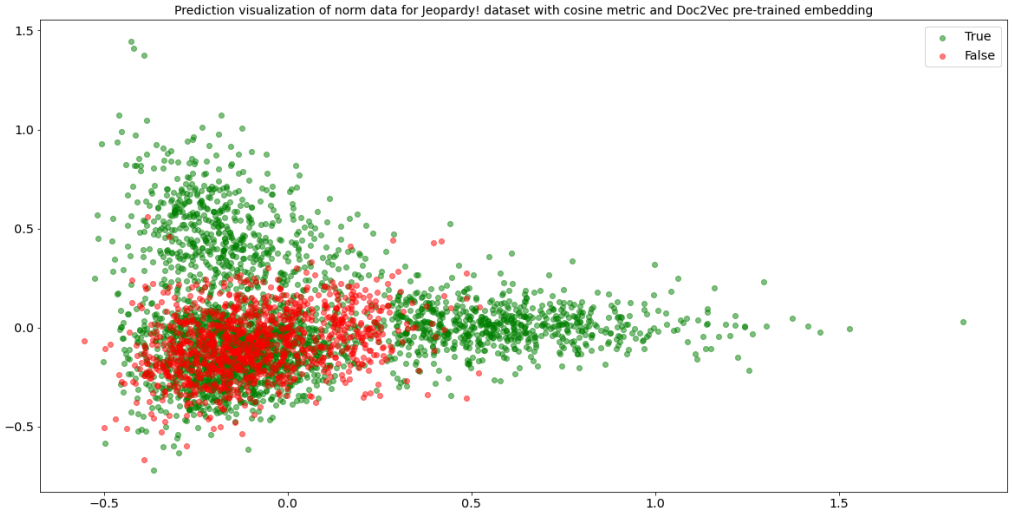
(a) Pre-defined Labels

(b) Custer error

Figure 7.36: *Jeopardy!*: Cluster result and error for pre-processed data transformed with TF-IDF (`n_categ=5`).



(a) Cluster result doc2vec



(b) Custer error

Figure 7.37: *Jeopardy!*: Cluster result and error for pre-processed data transformed with doc2vec (`n_categ=5`).

Referring to the plots shown in sections A.5, A.5 and A.6 of the appendix, the result gets even worse by taking more than 5 clusters into account.

7.5 Reddit

The visualization of the pre-defined labels (see Figure 7.38) already suggests that the cluster results for the *Reddit* data set might not be accurate.

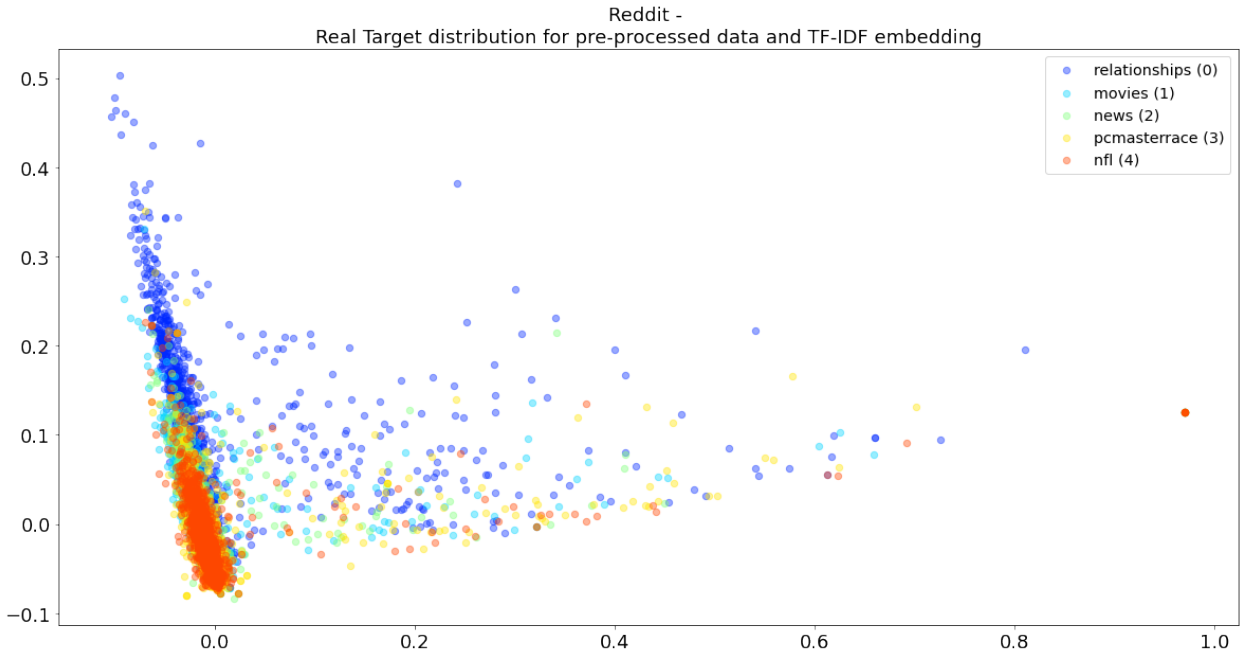
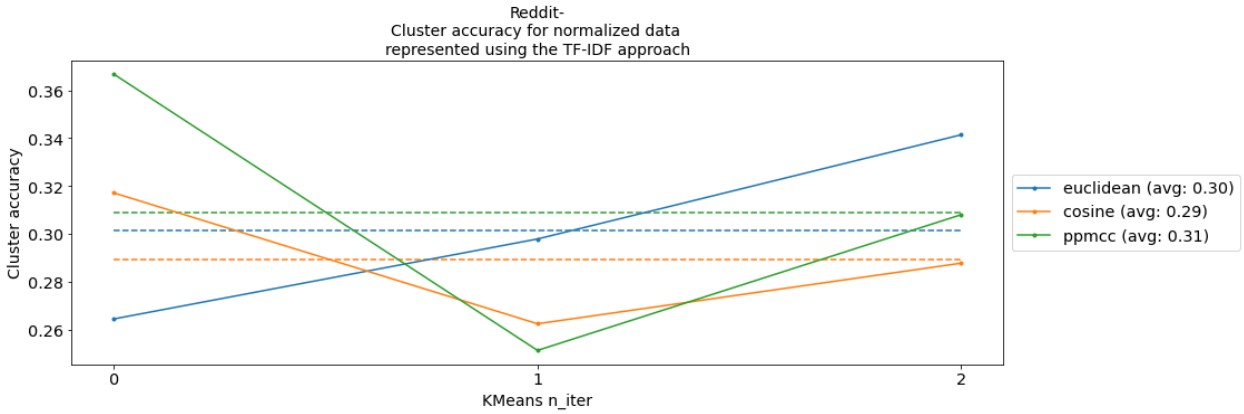
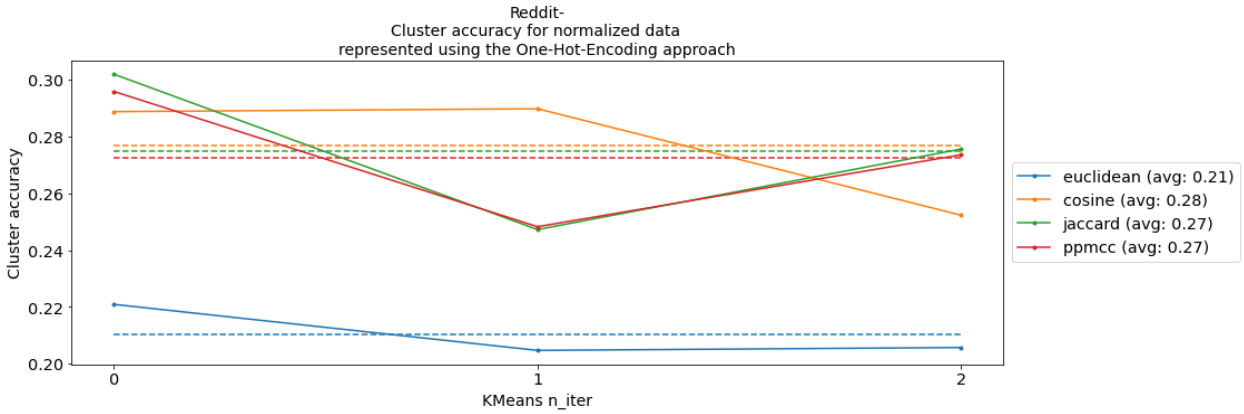


Figure 7.38: *Reddit*: Pre-defined clusters for pre-processed data transformed using TF-IDF (`n_categ=5`).

This is also reflected in Figure 7.39, which shows the accuracies for the different representation methods.

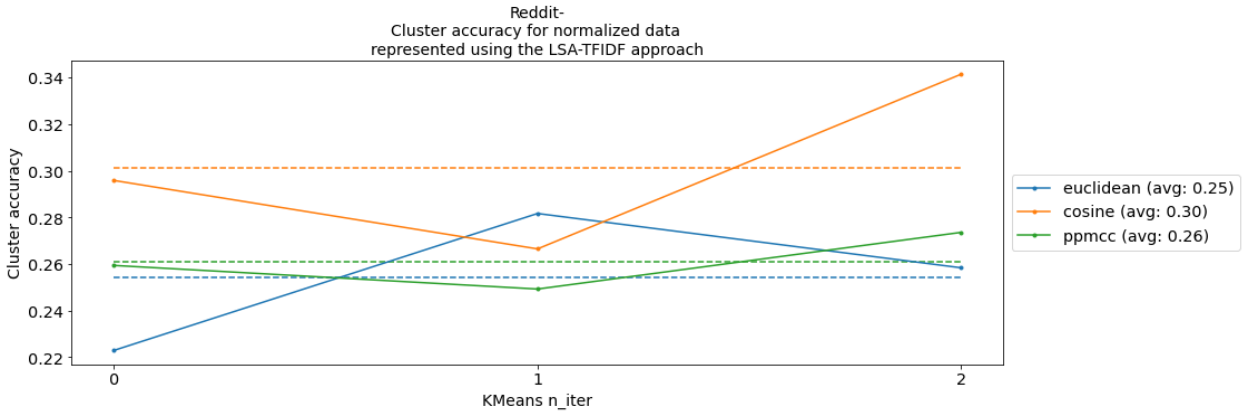


(a) TF-IDF encoding

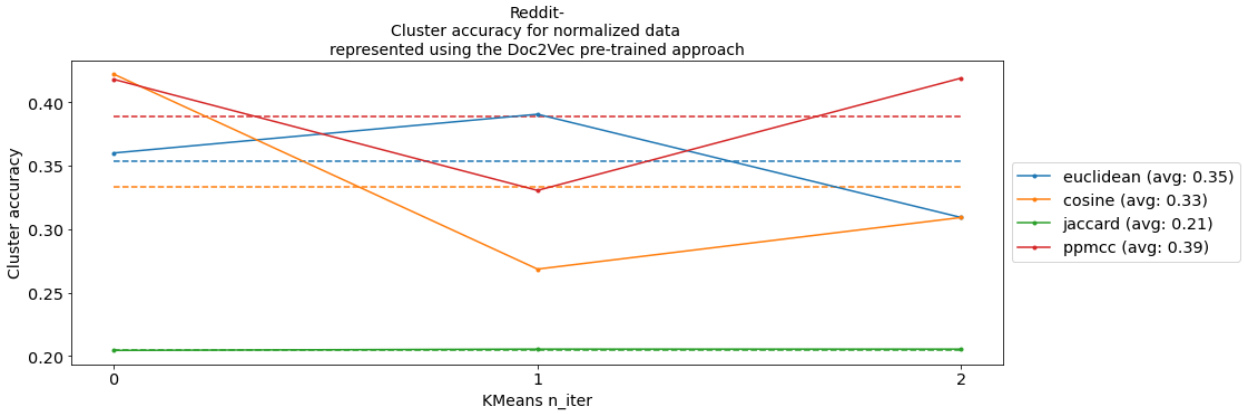


(b) One-hot encoding

Figure 7.39: *Reddit*: Cluster accuracy for pre-processed data using the BoW approach.



(a) LSA



(b) Doc2Vec

Figure 7.40: *Reddit*: Cluster accuracy for pre-processed data using the word embeddings.

CHAPTER 8

Conclusion and Outlook

As a literature review showed, clustering is a widely used technique to partition a set of data to homogeneous and well separated groups. In this thesis, text document clustering was investigated applying the k-Means algorithm to three different data sets using different distance measures and data characteristics (raw, pre-processed, uncleaned, cleaned). The following components that affect the cluster result have been identified:

1. Data characteristics
2. Data pre-processing
3. Transformation of data into word vectors
4. k-Means parameters such as `n_iter` or the metric used to compute the similarity of data objects and clusters

The respective methods were compared and evaluated in experiments. The most important results are summarized in Table 8.1

	TF-IDF		***LSA***	***Doc2Vec***	
	euclidean	**cosine**	**cosine**	**euclidean**	**cosine**
20 newsgroups `n_categ=5`	61	69	82	76	57
Jeopardy! `n_categ=10`	19	35	33	43	43
Reddit `n_categ=5`	30	29	30	35	33

Table 8.1: Accurracy overview of selected parameters (in %).

From the experiment, pre-processing, i.e. cleaning, normalization and stop-word removal, does play a significant role because it results in an improved cluster performance and accuracy. Especially the BoW vectors benefit from pre-processing. When using the BoW representation approach, TF-IDF in combination with the cosine or *PPMCC* metric provides the best results in most cases. Furthermore, it is advantageous to apply LSA to the TF-IDF data. The TSVD reduces the number of features to the most important main components which has among other things a positive effect on the computing time. For example, for the *20 newsgroups* data set a reduction from 39961 to 2169 features was achieved. For the *Jeopardy!* data set 77.5% of the features could be neglected to reach a threshold of 80%. The pretrained word embedding models have also produced useful results. Their advantage is that they contain a much smaller number of features for large data sets. Among other things, this allows faster clustering and of course the determination of the semantic similarity of words and not just the lexical one, as with BoW. Depending on the data set word embeddings have proven to be effective if they contain many clusters with similar topics. Otherwise, however, TF-IDF and even LSA achieved better results. One disadvantage of word embeddings is that a large and cleaned text corpus is required to generate a meaningful vector representation. Furthermore, the algorithms are very computationally intensive. Moreover, the problem arises that these models only recognize words contained in their training corpus and that they need a lot of memory.

8.1 Future Work

The experiments could be extended as follows: First, the effect of text preprocessing could be studied in more detail. In this work the text pre-processing included filtering, normalization, the stop words removal and lemmatization. It could be investigated separately for each data set which pre-processing methods improve the clustering result. Furthermore, instead of the k-Means method, other clustering algorithms, such as the also explained hierarchical methods, could be used. Also with regard to the text representation other ideas could be developed. For example, further research could be carried out to establish the impact of n-grams in terms of the reduced feature space and the clustering result. With regard to word embeddings, other pre-trained models could be used or own models especially for the available data sets could be trained.

Bibliography

[1] *20 newsgroups*. URL: http://qwone.com/~jason/20Newsgroups/ (visited on 04/08/2020).

[2] *200,000+ Jeopardy Questions*. URL: https://knowledge.domo.com/Training/Self-Service_Training/Onboarding_Resources/Fun_Sample_Datasets (visited on 04/08/2020).

[3] C.C. Aggarwal and C.K. Reddy. *Data Clustering: Algorithms and Applications*. CRC Press, 2016. ISBN: 9781498785778. URL: https://books.google.com/books?id=p8b1CwAAQBAJ.

[4] F. N. A Al Omran and C. C. Treude. "Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017, pp. 187–197.

[5] M. Allahyari et al. "A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques". In: *CoRR* (2017).

[6] F. Almeida and G. Xexéo. "Word Embeddings: A Survey". In: *CoRR* abs/1901.09069 (2019). URL: http://arxiv.org/abs/1901.09069.

[7] Sumayia Al-Anazi, Hind AlMahmoud, and Isra Al-Turaiki. "Finding Similar Documents Using Different Clustering Techniques". In: *Procedia Computer Science* 82 (2016). ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.04.005. URL: http://www.sciencedirect.com/science/article/pii/S1877050916300199.

[8] D. Arthur and S. Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: vol. 8. Jan. 2007, pp. 1027–1035. DOI: 10.1145/1283383.1283494.

[9] Y. Bengio et al. "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res.* 3 (2003). ISSN: 1532-4435.

[10] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business, LLC, 2006. ISBN: 9780387310732.

[11] R.C. Blattberg, B.D. Kim, and S.A. Neslin. *Database Marketing: Analyzing and Managing Customers*. International Series in Quantitative Marketing. Springer New York, 2010. ISBN: 9780387725796. URL: https://books.google.de/books?id=xLq8BAAAQBAJ.

[12] P. Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *CoRR* abs/1607.04606 (2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606.

[13] K.R. Bokka et al. *Deep Learning for Natural Language Processing: Solve your natural language processing problems with smart deep neural networks*. Packt Publishing, 2019. ISBN: 9781838553678. URL: https://books.google.com/books?id=HxmdDwAAQBAJ.

[14] S. Curiskis et al. *Data for:An evaluation of document clustering and topic modelling in two online social networks: Twitter and Reddit*. 2019. URL: https://data.mendeley.com/datasets/85njyhj45m/1 (visited on 04/10/2020).

