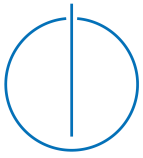
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

A Recommender System using Clustering with Sparse Grid Density Estimation

Matthias Fischer



DEPARTMENT OF INFORMATICS

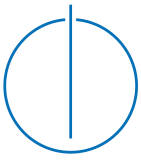
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

A Recommender System using Clustering with Sparse Grid Density Estimation

Ein Vorschlagssystem durch Clustering mithilfe Dünngitter-Dichteschätzung

Author:	Matthias Fischer
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	M. Sc. Kilian Röhner
Submission Date:	April 15, 2016



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, April 15, 2016 Matthias Fischer

Acknowledgements

This thesis marks the end of a chapter of life called study and it is time to look back and say thanks to the people who made it a good and successful one.

First, thanks go to my advisor Kilian Röhner for making it possible for me to work on this great subject as well as for all the fruitful discussions and all the provided feedback.

Special thanks go also to the admins of the SCCS chair, namely Roland Wittmann and Sebastian Rettenberger, for all the technical support.

I also want to say thanks to my advisors of previous works during this study for all the things I have learned from them, namely Gerrit Buse, who has been like a mentor for me over several years, and also Philipp Neumann.

Thanks as well to all the lecturers and tutors for without them this study would also not have been possible.

Of course, thanks go to all of my friends as well—without you these years would have been rather dreary!

I am also indebted to my family and especially my parents for all the support over my whole lifetime.

Finally, i will give thanks to the one who is the source of all good, my creator, lord and savior,

For I am not ashamed of the gospel of Christ:
for it is the power of God unto salvation
to every one that believeth.
— Romans 1,16

Abstract

This thesis is part of a project with the Q&A website *gutefrage.net*, two main goals of this project that will be addressed here are an online clustering of the questions and the development of a new recommender system. Clustering as well as the implementation of recommender systems are subject of active research, especially in the context of "Big Data". Currently, there are over 16 million questions growing by an amount of about eight thousand each day. Thus, it is not feasible to reprocess the whole data set all the time, processing the new questions has to build up on the already processed questions. An algorithm to do so has been developed before, where the questions are processed batch-wise and transformed into a global feature space. The actual clustering is based on density estimation, identifying clusters by regions of high density in the feature space. Sparse grids are used to estimate the density of a batch and to update the global density estimation accordingly. While this approach is efficient, the analysis will show that the quality of the clustering is not quite acceptable yet, with the main problem being that there is actually no relation between questions in different batches. We present a solution to this issue that is able to somehow relate the batches using the clustering of the previously processed batches to add virtual questions with fixed positions. We provide also an improved dissimilarity measure for the questions, especially for the question titles, which is needed for the feature space transformation. Furthermore, we extend the flat clustering to an hierarchical clustering algorithm that is able to detect more clusters and subdivide larger clusters at deeper levels. Opposed to the flat clustering algorithm, it is also independent from the choice of a high density threshold. The clustering is then used to implement the recommender system, where we already get acceptable recommendations for moderately large data sets.

Contents

1 Introduction

"Give a man a fish and you feed him for a day. Teach a man to fish and you feed him for a lifetime." This proverb states that it is more desirable to teach somebody how to achieve something than just giving the plain result. The very same principle applied to programming computers is the basic idea of machine learning. Especially as the available data grows with every day, we need methods that are able to extract important information from data sets in an at least partially automated way, which is what data mining and especially clustering is about. The handling of "Big Data" thereby is still a demanding task and subject of active research.

This thesis is part of a project with the Q&A website *gutefrage.net*, where we have over 16 million questions, growing every day by about eight thousand. Two main goals are to cluster the questions in an efficient way and to develop a new recommender system. From the first one, we especially hope to get insight into currently important topics, the latter one should help users to find what they are searching for.

From a research perspective, the methods developed for the clustering as well as for the recommender system could of course be applied for similar problems as well. We especially analyze the clustering approach taken so far here and discuss several improvements. The development of the recommender system builds up on the clustering.

We start with introducing several methods that are used for the clustering here and put them into a more general context in Chapter 2. In particular, density-based clustering methods are discussed, as the clustering algorithm developed here builds up on density estimation as well. Furthermore, we use sparse grids for the density estimation and thus, the sparse grid technique is also introduced, which is especially useful for high-dimensional problems. Finally, an overview of recommender systems in general is given.

The single steps of the clustering algorithm are then described in Chapter 3. We analyze the state of the clustering before this thesis and discuss several improvements.

How the clustering can be used for the recommender system is discussed in Chapter 4. We also give a short outlook how the recommender system could be extended in the future, using information beyond the clustering.

The results for different data sets, comparing different approaches and differ-

ent parameter choices for the clustering as well as for the recommender system, are discussed in Chapter 5. We especially evaluate what the different improvements brought us and where further improvements are needed for the clustering.

We finish with a summary and an outlook in Chapter 6, providing particularly some ideas for further investigation.

2 Machine Learning

According to [29], "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with Experience E." In this thesis, we basically face two machine learning tasks: clustering a large amount of questions and use this clustering to compute recommendations. To do so, we use several machine learning techniques that are introduced in this chapter. We start with a short introduction of data mining and go on with the more specific task of clustering. Next, we give an introduction to density estimation and show how it can be used for clustering. Afterwards, we introduce the sparse grids technique and show how it can be used in the context of data mining. We finish this chapter with an introduction to recommender systems.

2.1 Data Mining

The term data mining is not used consistently in the literature. Sometimes it is synonymously used with the term knowledge discovery in databases (KDD), which according to [16] "is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data." But e.g. in [14], data mining is seen as "a particular step in this process" and defined a bit more specific as "the application of specific algorithms for extracting patterns from data." The steps in the KDD process in [14] are:

- Data Selection
- Preprocessing
- Transformation
- Data Mining
- Interpretation/Evaluation

We focus on this particular sub-step here. Prediction and description are commonly named as the two main goals of data mining, the standard tasks are usually grouped into classification, regression and clustering [10, 14, 19]. In classification as well as in regression we learn a function from a training data set, i.e.

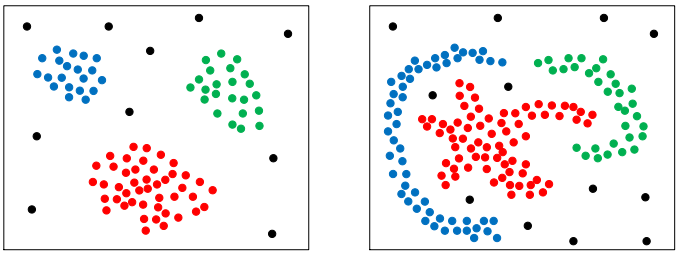


Figure 2.1: An example of three convex clusters on the left and three non-convex clusters on the right, marked by the colors red, blue and green. The black points are noise.

data items for which the function values are known. Thus, this type of learning is also called supervised. In the case of classification the learned function maps every data item to predefined classes, in the case of regression we have a real-valued function. Both tasks are mainly predictive. In contrast, clustering is an unsupervised, descriptive task. As this is one main task in this thesis, we discuss it more detailed in the next two sections.

2.2 Clustering

In clustering, we try to group data items into so-called clusters, see Fig. 2.1 for two simple examples. The traditional approach for clustering is to minimize the similarity of the objects inside a cluster and to maximize the dissimilarity of objects in different clusters [16, 24]. Thus, we need some kind of dissimilarity measure, which is mostly the euclidean distance. Most clustering methods lead to mutually exclusive or overlapping clusters or to a hierarchical representation.

In the first case, the classic approach is to use partitioning methods such as k-means or k-medians, where k is the pre-specified number of clusters [2, 24]. The main drawback of these methods is that they are not able to identify non-convex clusters, see the right example in Fig. 2.1. Several methods have been introduced to handle this problem, such as kernel methods and spectral clustering. The basic idea of kernel methods is to map the data into a higher-dimensional feature space, where a linear partition leads to a non-linear partition in the original space. Quite often, kernel methods are generalizations of the classic partitioning methods, e.g. kernel k-means. Spectral methods build up on an undirected graph representation, where the nodes represent the data items and the potentially weighted edges represent their relationships. Clustering the data can then be reduced to graph partitioning problems, usually based on

variations of the min-cut problem. Constructing the graph usually depends on a pair-wise similarity or dissimilarity measure between the data items. Popular choices are [26]:

- ϵ-neighborhood graph: Connect nodes with dissimilarity less than ϵ, usually unweighted.
- k-nearest neighbor graph: Connect each node to its k nearest neighbors, i.e. nodes with smallest dissimilarity or largest similarity and ignore the directions to get an undirected graph. Weight the edges by their corresponding similarity.
- fully connected graph: Connect all nodes and weight edges with corresponding similarity.

For a more detailed introduction and an overview of different clustering methods see e.g. [2, 12, 15, 26].

In the case of overlapping clusters at one level, fuzzy clustering algorithms are used, which are mostly based on a fuzzification of the classic clustering algorithms, see [7] for an overview.

Hierarchical clustering partitions the data at several levels, i.e. clusters can recursively be divided into sub-clusters. There are two main approaches for hierarchical clustering, resulting in so-called dendrograms. Both are iterative methods based on a dissimilarity measure between the current set of clusters [2, 17]:

- Agglomerative: Use a bottom-up approach. We typically start with all data points as clusters and merge them iteratively together, resulting in a binary-tree structure. Typical choices to merge the clusters are single-linkage, all-pairs linkage, centroid linkage and sampled linkage.
- Divisive: Use a top-down approach. We typically start with all data points as one cluster and can use any flat clustering algorithm in each iteration, resulting in an arbitrary tree-structure.

In this work, we will use a top-down approach which is different to the classic divisive algorithms, as data items can disappear at some level, see Section 3.3. The flat clustering algorithm is based on density estimation, we discuss this idea in the next section.

2.3 Clustering with Density Estimation

Before discussing the idea of density-based clustering, let's take a brief look at density estimation in general. Given a random variable X and a sequence of

observations $X_1,\ldots,X_N$ we want to estimate its probability density function f. If we already know—or make an assumption about—the distribution of X, only the parameters of the density function have to be estimated, e.g. the mean μ and the variance σ^2 in the case of the normal distribution. This is called parametric density estimation. A classic approach to estimate the parameters is to use the maximum likelihood estimator, see e.g. [3]. Non-parametric density estimation makes no assumption about the underlying distribution and is thus more flexible. Classic approaches are the usage of histograms, kernel estimators or grid-based methods. We take a brief look at a grid based method introduced in [20] as this is the underlying method used in this work, for more details and further methods see e.g. [3, 22]. Starting with an overfitted initial guess of the density f_ϵ based on observations $X_1,\ldots,X_N$,

$$f_\epsilon = \frac{1}{N}\sum_{i=1}^{N} \delta_{X_i}, \tag{2.1}$$

with the Dirac delta function δ_{X_i} centered on X_i, we seek to find $\hat{f}$, such that:

$$\hat{f} = \underset{u\in\mathcal{V}}{\arg\min} \int_\Omega \left(u(x) - f_\epsilon(x)\right)^2 dx + \lambda \int_\Omega \left(u''(x)\right)^2 dx, \tag{2.2}$$

where $\mathcal{V}$ is a functional space and λ is the regularization parameter. The left term ensures that the solution fits the data while the right term smooths it. We can transform this to the variational equation,

$$\int_\Omega (\hat{f}(x)s(x) - f_\epsilon(x))^2 dx + \lambda \int_\Omega (\hat{f}''(x)s''(x))^2 dx = \frac{1}{N}\sum_{i=1}^{N} s(X_i), \tag{2.3}$$

for all s in the test function space $\widetilde{\mathcal{V}}$, see [20]. To solve this problem we have to choose an appropriate ansatz and test function space. In this work, we take the Ritz-Galerkin approach and choose the sparse grid space as ansatz and test function space, based on [30]. We will discuss the sparse grid technique in the next Section 2.4. The variational Equation 2.3 then leads to a system of linear equations, that is solved using the conjugate gradient method.

Now, the idea of density-based clustering as introduced in [13] is that "within each cluster we have a typical density of points which is considerably higher than outside of the cluster". What remains is to formalize what a high density region is. One approach can be found in [13], leading to the famous DBSCAN algorithm. We will follow the approach described in [30], see Section 3.1.

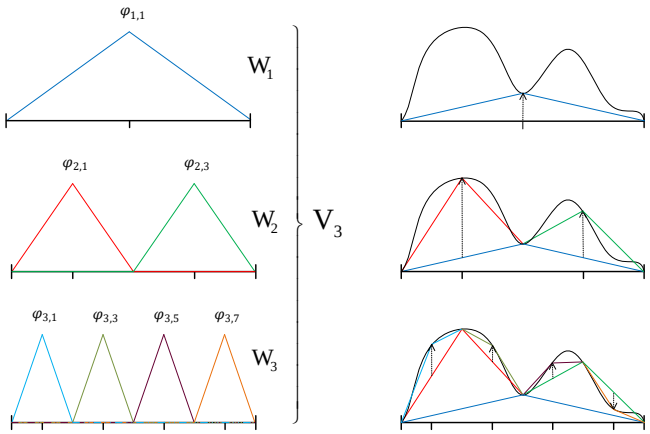


Figure 2.2: The linear hat basis functions in 1-D up to level 3 (left) and how they are used for interpolation (right). The arrows indicate the values of the surpluses for the current level.

2.4 Data Mining with Sparse Grids

Sparse grids have been successfully applied to solve the classic data mining tasks classification and regression, see e.g. [18, 32, 33], as well as for density-based clustering in [30]. Sparse grids in general are a discretization technique that try to overcome the curse of dimensionality, i.e. the exponential dependency of the number of grid points on the number of dimensions. In the context of classification and regression they are used to represent the function to learn, while in clustering we will use them to represent the density function. Thus, let's look at how a standard interpolation problem is solved using sparse grids. We follow the notation of [33].

First, consider the one-dimensional case. Given a function $f : \Omega \to \mathbb{R}$, assume for simplicity $\Omega = [0, 1]$, we seek to find an interpolant u, such that:

$$u(x) \approx f(x). \tag{2.4}$$

Typically, u is the weighted sum of some basis functions, as it is the case for sparse grids. Sparse grids use a hierarchical basis, we identify each basis function $\varphi_{l,i}$ by its level l and index $i \in I_l$,

$$I_l = \{2k - 1, k = 1, \ldots, 2^{l-1}\}. \tag{2.5}$$

We will use the linear hat basis function,

$$\varphi_{l,i}(x) = \max(1 - |2^l x - i|, 0). \tag{2.6}$$

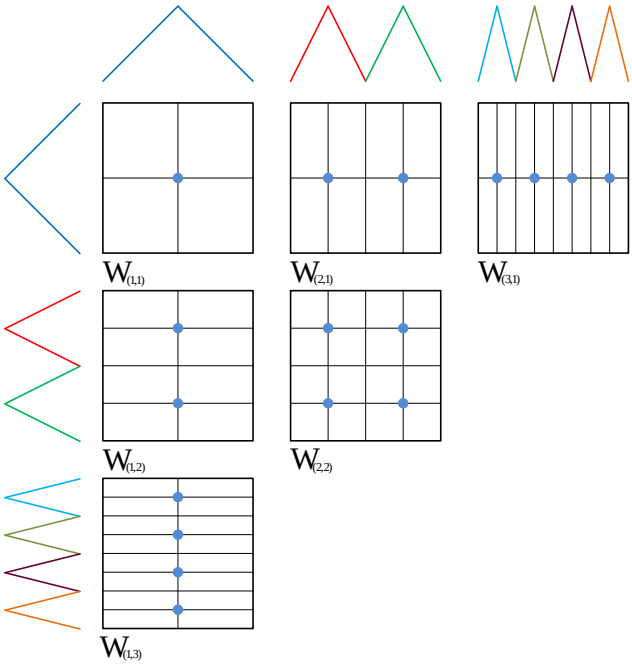


Figure 2.3: Subspace-tableau of the 2-D sparse grid space V_3 showing all grid points and indicating the tensor product construction for the 2-D basis functions from 1-D hat basis functions.

The subspace W_l is then defined by,

$$W_l = \text{span}\{\varphi_{l,i}, i \in I_l\}. \tag{2.7}$$

The sparse grid space V_n is then the direct sum of its subspaces,

$$V_n = \bigoplus_{l \leq n} W_l. \tag{2.8}$$

The regular sparse grid interpolant $u_n \in V_n$ of level n with surpluses $\alpha_{l,i}$ is thus,

$$u_n(x) = \sum_{l=1}^{n} \sum_{i \in I_l} \alpha_{l,i} \varphi_{l,i}(x). \tag{2.9}$$

The hierarchical basis functions and the sparse grid interpolation are illustrated in Fig. 2.2.

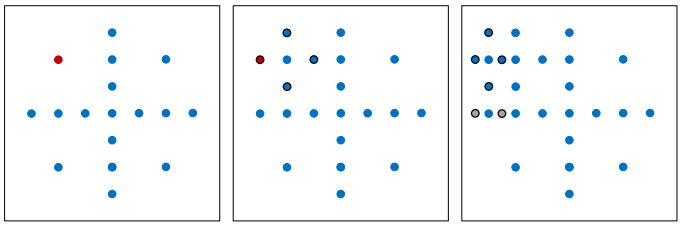


Figure 2.4: Starting with the 2-D sparse grid on the left, we obtain the sparse grids in the middle and on the right by refining the red grid points, i.e. inserting all children in the hierarchical basis. The newly inserted grid points are black-rimmed. The grey grid points are the hierarchical ancestors that will be inserted as well. Based on [33].

For the d-dimensional case, we extend the subspaces $W_{\vec{l}}$ and the basis functions $\varphi_{\vec{l},\vec{i}}$, with level vector $\vec{l} = (l_1, \ldots, l_d)$ and index vector $\vec{i} = (i_1, \ldots, i_d)$, using a tensor product approach,

$$\varphi_{\vec{l},\vec{i}}(x) = \prod_{j=1}^{d} \varphi_{l_j,i_j}(x), \tag{2.10}$$

$$W_{\vec{l}} = \bigotimes_{j=1}^{d} W_{l_j} = \text{span}\,\{\varphi_{\vec{l},\vec{i}}, i_j \in I_{l_j} \forall j = 1, \ldots, d\}. \tag{2.11}$$

The d-dimensional sparse grid space is then derived by,

$$V_n = \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}, \tag{2.12}$$

see Fig. 2.3.

For many problems it is desirable to have adaptivity. Sparse grids come with a quite natural adaptivity criterion, as higher surpluses indicate higher errors. We will use spatially adaptive sparse grids, i.e. the only restriction for inserting a grid point is that its hierarchical ancestors are inserted as well—as most sparse grid algorithms require this. See Fig. 2.4 for a simple refinement example.

