Institut für Informatik
Lehrstuhl für Programmier- und Modellierungssprachen

LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Multidimensional clustering of Massive Open Online Course offers

Applying unsupervised learning algorithms FCM and SOM
to MOOC textual descriptions

Kai-Henning Wilker

Computer Science Bachelor

Aufgabensteller: François Bry
Betreuer: Yingding Wang
Abgabetermin: 26.12.2016

Ich versichere hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 26.12.2016

. . . . . .

(Unterschrift des Kandidaten)

Abstract

A recent trend in teaching at universities has been the development of Massive Open Online Courses (MOOCs). The textual descriptions of MOOC offers are an interesting new data set, which has not yet been analysed extensively. The aim of this bachelor thesis is to detect clusters within the set of MOOCs based on their textual English descriptions. All the MOOCs within the same cluster shall share a common topic according to their textual description.

While there are many clustering algorithms already available, the main focus of this thesis is to use unsupervised learning algorithms for clustering MOOCs. The two candidates of unsupervised learning algorithms, which are considered in this research thesis, are Fuzzy C-Means and Self-organizing Maps. Both of these algorithms have particular properties that allow them to properly cluster MOOCs, which have more than one topic.

The cluster analysis is divided into the following steps: Firstly, the textual descriptions of MOOCs are represented as numerical "bag of words" vectors and are then transformed into lower dimensional vectors, either using Latent Semantic Indexing or Locality Preserving Indexing. This transformation of input data is demanded by most clustering algorithms. Secondly, the aforementioned two clustering algorithms are implemented and applied to a test data set. Thirdly, the quality of the emerged clusters is evaluated by comparing them to a gold standard, which is achieved through manually clustering the topics.

This thesis shows that the MOOC textual descriptions can be clustered successfully with Self-organizing Maps. In contrast, Fuzzy C-Means did not detect as many useful clusters as Self-organizing Maps and needs some future improvement to work well with MOOC textual descriptions.

The meta information of the automatically generated clusters may be used in a MOOC recommendation system in the near future.

Contents

Chapter 1

Introduction

1.1 Motivation

What are MOOCs?
Massive Open Online Courses (MOOCs) are a relatively new trend in univerity teaching. These courses are online courses, which can be joined by everyone and work exclusively via internet. Therefore, the only restriction for enrolling in a MOOC is having an established internet connection. By this time, various renowned universities and institutions offer MOOCs. Some of them, such as Stanford and Princeton University, are offering MOOCs through *Coursera*, others, such as Massachusetts Institute of Technology and Harvard University, through *edX*. Additionally, important internet companies like Google and Amazon are providing their own MOOCs through *Udacity*.

MOOCs usually contain video lectures, regular practical assignements and forums or chat rooms, in which the students can communicate with each other. Many MOOCs conclude their class with a larger project, which the students have to carry out on their own. While the topics of available MOOCs range from arts or music to business or science, there is a majority of them with a focus on computer science. Thus, the use of MOOCs can be especially helpful for computer science students in their studies.

Problems of finding the right MOOC
Unfortunately, many students feel overwhelmed with finding the right course for their needs within the massive amount of available MOOCs and MOOC platforms. This bachelor thesis is part of the project *IROM* conducted by Yingding Wang, which aims to solve this problem by developing an "intelligent recommendation system" for MOOC offers.

For this recommendation system, a classical full-text search through all MOOC descriptions might not be a sufficent approach. For example, one student may want to learn programming in Java. With the help of a classical search, the student could easily find a course called "Introduction into Programming with Java". However, there may exist an even better and more helpful MOOC titled "Object-oriented Programming 101". Although the latter course may be the better choice, a standard full-text search engine may not place this one as high as the first one, since the title differs from the search term.

Another problem the recommendation system has to face is that beginners sometimes

do not know the correct terms to describe their topic of interest. For example, a student may want to learn how to create websites for mobile devices and knows that the languages she must learn are HTML and CSS. Therefore, she can effortlessly find the course "Intro to HTML & CSS". Nevertheless, she probably does not know that websites, which can be viewed on a desktop just as well as on smartphones, are referred to as "responsive" websites. In consequence she will have a hard time finding the course "Responsive Web Design Fundamentals", a MOOC that actually would be perfect for her needs.

1.2 Objective

The **objective of this thesis** is to detect clusters in the textual descriptions of MOOC offers. *Clusters* are groupings of objects, in which objects of the same cluster are similar to each other, while objects of different clusters are dissimilar. What these objects are and how similarity between them is defined, depends on the data, which one wants to analyze.

In this project, the clusters are groups of MOOCs, whose textual descriptions are semantically similar. This means that MOOCs within the same cluster shall have similar topics. Afterwards, these clusters can be used for the recommendation system described above. Then, the aforementioned problems of finding the right MOOC can be solved by using these clusters.

Clustering algorithms
There are many different algorithms for cluster analysis already available. This project focuses on utilizing the the two clustering algorithms *Fuzzy C-Means*[1] and *Self-organizing Maps*[2]. These two algorithms were chosen, because both have properties that make them especially apt for clustering MOOCs.

Firstly, the classical clustering algorithms usually result in a "strict clustering", in which each data point belongs to exactly one cluster. In contrast to that, Fuzzy C-Means creates a "fuzzy clustering", where data points are able to belong to more than one cluster. As MOOCs can address more than one topic (e.g. introductory courses that outline a larger field without specializing into one topic), a fuzzy clustering may be more appropriate than strict clustering.

Secondly, Self-organizing Maps – although creating a strict clustering – yields a neighbourhood relation between the clusters. In this relation, neighbouring *clusters* are similiar to each other. When used for clustering MOOCs, this algorithm can thereby show, whether topics of MOOCs are related. This could be an interesting feature for a MOOC recommendation system.

Data source
Since the majority of courses are conducted in English, only MOOCs with textual descriptions written in English are considered for the clustering. Nevertheless, clustering MOOCs with multiple languages could certainly be an interesting future project, albeit this is out of the scope of this thesis.

[1] Source: [BEF84].
[2] Source: [Koh90].

The textual descriptions of all available MOOCs need to be gathered automatically by a crawler. Due to the large number of different MOOC platforms, implementing this crawler is an extensive task. Therefore, a different thesis within the *IROM* project is responsible for this implementation. Using the results of that crawler, a test data set was created as reference. This data set is the basis of all clustering done within this research project.[3]

1.3 Working Approach

As explained in the introduction, the objective of this project is to cluster MOOC textual descriptions. The overall strategy to obtain the clusters from the textual descriptions is as follows:

Firstly, the texts need to be represented as numerical vectors, in order to be usable for the clustering algorithms. For this, the *Vector Space Model*[4] was used, wherein each term present in the document corpus is represented by one dimension of the vector. The main obstacle in this process is the large number of dimensions of the resulting vectors. To reduce the number of dimensions, the algorithms *Latent Semantic Indexing* (LSI)[5] and *Locality Preserving Indexing* (LPI)[6] were applied. This first step is the topic of chapter 3.

Secondly, the clustering algorithms *Fuzzy C-Means* (FCM) and *Self-organizing Maps* (SOM) were utilized to detect the clusters. These algorithms are profoundly affected by different, a priori defined meta-parameters, like the number of desired clusters. Besides implementing the algorithms, the central task in cluster analysis is to determine these meta-parameters. This is done by running the algorithms with various values for these meta-parameters. Then, the quality of each clustering is measured by calculating different evaluation indices. Comparing those indices, the optimal meta-parameters for the specific data set can be found. This second step is the topic of chapter 4 and chapter 5.

In both steps, one must choose between two alternative algorithms, culminating in four distinct ways to obtain the clusters. These approaches depend on the choice of dimensionality reduction method (LSI or LPI) and clustering algorithm (FCM or SOM). The **third part** of this thesis is to evaluate the quality of the clusterings and compare the results of those four approaches. For that, human experts clustered the reference data set manually, creating an "ideal" clustering of the given data. The quality of the clusterings, which were found automatically, was then assessed by comparing them to the ideal clustering. The results of this evaluation can be found in chapter 5.

[3] This reference data set can be found in the source code repository that is attached to this bachelor thesis.

[4] Source. [MRS08, p. 120].

[5] Source: [DDF+90].

[6] Source: [CHH05].

1.4 Related Work

Cluster analysis is a large area in which research has been done for decades. This has resulted in many different approaches to clustering. A fine introduction to the field can be found in [HK06, chapter 7], which summarizes many classical clustering algorithms. Applying clustering methods to large collections of documents is an important part of Information Retrieval; [MRS08, chapters 16 and 17] contains an introduction to this topic. Besides, Self-organizing Maps is a type of artifical neural network. Neural networks themself are a vast research area, as well; the book [Hay99] introduces this topic comprehensively and contains a full chapter, that is dedicated to Self-organizing Maps.

The Fuzzy C-Means clustering algorithm was developed by Bezdek et al. in the early 1980s and first published in [BEF84]. Around the same time, Kohonen established the Self-organizing Maps algorithm; a summary of this algorithm can be found in [Koh90] and a subsequent monography on this topic was written in [Koh01]. Both clustering algorithms and their applications have been extensively studied. One exemplary paper, which provides a comparison of both algorithms on a large number of different data sets, is [ML06].

To convert the MOOC textual descriptions into vectors, the vector space model as described in [SB88] was used. Additionally, two document representations based on this model were applied: Firstly, the classical Latent Semantic Indexing, which was originally published in [DDF$^+$90], and secondly, the more recent Locality Preserving Indexing, as developed in [HCLM04] and [CHH05].

A further important topic of this thesis is evaluating the quality of the detected clusters. This evaluation was mainly based on the studies found in [HBV01], [WZ07] and [AGM$^+$13].

Since the MOOC concept gained popularity just in the last years, there has been no comprehensive cluster analysis of MOOC textual descriptions. In this thesis, well-researched algorithms are applied to these novel data. This bachelor thesis is (to the best knowledge of the author) the first project that cluster-analyses MOOC textual descriptions via Fuzzy C-Means and Self-organizing Maps.

Chapter 2

System Design

2.1 The MOOC clustering application

In order to realize the cluster analysis of MOOC textual descriptions, a MOOC clustering application was implemented. In this chapter, the clustering application's functionality and the concepts behind it's implementation are introduced.

The MOOC clustering application accepts the data, that were collected by the crawler as input. These data consist of information about the gathered MOOCs: the title of the course, it's URL, it's publisher, and most important it's description. The textual descriptions are then used to cluster the MOOCs. The results of this procedure are saved in a database, and can then be accessed via a REST interface. In the future, this interface will be used by the MOOC recommendation system, in order to access the clusters. This process is illustrated in figure 2.1.

The software is written in the programming language Python 2. One of the advantages of Python is, that it provides the fast and easy to use mathematics libraries NumPy and SciPy. Besides, the NoSQL database system MongoDB was used to save the created vectors and clusterings.

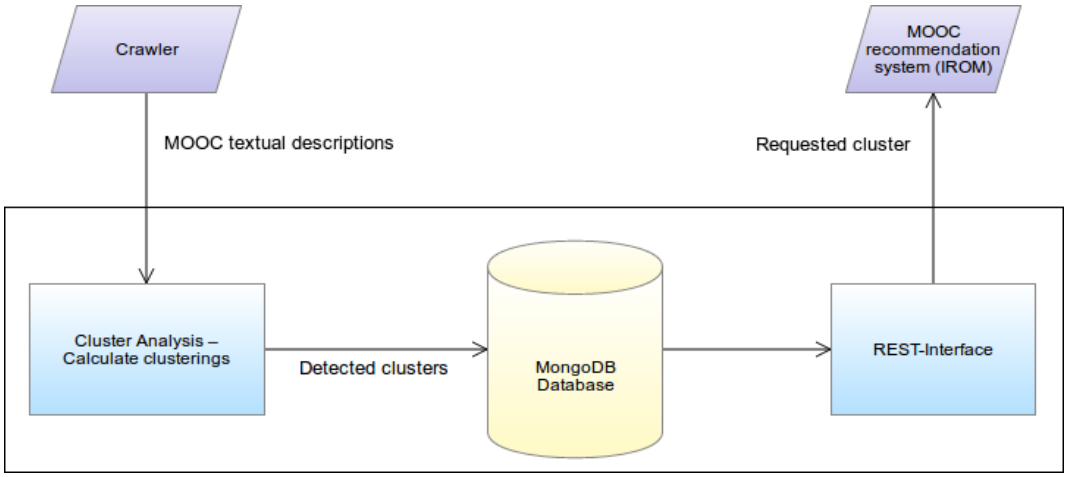

Figure 2.1: *Information flow of the MOOC clustering application.*

2.2 Cluster analysis flowchart

The main component of the clustering application is the code that creates the clusterings. Figure 2.2 contains a flowchart, which summarizes this cluster analysis code.

At first, the MOOC textual descriptions need to be transformed into vectors. This is done by using the TF-IDF model, which is described in section 3.1.2. Afterwards, the vectors are reduced by either the LSI or the LPI algorithm. The implementation details of these two algorithms can be found in sections 3.2.3 respectively 3.3.3. Next, the actual clustering is done, using either FCM or SOM; see sections 4.1.4 and 4.2.4 for more information on the implementation of these algorithms. After the clusters were built successfully, for each clustering special evaluation indices are calculated. These indices can help to determine the quality of the clustering; this topic is discussed in section 5.1. In the end, all results are saved in the database, so that they can be retrieved later on.

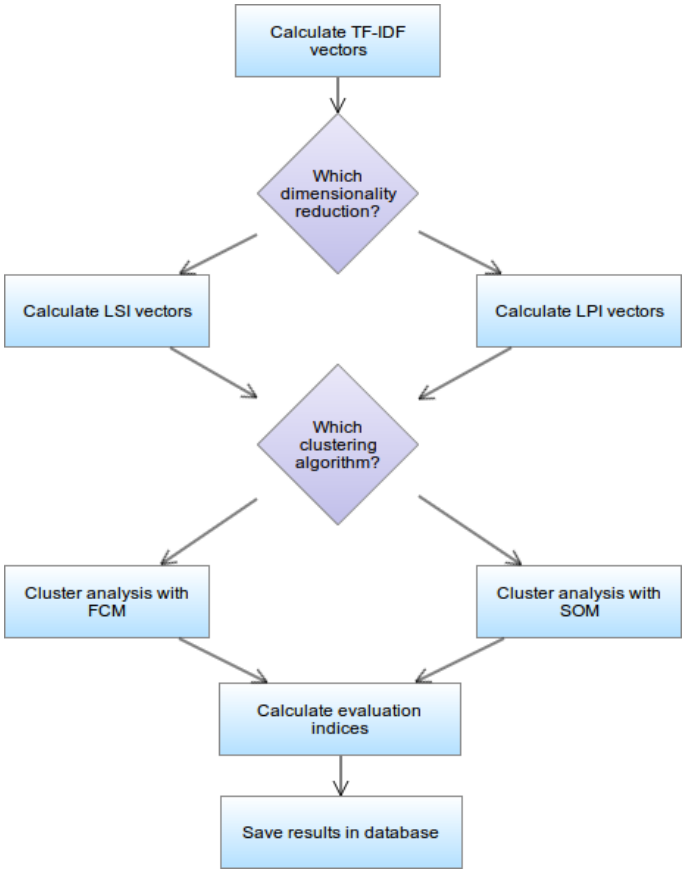

Figure 2.2: *Flowchart of the cluster analysis code.*

2.3 Modules overview

The Python code was designed after the *model-view-controller* pattern[1], as illustrated in figure 2.3. In the following, the individual parts of this pattern will be presented.

