Universiteit Utrecht

Msc Thesis

ICA-5487447

How do clusters evolve?

Papachristou Dimitra

August, 2016

Supervisor: prof. dr. A.P.J.M. Siebes

Contents

Abstract

An American computer programmer and science fiction writer, Daniel Keys Moran, once said that although we can have data without information, we cannot have information without data [1]. This quote describes pretty much the whole concept around data mining and the significance of making sense out of data. The original motivation for this thesis arises from a digital humanities' problem. The aim of this project is to build an algorithm which can identify an evolving process between clusters of objects.

1 Introduction

We live in the era of data. Every day, 2.5 quintillion bytes of data are produced and stored. Data alone are of no use as they can provide only poor information. Making sense from the data available is a complex problem that spans many areas of research with applications in any discipline. It is estimated that more than 80% of the information is stored in text form [73]. Due to this explosion of the available data, more and more researchers focus on text mining. Text mining is a knowledge discovery technique that uses computational intelligence to discover previously unknown information by extracting information from written resources [50]. Text mining tasks include document classification [4], document clustering [5] and theme identification [6]. Topic models are statistical models that perform text analysis and text mining to a large unclassified collection of texts. They have received a lot of attention in the last decade. According to Google Scholar, the most popular topic modeling paper [7], written by D. Blei et al., has received 13,700 citations since 2003.

2 Topic modeling

No one likes their sayings to be pointed out as 'off topic'. Some online forums do not allow you to post an 'out of topic' comment in their websites. This reveals the importance of knowing the topic of a discussion. Topic is the subject of a conversation, what a newspaper article is about or the theme of a student's essay. Topic modeling represents a suite of algorithms that identifies topics in a collection of documents. The difference between topics and words is that words are observable throughout an article or a document, whereas topics are not. They are latent and harder to be identified. The resulted topics of topic modeling algorithms can be used to search, visualize and summarize large collections of texts. A variety of domains can profit from topic modeling, from Image Retrieval to Bioinformatics. There are several examples in literature in diverse fields including document summarization [8], text segmentation [9] and informational retrieval [10]. Topic models identify the 'hidden' latent structure behind a text by clustering the words using similarity. Topic refers to a collection (cluster) of words that frequently occurs together and represents the topic as a whole. An example of resulted topics from a topic modeling algorithm is illustrated in Figure 1. The corpus used was a 1.8 million collection of articles from the New York Times [11]. In this example, the words inside a topic make sense, such as "game team play" and "restaurant menu food".

One way to classify the algorithms used for topic modeling is according to the use of the "bag of words" assumption. As the name indicates, this class of algorithms treats the words of a document as being a group of objects in a woman's bag. We all know how messy this can be and how the objects are shuffled; hence, the order of the words in a document does not matter. The remaining algorithms do take into account the sequence of words during their analysis. Another distinction between topic models denotes the type of the used technique, whether it is supervised or not. A supervised method requires existing labels or annotations in the documents whereas in an unsupervised procedure, topics emerge from the analysis of the initial texts. In this paper, a distinction regarding the adoption of time assumption is presented. In section 2.1, topic models that do not consider the publication date of the documents ('static') are exploited whereas topics that accept time association ('evolution') follow in section 2.2.

Figure 1: Topics found by analyzing 1.8 million articles from the New York Times. [11]

2.1 Static topic modeling algorithms

An early topic modeling algorithm which remains popular was introduced by Papadimitriou et al. in 1998 [12] under the name Latent Semantic Indexing (LSI). LSI, also known as Latent Semantic Analysis (LSA) is a 'bag of words' and unsupervised technique. The model attempts to find similar documents (sharing a topic) by mapping both the words of the documents and the documents themselves into a "concept" space. Similarity between documents can be measured by using a distance metric, such as the cosine distance. Precisely, LSI starts by representing a corpus through a huge term-document matrix, the tf-idf matrix A. Tf–idf, short for term frequency–inverse document frequency, is a statistical measure that determines words in a collection of documents that are useful for defining the topic of each document [13]. Matrix A consists of rows which represent the words in the corpus and columns which denote each document. Each cell A[i,j] corresponds to the frequency of i term in the j document. Local and global weighting are applied due to the fact that the importance of terms varies within and among the documents. After the creation of the co-occurrence matrix, a mathematical technique named singular value decomposition (SVD) is applied to transform the document-term-matrix into a high dimensional space containing both documents and words concurrently [14]. SVD, first published by G. Golub et al. in 1965 [15], decomposes the matrix A into three new matrices:

$$A = T\Sigma V^T$$

where T is the m by $r = rank(A)$ term-topic matrix, Σ is the r by r singular value matrix and D is the n by r document-topic matrix [16]. Next, the algebraic method SVD achieves a reduction of the dimensionality of the matrix by isolating the singular values of A [17]. LSI interferes in this procedure by choosing a reduction factor, smaller than the rank of matrix A, namely K. Through this approach, terms and documents are converted to points in a smaller, k-dimensional space. However, there is no perfect choice for K and the proper way to decide the K value is still an open question.

LSI can also point out similarity between documents that do not have common words. This occurs when the documents share frequently co-occurring terms. The main drawback of LSI is that the method is not statistically oriented. LSI assumes that the data are normally distributed which may result in negative numbers inside the co-occurrence matrix, a bad approximation for frequency counts [18]. Another disadvantage of LSI approach is that the method faces difficulties in dealing with polysemy (a word having more than one meaning) and synonymy (different words sharing the same meaning). In addition, SVD is a time consuming procedure with vast storage requirements.

The LSI model was later used for the develop of probabilistic Latent Semantic Indexing (pLSI) by Hofmann in 1999 [19]. pLSI introduces the actual use of topics as it assumes that there is an additional level between words and documents: topics. pLSI is also an unsupervised method, meaning that the topics do not exist before the analysis. Topics are discovered from the texts, each topic consists of a list of words and the documents can then be classified according to the resulted topics. Similarly with LSI, the pLSI model depends on the frequencies of terms (words) in the documents. Although pLSI follows the idea of LSI, it is statistically oriented. pLSI defines a proper generative data model [20] and hence, overcomes the drawbacks of LSI. The algorithm estimates the probability of a word w belonging in a document d through simple Bayesian probability. The model treats the documents as mixture proportions of latent topics [21]. pLSI makes the following assumptions: a document d is generated with a probability p(d), a topic z is selected with a probability $p(z|d)$ and a word, w, is generated with a probability $p(w|z)$. The probability that a word is selected by a document is:

$$p(w, d) = \Sigma_z p(d)p(w|z)p(z|d)$$

The parameters and the topics are estimated using the Expectation Maximization (EM) algorithm [22]. The above equation can be seen as a matrix factorization similar to SVD where T matrix relates to $p(w|z)$, Σ to $p(d)$ and V to $p(z|d)$. The plate notation of pLSI can be seen in Figure 2 where the rectangles (plates) represent replication. M denotes the sequence of the documents $\{d_1, d_2, \ldots, d_M\}$ and N the words in each document d_i $\{w_{i1}, w_{i2}, \ldots, w_{iN}\}$. As depicted, a topic z depends on the document d and a word w, on the topic z. Furthermore, a word w is conditionally independent of d, given z. The variables d and w are the observed ones (shaded) whereas the variable z represents the latent, hidden variable (non-shaded).
The generative model of PLSI suffers from at least two critical problems [7] [23]. Firstly, since the probability distribution of each document on the hidden topics is estimated independently, the number of parameters grows linearly with the size of the corpus leading to overfitting problems [24]. Secondly, as the topics arise from the training documents, it is difficult for the model to be applied to new, unobserved documents [26].

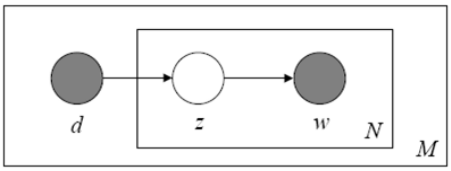

Figure 2: Plate notation for pLSI [25].

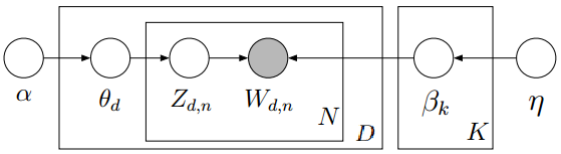

Figure 3: Plate notation for LDA [29].

In 2003, Blei introduced the Latent Dirichlet Allocation (LDA) which is perhaps the most common topic model currently in use [7]. Matthew L. Jockers, attempted to explain the gist of LDA without using any mathematical equation, via writing a vignette [30] which can be easily used as a bedtime story. The story takes place in a small mythical town where writers from over the world visit to seek inspiration for their novels and books. In this bizarre city, there is a place called the 'LDA buffet' which instead of food, serves words divided into several baskets (themes). Two writers, inhabitants of the town, borrowed their themes from the buffet and right after they completed their manuscripts, they got mugged by a thief. The robber had been banned from the buffet and wanted to find out the today's menu, the topics that were served and used by the writers that day. He already knew the amount of topics served (k) and wanted to discover which these topics were by looking at the stolen manuscripts. The LDA topic model tries to do exactly the same thing with the story's burglar, i.e.: identify the topics in a given corpus assuming that a fixed number of topics was used. The story continues with the thief shuffling the words of the scripts in a big jug and then dividing the words into k plates. After the random separation, the thief examines each word of the articles in the context of the other words that are distributed throughout each of the bowls and in the context of the manuscript from which it was taken. Then, based on calculated probabilities (more on that later) he decides whether the world should be moved in another plate or not. Every time that he considers a word, he assumes that all the other words are correctly distributed in the bowls. This procedure should be repeated until the assignment of the bowls do

not change meaning that the topics are correctly recognized. The output of the robbery would not only be the resulted topics (bowls) but the ingredients of the documents as well. For example, an output could be that story 1 consists of 40% of topic A whereas this topic constitutes the 50% of story 2.
In more detail, LDA is an extension of pLSI and therefore a topic is assumed to be a probability distribution over a fixed vocabulary of words and the documents consist of a mixture of topics. However, LDA handles the mixture topic proportions of a document as hidden variables drawn from a single Dirichlet (Dir) distribution, instead of actual parameters of the model. Consequently, LDA overcomes the overfitting problem of pLSI and has the ability of calculating probabilities for new, emerging documents. The difference between these models can be seen in their plate notation as well. The plate notation of LDA is depicted in Figure 3. Again, the shadowed node represents the observed variable of the model, the word, whereas the non-observed ones the latent variables. As shown, there is an extra layer in the model where D denotes the collection of documents and K the specified number of topics. Let V be the size of the vocabulary, $\vec{\alpha}$ a positive K-vector and $\vec{\eta}$ a positive V-vector. Then, the $Dir(\alpha)$ denote a K-dimensional Dirichlet and $Dir(\eta)$ a V-dimensional Dirichlet. The hidden topical structure is represented by: the topics β_k, the per-document topic proportions θ_d and by the per-word topic assignments $z_{d,n}$. The Dirichlet distribution is given by:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i \theta^{\alpha_i - 1}$$

The Dirichlet distribution is specified by the positive K-vector parameter α and Γ denotes the Gamma function:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

Blei et al. [7] chose the Dirichlet distribution based on its properties that support the parameter estimation as well. These properties include that Dirichlet is an exponential distribution, it has finite dimensional sufficient statistics and is conjugate to the multinomial distribution.
The generative process of LDA can be summarized as [29]:

1. For each topic k, draw a distribution over words $\beta_k \sim Dir(\eta)$.
2. For each document,
   (a) Draw a vector of topic proportions $\theta_d \sim Dir(\alpha)$.
   (b) For each word,
      i. Draw a topic assignment $z_{d,n} \sim Mult(\theta_d)$.
      ii. Draw a word $w_{d,n} \sim Mult(\phi_{z_{d,n}})$.