[15] *Data Never Sleeps 7.0*. 2019. URL: https://www.domo.com/learn/data-never-sleeps-7 (visited on 02/16/2020).

[16] S. Deerwester et al. "Indexing by latent semantic analysis". In: *Journal of the American Society for Information Science* 41.6 (1990). DOI: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9. URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9.

[17] *English Available pretrained statistical models for English.* URL: https://spacy.io/models/en#en_core_web_sm (visited on 04/14/2020).

[18] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. "From Data Mining to Knowledge Discovery in Databases". In: *AI Magazine* 17.3 (1996). DOI: 10.1609/aimag.v17i3.1230. URL: https://wvvw.aaai.org/ojs/index.php/aimagazine/article/view/1230.

[19] J.R. Firth. *A Synopsis of Linguistic Theory, 1930-1955.* 1957. URL: https://books.google.com/books?id=T8LDtgAACAAJ.

[20] Y. Goldberg and G. Hirst. *Neural Network Methods in Natural Language Processing.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2017. ISBN: 9781627052955. URL: https://books.google.com/books?id=Za2zDgAAQBAJ.

[21] Wael H. Gomaa and Aly A. Fahmy. "Article: A Survey of Text Similarity Approaches". In: *International Journal of Computer Applications* 68 (2013).

[22] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. "On Clustering Validation Techniques". In: *Journal of Intelligent Information Systems* 17 (2001), pp. 107–145.

[23] A. Huang. "Similarity measures for text document clustering". In: *Proceedings of the 6th New Zealand Computer Science Research Student Conference* (Jan. 2008).

[24] T. Ah-hwee. *Text Mining: The state of the art and the challenges.* 1999.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data Clustering: A Review". In: *ACM Comput. Surv.* 31.3 (1999). ISSN: 0360-0300. DOI: 10.1145/331499.331504. URL: https://doi.org/10.1145/331499.331504.

[26] D. Jurafsky et al. *Speech and Language Processing.* Pearson Education, 2014. ISBN: 9780133252934. URL: https://books.google.com/books?id=Cq2gBwAAQBAJ.

[27] A.I. Kadhim, Cheah Y., and N.H. Ahamed. "Text Document Preprocessing and Dimension Reduction Techniques for Text Document Clustering". In: (2014). DOI: 10.1109/ICAIET.2014.21.

[28] D. Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: http://www.dkriesel.com/en/science/neural_networks.

[29] A. Kumar and S. Chandrasekhar. "Text Data Pre-processing and Dimensionality Reduction Techniques for Document Clustering". In: 1 (2012). ISSN: 2278-0181.

[30] E. Kumar. *Natural Language Processing*. I.K. International Publishing House Pvt. Limited, 2011. ISBN: 9789380578774. URL: https://books.google.com/books?id=FpUBFNFuKWgC.

[31] J. Kun, J. Xu, and B. He. "A Survey on Neural Network Language Models". In: (June 2019).

[32] Q.V. Le and T. Mikolov. "Distributed Representations of Sentences and Documents". In: *CoRR* abs/1405.4053 (2014). arXiv: 1405.4053. URL: http://arxiv.org/abs/1405.4053.

[33] Vijaymeena M K and Kavitha K. "A Survey on Similarity Measures in Text Mining". In: *Machine Learning and Applications: An International Journal* 3 (Mar. 2016), pp. 19–28. DOI: 10.5121/mlaij.2016.3103.

[34] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Series in Solid-State Sciences. Springer US, 2010. ISBN: 9780387098234. URL: https://books.google.com/books?id=alHIsT6LBl0C.

[35] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 9781139472104. URL: https://books.google.de/books?id=t1PoSh4uwVcC.

[36] N. Matthews. "Measurement, Levels of". In: (Jan. 2017). DOI: 10.1002/9781118901731.

[37] T. Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[38] J. Moor. "The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years". In: *AI Magazine* 27.4 (2006). DOI: 10.1609/aimag.v27i4.1911. URL: https://www.aaai.org/ojs/index.php/aimagazine/article/view/1911.

[39] D. Pham, S. Dimov, and C. Nguyen. "Selection of K in K -means clustering". In: *Proceedings of The Institution of Mechanical Engineers Part C-journal of Mechanical Engineering Science - PROC INST MECH ENG C-J MECH E* 219 (Jan. 2005), pp. 103–119. DOI: 10.1243/095440605X8298.

[40] D. Reinsel, J. Gantz, and J. Rydning. "Data Age 2025: The evolution of Data to Life-Critical". In: *IDC White Paper* (2017). URL: https://assets.ey.com/content/dam/ey-sites/ey-com/en_gl/topics/workforce/Seagate-WP-DataAge2025-March-2017.pdf.

[41] H. Rubenstein and J.B. Goodenough. "Contextual Correlates of Synonymy". In: *Commun. ACM* 8.10 (1965). ISSN: 0001-0782. DOI: 10.1145/365628.365657. URL: https://doi.org/10.1145/365628.365657.

[42] D. Sailaja et al. "An Overview of Pre-Processing Text Clustering Methods". In: 6 (2015). ISSN: 0975-9646.

[43] EMC Education Services. *Data Science & Big Data Analytics. Discovering, Analyzing, Visualizing and Presenting Data*. John Wiley & Sons, Ltd, 2015. ISBN: 9781119183686. DOI: 10.1002/9781119183686.ch9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119183686.ch9.

[44] M.G. Siegler. *Eric Schmidt: Every 2 Days We Create As Much Information As We Did Up To 2003*. 2010. URL: https://techcrunch.com/2010/08/04/schmidt-data/ (visited on 04/02/2020).

[45] *spaCy 101: Everything you need to know*. URL: https://spacy.io/usage/spacy-101 (visited on 04/20/2020).

[46] Lilian Weng. *Learning Word Embedding*. URL: https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html (visited on 04/18/2020).

[47] WT Williams. "Principles of clustering". In: *Annual Review of Ecology and Systematics* 2 (1971), pp. 303–326.

[48] *word2vec*. URL: https://code.google.com/archive/p/word2vec/ (visited on 04/18/2020).

[49] *word2vec*. URL: https://www.kaggle.com/yoshikiohira/doc2vec-english-binary-file (visited on 04/18/2020).

[50] J. Wu. *Advances in K-means Clustering: A Data Mining Thinking*. Springer Theses. Springer Berlin Heidelberg, 2012. ISBN: 9783642298073.

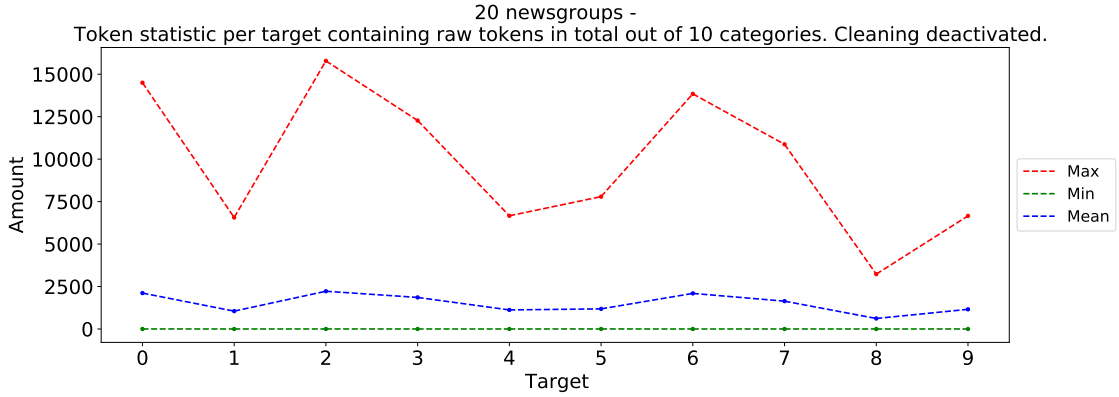
APPENDIX A

Images

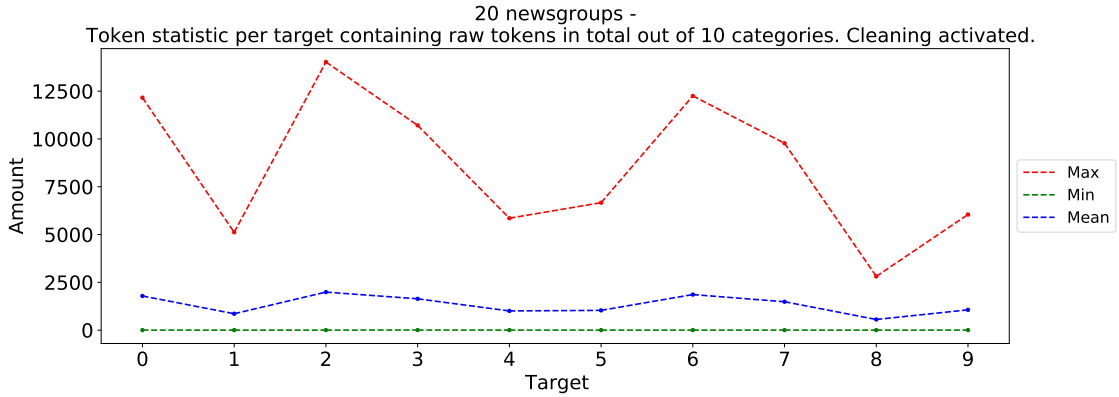
A.1 Data never sleeps 7.0

Figure A.1: Data never sleeps 7.0 - The most popular platforms where data is generated every minute in 2019. Image source [15].

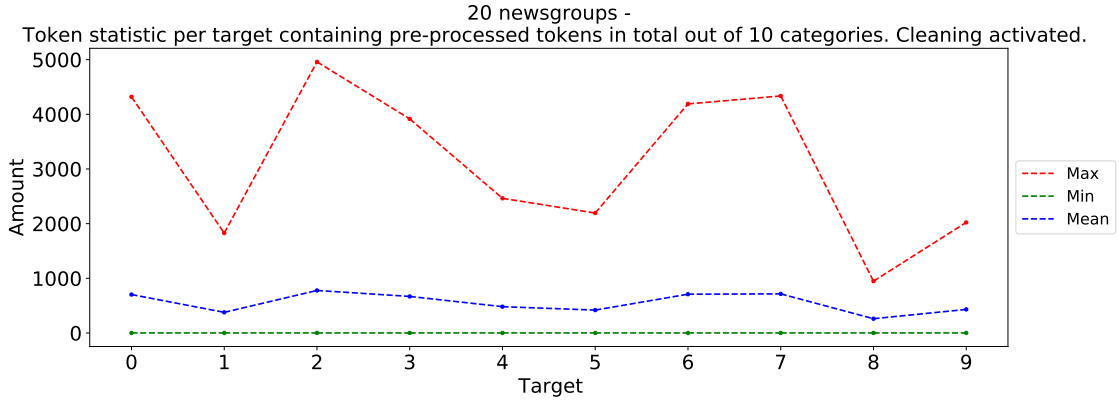
A.2 Tokenization Result of data sets



(a) Uncleaned raw data

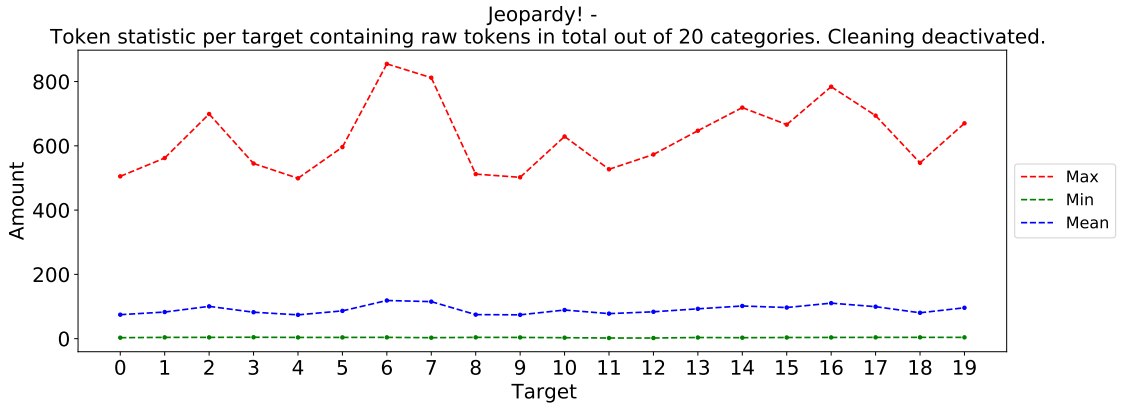


(b) Cleaned raw data

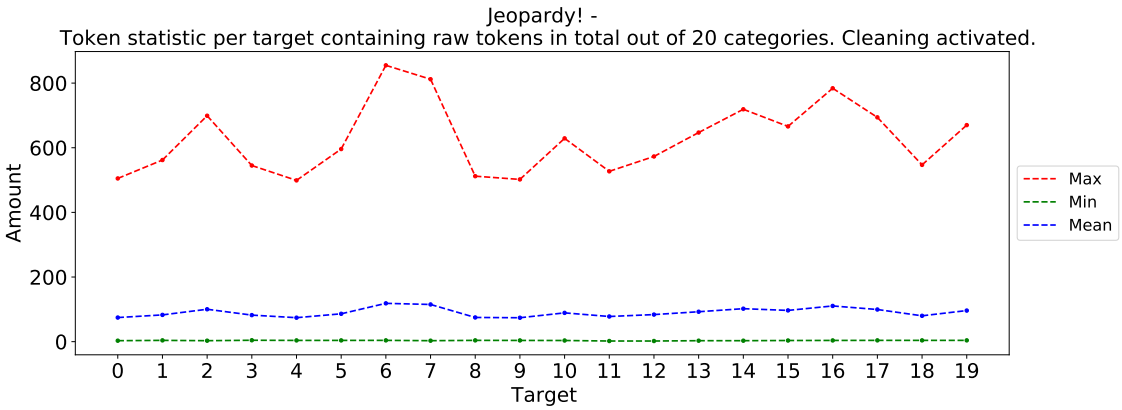


(c) Cleaned pre-processed data

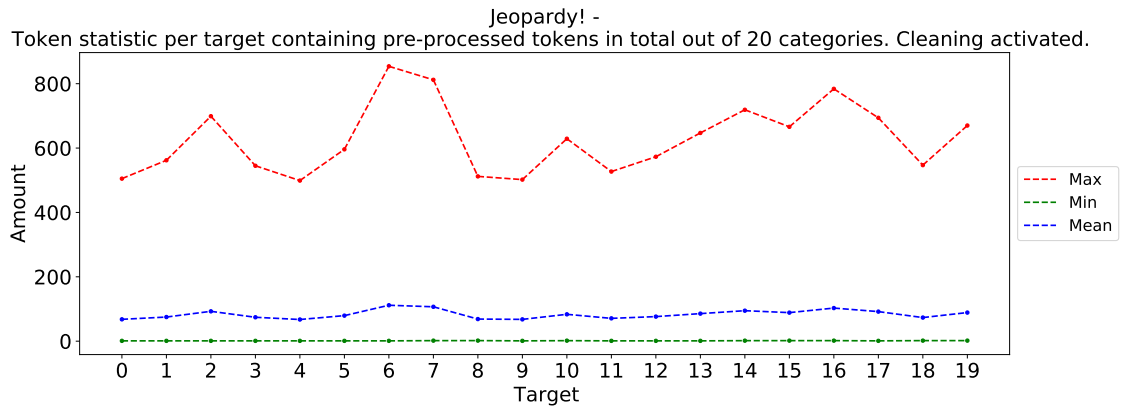
Figure A.2: Maximum, minimum and average number of tokens of *20 newsgroups* data set (`n_categ=10`).