[1]Source: [LR01].

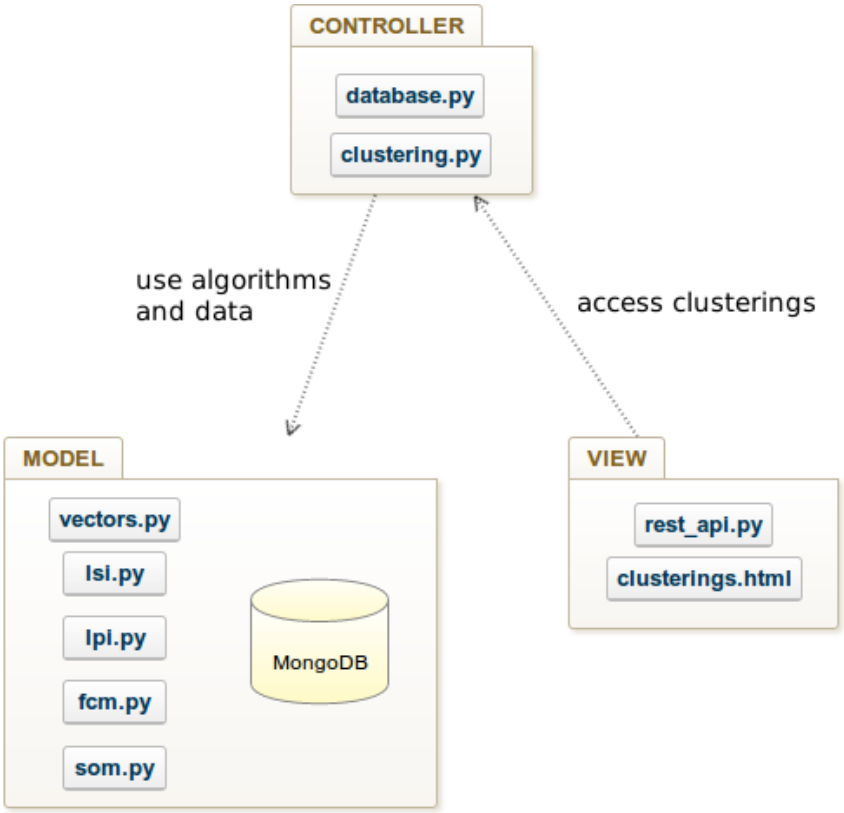

Figure 2.3: *Overview of implemented Python modules.*

Model

This component contains the main logic and data modules. The clustering algorithms are implemented in the modules `fcm.py` and `som.py`. Respectively, the algorithms for reducing the dimension of the vectors are written in `lsi.py` and `lpi.py`. Additionally, the basic functions for creating the TF-IDF vectors and to save the vectors in the database can be found in `vectors.py`.

All the data (i.e. the vectors and the various clusterings) are saved in a MongoDB database. It's layout will be presented in the subsequent section.

View

This component handles the access to the data by external users. Other programs like the MOOC recommendation system are able to view the clusters via a REST interface. The server that runs this interface is implemented in `rest_api.py`. Furthermore, human users can examine the clusterings by using the GUI that is implemented in `clusterings.html`. This GUI is a graphical frontend for the REST interface and was written in HTML and JavaScript.

Controller

This component deals with the overall creation of the clusterings, using the modules `clustering.py` and `database.py`. The controller organizes the different cluster analyis steps by calling every algorithm of the model in the right order and with the correct input. Moreover, it ensures that all (interim-)results are saved properly in the database.

2.4 Database design

As database system, the NoSQL database MongoDB was used. In contrast to relational databases, this database system is *document-oriented*: the contents are saved as documents, which are encoded in BSON format.[2] These documents are then grouped into different collections.

Corpus and dictionary
The main collection in the database of this project is the `corpus` collection. This collection includes all information about the clustered MOOCs, including their textual descriptions and their vectors. An exemplary document of this collection looks like this:

```
{
    'id' : 123,
    'title' : 'Inferential and Predictive Statistics for Business',
    'text' : 'This course provides an analytical framework ...',
    'data_clean' : 'cours provid analyt framework ...',
    'url' : 'https://www.coursera.org/learn/business-statistics',
    'from' : 'Coursera',
    'vec_tf_idf' : { ... },
    'vec_lsi_20' : { ... },
    ...
}
```

A corpus document consists two versions of the textual description: the original description from the website saved under the key `text`, and the preprocessed description, that contains only keywords (see section 3.1.1 for details), saved under the key `data_clean`.

Additionally, one should note that the vectors are not saved as arrays, but as key-value-objects. These objects have the form {`dimension : non-zero-value, ...`}. Thus, only the dimensions of the vector that have a non-zero value are saved in the object; all the remaining dimensions implicitly have the value zero. In this way, the vectors can be stored needing less disk space, and the documents can be retrieved faster.

Besides the `corpus` collection, there is the `dictionary` collection. When the textual descriptions are transformed into TF-IDF vectors, every dimension of one vector corresponds with exactly one word (details can be found in section 3.1.2). This correspondence is saved in the `dictionary` collection: each document is one *(word, dimension)* pair.

Clusterings
For each clustering result, an individual `clustering` collection is created. In these collections, one document represents one cluster. Then, one such document has the list of all the MOOCs, which are included in the respective cluster.

Furthermore, there is the `clusterings_meta` collection, which contains additional information about the clusterings. For each clustering, one document is kept in this collection. This document indicates, which clustering algorithm and which vectors were used in this specific cluster analysis. Moreover, the values of the various meta-parameters of the clustering algorithm are listed in this document.

[2] BSON is a binary-encoded data format similar to the JSON format. MongoDB uses BSON as main format for data representation. Source: [dt16].

Organizing the collections
The catalog of available MOOCs changes over time and so does the corpus of MOOC textual descriptions. That is why every collection (except `clusterings_meta`) is marked with a timestamp in it's title. This timestamp represents the point in time, when the collection was created. Thus, everytime the crawler gatheres a new list of MOOCs, new `corpus` and `dictionary` collections are created. The following shows an exemplary excerpt of collections:

```
1473361442_corpus
1473370221_dictionary
1473875019_clustering_FCM
1473878416_clustering_FCM
1473879070_clustering_SOM
...
1475053439_corpus
1475053581_dictionary
1475055777_clustering_SOM
...
clusterings_meta
```

This design has the advantage, that *all* clusterings are stored, even the outdated ones. It may be interesting to analyze in the future, how the clusterings of MOOCs will have changed over time.

Chapter 3

Vector Representations of MOOC Textual Descriptions

All clustering algorithms usually need real-valued vectors as input data. Thus, in order to cluster text documents one initially must find an appropriate way to convert the texts into vectors. This representation of documents as vectors is called the *Vector Space Model*, which is a fundamental concept in the field of Information Retrieval.[1]

This chapter at first explains how to represent documents as vectors on the basis of the "bag of words" assumption. Afterwards, two algorithms for reducing the dimensionality of the vectors, which were used in this project, are presented.

3.1 "Bag of words" and vector representations

Most approaches to encoding documents as vectors are based on the assumption that texts can be represented as sets of words. This means that a document is solely described by it's vocabulary, without using any further information about the structure or the language of the text. This assumption is called the "bag of words"-assumption. It may not be obvious from a linguistical point of view, why this idea works well. Despite that, methods based on this assumption have been applied successfully over the last years.[2]

The rough idea for a "bag of words" vector representation is to match each word present in the document corpus with a dimension in the vector space. The vectors are then created by putting a "score" of each word (for example the number of occurences of this word in the document) into the vector's corresponding dimension. One of the main advantages of this idea is that no further information about language, grammar or word order are needed, making it very easy to use.

[1] Source: [MRS08, p. 120].

[2] One of the first experimental evidence for this claim can be found in [SB88].

3.1.1 Preprocessing: from words to keywords

Before calculating the vectors, the documents should be preprocessed. Because the crawler used in this project already returned the MOOC descriptions preprocessed, only a very short summary of the preprocessing steps will be given.

Firstly, there are words which are contained in virtually every text and therefore have no significance in representing the document's content. Such words are called "stop words". In English words like "and", "the", "to" etc. are considered stop words. These words can just be filtered out of the documents beforehand.[3] Specifically in the case of MOOC textual descriptions, there are additional organizational terms or names that might deviate from the actual course topic. One example is the name "University of Illinois" – all courses conducted by this university placed their university's name on the MOOC description. As a result, MOOCs of this university were more likely to be clustered together, regardless of what the course is actually about.

This example shows that broad stop word lists made for general use do not suffice, but must be extended by stop words specific of the document corpus. Moreover, the stop word list needs to be specific to the subsequent application of the clusters – some words, which distract from one clustering goal, might be vital for a different clustering goal.

Secondly, words take different shapes within a document. The word *car* for example can also occur as *Car*, *cars*, *car's* or *cars'*. All those occurence should refer to the same word; therefore all occurences of that kind are replaced by one "word stem". This process is called "stemming".[4] After the preprocessing, the documents consist of "keywords" (or "terms") rendered by the stemming.

3.1.2 Scoring: from Boolean to TF-IDF vectors

After the preprocessing, the documents can be turned into n-dimensional vectors, where n is the number of different keywords in the whole document collection. In the following, denote the terms as $t_1, \cdots, t_n$ and the documents as $d_1, \cdots, d_m$ with m the number of documents. As outlined above, each dimension i in the vector space corresponds to the term t_i. For the document d_j the respective vector is:

$$v_j = \begin{pmatrix} score(d_j, t_0) \\ score(d_j, t_1) \\ \vdots \\ score(d_j, t_n) \end{pmatrix} \in \mathbb{R}^n$$

The different vector space models vary in how they define the function $score(d_j, t_i)$.

The simplest model is the *Boolean model*:[5] the score of a term is unity, if the term occurs in the document, and otherwise zero:

$$score_{\text{boolean}}(d_j, t_i) := \begin{cases} 1 & \text{if } t_i \in d_j \\ 0 & \text{otherwise} \end{cases}$$

[3] Source: [MRS08, section 2.2.2].
[4] Source: [MRS08, section 2.2.4].
[5] Source: [KA08, section 5.2.1].

The *raw term frequency model* (RTF)[6] also takes into account, how often one term occurs in the document: $score_{\text{RTF}}(d_j, t_i)$ is number of occurences of the term t_i in the document d_j. The RTF model has the drawback that longer documents automatically have higher scores compared to short documents. Subsequently, the *term frequency model* (TF) was developed, which normalizes the vector:

$$score_{\text{TF}}(d_j, t_i) := \frac{score_{\text{RTF}}(d_j, t_i)}{length(d_j)}$$

where $length(d_j)$ denotes the number of terms present in the document d_j.

Obviously, some words are more important in defining the topic of a document than others. The *term frequency – inverse document frequency model* (TF-IDF) tries to take this into account: it claims that terms, which only exist in a small number of documents, are more distinctive than terms that occur in many documents. There are a couple of different definitons of the score function for the TF-IDF model; the definition in [MRS08, section 6.2] is:

$$score_{\text{TF\_IDF}}(d_j, t_i) := score_{\text{TF}}(d_j, t_i) \cdot \log \frac{m}{df(t_j)}$$

where m is the number of documents and $df(t_j)$ is the number of documents which contain the term t_j.

Because the TF-IDF model contains more information than the other vector space models mentioned, it was mainly used within this project, disregarding the other models.[7]

Why dimensionality reduction?
All these different models work within $\mathbb{R}^n$, where n is the number of terms. Even for relatively small corpora, n tends to be very large. For example, the collection of all *Coursera* MOOC descriptions has about 20,000 terms. Clustering such high-dimensional data turns out to be very problematic: as documents usually only consist of a minority of all the terms in the corpus, the corresponding vectors are sparse (i.e. contain mostly zeros) and only few dimensions are actually relevant. As a result, the distance measure between two vectors becomes futile and the clustering results are arbitrary. Additionally, one also has to cope with performance issues. This difficulty has been called the "curse of dimensionality".[8]

To solve this issue, several techniques for reducing the dimensionalty of the vectors were developed. The gist of these methods is to find a subspace with much smaller dimension that the original vectors can be projected into, while preserving as much information about the original space as possible. In this project, the classic *Latent Semantic Indexing* as well as the more recent *Locality Preserving Indexing* methods were used, which will be covered in the following.

[6] Source: [Bry13, chapter 1].

[7] Indeed, early experiments on small test corpora showed that using TF-IDF vectors yields better clustering results, compared to using the other models.

[8] Source: [HK06, pp. 400, 401].

3.2 Latent Semantic Indexing

The document representation method Latent Semantic Indexing (LSI) was first published in [DDF+90]. Roughly, it's aim is to find group of words of the same "topic"; instead of representing each term by one dimension, with LSI now each topic is represented by one dimension. The number of desired topics can be chosen freely. Thus, the dimensionality of the vector space can be reduced drastically. The LSI algorithm finds these topics using solely the "bag of words" vectors as presented above, i.e. it makes the thematic structure, which is "latent" in the original space, more visible.

The following is structured as follows: at first, the intuition behind LSI will be explained. Afterwards, the actual algorithm will be summarized.

3.2.1 Intuition

Intuitively, LSI is based on Principal Component Analysis (PCA).[9] Given a set of data points, PCA is a statistical method to find the directions, which approximate the data points as good as possible. These directions are called principal components.

The algorithm works iteratively: at first, one considers all straight lines that pass through the mean of the data points and calculates the Euclidian distance between each data point and the line. The sum of these distances is the "reconstruction error". Then one has to choose the line that has the minimal reconstruction error; this is also the line where the data points, when projected onto the line, have the largest spread. By construction, this line is the "best" one-dimensional linear representation of the original space ("best" in the sense that, if one projects all data points onto the line and then tries to reconstruct the original data points from the projected points, the reconstruction has the smallest possible deviation). The vector that represents the found line is the first principal component.

An illustration of this idea is given in figures 3.1 and 3.2: the first figure shows an example data set along with the direction of the first principal component. The second figure shows that the points projected onto the direction of the first principal component are spread much more than the points projected onto the x axis.

After this first step, a new straight line is calculated in the same way with the restriction, that the new line has to be orthogonal to the previous line. This step is repeated until there are as many lines as dimensions of the original space.

In this iterative process, each new direction is less important for a reconstruction then the directions found before. Therefore, in order to find a k-dimensional approximation of the original data points, one can just project the data points onto the space spanned by the first k principal components.