The LDA model assumes the above generative process for a corpus and then tries to backtrack the procedure in order to find the topics in the collection of the documents (thief's technique). The inferential problem of LDA is to calculate the posterior distribution of the hidden variables given a document and the corpus parameters α and β [28] using the following equation [7]:

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)}$$

As exact inference of the posterior distribution is challenging, various approaches for approximate inference can be used such as Laplace approximation, variational approximation, and Markov chain Monte Carlo (also known as Gibbs sampling [27]). Via this process, the probability of each topic per word is calculated. The assigned topic for each word is simply the most likely topic. This procedure is repeated until it converges. During the loop, at each iteration, the algorithm assumes that all topic assignments except for the current word (to be assigned) are correct in each iteration.
The main drawback of the LDA model is that it is unable to model any relationships between topics. In the Dirichlet distribution, the components are independent of each other and there are no relationships among topics [31]. As a consequence, topics cannot share words, i.e.: the same

word cannot occur in more than one topic.

The later topic models are generally extensions of LDA, such as Correlated topic model (CTM). CTM improves the LDA technique by modeling also the correlations among topics and was introduced by Blei et al. in 2007 [32]. The model assumes a logistic normal distribution in order to model the relations among topics. The CTM follows the same generative process with LDA except that the topic proportions are drawn from a logistics normal instead of a Dirichlet distribution. The plate notation of CTM is shown in Figure 4. However, the added flexibility of the CTM comes at a computational cost [32] as the CTM requires a lot of calculations. Additionally, existing variations of LDA and LDA model itself, can only deal with a small corpus and model a limited number of topics [33]. For a massive collection of documents containing a large number of topics, LDA algorithms are often inefficient [34].

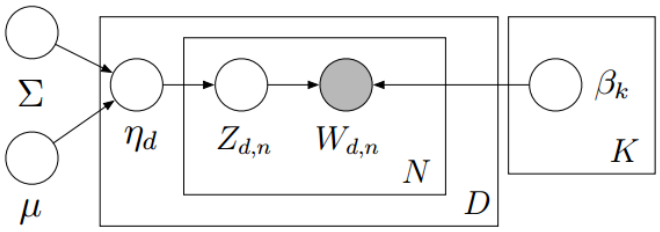

Figure 4: Plate notation for CTM [29].

2.2 Evolution topic modeling algorithms

A new cluster of algorithms that has attracted much attention is the evolution topic models. The idea behind the topic evolution models is that topics evolve in time and analyzing articles that differ a lot in their publication dates can cause problems when defining the topics. Using these models, a researcher can comprehend the evolution of the topics and how a topic has changed over time. The first paper among these models was published in 2006 by Wang et al., under the name Topic Over Time (TOT) [35]. In this model, each topic is a fixed word distribution and the algorithm models the probability that a document includes a topic based on the publication date of the document. In the original paper, the authors describe the generative process of the model in two ways. The plate notations of these two alternatives are depicted in Figure 5. As shown, both methods follow the LDA notation. The generative process of the first notation (a) can be summarized as:

1. For each topic z, draw a Multinomial distribution $\phi_z \sim Dir(\beta)$.
2. For each document d, draw a Multinomial $\theta_d \sim Dir(\alpha)$.
3. For each word $w_{d,i}$ in document d,
   (a) Draw a topic assignment $z_{d,i} \sim Mult(\theta_d)$.
   (b) Draw a word $w_{d,i} \sim Multi(\phi_{d,i})$
   (c) Draw a timestamp $t_{d,i} \sim Beta(\psi_{z_{d,i}})$

In the above process, a timestamp is generated for every word of the documents. However, all words that correspond to the same document have the same timestamp as well. The other generative process of the TOT algorithm is presented in Figure 5 (b). Instead of generating a timestamp for each word, one timestamp for each document in the corpus is generated.

TOT is an LDA-style topic model where a topic is associated with a continuous distribution over time without any Markov assumption or discretization of time. The algorithm builds a narrow-time-distribution topic when a word co-occurrence occurs for a short time, whereas it creates a broadtime-distribution topic for a long existing word co-occurrence.

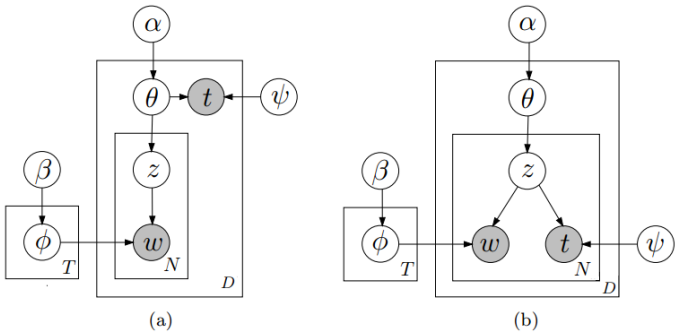

Figure 5: Plate notation for TOT [35].

Similar to LDA, the distributions are intractable for exact inference and hence, the authors used Gibbs sampling for approximate inference. An example output of the model would be a diagram over time where we 'track' a chosen topic and observe its occurrences throughout time. The drawback of TOT model is the assumption that topics and their meaning are constant over time which concludes that a change in a topic involve a change in topic co-occurrence instead of changes in the word distribution belonging to each topic. Therefore, we cannot see any evolution of the topics.

Later this year, Blei et al. proposed another evolution model based on the LDA approach, the Dynamic Topic Model (DTM) [36]. The algorithm organizes the documents on (disjoint) time slices and hence, a topic is a sequence of distributions over the words. Each time slice is modeled by a K-component LDA topic model in which the k-th topic represents a smooth evolution of the k-th topic at the previous time slice. For the detection of the evolution sequence, the model uses Gaussian (Normal) distributions, specifically, the model chains the parameters for each topic k ($\beta_{t,k}$) at different time slices as being evolved with a Gaussian noise from its predecessor:

$$\beta_{t,k}|\beta_{t-1,k} \sim N(\beta_{t-1,k}, \sigma^2 I)$$

Each topic is a sequence of distributions over words. In the LDA model, topic proportions, θ, are drawn from a Dirichlet distribution. In DTM, the authors use a logistic Normal distribution again in order to express the uncertainty concerning the topic proportions:

$$\alpha_t|\alpha_{t-1} \sim N(\alpha_{t-1}, \delta^2 I)$$

The generative process of the DTM in slice t can be summarized as:

1. For each topic z, draw a Normal distribution $\beta_{t,k}|\beta_{t-1,k} \sim N(\beta_{t-1,k}, \sigma^2 I)$.
2. For each topic z, draw a Normal distribution $\alpha_t|\alpha_{t-1} \sim N(\alpha_{t-1}, \delta^2 I)$.
3. For each word document d,
   (a) Draw $\theta_d \sim Dir(\alpha)$.
   (b) For each word w,
      i. Draw a topic assignment $z_{d,i} \sim Mult(\theta_d)$.
      ii. Draw a word assignment $w_{t,d,n} \sim Mult(f(\beta_{t,z}))$.

π simply maps the multinomial natural parameters to the mean parameters. The graphical representation of the DTM is shown in Figure 6. Notice that, when the horizontal arrows are removed, every slice is a separate LDA model. Similarly with LDA, estimating the parameters of the modes constitutes an inference problem. The authors apply variational methods, in particular, the variational Kalman filtering and the variational wavelet regression for approximate inference. An example output of the algorithm could be a graphical representation of a chosen topic where the top-ranked words would be displayed. That way, the changes in word's popularity could be recognized. The disadvantage of this model is that it assumes that there is a fixed number of

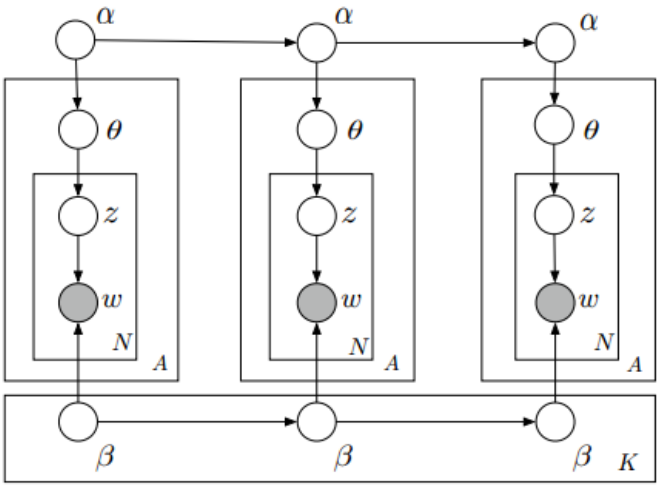

Figure 6: Plate notation for DTM [36].

topics in every time slice which is quite unrealistic. Topics can emerge or die or appear with a different frequency such as Olympics (every 4 years) or Asia Cup (every 2 years).

The most recent topic evolution model approach tries to add information about the topic evolution from the document itself. Scientific papers include a value information in their citations and the papers they refer to. Similarly, online articles such as blogs or twitter posts contain data in the form of hyperlinks. An evolution topic model that takes into consideration the citations of the papers was published in 2001 by Jo et. al [37]. The authors state that citation supports the topic detection as well as the discovery of the topic evolution. If a document d_i cites a lot of papers from topic B, it is likely that it contains the topic successor of topic B. The model was not named by its creators, in this paper is called Citation Evolution Topic Model (CETM) for convenience. The CETM tries to discover the topology of topic evolution inherent in a corpus. A topic is defined as a quantized unit characterized by evolution changes, i.e.: changes in topic's word distribution. The algorithm starts by looking at the collection of the documents in chronologically order and selects whether the document introduces a new topic or not. A new topic is identified as the specified content cannot be explained by any of the found documents as well as that is being used by a sufficient amount of following documents. CETM uses the mixture of word distributions and a word distribution θ is drawn from a multinomial distribution over the vocabulary V of the corpus. In the learning phase of the algorithm, the LDA topic model is integrated while Lagrangian multipliers are also used for inference. After the discovery of the existed topics, the CETM builds a graph having as nodes the topics and the edges of the graph demonstrate the citation-relation of the topics. An edge is created between two topics if they are semantically related with a probability greater that the random chance. In the output graph the speed of the evolution of each topic can be absorbed as well as the stability of a topic. However, the method does not consider the direction of the relationships between the documents concluding that a milestone paper would not be recognized by the algorithm . The main drawback of the TEM arises from the use of the citation of the documents. This research has focused on modeling scientific papers thus it is not applicable to other data sets.

3 Clustering

Clustering is a method extensively used in the domain of data mining. It is defined as the process of finding groups of similar objects in the given data [39]. A similarity measure is used to assess the similarity between the objects. Cluster analysis should not be confused with classification as the latter requires predefined classes in the data (labeled data). Hence, document clustering refers to the unsupervised technique of clustering documents/texts/sentences etc. Document clustering

can be beneficial regarding documents retrieval [40] and supporting browsing [41]. In addition, clustering can handle the document clusters to provide summary insights into the overall content of the underlying corpus [65]. Consequently, clustering is strictly related with topic modeling. In this section, the main types of clustering [66] are described and a translation from topic modeling to clustering follows.