(a) Uncleaned data

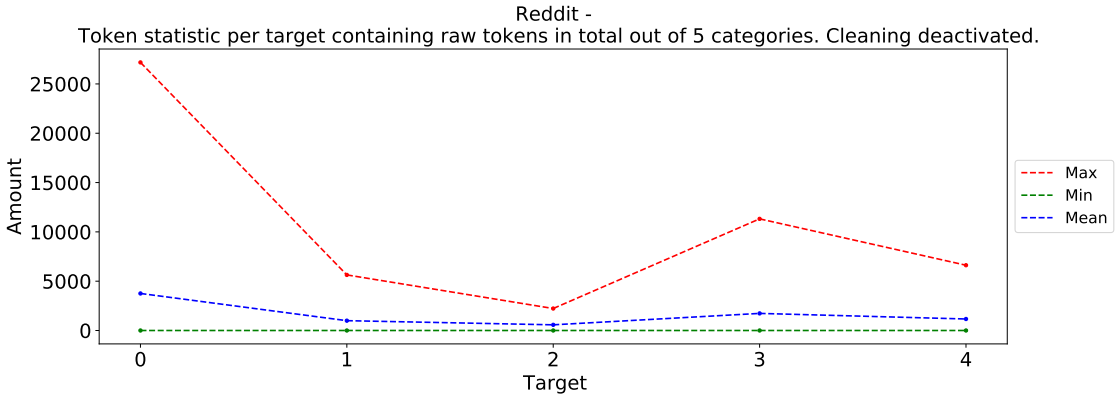


(b) Cleaned data

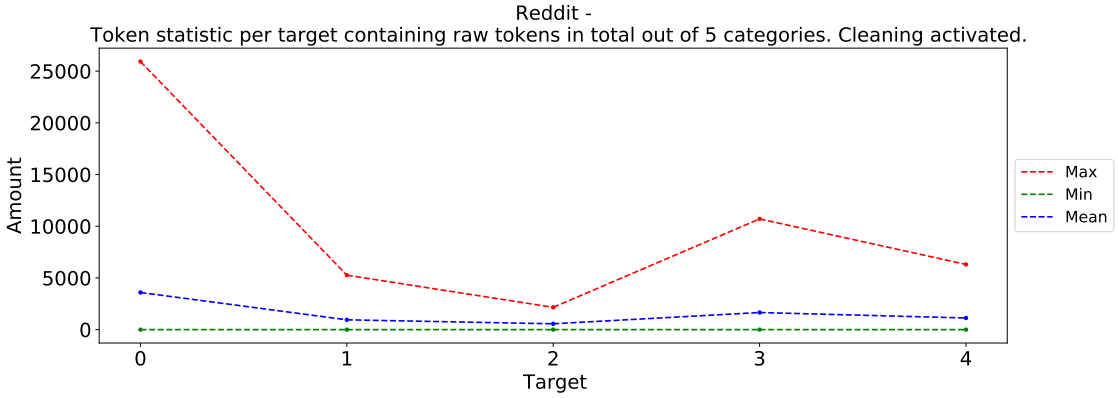


(c) Cleaned data

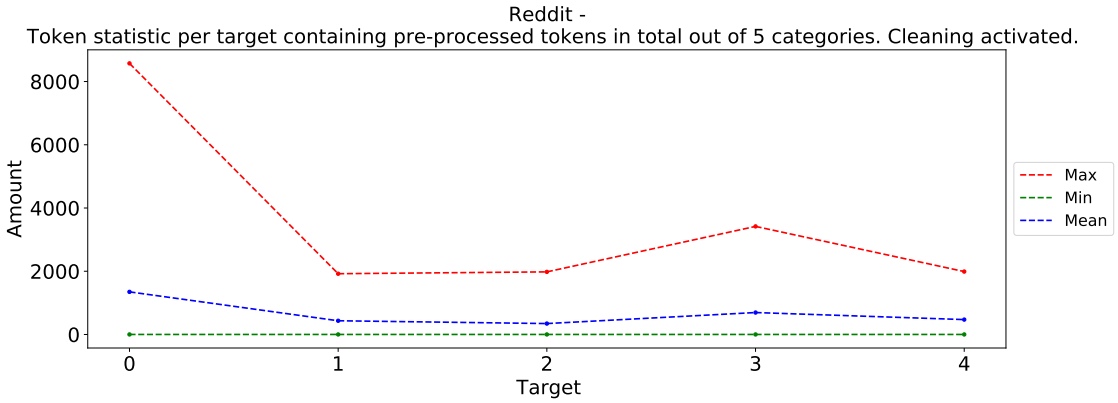
Figure A.3: Maximum, minimum and average number of tokens of *Jeopardy* data set (`n_categ=20`).



(a) Uncleaned data



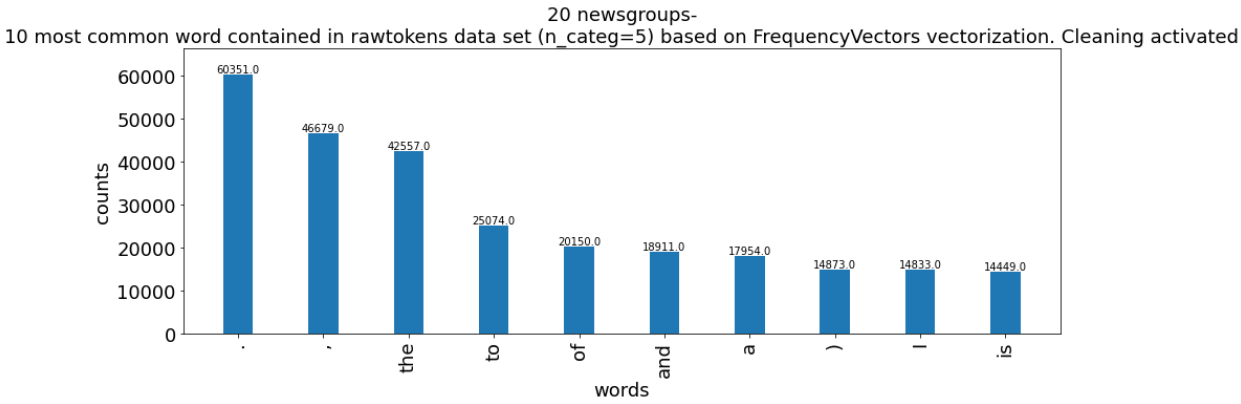
(b) Cleaned data



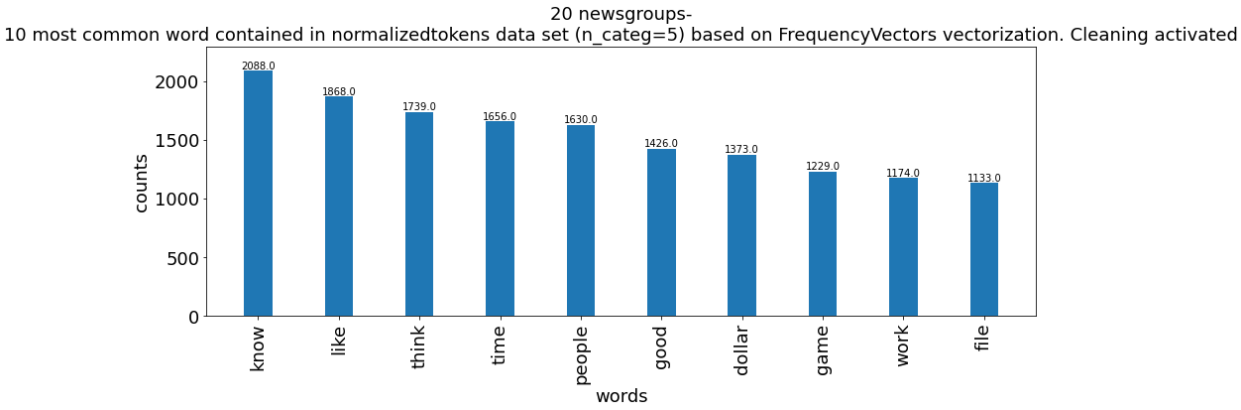
(c) Cleaned data

Figure A.4: Maximum, minimum and average number of tokens of *Reddit* data set (`n_categ=5`).

A.2.1 10 most common words of data sets using different BoW representation techniques

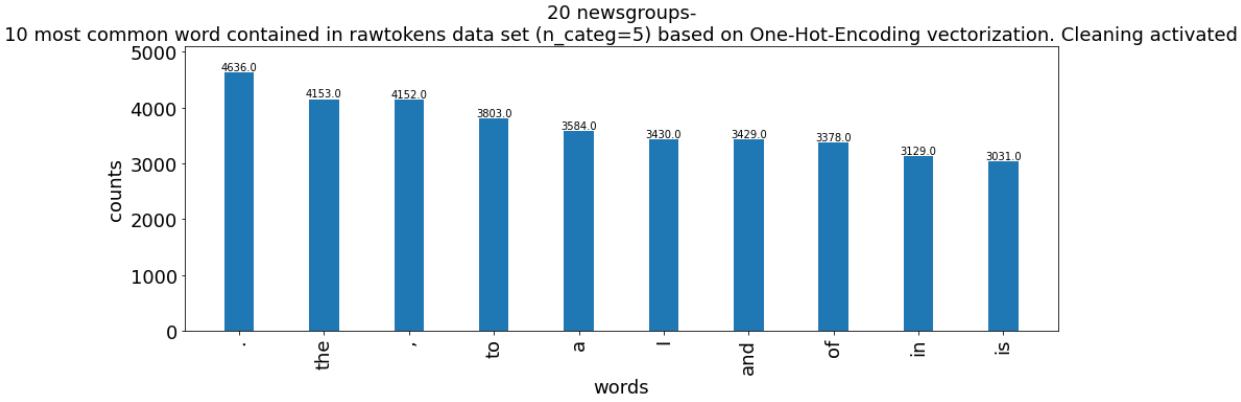


(a) Raw documents

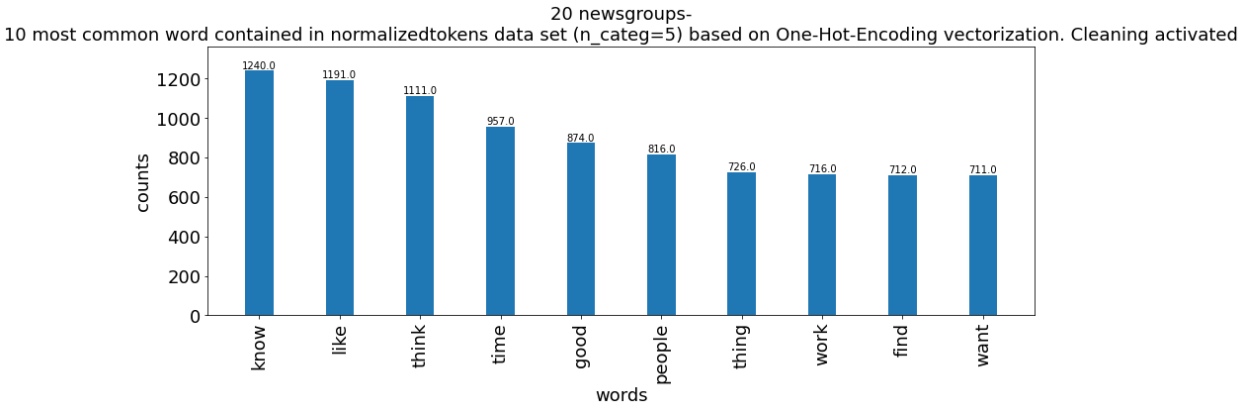


(b) Pre-processed documents

Figure A.5: 10 most common words of *20 newsgroups* with term frequency encoding.

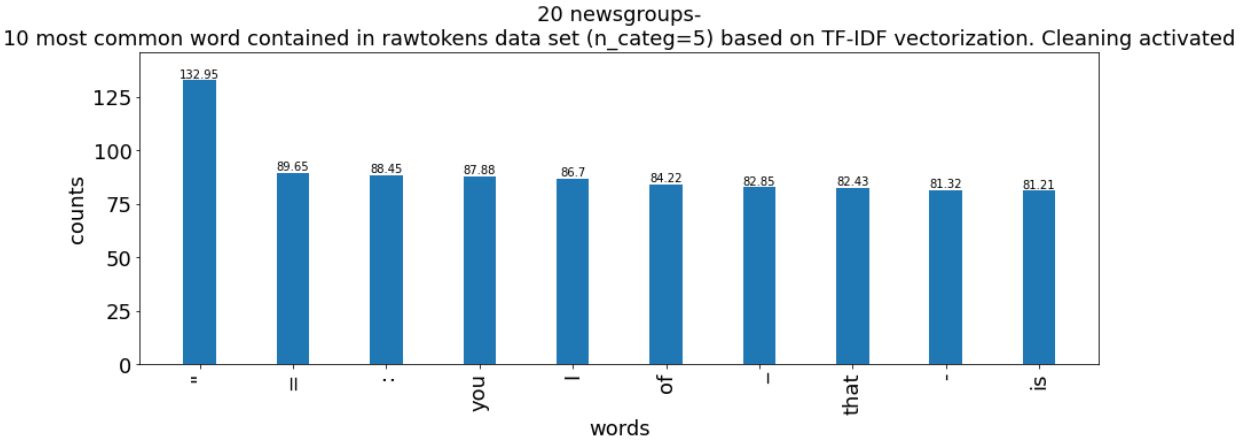


(a) Raw documents

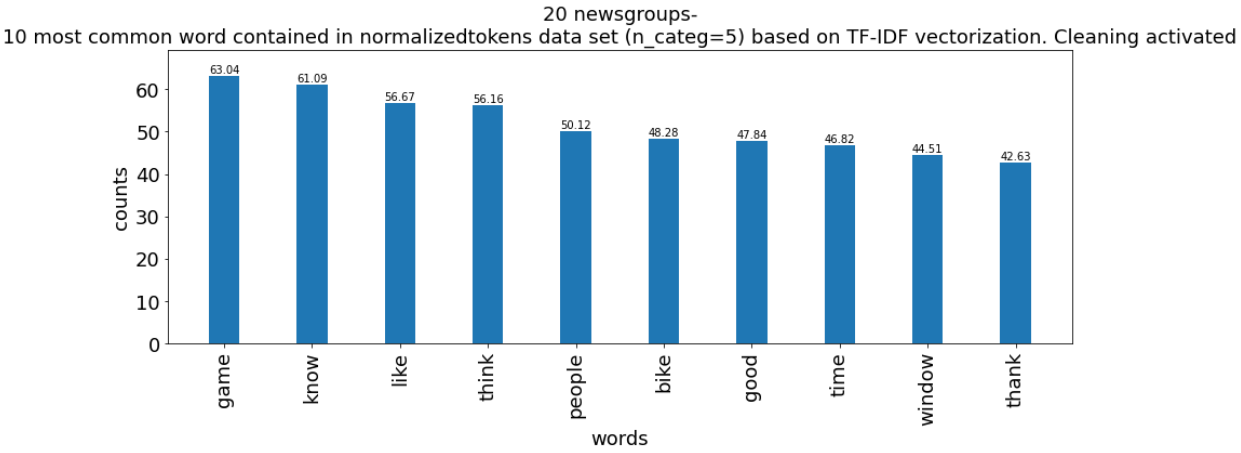


(b) Pre-processed documents

Figure A.6: 10 most common words of *20 newsgroups* with one hot encoding.

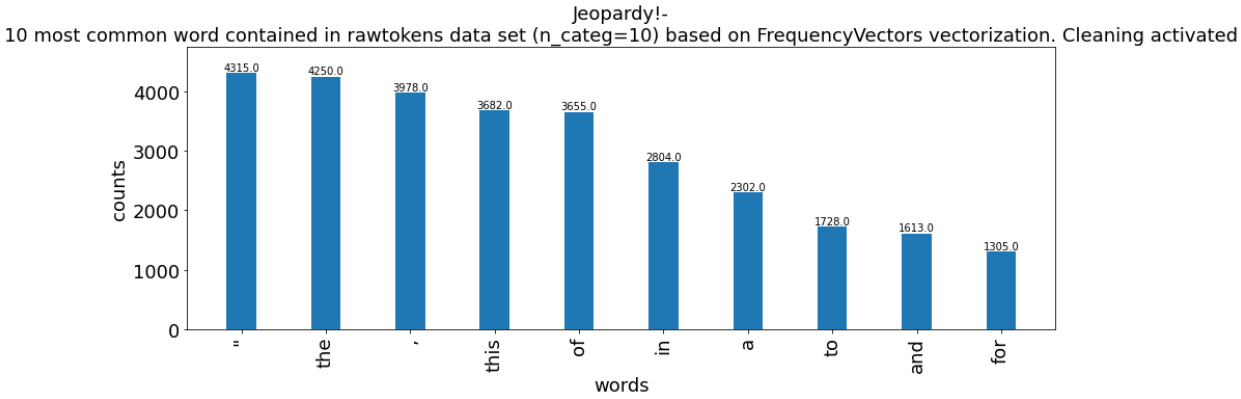


(a) Raw documents

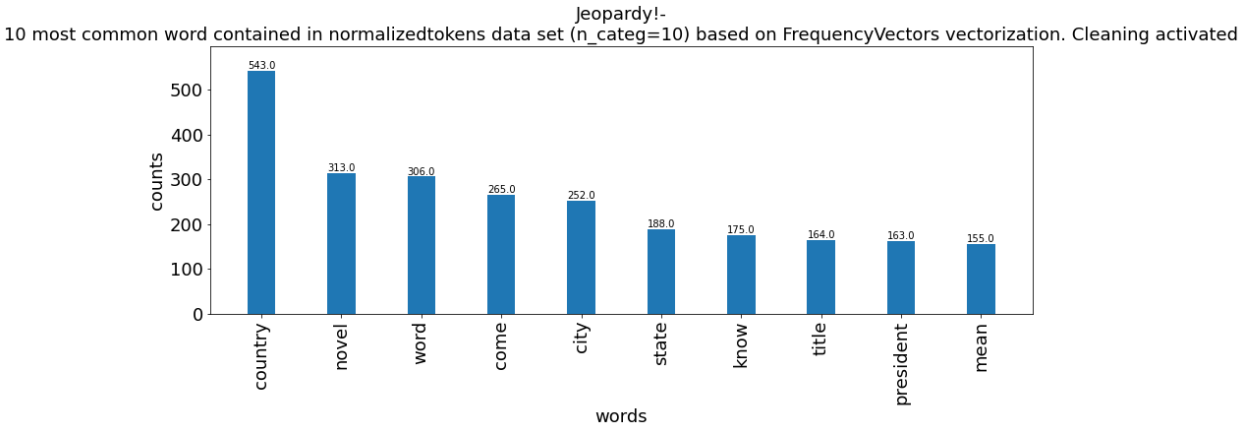


(b) Pre-processed documents

Figure A.7: 10 most common words of *20 newsgroups* with TF-IDF encoding.

