Data Mining
Assignment 1: Clustering High Dimensional Data with COPAC

Group 8:
Roman Feldbauer, Elisabeth Hartel, Jiri Mauritz, Thomas Turic

November 17, 2017

1 Task

The assignment consists of implementation of one algorithm and comparison of the results with the corresponding algorithm implemented in the Environment for DeveLoping KDD-Applications Supported by Index-Structures (ELKI). For the task a correlation clustering algorithm, specifically COrrelation PArtition Clustering (COPAC) was chosen.

2 Description of COPAC

COPAC is a novel correlation clustering algorithm that does not require any presumptions concerning the number of clusters or their dimensionality. The algorithm computes the local correlation dimensionality for each object and then partitions the data set based on this dimension. It calculates a correlation distance measure which is used to cluster the objects within each partition with Generalized DBSCAN.

The first step of COPAC is to assign a local correlation dimensionality λ_p to each object p in the dataset. Assume that the cluster in which p is most likely found can be found in the local neighborhood of p. Thus we compute the set of k-nearest neighbors N_p and use it to calculate the covariance matrix Σ_{N_p}

$$\Sigma_{N_p} = \frac{1}{|N_p|} \sum_{X \in N_p} (X - \bar{X})(X - \bar{X})^T \tag{1}$$

with $\bar{X}$ the centroid of all points in N_p. The covariance matrix Σ_{N_p} is a positive semidefinite square matrix and can therefor be decomposed to the Eigenvalue matrix E_{N_p}

$$\Sigma_{N_p} = V_{N_p} E_{N_p} V_{N_p}^T \tag{2}$$

The Eigenvalue matrix consists of a diagonal matrix storing the d non-negative eigenvalues e_i of Σ_{N_p} in decreasing order. The e_i can be used to determine the local correlation dimensionality of p: the smallest number of eigenvalues e_i explaining a portion of at least α of the total variance:

$$\lambda_p = min_{r \in \{1,\dots,d\}} \left\{ r \left| \frac{\sum_{i=1}^{r} e_i}{\sum_{i=1}^{d} e_i} \geq \alpha \right. \right\} \tag{3}$$

Values for α are typically chosen between 0.8 and 0.9. The next step of COPAC is to assign the objects to a partition D_{λ_p} with each partition containing all points with the same correlation dimensionality λ_p. The partition D_d contains only noise points since there is no linear dependency among these points.

Once the objects of the database are partitioned, we can search for the correlation clusters which can be done separately for each partition. For clustering a distance measure needs to be defined that assesses how well two points share a common hyperplane. Here we start with a correlation distance matrix $\hat{M}_p$

$$\hat{M}_p = V_{N_p} \hat{E}_p V_{N_p}^T \tag{4}$$

with V_{N_p} the previously computed Eigenvector matrix (equation 2) and an adapted Eigenvalue matrix $\hat{E}_p$ with entries $\hat{e_i}$

$$\hat{e_i} = \begin{cases} 0 & \text{if } i \leq \lambda_p \\ 1 & \text{if } i > \lambda_p \end{cases} \tag{5}$$

The correlation distance matrix of p is used to derive a weighted distance measure

$$cdist_p(p, q) = \sqrt{(p - q)\hat{M}_p (p - q)^T} \tag{6}$$

Since this distance measure is based on the local neighborhood of p and therefore generally $cdist_p(p, q) \neq cdist_q(q, p)$, we need to define a correlation distance between p and q that fulfills symmetry and reflexivity:

$$cdist(p, q) = max\{cdist_p(p, q), cdist_q(q, p)\} \tag{7}$$

This correlation distance can be used to define the neighborhood predicate $NPred(p, q)$

$$NPred(p, q) \Leftrightarrow cdist(p, q) \leq \epsilon \tag{8}$$

which can be used analogously to the ϵ-neighborhood for the generalized DBSCAN algorithm. In addition, a generalized minimum weight of $N_{NPred(p)}$ can be defined to determine whether an object is a core point:

$$MinWeight(N_{NPred(p)}) \Leftrightarrow |N_{NPred(p)}| \geq \mu \tag{9}$$

These parameters are used for the run of GDBSCAN on each partition. Pseudo code for COPAC is described in algorithm 1 and for GDBSCAN in algorithm 2.

