6. DEZEMBER 2020

REPORT – PROGRAMMING PROJECT: HISC ALGORITHM & ENZYMES DATASET

FOR VU DATA MINING

RAPHAEL BEDNARSKY
MAXIMILIAN FAISSNER
PETER HUNYADI
LAURA JAHN
NIKOLA VINKO

TABLE OF CONTENTS

1. INTRODUCTION

In this intermediate submission we had to choose a dataset and a clustering algorithm which is suited to the dataset. After that we had to implement the algorithm in Python and compare the results with the algorithm implemented in ELKI.

After that we had to validate and analyse the results of our implementation and perform an exploratory data analysis.

We chose the dataset ENZYMES with the algorithm HiSC. In the following chapters we will describe the usages and the challenges of our implementations.

2. CLUSTER ANALYSIS

2.1. What is Clustering?

On the one hand clusters divide data into groups, so that you can get a better understanding of the similarity of the data. For example, if you want to find similar functionality of genes. On the other hand, cluster analysis is used for utility like compressing data and find the nearest neighbours of a point.

The goal of cluster analysis is to find groups based on the information in the given data where the datapoints are related or similar to the other points in the group. Below you can see an example of how that could look like:[1]

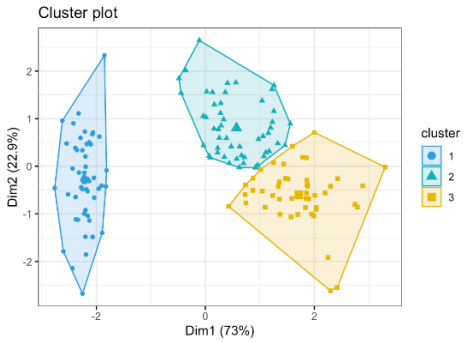

Figure 1: Clustering datapoints in a 2-dimensional feature space. [2]

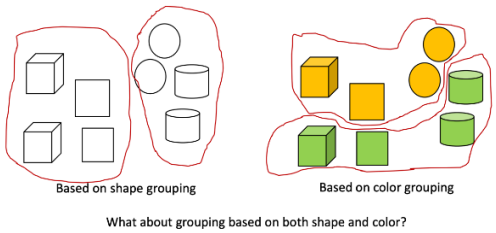

Figure 2: Choosing dimensions for clustering[3]

.

But what if you get higher dimensions in the dataset? How to cluster for example the figure below?

---

[1] https://www.datanovia.com/en/blog/k-means-clustering-visualization-in-r-step-by-step-guide/

[2] https://link.springer.com/chapter/10.1007/978-3-662-08968-2_16, on 6th of December 2020

[3] Data Mining Course, p.3: https://moodle.univie.ac.at/pluginfile.php/11513657/mod_resource/content/2/DM-2-HDData.DimensionalityReduction.pdf

2.2. Curse of Dimensionality

If you add more dimensions to a feature set the similarity of two points will probably decrease and therefor the clustering will be more difficult. -> This is called the Curse of Dimensionality.

To avoid this there is a method called Feature Reduction. The idea is to find a low dimensional feature space where redundant and irrelevant features are summarized and so this lower dimensional space represents the original space in a good way but you can cluster it better and easier. But you have to be careful by reducing dimensions because different attributes are relevant for different clusters.

There are a few ways how to perform the Feature Reduction like Principal Component Analysis (PCA), Singular value decomposition (SVD), Kernel PCA etc.[2]

2.3. Subspace Clustering

The goal of clustering high dimensional data is to search for clusters in subspaces of the original feature space. But there are some challenges you have to face:

1. *Find the correct subspace of each cluster*
2. *Find the correct cluster in each relevant subspace*

Because both challenges depend on each other you have to integrate the subspace search into the clustering process. There are some methods how to do this:

- Bottom-Up Approaches:
  Find all clusters in all subspaces. (Subspace Clustering)
- Top-Down Approaches:
  Each point is assigned to one subspace cluster or noise. (Projected Clustering, Correlation Clustering)
- Some novel Approaches:
  Find the best subspace for all clusters, each point is assigned to one cluster. (Subspace - centred)
- Multiple clusterings:
  Each point is assigned to multiple clusters in different subspaces. (Non-redundant)[4]

In the following chapters we will describe the algorithm we implemented which has a Top-Down Approach in a more detailed way.