2.5 Recommender Systems

The general goal of recommender systems is to propose data items to users which could be interesting for them as well. The computation of the recommendations will be based on informations we have about the users and/or the data items.

For example, a user who liked some book B could be interested in books of the same author or the same genre, but also in books liked by other users that liked B as well.

Implementations of recommender systems can be classified into personalized and non-personalized methods [31]. In the personalized setting for recommender systems we have a set of users U, a set of Items I and an utility or rating function $r : U \times S \to \mathbb{R}$. The recommender systems now tries to estimate the rating function and then recommends the N items with the highest rating for each user [1, 27]. Common approaches for the implementation of personalized recommender systems are [1, 6, 27]:

- Content-based: Compute recommendations based on the similarity to the content of items the user liked in the past. See [31] for an overview of mainly content-based recommender systems.
- Collaborative: Compute recommendations based on the similarity to other users. See [36] for an overview.
- Hybrid: Combine the content-based and collaborative approach. See [1] for an overview.

In the non-personalized settings, there are two main approaches [31]:

- Aggregated Opinion Approach: Rate the items based on ratings given by the users and recommend the items with best ratings to all users. The rating function is reduced to $r : I \to \mathbb{R}$.
- Basic Item Association Recommender: Compute recommendations based on the item or items (e.g. items in the cart for an online shop) the user is currently viewing. The rating function could be defined as $r : 2^I \times I \to \mathbb{R}$. Setting $U = 2^I$, this is obviously a special case of the rating function for the personalized setting. Thus, we could also see a set of items as a class of users.

In this thesis, we will discuss the implementation of a non-personalized recommender system that uses a combination of the two named approaches, see Chapter 4.

3 Clustering with Sparse Grid Density Estimation

The clustering algorithm we present in this chapter builds up on [35]. We first give a short summary of the general work flow of the algorithm described there and discuss several improvements in the following sections.

3.1 Work Flow of the Clustering Algorithm

For each question we have a title and assigned tags, this is the only data we use for the clustering. As discussed in Section 2.2, we need some kind of dissimilarity measure for the questions. As the algorithm is thought to deal with a large (over 16 million) and also growing number of questions, it is not feasible to compute this dissimilarity pair-wise for all questions. Also we do not want to reprocess the whole data set when new questions are added. Thus, the idea is to process the questions batch-wise, i.e. we only compute dissimilarities between questions in a batch of rather small size (e.g. 1000). The main steps for a generalized version of the algorithm described in [35] are:

1. Compute dissimilarities $\delta_{i,j}$ between the titles for all questions i, j in the current batch. Different measures are discussed in Section 3.2.1.

2. Compute weighted dissimilarities by scaling $\delta_{i,j}$ down if questions i and j have at least one common tag. Different approaches discussed in Section 3.2.2.

3. Optional: Compute all-pairs-shortest-path for the dissimilarities to obtain distances $d_{i,j}$.

4. Transform the questions using obtained distances or dissimilarities into the euclidean feature space via multi-dimensional scaling (MDS). Different approaches discussed in Section 3.4.

5. Estimate the density function of the feature space $\mathbb{F}$ via sparse grid density estimation for the current batch. Update the global density estimation accordingly.

6. Optional: Compute a clustering of the questions using a density-based clustering algorithm. Different approaches discussed in 3.3.

3.2 Dissimilarity Measures

As already mentioned, we only use the title and the tags of a question to compute pair-wise dissimilarities. In this section, we first discuss different dissimilarity measures for two titles. Afterwards, we look at two approaches to use the tags for the overall dissimilarity measure.

3.2.1 String Measures

One of the most common metrics for the dissimilarity of two strings is the Levenshtein distance, first proposed in [25]. It is also used for the clustering algorithm we build up on here in [35] to compute the distance of two question titles. The Levenshtein distance $l(s_1, s_2)$, $l : \Sigma^* \times \Sigma^* \to \mathbb{N}_0$ with alphabet Σ of two strings s_1, s_2 is the minimum number of insertions, deletions and substitutions needed to get from s_1 to s_2. Obviously, we have $l(s, s) = 0$, $l(s_1, s_2) = l(s_2, s_1)$ (given the minimum sequence of insertions, deletions and substitutions to get from s_1 to s_2, just reverse the sequence to get from s_2 to s_1 and the other way around) and the triangle inequality $l(s_1, s_3) \leq l(s_1, s_2) + l(s_2, s_3)$ (given the minimum sequences to get from s_1 to s_2 and from s_2 to s_3, the composition of both sequences leads to a sequence that transforms s_1 to s_2). Thus, the Levenshtein distance is a metric.

The main disadvantage when measuring the dissimilarity of two question titles via the Levenshtein distance is that it does not take the reordering of words into account. More formally, we can see a question title t as a string of the following form with words $w_1, \dots w_n$ and delimiter $d \notin \Sigma$:

$$t = w_1 d w_2 \dots d w_n. \tag{3.1}$$

Now, in the simplest case, assume we have titles $t_1 = w_1 d w_2$ and $t_2 = w_2 d w_1$ with $w_1 \neq w_2$, then $l(t_1, t_2) > 0$ but we would like to have a distance of 0. The new approach is to see a title as unordered set of words,

$$t = \{w_1, \dots, w_n\}. \tag{3.2}$$

To compute a distance of two titles, we compare the words pair-wise using the Levenshtein distance. Now we are also not interested in the exact distance but rather in the fact if two words have the same or a similar meaning or not. Taking into account typing errors, misspellings, different grammatical forms and so on, we try to recognize this using the Levenshtein distance relative to the word

Algorithm 1 The weighted pair-wise Levenshtein algorithm to compute a dissimilarity value for two question titles.

function weighted_pairwise_levenshtein(t_1, t_2)
    $sum \leftarrow 0$ ▷ weighted sum of matched words
    for w $\in t_1 \setminus \mathbb{S}$ **do** ▷ $\mathbb{S}$ is the set of stop words
        for v $\in t_2 \setminus \mathbb{S}$ **do**
            ▷ use Levenshtein distance $l(\cdot, \cdot)$ to recognize similar words
            if $l(\mathrm{w}, \mathrm{v}) \leq c \cdot \max(|\mathrm{w}|, |\mathrm{v}|)$ **then** ▷ $c = \frac{1}{4}$ in our case
                ▷ $\omega(\cdot)$ is the word weighting function
                $sum \leftarrow sum + \min(\omega(\mathrm{w}), \omega(\mathrm{v}))$
            end if
        end for
    end for
    $weightsum1 \leftarrow \sum_{\mathrm{w} \in t_1 \setminus \mathbb{S}} \omega(\mathrm{w})$
    $weightsum2 \leftarrow \sum_{\mathrm{w} \in t_2 \setminus \mathbb{S}} \omega(\mathrm{w})$
    $similarity \leftarrow \min\left(1, \frac{sum}{\max(weightsum1, weightsum2)}\right)$
    if $similarity = 0$ **then**
        ▷ no common words
        return 1
    else
        ▷ at least one common word not in $\mathbb{S}$
        ▷ transform dissimilarity with parameters w_{sim}, e_d
        return $w_{sim} \cdot (1 - similarity)^{e_d}$
    end if
end function

length of the longer word. To get a similarity value for the titles, we count the number of similar words and set them relative to the larger number of words of the two titles currently compared. In practice, this almost always leads to a similarity value in $[0, 1]$, only if words are matched multiple times we could have similarity values greater 1. In these cases, we just set the similarity value s to 1 and thus, we can use $1 - s$ as dissimilarity value. To achieve that similar questions have a really small dissimilarity value, we take this dissimilarity value to the power of some constant parameter $e_d > 1$ (as dissimilarity values close to 0 are very rarely otherwise). For a somehow clear distinction between questions that have nothing in common (i.e. they have dissimilarity value 1) and questions that have at least one common word, we also scale the latter by some constant factor w_{sim}.

Another obvious observation is that in most question titles there are many meaningless words like articles, prepositions and so on. They should not contribute to the dissimilarity measure and thus, we exclude them using a set of stop words $\mathbb{S}$ (downloaded from [9]). Going further, there are typically words in a data set with significant higher frequency than others. Thus, it can be useful to weight the words according to their frequency in the data set. We discuss different choices for the weighting function $\omega : \Sigma^* \to [0, 1]$ in Section 5.1.1. Putting things together, you can find a pseudo code for the whole procedure in Algorithm 1. Note, that the dissimilarity measure defined by this algorithm does not define a metric, as the triangle inequality can be violated. But at least for the classical MDS approach, the assumption is that the dissimilarities are distances, see Section 3.4.1. We can compute distances from the dissimilarities using the Floyd-Warshall algorithm. As this is only done for the questions in a batch and also the algorithm can be easily parallelized and scales very well, the complexity of $\mathcal{O}(N^3)$ with batch size N is not critical. Also, note that the weighted dissimilarities obtained by scaling the dissimilarities down using the tags are not necessarily distances either, for both approaches that are discussed in the next section.

3.2.2 Usage of Tags

The approach to use the tags suggested in [35] is via the Laplacian L of the undirected graph $G = (V, E)$ where the nodes $V = \{q_1, \ldots, q_N\}$ represent the questions and $\{q_i, q_j\} \in E$ if and only if questions q_i and q_j share at least one common tag. The elements of the Laplacian $L = (l_{i,j})_{1 \leq i,j \leq N}$ are given by:

$$l_{i,j} = \begin{cases} deg(q_i) & \text{if } i = j \\ -1 & \text{if } \{q_i, q_j\} \in E \\ 0 & \text{otherwise} \end{cases}. \tag{3.3}$$

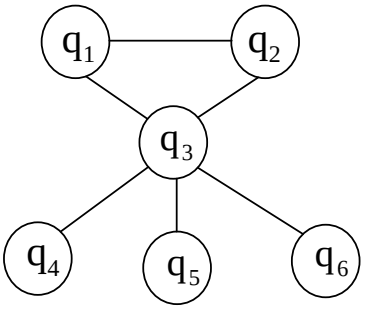


Figure 3.1: Example of the undirected graph representation of five questions where an edge between two questions indicates that they share at least one common tag.

The graph-weighted dissimilarities $\delta^L_{i,j}$ are then obtained by:

$$\delta^L_{i,j} = \begin{cases} \frac{\delta_{i,j}}{l_{i,i}+l_{j,j}} & \text{if } l_{i,j} = -1 \\ \delta_{i,j} & \text{otherwise} \end{cases}. \tag{3.4}$$

Consider the example shown in Fig. 3.1 and assume questions q_1, q_2, q_3 share pair-wise exactly one common tag. As $deg(q_1) = deg(q_2) = 2$ and $deg(q_3) = 5$, we would scale $\delta_{1,3}$ and $\delta_{2,3}$ by a factor of $\frac{1}{2+5} = \frac{1}{7}$, but at the same time we would scale $\delta_{1,2}$ only by a factor of $\frac{1}{2+2} = \frac{1}{4}$, although all three pairs share exactly the same number of tags—which seems a bit inconsistent. To overcome this issue, we use the adjacency matrix of the weighted undirected graph $\hat{G} = (V, E, \omega)$ with the weighting function $\omega : E \to \mathbb{N}$ here,

$$\omega(\{q, \hat{q}\}) = |T \cap \hat{T}|, \tag{3.5}$$

where T, $\hat{T}$ is the set of tags of q, $\hat{q}$, respectively. The weighted dissimilarities are then computed by:

$$\delta^A_{i,j} = \begin{cases} \frac{\delta_{i,j}}{1+\omega(\{q_i,q_j\})} & \text{if } \{q_i, q_j\} \in E \\ \delta_{i,j} & \text{otherwise} \end{cases}. \tag{3.6}$$

3.3 Hierarchical Clustering

In this section, we describe the extension of the flat density-based clustering algorithm that is used in [35] and was first described in [30] to a hierarchical clustering algorithm. But let's first summarize the main steps of the flat algorithm:

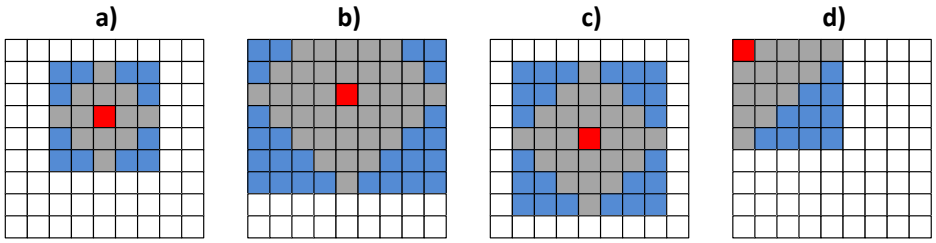


Figure 3.2: Samples for a 9×9 Cartesian grid, each square represents a sample. We want to find the k-nearest neighbors for the red square. The gray squares depict the samples for which we know that the distance is $\leq \Delta$ which are at least $\frac{\Delta^2+3\Delta}{2}$ (equality for example **d)**), where $\Delta = 2$ in **a)**, $\Delta = 4$ in **b)**, **d)** and $\Delta = 3$ in **c)**. Together with the blue squares, they form the limited search space, including all squares with distance at most Δ. Choosing $\Delta = 2$ would be sufficient for $k \leq 5$, $\Delta = 3$ would be sufficient for $k \leq 9$ and $\Delta = 4$ would be sufficient for $k \leq 14$.

1. Sample over the feature space $\mathbb{F}$ and compute the density $p(s_i)$ using the sparse grid density estimation for each sample s_i, $1 \leq i \leq n$.

2. Compute the undirected k-nearest neighbor graph $G_k = (S, E_k)$, with
   - $S = \{s_1, \ldots, s_n\}$,
   - $\{s_i, s_j\} \in E_k$ if and only if $s_i \in \mathcal{N}_k(S, s_j)$ or $s_j \in \mathcal{N}_k(S, s_i)$, where $\mathcal{N}_k(S, s_i)$ denotes the set of k-nearest neighbors of s_i in the set S according to the euclidean distance.

3. Remove the samples with density below a high density threshold ϵ, i.e. compute the subgraph $\hat{G}_{k,\epsilon} = (\hat{S}_\epsilon, \hat{E}_k)$, with
   - $\tilde{S}_\epsilon = \{s_i, p(s_i) < \epsilon\}$,
   - $\tilde{E}_{k,\epsilon} = \{\{s_i, s_j\} \in E_k, s_i \in \tilde{S}_\epsilon \text{ or } s_j \in \tilde{S}_\epsilon\}$,
   - $\hat{S}_\epsilon = S \setminus \tilde{S}_\epsilon$, $\hat{E}_{k,\epsilon} = E_k \setminus \tilde{E}_{k,\epsilon}$.

4. Compute the connected components $\hat{S}_1, \ldots, \hat{S}_m$ in $\hat{G}_{k,\epsilon}$ that represent the clusters and label them with the label function $l : \hat{S} \to \mathbb{N}$, $\forall i = 1, \ldots, m, \forall s \in S_i : \; l(s) = i$.

The straightforward approach to compute the k-nearest neighbor graph G_k would be to compute the euclidean distances $\|s_i - s_j\|_2$ between samples s_i,

s_j pair-wise and then choose the k smallest distances. This leads to a complexity of $\Omega(d|S|^2)$. For a large number of samples or when we repeat the routine frequently, as we will do later for the constrained MDS for every batch, this can become inefficient. As we use a Cartesian grid with equal step-size for each dimension as samples, we can improve the complexity to $\mathcal{O}(dk|S|)$. Note, that $|E_k| = \Omega(k|S|)$ and the number of dimensions d is small. The idea is to limit the search space to size $\mathcal{O}(k)$ for each sample. For simplicity, we only consider the two-dimensional case here and assume that the Cartesian grid has step-size 1. Note, that it is possible to extend the approach to the d-dimensional case using the same idea. For an arbitrary sample, we have at least $\frac{\Delta^2+3\Delta}{2}$ other samples with euclidean distance at most $\Delta \in \mathbb{N}$, see Fig. 3.2. Further, we can limit the number of samples with a distance smaller than Δ by $\mathcal{O}(\Delta^2)$, also shown in Fig. 3.2. Thus, if we choose Δ such that,

$$\begin{aligned} & \frac{\Delta^2 + 3\Delta}{2} \geq k \\ \Rightarrow \quad & \Delta = \left\lceil \frac{-3}{2} + \sqrt{\frac{9}{4} + 2k} \right\rceil, \end{aligned} \tag{3.7}$$

we only have to compute the distance to $\mathcal{O}(\Delta^2) = \mathcal{O}(k)$ samples. What remains is to compute the k samples with smallest distance. If we sort the samples in the search space according to their distance, we would get an overall complexity of $\mathcal{O}(kd|S| + k\log(k)|S|)$, which would be already fine as k is rather small. If we want to avoid the additional summand of $\mathcal{O}(k\log(k)|S|)$, we can pre-compute the distances for the maximum search space of size $(2\Delta+1)^2-1$. Then, we would sort the d-dimensional array according to the distances and store the difference of the indices for each dimension. We can then just go through the sorted array for each sample mapping the index-differences to actual indices and skip the samples if they do not exist for the current sample. The overall complexity is then in $\mathcal{O}(dk|S|)$.

What remains is to map the actual questions to the clusters defined by the samples. The approach we take here is to map each question to the cluster assigned to the sample with the smallest distance that was not removed. Formally, the function $m : Q_{\mathbb{F}} \to \mathbb{N}$ that maps the feature representations of the questions $Q_{\mathbb{F}}$ to cluster labels with the label function l is defined as,

$$m(q) = l(\min(\arg\min_{s \in \hat{S}}(\|q - s\|_2))). \tag{3.8}$$

Note, that the min is just for the case where there are several samples with minimum distance, which rarely happens in practice.

One obvious drawback of the flat clustering algorithm and also for many other methods based on density estimation is that we have to choose a high density

threshold—and what a good choice is depends on the actual data set. Approaches to avoid an explicit choice of a high density threshold are e.g. discussed in [5, 28] and e.g. implemented in the statistical software **R**, see [4]. The main idea that is used there and that we also employ here is to discover all clusters that are extracted by some threshold. Based on this idea, we iterate over increasing density thresholds in the range $[\min_{1\leq i\leq n}(p(s_i)), \max_{1\leq i\leq n}(p(s_i))]$ using a step-size ϵ_{step} determined by a given number of steps n_{steps}. At first glance, this could sound like replacing one parameter by another, but for sufficient large number of steps, the resulting cluster tree should stay more or less the same. For each density threshold, we run the flat clustering algorithm and check if there are any new clusters split up by this density threshold. The new clusters are then added as children to the cluster they belonged to before. In the context of clustering questions, a cluster should represent some topic and its child clusters should represent more specific topics, e.g. if the topic of the parent cluster is music, then its child clusters could have topics like baroque music, classical music, rock music and so on. Now if we always keep the cluster hierarchy in the way described before, this will probably not be the case. There could be clusters that have the same parent but have (nearly) nothing in common, e.g. they could be connected by some noise for rather low density thresholds. In this case, we want to delete the parent cluster and move the children up. If a cluster is split up by threshold ϵ, we try to detect this using the last graph $\hat{G}_{k,\epsilon-\epsilon_{step}}$ that connected the child clusters. A pseudo code sketch for the split routine is given in Algorithm 2, the underlying idea is illustrated in Fig. 3.4. The whole hierarchical clustering procedure is sketched in Algorithm 3 and illustrated in Fig. 3.3.