Algorithm 1 COPAC(D, k, μ, ϵ)

```
for each point p in D do
    // step 1: assign local correlation dimensionality $\lambda_p$ to each object p in D
    calculate $N_p$
    calculate covariance matrix $\Sigma_{N_p}$
    calculate $\lambda_p$ using Eigenvalue matrix $E_{N_p}$
    // step 2: partition data set based on $\lambda_p$
    assign p to partition $D_{\lambda_p}$
end for
for each partition $D_i$ with i=1 to (d-1) do
    // step 3: calculate correlation distance for each point
    for each point p in D do
        calculate correlation distance matrix $\hat{M}_p$
        for each point q in D do
            calculate $cdist_p(p, q) = \sqrt{(p - q)\hat{M}_p(p - q)^T}$
        end for
    end for
    for each point p in D do
        for each point q in D do
            define $cdist(p, q) = max\{cdist_p(p, q), cdist_q(q, p)\}$
            if $cdist(p, q) \leq \epsilon$ then
                $NPred(p, q) = TRUE$
            else
                $NPred(p, q) = FALSE$
            end if
        end for
        if $|N_{NPred(p)}| \geq \mu$ then
            $MinWeight(N_{NPred(p)}) = TRUE$
        else
            $MinWeight(N_{NPred(p)}) = FALSE$
        end if
    end for
    // step 4: for each partition determine correlation clusters using DBSCAN

    call GDBSCAN($D_i, NPred, MinWeight$)
end for
mark points in $D_d$ as noise
return cluster assignment for each point
```

Algorithm 2 GDBSCAN($D, NPred, isCorePoint$)

```
C = 0
for each unvisited point p in D do
  mark p as visited
  $N = NPred(p)$
  if isCorePoint(p,N) then
    C = next cluster
    add p to C
    for each point q in $N$ do
      if q is not visited then
        mark q as visited
        $N' = NPred(q)$
        if isCorePoint(q,N') then
          add $N'$ to $N$
        end if
      end if
      if q not member of any cluster yet then
        add q to cluster C
      end if
    end for
  else
    mark p as noise
  end if
end for
return cluster assignment for each point
```

Input parameters for COPAC are k, μ and ϵ. Parameter $k \in \mathbb{N}^+$ specifies the amount of points that determine the neighborhood of a point p. Since it is used to calculate the covariance matrix, k should not be too small to avoid an unstable matrix. It should also not be too high since we want to capture local correlation. It is recommended to use $k \geq 3d$. The parameter $\mu \in \mathbb{N}^+$ determines the minimum number of points in a cluster and should be set $\mu \leq k$. $\epsilon \in \mathbb{R}^+$ is used to specify the neighborhood predicate.

COPAC exhibits better robustness against noise and parameter settings than ORCLUS and 4C. It is also more efficient since COPAC processes each data object only once during the determination of the correlation clusters and DBSCAN is only run on partitions of the data. According to [1] COPAC has improved robustness, completeness, usability, and efficiency.

3 Implementation

We implemented COPAC in Python, making extensive use of libraries from the Scientific Python stack, that is, NumPy, SciPy, and scikit-learn. These packages provide efficient, heavily tested, and community-proven functions ranging from linear algebra (e.g. Eigen decomposition) to machine learning (e.g. nearest neighbor queries, or clustering).

The clustering algorithm COPAC is provided in the 'copac.py' module inside the 'cluster' package, which follows closely the APIs and conventions of scikit-learn. In order words, its usage is identical to other clustering methods provided by scikit-learn, and should thus be straightforward. Extensive docstrings give additional guidance for users.

Special care was taken during development for performance. The inner loops of distance matrix computations have been vectorized in order to efficiently use vector extensions of modern CPUs, which often reduces runtime by one or two orders of magnitude. Most time is spent on linear algebraic operations, which are parallelized due to NumPy's OpenMP-enabled implementation. We also use the fast Cython implementation of DBSCAN from scikit-learn. Additional tricks and tweaks were applied where possible, such as avoiding to calculate the squareroot of distances unless absolutely necessary.

To use COPAC, you may either

- just copy 'copac.py' to your working directory, or
- add the package to your PYTHONPATH, or
- install the 'cluster' package by calling

```
python3 setup.py install
```

  from the cloned directory.

Note, that we require at least Python 3.5 as well as numpy, scipy, and scikit-learn. Some evaluation notebooks may require Python 3.6 for the new and very handy f-strings. Consider using Anaconda for easy package handling.