---

[4] Data Mining Course:
https://moodle.univie.ac.at/pluginfile.php/11513692/mod_resource/content/4/DM-3-HDData.ClusteringHighDimensionalDataPart1.pdf

3. HISC ALGORITHM

HiSC[4] is a hierarchical subspace clustering algorithm, first introduced in 2006 by Elke Achtert et al. Like PreDeCon or PROCLUS, HiSC is a top-down-based approach: Each data point gets an assigned subspace which is a subset of the d-dimensional feature space. In contrast to other subspace clustering algorithms, HiSC allows for overlapping subspace clusters, which enables the detection of hierarchies, i.e. nested cluster formations.

2.1. HiSC as OPTICS extension

In general, HiSC can be seen as a subspace extension of OPTICS, which is in itself an extension of DBSCAN. The basic approach of DBSCAN will not be explained at this point, however, a quick introduction of OPTICS is necessary since cluster extraction works for HiSC and OPTICS in a similar fashion.

OPTICS attempts to improve upon DBSCAN by sorting neighbours within the ε-neighbourhood with the k-nearest-neighbours algorithm. This tries to alleviate one of DBSCANs weaknesses: clusters of varying densities. A core distance is computed, which replaces the Boolean-based densities of DBSCAN. Secondly, reachability distances are calculated.

OPTICS connects every point greedily with other points with a priority queue: For each unprocessed point, remaining reachability distances are sorted, the closest point is then processed next. This generates an ordering, which is presented as output in the form of spanning tree (visualized as predecessor plot) and of a reachability plot (ordering on x-axis and reachability distance on y-axis):

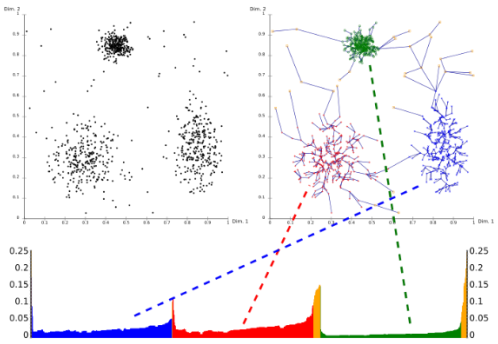

Figure 3: Visualization of a typical OPTICS output[5]: Top right figure represents the spanning tree, which is calculated if considering all predecessors per data point. The bottom figure shows the reachability plot, where clusters are segregated in the valleys.

Note that the actual cluster detection (i.e. detection of valleys in the reachability plot), is not part of the OPTICS (or HiSC) algorithm.

---

[4] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, Arthur Zimek: **Finding Hierarchies of Subspace Clusters.** Proc. 10th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'06)

[5] Source: https://de.wikipedia.org/wiki/OPTICS

As extension to OPTICS, HiSC enables to find subspace clusters of different dimensionalities. The presented reachability plot-based approach to detect clusters allows now to detect subspace cluster embeddings (lower dimensional into higher dimensional clusters), e.g. 1D clusters which are embedded in a 2D cluster in a n-dimensional feature space. Hierarchies can be assigned now based on the ordering of detected clusters.

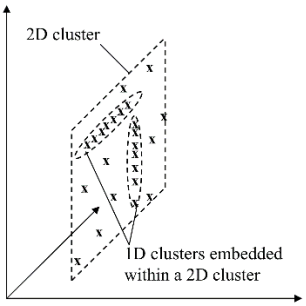

Figure 3: Hierarchically nested subspace clusters[6].

2.2. Algorithm Description

HiSC first assigns subspace preference vectors to every datapoint. Subspace variances are approximated by comparing the variance for each dimension of each point to its nearest neighbour. The neighbourhood of each point can be estimated with the k-nearest neighbors algorithm. This requires an input parameter k, which is precisely the input parameter for the kNN algorithm.

Variances are calculated with the following formula: (the nearest neighbour of point p is denoted $NN_k(p)$, $\pi_{A_i}(p)$ denotes the data distance vector from point p)

$$Var_{A_i}(NN_k(p)) = \frac{\sum_{q \in NN_k(p)} (\pi_{A_i}(q) - \pi_{A_i}(p))^2}{|NN_k(p)|}$$

Similar to PreDeCon, relevant subspaces of each data points are calculated with the help of an additional input parameter α ($\in \mathbb{R}$, between 0 and 1). This key parameter controls the jitter of subspace clusters. As a difference to PreDeCon, the resulting preference vectors ${w^i}_p$ only have Boolean entries.

