Documentation - PreDeCon Algorithm

Group 3: Brandl Moritz, Miklautz Lukas, Mitsch Raphael

November 19, 2017

1 Introduction

In this paper the PreDeCon (subspace PREference weighted DEnsity and CONnected clustering) algorithm is explained. The algorithm was created by Böhm, Christian, et al.[BRKK04] in 2004. PreDeCon is a clustering algorithm that was developed to perform well in high-dimensional feature spaces. High-dimensional feature spaces cause many clustering algorithms to break down, due to the curse of dimensionality. In this case this refers to the phenomenon, that clustering algorithms which calculate distances in the full dimensional space deteriorate in performance, because the distances in such spaces lose their meaning. That's why specialized clustering algorithms were developed, which are based on the observation that clusters exist only in specific subspaces of the full feature space. There are several approaches to this problem, two of them will be explained here briefly. Subspace clustering tries to find all possible clusters in all axis-parallel subspaces in a bottom-up way, here it is possible for clusters to overlap, this means that points can be assigned to multiple clusters, see [AGGR98] and [KKK04]. In projected clustering all points are uniquely assigned to clusters or noise in a top-down approach, therefor clusters cannot overlap.

2 The PreDeCon Algorithm

PreDeCon belongs to the group of projected clustering algorithms. It follows an instance-based approach, which calculates for each point in the data base a subspace preference in the entire feature space and merges the points with similar preferences together to form a cluster. The rational behind this approach follows the locality assumption, which states that "the subspace preference can be learned from the local neighborhood in the d-dimensional space", as mentioned in the courses slides. PreDeCon is based on the density-based clustering algorithm DBSCAN [EKS+96] and can be seen as an extension of it for high-dimensional spaces. PreDeCon has the following features[BRKK04]:

- the clustering result is determinate
- it is robust against noise
- it has a worst-case time complexity of $O(dn^2)$

2.1 Underlying Ideas

The underlying idea of PreDecon is the "subspace preference cluster" [BRKK04], which expands the idea of density connected set of points by a subspace preference vector. Note, that from now on all points p, q are considered as elements of the data base. The subspace preference vector skews the ϵ-neighborhood of point p in the direction of the lowest attribute variance, where the attribute variance of a point p is given by:

$$VAR_{A_i}(N_\epsilon(p)) = \frac{\sum_{q\in(N_\epsilon(p))}(dist(\pi_{A_i}(p), \pi_{A_i}(q)))^2}{|N_\epsilon(p)|}$$

From the attribute variance the subspace preference vector of a point p is derived as

$$w_i = \begin{cases} 1 & \text{if } VAR_{A_i}(N_\epsilon(p)) > \delta \\ \kappa & \text{if } VAR_{A_i}(N_\epsilon(p)) \leq \delta \end{cases}$$

and the preference weighted similarity measure associated with a point p can then be calculated as a simple weighted Euclidean distance [BRKK04]. Here the parameter δ specifies the threshold for a low variance. Due to the weighting factor this similarity measure is asymmetric, this means that $dist_p(p, q) = dist_p(q, p)$ does not hold anymore. This can be corrected by using the maximum of the two distances, see [BRKK04]. Look at figure 1, to see the consequences of the preference weighted Euclidean distance. With this new weighted distance function the preference weighted ϵ-neighborhood for each point can be calculated. Another important concept which has been adapted from DBSCAN are the preference weighted core points. Similar to the definition of core points in DBSCAN a preference weighted core point needs to have a minimum number of points in its preference weighted ϵ-neighborhood, denoted by the parameter μ. The PreDeCon algorithm allows the user to specify additionally a maximum preference dimensionality of the ϵ-neighbourhood of a point, denoted as λ. The last notion in order for PreDeCon to work is the direct preference weighted reachability. It states that a point p is direct preference weighted reachable from a point q if q is a preference weighted core point, the preference weighted dimensionality of the ϵ-neighborhood of point p is smaller or equal to λ and p is in the preference weighted epsilon neighborhood of q. For a more detailed explanation of these concepts refer to [BRKK04].