To cluster your data, simply do the following in your Python shell/Jupyter notebook/script:

```
from cluster import COPAC
# load some X here ...
copac = COPAC(k=10, mu=5, eps=.5, alpha=.85)
y_pred = copac.fit_transform(X)
```

The docstrings offer some hints at how to choose good hyperparameters.

4 Evaluation

We evaluated the accuracy of our algorithm on both generated and real-world datasets. To compare results with some established implementation of COPAC, we also performed clustering by the ELKI implementation of COPAC algorithm on the same data with the same parameters. The latest version of ELKI package was used (0.7.1).

As a comparison metric, we are using Adjusted Mutual Info score (AMI) implemented by Scikit Learn package [1], which computes a similarity of two clusterings regardless of the data points ordering. Also, it lacks the disadvantage of mutual information score (MI), which favors clusterings with a higher number of clusters. Identical clusterings get AMI score 1.0 and independent clusterings receive AMI score around 0.0 (can be negative).

4.1 Clustering of an artificial dataset

We have generated two well-separated clusters in 2-feature space and let the COPAC algorithm perform the clustering. Figure 1 shows the result of our implementation with AMI score less than 10^{-4}. Clearly, despite the distance between the clusters, the algorithm failed to find any pattern.

Results of the ELKI implementation is in the figure 2. Both scatter plot and negative AMI score indicate as bad behavior as in the case of our implementation. This suggests that the inability to retrieve reasonable clusters from the given artificial data is rather due to limitations of the algorithm than due to implementation errors. To get better insight, we computed AMI score between both implementations to see the mutual correspondence. The score of 0.084 is not much but it is still higher than the correspondence to the true labels. While the individual clustering indices are at a chance level for both implementations, their results compared to each other are at levels above chance. This brings us to the conclusion that the algorithm cannot handle the given dataset, and both implementations fail in a similar way.

[1] `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html`

Figure 1: Our implementation of COPAC: 2-feature space clustering. Parameters: $k = 6, \mu = 6$ and $eps = 2$

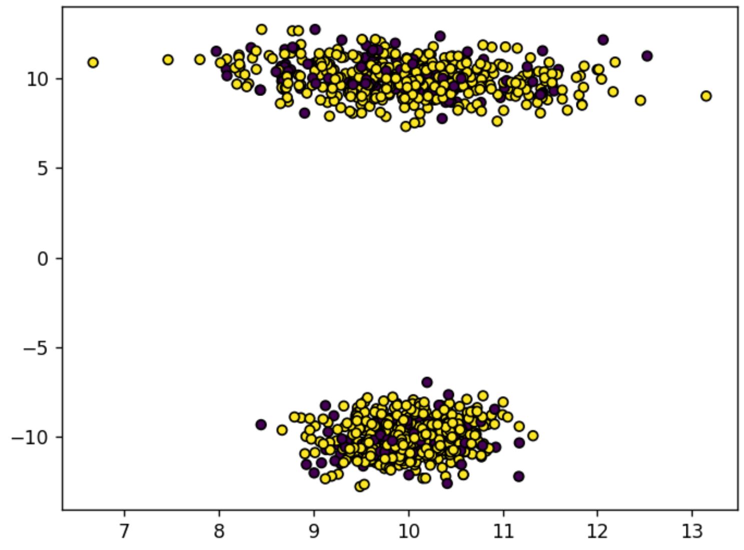


Figure 2: ELKI implementation of COPAC: 2-feature space clustering. Parameters: $k = 6, \mu = 6$ and $eps = 2$