$${w^i}_p = \begin{cases} 0 \;\; if \;\; Var_{A_i}(NN_k(p)) > \alpha \\ 1 \;\; if \;\; Var_{A_i}(NN_k(p)) \leq \alpha \end{cases}$$

Each point gets an assigned dimensionality integer λ_p, which is used in the main loop for sorting purposes (described later). It is calculated as the sum of zero-elements in each vector ${w^i}_p$:

$$\lambda_p = \sum_{i=1}^{d} \begin{cases} 1 \;\; if \;\; {w^i}_p = 0 \\ 0 \;\; if \;\; {w^i}_p = 1 \end{cases}$$

[6] Source: Original HiSC Publication

As the next step, the main loop is initialized with a priority queue. Similar to OPTICS, this queue has a precise ordering which will be now be described. Starting with a set which contains a random datapoint (first row is selected), the algorithm sorts all remaining data points based on 2 metrics and then greedily adds the closest point to the set. The resulting ordering is precisely the succession of data points added to this set, as seen in the reachability plot on the x-axis, starting from left to right.

In order to sort the remaining points, the following metrics have to be defined:

The subspace preference vector between 2 points p and q $w(p,q)$ counts the number of zero elements between 2 preference vectors (attribute-wise logical AND-conjunction):

$$w(p,q) = w_p \wedge w_q$$

Similar as λ_p, the integer $\lambda(\mathrm{p},\mathrm{q})$ is now the number of zero-elements in $w(p,q)$.

Based on the calculated values between 2 points, the following 2 metrics are now calculated for sorting purposes: The metric d_1 adds 1 to $\lambda(\mathrm{p},\mathrm{q})$ if the weighted Euclidean distance between 2 points exceeds α:

$$d_1 = \lambda(\mathrm{p},\mathrm{q}) + \begin{cases} 1 \; if \; max\left\{dist_{w_p}(p,q),\; dist_{w_p}(q,p)\right\} > \alpha \\ 0 \; else \end{cases}$$

The final metric d2 is defined as the inverse weighted Euclidean subspace distance (weighted on $w(p,q)$).

$$d_2 = dist_{\overline{w}(p,q)}(p,q)$$

In the main HiSC loop, the outer loop defines the last processed point, starting from a random point as described above. The inner loop now selects the closest point based on the subspace dimensionality d_1. If there are ties, the weighted distance d_2 is used as 2nd metric. Third, if d_1 and d_2 are equal, the cluster ID is used to select remaining ties.

2.2. Implementation

The HiSC algorithm was implemented in Python 3. As required dependencies, NumPy, matplotlib and sklearn are needed as 3rd party libraries. Sklearn is used for the kNN algorithm, as well as for metric calculations (e.g. NMI score).

Currently, 2 files with Python code are provided:

HiSC.py: Contains the main algorithm and functions for cluster detection & visualization (reachability plot). All defined metrics from the last section have analogous functions, see helper functions at the bottom of the file.

HiSC_sample_notebook.ipynb: Contains sample code how to invoke the algorithm with several sample demonstration inputs. Since the algorithm requires visual inspection of the reachability plot before further processing, a command line interface is not provided.

2.2. Sample inputs and result visualization

To test the validity of the implementation, a selection of sample inputs is provided in the Jupyter notebook file. With the same input parameters, we confirmed that our implementation generates the same results as the ELKI reference implementation (see log_files directory).

Based on the input file subspaces_5d.csv, the output of the current implementation will be explained. Used input parameters: alpha=0.02, k=4. First, a predecessor per data points gets assigned. This can be visualized in the predecessor plot – plotted from the perspective of 2 different input dimensions below:

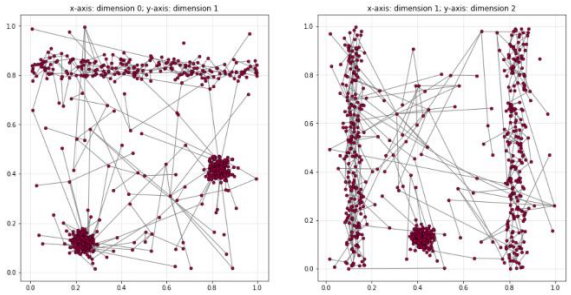

Figure 4: Predecessor plot (Spanning Tree visualization) of input file subspaces_5d.csv.

