Data Mining
Programming Assignment 1: Subspace Clustering

Sonja Biedermann Thilo Hannes Fryen
Chan Dat Dennis Huyen David Turner

November 16, 2018

Contents

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 α and β and a relation between

these two, by which one can be calculated from the other. They suggest an α 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.

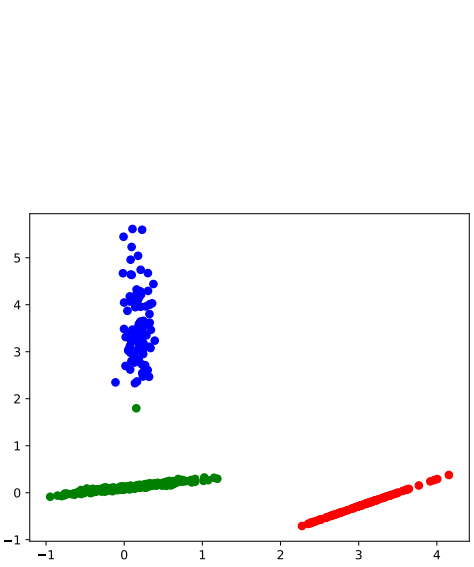
Notably we use `numpy.linalg.eigh` to decompose the cluster matrices into eigenvalues and eigenvectors. We know PCA could also be used but online literature[1] suggests that the LAPACK routines utilized in NumPy's implementation of `eigh` perform a tiny sliver better than SVD, which we expect to be used 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 initalization 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.

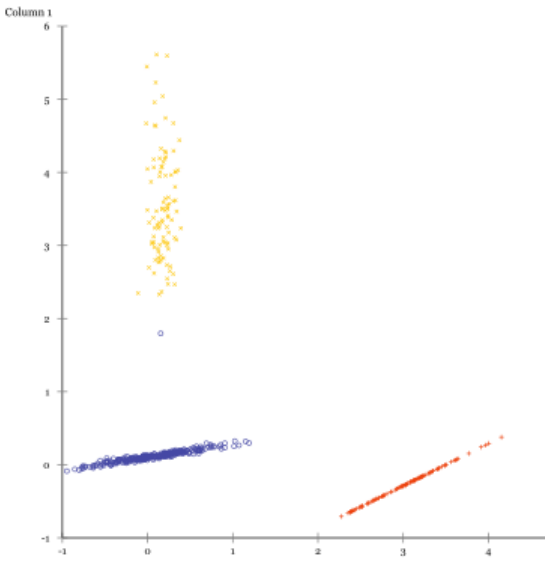
2 Evaluation

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.

[1] `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 Dataset 1: Small

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.

2.2 Dataset 2: Small with Noise

2.3 Dataset 3: Clusters that are noise in some subspaces

2.4 Dataset 4: Clusters with different rotations in some subspaces