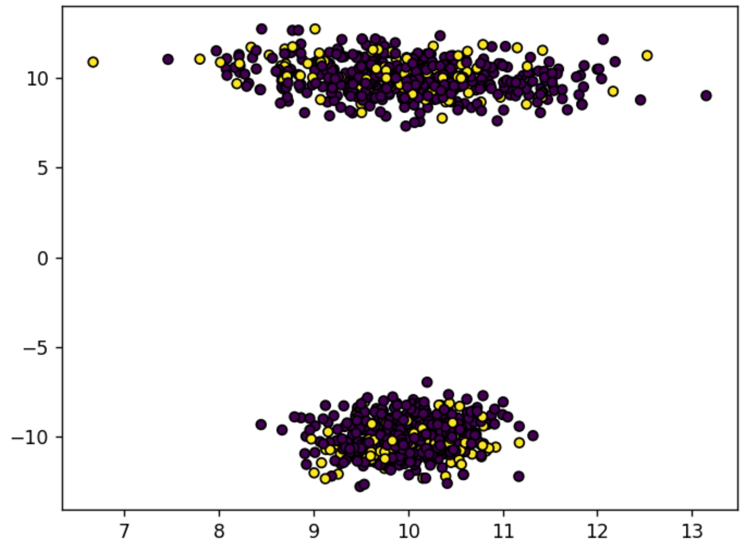
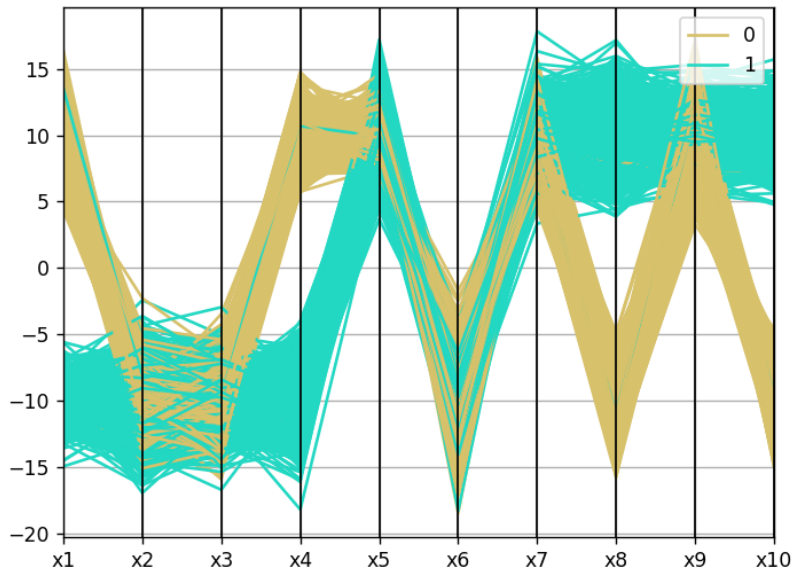


Figure 3: True labels of 10-feature space in parallel coordinates.



As a next step, we generated clusters in 10-dimensional space to see the behavior in high dimensional data. Since it is not possible to visualize results by scatter plot, we present them in the parallel coordinates. The figure 3 displays generated clusters with the true labels. We can see that the clusters are defined by the features x1, x4, x8 and x10. Other features do not contribute to the cluster separation. Along with the dimensionality, we have to adjust the parameters as well: $k = 30, \mu = 30, eps = 10$.

In the figure 4, we can see rapid improvement of our implementation in comparison to the 2-feature space. The points are separated by color in the crucial features. The cluster 2 corresponds to the cluster 0 from the figure 3 and 1 corresponds to 1. Except for the noise, the algorithm found one more cluster, though with a low number of points. AMI score equals to 0.433, which is significantly higher than the score of the 2-feature clustering.

ELKI implementation (figure 5) found five clusters instead of two, however, we can still find the correspondence to the true labels after the unification of some clusters. AMI score is as high as 0.563, which means it scores better than our implementation.

The mutual AMI score of the implementations is 0.553, hence, they produced relatively similar results. We can conclude, that the COPAC algorithm generally behaves better in high dimensional spaces.

