

# Data Mining

## Programming Assignment 1: Subspace Clustering

Sonja Biedermann      Thilo Hannes Fryen  
Chan Dat Dennis Huyen      David Turner

November 17, 2018

### Contents

<b>1</b>	<b>ORCLUS</b>	<b>1</b>
1.1	Description and Pseudocode . . . . .	1
1.2	Implementation . . . . .	2
<b>2</b>	<b>Evaluation</b>	<b>2</b>
2.1	Methodology . . . . .	3
2.2	Simple dataset . . . . .	3
2.3	Simple dataset with added noise . . . . .	4
2.4	Clusters that are noise in some subspaces . . . . .	4
2.5	Clusters with different rotations in some subspaces . . . . .	4
2.6	Summary . . . . .	4

## 1 ORCLUS

We decided to implement the ORCLUS algorithm. ORCLUS is a spin on  $k$ -means which utilizes eigensystems to adapt the distance measure to non axis parallel correlation clusters.

We picked this algorithm because the paper is easy to read and contains good pseudocode which is of utmost importance when trying to actually implement the described algorithm.

Furthermore, since  $k$ -means was the first clustering algorithm we studied, it will forever have a special Gaussian-shaped spot in our heart—which makes ORCLUS, as a spiritual successor, a natural choice.

### 1.1 Description and Pseudocode

The algorithm proceeds in rounds in which the dimensionality of the clusters and the number of clusters is gradually reduced. The rate at which this happens should not be too

quickly, to which end the authors propose two parameters  $\alpha$  and  $\beta$  and a relation between these two, by which one can be calculated from the other. They suggest an  $\alpha$  of 0.5, which we've also adopted.

In each round, the algorithm undertakes an assignment step which is the same as in  $k$ -means. Every point is assigned to the closest centroid. However, as a next step, the eigenvectors associated with the smallest spread are computed on a per cluster basis—the rationale behind this is that those vectors define a subspace in which the points cluster well, i.e. have a low contribution to the overall cluster energy. The cardinality of this vector set dictates the dimensionality of the subspace to which the points are projected.

To reduce the amounts of clusters—the algorithm starts with more seeds than requested by the user, we've chosen to start with 5 times as many as the authors offer no suggestions—a merging step is performed next. Although quite lengthy, this operation is pretty simple. The objective is to find pairs of clusters which can be merged such that the overall cluster energy stays low. For this, all pairs of clusters are examined, their centroid and energy computed and then the cluster-pairs with the lowest energy are picked to be merged, being careful to update all relevant data structures after each merge.

These three steps are repeated until the desired dimensionality and number of clusters are reached. After a final assignment step the clustering is returned.

## 1.2 Implementation

We chose to implement the algorithm in Python 3. The implementation is rather straight forward and only depends on NumPy and some functionality from Python's standard library.

Notably we use `numpy.linalg.eigh` to decompose the cluster matrices into eigenvalues and eigenvectors. We know PCA could also be used but online literature<sup>1</sup> suggests that the LAPACK routines utilized in NumPy's implementation of `eigh` perform a tiny sliver better than SVD, which we expect to be used in `sklearn.decomposition.PCA`. However, this probably would make no practical difference whatsoever.

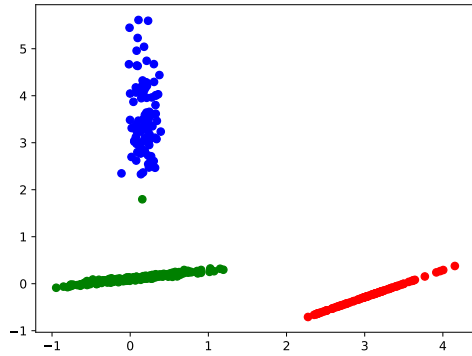
The initial seeds are distributed using the `kmeans++` initialization strategy, although we have also implemented a completely random initialization strategy. If the initial seed count is chosen to be high enough, this strategy actually seems to be beneficial, as it returns good partitions and is also faster.

## 2 Evaluation

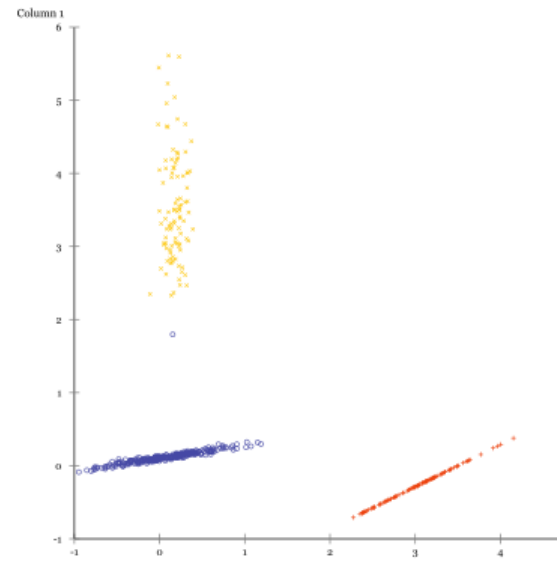
We will be comparing ourselves to the ELKI implementation on 4 synthetic datasets using the NMI as scoring method. The datasets were generated using ELKI's data generator. We used flat Gaussian distributions as correlation clusters and added noise points to disturb the algorithm.