3.1 Hierarchical vs. Partitional clustering algorithms

One of the most popular categories of clustering algorithms is the hierarchical clustering [42]. The idea behind hierarchical clustering is that, depending on the agreed level of detail, the clustering algorithm can further split the resulted clusters into smaller (or merge into larger) ones. Hierarchical clustering produces a conceptual hierarchy and thus is distinguished as a nested sequence of partitions [43]. The output of such an algorithm is a dendrogram, a tree-like structure which represents the nested clusters. Hierarchical clustering is then divided into two categories: agglomerative and divisive [44]. Assume that N items are to be clustered. The algomerative clustering (aka bottom-up) starts with N clusters, each of them containing a single item. The final clusters are formed after some merging operations. Specifically, at each iteration, the two more similar objects (groups) are merged. The procedure is repeated until all the objects are merged into one single cluster. On the contrary, the divisive clustering (aka top-down) initializes all the N items in one cluster and iteratively divides the cluster(s) until N clusters are created. From a computational perspective, the divisive hierarchical clustering is more problematic than the agglomerative as it requires all possible divisions of the constructed clusters into subsets to be taken into account. In Figure 7 the dendrogram of these two approaches is shown. Hierarchical

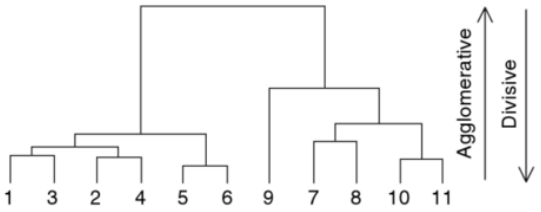

Figure 7: Hierarchical clustering dendrogram. [45]

clustering does not require the number of clusters to be selected in advance [46]. The join (or splitting) of the clusters at each step is based on a distance measure (linkage) that comprehends how similar or not the clusters are. The intergroup similarity can be defined with a variety of ways given a distance measure between the objects (points) of the clusters. The four well-known metrics to measure the cluster similarity (distance) are described below [47]:

- Single linkage

The distance d between two clusters is defined as the minimum distance between the cluster objects. The shortest distance of any item of one cluster (A) from any member of the other cluster (B) [48] is also known as Nearest Neighbor Cluster Distance:

$$d(A, B) = \min_{x \in A, y \in B} |x - y|$$

The single linkage is a widely used and simple method with efficient implementations. However, the algorithm is likely to fail on detect clusters with irregular shapes and is sensitive to outliers in the data set. In addition, the metric can produce 'chaining' where close objects belonging to different groups can lead to early merging of those groups.

- Complete linkage

The distance d between two clusters is defined as the maximum distance between the cluster objects. The longest distance of any item of one cluster (A) from any member of the other cluster

(B) [49] is also known as Furthest Neighbor Cluster Distance:

$$d(A,B) = \max_{x \in A, y \in B} |x - y|$$

Complete linkage overcomes the chaining' problem but is yet sensitive to outliers. The distance used may as well be considered as the diameter of the new cluster (maximum within-cluster distance).

- Group average

The distance d between two clusters is defined as the average distance between the objects of the clusters:

$$d(A,B) = \frac{1}{|A| \cdot |B|} \cdot \sum_{x \in A, y \in B} d(x,y)$$

This method appears as a compromise of the two above ones. It has been proven to be efficient and widely used in experiments. Nevertheless, is it computationally expensive for large data sets as the computation of the average similarity between a large number of pairs is required.

- Centroid linkage

The distance d between two clusters is defined as the distance between the cluster centroids C_A and C_B, where the centroid is the center of a cluster, the average across all the objects:

$$d(A,B) = |C_A - C_B|$$

This metric appears to be robust to outliers in the data set. A drawback of the method is that the centroid of each cluster moves as more clusters are merged. Hence, the distance of different clusters may decrease making the results difficult to comprehend. The choice of the cluster distance metric defines the clustering strategy and hence the properties of the resulting clusters and the dendrogram itself [50]. This reveals the importance of the metric used while the pros and cons of each method should be taken into account by the user.

In contrast, patritional (or partitioning) clustering [51] creates only one set of clusters given a parameter k denoting the desired number of clusters [52]. This method yields a single partition of the input data into k non-overlapping clusters, which are constructed due to global and local criteria applied by the algorithm. The algorithm starts with an initial cluster partition which is iteratively improved until the criteria are reached. Typically, the global criteria involve maximizing some measure of *similarity* within each cluster, while maximizing the *dissimilarity* of different clusters [53].
The most commonly used and the classical example of partitional clustering method is the k-means algorithm and its many variants [54]. The main idea is to create k centroids, one of each cluster. The position of the centroids is crucial and they are initialized as far from each other as possible. The centroid is the mean point (center) of each cluster. Each point is then assigned to a cluster according their similarity (e.g.: distance) to the cluster centroid. Next, the new centroid of each cluster is computed based on the new assigned members of the cluster and the procedure is repeated until the process converges, i.e. the elements of the clusters do not change in two sequential iterations. The algorithm aims at minimizing the following objective function:

$$\sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i - c_j||^2$$

where n is the number of objects in the data set, x_i a data point and c_j the centroid of a cluster. The k-means is a relatively efficient algorithm which has been used in literature for many years. However, the results critically depend on the initial choice of cluster centroids [55].

3.2 Hard vs. Overlapping Clustering algorithms

Partitional clustering can be later divided into hard and overlapping Clustering. Hard clustering generates a hard partition where each data element is assigned to one and only one cluster. Hard clustering is also known as exclusive clustering and is generally simpler and with a lower time complexity than the overlapping clustering algorithms [56]. As defined [57], a hard partition of a data set Z is a set of subsets $\{A_i | 1 \leq i \leq c\}$ with the following properties:

$$\bigcup_{i=1}^{c} A_i = Z$$

$$A_i \cap A_j = \emptyset,\ 1 \leq i \neq j \leq c$$

$$\emptyset \subset A_i \subset Z,\ 1 \leq i \leq c$$

First equation shows that the union of all subsets A_i equals Z, all the items. Next, these subsets should be disjoint and the last property indicates that none of the subsets is empty nor contains the whole items. An example where hard clustering would be suitable is the case of blood type clustering. If we were about to cluster people regarding their blood type, each of them should be assigned into one and only cluster.
In terms of membership functions, hard clustering can be formulated as:
Let μ_{ik} be the membership function of the ith subset A_i of Z for the cluster k. The elements should satisfy the following conditions:

$$\mu_{ik} \in \{0, 1\},\ 1 \leq I \leq c,\ 1 \leq k \leq N$$

$$\sum_{i=1}^{c} \mu_{ik} = 1,\ 1 \leq k \leq N$$

$$0 < \sum_{i=1}^{c} \mu_{ik} < N,\ 1 \leq I \leq c$$

As shown, the membership function is either 0 or 1 while for each element, the sum over all the clusters equals 1, i.e.: it can only belong to one cluster. The last condition illustrates that there is none cluster with 0 or N (all) elements.

In contrast, there are a lot of situations where an object belongs to more than one cluster. Hence, an overlapping or non-exclusive clustering which allows one element to belong to more than one cluster simultaneously feels more natural as the world is not just black and white [58]. For example, a data point referring to a person can belong to the clusters 'Students' and 'Employees' as he/she can study at a University and work at the same time. A visual representation of hard and overlapping clustering is illustrated in Figure 8. The clusters in each picture are indicated by dotted circles and in case (a) a hard clustering is displayed whereas in (b) an overlapping one.

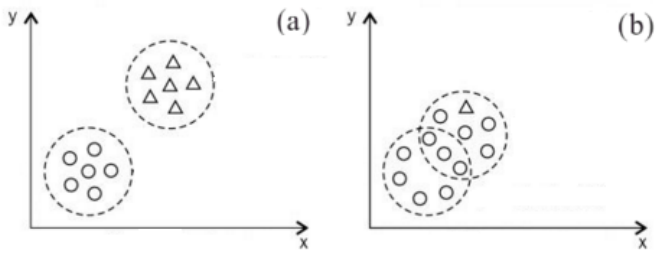

Figure 8: Hard vs. overlapping Clustering. [59]

Overlapping clustering is also known as soft or fuzzy clustering [60]; characterizing a clustering type with proportions. In fuzzy clustering, each element has a *degree* of membership for each cluster between 0 (absolutely does not belong) and 1 (absolutely belongs). Mathematically, a

fuzzy set S is characterized by a membership function which associates each point x in the data set to a real number in the interval [0,1] which represents the degree of membership of x in S [61]. Hence, the difference is that the membership function can be any number in the interval [0,1] instead of the set {0,1}.

$$\mu_{ik} \in [0, 1]$$

$$\sum_{i=1}^{c} \mu_{ik} = 1,\ 1 \leq k \leq N$$

$$0 < \sum_{i=1}^{c} \mu_{ik} < N,\ 1 \leq I \leq c$$

Fuzzy clustering can be also found as probabilistic clustering in the literature. Again, the sum of the membership weights (probabilities) of each data element must sum to 1.
The most famous fuzzy clustering algorithm is the Fuzzy c-Means algorithm. The algorithm is very similar to k-means and aims to minimize the following:

$$\sum_{i=1}^{c} \sum_{k=1}^{n} \mu_{ik}^{m} \cdot ||x_i - c_j||^2,\ 1 \leq m < \infty$$

where c is the number of subsets, clusters, and N the size of the dataset. μ_{ik} denotes the membership function and x_i, c_j the data points and the centroids of the clusters respectively. In addition, m, is a parameter determining the fuzziness of the resulting clusters. When m is large, the memberships are smaller resulting to fuzzier clusters. When m in 1, the memberships converge to 0 or 1 resembling the hard clustering. In the absence of experimentation or domain knowledge, m is commonly set to 2 [62]. At each iteration, Fuzzy c-Means calculates also the coefficients of being in the clusters for each point and stops when the centroids coverage.

3.3 Complete vs. Partial Clustering algorithms

The complete clustering algorithm requires every data element to be assigned to a cluster when the algorithm terminates [63]. In mathematical terms, complete clustering is a hard clustering method. Similarly, the set of subsets $\{A_i | 1 \leq i \leq c\}$ of a dataset Z should satisfy the same equations with hard clustering (section 3.2). An example where complete clustering is needed could be for the arrangement of the different student groups for a lab session. The professor wants to assign each and every student to a group.

On the other hand, a partial clustering algorithm allows some objects not to belong to an existing cluster. Through this technique, outliers or noise in the data set do not influence the resulted clusters. The motivation for this clustering is that some elements of a data set can be one-of-a-kind and cannot be similar with any other object. Thus, the union of the subsets does not equal the dataset:

$$\bigcup_{i=1}^{c} A_i \subseteq Z$$

For example, in a notable episode from Seinfeld [64], the character named George reveals to his future wife that he wants his first child to be named 'Seven'. Imagine a data set with names as elements which are to be clustered. The name 'Seven' would (probably) denote an outlier which could be avoided with a partial clustering technique. In Figure 9, a visualization of the difference between complete (a) and partial (b) clustering is presented.