The key idea of LSI is to utilize this process for document vectors (usually TF or TF-IDF vectors). Each principal component then represents one topic.

[9] A more in-depth explanation of PCA can be found in [Wis13], which this section is based on.

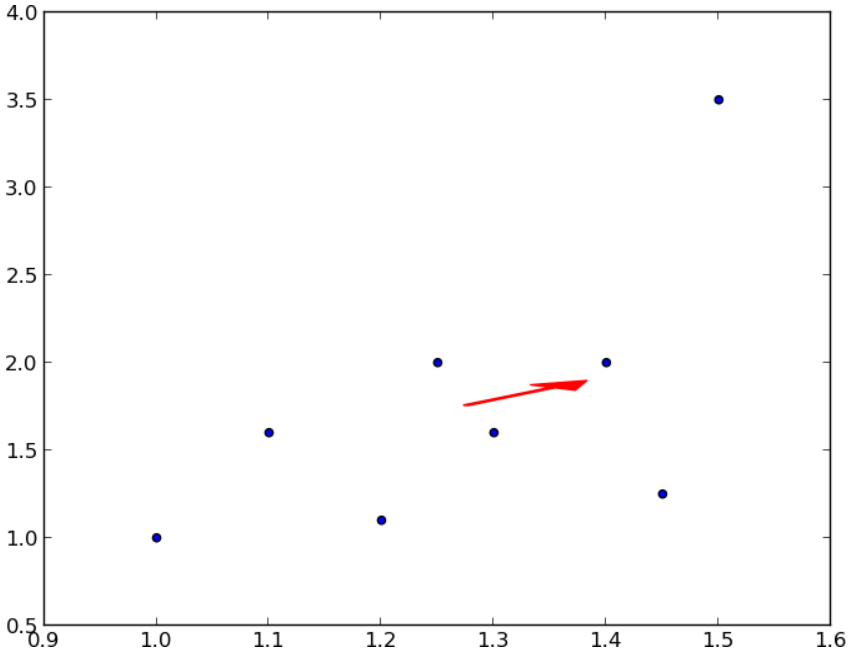

Figure 3.1: *A small two-dimensional example data set. The red arrow represents the first principal component found by PCA.*

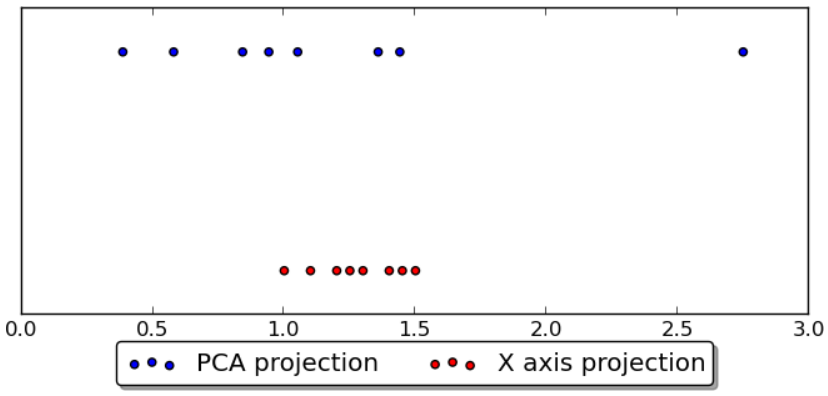

Figure 3.2: *The example data points projected onto the direction of the first principal component compared to the projection onto the x axis.*

3.2.2 Algorithm

The PCA algorithm described above is geometrically intuitive, but is relatively slow, especially, if the number of dimensions is higher than the number of input vectors (which is usually the case with document vectors). For that reason, the LSI algorithm uses Singular Value Decomposition (SVD), a linear algebra technique which can find the principal components faster.[10]

The Singular Value Decomposition theorem[11] states, that for every matrix $C \in \mathbb{R}^{m \times n}$ with rank r, there is a decomposition

$$C = U \Sigma V^T$$

with the following properties:

- $U \in \mathbb{R}^{m \times m}$ is a matrix with orthogonal columns,
- $V^T \in \mathbb{R}^{n \times n}$ is a matrix with orthogonal rows,
- $\Sigma \in \mathbb{R}^{m \times n}$ is diagonal in the sense that the only non-zero values of Σ are $\Sigma_{i,i}$ for $1 \leq i \leq r$.

The diagonal elements $\Sigma_{i,i}$ of Σ are called "singular values". If the singular values are non-negative and ordered descendingly, the decomposition is unique. Many mathematical software libraries (for example NumPy, which was used in this project) provide the calculation of SVD. Details on how to compute a matrix's SVD can be found in [TBI97, lecture 31].

To compute the principal components of a set of vectors employing SVD, the input vectors $x_1, \cdots, x_n$ have to be combined into one matrix $C = [x_1, \cdots, x_n]$. In the decomposition $C = U \Sigma V^T$, the columns of V are exactly the principal components.

In the SVD, $\Sigma \in \mathbb{R}^{m \times n}$ can be simplified to $\Sigma_r \in \mathbb{R}^{r \times r}$, as all other entries outside of this submatrix are zero anyway. Considering the matrix product $U \Sigma V^T$, the $(m - r)$ rightmost columns of U are going to be multiplied with zeros, as do the bottom $(n - r)$ rows of V^T. Thus, these columns respectively rows are redundant and can therefore be omitted: $U \in \mathbb{R}^{m \times m}$ can be represented by $U_r \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{m \times m}$ can be represented by $V_r \in \mathbb{R}^{r \times n}$, yielding the *truncated* SVD $C = U_r \Sigma_r V_r^T$.[12]

LSI uses the truncated SVD to find the desired small-dimensional space. The LSI algorithm is as follows:[13]

Input Let $x_1, \cdots, x_n \in \mathbb{R}^m$ be the "bag of words" vectors of all documents.
Let $k \in \mathbb{N}$ be the number of topics, i.e. the dimensionality of the requested subspace.

Output The algorithm returns $y_1, \cdots, y_n \in \mathbb{R}^k$, the projections of the original vectors into the subspace.

1. SVD Let $X = [x_1, \cdots, x_n] \in \mathbb{R}^{m \times n}$ be the document matrix. Compute the truncated SVD $X = U_r \Sigma_r V_r^T$ where r is the rank of X.

[10] Source: [Wis13, section 2.19].
[11] Source: [MRS08, section 18.2].
[12] Source: [MRS08, pp. 408, 409].
[13] Source: [MRS08, sections 18.3, 18.4].

2. **Truncation** The requested number of dimensions is k. Therefore, all singular values in Σ_r except the first k ones are replaced by zero, resulting in $\Sigma_k \in \mathbb{R}^{k \times k}$. This is parallel to keeping only the first k principal components in PCA. Analogously, more rightmost columns in U_r and more bottom rows in V_r^T can be removed, producing $U_k \in \mathbb{R}^{m \times k}$ and $V_k^T \in \mathbb{R}^{k \times n}$.
3. **Subspace projection** The original vectors can now be projected into a k-dimensional space by using the matrix U_k: the projected vectors are $y_i := U_k^T x_i$.

3.2.3 Implementation

Since the NumPy library offers an easily accessible implementation of SVD, programming the LSI algorithm in Python within this project was very straight-forward.

The open-source library Gensim, a framework for calculating document representations, contains an "incremental" version of the LSI algorithm.[14] This beares the advantage that only parts of the document vector matrix have to be directly in memory, while for standard SVD the whole matrix must be present in memory. Altough in this project the matrix's size hardly hit the RAM limit, using Gensim could be favorable in future projects with larger corpora. A downside of Gensim is, that it only works with TF-IDF vectors as input, making Gensim less flexible then our manual implementation.

3.3 Locality Preserving Indexing

Locality Preserving Indexing (LPI) is a document representation technique published by Cai, He et al. in [HCLM04] and [CHH05] which is based on the dimensionalty reduction method Locality Preserving Projections (LPP) [HN04]. It was developed as an alternative to LSI. LPI is based on the assumption that semantically similar documents lie near each other in the term-document space. The following is divided into two sections: firstly, an intuitive explanation of the main idea behind LPI and secondly, a summary of the algorithm that was implemented in this project.

3.3.1 Intuition

The *unsupervised* learning algorithm LPI can be seen as an approximation to the *supervised* learning algorithm Linear Discriminant Analysis (LDA).[15] Both algorithms aim to find a subspace with small dimension in which the high-dimensional vectors can be projected into. Ideally, these projections do not lose any important information necessary for later clustering of the data. While LPI as an unsupervised algorithm only works on the vectors without any additional information, LDA furthermore needs an a priori defined classification of the vectors, thus making LDA a supervised algorithm.

LDA seeks to find a projection in which vectors of the same class are near each other and vectors of different classes are far away from each other. For that, the mean vectors of each class normalised by the spread of the respective class are examined. By solving an eigen-vector problem, the projection that scatters these mean vectors best possible

[14] Source: [ŘS10, section 2.1].
[15] Source: [HCLM04, section 1].

is found. The projection provided by LDA optimally preserves the discriminatory information of the given classes.[16]

The main idea of LPI is to replace LDA's a priori defined classes by newly created "neighbourhoood-classes". These classes contain the local information of each data point by putting the vectors' nearest neighbours in the same class. With the new classes, the LDA algorithm can be used in order to find the subspace projection. Thereby, the vectors that were close to each other in the original space are also close to each other in projected space, i.e. preserving the locality of the original space.

3.3.2 Algorithm

Different versions of the LPI algorithm are published ([HCLM04], [CHH05]). In this project, the latest version (as described in [CHH05, section 3.2]) was implemented. The algorithm is as follows:

Input Let $k \in \mathbb{N}$ be the dimensionality of the desired subspace.
Let $p \in \mathbb{N}$ be the number of nearest neighbours considered in constructing the "neighbourhood-classes".
Let $x_1, \cdots, x_n \in \mathbb{R}^m$ be the documents vectors; the vectors have to be normalised ($||x_i|| = 1$).[17]

Output The algorithm returns the subspace projections $y_1, \cdots, y_n \in \mathbb{R}^k$ of the original vectors.

The algorithm consists of the following steps:

1. **Build the neighbourhood graph** Define G as an undirected graph with n vertices, where each node v_i represents the vector x_i. Two nodes v_i and v_j are connected if and only if x_j is one of the p nearest neighbours of x_i (or vice versa). Then define the graph's adjacency matrix $S \in \mathbb{R}^{n \times n}$ in the following way:

$$S_{i,j} := \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$

Additionally, let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix where each diagonal element is the sum of one row in S, i.e.

$$D_{i,i} := \sum_{j=1}^{n} S_{i,j}$$

Besides, let $L := D - S$.

2. **Singular Value Decomposition** First, set

$$x_{\text{mean}} := \frac{1}{\sum_{i=1}^{n} D_{i,i}} \left(\sum_{i=1}^{n} D_{i,i}\, x_i \right) \in \mathbb{R}^m$$

[16] Source [CHH05, section 4.2].
[17] Therefore, raw-term-frequency vectors can't be used in this algorithm.

and remove this constant from every document vector: $\hat{x}_i := x_i - x_{\text{mean}}$. Then define $\hat{X} := [\hat{x}_1, \cdots, \hat{x}_n] \in \mathbb{R}^{m \times n}$ and compute the Singular Value Decomposition of $\hat{X}$:

$$\hat{X} = U \Sigma V^T,$$

where $U \in \mathbb{R}^{m \times r}$, $\Sigma \in \mathbb{R}^{r \times r}$ and $V \in \mathbb{R}^{r \times n}$ with $r = rank(\hat{X})$.[18]

This creates the projection matrix $W_{SVD} := U$. Projecting the document vectors into the SVD subspace is done by $\tilde{x}_i := W_{SVD}^T \hat{x}_i \in \mathbb{R}^r$. Unlike in Latent Semantic Indexing, no non-zero singular values are cancelled out; that is why the new subspace has r dimensions.

3. **Locality Preserving Projection** Let $\tilde{X} = [\tilde{x}_1, \cdots, \tilde{x}_n] \in \mathbb{R}^{r \times n}$ be the preprocessed term-document matrix from above. In order to find the Locality Preserving Projection, one has to solve the following generalized eigenvalue problem:[19]

$$\tilde{X} L \tilde{X}^T a = \lambda \tilde{X} D \tilde{X}^T a$$

Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_l$ be the found eigenvalues, ascendingly sorted; let $a_1, \cdots, a_l$ be solutions of the equation above corresponding to each eigenvalue.

The projection space should have the dimensionality k. Therefore, only the first k eigenvalues $\lambda_1, \cdots, \lambda_k$ and their solutions $a_1, \cdots, a_k$ are considered in defining the projection matrix: set $W_{LPI} := [a_1, \cdots, a_k] \in \mathbb{R}^{n \times k}$.

Now, all ingredients for the subspace projection are in place: with $W := W_{SVD} W_{LPI}$ the new k-dimensional vectors are $y_i := W^T \hat{x}_i$.

3.3.3 Implementation

Within the scope of this project, a Python implementation of the algorithm described above was created. For the mathematical operations, especially the SVD projection, the implementation makes use of the open-source libraries NumPy and SciPy.

Finding the p nearest neighbours was implemented as a brute-force search; this suffices due to the comparatively small size of the MOOC test corpus. In order to use the algorithm for much larger corpora, one may have to optimize this search for performance reasons.

Determining suitable values for the size p of the neighbourhood classes and for the dimensionality k of the requested projection space depends on the shape of the data one wants to cluster. In [CHH05, section 5], the optimal values for p ranged from 6 to 15. In the course of the evaluation, the optimal value for p when clustering MOOC textual descriptions turned out to be $p = 14$. The evaluation of the optimal dimensionality k will be discussed in section 5.1.4.

Improving the computation time
For choosing the best LPI subspace approximation later on, one needs many different

[18] Here it is necessary to use the truncated SVD (as described above). Since this method removes all zero-valued singular values, this ensures that $\tilde{X}$ in step 3 is of full rank and the eigenvalue problem in step 3 is solvable. Details and proof of this fact can be found in [CHH05, section 3.3].

[19] This is a more general version of the eigenvalue problem that is considered in LDA. Details can be found in [He05, section 2.5].

subspace candidates readily computed. Calculating all these different subspace approximations with the algorithm above is quite time-consuming. Therefore, a novel *cached* LPI algorithm was implemented in this project as well: in the LPI algorithm presented above, the first two steps are only dependent on the input vectors, but are independent of the dimensionality $k \in \mathbb{N}$ of the new subspace. That is why the results of steps 1. and 2. can be cached. Then, these results can be used again, when one computes a different dimensional approximation of the same input vectors. Since the first two steps take more than half of the computation time of the whole LPI algorithm, the cached LPI algorithm improves the overall computation time significantly.