3.4 Multidimensional Scaling

One main step when processing a batch is to transform the questions into the euclidean feature space $\mathbb{F}$. The approach taken in [35] is to use classical multidimensional scaling (MDS) and then transform the obtained solution into the feature space by translation and scaling it accordingly. The nice thing about this is that it has an analytical solution, the problem however is that when applied only to questions of a single batch there is no relation between the batches. Thus, questions from different batches that are similar are mapped to locations with small distances just by chance. This is why we will add constraints that relate the already processed questions to the questions in the current batch. But to understand this approach, we give a short introduction to classical MDS, a more elaborate introduction to MDS and its variations can be found in [8].

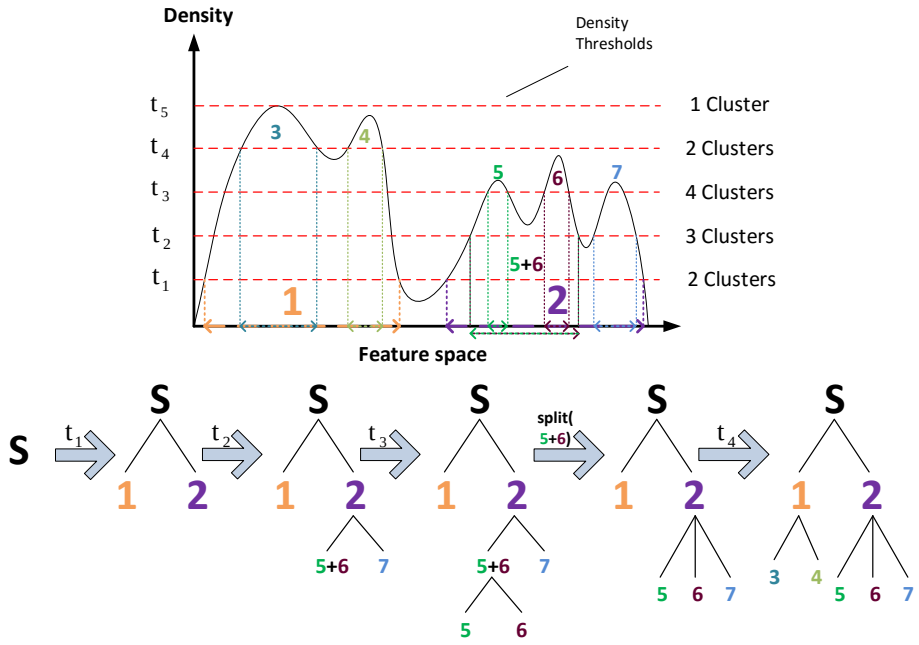


Figure 3.3: Illustration of the hierarchical clustering algorithm for an one-dimensional feature space with 5 steps. Starting with the samples S as one cluster, the development of the cluster tree is shown from the left to the right using the density thresholds $t_1, \ldots, t_5$. Note, that the clusters that would be the only child of their parent are not shown, as they are removed in a first post-processing step for the cluster tree. After using the threshold t_3, we move the clusters 5 and 6 up. For the threshold t_5 nothing changes. With a single threshold, at most 4 clusters could be detected. Every threshold is labeled with the number of clusters it detects.

Algorithm 2 Algorithm to decide if child clusters should be moved up.

function split_child(parent, child, $G = (V, E)$)
    $V_p \leftarrow$ parent.nodes() ▷ including child nodes
    $V_c \leftarrow$ child.nodes()
    connections_parent $\leftarrow |\{e = \{i, j\} \in E, i, j \in V_p\}|$
    max_connections_parent $\leftarrow \frac{1}{2}|V_p|(|V_p| - 1)$
    connectivity_parent $\leftarrow \frac{\text{connections\_parent}}{\text{max\_connections\_parent}}$
    connections_child_parent $\leftarrow |\{e = \{i, j\} \in E, i \in V_c, j \in V_p\}|$
    max_connections_child_parent $\leftarrow |V_c|(|V_p| - |V_c|)$
    connectivity_child_parent $\leftarrow \frac{\text{connections\_child\_parent}}{\text{max\_connections\_child\_parent}}$
    return $\frac{\text{connectivity\_child\_parent}}{\text{connectivity\_parent}} \leq t$ ▷ t is the split threshold parameter
end function
function split(parent,$G = (V, E)$)
    for child in parent.child_clusters() **do**
        if split_child(parent, child,G) **then**
            return True
        end if
    end for
    return False
end function

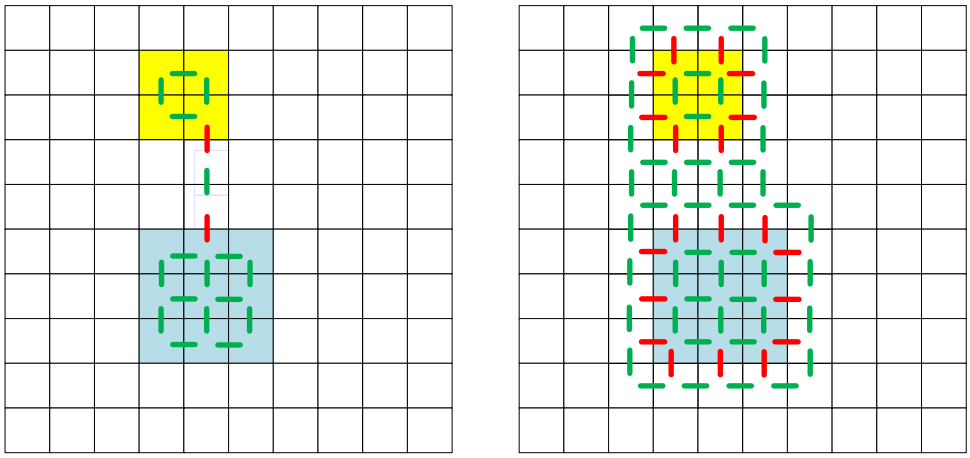


Figure 3.4: Two examples for the split algorithm in two dimensions with 10×10 samples and $k = 4$. For both examples, the yellow and the blue samples depict two clusters that are detected by some density threshold ϵ, the parent nodes of interest are the union of both plus the red patterned samples (samples belonging to the parent cluster with density less than ϵ, but greater than $\epsilon - \epsilon_{step}$). The bold red lines are the connections between the child and parent clusters, together with the green lines they form the connections inside the parent. In the example on the left, there are 19 connections inside the parent and the parent has 15 nodes. Thus, the connectivity of the parent is $\frac{19}{\frac{15 \cdot 14}{2}}$. There is only one child-parent connection for the yellow and blue cluster, respectively. The yellow cluster has 4 nodes and thus, its connectivity to the parent is $\frac{1}{4 \cdot (15-4)}$. The blue cluster has 9 nodes and thus, its connectivity to the parent is $\frac{1}{9 \cdot 6}$. The connectivity-ratio is then ≈ 0.126 for the yellow cluster and ≈ 0.102 for the blue cluster. Choosing e.g. the split threshold $t = 0.2$, we would delete the parent and move the children up. In the right example, there are many more child-parent connections, leading to connectivity ratios ≈ 0.652 for the yellow cluster and ≈ 0.502 for the blue cluster. Thus, choosing the same split threshold $t = 0.2$, we would not move up the child clusters.

Algorithm 3 The hierarchical clustering algorithm using the flat k-nearest neighbor algorithm for each investigated density threshold.

```
1: function cluster_hierarchical(S, k, n_steps)          ▷ S is the set of samples
2:     p_min ← min_{s∈S}(p(s))          ▷ p(·) is the sparse grid density estimation
3:     p_max ← max_{s∈S}(p(s))
4:     ε_step ← (p_max − p_min) / n_steps
5:     compute G_k
6:     G_{k,p_min} ← G_k
7:     for i = 1 to n_steps do
8:         ε ← p_min + i · ε_step
9:         compute G_{k,ε} and discover connected components C_1, ..., C_m
10:        updated ← ∅
11:        for j = 1 to m do
12:            select c ∈ C_j                          ▷ Choice of c does not matter
13:            parent ← c.get_cluster()                ▷ most specific cluster of c
14:            parent.add_child(C_j)
15:            updated ← updated ∪ {parent}
16:        end for
17:        for parent ∈ updated do
18:            if parent.num_children > 1 then
19:                if split(parent, G_{k,ε−ε_step}) then
20:                    ▷ move child clusters up and delete parent
21:                    parent.get_parent().add_children(parent.get_children())
22:                    parent.get_parent().remove_child(parent)
23:                end if
24:            else
25:                parent.remove_children()
26:            end if
27:        end for
28:    end for
29: end function
```

3.4.1 Classical MDS

The main idea of MDS is to embed N objects in a space of given dimensionality d such that their pair-wise dissimilarities $\delta_{i,j}$ are represented as close as possible. For simplicity, we use $\mathbb{R}^d$. The dissimilarities should be non-negative, symmetric and the dissimilarity of an object to itself should be zero. In classical MDS, the additional assumption is that they also fulfill the triangle inequality. Thus, they represent a metric. In classical MDS, we minimize the loss function $L : \mathbb{R}^{d \times N} \to \mathbb{R}$,

$$L\left(X = (x_1, \ldots, x_N)\right) = \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\|x_i - x_j\|_2 - d_{i,j}\right)^2. \tag{3.9}$$

The analytic solution $\hat{X}$ can be computed by [8]:

1. Compute $D^2 = (d_{i,j}^2)_{1 \leq i,j \leq N}$
2. Apply double centering to D^2:
   - $B_D = -\frac{1}{2} J D^2 J$, with
   - $J = I - N^{-1} \mathbb{1}\mathbb{1}^T$, where I is the identity matrix and $\mathbb{1}$ is the column vector of ones.
3. Compute the eigendecomposition $B_D = Q \Lambda Q^T$
4. Let Λ_+ denote the matrix containing the first d eigenvalues greater than zero and Q_+ is the matrix with the first d columns of Q. Then the solution is given by $\hat{X} = Q_+ \Lambda_+^{\frac{1}{2}}$.

To obtain the features for the questions, what remains is to transform the solution into the feature space (done by translation and scaling it accordingly).

3.4.2 Constrained MDS

As mentioned, the main problem when using classical MDS as described to transform the questions into the feature space is that we have no relation between the batches. In the constrained MDS approach we describe now, the basic idea to relate the current batch with the batches processed before is to add virtual questions, minimizing basically the same loss function and fix the position of the virtual questions with additional constraints. For the virtual questions we have a virtual title and virtual tags, just as for the real questions. To compute the virtual questions we use the clusters from the previous batches. The virtual title will then consist of the keywords in the cluster, i.e. the words which occur most

frequently in the question titles in the cluster. Similar like in the dissimilarity computation, we exclude words of the stop word set and weight them according to their global frequency. Also, we require that the word frequency in the cluster at least doubles the global frequency. The same is done to compute the virtual tags (except that we do not exclude any tag a priori).

The virtual questions should, in the best case, represent all questions in a cluster. We try to achieve this to some extent by computing as many keywords such that the sum of their weighted frequencies (i.e. occurrences of the word multiplied by its weight and divided by the number of questions in the cluster) exceeds a given threshold ϵ_{freq}. If this threshold is not exceeded for a given maximum number of keywords K_{max}, this cluster is not used to add a virtual question. At the same time, even if the threshold is already exceeded, we compute at least a minimum number of keywords K_{min}—whenever possible. If a virtual question is added, exactly K_{min} virtual tags are computed (or less if there are not enough tags that have at least double the global frequency).

To compute the dissimilarity between a virtual and a real question, we use the same measure as for the real questions, with the slight variation that we divide the dissimilarity by the number of words in the virtual title, as we also want small dissimilarities between real and virtual questions if only one keyword of the cluster matches.

The main assumption in this approach is that clusters can be represented by a rather small number of keywords. Using the pair-wise Levenshtein approach, the majority of pairs of questions will have the maximum dissimilarity value of 1—which is also not surprising as there is really no direct relation between most questions. Thus, our assumption will be violated for most of the batches we get in practice. Our approach to overcome this problem is to remove questions in a batch that have nearly no relations to other questions in the batch and also no relation to already computed virtual questions. These questions could then potentially be added in a later batch again, for the experiments in Chapter 5 we add them once again and if they are removed again we remove them finally.

Formalizing and Solving the Constrained MDS Problem

We now formalize the constrained MDS approach and discuss how to solve it. Let N denote the current batch size and M the number of virtual questions added so far. For the virtual questions $v_1, \ldots, v_M$ with respective clusters $C_1, \ldots, C_M$ and the feature representations of the contained questions $Q_{\mathbb{F},1}, \ldots, Q_{\mathbb{F},M}$ we have positions $P = (p_1, \ldots, p_M) = (p_{i,j}) \in \mathbb{R}^{d \times M}$ that are given by the arithmetic

mean of the features,

$$p_m = \frac{1}{|Q_{\mathbb{F},m}|} \sum_{q_f \in Q_{\mathbb{F},m}} q_f,\ 1 \leq m \leq M. \tag{3.10}$$

With the dissimilarity matrix $\Delta_r = (\delta_{i,j}) \in \mathbb{R}^{N \times N}$ for the real questions and the dissimilarity matrix between real and virtual questions $\Delta_v = (\hat{\delta}_{i,m}) \in \mathbb{R}^{N \times M}$ and the distance matrix for the virtual questions $D_P = (\hat{d}_{i,j}) \in \mathbb{R}^{M \times M}$,

$$\hat{d}_{i,j} = \|p_i - p_j\|_2 , \tag{3.11}$$

we construct the overall dissimilarity matrix $\Delta = (\delta_{i,j}) \in \mathbb{R}^{(N+M)\times(N+M)}$,

$$\Delta = \begin{pmatrix} \Delta_r & \Delta_v \\ \Delta_v^T & D_P \end{pmatrix}. \tag{3.12}$$

Computing all-pairs-shortest-path for Δ, we get the distance matrix $D = (d_{i,j}) \in \mathbb{R}^{(N+M)\times(N+M)}$. While we stick to D in the following, it is also possible to use Δ instead, as we use a least squares solver. To be able to update the global sparse grid density estimation, we have to transform the questions always to the same feature space $\mathbb{F} = [a, b]^d$, which means that we have linear boundary constraints for the least squares solver. With the features of the real and virtual questions $X = (x_1, \ldots, x_{N+M}) \in \mathbb{F}^{N+M}$ we seek to minimize,

$$\min_{X \in \mathbb{F}^{N+M}} \left(\begin{array}{l} \sum_{i=1}^{N+M} \sum_{j=1}^{N+M} \left(d_{i,j} - \|x_i - x_j\|_2\right)^2 + \\ \sum_{m=1}^{M} \left(0 - \|p_m - x_{N+m}\|_2\right)^2 \end{array} \right), \tag{3.13}$$

where the upper term is the loss function of the classcial MDS for D and the lower term ensures that the transformed positions of the virtual questions do not differ too much from their original positions. Using this function directly, we would have $(N+M)^2 + M$ constraints and $d \cdot (N+M)$ variables. Thus, the Jacobi matrix, which is needed for the least squares solver, would have a size of $\Omega(d \cdot (N+M)^3)$. Already when $d = 2$ and $N + M = 1000$, this would lead to a size of over 8GB (single-precision), which makes this approach quite inefficient. To overcome this problem, we transform the minimization problem. The first step is to square all euclidean norms and the corresponding dissimilarities in the minimization problem, as this simplifies the computation of the Jacobian later

on,

$$\min_{X\in\mathbb{F}^{N+M}} \left(\sum_{i=1}^{N+M} \sum_{j=1}^{N+M} \left(d_{i,j}^2 - \|x_i - x_j\|_2^2\right)^2 + \sum_{m=1}^{M} \left(0^2 - \|p_m - x_{N+m}\|_2^2\right)^2 \right). \tag{3.14}$$

By defining the functions $f_i : \mathbb{F}^{N+M} \to \mathbb{R}$, $g_m : \mathbb{F}^{(N+M)} \to \mathbb{R}$,

$$f_i(X) = \sum_{j=1}^{N+M} \left(d_{i,j}^2 - \|x_i - x_j\|_2^2\right)^2, \qquad 1 \le i \le N+M, \tag{3.15}$$

$$g_m(X) = \|p_m - x_{N+m}\|_2^2, \qquad 1 \le m \le M, \tag{3.16}$$

we can rewrite 3.14 as,

$$\min_{X\in\mathbb{F}^{N+M}} \left(\sum_{i=1}^{N+M} \left(\sqrt{f_i(X)}\right)^2 + \sum_{m=1}^{M} \left(g_m(X)\right)^2 \right), \tag{3.17}$$

leading to a least squares problem with $N + 2M$ constraints and $d \cdot (N + M)$ variables. Thus, for $d \ge 2$ we have more variables than constraints. We use the Intel MKL least squares solver here, see [21], that uses a trust region algorithm, see [11]. The solver requires to have at least as much constraints as variables. Thus, we solve the problem dimension-wise, i.e. starting with an initial guess we solve 3.17 varying only the coordinates of the current dimension and keeping the other variables constant. Formally, let X_0 denote the initial guess,

$$X_0 = \begin{pmatrix} x_0^1 \\ \vdots \\ x_0^d \end{pmatrix}, \tag{3.18}$$

then we compute the partial solutions $\hat{x}^s \in [a, b]^{1\times(N+M)}$ by,

$$\hat{x}^s = \underset{x^s\in[a,b]^{1\times(N+M)}}{\arg\min} \left(\sum_{i=1}^{N+M} \left(\sqrt{f_i(X_s)}\right)^2 + \sum_{m=1}^{M} \left(g_m(X_s)\right)^2 \right), \ \forall s = 1, \ldots, d, \tag{3.19}$$

where X_s denotes the intermediate solution,

$$X_s = \begin{pmatrix} \hat{x}^1 \\ \vdots \\ \hat{x}^{s-1} \\ x^s \\ x_0^{s+1} \\ \vdots \\ x_0^d \end{pmatrix}. \tag{3.20}$$

Thus, we have now to solve d least squares problems with $N + 2M$ constraints and $N + M$ variables. The global solution $\hat{X}$ is then given by,

$$\hat{X} = X_d = \begin{pmatrix} \hat{x}^1 \\ \vdots \\ \hat{x}^s \end{pmatrix}. \tag{3.21}$$

If we want to improve the global solution, we can repeat the procedure using $\hat{X}$ as initial guess. A nice property of this approach is that we could easily control the number of dimensions. More specific, if e.g. the euclidean norm of the residuum is too large, we could increase the number of dimensions without recomputing the solution for the other dimensions. If we do not get a significant better solution by adding an additional dimension, we could of course also just throw away this dimension again. All in all, these are the steps we have to do for the constrained MDS:

1. Compute dissimilarities between real and virtual questions using the pairwise Levenshtein approach and compute distances between the virtual questions using the euclidean distance.
2. Build the overall dissimilarity matrix. Optionally compute all-pairs-shortest-path.
3. Solve the d least squares problems, repeat until solution does not improve significantly (with a limited number of iterations).
4. Compute a clustering for the features of the real questions in the batch. Compute cluster keywords, tags and center. Add a virtual question for all clusters where the weighted frequency sum of the keywords exceeds a given threshold.
5. If the number of virtual questions exceeds a given maximum, cluster the whole data processed so far and overwrite the current virtual questions with the virtual questions computed from this global clustering.