A reachability plot can be computed based on the computed ordering:

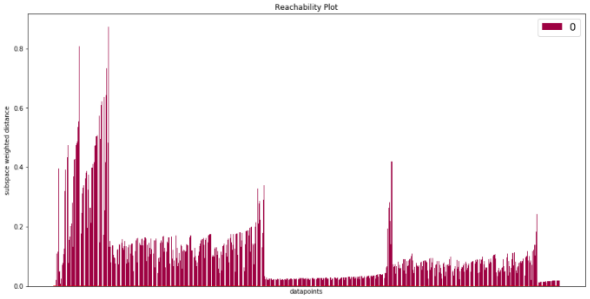

Figure 5: Reachability plot of input file subspaces_5d.csv (alpha=0.02, k=4).

Based on a threshold subspace weighted distance (y-axis in figure 5), the reachability plot can now be segregated into sections. Points can only be assigned to clusters, if the weighted distance is below the threshold and a minimum number of consecutive entries are below the threshold. The now generated cluster sections can be visualized:

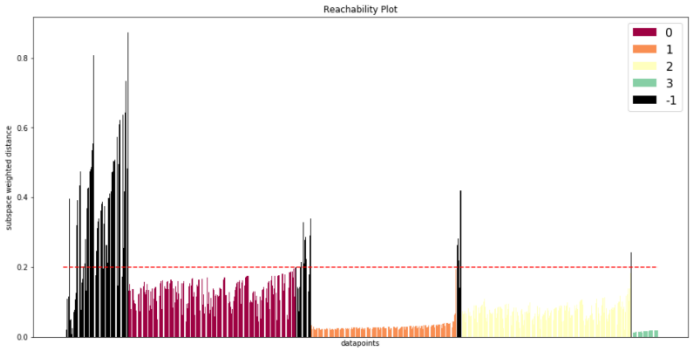

Figure 6: Reachability plot of input file subspaces_5d.csv with predicted labels (threshold distance of 0.2)

As seen in figure 4, if we apply the threshold distance 0.2, we detect 4 clusters instead of 3. A higher distance would merge 2 clusters which should not be connected. This scenario was often observed with HiSC in many tested sample datasets.

It would be possible to, for example, calculate the relationship of the resulting clusters based on centroids. Then, a dendrogram with 4 entries (e.g. average linkage based) can show that the distances between 2 of the clusters is minimal (see green and orange overlapping clusters in figure 7). However, this would not work for the general case where normality assumptions are not met. Automatic cluster detection for HiSC (or OPTICS) is also not included in ELKI. If there are suggestions for this part of the project, this will be subject to change for the final submission.

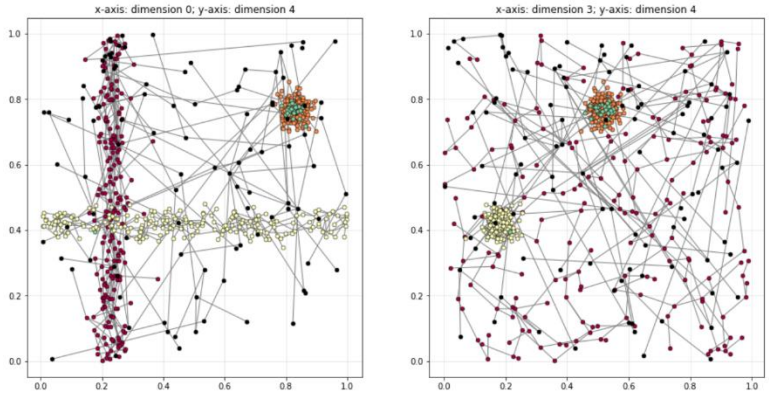

Figure 7: Predecessor plot of input file subspaces_5d.csv with empirical labels (4 instead of 3). Note that different dimensions are plotted here compared to figure 4.

2.3. Comparison Publication & ELKI implementation

The HiSC algorithm has a reference implementation in the ELKI project[7]. This implementation was provided by one of the authors of the original paper, Elke Achtert. Therefore, there should be not question that the ELKI implementation works as intended.

The ELKI implementation does not calculate the inverse weighted subspace distance – instead, d2 is not inverted and sorted descending instead of ascending. Presumably, this is done for numerical reasons. This change was also adapted for our code base.

We are unsure if the sorting of d_1 is correctly described in the original paper. ELKI sorts d_1 descending, which we adapted as well. Other sorting successions yield no valid clustering results. The sorting succession can easily be adapted in the code by changing the sorting lambda function, which now reproduces by default the behaviour of ELKI.

[7] https://elki-project.github.io/howto/clustering