Graph weighting scheme
In the neighbourhood graph, the weights of each edge are set to a constant 1. These weights could be replaced by more refined weightings, for example the dot-product of the two vectors. But the experimental results provided in [CHH05, section 5.5.2] show, that the later clustering is insensitive to the choice of the weights. This is why the simple 0-1-weighting scheme was used. However, the implementation also works with any other weighting scheme.

LPI as part of Gensim
The LPI algorithm described above is *not* incremental, because the full matrix must be stored in RAM in order to compute the Locality Preserving Projection. Therefore, this algorithm can not be part of the Gensim framework, since Gensim is designed to be independent of any corpus size.[20] To the knowledge of the author, no incremental version of the LPI algorithm has yet been developed. However, it would be an interesting further research project to find such an incremental algorithm and to implement it as part of the Gensim framework.

[20] Source: [ŘS10, section 2].

Chapter 4

Clustering Algorithms

The main goal of this bachelor thesis is to find groups of MOOCs on the same (or similar) topic by using only the textual descriptions of the MOOCs as data, without any additional information about the groups. This process of grouping objects into classes of similar objects is called cluster analysis or clustering. The groups found in this process are then called clusters. Ideally, the objects within one cluster are similar to each other, while objects of different clusters are dissimilar.[1]

Out of the numerous algorithms for cluster analysis that were developed in the last decades, this project focuses on two clustering algorithms: Fuzzy C-Means and Self-organizing Maps. This chapter presents these two algorithms as well as discusses the challenges that occured while applying these algorithms to MOOC textual descriptions.

4.1 Fuzzy C-Means

The first algorithm considered in this thesis is *Fuzzy C-Means* (FCM). The algorithm was developed by Bezdek et al. in the 1980s. They published their algorithm in [BEF84], which is the source for this section. FCM is a *fuzzy*, *prototype-based* clustering algorithm. Before going into the details of the FCM algorithm, these preliminary ideas are explained intuitively.

4.1.1 Intuition behind Fuzzy C-Means

4.1.1.1 Intuition of prototype-based clustering

The basic idea behind prototype-based clustering is that each cluster of vectors is represented by only one vector, called the prototype or center of the cluster. The clusters are then found by only manipulating the prototypes and afterwards assigning each object to one prototype. One classical prototype-based clustering algorithm is K-Means. FCM can be considered a derivative of K-Means, because they share the fundamental algorithmic structure. The K-Means algorithm is as follows:[2]

[1] Source: [HK06, section 7.1].
[2] Source: [HK06, p. 403].

Input $K \in \mathbb{N}$ the number of clusters.
$x_1, \cdots, x_n$ the input vectors.

Output A set of K clusters.

1. **Initialization** Initialize randomly K prototype vectors.
2. **Repeat until the clusterings don't change:** Assign each vector x_i to the nearest prototype vector. This assignment creates a new clustering. Then, each cluster prototype is defined as the mean of the respective newly found cluster.

Figure 4.1 illustrates one iteration step of this algorithm on an exemplary data set. The overall structure of the FCM algorithm is very similar to the one shown above. The FCM algorithm varies in the iteration step, in which FCM also takes the fuzzy clustering into account when calculating the prototypes and assigning the input objects to the prototypes.

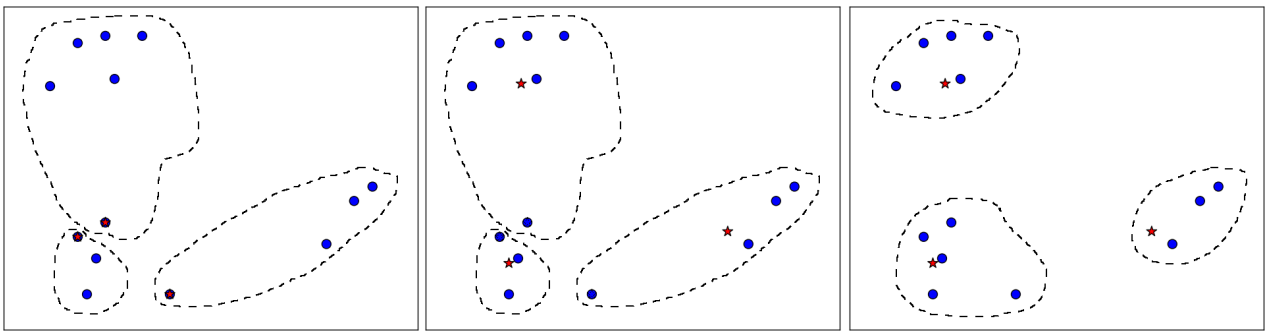

Figure 4.1: *Illustration of one K-Means iteration. The cluster prototypes are represented as red stars. In the beginning, the cluster prototypes are initialized as random vectors out of the set of input vectors.*

4.1.1.2 Intuition of fuzzy clustering

The main difference between K-Means and FCM is that FCM results in a *fuzzy* or soft clustering, while K-Means creates a *strict* clustering. In the strict clustering of K-Means, each data point is assigned to exactly one cluster. In the fuzzy clustering of FCM, each input vector is assigned a membership value, which determines the rate to which the input vector is belonging into one cluster. Each vector's membership to one cluster is a real-valued number between zero and one. A membership close to one implies that the vector is very similar to the cluster, while a membership close to zero shows that the vector beares little similarity with the cluster. Furthermore, the sum of all memberships of each input vector to the various clusters has to be equal to one.[3]

The idea of fuzzy clustering is motivated by the observation that real-world data sets can often not be grouped strictly into different categories. In the case of MOOCs, some courses cover more than one topic or are interdisciplinary. Hence for example the Coursera-based MOOC "Game Theory" that is not just about the mathematical theories on game theory, but also explains it's applications in the social sciences. This course should therefore be clustered together with other MOOCs on mathematics as well with social sciences MOOCs.

[3]Source: [BEF84, introduction].

4.1.2 The Fuzzy C-Means algorithm

Before presenting the actual algorithm, the notation will be fixed:

Let $x_1, \cdots, x_N \in \mathbb{R}^n$ be the input vectors which are going to be clustered. $C \in \mathbb{N}, C > 1$ is the number of resulting clusters. The prototypes will be called $v_1, \cdots, v_C \in \mathbb{R}^n$. Since every prototype represents one cluster, there are as many prototypes as clusters; thus, v_i represents cluster i. The cluster memberships are represented as a matrix $U \in [0,1]^{C \times N}$; therein $U_{i,k} \in [0,1]$ is the membership of input vector x_k to cluster i.

Furthermore, there is a fixed weighting exponent $1 \leq m \leq \infty$. This exponent defines the "fuzzyness" of the clustering: for $m \to 1$ the clustering becomes hard (in this case, the result of FCM is equal to that of K-Means). For $m \to \infty$ the clustering becomes increasingly fuzzy; at some point, every input object belongs to every cluster equally. That is why a "good" choice of m is vital for the quality of the clustering. Bezdek et al. recommend to choose m in the range of $1.5 \leq m \leq 3.0$.[4]

Lastly, one has to fix a constant $\epsilon > 0$, which determines the precision of the computation. The algorithm stops, if the incremental change of the membership matrix is smaller then ϵ. Accordingly, the smaller ϵ is, the more accurate is the result and the higher is the computational cost.

With this notation, the FCM algorithm is as follows:[5]

1. **Initialization** Initialize the prototypes v_i as well as the membership matrix U randomly.
2. **Repeat** (until exit condition is complied)

   2a. Update prototypes using the following equation:

   $$v_i := \frac{\sum_{k=1}^{N} (U_{i,k})^m \cdot x_k}{\sum_{k=1}^{N} (U_{i,k})^m}$$

   2b. Update memberships using the following equation:

   $$U_{i,k} := \left(\sum_{j=1}^{C} \left(\frac{\|v_i - x_k\|}{\|v_j - x_k\|} \right)^{2/(m-1)} \right)^{-1}$$

   2c. Exit condition Exit the loop if the clustering virtually stops changing. In order to do this, compare the membership matrices U_{pre} before and U_{post} after one iteration. If $\|U_{\text{pre}} - U_{\text{post}}\| < \epsilon$, exit the loop.

A good choice of the meta-parameters – mainly C and m – is vital to the quality of the result. This illustrates figure 4.2: this figure contains an exemplary data set, which can be grouped into three obvious clusters. However, the figure shows the result of FCM with $C = 2$: the two clusters marked in green and blue are found by FCM, but as the number of clusters is set to two instead of three, the third cluster is not recognized. Instead, the three cyan colored points are assigned to both other clusters with a membership of 50%.

[4] Source: [BEF84, p. 193].
[5] Source: [BEF84, p. 194].

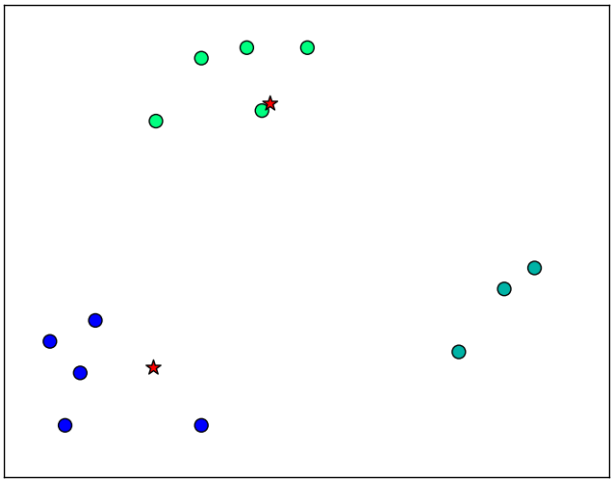

Figure 4.2: *The result of an exemplary FCM run with $C = 2, m = 2.5$. The red stars represent the two cluster prototypes. The blue points belong to the first cluster, while the green points belong to the second cluster. The cyan colored points belong to both clusters evenly.*

Choosing the meta-parameters of the algorithm needs to be evaluated empirically on basis of the actual input data. Chapter 5 shows the process of how to evaluate these meta-parameters, as well as presents the results in the case of the MOOC text descriptions.

4.1.3 Difficulties with Fuzzy C-Means clustering

The result of FCM highly depends on the first random initialization of the cluster prototypes and membership matrix. The FCM algorithm tries to minimize the distance of the data points to their assigned cluster prototype(s), but whether the algorithm finds the global minimum or is stuck in a local minimum often depends on the initialization.[6] To cope with this diffculty, the original FCM algorithm was improved in the following way: for every clustering, the FCM algorithm is started in multiple threads with different random initializations. After the calculations end, for every clustering a score is calculated, which represents the quality of the clustering.[7] Then, the clustering with the best score is returned. In this way, the sensitivity of FCM to it's initialization is reduced.

There are also other algorithms that, instead of initializing randomly, choose the cluster prototypes more deliberately. One example is the "Fuzzy Subtractive Clustering" method developed in [LA11]. Comparing the clustering of MOOCs by this method with the clustering by FCM could be interesting future work.

Another problematic aspect of the FCM algorithm is that it is very sensitive to the *concentration of norm phenomenon*. This phenomenon states that "under certain assumptions the relative distances from any point to its closest and farthest neighbour tend to be almost identical for high-dimensional data".[8] In this case all the center

[6] Source: [LA11, section 1].

[7] As score function, the partition coefficient was used. This score function is explained in detail in section 5.1.3.

[8] Source: [KHJ12, p. 15].

prototypes – independent of their initialization – move to the same centre of the data; because of that, the resulting clusters are worthless. Generally speaking, all clustering algorithms have trouble with high-dimensional input data – this is one of the reasons the dimension reduction techniques presented in chapter 3 were developed. But FCM is particulary affected by this problem, as in comparsion to SOM the problem already occurs for much smaller dimensionsionalities. For example, SOM had no problem with clustering 30-dimensional vectors. In contrast, FCM was not able to cluster these vectors properly, since the resulting clusters were totally fuzzy (i.e. almost all input vectors belong to almost all clusters with small membership).

4.1.4 Implementation

Since the FCM algorithm is comparatively short, it can be easily implemented. However, a straight-forward implementation of FCM with Python 2 using NumPy turned out to be very slow. Clustering a corpus of approximately 1200 MOOC textual descriptions took about two hours. The bottleneck was the updating of the membership matrix (step **2b**), which was implemented with canonical loops, modeled after the original equation:

```
for i in range(0, C):
    for k in range(0, number_of_documents):
        membership = 0.0
        for j in range(0, C):
            fraction = (dist(centers[i], vectors[k]) / dist(centers[j],
                vectors[k]))
            membership += fraction ** (2.0/(m-1.0))
        partition[i][j] = 1.0 / membership
return partition
```

In this implementation, every element of the membership matrix is computed seperately. This poses a problem, because NumPy is optimised for operations that involve entire matrices or vectors, not single scalars. To compute the membership update only with matrix operations, the equation of step **2b** needs to be rewritten:

$$
\begin{aligned}
U_{i,k} &= \left(\sum_{j=1}^{C} \left(\frac{\|v_i - x_k\|}{\|v_j - x_k\|} \right)^{2/(m-1)} \right)^{-1} \\
&= \|v_i - x_k\|^{-2/(m-1)} \cdot \left(\sum_{j=1}^{C} \left(\frac{1}{\|v_j - x_k\|} \right)^{2/(m-1)} \right)^{-1} \\
&= \|v_i - x_k\|^{-2/(m-1)} \Big/ \left(\sum_{j=1}^{C} \|v_j - x_k\|^{-2/(m-1)} \right)
\end{aligned}
$$

Considering this transformed equation, the computation can be done with the following code, using NumPy:

```
# cdist computes all distances between all vectors and all centers
partition = cdist(vectors, centers) ** (-2.0/(m-1.0))
return (partition.T / partition.sum(axis=1)).T
```

This change reduces the runtime of an entire FCM clustering from the two hours mentioned above to only four minutes.

4.2 Self-organizing Maps

As a second clustering method, the artificial neural network *Self-organizing Map* (SOM) was utilized in this research project. This type of neural network creates a spatial, two-dimensional representation of the input data, which is called a map. To compute this map, an unsupervised learning algorithm is used, i.e. the map "organizes itself" without use of any information but the input data. The mechanics of the SOM were inspired by observations on how the brain responds to sensory information.[9] A SOM can be used for example to visualize high-dimensional data in a two-dimensional space. When interpreting the neurons in the neuronal network as cluster prototypes, SOMs can be also considered a clustering algorithm. This approach was used in this bachelor thesis and will be explained in detail in this section.

Self-organizing Maps, also called *Kohonen Maps* after their inventor Teuvo Kohonen, were developed in the early 1980s. A compact overview of the topic can be found in [Koh90]. For a more comprehensive treatment, see the monography [Koh01]. Both are basis of this section.

4.2.1 Intuition behind SOM

Principally, a SOM is a two-dimensional lattice of neurons.[10] Each neuron is identified by it's coordinate on the lattice. Additionally, each neuron holds a weight vector of the same dimension as the input vectors. This weight vector represents the neuron's position in the (possibly high-dimensional) input data space. The training (or "self-organization") process of the SOM aims to "bend" the lattice of neurons over the set of input vectors, such that the map together with it's weight vectors represents the input data.