Now, we show that the Jacobian matrix can be computed efficiently, i.e. in $\mathcal{O}\left(d \cdot (N+M)^2\right)$. Note, that there are $\Omega\left((N+M)^2\right)$ non-zero entries in general and d is very small compared to $N+M$. With $X = (x_1, \ldots, x_{N+M}) = (x_{i,j}) \in \mathbb{F}^{N+M}$ we have for a fixed dimension t, $1 \leq t \leq d$, and for all i, k, $1 \leq i, k \leq N+M$,

$$\frac{\partial \sqrt{f_i(X)}}{\partial x_{k,t}} = \frac{1}{2} \cdot \frac{1}{\sqrt{f_i(X)}} \cdot \frac{\partial f_i(X)}{\partial x_{k,t}} \tag{3.22}$$

Plugging in the definition for f_i in 3.15 we get for its partial derivation,

$$\begin{aligned} \frac{\partial f_i(X)}{\partial x_{k,t}} &= \frac{\partial \left(\sum_{j=1}^{N+M} \left(d_{i,j}^2 - \|x_i - x_j\|_2^2 \right)^2 \right)}{\partial x_{k,t}} \\ &= \sum_{j=1}^{N+M} 2 \cdot (d_{i,j}^2 - \|x_i - x_j\|_2^2) \cdot \frac{\partial \left(- \|x_i - x_j\|_2 \right)^2}{\partial x_{k,t}}. \end{aligned} \tag{3.23}$$

For the remaining partial derivation we get,

$$\begin{aligned} \frac{\partial \left(- \|x_i - x_j\|_2 \right)^2}{\partial x_{k,t}} &= \frac{\partial \left(- \sum_{s=1}^{d} \left(x_{i,s} - x_{j,s} \right)^2 \right)}{\partial x_{k,t}} \\ &= \begin{cases} - \left(2x_{k,t} - 2x_{j,t} \right), & \text{if } i = k \text{ and } i \neq j \\ - \left(2x_{k,t} - 2x_{i,t} \right), & \text{if } j = k \text{ and } i \neq j \\ 0, & \text{otherwise} \end{cases}. \end{aligned} \tag{3.24}$$

Plugging in reversely, the Jacobian for the first part of the constraints is given by:

$$\frac{\partial \sqrt{f_i(X)}}{\partial x_{k,t}} = \begin{cases} \frac{2}{\sqrt{f_i(X)}} \cdot \sum_{j=1}^{N+M} (d_{i,j}^2 - \|x_i - x_j\|_2^2)(x_{j,t} - x_{k,t}), & \text{if } i = k \\ \frac{2}{\sqrt{f_i(X)}} \cdot (d_{i,k}^2 - \|x_i - x_k\|_2^2)(x_{i,t} - x_{k,t}), & \text{if } i \neq k \end{cases}. \tag{3.25}$$

Consider the computation of a fixed row i of the Jacobian matrix, i.e. the row belonging to the function $\sqrt{f_i}$. The left term is thus constant and we only need to compute it once. The function $f_i(X)$ and thus the left term can be computed in $\mathcal{O}(d(N+M))$. The remaining right term for the case $i = k$ can be computed in $\mathcal{O}(d(N+M))$ as well and this case obviously occurs only once for fixed i. The remaining right term for the case $i \neq k$ can be computed in $\mathcal{O}(d)$ and this case occurs $\mathcal{O}(N+M)$ times. Overall, we get a complexity of $\mathcal{O}(d(N+M))$ for one

row and thus, the complexity for all $N + M$ rows is in $\mathcal{O}(d(N + M)^2)$. What remains is the Jacobian matrix for the functions g_m,

$$\begin{aligned}\frac{\partial g_m(X)}{\partial x_{k,t}} &= \frac{\partial \|p_m - x_{N+m}\|_2^2}{\partial x_{k,t}} \\ &= \frac{\partial \sum_{s=1}^{d} \|p_{m,s} - x_{N+m,s}\|_2^2}{\partial x_{k,t}} \\ &= \begin{cases} 2(x_{N+m,t} - p_{m,t}), & \text{if } k = N + m \\ 0, & \text{otherwise} \end{cases},\end{aligned} \tag{3.26}$$

which can obviously be computed in $\mathcal{O}(M(N+M))$. Thus, the total complexity of the overall system is still $\mathcal{O}(d(N + M)^2)$.

Weighted Constrained MDS

We extend the constrained MDS approach to a weighted version now. There are two main reasons to introduce weights:

- Virtual questions represent a cluster of questions and are the basis for the relation of the batches. Especially questions with small dissimilarities to a virtual question should get transformed close to this virtual question to be in the same cluster at the end. Thus, the constraints for the dissimilarity values between a virtual and a real question should be weighted higher.
- Satisfying the constraints for small dissimilarities as close as possible is more important than satisfying those with large dissimilarities. For questions with small dissimilarities it is most of the times required that their features have also a small distance to have them in the same cluster. For questions with large dissimilarities, the exact distance of their features does not matter as long as they are somehow separated. Also, considering two questions that both have a small dissimilarity value to a third question it could even be better not to represent their computed dissimilarity.

Based on these observations, we use weights $w_{i,j}$ for all $1 \leq i, j \leq N + M$ of the following form here:

$$w_{i,j} = \begin{cases} w_{v \leftrightarrow r} \cdot (w_{\min} + (1.0 - d_{i,j})^{e_s}), & \text{if } i \leq N, j > N \text{ or } i > N, j \leq N \\ (w_{\min} + (1.0 - d_{i,j})^{e_s}), & \text{otherwise} \end{cases}, \tag{3.27}$$

where w_{min} is the constant minimum weight and $w_{v \leftrightarrow r} > 1$ is the constant weight factor for the constraints between virtual and real questions. The term $(1.0 -$

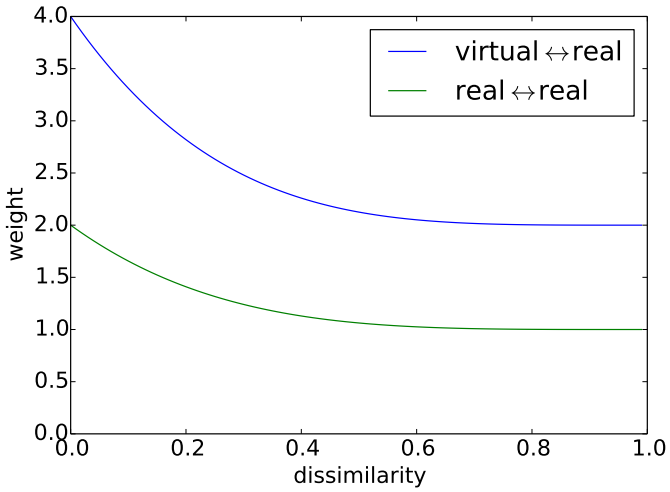


Figure 3.5: The weights for the constraints dependent on the dissimilarity value between the questions for $w_{min} = 1$, $w_{v\leftrightarrow r} = 2$ and $e_s = 4$.

$d_{i,j})^{e_s}$, with parameter e_s, ensures that the constraints between more similar questions are weighted higher than constraints between less similar questions (remember that $0 \leq d_{i,j} \leq 1$). The weights dependent on the dissimilarity value of the constraints are plotted in Fig. 3.5. Instead of f_i we use f_i^w now,

$$f_i^w = \sum_{j=1}^{N+M} w_{i,j} \cdot \left(d_{i,j}^2 - \|x_i - x_j\|_2^2\right)^2, \, 1 \leq i \leq N+M. \tag{3.28}$$

The Jacobian matrix can be derived similar to the unweighted version, resulting to,

$$\frac{\partial \sqrt{f_i^w(X)}}{\partial x_{k,t}} = \begin{cases} \frac{2}{\sqrt{f_i^w(X)}} \cdot \sum_{j=1}^{N+M} w_{i,j} \cdot \left(d_{i,j}^2 - \|x_i - x_j\|_2^2\right) \cdot (x_{j,t} - x_{k,t}), & \text{if } i = k \\ \frac{2}{\sqrt{f_i^w(X)}} \cdot w_{i,k} \cdot \left(d_{i,k}^2 - \|x_i - x_k\|_2^2\right) \cdot (x_{i,t} - x_{k,t}), & \text{if } i \neq k \end{cases}. \tag{3.29}$$

4 Development of a Clustering-Based Recommender System

In this chapter we discuss the development of the implemented recommender system. We start with the goals and put them into the general context of recommender systems. Afterwards, we describe how we use the hierarchical clustering to compute recommendations and discuss how additional data can be used.

4.1 Goals

The recommender system developed here is thought to recommend questions that could help answering the actual question of the user (which could differ from the one currently visited), but also subsequent questions (e.g. somebody who asks how to install a software could also be interested in how to use it). The context ("Q&A website") of this recommender system is a bit different to the context of most other recommender systems, due to the following facts:

- In most cases, we have no information about the user except that he visits a certain question currently.
- Even if we have more information about a user, the current question can be independent of those visited before (especially if the time frame between is not rather small).
- Thus, the currently visited question gives the most certain information about the actual question of the user.

Following these observations, a non-personalized approach is used here. More specific, the recommendations to a user will only depend on the currently visited question. To achieve the initially mentioned goals, for a given question we have the following two criteria:

1. Recommend questions that are similar.
2. Recommend questions of good quality.

As we have to deal with a large amount of questions, we could add that this has to be done somehow efficiently. How we try to achieve these goals is discussed in the next section.

4.2 Implementation

As already mentioned, the recommender system is based on the previously computed clustering. We start this section with describing how we use the clustering and go on with a discussion how additional data that is available can be used.

4.2.1 From Clusters to Recommendations

The basic idea to use the clustering is quite simple: as clusters should form a group of similar questions, they can be used as search space for possible recommendations. But, as we have a hierarchical clustering, there is more information about the clusters. Also, especially the top level clusters clusters can become very large. Thus, we only use the questions of a cluster that have this cluster as most specific cluster, i.e. they are not clustered on a deeper level. Remember that the idea of the hierarchical clustering is that the ancestors of a cluster represent more general topics and the descendants represent more specific topics. Following this idea, we can extend the search space by the two parameters l_{up} and l_{down}, i.e. the number of levels we look up and down in the cluster tree, adding the respective clusters (again only the questions that have this cluster as most specific cluster) to the search space. The possibility to add the siblings of a cluster as well is also implemented but not recommended at least for the top level clusters, as this would make the search space quite large. The underlying idea would be that—as these clusters have the same parent—we could find especially subsequent questions there.

Now, having defined the search space for a question, what remains is the question "how to search in the search space". We re-use the pair-wise Levenshtein approach here, see Section 3.2.1, but now using simply the similarity value as a rating and then take the questions with the highest ratings. Using this approach, we will hopefully achieve our first goal to "recommend questions that are similar". How we try to achieve the second goal will be discussed in the next section.

Quite obvious, the quality of the recommendations and the run time for the recommender system depend on the quality of the clustering. On the other hand, we could use the quality of the recommendations to compare the quality of different clusterings to some extent. Though it is not necessarily needed that all questions in a cluster are pair-wise similar, we need to have at least some similar questions in the (related) cluster(s) of the current question, to be able to compute good recommendations. For the run time, it is important that the size

of the clusters is somehow balanced and that we have a reasonably high number of clusters, as the run time for one cluster is quadratic in the size of the search space (assuming the rating computation for one question relative to the current question is in $\mathcal{O}(1)$).

4.2.2 Usage of Additional Data

The additional data we consider here are the user votes and the log data. The usage of the user votes is straightforward. Using weights w_v for the votes and w_s for the similarities, we compute the combined rating $r(q_c, q)$ for a question q with $v \in \mathbb{N}_0$ up-votes and similarity s to the current question q_c by:

$$r(q_c, q) = w_v \cdot \log(v + 1) + w_s \cdot s. \tag{4.1}$$

For now, the log data is not used, but we give a short outlook here of how it could be used in the future. The first idea that is already implemented is to use the log data to validate the computed recommendations as well as the clustering. For this purpose, we can see every entry of the log data as a visit of a question for which we have the following informations:

- user ID,
- session ID,
- time,
- visited question.

The idea is to reconstruct the order of questions an user visited during a session. These questions have a strong relation in most of the cases. With the question q_{start} the user started with and the set of questions R the user visited afterwards, we give scores for each $q_r \in R$ if q_r is a recommendation to q_{start}. We could also use this to validate the clustering, only checking if q_r and q_{start} have the same most specific cluster then.

To use the log data also for the recommender system in the future, we suggest three approaches:

- Use the idea for the validation to extend the search space for the related questions.
- Count the visits of the questions for a given time back in the past and use them similar to the user votes.

- Try to recognize if an user found his answer when visiting a question. Given the questions of a session $q_1, \ldots, q_n$ in the order visited, if q_i and q_{i+1} are not similar and the time the user visited q_i extends a given threshold (perhaps dependent on the content size), the user probably found the answer to his actual question at q_i. Again, we can now give an up-vote to q_i and use this similar to the user votes.

5 Results

In this chapter, we discuss the results for the clustering and the recommender system using the following data sets:

- **bitcoin**: Data set extracted from *stackexchange.com* containing over **8 thousand** questions about the digital currency bitcoin in English language.
- **autofrage**: A side portal from *gutefrage.net* containing over **15 thousand** questions about cars in German language.
- **computerfrage**: Another side portal from *gutefrage.net* containing over **67 thousand** questions about computers and other technical stuff in German language.
- **gutefrage**: The questions from *gutefrage.net*, which are over **16 million** and also in German language.

Experiments for the first three data sets were done on a dual socket Intel Xeon Processor E5-2670 (2.6GHz, 8 cores and 16 threads per socket) with 128GB RAM and QDR infiniband, experiments for the **gutefrage** data set were done on a quad socket AMD Bulldozer Opteron 6274 (2.2GHz, 16 cores and threads per socket) with 256GB RAM and QDR infiniband. The actual clustering as well as the recommender system is implemented in C++, it is already parallelized for most of the parts for shared memory systems using OpenMP. Several libraries are used, especially the SG^{++} toolbox [33] for the spare grid density estimation, the Intel MKL [21] to solve the constrained MDS problem via least squares and the math library Armadillo [34]. There are also lots of parameters for the whole program, if not stated otherwise we use the following here:

- **String measure**: weighted version of pair-wise Levenshtein
  - Weighting function: $\omega(f_\mathrm{w}) = a \cdot f_\mathrm{w}^2 + b \cdot f_\mathrm{w} + c$,
    - f_w is the absolute frequency of word w in the data set Q
    - a, b, c computed such that $\omega(0) = 1$, $\omega\left(\frac{|Q|}{10}\right) = 0$, $\omega'(0) = 0$
  - scaling factor w_{sim} for questions with at least one common word: $\frac{3}{4}$

  - exponent for the dissimilarity value e_d: 4

- **Remove unrelated questions from batch**: yes, ***if***
  - no relation (i.e. max. dissimilarity value) to at least 98% of the questions in the batch ***and***
  - no relation to a virtual question (relation to virtual question if dissimilarity value $< \frac{1}{2} w_{sim}$)

- **Use tags**: yes, via adjacency matrix

- **MDS**: constrained
  - #dimensions: 2
  - Compute distance matrix from dissimilarity matrix: no
  - min. weight w_{min} for a constraint: 1
  - weight factor for constraints between virtual and real questions $w_{v \leftrightarrow r}$: 20
  - exponent for the similarity values e_s: 4
  - max. #virtual questions M_{max}:
    * start with 100
    * ensure $M_{max} \geq 10$ times #global clusters
  - weighted frequency sum threshold ϵ_{freq}: 25%
  - max. #keywords K_{max}: 20
  - min. #keywords K_{min}: 5

- **Sparse grid density estimation**:
  - start level: 4
  - #refinement steps: 6
  - #refinement points: 300
  - regularization parameter λ: 10^{-5}
  - learning rate: $\frac{1}{\#\text{batches processed}}$

- **Clustering**: hierarchical
  - number of nearest neighbors k: 8
  - n_{steps} to iterate over the density range: 100
  - split threshold t: 0.4

  - number of samples per dimension: 100

- **Recommender system**:
  - look l_{up} levels up in the cluster tree: 0
  - look l_{down} levels down in the cluster tree: 0
  - use sibling clusters: no
  - weight for the similarity w_s: 1
  - weight for the votes w_v: 0
  - #recommendations computed: 6

- **Scaling**:
  - Sparse grid scaled to $[0, 1]$
  - Features scaled to $[0.1, 0.9]$
  - Dissimilarity values scaled to $[0, 0.4]$

5.1 Clustering

In this section, we look at the results for the improvements discussed in Chapter 3 and compare them to the previous approaches. We start with the dissimilarity measures, focusing mainly on the string metrics but also have a look at the different approaches to use the tags as well as different weighting functions for the words. Afterwards, we compare the flat clustering with the hierarchical clustering results and also look at the influences of some of the parameters. We go on with a comparison of the classical and constrained MDS, discussing also some parameters that influence the constrained MDS. We finish this section by discussing the current overall state of the clustering.

5.1.1 Dissimilarity Measures

We start this section with a little example of 10 questions that are taken from the **bitcoin** data set, see Tab. 5.1. The idea is that we have two groups of three questions that are very similar and two groups of two questions that have also something in common. We look at the dissimilarty matrices of three different string measures for the titles here:

- Levenshtein metric, see Tab. 5.2,
- pair-wise Levenshtein without weighting the words, see Tab. 5.3,
- weighted pair-wise Levenshtein, see Tab. 5.4.