4 From topic modeling to clustering: a translation

A specific document belongs to a given cluster with a probability θ. Topic modeling attempts to model the value of that probability [65]. That way, the various topics of a corpus can be considered as clusters and the documents as the elements to be assigned. Therefore, topics are analogous to

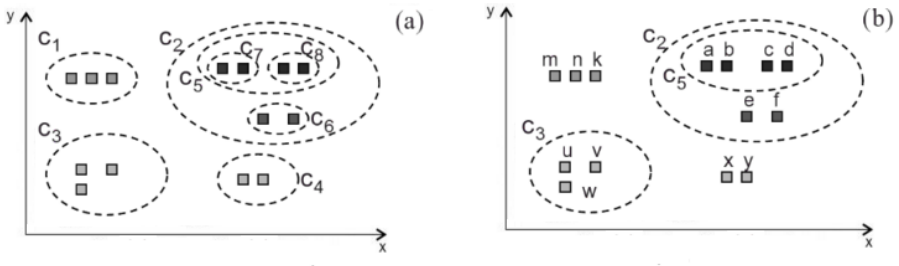

Figure 9: Complete vs. Partial Clustering. [59]

clusters and the amount of clusters is the same with the number of topics (usually denoted by k). In topic modeling, a document is a mixture of topics, i.e. a document can consist of more than one topic. This can be translated in cluster analysis by the use of soft clustering where a document is allowed to appear in multiple (overlapping) clusters. To conclude, the discovery of topics in a corpus that topic modeling algorithms achieve can be done by applying a clustering algorithm in the collection of the documents.

5 Frequent Itemset Mining

Frequent Itemset Mining (FIM) is an essential and interesting branch of data mining and data analysis. FIM searches for regularities in a database in order to extract useful information. The method attempts to discover interesting patterns that best describe the data, such as association rules or correlations. The original motivation for FIM came from the need to analyze the 'shopping basket', a research on customer's behavior regarding their purchased products [67]. An example of an association rule is that if a customer buys **eggs** and **pizza**, he will also buy **beer** with a probability of 60%. Possible applications of the resulted rules are: improvement of store layout, promotion of products, cross-selling (suggestion of other products) and many others.

The FIM problem considers a set of distinct items $I = \{i_1, i_2, ..., i_n\}$ and a transactional database D. A transaction $t \subseteq I$ contains items that correspond to the purchased products a costumer bought. A pattern X is a set of items (itemset) and is contained in t iff $X \subseteq t$. The *support* of a pattern X is the number of the supersets of X in D, i.e.: how many times these products are bought together. An itemset is then considered as *frequent* if its support is greater that a given threshold, *minsup*.

A major problem in Frequent Itemset Mining is the explosion of the number of resulted patterns due to the difficulty of finding only the most interesting sets. Mining the database with strict constraints leads to few patters that usually are of no use as they are the most obvious (common) ones. In contrast, losing the constraints produces an enormous number of resulted patterns. Fortunately, many approaches using different means have been inventing regarding this problem.

5.1 KRIMP in general

To tackle the problem of pattern explosion, Siebes et al. [68] proposed an algorithm that uses the MDL principle, i.e.: the best set of frequent itemsets is the set that compresses the database best. Later that year, the above compression method was named KRIMP by Leeuwen et al. [69]. The name of the algorithm was inspired by the Dutch word 'krimp' which means 'to shrink'. MDL stands for minimum description length and is capable of finding the model that describes the database best [70]. More formal, the MDL principle can be described as follows: Given a set of models H, the best model H $\in H$ is the one that minimizes

$$L(H) + L(D|H)$$

where $L(H)$ is the length, in bits, of the description of H and $L(D|H)$ is the length, in bits, of the description of the data when encoded with H.

The models considered by the algorithm are code tables. A code table consists of a two-column table in which the left-hand side contains itemsets while the right one, a code for each of the itemsets. The algorithm gives a unique encoding to each possible transaction and aims to find the one code table which compresses the whole database best. KRIMP follows a heuristic and simple greedy search strategy [71]:

1. Start with a standard code table ST, containing the singletons only of itemsets $X \in I$.
2. Add the itemsets from one by one. If the resulting codes lead to a better compressed size, keep the code. Otherwise, discard the code.

It has been proven that KRIMP achieves a dramatic reduction in the number of resulted itemsets and picks the itemsets that matter most [71]. The output of the algorithm is the optimal code table for the given database.

5.2 How does KRIMP fit in?

Apart from compression, code tables can be used for classification as well. The KRIMP classifier performs excellent comparing to the best classifiers available [71]. Furthermore, KRIMP can also be used for clustering purposes as shown in 2009 by van Leeuwen et al. [72]. The clustering problem can be formulated as follows: Let I be a set of items and db a bag of transactions over I. Find a partitioning $db_1, ..., db_k$ of db and a set of associated code tables $CT_1, ..., CT_k$ such that the total encoded size of db

$$\sum L\big(CT_i, db_i\big)$$

is minimized. The authors describe the goal of the problem as finding the mixtures of samples from different distributions. The different distributions denote the different buying patterns that occur in a transactional database. In the paper, it is concluded that the optimal decomposition can be driven from the original code table CT. In order to find that partitioning of the database, the proposed algorithm does not consider any distance metric and no parameter has to be set. In fact, KRIMP algorithm is applied to the whole dataset that results in a CT and a heuristic procedure follows. The heuristic finds the set of the subsets of the CT that minimizes the total encoded size of db, denoting the clusters.

6 Our problem: evolving clusters

The original motivation for this thesis arises from a digital humanities' challenge. Digital Humanities are defined by the Digital Humanities Quarterly journal as

> "a diverse and still emerging field that encompasses the practice of humanities research in and through information technology, and the exploration of how the humanities may evolve through their engagement with technology, media, and computational methods."

Historians expressed the need of analyzing themes or topics in a corpus over time using digital humanities tools. Until now, topic modeling algorithms have been mostly used in order to identify the topics in each epoch. The term 'epoch' indicates a period of time lasting one year or more. After the detection of topics in each epoch, historians tended to research these topics and tried manually to spot any differences or similarities with a view to understand their evolution. Not only the notion of doing things "manually" irritates the computer science family, but an algorithm widely applicable to any data set is preferred.As discussed in section 2, static topic modeling algorithms have various drawbacks that make them unsatisfactory about the defined problem. Each of the presented static topic models constitute an improved version of the previous one. The last one, a variation-extension of LDA technique does not perform sufficient enough in a quite large data

set. Likewise, the evolution topic models are not a suitable solution to this problem either. The existing evolution topic models lack of flexibility and can only be sufficiently applied in specific occasions. They expect data sets with particular properties, such as containing a fixed number of topics, a number of references, to mention but a few.

A solution to handle this problem was inspired by the parallelism between a transactional database and a corpus. To conclude, the research question of this thesis is:

"Given a set of items I and a database db over I divided into epochs, determine the clusters in db and how they evolved over time."

7 KRIMPevol algorithm

In the previous sections, a literature review on the existing methods that are currently used in order to identify evolution in a database, was performed. This section includes a detailed description of the proposed algorithm, KRIMPevol; a mean to identify an evolving process between clusters of objects that belong to different time intervals. Furthermore, the implementation and the experiments performed in order for its validation and the conclusions drawn follow.

The name that was selected for the algorithm indicates its ancestor -the KRIMP algorithm- and emphasizes its capabilities regarding the evolution process. But, what is meant with the word 'evolution'? Clusters are groups of similar objects that share some characteristics. An object can be anything: from a person to a word, from a planet to ant species. Clusters can exist today and tomorrow, they can be found in the present, in the past or in the future. Therefore, clusters of the same population can exist in different time intervals. It would be of interest to zoom into a cluster and see how it evolved through the years, i.e. the changes of its components from time to time. This is when KRIMPevol steps in. As input, KRIMPevol algorithm takes a database over a set of objects (items) that are divided into epochs. With the term epoch, a fixed period of time with a specific duration is assumed. The desired output of the algorithm is a graph containing the identified clusters *within* each epoch and the correlations between the clusters *over* the epochs. The pseudo-algorithm of the algorithm KRIMPevol follows:

Input: *db* (database), *dur* (duration of each epoch)
Output: diagram containing the clusters and their correlations

Algorithm KRIMPevol

1: split the db into epochs of duration *dur*
2: **for** each epoch **do**
3: apply KRIMP clustering - find the best set of clusters
4: **for** each cluster **do**
5: compute dissimilarities between clusters
6: correlate the clusters depending on their dissimilarities

At step 3, the optimal partition of each epoch is determined using the algorithm build by van Leeuwen et al. in 2009 [72]. The iterative algorithm uses the MDL principle and, without any prior knowledge, splits the data randomly in a fixed number of parts representing the clusters (k). Then, the KRIMP compressor is applied to each part and the transactions are re-assigned to the specific compressor that encodes the transaction best (shortest). The algorithm attempts to cluster the database for all the possible numbers of clusters within the range $[2,|db|]$ where $|db|$ represents the size of the database, i.e. the number of transactions. Each time, the found total encoded size (dbsize) is compared with the best found so far and thus the best size and best k is determined. Due to the fact that each transaction is randomly initialized, each experiment was repeated 10 times and the smallest dbsize found was chosen as the output (as proposed by the

authors).

At step 5, the dissimilarities between the clusters are calculated. A cluster, is a database part which has its own descriptive codetable (CT). Therefore, in this phase of the algorithm, the dissimilarity measure implemented by Vreeken et al. [74] can be used. The generic and symmetric dissimilarity measure between two databases, x and y, is defined as the maximum of two mirrored aggregated code length difference measurements through the equation:

$$DS(x,y) = max\left\{\frac{CT_y(x) - CT_x(x)}{CT_x(x)}, \frac{CT_x(y) - CT_y(y)}{CT_y(y)}\right\}$$

A small dissimilarity shows a strong correlation between the two databases as the two databases are more or less the same, whereas higher scores indicate greater dissimilarity.

At step 6, the correlations of the clusters are drawn. In the previous step, the dissimilarities of clusters over the (different) epochs have been calculated. In this step, the internal dissimilarities of the clusters belonging in the same epoch are also needed. Theoretically speaking, when two clusters from two different epochs have a dissimilarity smaller than the minimum dissimilarity of each cluster in its own epoch (internal), they are related and they indicate an evolving process. If the dissimilarity is bigger, they are irrelevant.

Experiments were performed in order to test and validate the proposed algorithm. The focus was mainly in the two steps: step 3 and 6 respectively which are described in sections 9.1 and 9.2.

8 Experiments - First set

8.1 Database

In order to test the proposed algorithm, we first attempted to perform the experiments using a real database. As the motivation for creating the algorithm was the identification of themes -cluster of words-, a database of categorical data was chosen. A database which contains the registered names of the Dutch citizens from 1889 onward was provided by Gerrit Bloothooft. More precisely, each row in the db, represents a person (entity) where each entity holds four attributes, namely: 'family ID', 'name', 'gender' and 'year'. The first attribute is unique across all the families and can be used to indicate the siblings across the db as they share the same number ID. The column name contains only the first name of each person followed by the gender (M/F) and the year of birth. For the experiments of this thesis, the column gender was ignored. An example row of the Names db is: *44 Marcus M 1966.*

8.2 Pre-processing

The purpose of the pre-processing phase, is to create a database of names similar to a transactional db. As known, in a transactional db, each row represents a costumer and its purchased products, his/her shopping options. Having this in mind, a parallelism can be found between a customer and his/her products with a family and their chosen children's names. To create this transactional-like database, the column of family ID was used as an indicator to merge the names of all the children of each family in one row. For each family, the year of birth of the firstborn child was saved to be used for the next pre-processing step.