The training consists of a large number of training steps. In each step, one single input vector is considered. Then, the neuron whose weight vector is nearest to the input vector is updated: this neuron's weight vector is moved closer to the input vector. This neuron is called the winning neuron. Moreover, other neurons, which are close to the winning neuron on the lattice, are moved closer to the input vector as well by some degree.

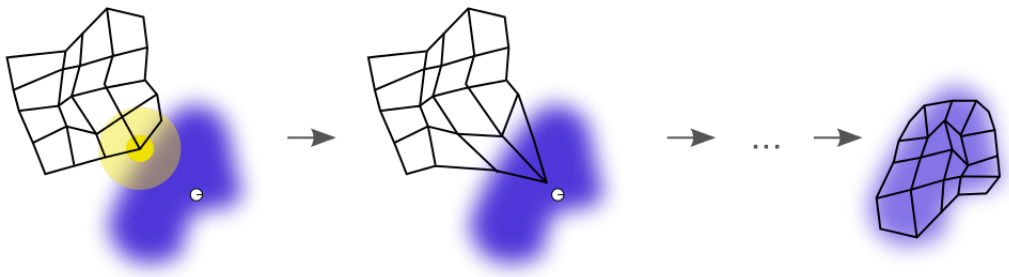

Figure 4.3: *SOM training illustration. The blue cloud represents the various input vectors. The black lattice represents the map; each junction in the lattice is one neuron's weight vector.*

[9] Source: [Koh90, section I].

[10] Theoretically, the map can also have a different dimension, but in practice other dimensions than two are rarely used.

An illustration of the training process is shown in figure 4.3.[11]

4.2.2 Neural network behind SOM

After this short intuitive introduction to SOMs, now a more precise definition of the artifical neural network and it's components will be given. An artifical neural network is "a machine that is designed to *model* the way in which the brain performs a particular task or function of interest".[12] Usually, neural networks contain a large number of interconnected computation units called "neurons", which are able to communicate and store information. For clarity, the neurons are often structured into different layers.[13]

The SOM neural network consists of only two layers: the input and the output layer, without any additional hidden layers (see figure 4.4).

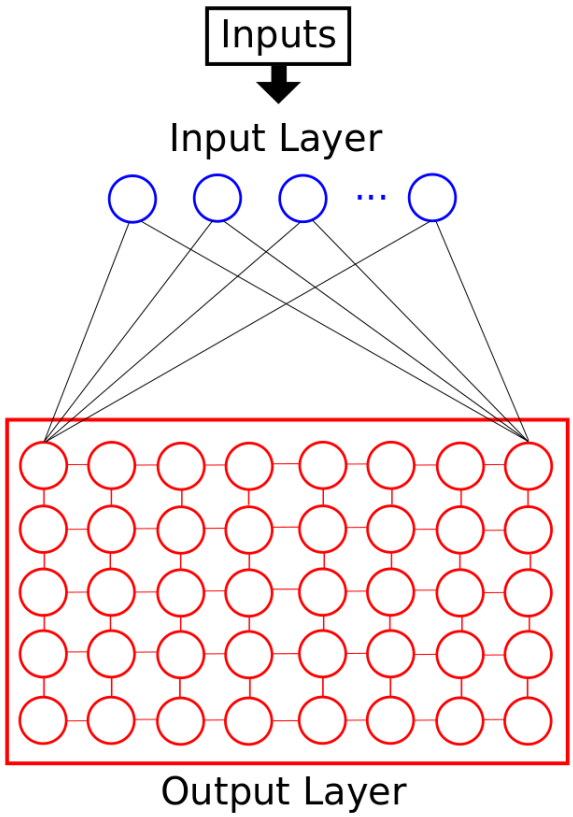

Figure 4.4: *Network architecture of the SOM. Every output node is connected with every input node, but for reasons of clarity not all those connections are indicated in the figure.*

If the input vectors have the dimension n, the input layer consists of exactly n neurons, each neuron corresponding to one input dimension. The output layer contains the aforementioned two-dimensional lattice of $k \times k'$ neurons (the map). If one wants to

[11] The source of this image is `https://en.wikipedia.org/wiki/File:Somtraining.svg`. The image was published under the Creative Commons Attribution-Share Alike 3.0 license.
[12] Source: [Hay99, p. 24].
[13] Source: [Hay99, chapter 1].

use the SOM for clustering, $k \cdot k'$ needs to be the number of clusters. Every input neuron is connected with every output neuron. For every output neuron j, call the coordinate of the neuron on the lattice $c_j \in \mathbb{N}^2$. Furthermore, let $w_j \in \mathbb{R}^n$ denote the neuron's according weight vector.[14]

During the learning process, the input vectors are propagated through the network several times. The SOM learning process works competitively: for each input vector $x_i \in \mathbb{R}^n$, the most similar weight vector w_j is found and is moved to closer to x_i. Besides, the neighbouring neurons of j are updated as well. This propagation is repeated several times with different input vectors.

4.2.3 SOM training algorithm

There exist different versions of SOM training algorithms.[15] First, the widely used algorithm that was implemented and used in this projected will be introduced. Afterwards, possible changes to the algorithm will be discussed.

As explained above, the weight vectors $w_j \in \mathbb{R}^n$ change incrementally with every step during the learning process. Therefore, in the following the weight vectors will be denoted as functions $w_j(t) : \mathbb{N} \to \mathbb{R}^n$ with t being a discrete time-coordinate, representing the current number of steps in the learning process. The initial weights $w_j(0)$ can be chosen randomly.

Consider a single input vector $x \in \mathbb{R}^n$. Furthermore, assume that already t training steps have been computed and call t_{max} the maximal number of steps before the learning process is ended. Then, the input vector is propagated through the neural network as follows:[16]

1. **Find winning neuron** Let j' be the neuron whose weight vector is closest to the input vector (using Euclidean distance):

$$j' := \arg\min_j \|x - w_j(t)\|$$

2. **Update all weight vectors** using the following formula:

$$w_j(t+1) := w_j(t) + \alpha(t) \cdot \eta_{j,j'}(t) \cdot [x - w_j(t)]$$

where $\alpha(t)$ is the learning rate and $\eta_{j,j'}(t)$ the neighbourhood function.

Learning rate $\alpha(t)$
The learning rate factor $\alpha(t)$ shall be a monotonically decreasing function with values in $[0, 1]$. It is important that the first values of α are near unity, because the ordering of the weight vectors happens in the first learning steps. Besides, for $t \to \infty$ the value of $\alpha(t)$ should converge to 0, in order to ensure that the learning process converges. One simple possible candidate for α is:

$$\alpha(t) := \alpha(0) \cdot \left(1.0 - \frac{t}{t_{max}}\right) \qquad \text{with } \alpha(0) := 0.9$$

[14] Source: [HNP05, section 3.2.2].
[15] See [Koh01, chapters 3 and 5] for more details.
[16] Source: [Koh01, chapter 3.1].

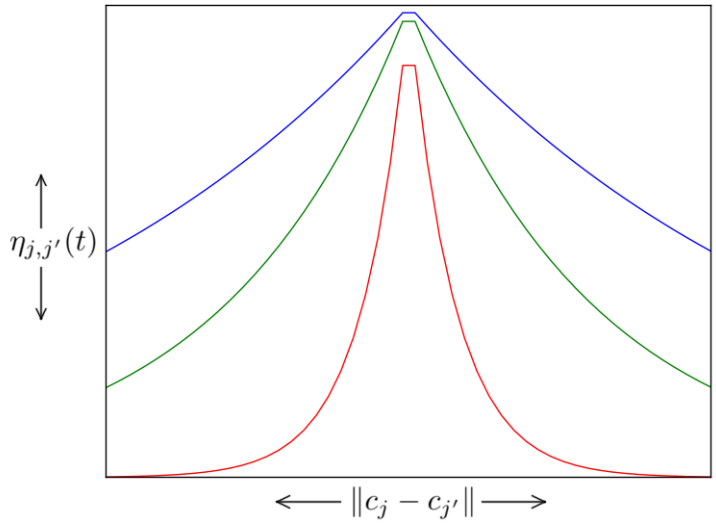

Figure 4.5: *Gaussian neighbourhood function. The figure shows plots of $\eta_{j,j'}(t)$ with three different fixed values for t. The blue plot has the fixed value t_1, the green one t_2, and the red one t_3, with $t_1 < t_2 < t_3$. The x-axis corresponds with the values of $\|c_j - c_{j'}\|$.*

For smaller SOMs like the ones used in this project, the actual shape of α – whether it is linear, exponential, etc. – has no significant impact on the map.[17]

Neighbourhood function $\eta_{j,j'}(t)$
The neighbourhood function $\eta_{j,j'}(t)$ indicates whether the neurons j and j' are neighbours on the two-dimensional lattice or not. If they are neighbours, the value of $\eta_{j,j'}(t)$ shall be one (or close to one), otherwise zero (or close to zero).

The neighbourhood relation is time-dependent: at the beginning of the learning process, the neighbourhood needs to have a big radius, such that almost all neurons are neighbours of one another. Otherwise, the map cannot be ordered globally. In the end however, the neighbourhood should be small, such that only neurons, which are directly connected on the lattice, are neighbours.

A standard candidate for $\eta_{j,j'}(t)$ is the Gaussian function:[18]

$$\eta_{j,j'}(t) := \exp\left(-\frac{\|c_j - c_{j'}\|}{2 \cdot \sigma(t)^2}\right)$$

In this definition, c_j and $c_{j'}$ are the aforementioned coordinates of the considered neurons. The function $\sigma(t)$ signifies the radius of the neighbourhood. Just like $\alpha(t)$, $\sigma(t)$ needs to be a monotonically decreasing function. The initial radius $\sigma(0)$ should be large enough, so that in the beginning of the training one neuron is a neighbour of the majority of all other neurons. In a SOM with a $(k \times k)$ lattice, one could for example set $\sigma(0) := \lfloor \frac{k}{2} \rfloor$. Then, $\sigma(t)$ can be defined linearly like $\alpha(t)$. Alternatively, [Hay99, p. 472] recommends using an exponentially decreasing radius:

$$\sigma(t) := \sigma(0) \cdot \exp\left(-\frac{t}{t_{max}}\right)$$

[17] Source: [Koh01, p. 88].
[18] Source: [Koh01, p. 87].

To illustrate the Gaussian neighbourhood function, figure 4.5 shows exemplary plots of $\eta_{j,j'}(t)$. The plots demonstrate, how the neighbourhood is shaped and how it shrinks over time.

Learning algorithm
The learning algorithm for SOMs consists of repeating the training step described above with different input vectors. As a "rule of thumb", the number of steps shall be at least 500 times the number of neurons in the map.[19] If the number of input vectors is smaller than the number of required training steps, the same input vector can be considered repeatedly by cyclically going through the whole set of input vectors.[20]

4.2.4 Clustering MOOCs with SOM

The SOM neural network and the respective training algorithm, which are described above, were implemented as a Python module using the NumPy mathematics library.

To cluster the MOOC textual descriptions with SOM, the map must have as many neurons as desired clusters. The weights of the neurons can then be initialized by random samples of the input vectors. When the SOM is sufficiently trained, the clusters can be obtained. For that, the winning neuron for each input vector – i.e. the neuron whose weight vector has the smallest distance to the input vector – needs to be found. This neuron represents one cluster, and the considered input vector is grouped into this cluster. After this process is done for every input vector, all clusters are computed and can be returned as a result.

Figure 4.6 contains an exemplary result of a SOM clustering of the MOOC textual descriptions. This example contains an 8x8 grid, i.e. there are 64 clusters. The labels for the clusters were created manually. One can observe in this example, how (up to some exceptions) clusters of similar or related topics are next to each other on the grid.

[19] Source: [Koh01, p. 88].
[20] Source: [Koh01, p. 120].

Entrepeneurship, start-ups	Finances basics	Accounting, finances	Marketing, social media, markets	English courses	Advanced English	Music and education	Music technology
Education, graphic design	Corporate Finance	Financal markets	Writing	Creativity, innovation	Calculus	Learning to teach, technology for teachers	Music history, music production
Data structures in Java	Programming introduction	Python programming introduction	Our place in the world: astronomy, biology	Communication, psychology at work	Logic, analysis, engineering	Artifical intelligence, deep learning	Video games, Game Theory
Algorithms & data structures	Algorithms	Managing business, people, yourself	Statistics	Security, testing, cryptography	Communication in English, foreign languages	Cloud computing, OS, Internet of Things	Game design
Data science, machine learning	Leadership, managemet, HR	Business strategy, marketing	Big Data	Politics, economy	Robotics, System Biology	Lliberal arts education	Business
Big Data technology	Applications of data science	Project planning	Frontend webdesign, AJAX	Advanced webdesign / web programming	Content strategy, SEO	Finances, investing	Digital marketing
Statistical programming	Data analysis introduction	Testing, UI design	Webdesign (HTML, CSS, JS)	Android / Google app services	Foundations of teaching	Economics	Business leadership
Business analytics	Genomes, molecular biology	Programming iOS	Mixed app programming	Programming Android	Android Development	Medicine, healthcare, illnesses	Public / global health

Figure 4.6: *Example result of a SOM clustering with a 8x8 grid. The cluster labels displayed in this figure were created manually.*

Chapter 5

Evaluation of Cluster Validity

Evaluating the quality of a clustering result is a very important issue in cluster analysis. As discussed in the previous chapters, the clusters of a given document corpus can be obtained in various ways depending on the vector representation and the clustering algorithm, resulting in different clusterings. Furthermore, the clustering algorithms depend on a number of a priori fixed meta-parameters, especially the number of clusters, which also influence the clustering. In order to pick the "best" clustering, the quality of the clusterings has to be assessed and compared. This process of evaluating the results of clustering algorithms is called *cluster validity*.[1]

In this process one has to always bear in mind that clusters in real-world data are not as evident as clusters in small artifical data sets. Indeed, clustering is always to some extent subjective. Therefore, in real-world data sets there is usually not one perfect clustering. Instead, different clusterings could all be valid, depending on one's point of view and on how one wants to use the clusters later on. This is one of the reasons, why there are so many different clustering algorithms. Likewise, there are numerous ways of evaluating cluster validity, each having a different stance on what a "good" cluster is.[2]

The approaches of evaluating cluster validity can be roughly divided into two groups: internal and external evaluation.

In *internal* evaluation, only internal data is used in the evaluation process – that is, solely the input vectors and the clusters found by the algorithm. The quality of the clustering is usually determined by (in some way) calculating the compactness and separation of the clusters: the clusters are compact, if the members of one clusters are close to each other; the clusters have a high separation, if the members of different clusters are very different. There are a number of different validity indices, which try to calculate a "score" out of these properties.

In *external* evaluation, the clusters are validated by using some prior knowledge about the input data. This prior knowledge can be for example an already existing classification of the data found by a supervised learning algorithm. Then, the clustering is compared with the classification and some validity index is computed that signifies how similar the clustering and the previous classification are.