ID	Title	Tags
1	is mining still profitable	mining-reward, mining-profitability
2	is asic bitcoin mining profitable	mining-profitability, altcoin, mining-reward
3	is bitcoin mining profitable yet	mining-profitability, mining-hardware
4	how can i keep my wallet secure	wallet, backup, security, encryption
5	which wallet is better and secure	wallet, bip32
6	is javascript wallet generator available at bitaddressorg relatively secure	wallet, security
7	how do i make a transaction	transactions, raw-transaction, multi-sig-transactions
8	what is this transaction i did not make	transactions
9	can i run bitcoind without downloading new blocks	blockchain, daemon
10	heartbleed when will bitcoind 091 be released for ubuntu 1204	ubuntu

Table 5.1: 10 questions taken from the **bitcoin** data set, where questions 1 to 3 and 4 to 6 are very similar, 7, 8 and 9, 10 are also somehow similar where 7 and 8 also share a common tag, while 9 and 10 do not.

For the Levenshtein metric, there is no clear distinction between similar questions and questions that have nothing in common. The first group has small dissimilarity values for all pairs, as also the order of the common words ("mining", "profitable" and "bitcoin") is the same. For the other groups, similar questions are only partially recognized as such. The unweighted pair-wise Levenshtein algorithm recognizes all questions of the same group as similar, while most of the dissimilar questions have the max. dissimilarity value. Exceptions are the values for the questions 2 and 3 to 9 and 10, where the words "bitcoin" and "bitcoind" (the latter is not a typo) are matched. As one might expect, the word "bitcoin" has a very high frequency in the data set **bitcoin**. Thus, questions that have only "bitcoin" in common should not be recognized as similar. With the weighted pair-wise Levenshtein algorithm, we achieve exactly this. Note, that we did use the whole data set to compute the weights. The dissimilar questions that were recognized as somehow similar have now also the max. dissimilarity value. The values for the similar questions only change slightly, as all matched words other than "bitcoin" have similar weights. The features of the questions for all three string measures are shown in Fig. 5.1. As already observed from the dissimilarity matrix, the groups of similar questions are only partially close in the feature space for the Levenshtein metric. For the unweighted pair-wise Levenshtein algorithm, the last three groups are recognized as expected, while the first group is split up due to the relations of questions 2 and 3 to the last group. For the unweighted pair-wise Levenshtein algorithm, all groups are together and well separated. Note, that we didn't scale the dissimilarity values further for the transformation.

Considering the tags, there are only common tags for questions in the same group. The dissimilarity matrix for the weighted pair-wise Levenshtein algorithm after using the tags via the Laplacian are shown in Tab. 5.5, values for using the tags via the weighted adjacency matrix are shown in Tab 5.6. The dissimilarity values are all scaled equally inside the groups for the Laplacian, while for the adjacency matrix dissimilarity values for questions that share two common tags are reduced more compared to those that only share one common tag. The features for the weighted pair-wise Levenshtein algorithm are not affected by the tags in a qualitative sense, as all groups were already well recognized before and there are no common tags between the groups. The features for the Levenshtein metric and the unweighted pair-wise Levenshtein algorithm using the tags via the Laplacian and adjacency matrix are shown in Fig. 5.2. Using the Laplacian, also questions of different groups tend to get merged together, while for the adjacency matrix the recognition of the different groups is improved at least for the Levenshtein metric. The qualitative result for the unweighted pair-wise Levenshtein approach is more or less equal to the result without using the tags. Still, for larger settings the tags will be useful for all measures, as there

ID	1	2	3	4	5	6	7	8	9	10
1	0	0.25	0.23	0.41	0.43	0.98	0.39	0.48	0.61	0.84
2	0.25	0	0.15	0.46	0.48	0.95	0.46	0.46	0.54	0.77
3	0.23	0.15	0	0.43	0.49	0.95	0.43	0.51	0.54	0.77
4	0.41	0.46	0.43	0	0.34	0.93	0.34	0.54	0.64	0.8
5	0.43	0.48	0.49	0.34	0	0.82	0.41	0.49	0.64	0.75
6	0.98	0.95	0.95	0.93	0.82	0	1	0.97	0.98	1
7	0.39	0.46	0.43	0.34	0.41	1	0	0.44	0.67	0.8
8	0.48	0.46	0.51	0.54	0.49	0.97	0.44	0	0.57	0.75
9	0.61	0.54	0.54	0.64	0.64	0.98	0.67	0.57	0	0.75
10	0.84	0.77	0.77	0.8	0.75	1	0.8	0.75	0.75	0

Table 5.2: The dissimilarity matrix using the Levenshtein metric for the 10 questions in Tab. 5.1, scaled to $[0, 1]$ by dividing by the max. dissimilarity. The similarities of the questions are only partially recognized, also some questions that are not similar have rather small dissimilarity values.

ID	1	2	3	4	5	6	7	8	9	10
1	0	0.047	0.0093	1	1	1	1	1	1	1
2	0.047	0	0.0029	1	1	1	1	1	0.24	0.36
3	0.0093	0.0029	0	1	1	1	1	1	0.24	0.36
4	1	1	1	0	0	0.097	1	1	1	1
5	1	1	1	0	0	0.097	1	1	1	1
6	1	1	1	0.097	0.097	0	1	1	1	1
7	1	1	1	1	1	1	0	0	1	1
8	1	1	1	1	1	1	0	0	1	1
9	1	0.24	0.24	1	1	1	1	1	0	0.36
10	1	0.36	0.36	1	1	1	1	1	0.36	0

Table 5.3: The dissimilarity matrix for the 10 questions in Tab. 5.1 using the pair-wise Levenshtein approach without weighting the words. The similar questions are well recognized while the dissimilar questions have the max. dissimilarity value of 1, except for 2,9; 2,10; 3,9 and 3,10. This is due to the fact that the words “bitcoin” and “bitcoind” are matched, note that bitcoind is not a typo.

ID	1	2	3	4	5	6	7	8	9	10
1	0	0.021	0	1	1	1	1	1	1	1
2	0.021	0	0.021	1	1	1	1	1	1	1
3	0	0.021	0	1	1	1	1	1	1	1
4	1	1	1	0	0	0.19	1	1	1	1
5	1	1	1	0	0	0.19	1	1	1	1
6	1	1	1	0.19	0.19	0	1	1	1	1
7	1	1	1	1	1	1	0	0	1	1
8	1	1	1	1	1	0	0	1	1	
9	1	1	1	1	1	1	1	1	0	0.38
10	1	1	1	1	1	1	1	1	0.38	0

Table 5.4: The dissimilarity matrix for the 10 questions in Tab. 5.1 using the weighted pair-wise Levenshtein approach. The similar questions are well recognized while the dissimilar questions have the max. dissimilarity value of 1. Opposed to the unweighted version, the questions 2,9; 2,10; 3,9 and 3,10 have also dissimilarity value 1, as the word "bitcoin" has weight 0 (the weights are computed using the whole data set).

are similar questions that have no common words but common tags.

We now look at different word weighting functions for the pair-wise Levenshtein algorithm. Quite obvious, words with higher frequency should have lower weights, thus, the weighting function dependent on the frequency of the words should be monotonically decreasing. The next idea to construct a weighting function is to set it to zero for frequencies that are greater than some threshold, we use 10% here. Inspired by tf–idf (term frequency–inverse document frequency), first introduced in [23], the first weighting function we came up with for a data set with N questions is:

$$\omega(f) = \max\left(0, \log\left(\frac{N}{10 \cdot f}\right)\right). \tag{5.1}$$

A quadratic function q_2 on $[0, \frac{N}{10}]$ with similar course can be constructed by claiming the following conditions:

$$q_2(0) = 1, \quad q_2\left(\tfrac{N}{10}\right) = 0, \quad q_2'\left(\tfrac{N}{10}\right) = 0. \tag{5.2}$$

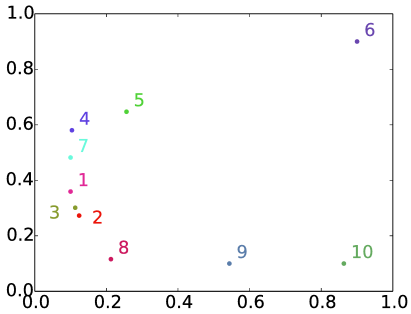
As previously mentioned, for $f > \frac{N}{10}$ we set $q_2(f) = 0$. This function decreases slower than the logarithmic one for the interesting frequencies. Both functions have in common, that the weights for words with moderate frequency are already

ID	1	2	3	4	5	6	7	8	9	10
1	0	0.0053	0	1	1	1	1	1	1	1
2	0.0053	0	0.0053	1	1	1	1	1	1	1
3	0	0.0053	0	1	1	1	1	1	1	1
4	1	1	1	0	0	0.047	1	1	1	1
5	1	1	1	0	0	0.047	1	1	1	1
6	1	1	1	0.047	0.047	0	1	1	1	1
7	1	1	1	1	1	1	0	0	1	1
8	1	1	1	1	1	1	0	0	1	1
8	1	1	1	1	1	1	1	1	0	0.38
10	1	1	1	1	1	1	1	1	0.38	0

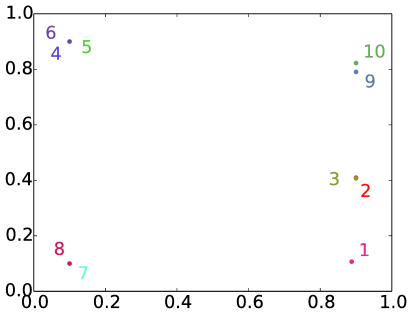
Table 5.5: The dissimilarity matrix for the 10 questions in Tab. 5.1 using the weighted pair-wise Levenshtein approach after using the tags via the Laplacian matrix. The dissimilarity values are reduced equally inside the groups of similar words, although some questions share one and some share two common tags.

ID	1	2	3	4	5	6	7	8	9	10
1	0	0.007	0	1	1	1	1	1	1	1
2	0.007	0	0.011	1	1	1	1	1	1	1
3	0	0.011	0	1	1	1	1	1	1	1
4	1	1	1	0	0	0.063	1	1	1	1
5	1	1	1	0	0	0.094	1	1	1	1
6	1	1	1	0.063	0.094	0	1	1	1	1
7	1	1	1	1	1	1	0	0	1	1
8	1	1	1	1	1	1	0	0	1	1
9	1	1	1	1	1	1	1	1	0	0.38
10	1	1	1	1	1	1	1	1	0.38	0

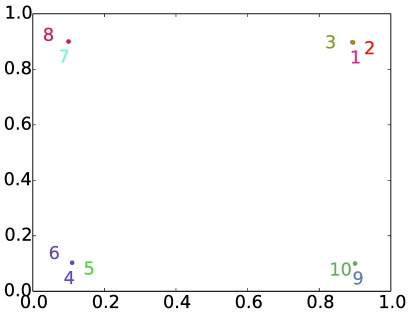
Table 5.6: The dissimilarity matrix for the 10 questions in Tab. 5.1 using the weighted pair-wise Levenshtein approach after using the tags via the weighted adjacency matrix. The dissimilarity values are reduced more for the questions that share two common tags than for the ones who only share one common tag.



(a) Levenshtein

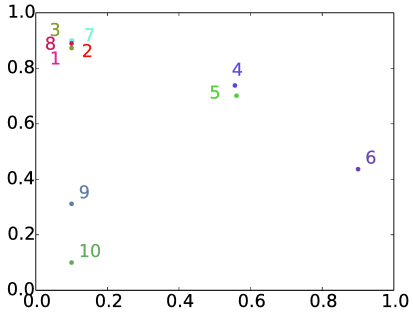


(b) Unweighted pair-wise Levenshtein

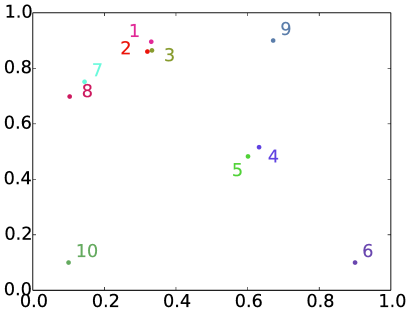


(c) Weighted pair-wise Levenshtein

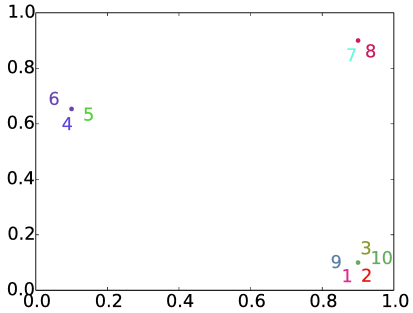
Figure 5.1: Transformation into the feature space for the 10 questions from Tab. 5.1 using the different string metrics without using the tags. For the Levenshtein metric, only questions of the first group (1 to 3) are close together. Using the unweighted pair-wise Levenshtein algorithm, three groups are recognized and the first group is split up due to the relations between the first and the last group. With the weighted pair-wise Levenshtein algorithm, all groups are recognized and well separated.



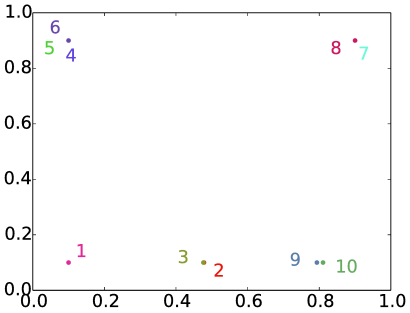
(a) Levenshtein, Laplacian



(b) Levenshtein, Adjacency



(c) Unweighted pair-wise Levenshtein, Laplacian



(d) Unweighted pair-wise Levenshtein, Adjacency

Figure 5.2: Transformation into the feature space for the 10 questions from Tab. 5.1 using the Levenshtein metric (top) and the unweighted version of the pair-wise Levenshtein algorithm (bottom) and using the tags via the Laplacian matrix (left) and the weighted adjacency matrix (right). The Laplacian matrix tends to merge different groups together, the adjacency matrix improves the result at least for the Levenshtein metric.

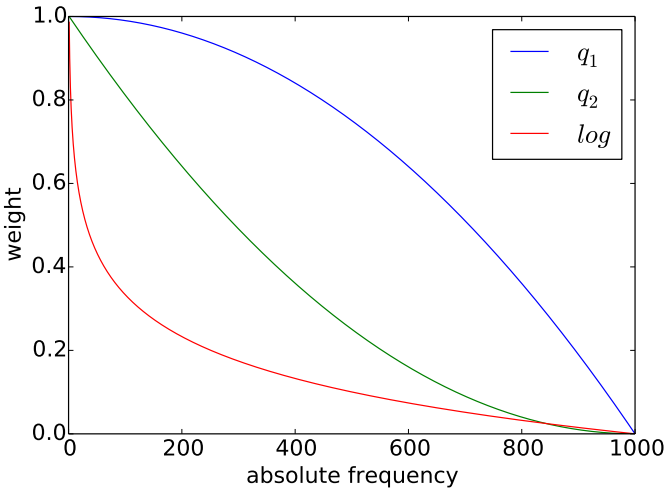


Figure 5.3: Two quadratic and a logarithmic word weighting function for a data set of 10 000 questions. All have in common that they are 0 for a relative frequency $\geq 10\%$, i.e. for an absolute frequency of 1000 in this case. The quadratic functions are both 1 for a frequency of 0, the first derivative for q_1 is 0 for a frequency of 0%, the first derivative of q_2 is 0 for a frequency of 10%. The logarithmic function has been normalized such that it is 1 for an absolute frequency of 1.

quite low, see Tab. 5.7 for some examples. Thus, small dissimilarity values will be rather rarely and especially for the constrained MDS approach, we will hardly find any keywords that can connect questions from different batches. This leads us to the also quadratic weighting function on $[0, \frac{N}{10}]$ that we use here, claiming the following conditions now:

$$q_1(0) = 1, \quad q_1\left(\tfrac{N}{10}\right) = 0, \quad q_1'(0) = 0. \tag{5.3}$$

Again, we set $q_1(f) = 0$ for $f > \frac{N}{10}$. Opposed to the other two convex functions, this function is now concave on $[0, \frac{N}{10}]$. Thus, words with moderate frequency have nearly the same weight, only words with very high frequency have significant lower weights. What "high frequency" means can obviously controlled by changing the threshold where we set the function to zero (second condition). A plot of all three functions is shown in Fig. 5.3.

word	abs. frequency	rel. frequency	q_1	q_2	log
analysis	10	$1.18 \cdot 10^{-3}$	0.999	0.978	0.658
java	21	$2.48 \cdot 10^{-3}$	0.999	0.953	0.548
connect	50	$5.92 \cdot 10^{-3}$	0.996	0.887	0.419
hash	101	$1.19 \cdot 10^{-2}$	0.985	0.776	0.315
network	194	$2.29 \cdot 10^{-2}$	0.947	0.594	0.218
bitcoind	259	$3.06 \cdot 10^{-2}$	0.906	0.481	0.175
bitcoin	2052	$2.42 \cdot 10^{-1}$	0	0	0

Table 5.7: Weights for some words from the **bitoin** data set using different weighting functions. For the logarithmic one and q_2, there is a much larger difference for the weights of words with low frequencies and moderately high frequencies than for q_1. Thus, for the logarithmic weighting function and q_2 we get less low dissimilarities and it will be tough to find keywords for the virtual questions that can connect the batches.

5.1.2 Hierarchical Clustering

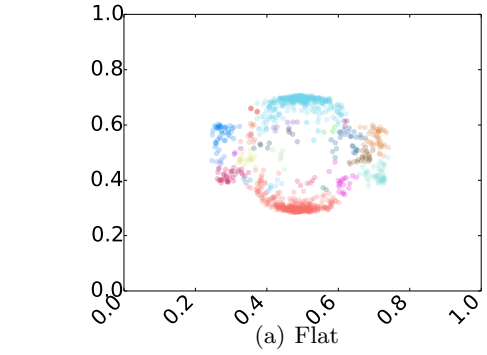
In this section, we look at some results for the hierarchical clustering. Starting with a comparison to the flat clustering algorithm, we look at the two unique parameters for the hierarchical clustering afterwards, i.e. the number of different density thresholds we explore and the split threshold. We finish this section with two approaches to rate the resulting cluster tree.