Figure 4: Our implementation of COPAC: 10-feature space clustering

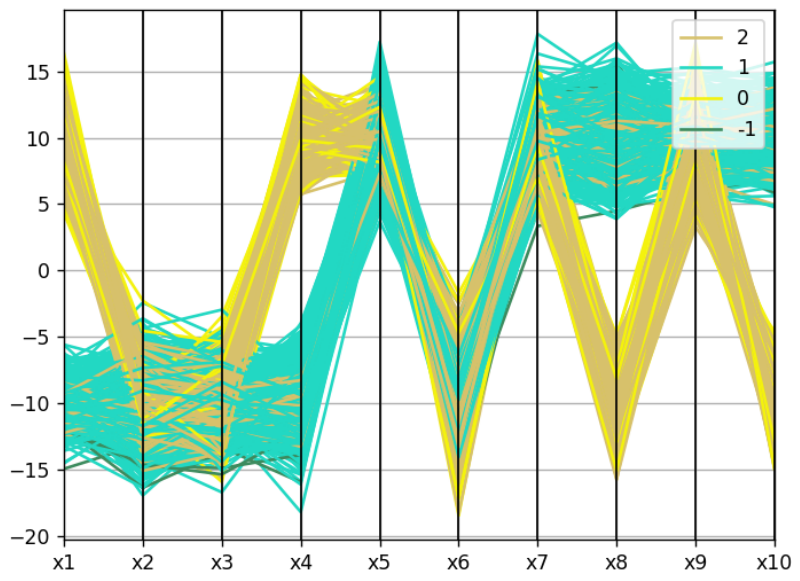
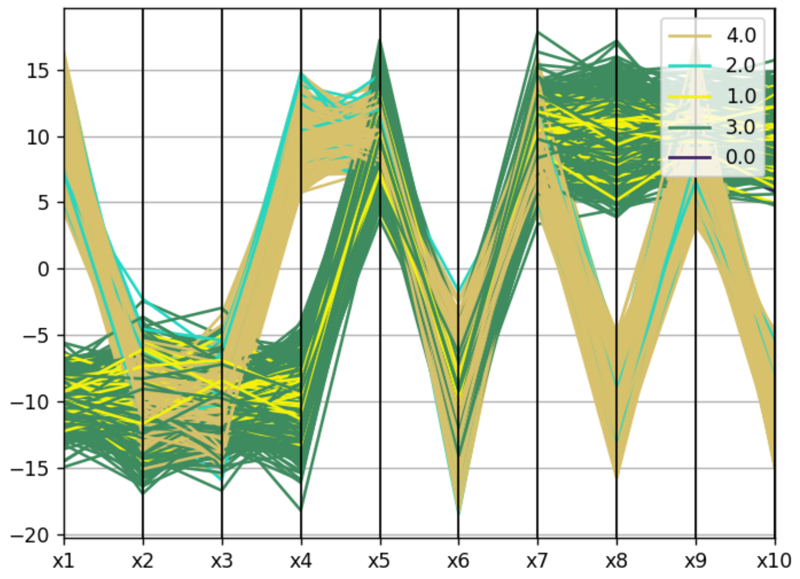


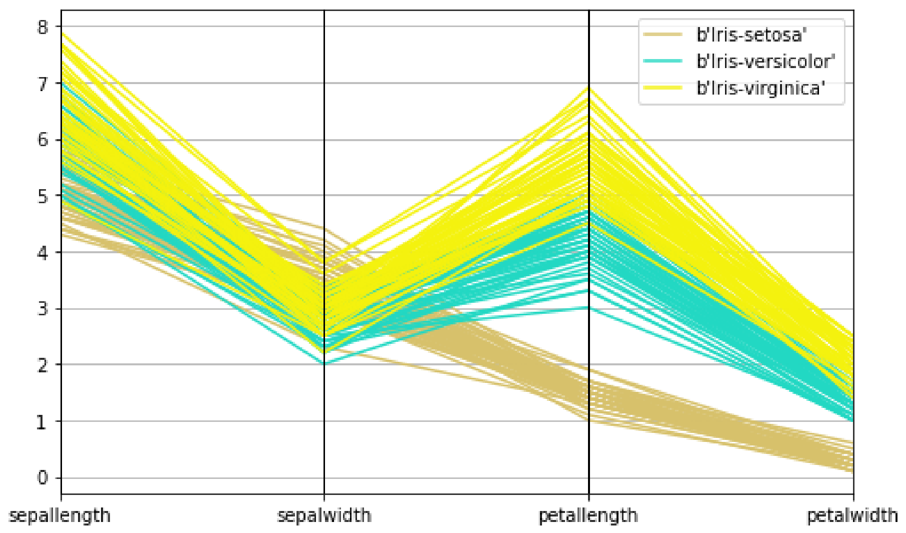
Figure 5: ELKI implementation of COPAC: 10-feature space clustering