As described in section 7, the KRIMPevol algorithm splits the database into epochs of a fixed duration. Using the year of birth of the eldest child in each family, the rows of the new database were sorted and split into smaller databases (epochs). Each epoch was chosen to have a duration of 10 years. This division resulted in 13 epochs with a decade duration whereas the last one contained the remaining 5 years. Moreover, the categorical names were coded in order to create a database comprised by integer numbers. To conclude, the last phase of the pre-processing included a cleaning process regarding the year of birth. Eight entries were found to have a year of

birth equal to zero. The first one belong to a family in which the second child was born in 2001. Therefore, the zero year of birth was obviously a mistake and was replaced with 2000. However, this was not the case for the other 7 persons. As the correct year of birth could not be indicated, the seven rows that had a mistaken year of birth were deleted.

An example row of an epochs is: *13 1988 846 855 998*. The first number, 13, is the family ID, the year of the firstborn child follows and then, the coded names of each child. The specific family has three children. In the version of epochs which was used for the experiments, the information containing the family ID as well as the year of birth were deleted.

8.3 Statistics

The Names db consists of 12,899,245 entries of four attributes. The minimum year of birth is 1889 whereas the maximum 2015. There is one entity registered in 1889 and there are six youngest citizens, born in 2015. The maximum number of children of the same family is 22 and the first child of that family was born in 1940. The identified number of the different families in the whole database is 4,379,762 whereas the amount of different names 253,468. It came as no surprise that the most common name in Netherlands, over the past 130 years is 'Maria'.

8.4 Results

Having the epoch databases ready to use, the algorithm KRIMPevol was applied. Unfortunately, the results were not as expected. The clustering algorithm could not identify any possible clustering within the epochs and the best number of clusters found (k) was always equal to $|db|$, the number of rows of each epoch. Even with bigger or smaller epochs, the result was the same. An example output of epoch 3, which was found during the experiments can be seen in Figure 10. As shown, the size of the database decreases as the number of clusters grows. At the last tested number of clusters, $|db|$, the smallest size of the database is achieved.

Figure 10: Clustering results - Epoch 3

Our mission was then redefined: why this approach is not functional? At first, the implemented algorithm was tested. With a view to validate the clustering algorithm, the database mushroom, borrowed form UCI repository, was used. The specific database was also used in [72] where the reported best k was 20. In our experiments, we also found that the smallest size of the db was achieved when the db was clustered in 20 parts. Not only the dbsize did not dropped as the number of clusters was increased, but the size of the database for k=20 was 38 times smaller than the dbsize for k=$|db|$. Next, the duration of the epochs and therefore the amount of rows were considered. The results were the same either with a duration of one year or fifty.

Hence, the problem arises from the form/type of the database. What are the differences between a transactional db and the used ones? After a lot of thinking and trials, the alphabet length of the database and the average length of the database rows were found responsible. In Table 1, a comparison between transactional databases and the Names db are shown. Again, three databases that were also used in [72] are compared.

db	nR	avgL	aL	%
Adult	48,842	15	95	0.01
Mushroom	8,124	23	117	0.06
Nursery	12,960	9	21	0.02
Names	4,379,762	3	253,468	1.93
Example epoch	559	2.58	217	15.05

Table 1: Characteristics of the databases

In Table 1, nR refers to the number of rows (transactions), avgL to the average length of each row and aL to the alphabet length of the database, the number of items (levels). The last column is the percentage of the unique items (items used only once) and help us to compare the databases. As shown, the percentage of unique items is really small in the transactional databases, whereas in the Names db is 100 times bigger. When the database is split into epochs, the unique items are even more as the amount of people gets smaller. Another big difference, is the average length of each row. In the Names db, only the siblings of each family are known which significantly narrows the database.

After the conclusion that the Names db cannot be used with KRIMPevol, the quest for a new working database began. Although it was hard to find a database with the specific requirements (transactional-like, over the years), the exploration was fascinating. Throughout this search, very interesting databases came across our path, such as a database containing the last words of every inmate executed since 1984 in Texas [77] or information on how Americans have met their spouses and romantic partners [78]. After the long search, the best option to overcome the encountered problems is to work with an artificial database. In the next section, the process followed to create such a database is described.

9 Experiments - Second set

The key to overcome the absence of an over-the-years transaction-like database was found in a scientific paper. Vreeken et al. [75] proposed an algorithm that generates a synthetic database given a database and its code table. Consequently, a database over the years can be constructed using a transactional-like as the starting one and creating the latter epochs using this algorithm. Synthetic data are defined as "any production data applicable to a given situation that are not obtained by direct measurement" according to the McGraw-Hill Dictionary of Scientific and Technical Terms [76]. The algorithm [75] was published in 2007 with the original purpose of preserving sensitive information in the data, a process of data anonymization. It follows the principles of KRIMP algorithm [68] and uses the code table found by compressing the database. After changing the frequencies of the itemsets in the code table, the algorithm generates new transactions separately. The transactions are formed using the itemsets of the CT which are picked based on their frequencies. The algorithm makes sure that each attribute gets a value for each generated transaction and hence, produces transactions of the same (maximum) length. The produced database consists of the same amount of rows (transactions) as the original one, but someone can simply produce more or less - depending on what is needed.

In this thesis project, the described algorithm was implemented and tested using an example database db_{ex}. After finding the best clustering (x) for the db_{ex}, the produced CTs were used by the algorithm to generate new (sub)databases for each cluster (without changing their frequencies). The subdatabases were later merged into a new big database of the same size as the original. By running again the clustering algorithm with the new constructed db, the best number of clusters

was again x and the resulted clusters were very similar to the previous original ones. This procedure validated our implementation of the algorithm and allowed us to continue.

9.1 Identification of clusters

This section contains a detailed description of the followed procedure for the validation of the KRIMPevol algorithm. As mentioned in Section 7, the focus is on two steps of the algorithm. In this section, the step 3 of the algorithm KRIMPevol is studied. The scope of these experiments is to understand the requirements under which the algorithm correctly identifies new created clusters or unused ones. In Section 9.2, the experiments focus on step 6: associating the clusters of different epochs using the dissimilarities found.

The database that was chosen as the starting one (first epoch) is the mushroom db. The database has 1824 instances (rows), with attributes corresponding to 23 species of mushrooms and therefore, 23 columns [79]. The whole database was first clustered using the clustering algorithm and was best clustered into 20 clusters. For the validation of the algorithm, we need to construct a new epoch and review the evolution between the constructed epoch (second) and the original one (first). The new database is constructed as follows: Using a combination (more on that in the next section) of three out of the twenty clusters, a new CT (newCT) is created. Following the previous described algorithm, the artificial corresponding database of the newCT is produced. Later, the produced database is merged with the remaining, already existing db parts, of the $20 - 3 = 17$ clusters and the new epoch (database) is ready to use.

9.1.1 Construction of newCT

The new CT is a rich combination of three of the CTs produced by the clustering of the database. As mentioned above, the database was clustered into 20 clusters. In this thesis, we performed 20 iterations to guarantee the accuracy of the results. By partitioning the database into 20 parts, the best encoded size achieved was 229705.85 and the average (internal) dissimilarity between the clusters of the database was 8.3. The dissimilarity found in [72] within the clusters of the same database was 7.8. The small difference is due to the randomized structure of the algorithm. In addition, the amount of iterations performed in this project were 20 (double the ones used in [72]), meaning that a slightly better clustering of the specific database was mighty achieved.

We arbitrary take into consideration three out of the twenty clusters, namely cluster 0, 7 and 14, with an amount of rows of 208, 147 and 192 respectively. The chosen clusters share a common feature: they are multipliers of 7, which I consider as my 'lucky number'. It turned out they share almost the same amount of rows which makes them a great choice. For the database build using the newCT, the amount of generated transactions is chosen to be 200, roughly the size of the used CTs. The merged database of the remaining 17 clusters sum up to 7577 rows and therefore the new db contains $7577 + 200 = 7777$ rows (it was a coincidence - I swear!).

To summarize, three CTs are used and the newCT is the combination of these CTs represented by the equation:

$$k_0 \cdot CT_0 + k_1 \cdot CT_7 + (1 - k_0 - k_1) \cdot CT_{14}$$

where $k_0 \in [0, 1]$ and $k_1 \in [0, 1 - k_0]$. The equation forms a 2-simplex or a so-called triangle. Imagine a triangle having as vertices the given CTs. A point p inside the triangle corresponds to a possible newCT. The distance of the point and each vertex represents the proportion of that CT_i in the newCT. The farthest a point from a vertex is, the less the specific CT_i will contribute to the newCT. Therefore, the centroid of the triangle symbolizes the newCT with equally contributed CTs (each 33.33%). When the point p is on the edge of the triangle, it consists only from the CTs of the vertices of the specific edge and so on.

If $k_2 = 1 - k_0 - k_1$, the next cases were considered:

				newCT		
Case No.	k_0	k_1	k_2	CT_0	CT_7	CT_{14}
1	0	0	1	-	-	100%
2	0	0.2	0.8	-	20%	80%
3	0	0.3	0.7	-	30%	70%
4	0	0.4	0.6	-	40%	60%
5	0.33	0.33	0.33	33%	33%	33%

Table 2: Construction of the newCT

The newCT contains the proportion shown in Table 2 for each case. When a CT contributes by α%, we randomly take the α% of the non-sigleton itemsets rows and the α% of sigleton ones and copy them to the new (empty) CT. Each case is described in detail in the following sections.

9.1.2 Case 1

In case 1, the following proportions are applied for the construction of the newCT: 0% of CT_0, 0% of CT_7 and 100% of CT_{14}. By constructing the newCT that way, the two clusters, 0 and 7, are 'deleted' and the algorithm is expected to understand that. Hence, the expected best number of resulted clusters (k) is $20 - 2 = 18$.

In Table 3, the dissimilarities of the newCT and the three CTs used can be found. As expected, the dissimilarity between the newCT and CT_{14} is zero since they are identical.

	CT_0	CT_7	CT_{14}
newCT	8.458473	7.716713	0.000000

Table 3: Dissimilarities with newCT - Case 1.

A snapshot diagram of the clustering results is demonstrated in Figure 11. The x axis of the figure represents the number of tested clusters (k) while the y axis the calculated size of the database for that specific partition. As it can be seen, the smallest size of the database (sizedb) is achieved for k=18, as expected.

Figure 11: Clustering results - Case 1

Furthermore, to guarantee that the algorithm does not find correlation between the clusters that do not exist in the second epoch, the dissimilarities between the clusters of the different epochs

are examined. As mentioned before, a cluster is associated with a new, evolving cluster iff the dissimilarity with the new db is smaller that the dissimilarity of the specific cluster in its own database.

In Table 4, the dissimilarities within the first epoch for the three clusters are shown. This table will be used as indicator for finding correlation between the clusters of the second and the first epoch in all cases. The minimum dissimilarities of each cluster are highlighted and they can be used as thresholds: when the dissimilarity between the clusters of the second and the first epoch are smaller than the minimum of the starting epoch, evolution is assumed. In Table 5, the dissimilarities of the three clusters of the starting epoch are compared with the new found clusters. The number 1 is used before the clusters of the second epoch for identification reasons.