Comparison of Flat and Hierarchical Clustering

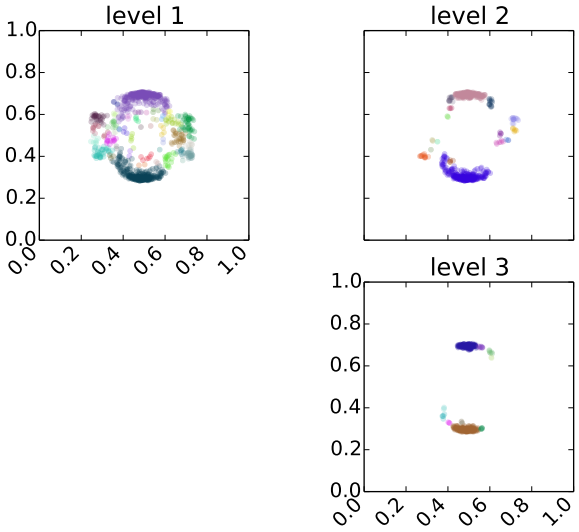
We start with a short discussion of the flat compared to the hierarchical clustering. The different clusterings for a single batch of the data set **autofrage** are shown in Fig. 5.4. We chose the density threshold automatically by iterating over the thresholds also used by the hierarchical clustering algorithm and maximized the number of clusters. For the first level, the larger clusters are more or less the same, only some are already split up at the first level for the hierarchical clustering. Also, the hierarchical algorithm tends to split regions with only few points into more clusters leading to 42 clusters at the first level, while the flat algorithm only detects 28 clusters. Obviously, the hierarchical algorithm is also able to split especially the larger clusters at a deeper level.

Parameters of the Hierarchical Clustering Algorithm

We look at the two main parameters of the hierarchical clustering algorithm now: the number of steps to explore different density thresholds and the split

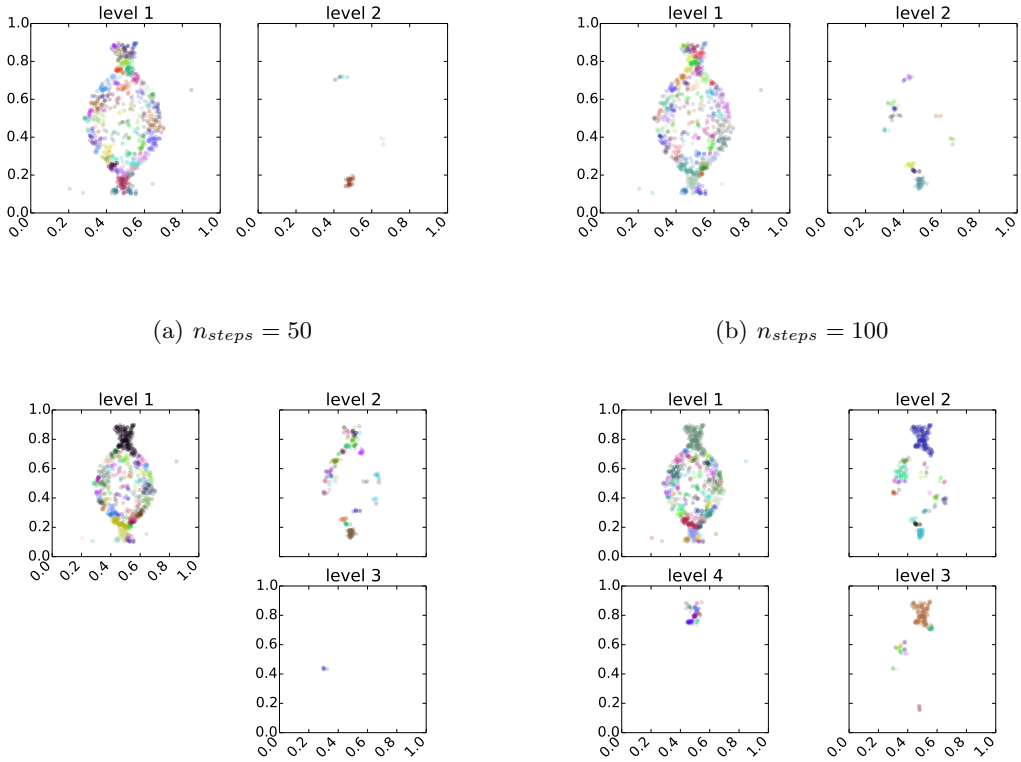


(a) Flat



(b) Hierarchical

Figure 5.4: Flat and hierarchical clustering for the first batch of the data set **autofrage** without removing questions, each color represents a cluster. The density threshold for the flat clustering is computed automatically by maximizing the number of clusters. This is done by iterating over the same thresholds that the hierarchical clustering algorithm uses, leading to a density threshold of 5.56. The flat clustering has 28 clusters, the hierarchical clustering has 42 clusters on the first level, 16 on the second level and 10 on the third and last level. For the first level, the clustering is quite similar to the flat one, only some points belong to different clusters and the hierarchical algorithm splits up some more clusters. Of course, the hierarchical clustering is now able to split up especially the larger clusters further.



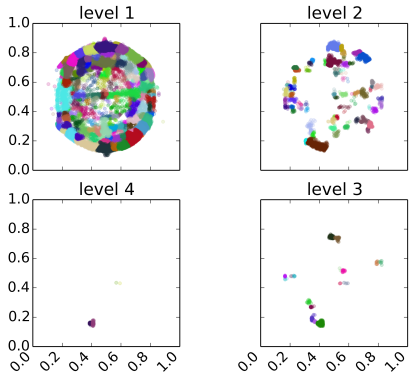
(a) $n_{steps} = 50$

(b) $n_{steps} = 100$

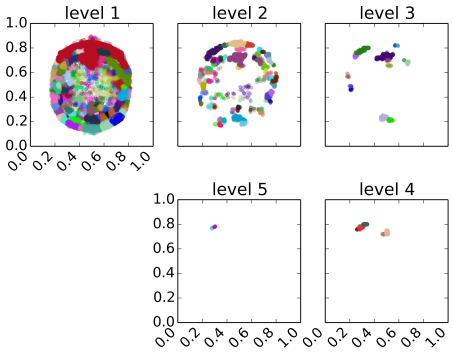
(c) $n_{steps} = 200$

(d) $n_{steps} = 1000$

Figure 5.5: Hierarchical clustering for the questions of the first batch (size 2000) of the data set **autofrage** for different number of steps. Each color represents a cluster. The result at the first level is similar for all clusterings. For a larger number of steps the first level clusters tend to grow and are subdivided at deeper levels.



(a) $n_{steps} = 100$



(b) $n_{steps} = 1000$

Figure 5.6: Global hierarchical clustering for the questions of the data set **autofrage** for 100 and 1000 steps, each color represents a cluster. The tendency observed for one batch that more steps lead to more levels is significantly weakened here. Most of the questions are only clustered up to level 3 in both cases. Note, that the features for the questions are not the same, as the number of steps also influences the processing of single batches (due to the computation of the virtual questions via the clustering).

threshold. The resulting cluster trees for different number of steps are shown in Fig. 5.5. While the clusters at the first level look similar for all number of steps, for more steps more and more questions are clustered at deeper levels and thus, the number of levels also increases slightly. The clusters for the first level are also partially larger for more steps, but they are then subdivided at deeper levels. For the global clustering, these tendencies are significantly weakened, see Fig. 5.6, as the density is smoother.

For the split threshold, similar tendencies can be seen. As expected, for low split thresholds we get deeper cluster trees, for higher thresholds the cluster tree tends to get more and more flat, see Fig. 5.7. Also, for lower split thresholds and thus more levels, we have larger clusters at the first level that are then subdivided at deeper levels. Opposed to the number of steps though, the influence of the split threshold is not weakened for the global clustering, see Fig. 5.8.

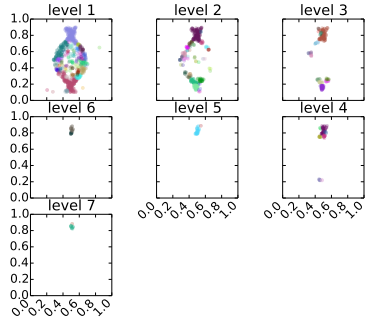
Ratings for the Clustering

Even for the moderately large data sets, it is tough to evaluate manually how good a clustering is compared to other clusterings. Thus, we discuss two approaches to rate different hierarchical clusterings in a more automated way here. This could also be useful for further parameter optimization. The first rating approach is based on the idea that clusters should form groups of similar questions. For the similarity measure, we use again the pair-wise Levenshtein approach, but using the similarity value without further transformations (the same that is used in the recommender system), see Section 3.2.1. We compute the pair-wise similarities of the questions in a cluster and their sum is normalized by its theoretical maximum value. Formally, we compute the similarity index I_{sim} of a cluster C with its N_C specific questions $Q_C = \{q_1, \ldots, q_{N_C}\}$ (excluding the questions that are clustered at a deeper level) and similarities $s_{i,j}$ by:

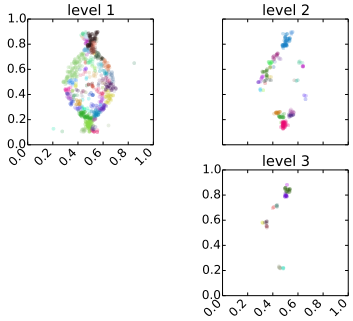
$$I_{sim}(C) = \frac{2}{|Q_C| \cdot (|Q_C| - 1)} \sum_{i=1}^{N_C - 1} \sum_{j=i+1}^{N_C} s_{i,j}. \tag{5.4}$$

Motivated by the procedure to compute the virtual questions from the clusters, the second approach to rate a clustering is by the summed weighted frequency of the cluster key words. Formally, the frequency index I_{freq} of cluster C with $\hat{N}_C$ questions (including questions of the child clusters) and key words $k_1, \ldots, k_{K_C}$ with their frequency $f_C(k_i)$ inside the cluster is given by,

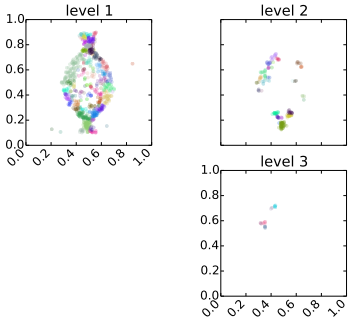
$$I_{freq}(C) = \frac{1}{\hat{N}_C} \sum_{i=1}^{K_C} \omega(k_i) \cdot f_C(k_i), \tag{5.5}$$



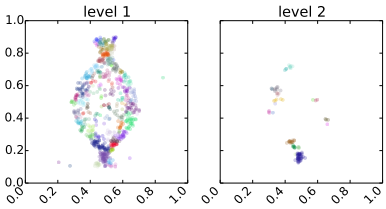
(a) $t = 0.1$



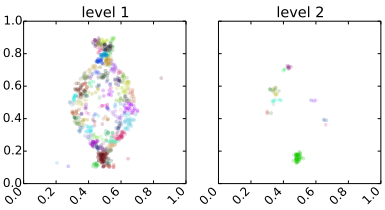
(b) $t = 0.2$



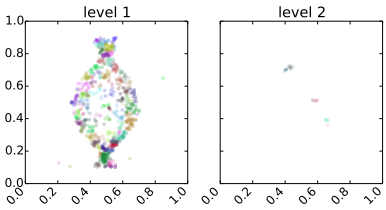
(c) $t = 0.3$



(d) $t = 0.4$

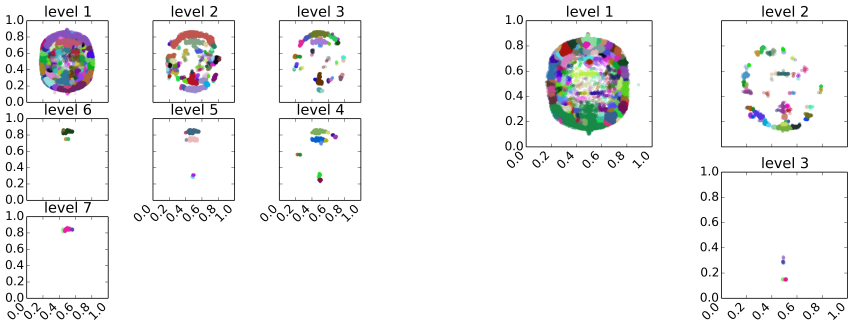


(e) $t = 0.5$



(f) $t = 0.6$

Figure 5.7: Hierarchical clustering for the questions of the first batch (size 2000) of the data set **autofrage** for different split thresholds, each color represents a cluster. As expected, there are more levels for smaller thresholds, for a threshold of $t = 0.6$ the clustering is almost flat.



(a) $t = 0.2$ (b) $t = 0.5$

Figure 5.8: Global hierarchical clustering for the questions of the data set **autofrage** for different split thresholds, each color represents a cluster. As in the case for one batch, the smaller split threshold of 0.2 leads to significantly more levels than the split threshold of 0.5. Note, that the features for the questions are not the same, as the split threshold also influences the processing of single batches (due to the computation of the virtual questions via the clustering).

where ω is the word weighting function. To rate the overall clustering, we look at the distribution of both rating indices. Histograms of both indices for different batch sizes are shown in Fig. 5.9. For increasing the batch size from 500 to 2000, the results get better, but when increasing it further to 5000 they get worse again. The first fact is quite intuitive, having more questions inside a batch, more direct dissimilarities between the questions are computed where we do not have to take the detour via the virtual questions. The latter is most probably explained by the fact that we get more and more constraints inside a batch while the number of constraints that connect the batches (constraints between virtual and real questions) are more or less the same. Thus, the features will more and more depend on the questions in the current batch and less on the virtual questions. Similar questions in different batches will then not be connected by the virtual questions and thus, probably not be in the same cluster in the end. Another issue could be that even without the virtual questions the MDS is not able to closely represent the given dissimilarity matrix, especially as we only use two dimensions here. An approach to overcome the first issue could be to increase the weight for the constraints between real and virtual questions dependent on the batch size (perhaps also dependent on the number of virtual questions). For the latter, we could try to increase the number of dimensions—in a static or dynamic way.

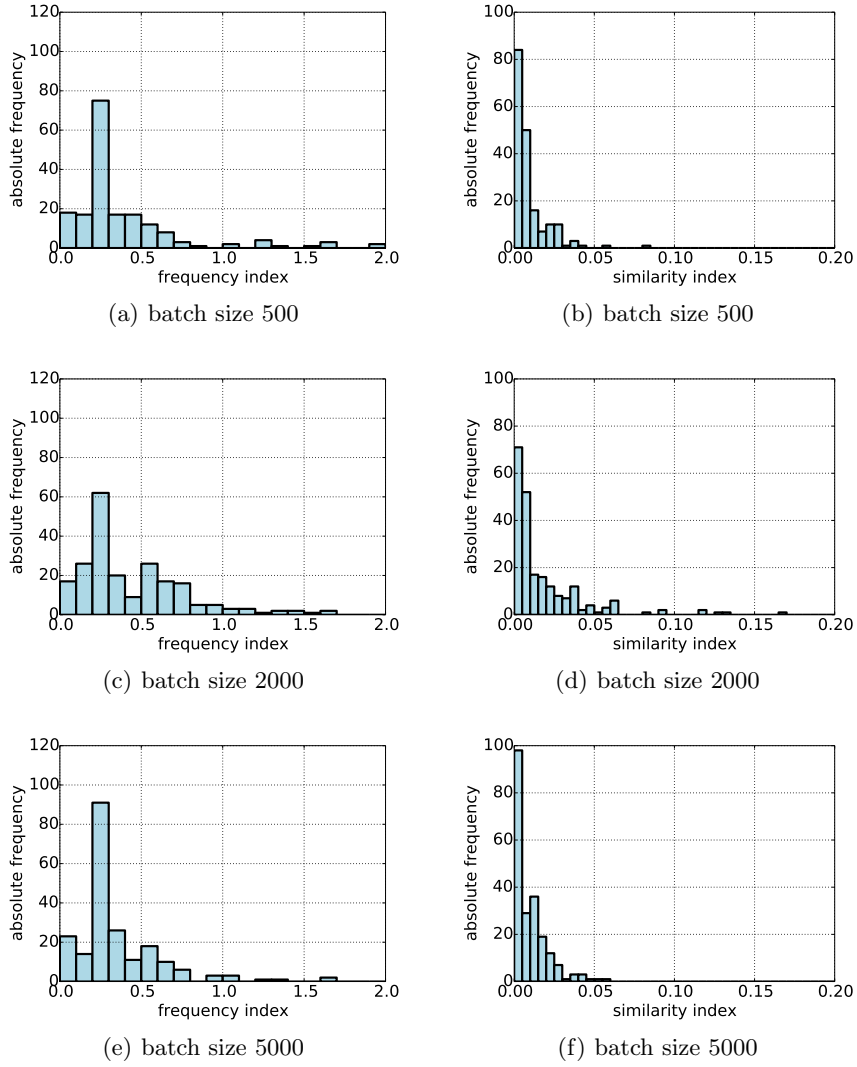
The rating indices for removing and not removing unrelated questions are shown in Fig. 5.10. Removing questions gives us a slight improvement of the clustering. The obvious drawback is that we have to reprocess the removed questions or that we do not have all questions clustered if not doing so. With the approach to remove questions chosen here, only for the first batches a larger number of questions is removed and for those where we add the once-removed questions again, as there are more and more virtual questions (questions do not get removed if they have small dissimilarity to at least one virtual question). It could be an approach for further investigation to evaluate if removing questions in a less conservative way is worth it.

5.1.3 Constrained Multidimensional Scaling

In this section, we first look at the differences between the classical and constrained MDS and then discuss the results for the constrained MDS a bit more in detail. Especially, we show the differences when using the dissimilarity matrix compared to the distance matrix and also how removing unrelated questions from a batch influences the resulting features.

In Fig. 5.11, features for the first batches of the data set **computerfrage** are shown for the classical and constrained MDS. For the classical MDS, we use the distance matrix (as the assumption for the classical MDS is that the dissimilarities are metric) opposed to the constrained MDS where we use the dissimilarity matrix. The approach to remove questions from the batches can not be applied directly to the classical MDS, as we have no virtual questions. When using only the first condition (at least related to 2% of the questions in the batch), we would always remove a large amount of questions and thus, we do not remove questions for the classical MDS. Considering the results obtained by the classical MDS, the batches are completely independent from each other and the features are always transformed around the diagonal from the bottom-left corner to the upper-right corner of the feature space. The constrained MDS typically arranges the features in an elliptic form, as most of the dissimilarities have the maximum value (0.4 due to the scaling). As we use a density-based clustering here, we want to have high density regions that are separated by low density regions. For the classical MDS, there are some high density regions for the single batches, but they are not well separated. For the constrained MDS, this is at least partially the case, though the clusters are also not too well separated for most of the batches.