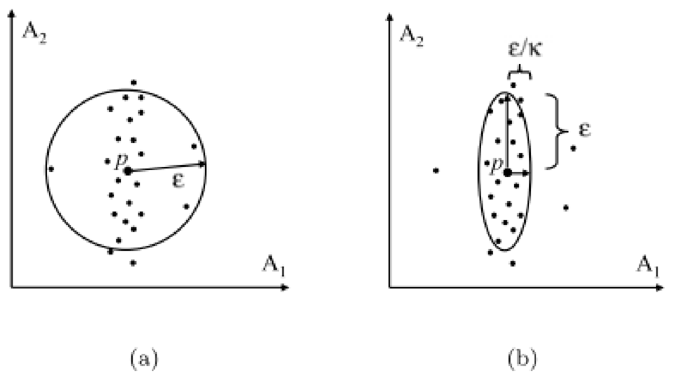


Figure 1: ϵ-neighborhood of p according to (a) simple Euclidean and (b) preference weighted Euclidean distance.(Figure taken from [BRKK04])

```
algorithm PreDeCon($\mathcal{D}$, $\varepsilon$, $\mu$, $\lambda$, $\delta$)
  // assumption: each point in $\mathcal{D}$ is marked as unclassified
  for each unclassified $o \in \mathcal{D}$ do
    if $\text{CORE}^{\text{pref}}_{\text{den}}(o)$ then        // expand a new cluster
      generate new clusterID;
      insert all $x \in \mathcal{N}^{\bar{w}_o}_{\varepsilon}(o)$ into queue $\Phi$;
      while $\Phi \neq \emptyset$ do
        $q$ = first point in $\Phi$;
        compute $\mathcal{R} = \{x \in \mathcal{D} \mid \text{DIRREACH}^{\text{pref}}_{\text{den}}(q, x)\}$;
        for each $x \in \mathcal{R}$ do
          if $x$ is unclassified then
            insert $x$ into $\Phi$;
          if $x$ is unclassified or noise then
            assign current clusterID to $x$
        remove $q$ from $\Phi$;
    else        // $o$ is noise
      mark $o$ as noise;
end.
```

Figure 2: Pseudo code of the PreDeCon algorithm.(Figure taken from [BRKK04])

2.2 The Algorithm

PreDeCon needs the following user specified parameters[BRKK04]:

- μ: Minimal number of points in ϵ-neighborhood
- ϵ: Distance parameter for neighborhood calculation
- δ: Variance threshold for subspace preference clusters
- λ: Threshold for dimensionality of ϵ-neighbourhood of a point
- κ: Weight for subspace preference vectors

In figure 2 the pseudo code of the original paper is shown. The inner workings of this algorithm are explained in the next section, when the implementation of PreDeCon are discussed.

3 The Pseudocode of our implementation

```
# Determine necessary properties.
self.calculate_epsilon_neighbourhoods()
self.calculate_variances_along_attributes()
self.calculate_subspace_preference_dimensionalities()
self.calculate_subspace_preference_vectors()
self.calculate_preference_weighted_similarity_measures()
self.calculate_preference_weighted_epsilon_neighbourhood()
self.calculate_preference_weighted_core_points()

# Use properties in order to assign clusters.
cluster_id = 0
# Loop over all points
for point in range(0, self.num_points):
    if not PreDeCon.is_classified(point, self.clusters):
        if point in self.core_points:
            core_point_i = point
            # Expand new cluster
            cluster_id += 1
            self.clusters[cluster_id] = []
            queue = deque(self.preference_weighted_neighbourhoods[core_point_i])
            # loop through points in preference_weighted_neighbourhood
            # of the core_point_i
            while queue:
                q_point = queue.popleft()
                # Get all points which are direct preference reachable
                # from point q: Definition 8:
                # DIRREACH(q,x) <=> q is core point,
                #                   PDIM(N_e(x)) <= lambda,
                #                   x is in preference weighted neighbourhood of q
                direct_preference_reachable_from_point_q = []
                if q_point in self.core_points:
                    for x_point in range(0, self.num_points):
                        if (self.subspace_preference_dimensionalities[x_point]
                            <= self.param_lambda) and \
                            (x_point in self.preference_weighted_neighbourhoods[q_point]) and \
                            (not PreDeCon.is_classified(x_point, self.clusters)):
                            # only insert points which are not classified yet
                            direct_preference_reachable_from_point_q.append(x_point)
                else:
                    # A point is always reachable from it self
                    direct_preference_reachable_from_point_q.append(q_point)
                # Loop over direct preference reachable points
                # and assign unclassified points to the cluster
                for reachable_point in direct_preference_reachable_from_point_q:
                    if not PreDeCon.is_classified(reachable_point, self.clusters):
                        queue.append(reachable_point)
                    if (not PreDeCon.is_classified(reachable_point, self.clusters)) or \
                            (reachable_point in self.clusters["noise"]):
                        self.clusters[cluster_id].append(reachable_point)
                        # points are uniquely assigned
                        if reachable_point in self.clusters["noise"]:
                            self.clusters["noise"].remove(reachable_point)
        else:
            self.clusters["noise"].append(point)
```