[1] Source: [HBV01, p. 123].

[2] A more elaborate discussion of this issue can be found in [EC02].

5.1 Internal Evaluation

5.1.1 Aim of internal evaluation

In the internal evaluation process, validity indices, which represent the compactness and separation of the found clusters, are computed. Using these indices, the meta-parameters like the number of clusters can be evaluated. To do this, the clustering algorithm is executed with various values of the meta-parameter, which one wants to determine. Afterwards, for every result the vaildity index is computed and the clustering with the best index is selected. The value of the meta-parameter of this selected clustering is therefore recommended.

However, one validity index might be biased against one clustering algorithm, if both use similiar definitions of compactness and separation. Thus, internal validity indices can be applied to learn, in which contexts a clustering algorithm works better than in others (i.e. tweaking the meta-parameters). In contrast, comparing two different clustering algorithms – in order to evaluate which algorithm works better on the given data – is not possible using only internal validity indices.[3]

Because strict and fuzzy clusters are fundamentally different, there are different indices for both types of clusterings.

In the following, the validity indices that were applied in this project are introduced. Afterwards, the results of the evaluation considering these indices are summarized.

5.1.2 Validity indices for strict clusters

As briefly discussed in the introduction to this chapter, there are many different validity indices available. Each of these indices has it's own formal definition of how a "good" cluster should be shaped. An extensive study, which compares 30 different validity indices for strict clusters, is [AGM+13]. Unfortunately, their results showed that there is no single best validity index. Instead, how well an index performs always depends on the input data.

For this bachelor thesis, two validity indices for strict clusters were used: Firstly, the classic **Dunn** index, and secondly, the **Calinski-Harabasz** index, which is one of the indices recommended by [AGM+13].

Notation
In the subsequent section, the following notation will be used: Let the set of input vectors be $X = \{x_1, \cdots, x_n\} \subseteq \mathbb{R}^m$. These input vectors are partitioned into k disjoint clusters $C_1, \cdots, C_k$, such that $X = \bigcup_{i=1}^{k} C_i$. Additionally, let $\hat{c}_i \in \mathbb{R}^m$ denote the cluster centers, which are defined as the mean vector of each cluster:

$$\hat{c}_i := \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$$

Dunn index
The Dunn index estimates the compactness and separation of clusters in the following

[3] Source: [EC02, section 4].

way: The compactness of one cluster is represented by it's *diameter*, which is the maximal distance between two vectors of the same cluster. Hence the following definition:

$$diameter(C_i) := \max_{x,y \in C_i} \|x - y\|$$

Analogously, the seperation of two clusters is represented by the *smallest distance* between two vectors of different clusters:

$$distance(C_i, C_j) := \min_{x \in C_i, y \in C_j} \|x - y\|$$

Thereby, the Dunn index of a clustering is defined as the ratio of the minimal distance between two clusters to the maximal diameter of one cluster.

$$Dunn(C_1, \cdots, C_k) := \frac{\min_{1 \leq i < j \leq k} distance(C_i, C_j)}{\max_{1 \leq i \leq k} diameter(C_i)}$$

By this construction, compact and well-separated clusterings have a high Dunn index, because their diameter is small and their distance is large. One downside of this validity index is, that it's computational costs are fairly big.[4]

Calinski-Harabasz index
Like the Dunn index, the Calinski-Harabasz index computes a ratio of separation to compactness, but estimates those properties differently. In Calinski-Harabasz, the compactness of a cluster is based on the distance of each vector to it's cluster center. Furthermore, the separation of the clusters is represented by the distance of the cluster centers to the *global* center of the input vectors. Therefore, let $\hat{x} \in \mathbb{R}^m$ be the global center:

$$\hat{x} := \frac{1}{n} \sum_{j=1}^{n} x_j$$

The Calinski-Harabasz index is then defined as follows:[5]

$$\begin{aligned} \mathrm{CH}(C_1, \cdots, C_k) : &= \frac{\sum_{i=1}^{k} |C_i| \cdot \|\hat{c}_i - \hat{x}\|}{k-1} \Big/ \frac{\sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \hat{c}_i\|}{n-k} \\ &= \frac{n-k}{k-1} \cdot \frac{\sum_{i=1}^{k} |C_i| \cdot \|\hat{c}_i - \hat{x}\|}{\sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \hat{c}_i\|} \end{aligned}$$

Similarly to the Dunn index, compact and well-separated clusterings will have a high Calinski-Harabasz index.

5.1.3 Validity indices for fuzzy clusters

In fuzzy clusterings, one data point can be member of more than one cluster. Therefore, the validity indices for strict clusterings described above can't be applied to fuzzy clusterings. Instead, special validity indices for fuzzy clusters, which take the membership matrix into account, were developed. In the following, a number of these indices

[4] Source: [HBV01, pp. 130, 131].
[5] Source: [AGM+13, section 3.2].

are presented. For that, the notation from section 5.1.2 needs to be extended by introducing the membership matrix $U \in [0,1]^{k \times n}$. Therein, $U_{i,j} \in [0,1]$ is the membership of input vector x_j to cluster i.

Partition coefficient
The partition coefficient was proposed by the inventors of Fuzzy C-Means in [BEF84, p. 194]. It is computed using only the membership matrix:

$$\mathrm{PC}(U) := \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{k} (U_{i,j})^2$$

The values of this index range in $[\frac{1}{k}, 1]$. To make the values of the partition coefficient independent of k, the following modification is recommended:[6]

$$\mathrm{MPC}(U) := 1 - \frac{k}{k-1} (1 - \mathrm{PC}(U)) \in [0,1]$$

In general, the clustering recommended by the (modified) partition coefficient, is the clustering with the highest index value. Both indices are very easy to compute, but do not take the values of the actual input vectors into account.

Fukuyama-Sugeno index
A validity index, that uses the membership matrix as well as the values of the input vectors, is the Fukuyama-Sugeno index.[7] For this, let $v_i \in \mathbb{R}^m$ denote the cluster prototype for cluster i and let $\bar{v} := \frac{1}{k} \sum_{i=1}^{k} v_i$ be the mean of those prototypes.[8] Moreover, let m be FCM's fuzzyness constant. The Fukuyama-Sugeno index combines two score functions, which represent the compactness respectively the separation of the clusters.

Firstly, the function $J_m(U)$ calculates the distances of the input vectors to the cluster prototypes, weighted by their membership to the respective cluster. If the values of $J_m(U)$ are low, this means the distances are small and the compactness of the clusters is high.

$$J_m(U) := \sum_{i=1}^{k} \sum_{j=1}^{n} (U_{i,j})^m \cdot \|x_j - v_i\|^2$$

Secondly, the function $K_m(U)$ measures the the separation of the clusters by calculating the distances of the cluster prototypes to the mean prototype $\bar{v}$. If these distances are high, this implies that the clusters are well-separated.

$$K_m(U) := \sum_{i=1}^{k} \sum_{j=1}^{n} (U_{i,j})^m \cdot \|v_i - \bar{v}\|^2$$

Together, these two function form the Fukuyama-Sugeno index:

$$\mathrm{FS}_m(U) := J_m(U) - K_m(U)$$

In contrast to the other indices mentioned above, the Fukuyama-Sugeno index recommends the clustering with the *smallest* index value.

[6] Source: [WZ07, section 4.1].
[7] Source: [WZ07, section 4.2].
[8] The cluster prototypes are found by FCM in step **2b** of the algorithm, see section 4.1.2.

5.1.4 Results of internal evaluation

The aim of the internal evaluation is to find suitable values for the different meta-parameters of the clustering algorithms. The two main parameters are the number of clusters and the number of dimensions of the input vectors (using either LSI or LPI to reduce the dimensionality of the vectors). Besides, there are also more technical meta-parameters, like FCM's fuzzyness constant m and the values for $\alpha(0)$ and $\sigma(0)$ in SOM. *All* of the meta-parameters affect the outcome of the algorithms. Therefore, if one evaluates the value of one meta-parameter, all the other meta-parameters need to be set to a fixed value. Then, the value of the meta-parameter, which one wants to evaluate, can be varied and the score of the indices presented above be plotted.

At first, the more technical parameters will be evaluated. These results will then be used to determine the number of clusters and the dimension of the input vectors.[9]

5.1.4.1 The technical parameters of FCM and SOM

Fuzzyness constant
The first meta-parameter to evaluate is the fuzzyness constant m of FCM (see section 4.1.2). [BEF84, p. 193] recommends m to be in the range of $1.5 \leq m \leq 3.0$. In this case, the results were clear: all indices for FCM conclusively recommended the value of m to be $m = 1.5$. Indeed, this conclusion was independent of the input vectors used and the number of clusters. For all larger values of m, the the clusters found by FCM were *too* fuzzy (i.e. the majority of MOOCs was assigned to more than four different clusters). This made the clustering inconclusive and therefore hardly useful.

Initial learning rate and initial neighbourhood radius
The next meta-parameters to evaluate are the initial learning rate factor $\alpha(0)$ and the initial neighbourhood radius $\sigma(0)$. Both are needed to define the learning rate function and the neighbourhood function in SOM. The value of $\alpha(0)$ needs to be in the range of $0 < \alpha(0) \leq 1$, and $\sigma(0)$ should lie in $0 < \sigma(0) \leq 5$ (see section 4.2.3).

Here, the best scores of the validity indices were found for $\alpha(0) = 1$ or $\alpha(0) = 0.75$ and $\sigma(0) = 1$ or $\sigma(0) = 2$, depending on the number of clusters. However, the scores of the validity indices varied only slightly. This suggests that these two meta-parameters do not affect the SOM clustering results significantly.

5.1.4.2 The number of clusters

Both clustering algorithms FCM and SOM depend on the number of requested clusters, and this meta-parameter can influence the results considerably. With this meta-parameter one has to again keep in mind, that clustering is subjective – as a result, different numbers of clusters could actually all be reasonable.

For the number of clusters, the result of the internal evaluation was not obvious, because some validity indices were inconclusive and contradicted each other. In the following, the results of the different validity indices are presented. In the course of this, the values for m, $\alpha(0)$ and $\sigma(0)$ were set as described above. The dimensionality of the LSI- and

[9] All data used for this evaluation can be found in a spreadsheet that is attached to this bachelor thesis.

LPI-reduced vectors were fixed as well; the number of dimensions of the vectors will be justified later (see section 5.1.4.3).

In the case of **FCM with LSI reduced vectors** (see figure 5.1, top row), the the score Modified Partition Coefficient (MPC) has only slight variations. The best score can be observed for the smallest number of clusters $c = 20$; the MPC decreases almost monotonically with increasing number of clusters. In contrast, the best Fukuyam-Sugeno (FS) index can be found for $c = 82$.[10] However, the plot of makes a lot of jumps and is altogether rather inconclusive.

For **FCM with LPI reduced vectors** (see figure 5.1, bottom row), the MPC plot is almost reversed to the one with LSI vectors: the MPC has the best score for the highest number of clusters, i.e. $c = 90$. But again, the variations in MPC scores are relatively small. The Fukuyama-Sugeno index reaches the first optimum at $c = 42$ and then with increasing c starts to jump around, but has a global optimum (like MPC) in $c = 90$. However, the FS index has a rather small variation in this case, as well.

In the case of **SOM with LSI reduced vectors** (see figure 5.2, top row), the Dunn index turns out to be impractical, since it's value is almost constant. The Calinski-Harabasz (CH) index recommends the smallest $c = 18$, as the plot is monotonically decreasing.

Considering **SOM with LPI reduced vectors** (see figure 5.2, bottom row), the Dunn index is impractical, as well. The CH index however reaches it's optimum at $c = 36$.

All in all, the internal evaluation – with the four validity indices applied – can unfortunately not recommend a number of clusters for the MOOC textual descriptions consistently.

After going through the MOOC textual descriptions and comparing clusterings with different numbers of clusters manually, the value $c = 64$ turned out to be a reasonable number of clusters. With this number of clusters, most automatically detected clusters had a coherent topic and were of feasible size. Therefore, the rest of the evaluation will use $c = 64$ as number of clusters.

[10] As opposed to all other indices, for the FS index the *lowest* score is the best one!