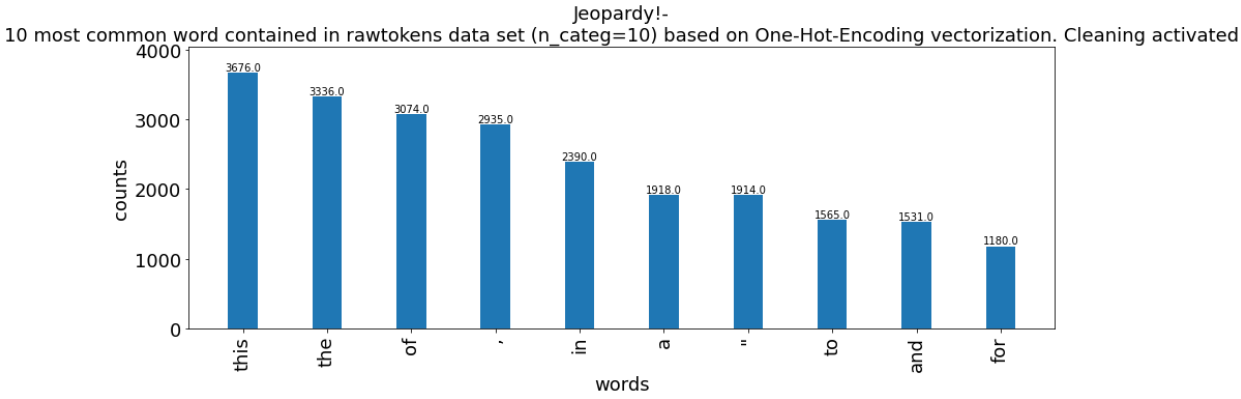


(a) Raw documents

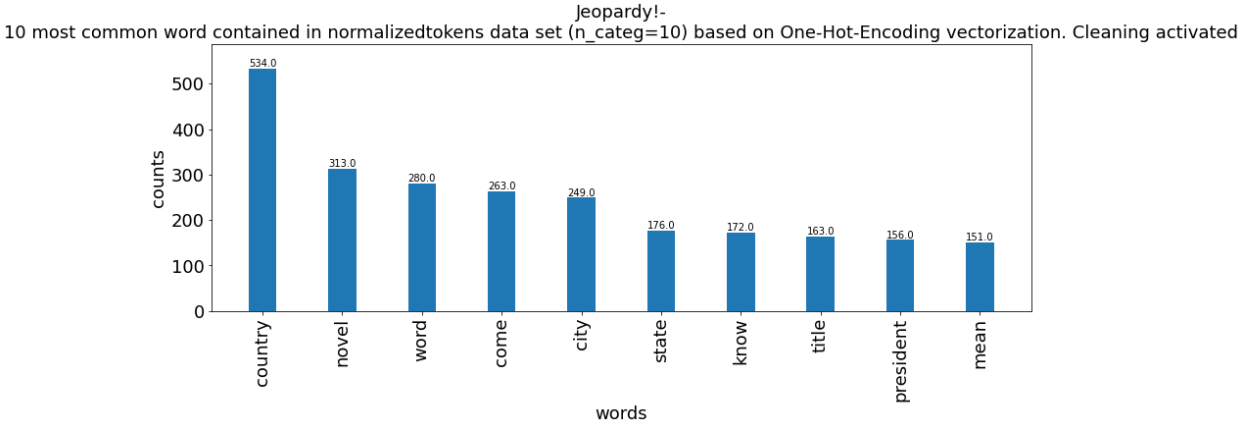


(b) Pre-processed documents

Figure A.8: 10 most common words of *Jeopardy!* with term frequency encoding.

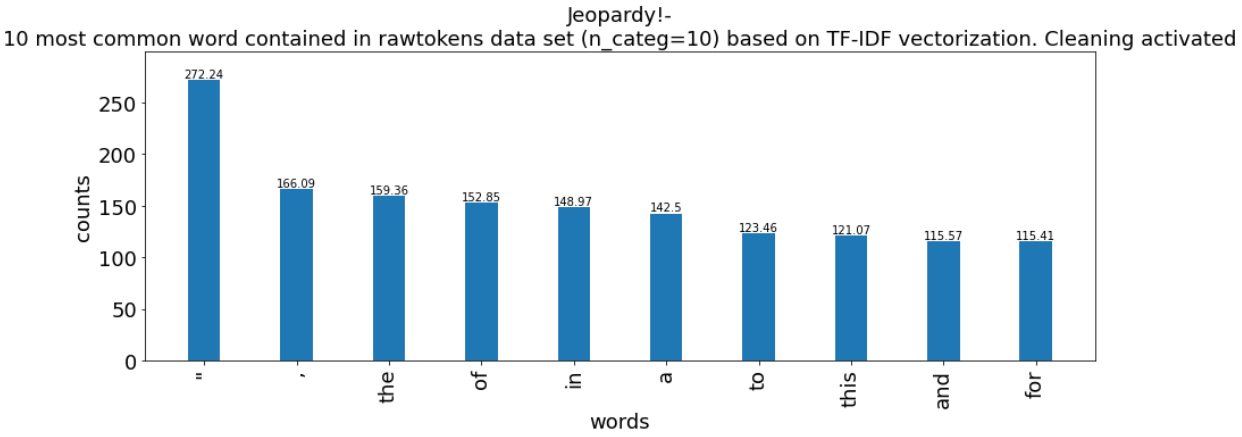


(a) Raw documents

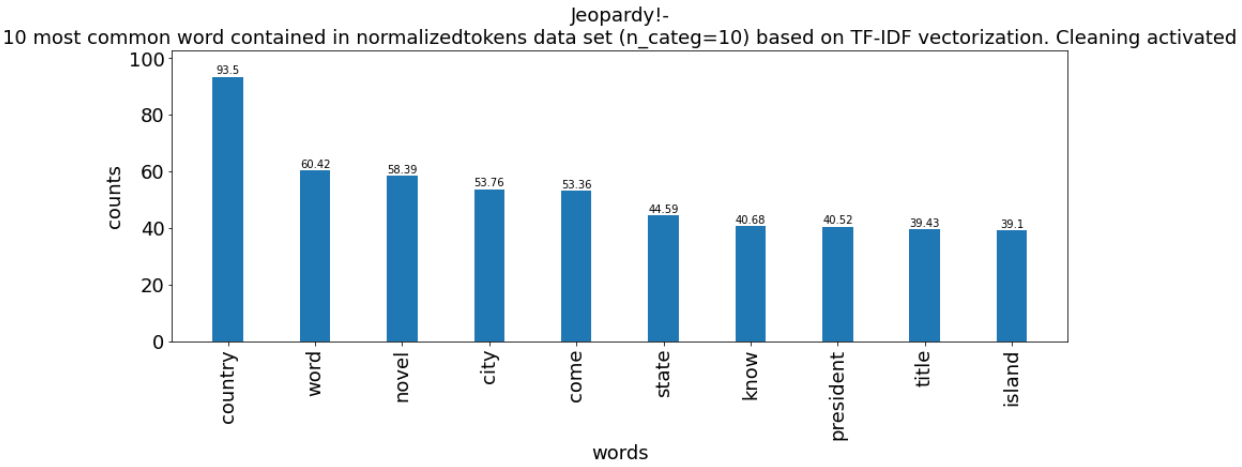


(b) Pre-processed documents

Figure A.9: 10 most common words of *Jeopardy!* with one-hot encoding.

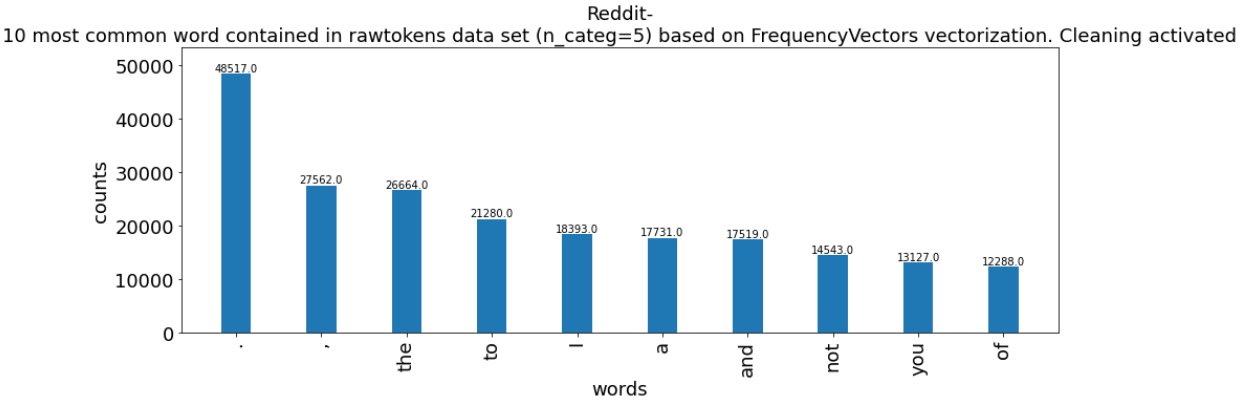


(a) Raw documents

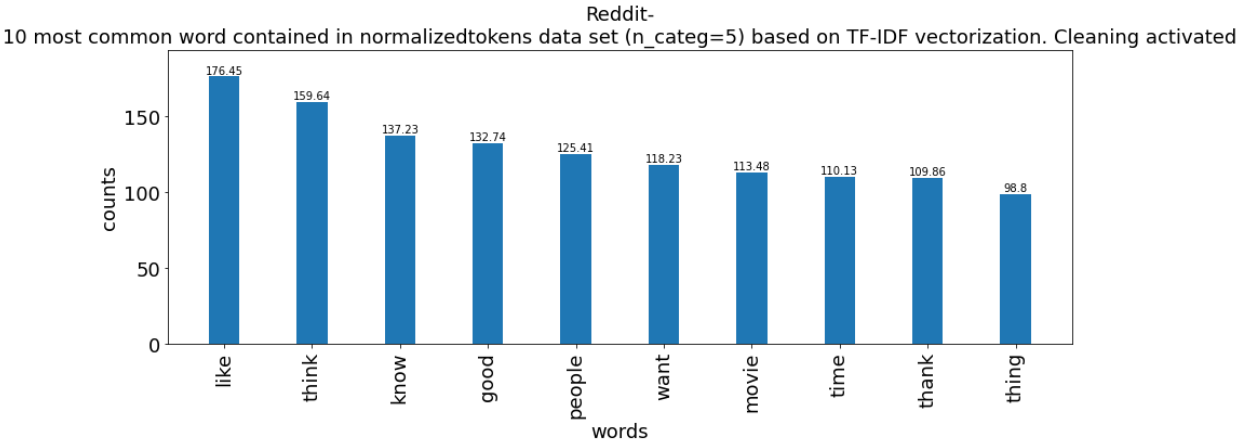


(b) Pre-processed documents

Figure A.10: 10 most common words of *Jeopardy!* with TF-IDF encoding.

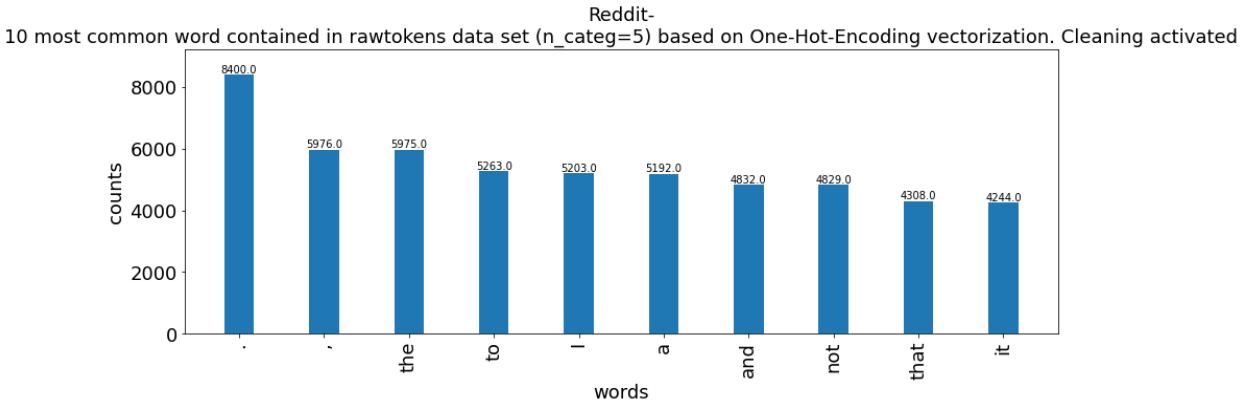


(a) Raw documents

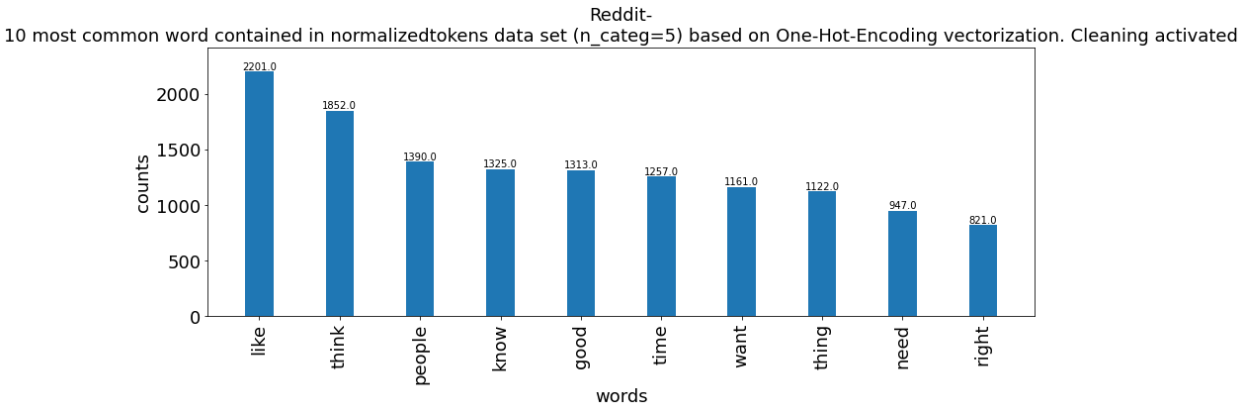


(b) Pre-processed documents

Figure A.11: 10 most common words of *Reddit* with term frequency encoding.

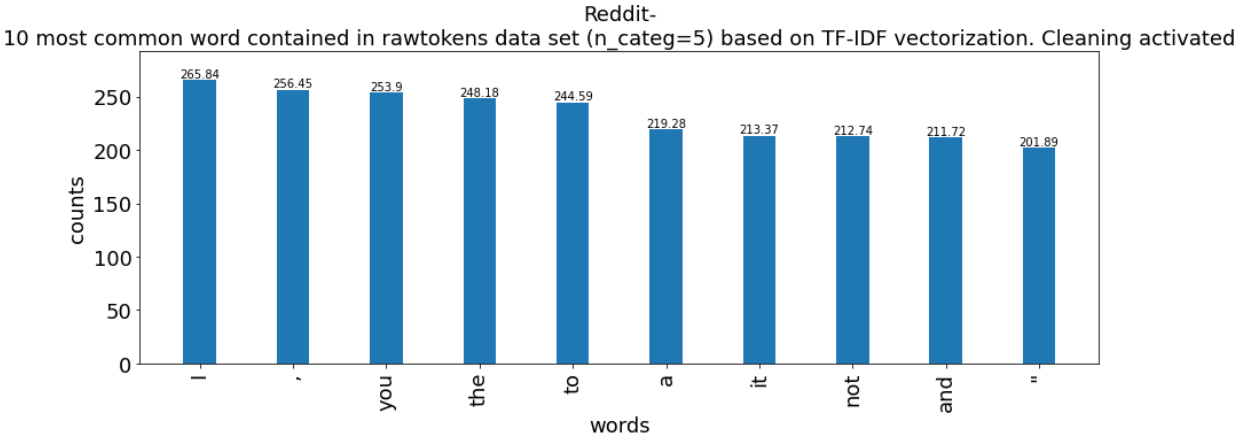


(a) Raw documents

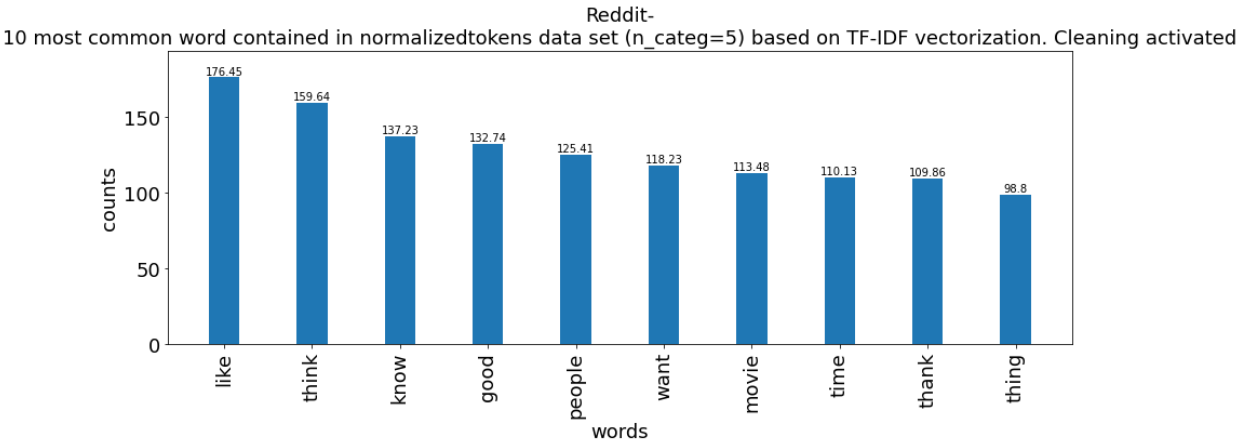


(b) Pre-processed documents

Figure A.12: 10 most common words of *Reddit* with one-hot encoding.