---

<sup>1</sup><https://stackoverflow.com/questions/50358310/how-does-numpy-linalg-eigh-vs-numpy-linalg-svd>



(a) Our implementation



(b) ELKI's implementation

Figure 1: Bla

## 2.1 Methodology

The evaluation should be fully reproducible. File `eval.sh` contains the calls made to our test scripts and ELKI. `summary.rb` is a Ruby script which processes the results, outputs an overview table and also generates a `gnuplot`-ready plot file for plotting a boxplot of the results.

We execute the test runs 5 times and note their resulting NMIs as well as the clusterings with the best NMI score.

## 2.2 Simple dataset

As a simple first dataset we feed a 2-dimensional data set containing 3 clusters that are very clearly separated, one of which is a bit fatter than the stereotypical correlation cluster, but which is axis-parallel. Figure 1 shows the results obtained by our implementation (Figure 1a) and the result obtained by the implementation contained in ELKI. They are identical—notably misclassifying one single point which would belong to the fat axis-parallel cluster.

dataset	DM			ELKI		
	min	max	avg	min	max	avg
test	0.987	0.987	<b>0.987</b>	0.537	0.987	0.811
test noisy	0.536	0.66	<i>0.634</i>	0.537	0.659	0.634
higher dimensional	0.271	0.323	<b>0.297</b>	0.188	0.356	0.258
paper	1.0	1.0	<b>1.0</b>	0.005	1.0	0.559

Table 1: Overview of results

### 2.3 Simple dataset with added noise

### 2.4 Clusters that are noise in some subspaces

### 2.5 Clusters with different rotations in some subspaces

### 2.6 Summary

Table 1 shows the results of our evaluation. The minimum, maximum and average over the 5 runs are shown split between our implementation (DM) and the implementation available in ELKI.

Interestingly, our implementation appears to have a slight advantage over ELKI’s. Notice especially the discrepancy with dataset **paper**, where ELKI’s implementation appears extremely unstable, while ours uncovers the cluster structure perfectly without fail. The biggest difference in our implementation is that we use a different initialization strategy, namely `kmeans++`—this might result in our algorithm having an easier time converging since the cluster centres are well-separated. However, our implementation performed better even when using a random seed picking strategy—we suspect that perhaps the eigenvalue decomposition is implemented differently, or we are just lucky.

Figure 2 shows a boxplot of the obtained

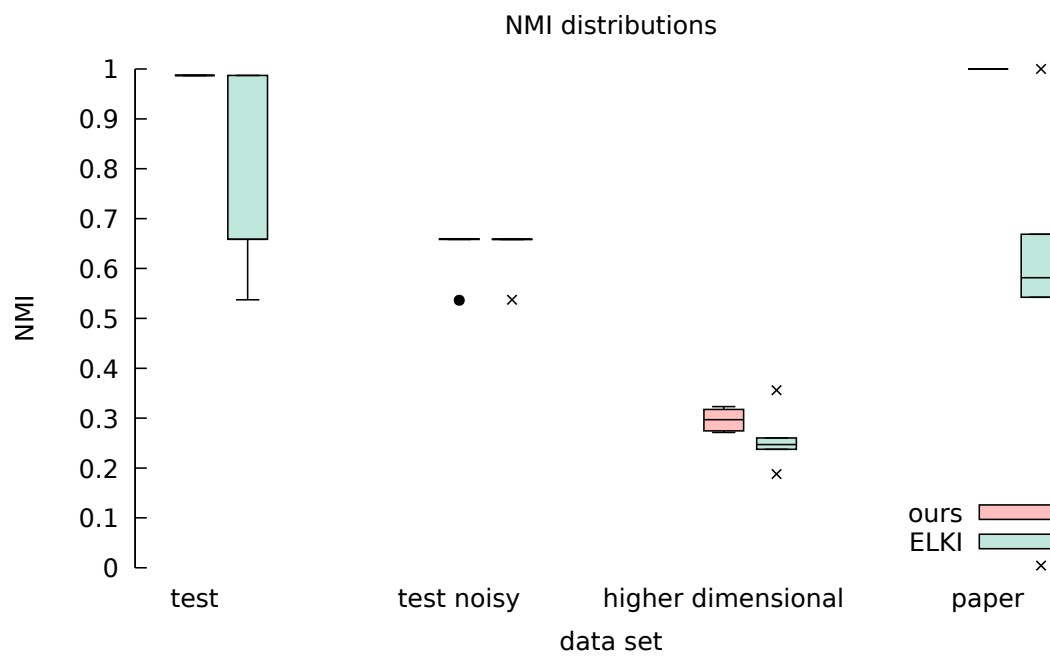


Figure 2: Boxplot illustrating stability