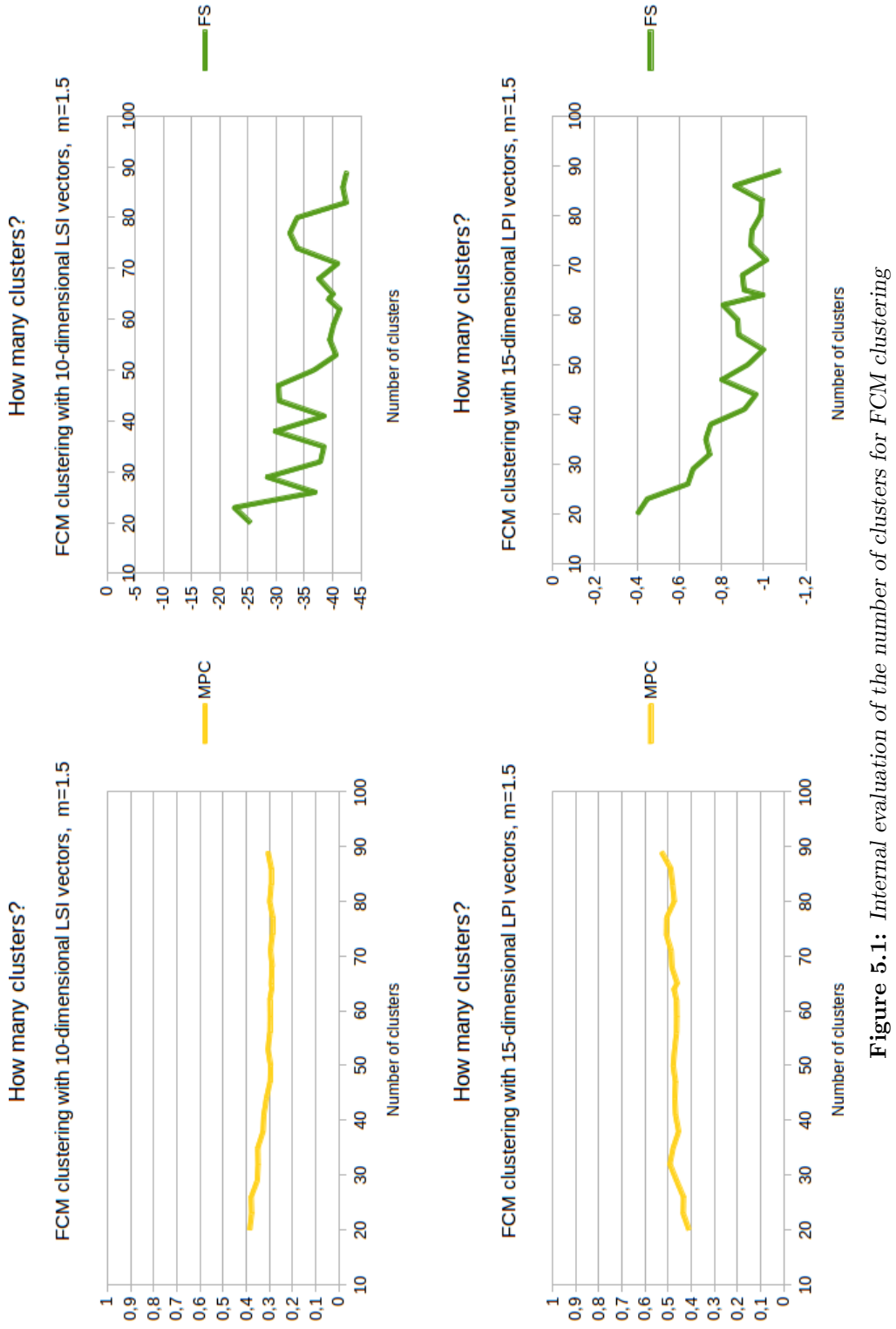

Figure 5.1: *Internal evaluation of the number of clusters for FCM clustering*

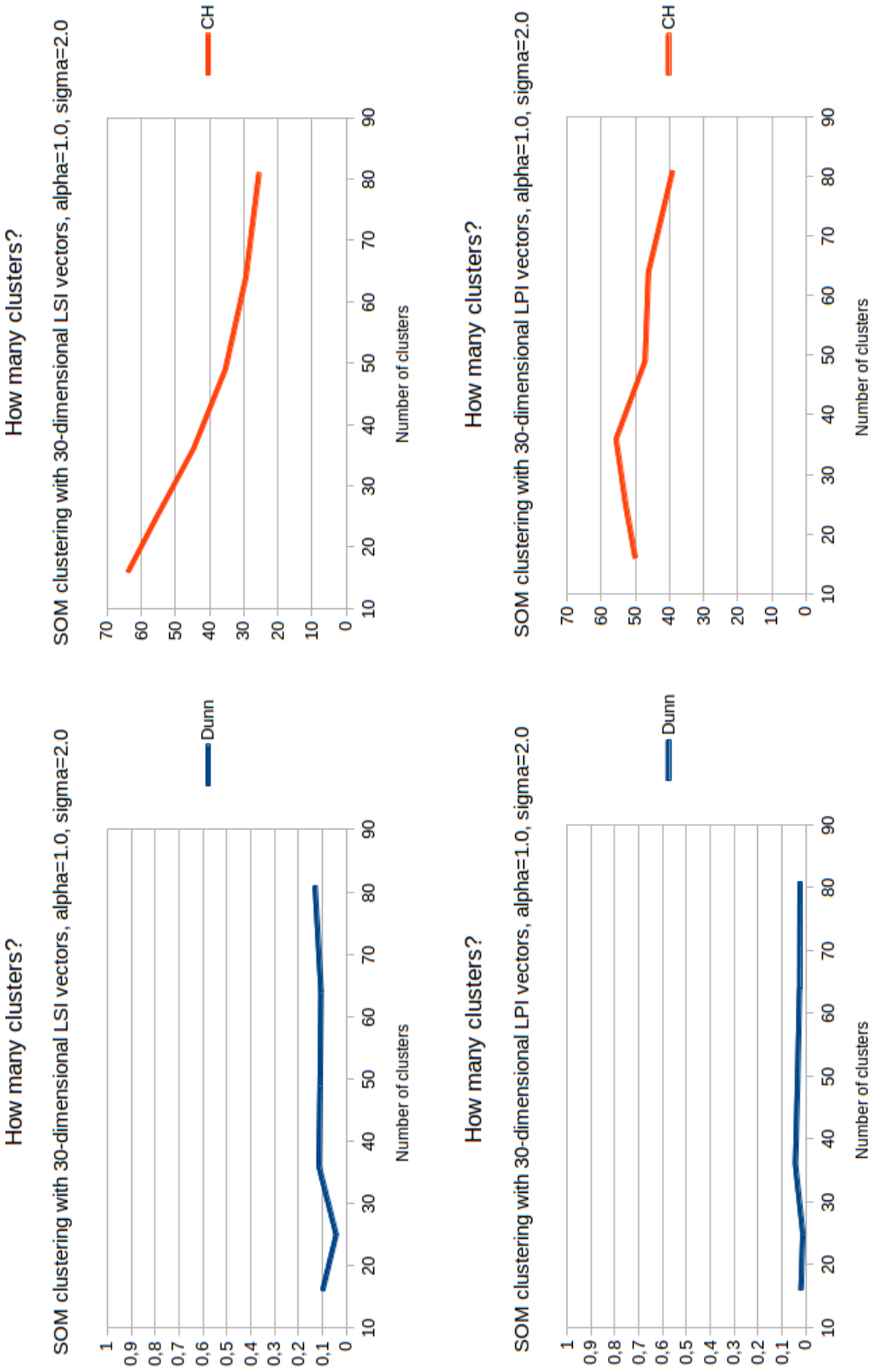

Figure 5.2: *Internal evaluation of the number of clusters for SOM clustering*

5.1.4.3 The number of dimensions of the input vectors

As described in chapter 3, the high-dimensional TF-IDF input vectors need to be reduced to a smaller dimensionality using LSI or LPI. This number of dimensions is the last meta-parameter for the internal evaluation.

In the case of clustering with **FCM using LPI reduction** (see figure 5.3), the MPC index is steadily decreasing the larger the dimension of the input vectors becomes. This means that MPC is recommending the smallest dimensionality $dim = 4$. In contrast, the Fukuyama-Sugeno index has a global optimum at $dim = 21$.

Considering **SOM using LPI reduction** (see figure 5.3 as well), the Dunn index reaches it's optimum at $dim = 28$, with very small (i.e. bad) scores for dimensions smaller than 10. The Calinski-Harabasz on the other hand decreases with increasing number of dimensions and therefore contradicts the Dunn index with recommending $dim = 4$.

For **FCM using LSI reduction** (see figure 5.4), the plot of the MPC index is similar to the one for LPI reduction and recommends $dim = 4$ as well. The optimum of the Fukuyama-Sugeno index is already reached for $dim = 9$.

When clustering with **SOM using LSI reduction** (see figure 5.4), both plots of the Dunn index and Calinski-Harabasz index are similar to the corresponding plots with LPI reduction. The optima are almost the same, too: for Dunn it is $dim = 27$, for Calinski-Harabasz it is $dim = 4$.

To summarize, for FCM the indices mostly suggest to use a rather small number of dimensions. When manually comparing the different clusterings, this is confirmed: for dimensionalities larger than 15, already hundreds of MOOCs don't belong to any cluster with a membership greater than 30%. These MOOCs therefore belong to too many clusters, so that the resulting clusters can not be used for a helpful MOOC recommendation system.

For SOM clustering, the two indices used suggest different results: the Dunn index recommends a large (around 28) number of dimensions, while the Calinski-Harabasz index recommends a minimal number of dimensions. However, the external evaluation (as described in section 5.2) confirmes the Dunn index and recommends a higher number of dimensions, too.

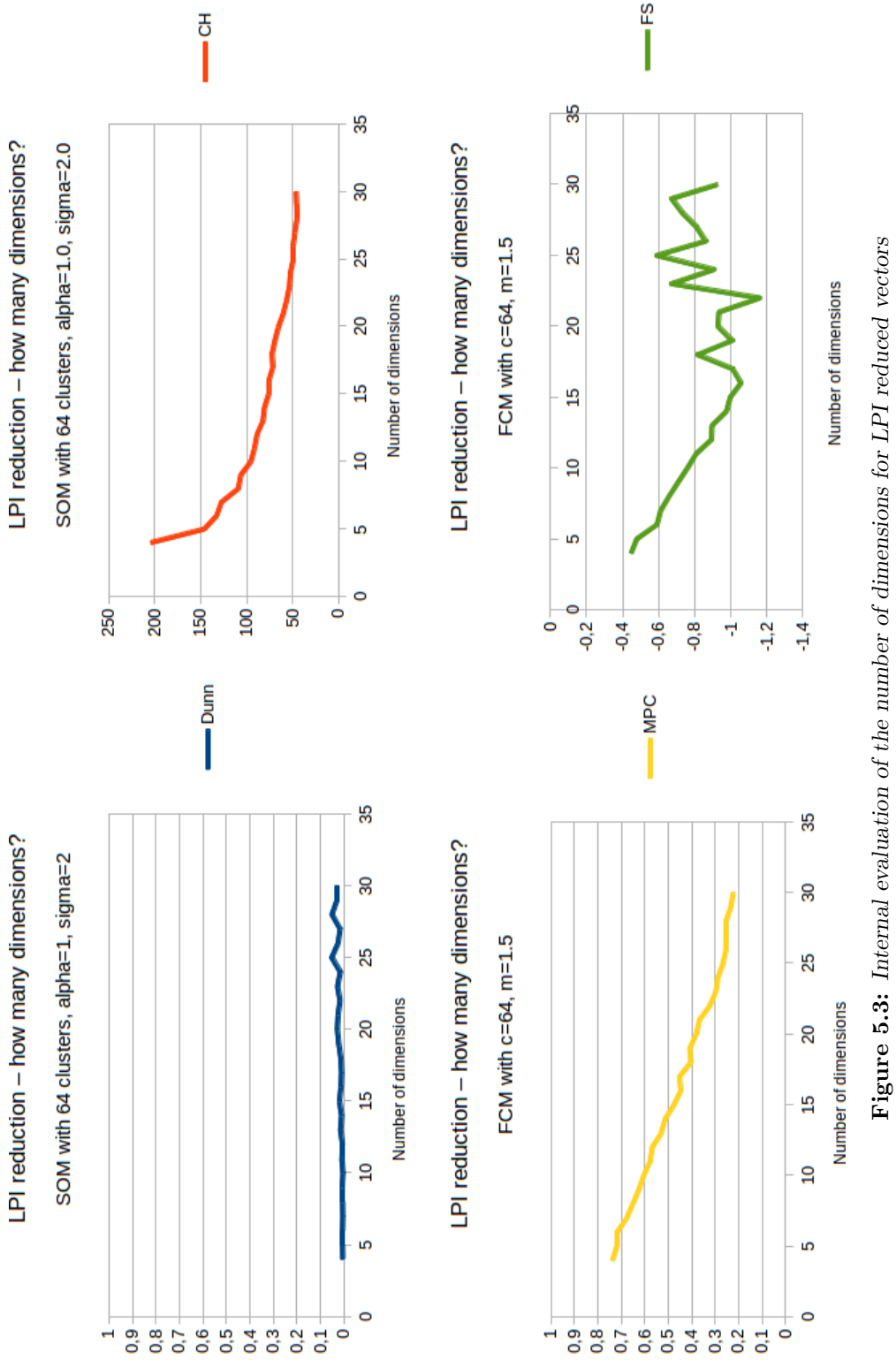

Figure 5.3: *Internal evaluation of the number of dimensions for LPI reduced vectors*

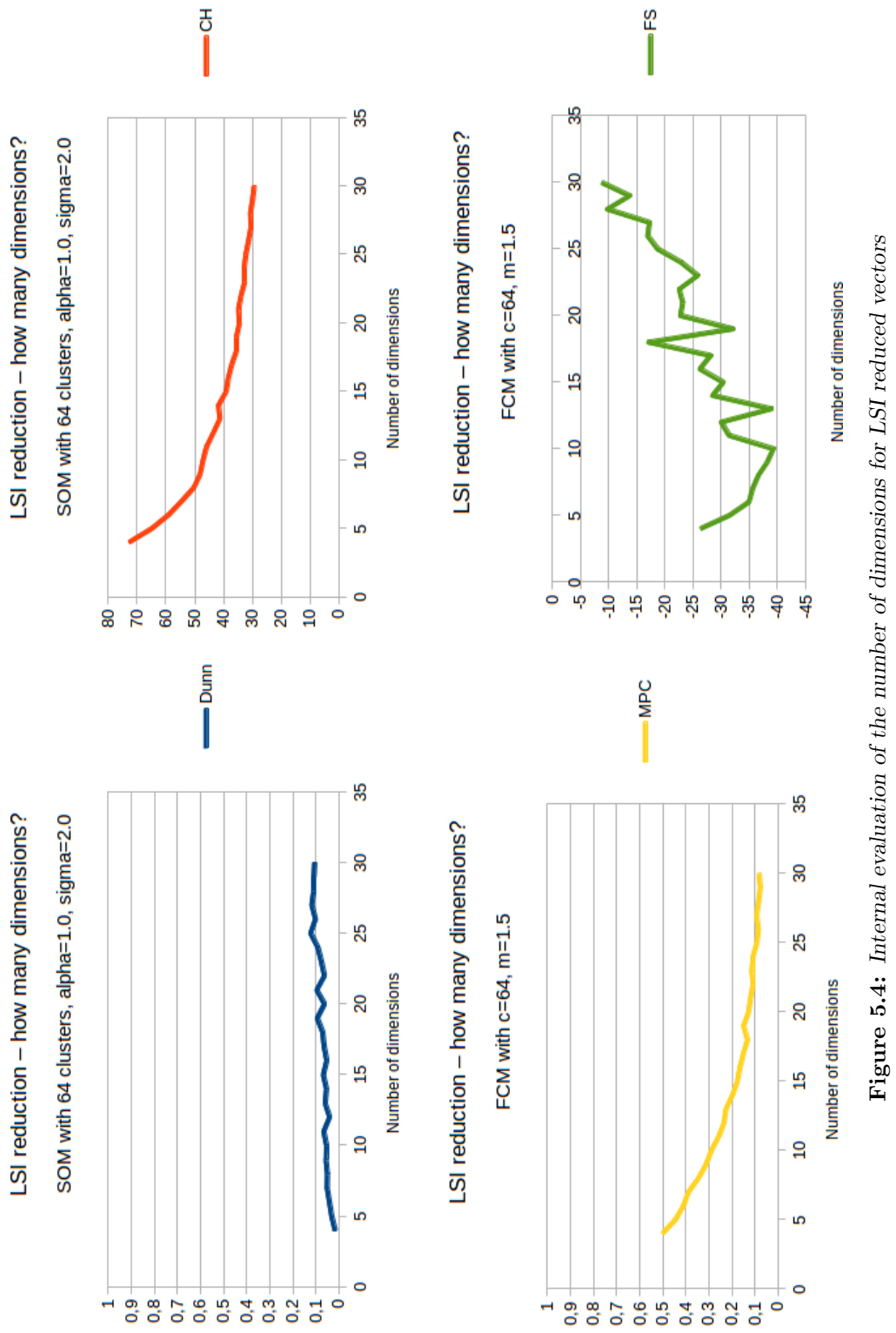

Figure 5.4: *Internal evaluation of the number of dimensions for LSI reduced vectors*

5.2 External Evaluation

5.2.1 Approach to external evaluation

In external evaluation, additional information about the data besides the input vectors are used to evaluate the clusterings. Because there was no classification of the MOOC offers already available, the author of this thesis created one clustering (with 64 clusters) manually. For the external evaluation, this manually created clustering is considered the *gold standard*, i.e. the "valid" clustering of the data. Then, all automatically created clusterings are compared to this gold standard.