(a) Raw documents



(b) Pre-processed documents

Figure A.13: 10 most common words of *Reddit* with TF-IDF encoding.

A.2.2 k-Means result accuracy for *20 newsgroups* data set using BoW representation and uncleaned data

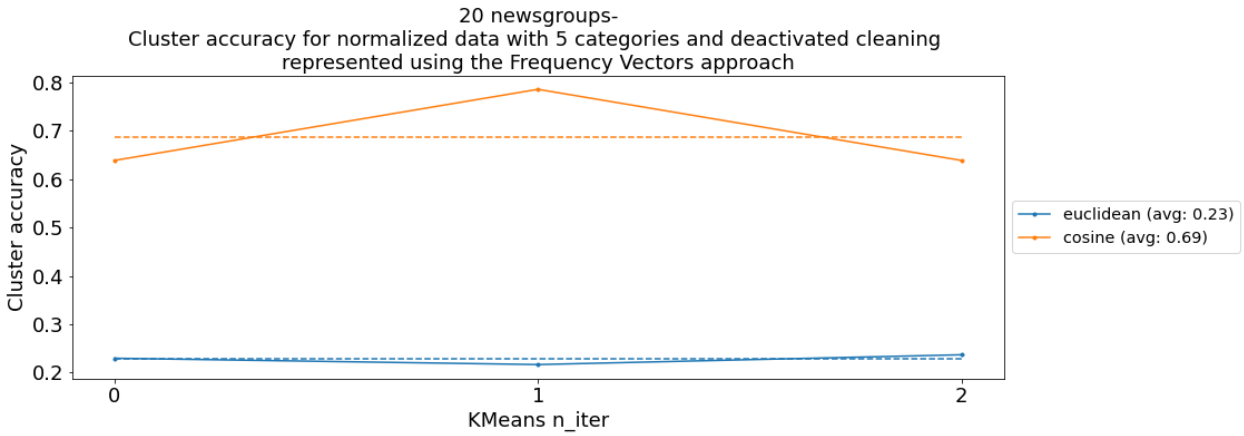


Figure A.14: *20 newsgroups*: Cluster accuracy for pre-processed data using term-frequency encoding.

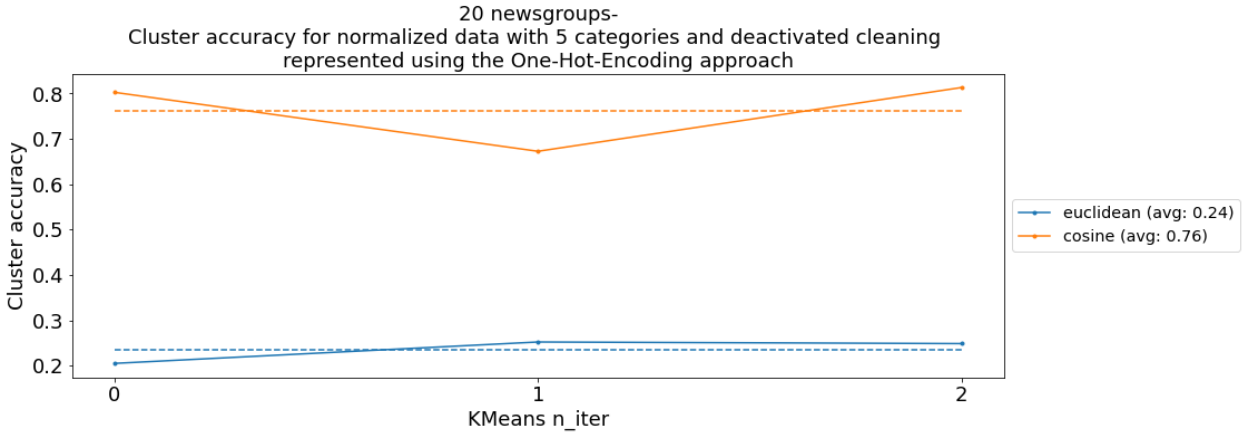


Figure A.15: *20 newsgroups*: Cluster accuracy for pre-processed data using one-hot encoding.

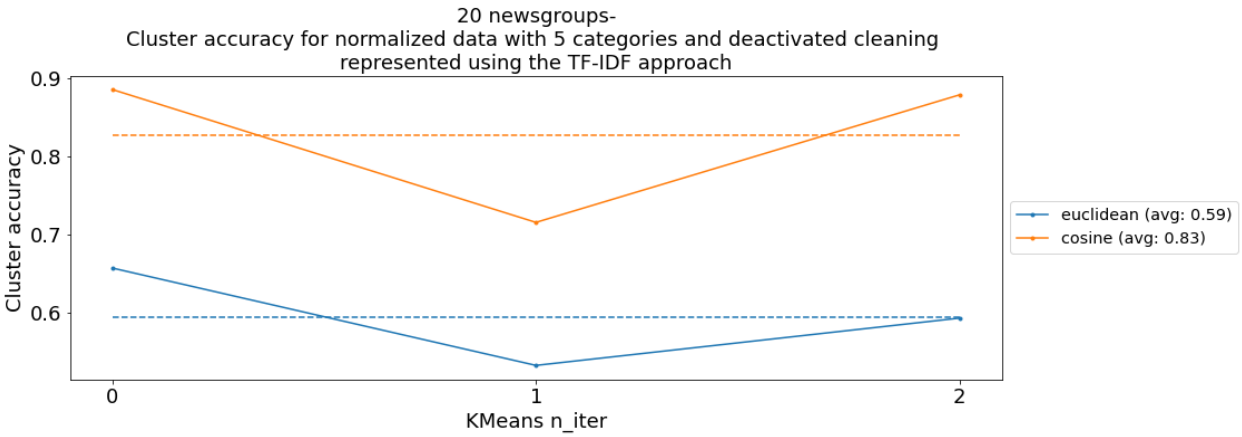
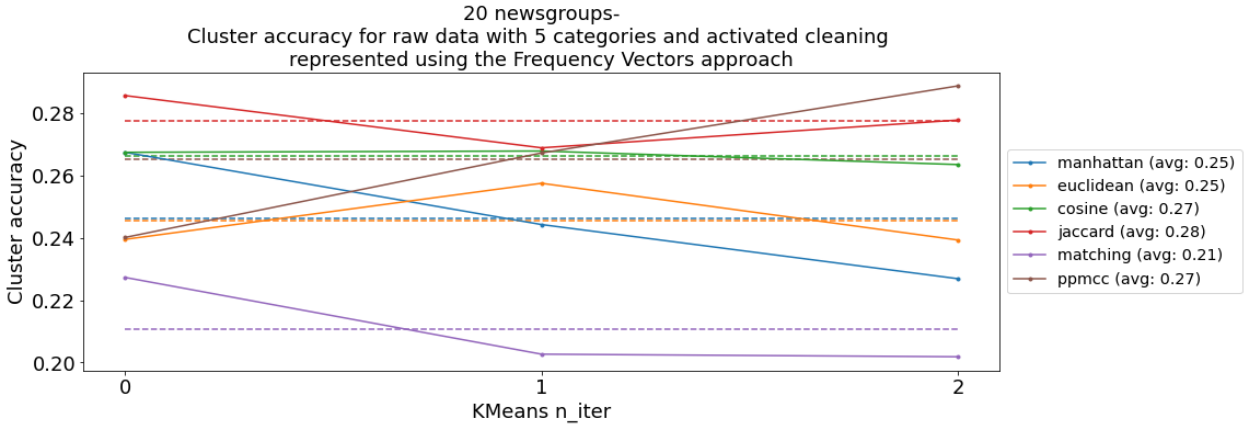
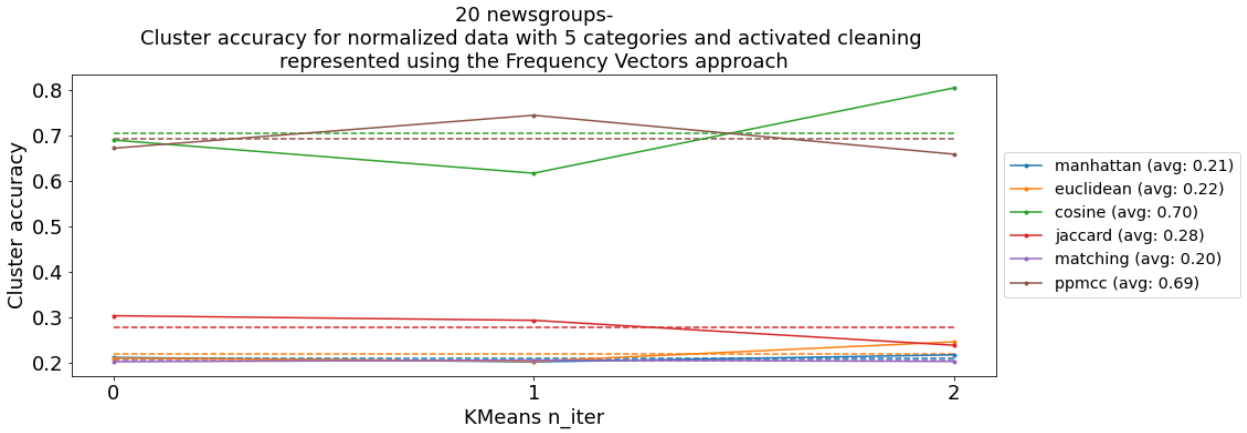


Figure A.16: *20 newsgroups*: Cluster accuracy for pre-processed data using TF-IDF.

A.2.3 k-Means result accuracy for *20 newsgroups* data set using BoW representation and cleaned data

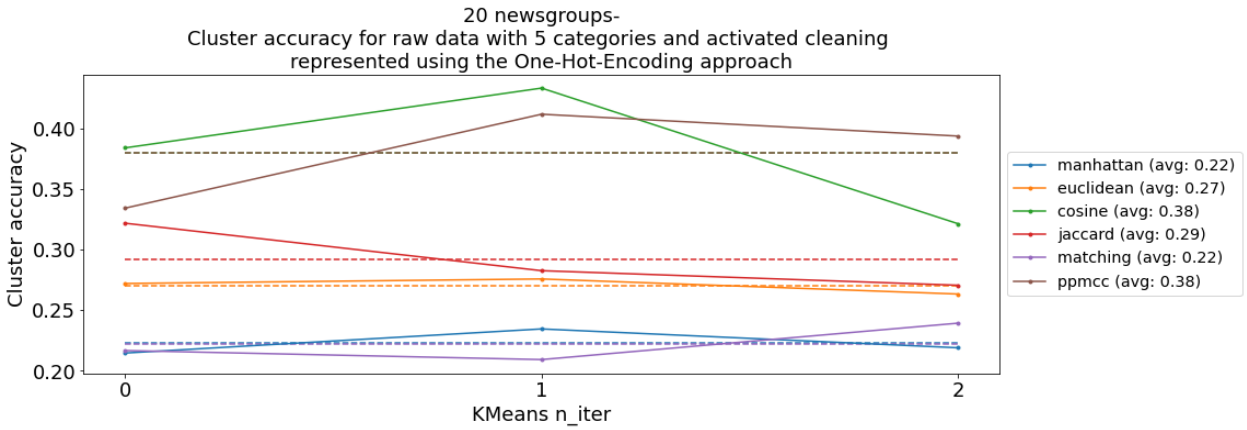


(a) Raw documents

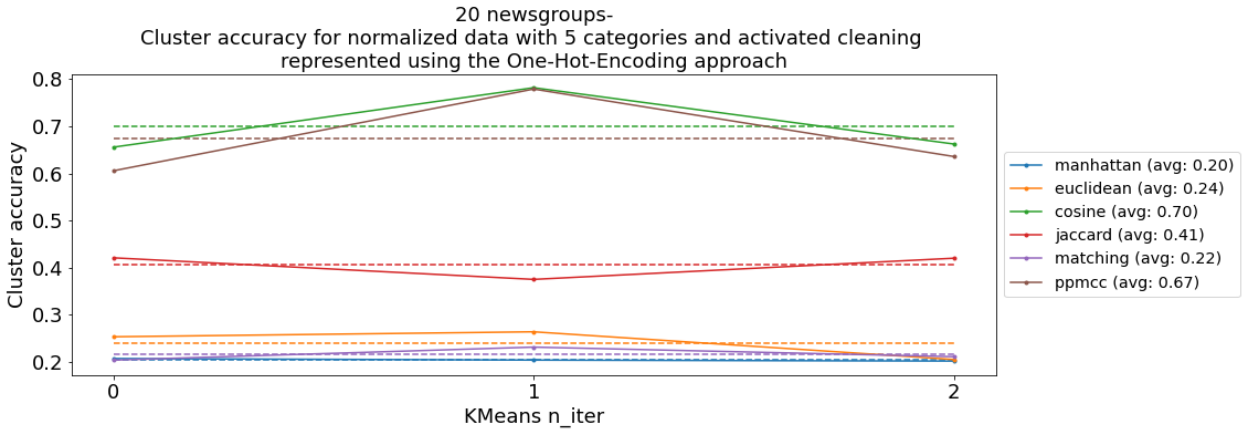


(b) Pre-processed documents

Figure A.17: *20 newsgroups*: Cluster accuracy for cleaned raw and pre-processed data using term-frequency encoding.

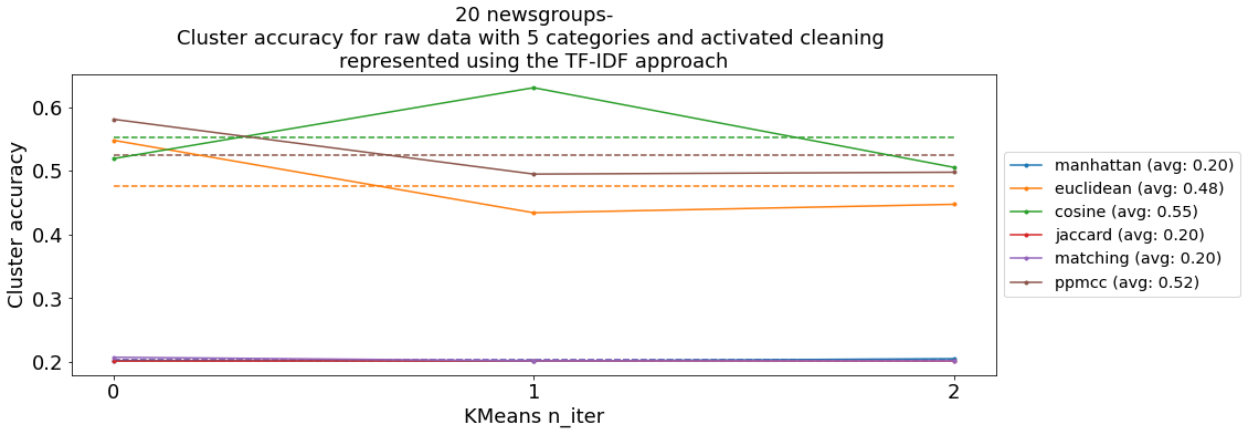


(a) Raw documents

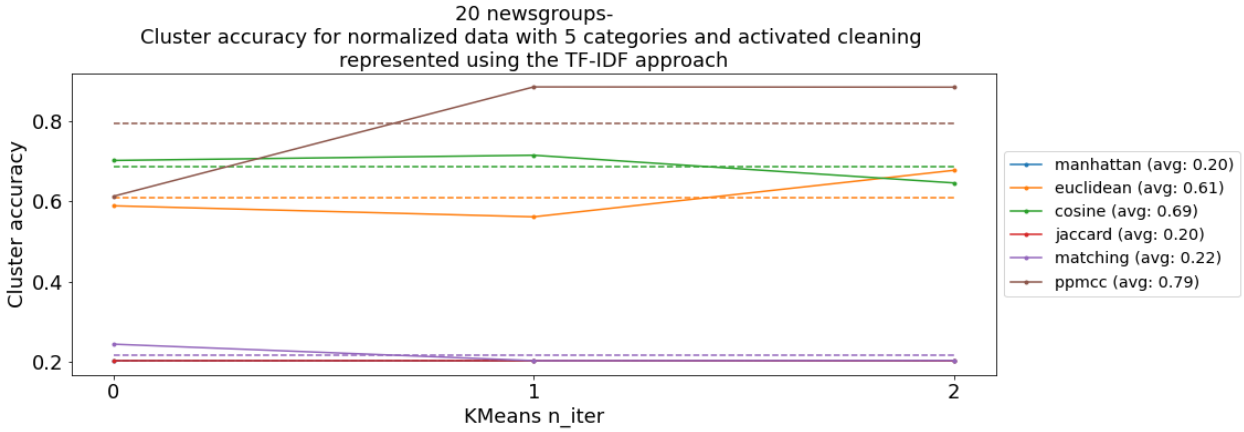


(b) Pre-processed documents

Figure A.18: *20 newsgroups*: Cluster accuracy for cleaned raw and pre-processed data using one-hot encoding.