4 Parameter Tuning

To compare and benchmark our algorithm we took the "Vary Density" data set from the Elki homepage. The ground truth can be seen in Figure 3

In order to find a suitable tuple of parameters, we first tried guessing a somewhat good combination of parameters and then do a grid search around those parameters to find the best combination. Therefore we first came up with following parameters as shown in Figure 4:

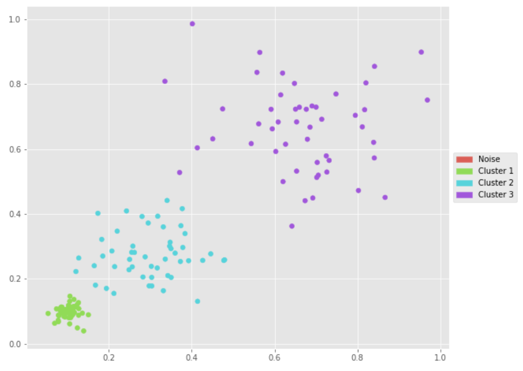


Figure 3: Ground truth for cluster assignments of data set Vary Density

1. epsilon: 0.3
2. mu: 10
3. delta: 0.1
4. kappa: 20
5. lambda: 2

By feeding the same parameter combination into elki we obtain the same clusters, shown in Figure 5

We then applied two different clustering performance evaluation methods onto the several models we calculated. We used Adjusted-Normalized-Information (AMI) and also V-Measure. Both measurements resulted in different "best" parameter combinations. While the AMI chose our initial parameter settings to be the best (Figure 4), the V-Measurement chose a completely different model. (Figure 7). The AMI is defined as an adjustment of the Mutual Information (MI) score to account for chance and it is independent of the absolute values of the class labels, meaning permutaitons will not change the score value. [PVG+11]

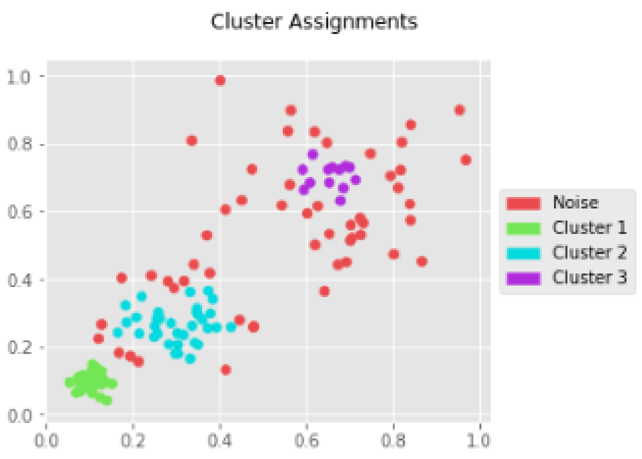


Figure 4: Result of PreDeCon-algorithm with initial parameters

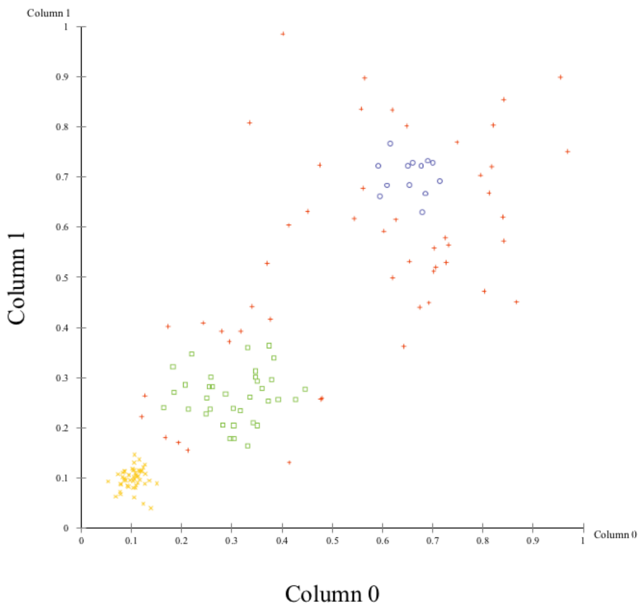
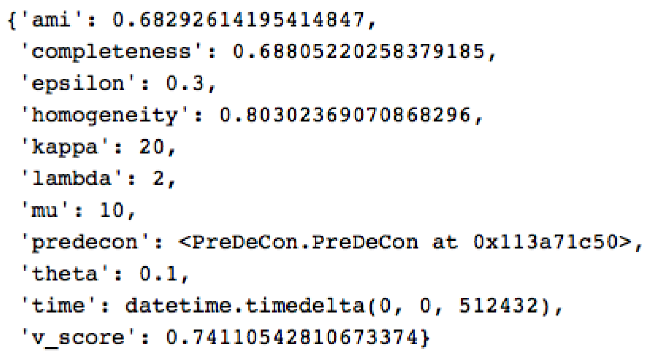