Arguably, this gold standard is very subjective, since it was constructed by only a single person. This is the main drawback of this approach. One could solve this problem by letting a number of people (at least two) create clusterings on their own and use them as gold standard. In this way, the gold standard would be more objective. Unfortunately due to time constraints, this was not possible in this research project.

The manually created clustering is strict. That is why this kind of external evaluation only works for SOM clustering. There are different ideas on how to evaluate fuzzy clusterings externally. Because there was not enough time to implement these ideas, they are described in the future work section 6.2.

5.2.2 Validity index for external evaluation

In order to compare a clustering with the gold standard, the **Purity** validity index was used. This index assignes each cluster to the gold standard cluster that occurs most frequent in the automatically created cluster. The Purity index then measures the validity by counting the correctly assigned input vectors.

Formally, the Purity index is defined as follows:[11] Let $C_1, \cdots, C_k$ denote the clusters detected by the clustering algorithms and $G_1, \cdots, G_k$ the clusters of the gold standard. Furthermore, let N be the number of input vectors. The Purity validity index can the be computed as follows:

$$Purity(C_1, \cdots, C_k, G_1, \cdots, G_k) := \frac{1}{N} \sum_{i=1}^{k} \max_j |C_i \cap G_j|$$

If a clustering has Purity index 0.5, this means that 50% of the input vectors are in the correct cluster with right to the gold standard.

5.2.3 Results of external evaluation

The gold standard was compared to various SOM clusterings with 64 clusters. The resulting Purity index ranged from 0.21 to 0.42. Here, a maximal rate of 42% correctly assigned MOOCs may seem low at first. But it is rather unlikely to have a very high percentage of correctly assigned MOOCs, because clustering MOOCs on the basis of their topic is an ambiguous task with numerous different clusterings, which can all

[11] Source: [RAAQ11, section II/B].

be considered valid. The manually assembled gold standard is just one of these valid clusterings.

Learning rate and neighbourhood radius
Considering SOM's initial learning rate $\alpha(0)$ and initial neighbourhood radius $\sigma(0)$, the results of the external evaluation affirm the internal evaluation results described in section 5.1.4.1. The best Purity score was found for a SOM clustering with $\alpha(0) = 1.0$ and $\sigma(0) = 2.0$. However, just like the internal indices, the Purity score changed only minimally with different values for $\alpha(0)$ and $\sigma(0)$: the difference usually did not exceed the value 0.01. This confirms the observation that these two meta-parameters do not affect the SOM clustering particularly.

Dimensionality of the input vectors
The number of dimensions of the input vectors did in contrast affect the clustering and the Purity index significantly. In general, the Purity index rose with an increasing number of dimensions. This supports the result of the Dunn index and contradicts the one by the Calinski-Harabasz index. For both LSI and LPI, the Purity index suggests to reduce the input vectors to a dimensionality around 30.

Comparing LSI and LPI
The best SOM clustering with LSI reduced vectors obtained a Purity score of 42%, while the best clustering with LPI reduced vectors received a Purity index of 37%. However, many clusterings obtained with LPI vectors were reasonable as well (from the author's point of view). Therefore, this evaluation does not decide ultimately, whether LSI is better for clustering MOOC textual descriptions than LPI.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

In this bachelor thesis, two clustering algorithms – FCM and SOM – as well as two dimensionality reduction methods – LSI and LPI – were compared. The evaluation showed that for clustering MOOC textual descriptions (with the application of recommending MOOCs in mind), SOM yields generally better results than FCM. On the two dimensionality reduction methods however, no clear winner was found, as both methods created vectors suitable for clustering.

Comparing FCM and SOM
In the FCM cluster analysis, even with a small fuzzyness constant $m = 1.5$, most of the time the results were *too* fuzzy. Especially when the dimension of the input vectors was larger, the clusters became so fuzzy, that the majority of MOOCs were assigned to too many clusters. This rendered the clusters useless. For example, with the vectors reduced to dimension 14 using LSI, only 500 of the approximately 1300 MOOCs were assigned to at least one cluster with more than 30% membership.

In contrast to FCM, SOM was able to cluster vectors with higher dimensionality (e.g. 30 dimensions). Indeed, SOM generally obtained *better* results when the input vectors were of higher dimensionality. This is the main reason, why the clustering of MOOC textual descriptions was more successful with SOM than with FCM.

Comparing LSI and LPI
The results of the comparison of LSI and LPI were not as clear as the ones of FCM and SOM, because the SOM clustering algorithm created reasonable clusterings with both LSI and LPI reduced vectors. The internal evaluation showed conflicting conclusions: both the Modified Partition Coefficient and the Calinski-Harabasz index favored the LPI reduction, while the Dunn and Fukuyama-Sugeno indices favored the LSI reduced vectors. In the external evaluation using the Purity index, the LSI reduced vectors performed slightly better than the LPI reduced vectors.

In order to have the advantages of both dimensionality reduction methods, one could develop a new method by concatenating both algorithms. For example, one could reduce the dimension of the vectors from 20,000 to 1,000 with LSI and then reduce those vectors even more to dimension 30 with LPI (or vice versa). This idea could be elaborated in a future project.

Lessons learned from evaluation process
The original idea for the internal evaluation process was to *automatically* find the best clustering using a validity index. This was unfortunately not possible, because many times the different validity indices contradicted each other or returned inconclusive results. As a result, one still needs to look through and compare various clusterings manually, which is a time-consuming job. Therefore, the author suggests for future cluster analyses to spend not too much time on the internal evaluation and to focus more on the external evaluation.

6.2 Future Work

6.2.1 Possible improvements for FCM and SOM

As this was the very first cluster analysis of MOOC text descriptions, there are still some questions open for further investigations. The approach to cluster analysis, which was used in this project, can be altered in a couple of ways. Firstly, only the traditional Euclidian distance function was used for the clustering algorithms, although one could also apply other distance functions. Those other distance metrices may be more suitable for the vector space of the MOOC corpus than the Euclidean one.

Furthermore, designing a new distance function specifically for the application of MOOC recommendation is very likely to lead to better results. Assume introductory and advanced MOOCs shall not be clustered together. Then, the new distance function can be implemented to increase the distance of two MOOCs that differ in level of difficulty. This ensures that these two MOOCs are probably not clustered together. How exactly such a novel application-centric distance function should be defined, needs some further investigation.

Secondly, the FCM algorithm was initialized by random cluster prototypes. In [LA11], Le and Altman developed a new automatic initialization method for FCM, which could be an improvement for the MOOC clusterings found by FCM.

An alternative idea for improving FCM's initialization is to construct the initial cluster prototypes from names of different MOOC topics. These names can for example be extracted from the MOOC categories present on the various MOOC platforms. Afterwards, the names of the topics are transformed into vectors just like the MOOC textual descriptions and the resulting vectors can be used as initial cluster prototypes.

Thirdly, there exists a different approach to SOM clustering: in this thesis, the SOM clustering was obtained by training a SOM where each neuron represents one cluster. However, a second approach was developed in [VA00], where the SOM has a very large number of neurons (larger than the number of documents in the corpus). This SOM is used as a two-dimensional representation of the document space and is, after the training of the SOM, clustered traditionally with K-Means. Since the SOM clustering performed well within our project, this new approach could be promising. For all these proposed alterations, it would be interesting to know whether the changes improve the cluster analysis or not.

6.2.2 Other methods for cluster analysis

Paragraph Vector
Of course, there are many more clustering algorithms and document representations than the ones used in this project. Because SOM worked better on the MOOC text descriptions, applying more refined clustering methods based on neuronal networks are particulary promising. Furthermore, Google researchers developed a new document representation called *Paragraph Vector*, which is based on an unsupervised learning algorithm.

With the help of neural networks, vector representations of single *words* (not documents) can be learned. These representations can be more refined than "bag of words" vectors. Paragraph Vector generalizes this idea in order to represent whole *documents* by using some sophisticated average of the vector representations of the words that are present in the document.[1] According to their study in [LM14], this representation can outperform classical "bag of words" representations. Utilizing these methods for a new clustering of MOOCs and comparing the results with clusterings from this thesis is possible future work.

Semi-supervised clustering
A particulary interesting idea to refine the cluster analysis is to use *semi-supervised* learning algorithms in addition to the unsupervised clustering algorithms. This notion of "semi-supervised clustering" is described in [CCM03]. The fundamental concept is to first create the clustering with an unsupervised algorithm. Then, the users of the clustering give some partial feedback (e.g. whether some document does or doesn't fit into it's cluster). After that, the clustering can be improved using this feedback. This idea can be related to the field of "Human Computation".

For generating the user feedback, the MOOC recommendation system that is developed as part of the *IROM* project should be used. The recommendation system utilizes the clusters to find helpful MOOCs, which are relevant to some search query. Then, if many users click on one recommended MOOC, it was relevant and therefore was probably put into the right cluster. Vice versa, if no user clicks on one MOOC although it is recommended, the MOOC was probably placed into a wrong cluster and due to that was not recommended properly.

Alternatively, the MOOC recommendation system could ask directly for feedback. For example, the system might add to each MOOC recommendation the question "Was this recommendation useful? Yes / No", which the the users then can answer. The replies to this question correspond directly to the question, whether the recommended MOOC belongs to the correct cluster.

Using these data to implement the semi-supervised clustering with user feedback described above is an attractive future project. Nevertheless, it will take some time until the recommendation system had enough traffic in order to generate substantial feedback data.

Extending the external evaluation
The feedback data of the MOOC recommendation system, as described above, could also be used to extend the external evaluation. In order to compare two different clusterings, both clusterings must be utilized by the recommendation system alternating.

[1] Source: [LM14].

When enough feedback data is gathered, this external evaluation strategy then recommends the clustering with better feedback by the users. Since this feedback is created by (hopefully) many different users and not just by a small number of experts, it has the potential to improve the external evaluation.

Improving the data

In this bachelor thesis, the only data available for clustering were the MOOC textual descriptions, due to limitations of the crawler. Nevertheless, most MOOC providers offer more information about the MOOCs, like the category of the MOOC, it's syllabus or the reviews posted by former students. The clustering results can probably be improved by taking these additional information into account. For example, one could aggregate MOOCs from one category and create an hierarchical clustering of this specific category. How exactly the additional data should be utilized in the clustering process, and how this would change the clustering results, needs to be evaluated in the future.

List of Figures

Bibliography

[AGM+13] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M. Pérez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013.

[BEF84] James C. Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2):191–203, 1984.

[Bry13] François Bry. Lecture notes in web information systems, October 2013.

[CCM03] David Cohn, Rich Caruana, and Andrew McCallum. Semi-supervised clustering with user feedback. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, 4(1):17–32, 2003.

[CHH05] Deng Cai, Xiaofei He, and Jiawei Han. Document clustering using locality preserving indexing. *Knowledge and Data Engineering, IEEE Transactions on*, 17(12):1624–1637, 2005.

[DDF+90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[dt16] MongoDB developer team. *BSON – Binary JSON*, accessed December 1, 2016. `http://bsonspec.org/`.

[EC02] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD explorations newsletter*, 4(1):65–75, 2002.

[Hay99] Simon O. Haykin. *Neural Networks. A Comprehensive Foundation*. Pearson Education, 2. edition, 1999.

[HBV01] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of intelligent information systems*, 17(2-3):107–145, 2001.

[HCLM04] Xiaofei He, Deng Cai, Haifeng Liu, and Wei-Ying Ma. Locality Preserving Indexing for document representation. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 96–103. ACM, 2004.

[He05] Xiaofei He. *Locality Preserving Projections*. PhD thesis, Computer Science Department, The University of Chicago, December 2005.

[HK06] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, USA, 2. edition, 2006.

[HN04] Xiaofei He and Partha Niyogi. Locality Preserving Projections. In *Neural information processing systems*, volume 16, page 153. MIT, 2004.

[HNP05] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62, 2005.

[KA08] Mei Kobayashi and Masaki Aono. Vector space models for search and cluster mining. In *Survey of Text Mining II*, pages 109–127. Springer, 2008.

[KHJ12] Frank Klawonn, Frank Höppner, and Balasubramaniam Jayaram. What are clusters in high dimensions and are they difficult to find? In *International Workshop on Clustering High-Dimensional Data*, pages 14–33. Springer, 2012.

[KKL+00] Teuvo Kohonen, Samuel Kaski, Krista Lagus, Jarkko Salojarvi, Jukka Honkela, Vesa Paatero, and Antti Saarela. Self organization of a massive document collection. *IEEE transactions on neural networks*, 11(3):574–585, 2000.

[Koh90] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[Koh01] Teuvo Kohonen. *Self-Organizing Map*. Springer, Berlin, Heidelberg, 3. edition, 2001.

[LA11] Thanh Le and Tom Altman. A new initialization method for the fuzzy c-means algorithm using fuzzy subtractive clustering. In *Proceedings of the 2011 International Conference on Information and Knowledge Engineering, Las Vegas USA*, pages 144–150, 2011.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[LR01] Avraham Leff and James T. Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[LSM91] Xia Lin, Dagobert Soergel, and Gary Marchionini. A self-organizing semantic map for information retrieval. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 262–269. ACM, 1991.

[ML06] Sueli A. Mingoti and Joab O. Lima. Comparing SOM neural network with Fuzzy c-means, K-means and traditional hierarchical clustering algorithms. *European Journal of Operational Research*, 174(3):1742 – 1759, 2006.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[RAAQ11] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M. Quiroz. Internal versus external cluster validation indexes. *International Journal of computers and communications*, 5(1):27–34, 2011.

[RS04a] M.E.S. Mendes Rodrigues and L. Sacks. Dynamic knowledge representation for e-learning applications. In B. Azvin M. Nikravesh, L.A. Zadeh and R. Yager, editors, *Enhancing the Power of the Internet – Studies in Fuzziness and Soft Computing*, pages 255–278. Springer, January 2004.

[RS04b] M.E.S. Mendes Rodrigues and L. Sacks. A scalable hierarchical fuzzy clustering algorithm for text mining. In *Proceedings of the 4th International Conference on Recent Advances in Soft Computing*, pages 269–274. RASC'2004, December 2004.

[ŘS10] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[TBI97] Lloyd N. Trefethen and David Bau III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1997.

[VA00] Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE transactions on neural networks*, 11(3):586–600, 2000.

[Wis13] Laurenz Wiskott. Lecture notes on principal component analysis, February 2013.

[WKK12] Roland Winkler, Frank Klawonn, and Rudolf Kruse. Problems of fuzzy c-means clustering and similar algorithms with high dimensional data sets. In *Challenges at the Interface of Data Analysis, Computer Science, and Optimization*, pages 79–87. Springer, 2012.

[WZ07] Weina Wang and Yunjie Zhang. On fuzzy cluster validity indices. *Fuzzy sets and systems*, 158(19):2095–2117, 2007.