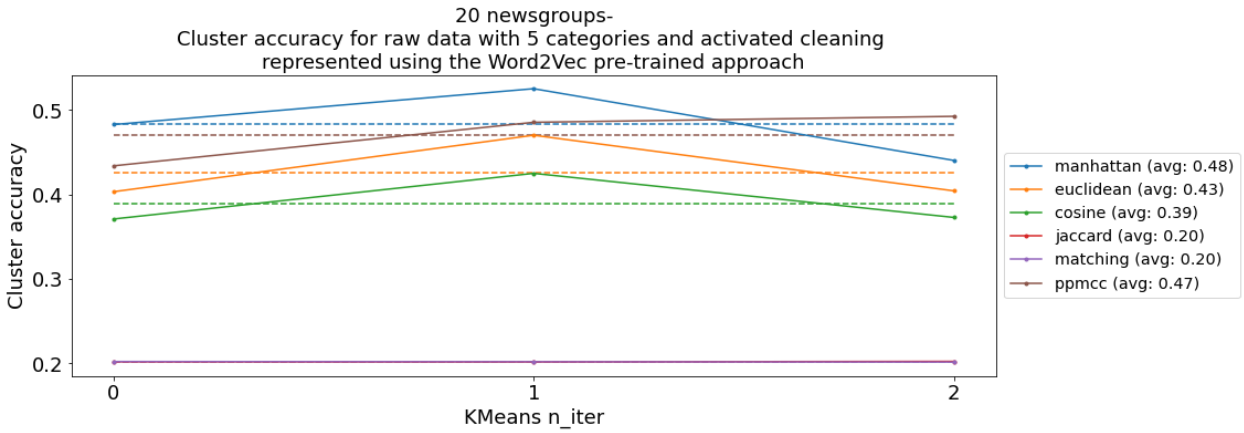
(a) Raw documents



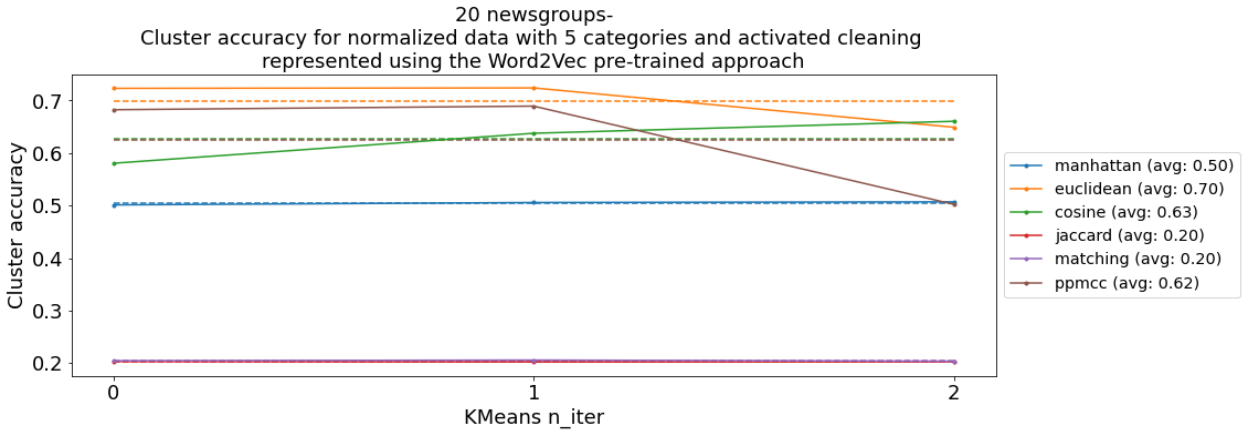
(b) Pre-processed documents

Figure A.19: *20 newsgroups*: Cluster accuracy for cleaned raw and pre-processed data using TF-IDF.

A.2.4 k-Means result accuracy for *20 newsgroups* data set using word embeddings

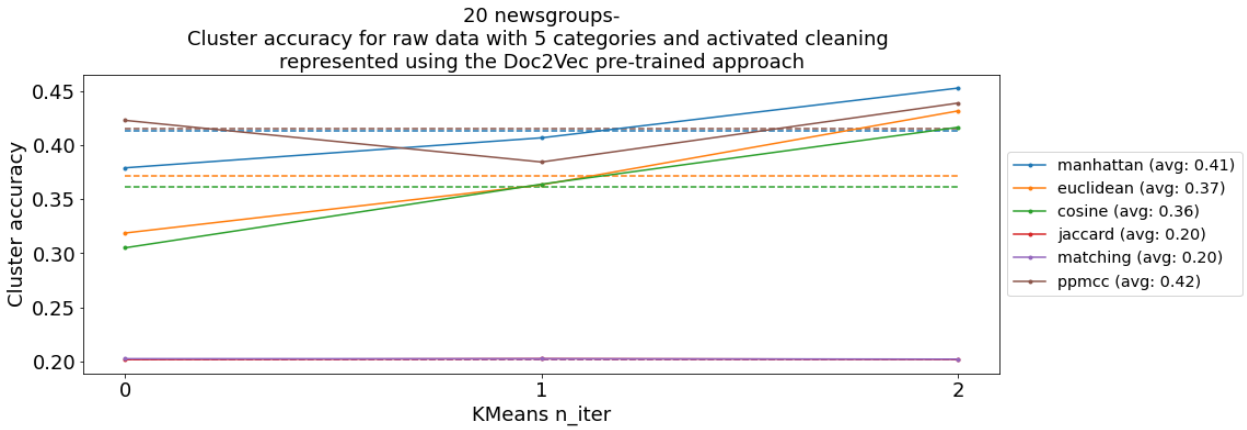


(a) Raw documents

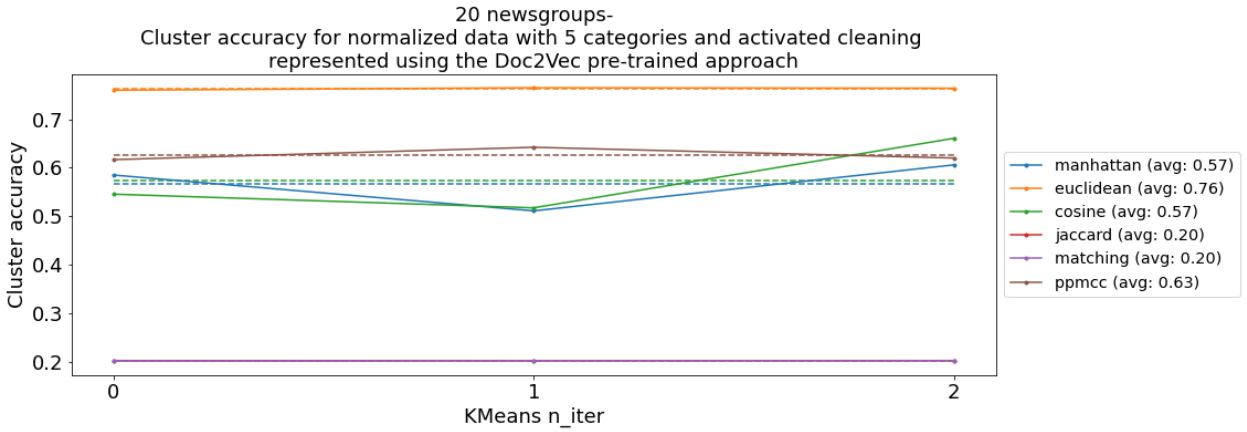


(b) Pre-processed documents

Figure A.20: *20 newsgroups*: Cluster accuracy for raw and pre-processed data using word2vec.

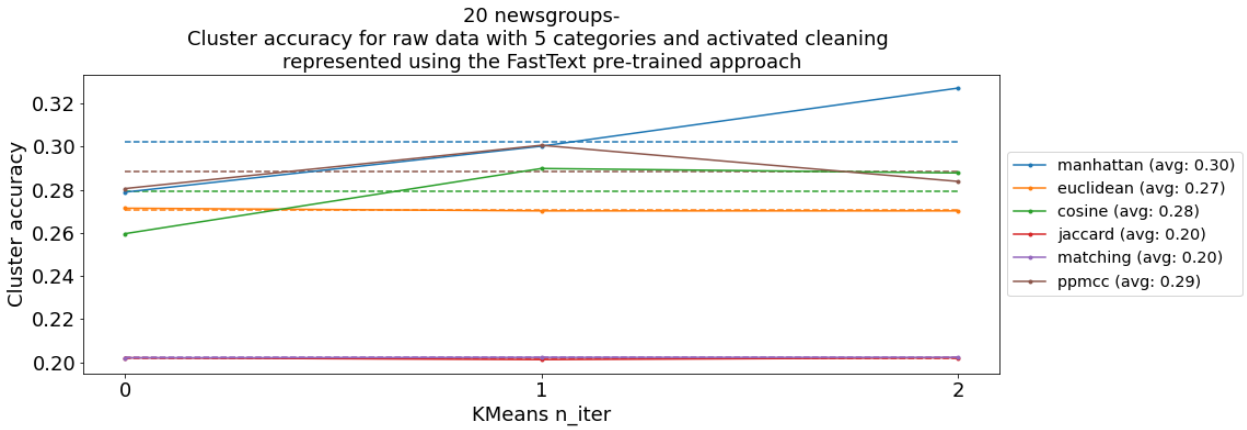


(a) Raw documents

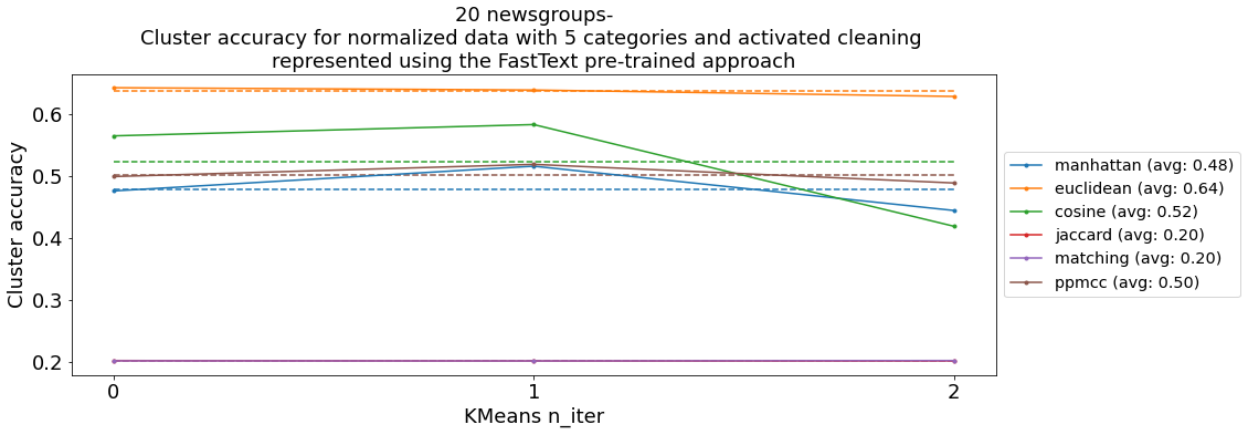


(b) Pre-processed documents

Figure A.21: *20 newsgroups*: Cluster accuracy for raw and pre-processed data using doc2vec.



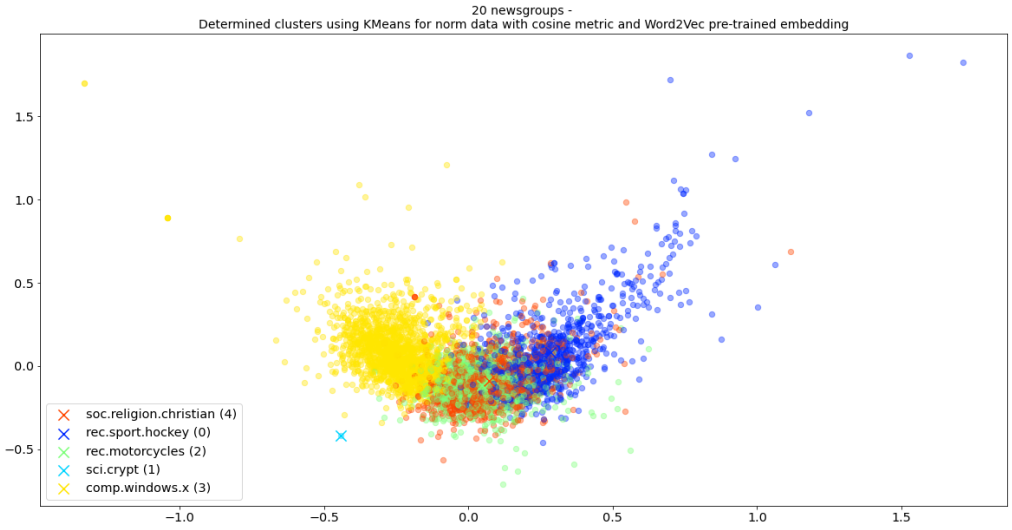
(a) Raw documents



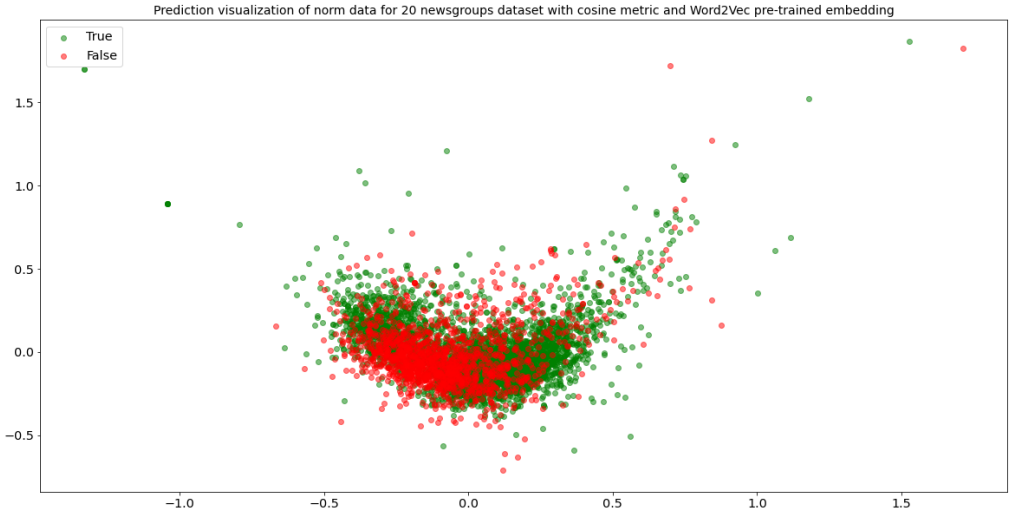
(b) Pre-processed documents

Figure A.22: *20 newsgroups*: Cluster accuracy for raw and pre-processed data using FastText

A.3 k Means clustering result for *20 newsgroups* data set using Word2Vec and Doc2Vec

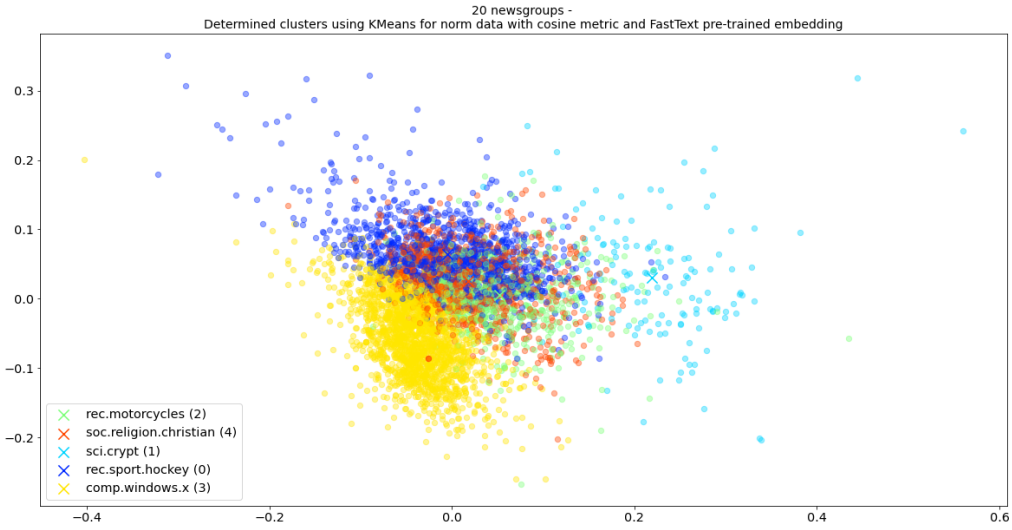


(a) Cluster result

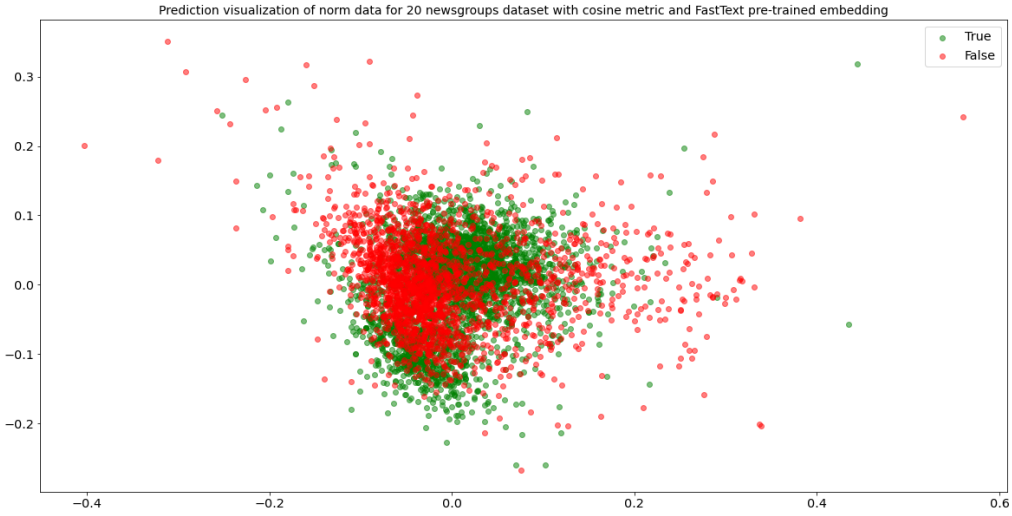


(b) Cluster error

Figure A.23: *20 newsgroups*: Cluster accuracy and error for pre-processed data using Word2Vec