	CT_0	CT_7	CT_{14}
CT_0	0	8.645203	8.458473
CT_1	9.880593	9.892202	9.564601
CT_2	9.81002	8.436834	7.479195
CT_3	9.391163	8.351768	7.517278
CT_4	8.677418	8.224043	8.081466
CT_5	9.526072	9.315437	8.742781
CT_6	10.177761	9.212404	9.010142
CT_7	8.645203	0	7.716713
CT_8	8.138637	7.548962	7.483003
CT_9	9.713433	8.847964	8.137517
CT_{10}	10.084228	9.11078	8.970266
CT_{11}	9.234586	8.44908	8.216666
CT_{12}	8.90373	7.21675	7.916137
CT_{13}	9.077066	8.388633	8.157391
CT_{14}	8.458473	7.716713	0
CT_{15}	9.169945	8.715112	8.17326
CT_{16}	10.920519	9.788195	9.39528
CT_{17}	8.914738	8.225952	7.996452
CT_{18}	9.493765	8.650794	8.358883
CT_{19}	9.172633	8.344824	7.673089

Table 4: Internal dissimilarities - First epoch

	CT_0	CT_7	CT_{14}
$1.CT_0$	9.192178	8.412497	8.054873
$1.CT_1$	9.138803	8.379508	8.148606
$1.CT_2$	10.914496	9.610587	8.573374
$1.CT_3$	8.610747	7.81611	7.430294
$1.CT_4$	9.266481	8.044403	7.325126
$1.CT_5$	11.271595	10.266768	9.913181
$1.CT_6$	9.540662	8.551606	8.046282
$1.CT_7$	8.892677	8.087378	7.955511
$1.CT_8$	10.723107	9.919362	10.100254
$1.CT_9$	9.306474	8.591633	6.411708
$1.CT_{10}$	9.197416	9.009255	8.630555
$1.CT_{11}$	9.196278	8.478708	8.200873
$1.CT_{12}$	8.582322	7.96973	7.981436
$1.CT_{13}$	9.426066	8.972515	8.708433
$1.CT_{14}$	9.020735	8.290096	7.790988
$1.CT_{15}$	10.209085	9.691139	8.888955
$1.CT_{16}$	8.730575	7.914072	6.513784
$1.CT_{17}$	9.563605	8.957379	6.407749

Table 5: Dissimilarities between epochs - Case 1

It is easy to see that the algorithm correctly does not recognize evolution for the unused clusters. The resulted dissimilarities are greater than the ones within the database and therefore, the clusters 0 and 7 are not related with any other. In contrast, evolution is identified for cluster 14, validating our steps.

9.1.3 Case 2

In case 2, the following proportions are applied for the construction of the newCT: 0% of CT_0, 20% of CT_7 and 80% of CT_{14}. In cases 2-4, both cluster 7 and 14 are used with different proportions each time. These experiments, using various proportions, help us discover the threshold above which both clusters are identified by the algorithm.

In Table 6, the dissimilarities of the newCT and the three CTs used can be found. As seen, the dissimilarity of newCT and cluster 7, comparing to Case 1, has decreased. In addition, the dissimilarity of newCT and cluster 14 is not zero anymore, as the proportion of the cluster in the newCT is lower in this case.

	CT_0	CT_7	CT_{14}
newCT	8.423302	6.390826	1.131787

Table 6: Dissimilarities with newCT - Case 2

Similarly with Case 1, in this case, the best clustering was achieved when the database was partitioned into 18 clusters. The algorithm did not recognize cluster 7 as a separate one. Therefore, we can conclude that a contribution of 20% percent from one cluster is not a sufficient percentage as it is ignored by the algorithm. The results of the clustering of the database can be found in Figure 12. It is clearly depicted as well that the total encoded size for k equal to 19 and 20 is much smaller than Case 1.

Figure 12: Clustering results - Case 2.

	CT_0	CT_7	CT_{14}
$1.CT_0$	9.670269	8.80819	7.944596
$1.CT_1$	9.418395	7.037136	6.403843
$1.CT_2$	9.611518	9.408417	7.042584
$1.CT_3$	9.952382	9.244427	9.021881
$1.CT_4$	10.0138	9.098221	9.366721
$1.CT_5$	9.291969	8.650148	8.290094
$1.CT_6$	9.871525	8.728652	8.761679
$1.CT_7$	9.036694	8.337583	8.136143
$1.CT_8$	8.784121	7.61713	7.920808
$1.CT_9$	8.651378	7.930661	7.618028
$1.CT_{10}$	10.222525	9.037161	8.844674
$1.CT_{11}$	10.241244	8.252476	7.720122
$1.CT_{12}$	9.382069	8.416459	5.782151
$1.CT_{13}$	9.977582	8.276221	8.764814
$1.CT_{14}$	9.321463	8.966295	8.671222
$1.CT_{15}$	8.761891	8.238931	7.886045
$1.CT_{16}$	8.548034	7.74383	7.654585
$1.CT_{17}$	8.781811	6.901844	7.550518

Table 7: Dissimilarities between epochs - Case 2

In Table 7, the dissimilarities between the clusters of the two epochs are shown. Comparing to Table 4 as well, correlation can be found for both the used clusters, with codetables CT_7 and

CT_{14}. Although in clustering results the cluster 7 was not identified, the dissimilarities showed a correct correlation.

9.1.4 Case 3

In case 3, the following proportions are applied for the construction of the newCT: 0% of CT_0, 30% of CT_7 and 70% of CT_{14}. In this case, the percentage of contibution of cluster 7 is increased, reaching a 30%. The dissimilarities between the newCT and the old ones are shown in Table 8.

	CT_0	CT_7	CT_{14}
newCT	8.478641	4.889553	1.543787

Table 8: Dissimilarities with newCT - Case 3

The dissimilarity is again pretty small regarding the cluster 14, as it was mainly used in the construction of the newCT. The dissimilarity of cluster 7 has significantly dropped down due to the higher participation and it is smaller than the previous case.

The clustering results can be found in Figure 13. It is clearly depicted that the chosen amount of clusters is again 18 and no more clusters are found. It is worth saying that, in Case 2, the difference between partitioning the database into 18 and 19 clusters is bigger than in Case 3. More precisely, the difference in dbsize in Case 2 between 18 and 19 cluster is 18000 whereas in Case 3 it is only 200. This difference is highlighted as it shows that the algorithm interpret the differences between the cases.

Figure 13: Clustering results - Case 3.

In Table 9, the dissimilarities are shown. Matching the previous case, relation is found to both used clusters with higher similarity on the most used one.

9.1.5 Case 4

In case 4, the following proportions are applied for the construction of the newCT: 0% of CT_0, 40% of CT_7 and 60% of CT_{14}. In Table 10, the dissimilarities of the newCT and the three CTs used are displayed.

	CT_0	CT_7	CT_{14}
$1.CT_0$	9.670269	8.80819	7.944596
$1.CT_1$	9.418395	7.037136	6.403843
$1.CT_2$	9.611518	9.408417	7.042584
$1.CT_3$	9.952382	9.244427	9.021881
$1.CT_4$	10.0138	9.098221	9.366721
$1.CT_5$	9.291969	8.650148	8.290094
$1.CT_6$	9.871525	8.728652	8.761679
$1.CT_7$	9.036694	8.337583	8.136143
$1.CT_8$	8.784121	7.61713	7.920808
$1.CT_9$	8.651378	7.930661	7.618028
$1.CT_{10}$	10.22253	9.037161	8.844674
$1.CT_{11}$	10.24124	8.252476	7.720122
$1.CT_{12}$	9.382069	8.416459	5.782151
$1.CT_{13}$	9.977582	8.276221	8.764814
$1.CT_{14}$	9.321463	8.966295	8.671222
$1.CT_{15}$	8.761891	8.238931	7.886045
$1.CT_{16}$	8.548034	7.74383	7.654585
$1.CT_{17}$	8.811181	7.948231	7.784175

Table 9: Dissimilarities between epochs - Case 3

	CT_0	CT_7	CT_{14}
newCT	8.593292	3.337453	2.896171

Table 10: Dissimilarities with newCT - Case 4

As the proportions have changed, the dissimilarity in the two last columns changed as well. The newCT resembles more the cluster 14 which is correct, as it contributed more in the creation. In addition, the dissimilarity has significantly dropped for cluster 7 due to the growing proportion. In cases 2-4, the dissimilarities of the newCT with the used ones are in accurate correspondence with the proportions.

The clustering results for this case can be found in Figure 14. As the percentages changed, the second cluster is identified as a new one and therefore the outputted result for the best k is 19. Consequently, when a cluster contributes in the newCT by 40%, then it is identified as a separate and independent cluster.

Figure 14: Clustering results - Case 4.

	CT_0	CT_7	CT_{14}
$1.CT_0$	9.733399	7.707999	8.480378
$1.CT_1$	8.400634	7.577913	7.650356
$1.CT_2$	9.179862	9.004689	8.841698
$1.CT_3$	9.484131	8.431836	8.341887
$1.CT_4$	9.128845	8.477995	5.611909
$1.CT_5$	8.971011	8.205572	8.253032
$1.CT_6$	8.908368	7.971514	7.618075
$1.CT_7$	10.23564	9.257343	9.082173
$1.CT_8$	8.633002	7.913876	7.646721
$1.CT_9$	9.240905	8.371577	8.160708
$1.CT_{10}$	9.576926	8.608169	8.384755
$1.CT_{11}$	9.673626	8.906916	8.686407
$1.CT_{12}$	8.481774	7.730519	7.688372
$1.CT_{13}$	9.127862	6.183703	8.182787
$1.CT_{14}$	10.62168	9.904812	10.17423
$1.CT_{15}$	8.766622	7.844529	7.17252
$1.CT_{16}$	8.672243	7.131415	6.565709
$1.CT_{17}$	9.736013	7.893491	8.298412
$1.CT_{18}$	8.811181	7.948231	7.784175

Table 11: Dissimilarities between epochs - Case 4

The dissimilarities of the different epochs are shown in Table 11. As illustrated, similarity is again found for both clusters. Considering the cases 2-4, we can conclude that with a percentage of 40, a new cluster emerges and is identified in the clustering results. Furthermore, based on Tables 3, 6, 8 and 10, it is clear depicted that as the involving percentages were linearly changing, the dissimilarities of the newCT were altered as well. This outcome is demonstrated in Figure 15. With blue color, the dissimilarity of cluster 14 are shown whereas the red line represents the dissimilarity of cluster 7; the lines follow the percentages of each CT used.

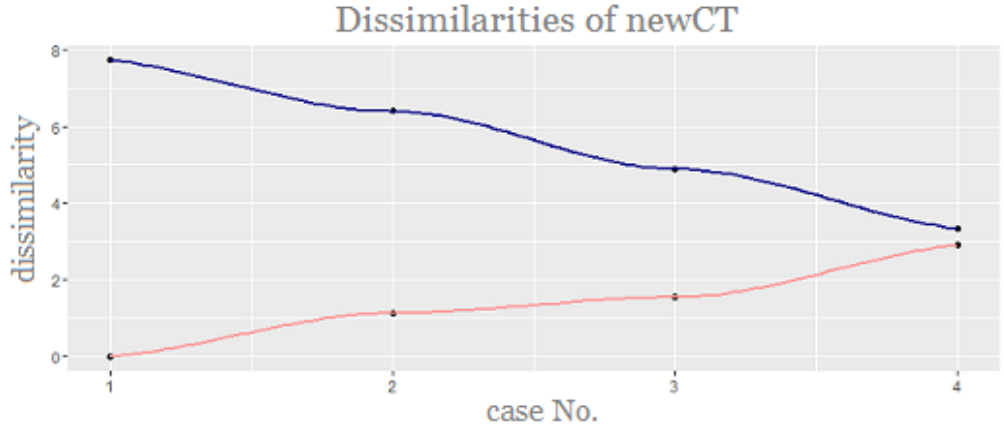

Figure 15: Dissimilarities of the newCT over the cases

9.1.6 Case 5

In this case, it is time to introduce all three clusters in the construction of the newCT. As concluded from the previous cases, a percentage below 30% is not recognized as a separate cluster and therefore, the case to be checked is when each cluster has a percentage of 33%, meaning that it contributes equally to the newCT.