Figure 5: Result of PreDeCon-algorithm in ELKI with initial parameters

When looking at Figure 8 we can see that according to the V-Measurement the best score is achieved by 7 clusters. In addition to the v-score we also have the completeness and homogeneity measures which both are actually used to calculated the v-score. by definition, the homogeneity measures if each cluster contains only members of a single class. We achieved a homogeneity of 0.9683 as can be seen in Figure 7. In the original data set, there are only 3 clusters and no noise points. In this clustering, the few noise points which are basically assigned to the cluster "noise" come from different ground-truth clusters. every other cluster is within only one of the ground-truth clusters. therefore this score is almost 1. On the other hand completeness is defined as: all members of a given class are assigned to the same cluster. as there are more than 3 clusters, this cannot be true, so this is why this score is at 0.6595 (see Figure 7).

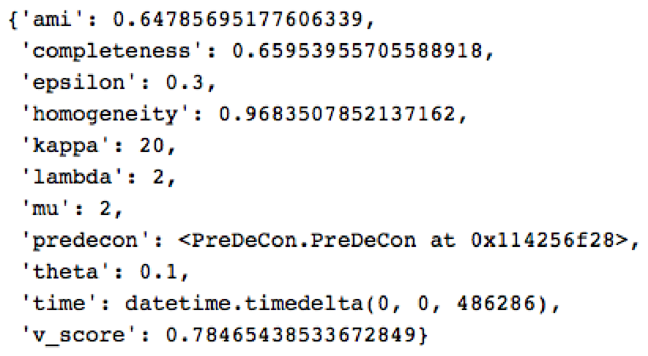
5 Benchmarking

We can already see that the clustering of both the PreDeCon implementation in elki as well as our own have the same result. Though, when it comes to runtime we are far worse off than elki. Elki needs about 6ms to finish the clustering of Figure 7, while our algorithm needs 486ms.



```
{'ami': 0.68292614195414847,
 'completeness': 0.68805220258379185,
 'epsilon': 0.3,
 'homogeneity': 0.80302369070868296,
 'kappa': 20,
 'lambda': 2,
 'mu': 10,
 'predecon': <PreDeCon.PreDeCon at 0x113a71c50>,
 'theta': 0.1,
 'time': datetime.timedelta(0, 0, 512432),
 'v_score': 0.74110542810673374}
```

Figure 6: Best result when looking at adjusted mutual information



```
{'ami': 0.64785695177606339,
 'completeness': 0.65953955705588918,
 'epsilon': 0.3,
 'homogeneity': 0.9683507852137162,
 'kappa': 20,
 'lambda': 2,
 'mu': 2,
 'predecon': <PreDeCon.PreDeCon at 0x114256f28>,
 'theta': 0.1,
 'time': datetime.timedelta(0, 0, 486286),
 'v_score': 0.78465438533672849}
```

Figure 7: Best result when looking at v score

References

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[BRKK04] Christian Bohm, K Railing, H-P Kriegel, and Peer Kroger. Density connected clustering with local subspace preferences. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 27–34. IEEE, 2004.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[KKK04] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 246–256. SIAM, 2004.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

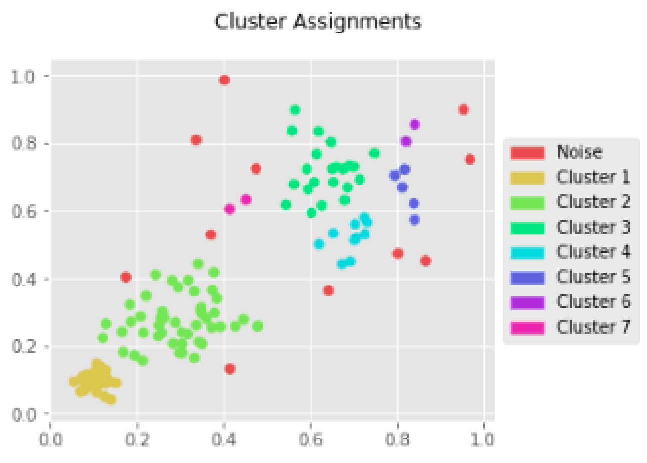


Figure 8: Clustering when choosing parameters of v-score winner