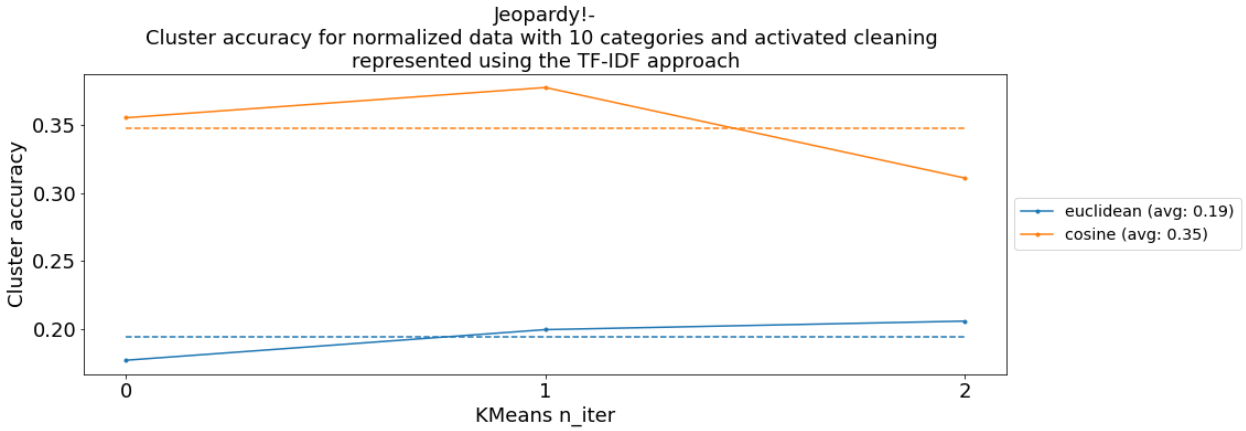
(a) Cluster result



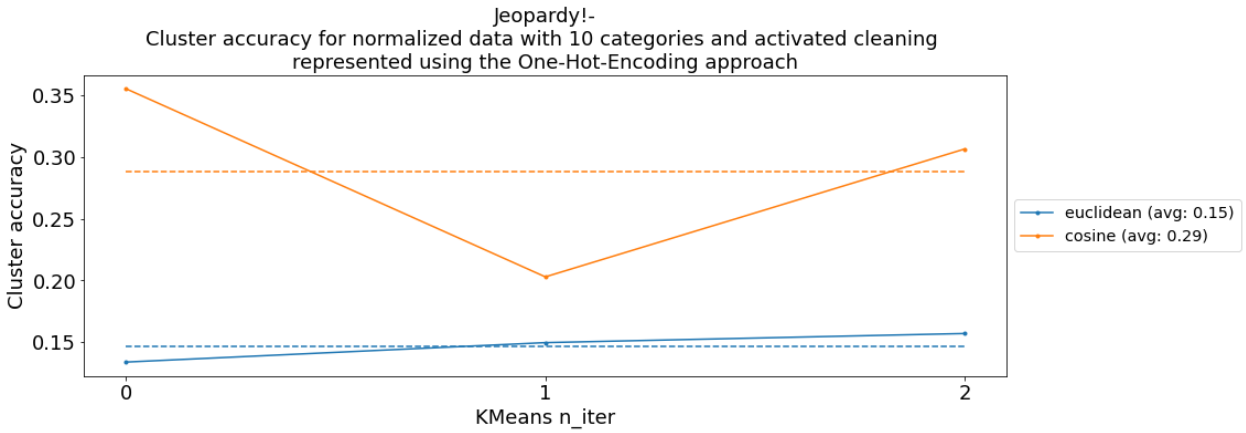
(b) Cluster error

Figure A.24: *20 newsgroups*: Cluster accuracy and error for pre-processed data using FastText

A.4 k Means clustering result for *Jeopardy!* data set using BoW approach

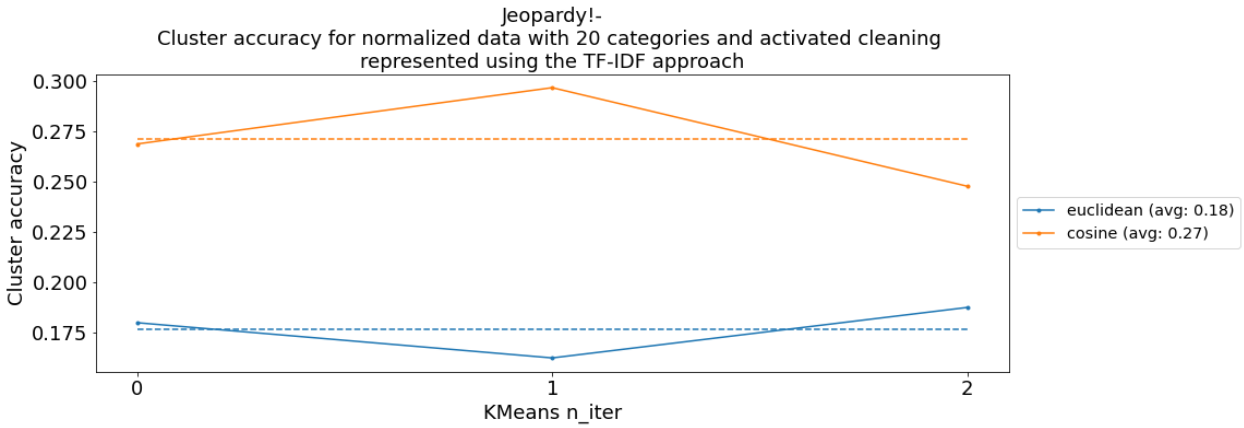


(a) TF-IDF encoding

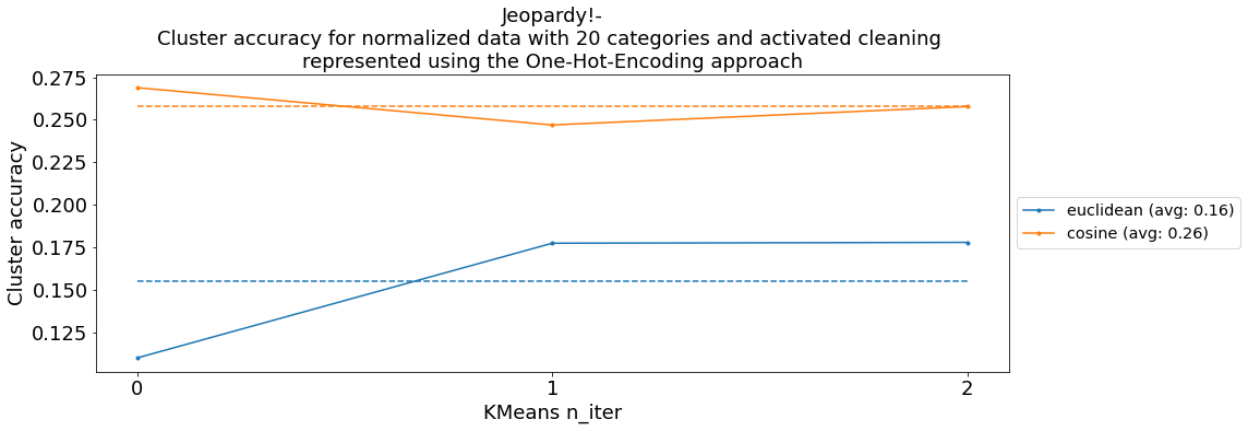


(b) One-hot encoding

Figure A.25: *Jeopardy!*: Cluster accuracy for pre-processed data using the BoW approach (`n_categ=10`).



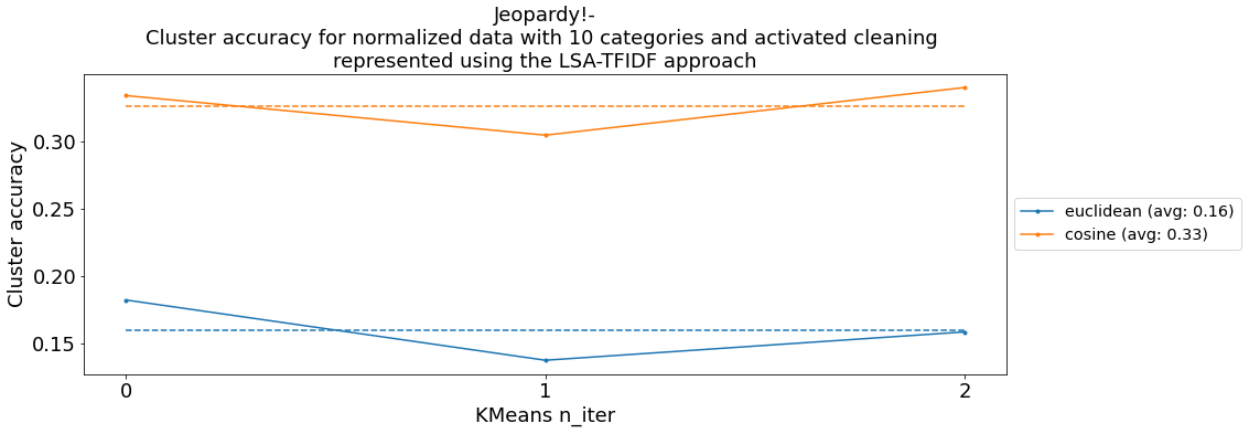
(a) TF-IDF encoding



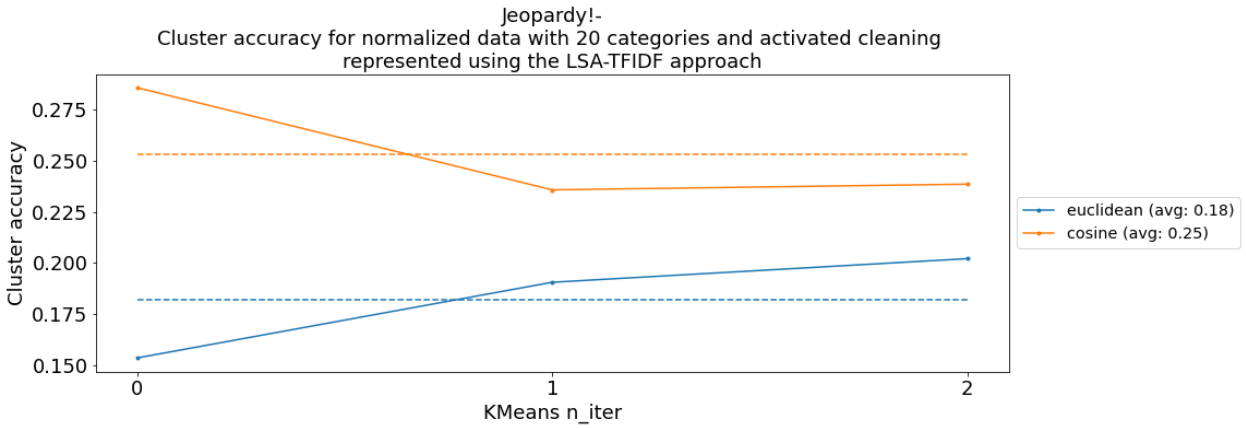
(b) One-Hot encoding

Figure A.26: *Jeopardy!*: Cluster accuracy for pre-processed data using the BoW approach (`n_categ=20`).

A.5 k Means clustering result for *Jeopardy!* data set using LSA



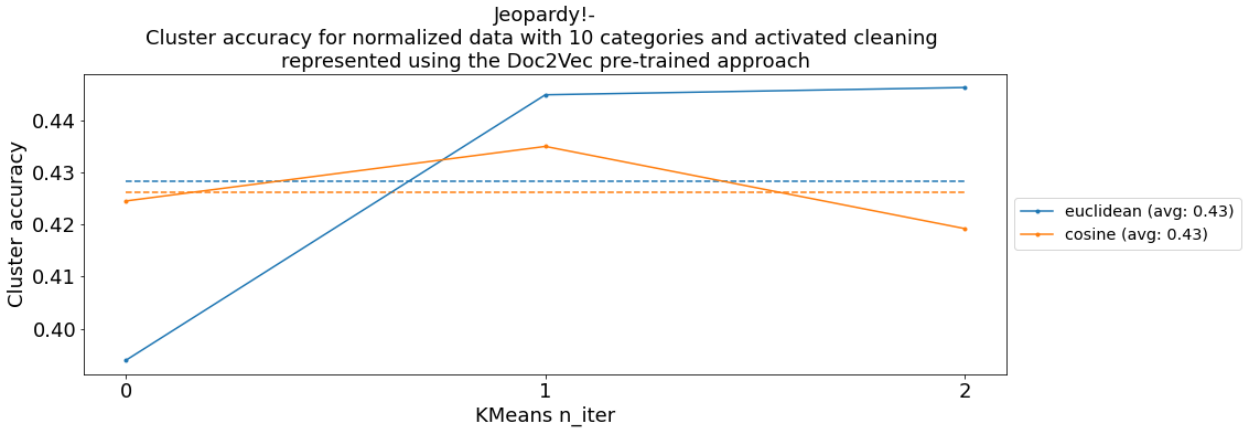
(a) `n_categ=10`



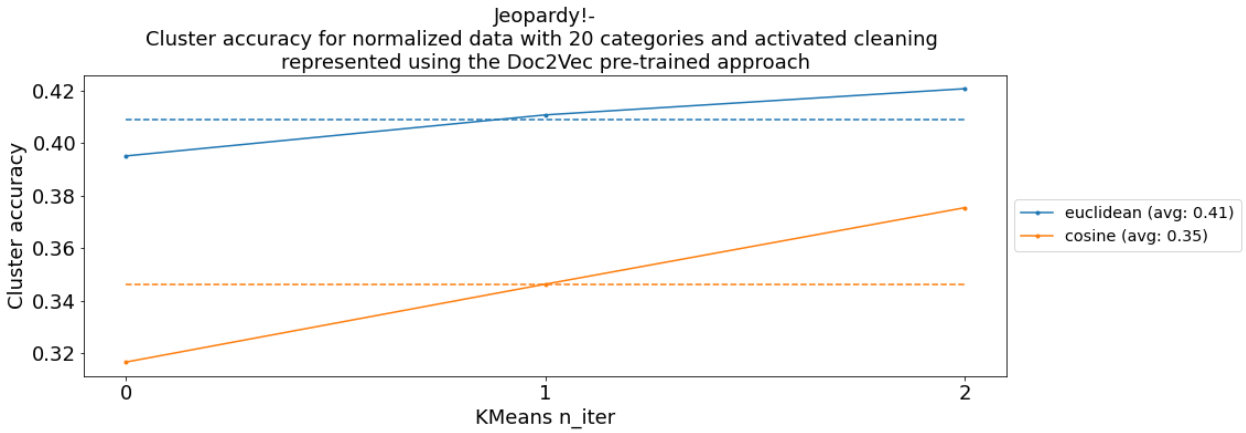
(b) `n_categ=20`

Figure A.27: *Jeopardy!*: Cluster accuracy for pre-processed data using the LSA.