The clear difference of using the dissimilarity matrix compared to the distance matrix (i.e. the metric obtained by computing all-pairs-shortest-path from the dissimilarity matrix) is shown in Fig. 5.12. For the distance matrix, questions inside a batch are mostly close together, while the virtual questions are mainly



(a) batch size 500

(b) batch size 500

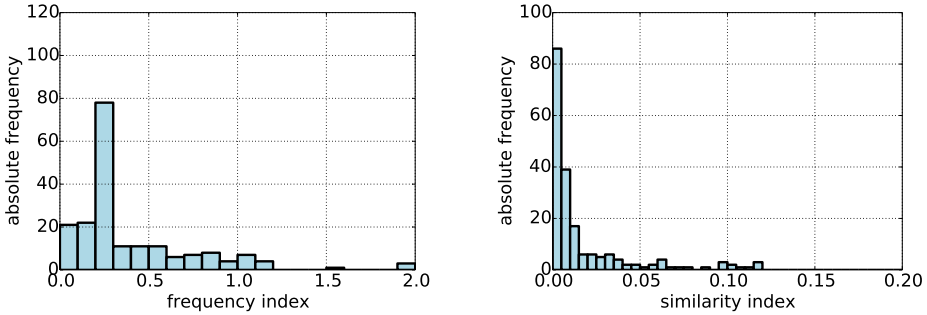
(c) batch size 2000

(d) batch size 2000

(e) batch size 5000

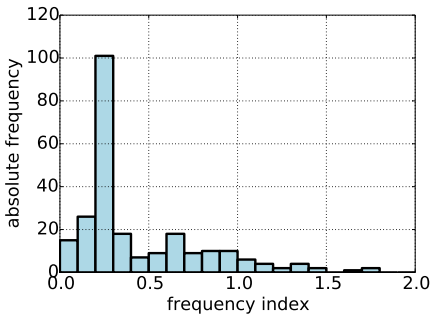
(f) batch size 5000

Figure 5.9: Cluster ratings for different batch sizes for the data set **computerfrage**. On the left, histograms for the frequency index (weighted frequency sum of cluster keywords), on the right, histograms for the similarity index (normalized sum of pair-wise similarities) of the clusters are shown. The x-axis is cut off at 2 (frequency index) and at 0.2 (similarity index), as larger values are rarely and result from very small clusters. We get better ratings when increasing the batch size from 500 to 2000, but the ratings get worse for a batch size of 5000 and we also get less clusters (213 for 5000, 224 for 2000 and 190 for 500). This is probably due to the fact that for too large batch sizes the influence of the virtual questions is very limited.

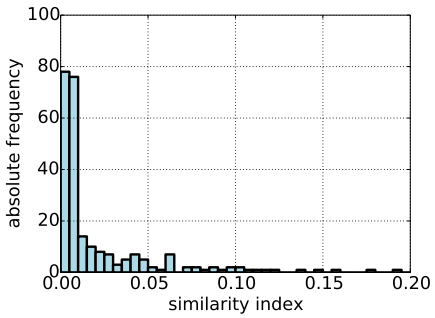


(a) no remove, frequency rating

(b) no remove, similarity rating



(c) remove, frequency rating



(d) remove, similarity rating

Figure 5.10: Cluster ratings for the data set **computerfrage** with (bottom) and without (top) removing unrelated questions. For the first, there are 199 clusters, for the latter there are 246 clusters (counting clusters at every level). Note, that we cut off the x-axis at 2 for the frequency rating and at 0.2 for the similarity rating, as the values greater than that are rarely and result from very small clusters. Overall, we can see a slight improvement when removing unrelated questions.

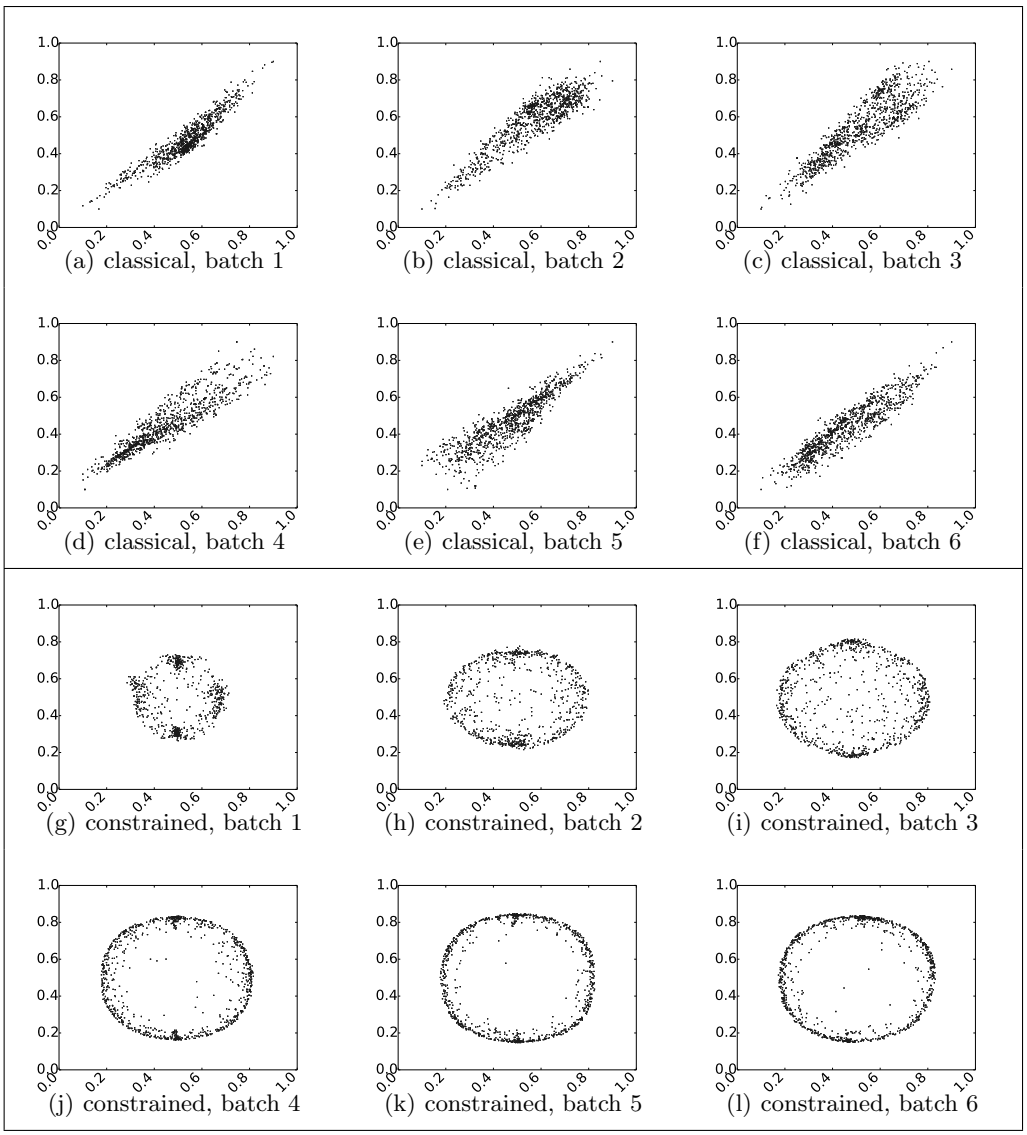


Figure 5.11: First six batches of the data set **computerfrage** using classical MDS with the distance matrix and constrained MDS with the dissimilarity matrix. Note, that for the classical MDS also no questions have been removed. For the classical MDS, features are always transformed at more or less the same area of the feature space and high density regions are not well separated from low density regions, which violates the assumption for the density-based clustering. For the constrained MDS, a larger area of the feature space is used. For the first batch, clusters are quite well separated while with increasing number of processed batches this is less the case.

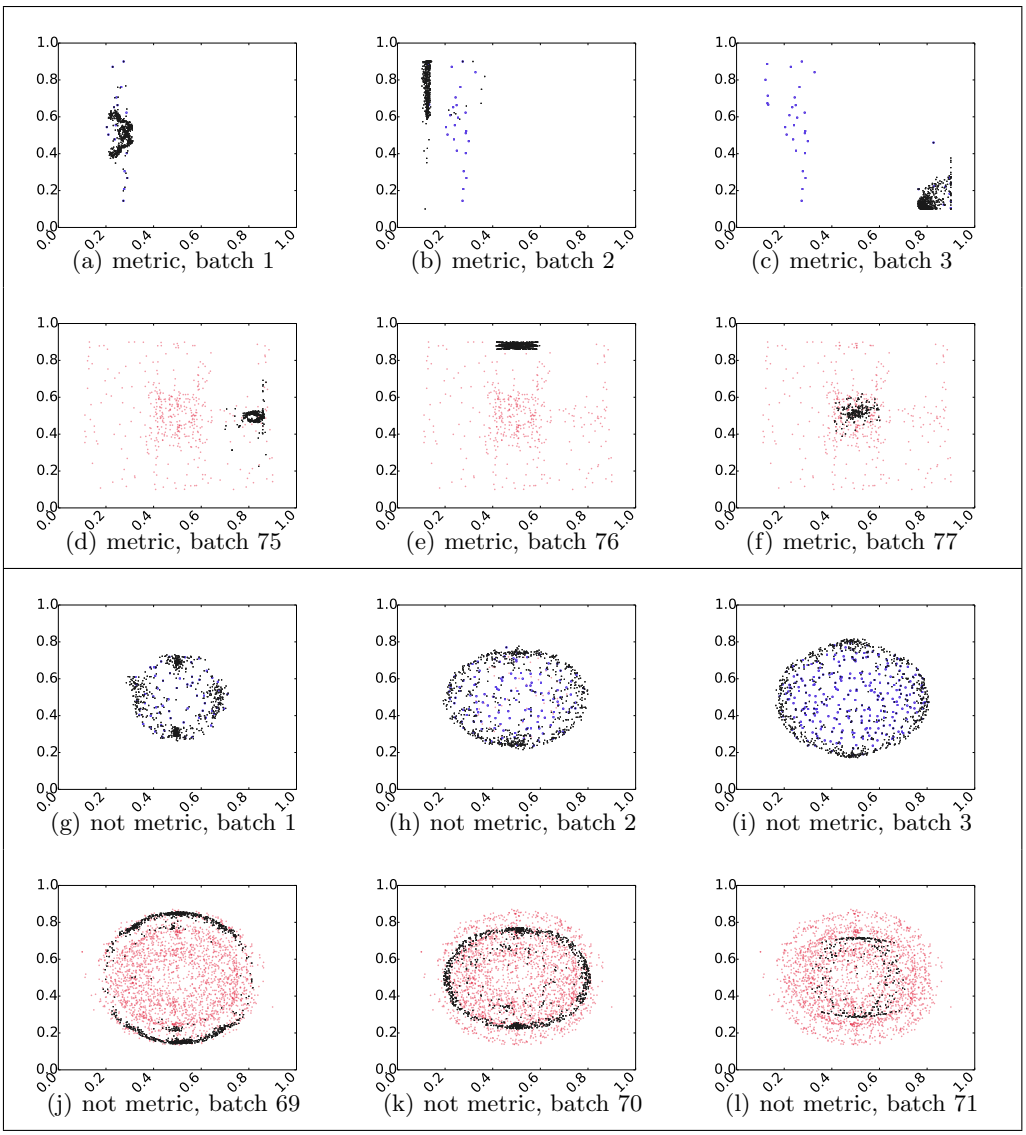


Figure 5.12: Features of the first and last three batches of full size of the data set **computerfrage** using the distance matrix (top) and the dissimilarity matrix (bottom). The virtual questions before processing the current batch are marked by the red points, for the first three batches also the resulting virtual questions are shown (marked by blue circles). For the distance matrix, the features of one batch are mostly close together and the question features for the different batches look more irregular. For the dissimilarity matrix, features spread out and are mostly aranged in an elliptic form due to the fact that most questions have max. dissimilarity value.

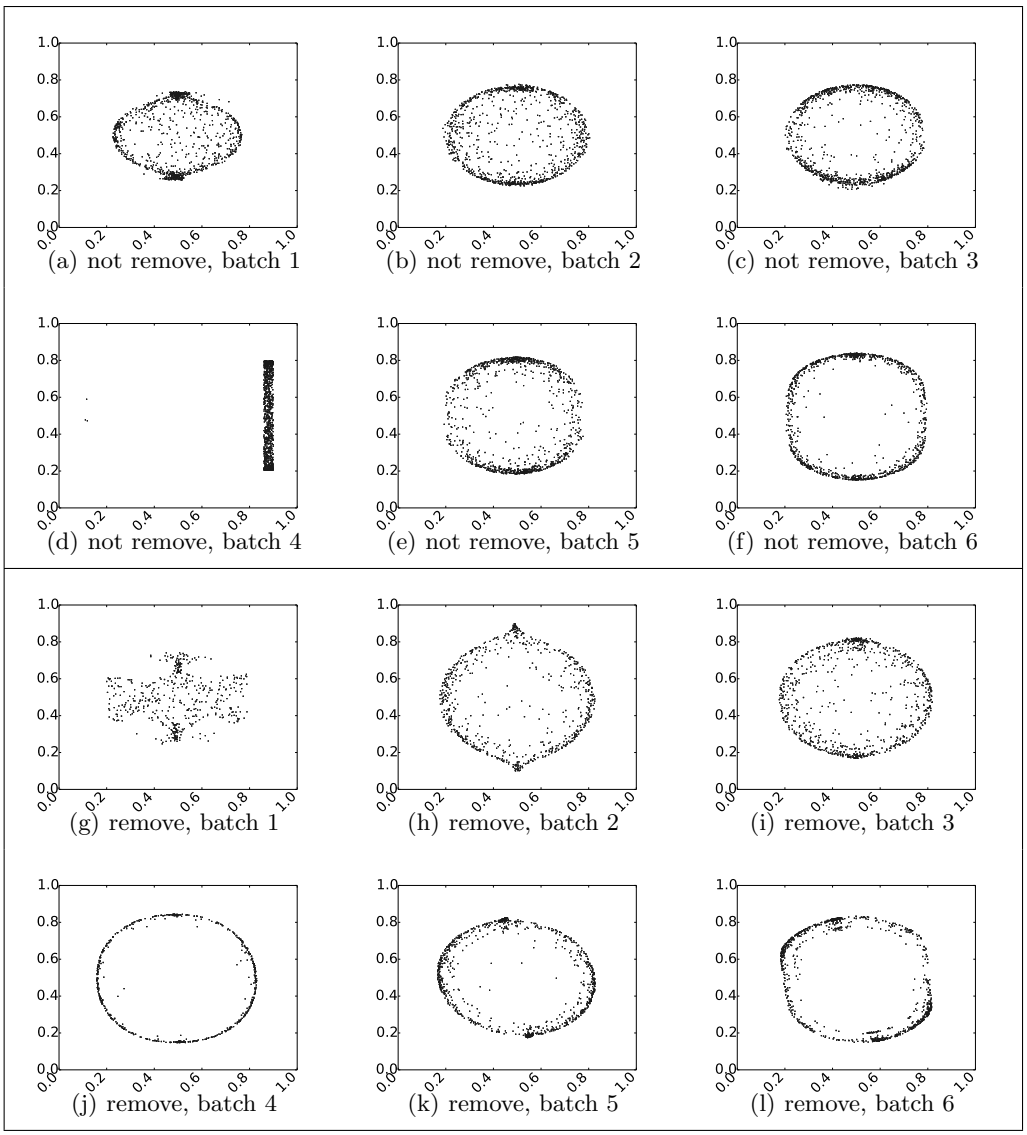


Figure 5.13: Features of the first six batches of the data set **autofrage** with (bottom) and without (top) removing unrelated questions. In batch 4 (bottom), the removed questions are processed. For the first batch, removing questions makes a clear difference, for the batches afterwards less questions are removed and thus, the results look more similar in most cases. However, with removing unrelated questions we seem to avoid effects like in batch 4 on the top (in this case, there are only very few strong relations to the virtual questions which are moved with the whole batch to the boundary of the feature space where we have no virtual questions yet).

ignored. When using the distance matrix, we have much more small dissimilarity values between real and virtual questions as well as between the real questions. Thus, the constraints between the real questions are quite well satisfied by transforming them somehow close together. For the constraints between real and virtual questions this is not possible without violating the position constraints. Thus, either the features of the virtual questions (with small dissimilarities to the virtual questions) will be moved a lot or the constraints between virtual and real questions will not even approximately be fulfilled in most cases – or a mixture of both, which makes the virtual questions almost useless in either case. Looking at it in a more abstract way, we loose critical information about the actual computed dissimilarities when computing the distance matrix. Even though it is reasonable to say that if questions q_1 and q_2 as well as questions q_2 and q_3 are closely related, then also questions q_1 and q_3 are probably related, the all-pairs-shortest-path algorithm does not care about how many questions are there that connect q_1 and q_3 and also not how many questions are involved to connect them. The constrained MDS with the pure dissimilarities is thus better suited to keep the important information and reject the information that probably also does not reflect reality (the dissimilarity measure will never be perfect).

Now, we discuss what we achieve by removing unrelated questions from a batch. Features for the first batches of the data set **autofrage** with and without removing questions are shown in Fig. 5.13. For the first batches (and especially for the first), we remove more questions as there are obviously not too many virtual questions yet (questions do not get removed if they have relatively small dissimilarity value to at least one virtual question), the number of removed questions per batch stabilizes typically quite fast to significantly less than 100. Exceptions are mainly the batches where we process the removed questions again. Thus, a significant difference for the features can be seen only for the first batches. However, for some batches questions are not spread out over the feature space as well when not removing questions, see e.g. the fourth batch in Fig. 5.13 on the top, note that is an extreme case though. This happens if there are only few relations to the virtual questions and is thus mainly avoided when unrelated questions are removed. Even though the noticeable difference for the features of single batches is rather small, the features are more spread out overall when removing unrelated questions.