In Table 12, the dissimilarities of the newCT and the old ones are shown. In this case, the dissimilarity is small for all clusters, even for cluster 0, as it contributes in the newCT as well.

	CT_0	CT_7	CT_{14}
newCT	4.482469	4.524824	6.190316

Table 12: Dissimilarities with newCT - Case 5

In Figure 16, the results of the clustering are depicted. The best clustering is achieved with an amount of 20 clusters. All three clusters are identified by the algorithm as separate and independent.

Figure 16: Clustering results - Case 5.

Looking at the dissimilarities in Table 13, all three clusters are found related with the second epoch as they achieve smaller dissimilarity than their threshold within the first epoch (Table 4). Consequently, the percentages used are big enough in order for all the three clusters to be recognized as separate ones.

9.2 Zoom into dissimilarities

Now that we know the requirements and what is needed for an evolution (change of clusters) to be identified (proportions), it is time to look closer to the dissimilarities found. The desired output of the algorithm is a graph in which not only the amount of clusters in each different epoch can be seen, but also the correlations between these clusters. For example, when a cluster in epoch 1 is associated with one of the second epoch, a line should exist between them. The thicker the line, the more correlated (similar) the clusters are. Hence, the scope of the experiments in this section is the interpretation and validation of the outputted dissimilarities within clusters of different epochs. By using our creativity and inventiveness we can check if the algorithm follows -and most important understands- our thinking.

Firstly, the ingredients of an evolving database that will be used for testing have to be found. We simply select some clusters (arbitrary - no lucky numbers this time) from the mushroom db which will be used for building the evolving database. These are the clusters with codetables: CT_1, CT_5, CT_6 and CT_{16}. For identification, the number before the codetable name corresponds to the epoch the specific cluster belongs. For example, $0.CT_1$ represents the CT of the first (initial) epoch, $1.CT_1$ the second etc.

	CT_0	CT_7	CT_{14}
$1.CT_0$	9.354919	8.901068	8.608921
$1.CT_1$	9.281156	8.152927	7.369147
$1.CT_2$	10.80854	9.955626	10.24014
$1.CT_3$	8.132377	7.807353	7.85833
$1.CT_4$	8.195073	8.064633	7.956905
$1.CT_5$	9.750019	8.751838	8.518875
$1.CT_6$	10.07563	9.112906	8.891978
$1.CT_7$	9.595093	6.096486	8.625916
$1.CT_8$	8.547653	8.001103	7.875218
$1.CT_9$	9.142222	8.439041	7.979182
$1.CT_{10}$	9.147262	8.130437	7.977593
$1.CT_{11}$	9.479187	8.714024	8.627788
$1.CT_{12}$	8.415635	8.354611	8.419516
$1.CT_{13}$	8.893149	8.184176	8.293511
$1.CT_{14}$	9.35381	8.49873	8.18197
$1.CT_{15}$	8.305014	7.812502	7.470197
$1.CT_{16}$	9.962105	8.91846	8.687659
$1.CT_{17}$	9.255911	7.839164	8.423586
$1.CT_{18}$	10.86114	10.02002	9.788415
$1.CT_{19}$	10.12273	8.612696	7.215286

Table 13: Dissimilarities between epochs - Case 5

To check the algorithm alongside with our creation process, both the internal and external dissimilarities have to be taken into account. As specified in the previous section, relations are found between two clusters of different epochs iff the dissimilarity between them is smaller than the internal dissimilarities in their epoch.

In this section, a new evolving database will be built using elements from the mushroom db. When a new epoch is created, the dissimilarities between the new and the previous epoch will be examined. Each time, the internal and the external dissimilarities are shown in a Table where the internal dissimilarities are indicated with gray background color.

9.2.1 First epoch

Starting with, the first epoch is chosen to be composed by three clusters, namely cluster 1, 5 and 6 of the mushroom db with codetables CT_1, CT_5 and CT_6. The databases of the specific clusters are merged into one database which was clustered again in order to avoid bias. Successfully, the algorithm yielded that the amount of clusters that achieves the smallest encoded size (47373.88) is again 3. The new clusters have names 0, 1 and 2. As we are not interested with the correspondence of the ones used, the naming is kept that way. The internal dissimilarities between the found clusters of the first epoch are shown in Table 14.

	$0.CT_0$	$0.CT_1$	$0.CT_2$
$0.CT_0$	-	9.131157	9.64668
$0.CT_1$	9.131157	-	9.007286
$0.CT_2$	9.64668	9.007286	-

Table 14: Dissimilarities - epoch 0

It can be concluded that in order for a cluster to be related with one of the first epoch, a dissimilarity smaller than 9.1, 9.0 and 9.0 should be found respectively for each cluster. These thresholds can be found by looking at the Table column-wise.

9.2.2 Second epoch

In the second epoch, a new cluster is introduced to the database in a 'hidden' way. The newCT will be constructed using a mix combination of an existing cluster and an emerging one, namely cluster 16. As shown in section 9.1, a small percentage of 30% allows us to have a cluster which is not yet identified. Therefore, the second epoch consists of the proportions: 100% of $0.CT_0$, 100% of $0.CT_1$ and newCT=70% of $0.CT_2$ + 30% of $0.CT_{16}$. The algorithm concluded that the best partition is achieved by 3 clusters, as expected. The dissimilarities between the clusters of epoch 0 and 1 are shown in Table 15.

	$0.CT_0$	$0.CT_1$	$0.CT_2$	$1.CT_0$	$1.CT_1$	$1.CT_2$
$0.CT_0$	0	9.131157	9.64668	5.30034	4.368648	7.573722
$0.CT_1$	9.131157	0	9.007286	4.445204	6.4843	7.218963
$0.CT_2$	9.64668	9.007286	0	8.776188	10.56329	2.833096
$1.CT_0$	5.30034	4.445204	8.776188	0	10.04717	7.888785
$1.CT_1$	4.368648	6.4843	10.56329	10.04717	0	9.074285
$1.CT_2$	7.573722	7.218963	2.833096	7.888785	9.074285	0

Table 15: Dissimilarities - epoch 0 & 1

We simply relate each new cluster with the one of the previous epoch that is more similar to. As seen, each cluster corresponds to only one of the previous database and vice versa; the minimum of each row is the minimum of each column as well. This accurate result correctly shows a correspondence of 1-1 between the epochs. However, although cluster 2 was used with a smaller proportion of 70%, the results show a great similarity of this cluster with the one of the new epoch. Therefore, the clear amount of the output dissimilarity does not indicate the level of correlation and this hypothesis is not validated by the results of the artificial database. The output figure of the algorithm is shown in Figure 17, where each cluster of epoch 0 is related with one of epoch 1, following the actions taken.

Figure 17: Output of epoch 0 & 1

Figure 18: Output of epoch 1 & 2

9.2.3 Third epoch

In the third epoch, the introduction of the new cluster is clearly made. The involving percentage of the cluster 16 is now 45% which exceeds the threshold for detection. Hence, the third epoch consists of the proportions: 100% of $0.CT_0$, 100% of $0.CT_1$ and newCT=65% of $0.CT_2$ + 45% of $0.CT_{16}$. For this epoch, clustering resulted to 4 clusters due to the high proportion of the new cluster. The dissimilarities between the clusters of epoch 1 and 2 are depicted in Table 16.

	$1.CT_0$	$1.CT_1$	$1.CT_2$	$2.CT_0$	$2.CT_1$	$2.CT_2$	$2.CT_3$
$1.CT_0$	0	10.047165	7.888785	5.793717	6.061742	5.994211	5.932117
$1.CT_1$	10.047165	0	9.074285	6.411897	5.852088	7.224501	8.504822
$1.CT_2$	7.888785	9.074285	0	6.230407	8.060714	7.038461	6.617685
$2.CT_0$	5.793717	6.411897	6.230407	0	8.539365	8.900065	7.279305
$2.CT_1$	6.061742	5.852088	8.060714	8.539365	0	8.856824	7.975933
$2.CT_2$	5.994211	7.224501	7.038461	8.900065	8.856824	0	7.813925
$2.CT_3$	5.932117	8.504822	6.617685	7.279305	7.975933	7.813925	0

Table 16: Dissimilarities - epoch 1 & 2

The third epoch consists of 4 clusters and therefore, we examine the dissimilarities in Table 16, row-wise. Unfortunately, the correlations between the clusters of the two epochs are not easily identified. Clusters 0 and 1 of the third epoch can be easily correlated with clusters 0 and 1 of the previous epoch respectively. However, there is also a strong similarity between cluster 0 of the second epoch and the two remaining ones, namely 2 and 3. The results of the correlation between epoch 1 and 2 can be found in Figure 18. As seen, three of the clusters of the third epoch are found related with one of the previous epoch - which is not what was created. The desired results would be an 1-1 correlation between two of the four clusters of epoch 2 while the remaining two would be related with one remaining of the old clusters, following the creation process of the epoch (illustrated in Figure 19).

After this observation, research and trials took place with a view to better interpret the dissimilarity numbers and correlations. Sadly, the correlation between clusters of different epochs seems more complex than we firstly thought. In Figure 19, a desired output of the epochs 0, 1 and 2 is depicted. As mentioned, the algorithm was unable to find a correct correlation between the last two epochs and this difficulty remained in all the experiments and trials made with different ways of construction. It is therefore concluded that KRIMPevol is capable of recognizing the amount of clusters in each epoch (and produce an output as shown in Figure 20) but more research and steps should be made for the correct recognition of the evolution. An idea for future research is the exploration of the dissimilarities and the construction of an equation that can better understand and output the correct correlations between the clusters.

Figure 19: Desired output of KRIMPevol

Figure 20: Output of KRIMPevol

10 Conclusions

The original motivation for this thesis arises from a digital humanities' challenge where Historians expressed the need of analyzing themes or topics in a corpus over time using digital humanities tools. The first six sections of this project consist of a literature review of the existing methods as well as the proposal of a new idea based on the KRIMP algorithm. The next sections includes a

detailed description of the KRIMPevol algorithm, a mean to identify an evolving process between clusters of objects that belong to different time intervals. The implementation and the experiments performed in order for its validation successfully verified the first part of the algorithm but were unable to do the same for the other part.

More precisely, the algorithm can correctly identify the number of clusters belonging to different epochs of a database and understand when different clusters occur or die. It creates a graph that illustrates the clusters in each epoch, like the one in Figure 20. However, correlations between clusters were not successfully implemented as the dissimilarity measure was not enough for this step. More research, steps or modifications on the measure so that can be used with different epochs have to be made to produce an output as the graph in Figure 19.

11 Future Work

The proposed idea for the establishment of evolution within a database was partially validated during the experiments. Regarding step 3 of the algorithm, the capability of KRIMPevol of recognizing emerging clusters as well as detecting when a cluster does not exist anymore was proven. Unfortunately, in order to find the specific correlations and connect the clusters of the different epochs, the simple measure of dissimilarity is not enough. Although for a small amount of clusters the correlation was easily identified, this was not the case for the later epochs. This result creates space for future work in order for this property of the KRIMPevol to be implemented as well. Therefore, a suggestion for future research would be to closely examine the correlation of internal and external epochs and trying to come up with an equation that can result into correct correlation.