A.6 k Means clustering result for *Jeopardy!* data set using word embeddings

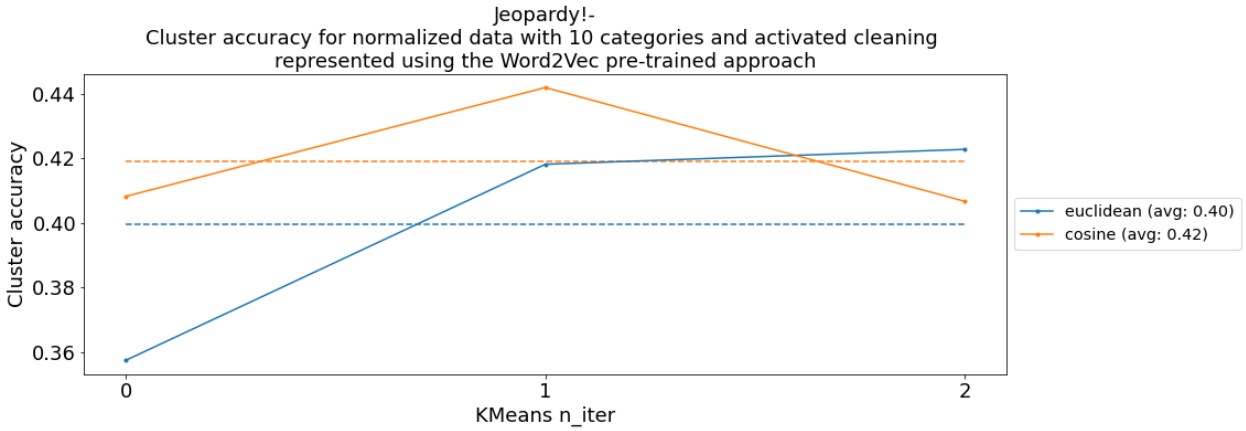


(a) `n_categ=10`

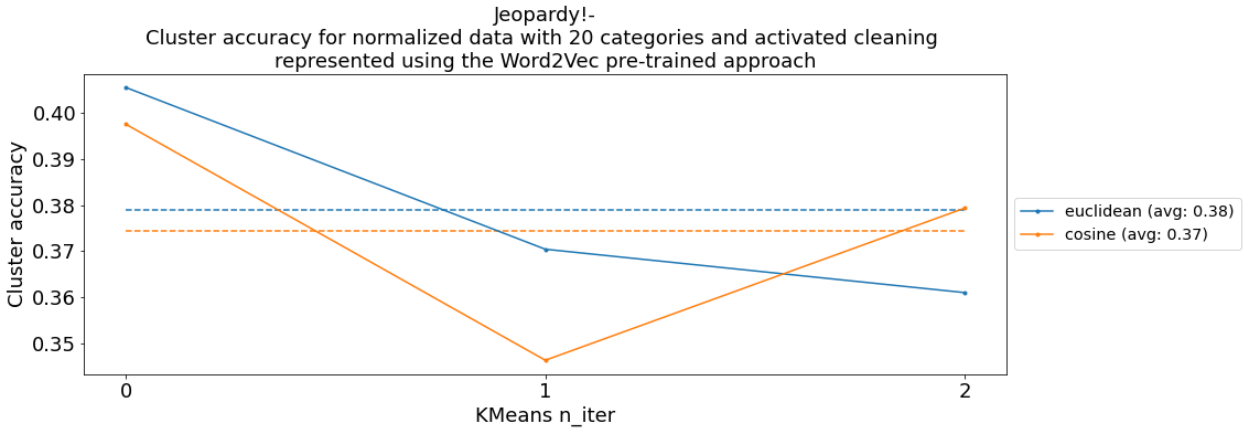


(b) `n_categ=20`

Figure A.28: *Jeopardy!*: Cluster accuracy for pre-processed data using the doc2vec.

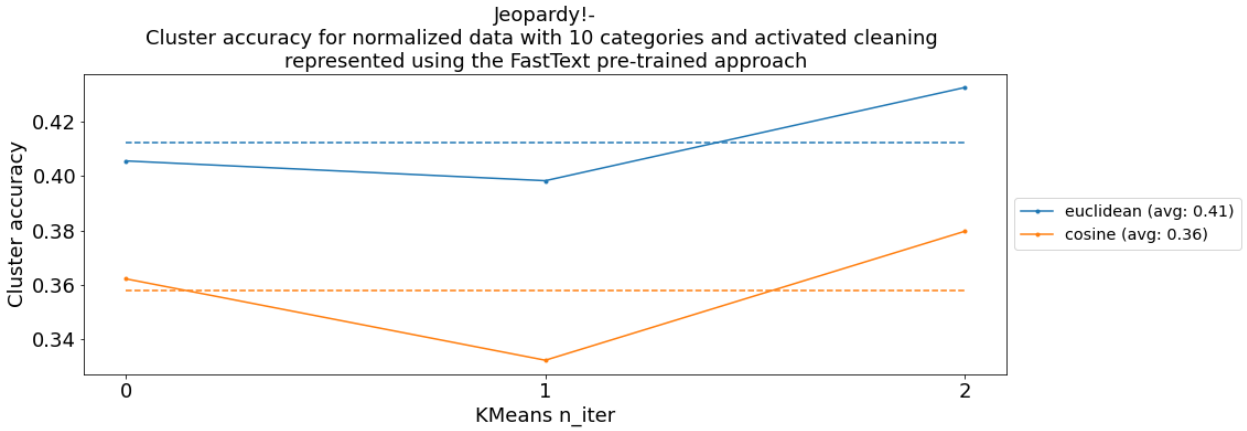


(a) `n_categ=10`

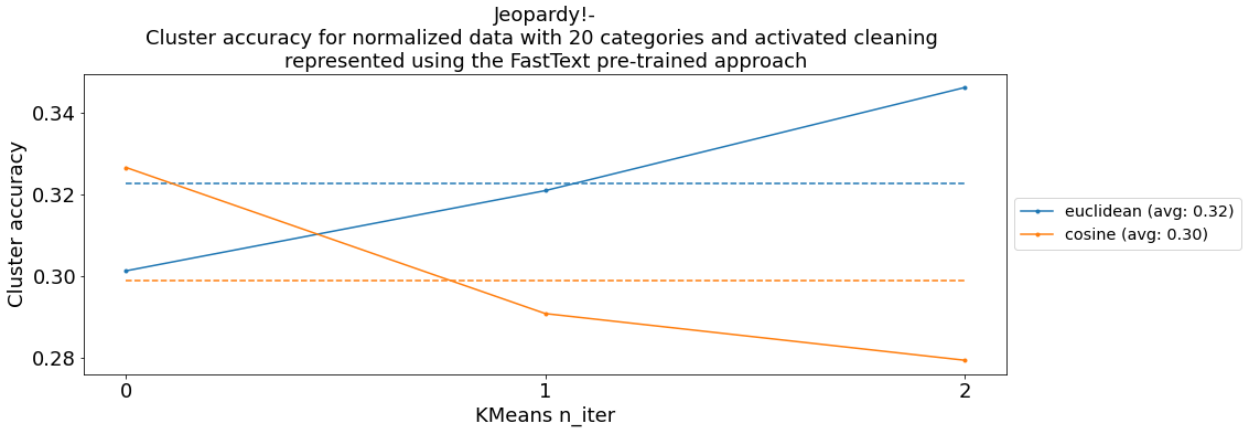


(b) `n_categ=20`

Figure A.29: *Jeopardy!*: Cluster accuracy for pre-processed data using the word2vec.



(a) `n_categ=10`



(b) `n_categ=20`

Figure A.30: *Jeopardy!*: Cluster accuracy for pre-processed data using the fastText.

APPENDIX B

Listings

B.1 Data Cleaning

Listing B.1: Code Snippet - Data Cleaning

```
def cleaning(text):
# Remove new line, new tab
text = re.sub(r"\n", ' ',text)
text = re.sub(r"\t",' ',text)
# Remove URL
text = re.sub(r'''(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()
    <>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s
    `!()\[\]{};:'".,<>?]))''', " ", text)
# Remove HTML
text = re.sub(r'<.*?>', '', text)
# Remove Email
text = re.sub('\S+@[a-zA-Z]+\.[a-zA-Z]+', '', text)
# Remove repeating chars
text = re.sub(r"!+", "! ",text)
text = re.sub(r"\.+", ". ",text)
text = re.sub(r"\?+", "? ",text)
text = re.sub(r"\*+", "* ",text)
text = re.sub(r"\>+", "> ",text)
text = re.sub(r"\<+", "< ",text)
# Clean shorthands
text = re.sub("\'s"," ", text)
text = re.sub("\'ve"," have ", text)
text = re.sub("\'re", " are ", text)
```

```
text = re.sub("\'ll", " will ", text)
text = re.sub("I'm", "I am", text)
text = re.sub("\'d", " would ", text)
text = re.sub("n't", " not ", text)
text = re.sub("can't", "can not", text, flags=re.IGNORECASE)
text = re.sub("i\.e\.", "id est", text, flags=re.IGNORECASE)
text = re.sub("e\.g\.", "for example", text, flags=re.IGNORECASE)
text = re.sub("e-mail", "email", text, flags=re.IGNORECASE)
# Remove comma between numbers
text = re.sub("(?<=[0-9])\,(?=[0-9])'", "", text)
# Special characters
text = re.sub("\$"," dollar ", text)
text = re.sub("\&", " and ", text)
text = re.sub("\%", " percent ", text)
# Remove non ascii character
text_non_ascii = ""
for i in text:
num = ord(i)
if(num>=0):
if(num <=127):
text_non_ascii=text_non_ascii+i
text = text_non_ascii
# Remove smiley faces such as :), :(, :-) and :-(
text = re.sub(r":\)|:\(|:-\(|:-\)",' ',text)

# Remove 's'
text = re.sub(' s ', " ", text)
# Remove extra spaces
text = re.sub("[\s]+", " ", text)
# Strip text
text = text.strip()

return text
```

APPENDIX C

Tables

C.1 Number of Tokens

		Uncleaned Data				*Cleaned Data*			
Data Set	**No. of Components**	**No. of Tokens**	**No. of Unique Tokens**	**No. of Documents**	**No. of Documents with empty Token List**	**No. of Tokens**	**No. of Unique Tokens**	**No. of Documents**	**No. of Documents with empty Token List**
20 newsgroups	5	1,371,218	72,961	4,847	0	1,193,891	68,419	4,847	4
	10	2,367,523	115,980	9,629	0	2,081,794	2,081,794	9,629	17
	20	4,667,159	210,114	18,286	0	4,358,138	193,575	18,286	5
Jeopardy!	5	57,799	12,523	3,869	0	57,943	12,450	3,869	0
	10	107,516	19,717	7,175	0	107,554	19,547	7,175	0
	20	185,097	28,563	12,543	0	184,899	28,206	12,543	0

Table C.1: Number of extracted tokens and corpus length of two different data sets corresponding to the specified number of categories and the cleaning type.

		Uncleaned Data				*Cleaned Data*			
Data	**Number of Components**	**Number of Tokens**	**Number of Unique Tokens**	**Number of Documents**	**Number of Documents with empty Token List**	**Number of Tokens**	**Number of Unique Tokens**	**Number of Documents**	**Number of Documents with empty Token List**
20 newsgroups	5	368,815	40,910	4,847	13	363,863	38,635	4,843	13
	10	658,384	64,401	9,629	29	652,468	60,979	9,624	27
	20	1,308,692	124,561	18,286	72	1,301,053	107,794	18,269	57
Jeopardy!	5	23,446	8,306	3,869	0	23,730	8,282	3,869	1
	10	44,004	13,198	7,175	0	44,549	13,111	7,175	1
	20	76,390	19,150	12,543	0	77,435	18,935	12,543	1
Reddit	5	100,739	14,294						

Table C.2: Number of extracted pre-processed features and corpus length of the three different data sets corresponding to the specified number of categories and the cleaning type.

C.2 Runtime Pre-Processing

		20 newsgroups		*Jeopardy!*			*Reddit*
	No. of Categories	**5**	**10**	**5**	**10**	**20**	**5**
Uncleaned	**Tokenization Time (sec.)**	198,9	349,3	32,0	60,1	117,9	162,1
	Pre-Processing Time (sec.)	2,57	5,36	0,14	0,26	0,61	1,7
Cleaned	**Cleaning Time (sec.)**	3,8	7,3	0,31	0,66	1,32	−−
	Tokenization Time (sec.)	174,7	319,6	34,1	60,1	101,65	168,6
	Pre-Processing Time (sec.)	2,57	5,09	0,15	0,28	0,48	1,9

Table C.3: Runtime Pre-Processing

				Computation Time (sec.)		
Data set	**Cleaning**	**Number of Categories**	**Matrix Dimensions**	**Term-Frequency**	**One-Hot**	**TF-IDF**
20 newsgroups	Uncleaned	5	$4,833 \times 40,910$	2.9	2.9	4.3
		10	$9,600 \times 94,401$	11.3	13.8	14.69
	Cleaned	5	$4,830 \times 38,635$	2.6	2.7	4.2
		10	$9,597 \times 60,979$	7.2	7.1	11.2
Jeopardy!	Uncleaned	5	$3,869 \times 8,306$	0.35	0.34	0.57
		10	$8,175 \times 13,198$	1.25	1.25	1.63
		20	$12,543 \times 19,150$	3.13	3.75	5.34
	Cleaned	5	$3,868 \times 8,282$	0.33	0.32	0.56
		10	$7,174 \times 13,111$	1.11	1.19	1.48
		20	$12,542 \times 18,935$	4.47	3.37	4.2
Reddit	Uncleaned	5	$9,828 \times 25,609$	4.28	5.6	8.2

Table C.4: Document term matrix computation time using Bag-of-words approach.

C.3 LSA

Data set	Cleaning	No. of Categories	Original Matrix Dimension	Reduced Matrix Dimension	Dim. Reduction	Computation Time k_tsvd (sec.)	Computation Time LSA (sec.)
20 news	*non-cleaned*	5	$4,833 \times 40,910$	$4,833 \times 2,247$	38,663	129	27.6
		10	$9,600 \times 64,401$	$9,600 \times 3,875$	60,526	514	89
		20	$4,830 \times 38,635$	$4,830 \times 2,220$	36,415	113	27.5
	cleaned	5	$4,830 \times 38,635$	$4,830 \times 2,220$	36,415	113	27.5
		10	$9,597 \times 60,979$	$9,597 \times 3,814$	57,165	446	89
Jeopardy!	*non-cleaned*	5	$3,869 \times 8,306$	$3,869 \times 1,982$	6,324	12.2	11.8
		10	$7,175 \times 13,198$	$7,175 \times 3,298$	9,900	45.4	38.1
		20	$12,543 \times 19,150$	$12,543 \times 4,989$	14,161	214	112.8
	cleaned	5	$3,868 \times 8,282$	$3,868 \times 1,968$	6,314	11.6	10.6
		10	$7,174 \times 13,111$	$7,174 \times 3,257$	9,854	45.4	34.7
		20	$12.542 \times 18,935$	$12,542 \times 4,891$	14,044	259.7	89.3
Reddit	*non-cleaned*	5	$9,828 \times 25,609$	$9,828 \times 3,333$	22,276	173.9	65.9
	cleaned	5	$9,795 \times 23,563$	$9,795 \times 3,195$	20,368	132.4	49.61

Table C.5: LSA applied to pre-processed data represented with TF-IDF. The matrix dimensions are specified as Documents× Terms.

C.4 Word Embeddings

			Number of Documents			*Matrix Computation Time (sec.)*		
Data Set	**Data Setting**	**Number of Categories**	**Word2Vec**	**Doc2Vec**	**FastText**	**Word2Vec**	**Doc2Vec**	**FastText**
20 newsgroups	Raw	5	4829	4828	4830	6.62	5.22	8.00
		10	9592	9593	9595	12.2	11.4	13.6
	Pre-Processed	5	4818	4822	4819	2.23	1.97	2.27
		10	9570	9576	9573	4.67	3.44	4.39
Jeopardy!	Raw	5	3868	3868	3868	0.5	0.33	0.73
		10	7174	7174	7174	1.2	0.77	0.98
		20	12,537	12,538	12,542	1.71	1.14	2.0
	Pre-Processed	5	3860	3864	3862	0.31	0.24	0.43
		10	7,155	7,161	7,158	0.55	0.57	0.55
		20	12,505	12,511	12,505	1.16	0.64	1.06
Reddit	Raw	5	9,759	9,744	9,781	6.16	4.81	7.30
	Pre-Processed	5	9,681	9,690	9,693	2.16	1.61	2.33

Table C.6: Word embedding matrix dimensions and computation time for cleaned data. Each matrix has the given number of rows and 300 columns.

APPENDIX D

Files

D.1 requirements.txt

All software libraries used in this thesis are listed in D.1.

Listing D.1: requirements.txt

```
numpy
sklearn
pandas
matplotlib
seaborn
wordcloud
spacy
gensim
scipy
tabulate
pyclustering
itertools
joblib
multiprocessing
```

D.2 Linguistic structure analysis

In this thesis the python free, open-source library `spaCy` is used to apply the linguistic concepts explained in chapter 2. After downloading, installing and loading the model a

`Language object` containing linguistic annotations is provided.

```
import spacy
from spacy import displacy
nlp = spacy.load('en_core_web_sm')
sentence = "The quick brown fox jumped over the lazy dog."
```

Processing the example sentence with the `nlp` object returns an object that contains all tokens, their linguistic features and relationships.

```
# Process sentence, i.e. split into words and annotate them
doc = nlp(sentence)
# Tokenization
print ([token.text for token in doc])
```

The computed tokens are:

```
['The', 'quick', 'brown', 'fox', 'jumped',
'over', 'the', 'lazy', 'dog', '.']
```

`spaCy` uses a statistical model to predict the part-of-speech tags. It returns the result as a simple or a detailed part-of-speech-tag.

```
# POS-Tagging
# Print Text, POS, Tag, Dep
# Text - original word text
# POS  - simple pos-tag
# TAG  - detailed pos-tag
# DEP  - Syntactic dependency
for token in doc:
print(token.text, token.pos_, token.tag_, token.dep_)
```

The result for the provided sentence is

```
The DET DT det
quick ADJ JJ amod
brown ADJ JJ amod
fox NOUN NN nsubj
jumped VERB VBD ROOT
over ADP IN prep
the DET DT det
lazy ADJ JJ amod
dog NOUN NN pobj
```

```
10  . PUNCT . punct
```

The part-of-speech tags and the syntactic dependencies can be visualized using the dependency visualizer `dep`.

```
1  displacy.render(doc, style="dep")
```