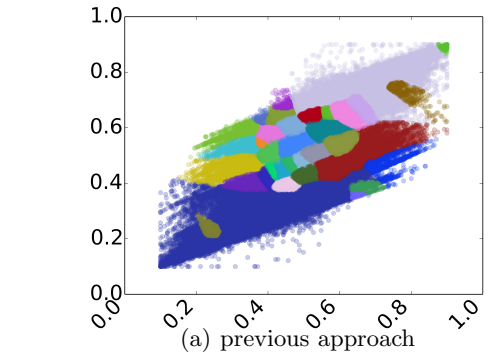
5.1.4 Overall Picture

After looking at the single steps that are needed for the clustering, we give a short overall picture of what the different improvements all together brought us. The resulting clusterings for the approaches taken before (Levenshtein met-

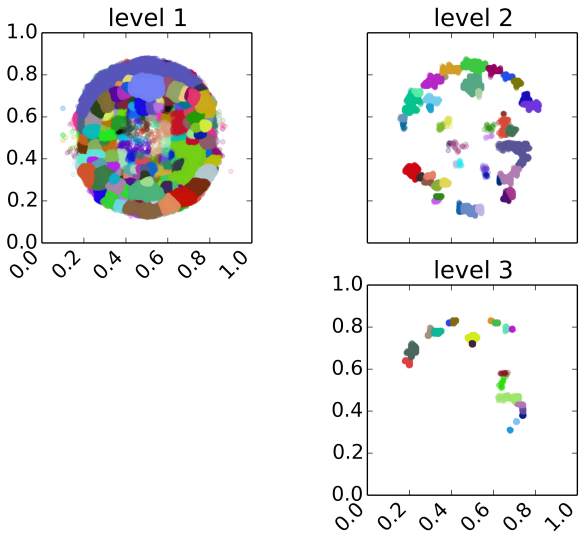
ric, Laplacian, dissimilarity matrix, classical MDS, no questions are removed, flat clustering) and for our standard parameters (especially weighted pair-wise Levenshtein, adjaceny matrix, dissimilarity matrix, constrained MDS, remove questions, hierarchical clustering) for the data set **computerfrage** are shown in Fig. 5.14. For the first one, relatively large parts of the feature space remain empty, we get also relatively few and large clusters. For the latter, features are more spread out filling nearly the whole feature space (remember that we scale the features to $[0.1, 0.9]$). We also get much more and smaller clusters, but some clusters are still too large and not meaningful. Especially for larger data sets, this becomes more and more a problem, see Fig. 5.15, e.g. when using the clustering for the recommender system (for the data set **computerfrage** this is not critical when properly parallelized). Thus, though the discussed improvements seem to be a step in the right direction, further investigation of the clustering will be needed.

5.2 Recommender System

In this section, we want to look at some results for the recommender system. As already mentioned, the run time of the recommender system and the quality of the recommendations depend on the underlying clustering. For the **gutefrage** data set, already after processing the first 1 million questions there are some too large clusters that make it infeasible to compute recommendations via the approach described in Chapter 4. Thus, we only discuss the recommendations for the smaller data sets here. Also, we do not use the votes here, as this is not really meaningful in an offline setting. In Tab. 5.8, the recommendations and their ratigns for two questions from the data set **computerfrage** are shown. For the first question the recommendations are quite good, while the second question has no relation to its recommendations, which is also indicated by the lower ratings. To somehow evaluate the overall quality of the recommendations, we use a brute-force recommender system here, i.e. the ratings are computed pair-wise for all questions in the data set. This is the theoretical best result we could obtain by any clustering based on the used rating function (which is the similarity value obtained by the pair-wise Levenshtein approach here). Of course, this can only be done for rather small data sets. Histograms of the recommendation ratings for the data set **autofrage** using different number of levels to look up and down in the cluster tree as well as for the brute-force recommender system are shown in Fig. 5.16. By increasing the number of levels, we can improve the recommendations slightly. Also, the number of questions with no recommendations (rating function is zero for all questions in the search space) is decreased from 3772 ($l_{up} = l_{down} = 0$) to 3141 ($l_{up} = l_{down} = 1$) and 3088



(a) previous approach



(b) improved clustering

Figure 5.14: State of the clustering before this thesis (top) and with the discussed improvements (bottom) for the data set **computerfrage**. Several improvements can be seen here. The classical MDS transforms the questions always into the same area of the feature space, thus, large parts remain empty. In contrast, the features for the constrained MDS are much more spread out. Opposed to the case for a single batch, the flat clustering is not able to split the questions into a number of clusters comparable to the hierarchical clustering, leading to very large clusters (largest cluster contains over 24000 questions, data set has about 67000 questions). On the other hand, also the hierarchical has some clusters at the first level that are still too large relative to the size of the data set (largest cluster contains over 5000 questions).

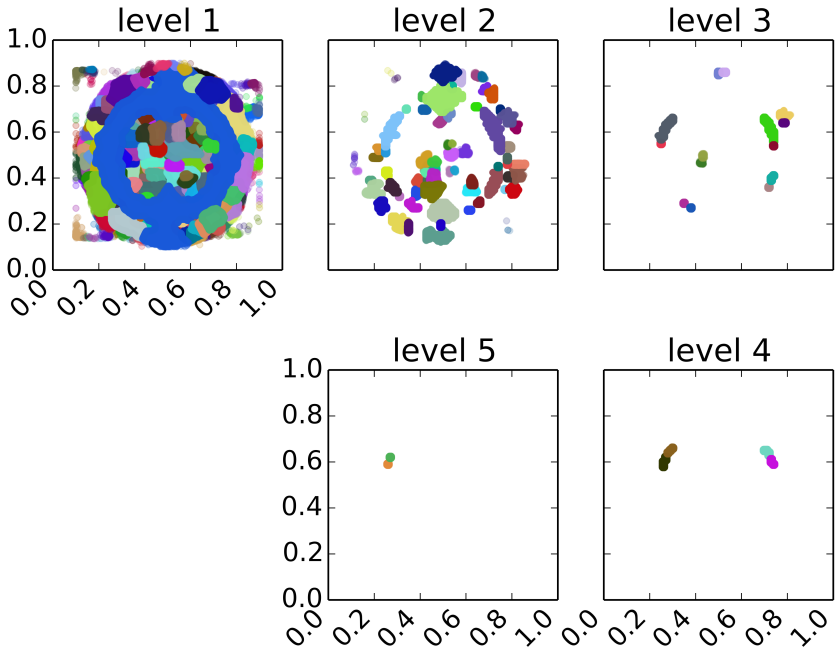
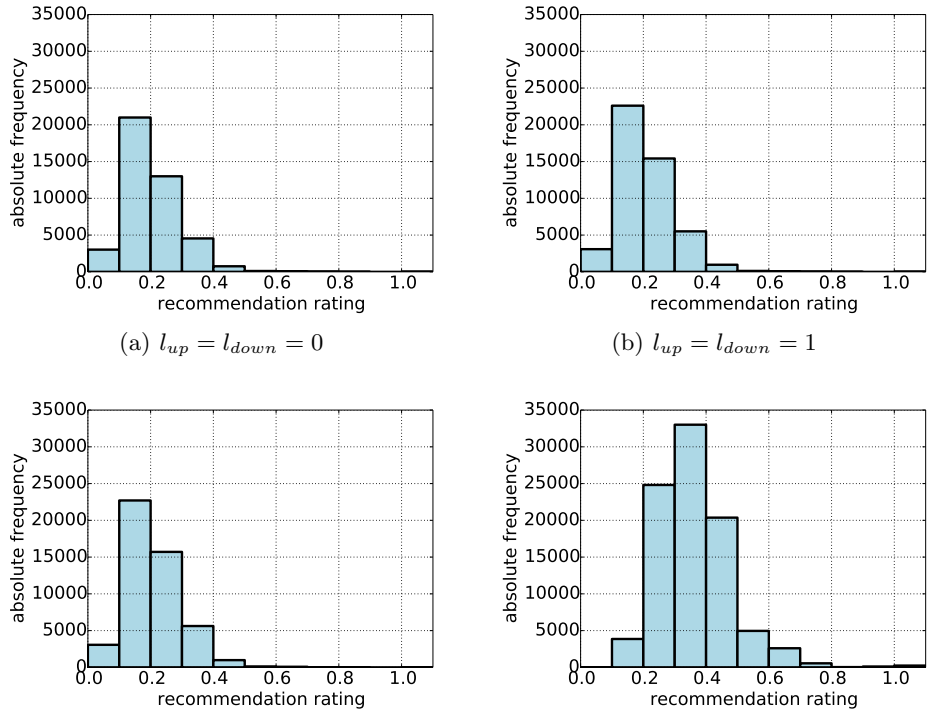


Figure 5.15: The cluster tree for the first 1 million questions from the data set **gutefrage**. There are some very large clusters, the largest cluster at the first level contains over 595 000 questions (more than half of the processed questions), over 322 000 of them not clustered at a deeper level (about a third of the processed questions). This shows that the clustering has to be improved further.

$l_{up} = l_{down} = 2$. Additionally, 907 questions have been removed before. For the brute-force recommender system, only 133 questions have no recommendations. Although we would not expect that there is a meaningful clustering leading to recommendations as good as for the brute-force recommender system, we can probably get closer by improving the clustering further.

question	recomendations (rating)
ist dieser gamer pc gut eigentlich schon oder	1. wird aus diesen teilen ein guter gamer pc (0.598996) 2. brauche hilfe beim zusammenstellen eines gamer pc (0.334987) 3. ist mein pc fuer games tauglich vor kurzen gekauft (0.332321) 4. pc ist immer an (0.331645) 5. ist das ein pc zum zockenets2 (0.331272) 6. gamer pc konfiguration im high end bereich (0.271904)
wie kann ich bei dem samsung ue 40 d 6540 kanaele verschieben	1. eignet sich die gtx 650 zum zocken (0.166915) 2. gutes gamer notebook bis 650 gesucht (0.166915) 3. wird mein nexus one das neue android 40 bekommen (0.166904) 4. wie laeuft das update auf android 40 ab (0.166904) 5. bringt es etwas pc viren in quarantaene zu verschieben (0.166896) 6. wie verschieb ich dateien vom pc auf einen usb stick (0.166896)

Table 5.8: The recommendations for two questions from the data set **computerfrage** with their ratings. For the first question, the recommendations are quite well while for the second questions there is no relation from the question to the recommendations, as also indicated by the ratings. Note, that the titles are normalized in a pre-processing step.



(a) $l_{up} = l_{down} = 0$

(b) $l_{up} = l_{down} = 1$

(c) $l_{up} = l_{down} = 2$

(d) brute-force

Figure 5.16: Histograms for the recommendation ratings of the data set **autofrage** for different levels to look up and down in the cluster tree to extend the search space. At the bottom right, the histogram for the brute-force recommender system is shown, i.e. ratings are computed pair-wise for all questions in the data set. This gives us the theoretical best result we can achieve using any clustering relative to the used rating function. We see slight improvements for increasing the levels (it can not get worse of course), but also for $l_{up} = l_{down} = 2$ the brute-force ratings are significantly better.

6 Summary

Of making many books there is no end,
and much study wearies the body.
— Ecclesiastes 12,12b

The two main goals of this work have been the analysis and improvement of the clustering algorithm and the implementation of a clustering based recommender system. Overall, the quality of the clustering could be improved significantly while keeping the algorithm efficient in the sense that it could be used in an online setting. On the other hand, especially for larger data sets we get still too large and not meaningful clusters. To extract meaningful information from the clustering or to use it for the recommender system for large data sets, the clustering has to be improved further. But before discussing what could be done in further investigation, let's recap the most important results.

The first important step for the clustering is to define a dissimilarity measure between the questions. Compared to the previous approach, i.e. the Levenshtein metric, we could achieve quite good results here with the weighted pair-wise Levenshtein approach. Similar questions are recognized as such in many cases and also dissimilar questions have mostly large dissimilarity values. The criteria via the Levenshtein metric to recognize words as similar works quite well and generally meaningless words are excluded by a stop word list as well as high frequent words in a specific data set are lower weighted. Also, the usage of the tags via the weighted adjacency matrix compared to the Laplacian seems to be more consistent.

To be able to process the questions via sparse grid density estimation, the next step is to transform the questions into the feature space. Considering the previous approach, i.e. using classical MDS, there has been no relation between the batches. This problem was addressed by the constrained MDS, adding virtual questions to a batch whose positions are fixed by additional constraints. To compute the virtual questions, the clusters obtained from the previous processed batches are used. Overall, we could relate the batches to some extent by this approach yet, though it is an open question if the main assumption, i.e. that a cluster can be represented by a reasonable low number of keywords combined with the most frequent tags, holds at least for a reasonable large amount of the clusters. Also, there are quite some parameters that influence the constrained

MDS in a direct or indirect way, where it is not quite clear how far from optimal the choices we made here are.

Considering the hierarchical clustering algorithm, we can split up the questions into more and thus smaller clusters than the flat clustering algorithm is able to for any density threshold. With the simple split method and the possibility to not cluster every data item at every level, we are able to compute more meaningful hierarchies compared to keeping the whole dendrogram as is done in the classic hierarchical clustering algorithms (which is the result when we never move child clusters up and cluster all nodes of a parent cluster also at the next level). With the split threshold, we are able to control the height of the cluster tree to some extent. While it is quite obvious that some choices for the split threshold are too low and some are too high, it is not quite clear if there is any optimal choice, as this could especially also depend on the underlying data (the split algorithm only operates on the samples).

Considering the recommender system, we get already acceptable recommendations for moderately large data sets. For larger data sets the clusters are still too large, making it infeasible to compute recommendations via the approach discussed here (at least we would have to skip large parts of the data set).

Now, let's look at some approaches that we think could be worth investigating to further improve the clustering. First, as already indicated, it could be worth to study some of the parameters in a more structured way. We already observed that at least some of them are not independent, so this is not trivial. To compare different parameter combinations, the ratings for the clustering provided in Section 5.1 as well as the log data as described in Section 4.2.2 and the recommendation ratings could be used. Sparse grids could be employed to deal with the high-dimensional parameter space. Still, the most relevant parameters should be identified first (which should be mainly the ones newly introduced in this thesis). It is not quite clear though, how much we can achieve by simply optimizing the parameters. Most probably, there will be also more fundamental changes necessary. Going through some of the main steps in the overall clustering algorithm, we suggest the following:

- **String measures**: While the pair-wise Levenshtein approach works quite well already, there are still especially similar questions that are not recognized as such. Two main reasons for this are that synonyms as well as compound words compared to their partwords normally will not be recognized as similar. Dealing with the first one could get quite expensive, but probably also not impossible. The second one, however, could be done without increasing the complexity significantly, as the distance of the substrings is computed by the algorithm for the Levenshtein metric anyway.

- **Feature space transformation**: A problem with the constrained MDS is that the feature space will be filled nearly everywhere and there is no space for new clusters, and we can not easily scale it as we would have to reprocess the global density estimation somehow. We suggest the following two approaches to deal with this:
  - We could make use of the fact that the constrained MDS algorithm is already designed to determine a reasonable number of dimensions automatically. Thus, if needed, we could increase the number of dimensions from time to time. Neither the features of the already processed questions nor the sparse grid density estimation would have to be reprocessed necessarily when increasing the number of dimensions. This could be achieved by setting the feature coordinates of the already processed questions to the middle of the sparse grid interval for the new dimension (which is $[0, 1]$ here). Due to the tensor product approach of the sparse grid basis functions, extending the global sparse grid with an additional dimension is then more or less trivial, as we can just copy the surpluses and potentially have to scale them to obtain a density (i.e. the integral of the sparse grid interpolant on the extended sparse grid domain is 1).
  - Instead of adding virtual questions to the feature space of the real questions, we could compute virtual questions from single batches and relate the batches by transforming the virtual questions of all batches into their own feature space. We could then cluster the virtual questions feature space and merge clusters from different batches by this.

- **Cluster post-processing**: To post-process the clusters, two basic operations would be needed:
  - **Split**: The main problem of the current clustering is that we get too large clusters. Thus, we could try to somehow split the clusters if they are too large or not meaningful, as e.g. indicated by the frequency index. This could be done by reprocessing the questions inside a cluster, cluster the resulting feature space and replace the cluster in the global cluster tree by the resulting local cluster tree. We could also just try to keep the cluster meaningful by removing questions from the cluster that are not represented by its virtual question.
  - **Merge**: As similar questions are not always transformed close together, there will be clusters that have essentially the same topic. This could be discovered by the similarities of the virtual questions.

So there is some work to do, many questions have been processed and many will be in the future. The remaining question is this: “Will the virtual questions get real?”—find some recommendations above!

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[2] Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2013.

[3] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[4] Adelchi Azzalini and Giovanna Menardi. Clustering via nonparametric density estimation: The r package pdfcluster. *Journal of Statistical Software*, 57(1):1–26, 2014.

[5] Adelchi Azzalini and Nicola Torelli. Clustering via nonparametric density estimation. *Statistics and Computing*, 17(1):71–80, 2007.

[6] Marko Balabanović and Yoav Shoham. Fab: Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, March 1997.

[7] Andrea Baraldi and Palma Blonda. A survey of fuzzy clustering algorithms for pattern recognition. I. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 29(6):778–785, 1999.

[8] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications (Springer Series in Statistics)*. Springer, 2 edition, August 2005.

[9] Kevin Bougé. Download stop words. `https://sites.google.com/site/kevinbouge/stopwords-lists`. Visited at: 2015-12-09.

[10] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. on Knowl. and Data Eng.*, 8(6):866–883, December 1996.

[11] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-region Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[12] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical Report TR-04-25, University of Texas Dept. of Computer Science, 2005.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pages 226–231, 1996.

[14] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[15] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recogn.*, 41(1):176–190, January 2008.

[16] William J. Frawley, Gregory Piatetsky-shapiro, and Christopher J. Matheus. Knowledge discovery in databases: an overview, 1992.

[17] Glenn Fung. A comprehensive overview of basic clustering algorithms, 2001.

[18] J. Garcke, M. Griebel, and M. Thess. Data mining with sparse grids. *Computing*, 67:225–253, 2001.

[19] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of Data Mining*. MIT Press, Cambridge, MA, USA, 2001.

[20] Markus Hegland, Giles Hooker, and Stephen Roberts. Finite element thin plate splines in density estimation, 2000.

[21] Intel. Intel(R) Math Kernel Library 11.2 Update 4 for Linux*. `https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation`.

[22] Alan Julian Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413):205–224, 1991.

[23] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[24] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data : an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley, New York, 1990. A Wiley-Interscience publication.

[25] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[26] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[27] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of Machine Learning*, pages 829–838. Springer, 2010.

[28] Giovanna Menardi and Adelchi Azzalini. An advancement in clustering via nonparametric density estimation. *Statistics and Computing*, 24(5):753–767, 2013.

[29] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[30] Benjamin Peherstorfer. *Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques*. Dissertation, Department of Informatics, Technische Universität München, October 2013.

[31] Nymphia Pereira and SatishKumar Varma. Survey on content based recommendation system, 2016.

[32] Dirk Pflüger. Data Mining mit Dünnen Gittern. Diplomarbeit, IPVS, Universität Stuttgart, March 2005.

[33] Dirk Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Dissertation, Institut für Informatik, Technische Universität München, München, February 2010.

[34] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical report, NICTA, September 2010.

[35] Denys Sobchyshak. Online graph clustering with sparse grids density estimation. Master's thesis, Fakultät für Informatik, Technische Universität München, October 2015.

[36] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.