4.2 Clustering of a real-world dataset

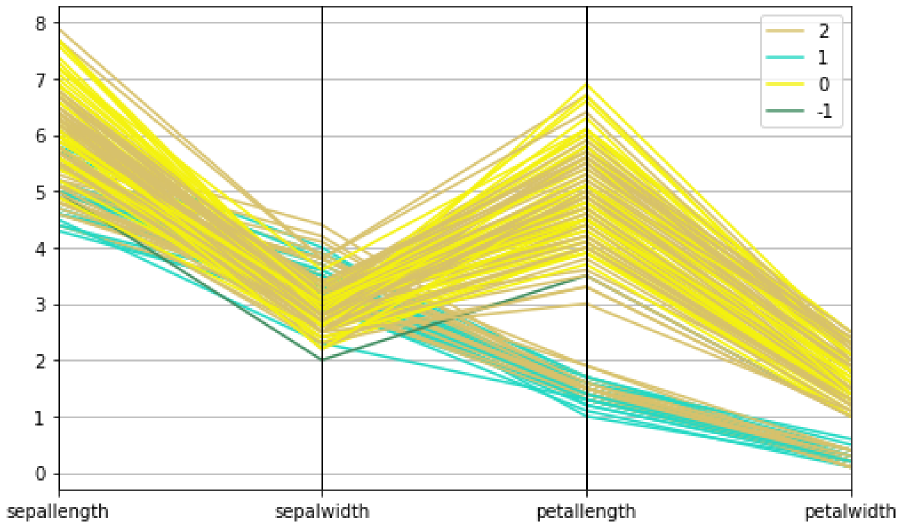
For the evaluation on the real-world data, we have chosen Iris – one of the most well-known datasets for clustering. Iris contains four features regarding the shape of a flower and is clustered into three final classes of the plant's species. Figure 6 shows the true clusters of the dataset.

Figure 6: True labels of Iris dataset



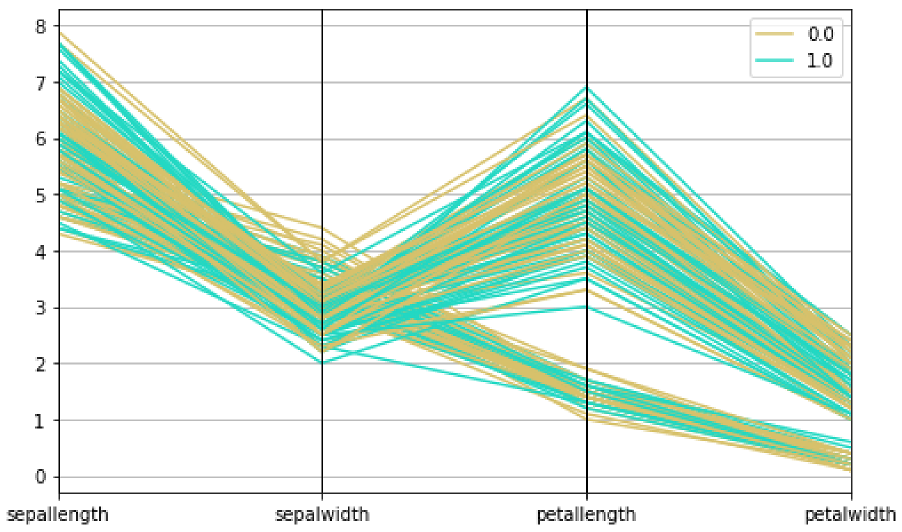
We used the following parameters to cluster the Iris dataset: $k = 8, \mu = 8, eps = 1$. Figure 7 contains the results of our implementation. The algorithm has chosen correctly 3 clusters (dark green lines are noise entries). We can see that clusters 0 and 1 are well separated, however, the cluster 2 interfere with both clusters in a random manner. The *versicolor* class is not well recognized from *virginica*. The AMI score of 0.22 shows that the algorithm performed poorly, even though there were some patterns recognized.

Figure 7: Our implementation of COPAC: Iris dataset



ELKI version of COPAC achieved AMI score 0.013 and found only two clusters, as we can see in the figure 8. In this case, our implementation performed better. Mutual AMI score is 0.07, which is very similar to the 2-feature space clustering of artificial data. Again, we can notice the trend of worse performance for low dimensional inputs.

Figure 8: ELKI implementation of COPAC: Iris dataset



5 Summary

COPAC is superior to other correlation clustering algorithms as there are no assumptions needed concerning the number of clusters or their dimensionality. It works therefore quite well on clusters of different dimensionalities. This requires of course data of high dimensionality. As can be seen in our evaluation, for data with low dimensions and consequently clusters with shared dimensionality COPAC is therefore not the best choice. Another example with clusters of different dimensionality is the synthetic dataset in the reference paper [1] which shows the advantage of COPAC in separating even intersecting clusters.

Our implementation in python compares quite well to the ELKI implementation of COPAC written in Java, as the mutual information scores of both implementations were quite similar in our evaluation. This shows that our implementation matches with the ELKI implementation on a qualitative level.

References

[1] Achtert, E and Bohm, C and Kriegel, H P and Kroger, P and Zimek, A, *Robust, Complete, and Efficient Correlation Clustering*. Proceedings of the Seventh Siam International Conference on Data Mining, pages 413-418. 2007.