References

[1] Daniel Keys Moran. (n.d.). BrainyQuote.com. Retrieved March 21, 2016, from BrainyQuote.com Web site: http://www.brainyquote.com/quotes/quotes/d/danielkeys230911.html

[2] Gelbukh, A. (2011). *Computational Linguistics and Intelligent Text Processing*. Springer.

[3] Berry Michael, W. (2004). Automatic Discovery of Similar Words. *Survey of Text Mining: Clustering, Classification and Retrieval*. Springer Verlag, New York, 200, 24-43.

[4] Rodríguez, M. D. B., Hidalgo, J. M. G., & Agudo, B. D. (1997). *Using WordNet to complement training information in text categorization*.

[5] Hotho, A., Staab, S., & Stumme, G. (2003). Ontologies improve text document clustering. *Proceedings of the third IEEE International Conference in Data Mining*. 541-544.

[6] Ponte, J. M., & Croft, W. B. (1997). Text segmentation by topic. *Proceedings of the Advanced Technology for Digital Libraries* . 113-125.

[7] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3, 993-1022.

[8] Haghighi, A., & Vanderwende, L. (2009). Exploring content models for multi-document summarization. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. 362-370.

[9] Sun, Y., Lin, L., Deng, H., Li, J., He, B., Sun, R., & Ouyang, P. (2008). Structural changes of bamboo cellulose in formic acid. *BioResources*, 3(2), 297-315.

[10] Wei, X., & Croft, W. B. (2006). LDA-based document models for ad-hoc retrieval. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 178-185.

[11] Hoffman, M. D., Blei, D. M., Wang, C., & Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1), 1303-1347.

[12] Papadimitriou, C. H., Tamaki, H., Raghavan, P., & Vempala, S. (1998). Latent semantic indexing: A probabilistic analysis. *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* . 159-168.

[13] Rajaraman, A., & Ullman, J. D. (2012). *Mining of massive datasets (Vol. 1)*. Cambridge: Cambridge University Press.

[14] Landauer, T. K. & Dumais, S. T. (1997). A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. *Psychological Review*, 104(2):211–240.

[15] Golub, G., & Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2), 205-224.

[16] Mens, T., Claes, M., Grosjean, P., & Serebrenik, A. (2014). *Studying evolving software ecosystems based on ecological models*. (pp. 297-326). Springer Berlin Heidelberg.

[17] Salton, G., & McGill, M. J. (1986). *Introduction to modern information retrieval*. New York.

[18] Manning, C. D., & Schütze, H. (2000). *Foundations of Statistical Natural Language Processing*. The MIT Press Cambridge, UK.

[19] Hofmann, T. (1999). Probabilistic latent semantic indexing. *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* . 50-57.

[20] Hofmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42, 177-196.

[21] Hinneburg, A., Gabriel, H. H., & Gohr, A. (2007). Bayesian folding-in with Dirichlet kernels for PLSI. *Proceedings of the Seventh IEEE International Conference*. 499-504.

[22] Dempster, A. P., Laird, N. M. & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(l):l-38.

[23] Zhai, C. X. (2008). Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141.

[24] Si, L. & Jin, R. (2005). Adjusting mixture weights of gaussian mixture model via regularized probabilistic latent semantic analysis. *Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*.

[25] Daud, A., Li, J., Zhou, L., & Muhammad, F. (2010). Knowledge discovery through directed probabilistic topic models: a survey. *Frontiers of computer science in China*, 4(2), 280-301.

[26] Clinchant, S., & Gaussier, E. (2012). *Textual information access: statistical models*. Wiley.

[27] Griffiths T. L., & Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Science of the United States of America*, 101, 5228-5235.

[28] Zhang, Z., & Zhang, R. (2008). *Multimedia data mining: a systematic introduction to concepts and theory*. CRC Press.

[29] Blei, D. M., & Lafferty, J. D. (2009). Topic models. *Text mining: classification, clustering, and applications*, 10(71), 34.

[30] Jockers, M. (2016, April 4). *The LDA Buffet is Now Open; or, Latent Dirichlet Allocation for English Majors*. Retrieved from: http://www.matthewjockers.net/2011/09/29/the-lda-buffet-is-now-open-or-latent-dirichlet-allocation-for-english-majors/

[31] Lee, S., Baker, J., Song, J., & Wetherbe, J. C. (2010). An empirical comparison of four text mining methods. *Proceedings of the In System Sciences (HICSS)* 1-10.

[32] Blei, D. M., & Lafferty, J. D. (2007). A correlated topic model of science. *The Annals of Applied Statistics*, 17-35.

[33] Chen, J., Zhu, J., Wang, Z., Zheng, X., & Zhang, B. (2013). Scalable inference for logistic-normal topic models. *Proceedings of the Advances in Neural Information Processing Systems*. 2445-2453.

[34] Zeng, J., Liu, Z. Q., & Cao, X. Q. (2012). A new approach to speeding up topic modeling.

[35] Wang, X., & McCallum, A. (2006). Topics over time: a non-Markov continuous-time model of topical trends. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* . 424-433.

[36] Blei, D. M., & Lafferty, J. D. (2006). Dynamic topic models. *Proceedings of the 23rd international conference on Machine learning*. 113-120.

[37] Jo, Y., Hopcroft, J. E., & Lagoze, C. (2011). The web of topics: discovering the topology of topic evolution in a corpus. *In Proceedings of the 20th international conference on World wide web*. pp. 257-266.

[38] Wang, X., Zhai, C., & Roth, D. (2013). Understanding evolution of research themes: a probabilistic generative model for citations. *In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* pp. 1115-1123.

[39] Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

[40] Anick, P. G., & Vaithyanathan, S. (1997). Exploiting clustering and phrases for context-based information retrieval. *Proceedings of the ACM SIGIR Forum*. 314-323.

[41] Cutting, D. R., Karger, D. R., & Pedersen, J. O. (1993, July). Constant interaction-time scatter/gather browsing of very large document collections. *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. 126-134.

[42] Frigui, H., & Krishnapuram, R. (1999). A robust competitive clustering algorithm with applications in computer vision. *Pattern Analysis and Machine Intelligence*, 21(5), 450-465.

[43] Han, J.W., Cai, Y., & Cercone, N. (1993). Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 5, 29-40.

[44] Xu, R., & Wunsch, D. (2005). Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16.3, 645-678.

[45] de Oliveira, J. V., & Pedrycz, W. (2007). *Advances in fuzzy clustering and its applications*. New York: Wiley.

[46] Friedman, J., Hastie, T., & Tibshirani, R. (2009). *The elements of statistical learning*. Springer.

[47] Johnson, A. & Wichern, D. (2002) *Applied Multivariate Statistical Analysis*, Prentice Hall, New Jersey.

[48] Sneath, P., & Sokal, R. (1973). *Numerical Taxonomy*. W.H. Freeman Co., San Francisco.

[49] King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317), 86-101.

[50] Berry, M. W. (Ed.). (2004). *Proceedings of the Fourth SIAM International Conference on Data Mining*. SIAM.

[51] Leung, Y., Zhang, J. S., & Xu, Z. B. (2000). Clustering by scale-space filtering. *Proceedings of the Pattern Analysis and Machine Intelligence*. 1396-1410.

[52] Guan, Y. (2006). *Large-scale clustering: algorithms and applications*. ProQuest.

[53] Maimon, O., & Rokach, L. (Eds.). (2007). *Soft computing for knowledge discovery and data mining*. Springer Science & Business Media.

[54] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley: University of California Press. 281-297.

[55] Markov, Z., & Larose, D. T. (2007). *Data mining the Web: uncovering patterns in Web content, structure, and usage*. John Wiley & Sons.

[56] Laplante, F., Kardouchi, M., & Belacel, N. (2015). Image Categorization Using a Heuristic Automatic Clustering Method Based on Hierarchical Clustering. *Proceedings of the Image Analysis and Recognition*. 150-158.

[57] Bezdek James, C. (1981). Pattern Recognition with Fuzzy Function Algorithms.

[58] Konkol, M. (2015). Fuzzy Agglomerative Clustering. *Proceedings of the Artificial Intelligence and Soft Computing*. 207-217.

[59] Withanawasam, J. (2015). *Apache Mahout Essentials*. Packt Publishing Ltd.

[60] Gan, G. (2011). *Data Clustering in C++: An ObjectOriented Approach*. Chapman and Hall/CRC.

[61] Zadeh, L.A. (1965). Fuzzy Sets. *Information and Control*, 8(1965) 338-353.

[62] Fuzzy clustering. (n.d.). In Wikipedia. Retrieved April 24, 2016, https://en.wikipedia.org/wiki/Fuzzy_clustering

[63] Samatova, N. F., Hendrix, W., Jenkins, J., Padmanabhan, K., & Chakraborty, A. (2013). *Practical Graph Mining with R*. CRC Press.

[64] "Seven.", *Seinfeld*. Fox. 1 Feb. 1996. Television.

[65] Aggarwal, C. C., & Zhai, C. (2012). *Mining text data*. Springer Science & Business Media.

[66] Tan, P. N., Steinbach, M., & Kumar, V. (2006). Cluster Analysis: Basic Concepts and Algorithms. In *Introduction to Data Mining* (pp. 487-567). Addison-Wesley.

[67] Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2), 207-216.

[68] Siebes, A., Vreeken, J., & van Leeuwen, M. (2006). Item sets that compress. *Proceedings of the SDM'06*. 393–404.

[69] van Leeuwen, M., Vreeken, J., & Siebes, A. (2006). Compression picks item sets that matter. *Proceedings of the Knowledge Discovery in Databases: PKDD 2006*. 585-592.

[70] Griinwald, P. D., Myung, I. J., & Pitt, M. A. (2005). *Advances in minimum description length: Theory and applications*. MIT press.

[71] Vreeken, J., Van Leeuwen, M., & Siebes, A. (2011). Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1), 169-214.

[72] van Leeuwen, M., Vreeken, J., & Siebes, A. (2009). Identifying the components. *Data Mining and Knowledge Discovery*, 19(2), 176-193.

[73] Gelbukh, A. (2011). *Computational Linguistics and Intelligent Text Processing*. Springer.

[74] Vreeken, J., Van Leeuwen, M., & Siebes, A. (2007). Characterising the difference. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 765-774.

[75] Vreeken, J., Van Leeuwen, M., & Siebes, A. (2007). Preserving privacy through data generation. *Proceedings of the Seventh IEEE International Conference on Data Mining*. 685-690.

[76] Synthetic data. (2009, November 29). McGraw-Hill Dictionary of Scientific and Technical Terms. Retrieved from: www.answers.com

[77] Texas Department of Criminal Justice. (2016, May 14). *Executed Offenders*. Retrieved from: http://www.tdcj.state.tx.us/death_row/dr_executed_offenders.html

[78] Rosenfeld, Michael J., Reuben J. Thomas, and Maja Falcon. (2016, May 16). *How Couples Meet and Stay Together (HCMST)*. Retrieved from: http://www.icpsr.umich.edu/icpsrweb/ICPSR/studies/30103?q=&paging.rows =25&sortBy=10

[79] Lichman, M. (2013). *UCI Machine Learning Repository*. Retrieved from: https://archive.ics.uci.edu/ml/datasets/Mushroom