Fast k-Nearest-Neighbor-Consistent Clustering

Lars Lenssen[1], Niklas Strahmann[1] and Erich Schubert[1]

[1]TU Dortmund University, Informatik VIII, Dortmund, 44221, Germany

Abstract

There are many ways to measure the quality of a clustering, both extrinsic (when labels are known) and intrinsic (when no labels are available). In this article, we focus on the k-Nearest-Neighbor Consistency measure, which considers a clustering as good if each object is within the same cluster as its nearest neighbors, and hence does not need labels. We propose a variant of the K-means clustering algorithm that uses the k-Nearest-Neighbor Consistency as a constraint while optimizing the sum-of-squares as in regular K-means, resulting in K-means clustering where the nearest neighbors are guaranteed to be in the same cluster. The new version provably yields the same results as the original consistency-preserving K-means algorithm of Ding and He, but needs fewer computation.

Keywords

k-Nearest-Neighbor Consistency, Cluster Analysis, Clustering-Quality Measure

1. Introduction

Evaluating the quality of a result in unsupervised learning is difficult, as we do not have labels. Many different clustering objectives have been proposed, and we have just as many different evaluation measures. Bonner [1] noted that "none of the many specific definitions that might be given seems 'best' in any general sense", and Estivill-Castro [2] wrote that clustering is "in the eye of the beholder", and every researcher and user may have different believes on what constitutes a cluster. Common principles for clustering include that nearby objects should be in the same cluster, whereas far away objects should belong to different clusters. In agglomerative hierarchical clustering, for example, we always merge the closest two clusters, K-means assigns points to the nearest cluster, DBSCAN connects neighboring points in dense areas, and spectral clustering attempts to find a minimum cut of the nearest-neighbor graph. Similar principles can be found in clustering quality measures (CQM) [3] such as the Silhouette [4], the Davies-Bouldin index [5], the Variance-Ratio criterion [6], the Dunn index [7], and many more. The Silhouette, for example, compares the average distance to points of the same cluster with the average distance to points of the nearest other cluster. Any unsupervised clustering quality measure implies an "optimal" clustering, that could also be found by exhaustive enumeration; whereas clustering algorithms often only find an approximation to the "optimal" clustering, but in an acceptable run time. For example, K-means may find different solutions when run multiple times because it gets stuck in local optima. Similarly, K-medoid algorithms based on swapping

LWDA'23: Lernen, Wissen, Daten, Analysen. October 09–11, 2023, Marburg, Germany

lars.lenssen@tu-dortmund.de (L. Lenssen); niklas.strahmann@tu-dortmund.de (N. Strahmann); erich.schubert@tu-dortmund.de (E. Schubert)

0000-0003-0037-0418 (L. Lenssen); 0009-0000-1034-6097 (N. Strahmann); 0000-0001-9143-4880 (E. Schubert)

can get stuck in local optima [8]. We can also optimize the Medoid Silhouette in a similar fashion as an example where an evaluation criterion was turned into an efficient algorithm [9]. But an undesired trivial clustering may score well for surprisingly many quality measures. For example the within-cluster-sum-of-squares (WCSS) criterion used by K-means clustering is optimal when every point is its own cluster. Hence we may need to add additional constraints such as keeping the number of clusters fixed. In other cases, it is common to use one criterion to find candidate solutions (e.g., optimizing WCSS with K-means), and then use a different criterion for model selection.

In this article, we are interested in a relatively unknown quality criterion called the k-Nearest-Neighbor Consistency [10], which measures if the nearest neighbors of each point belong to the same cluster. This is closely related to the central idea of clustering that similar objects should be in the same cluster, but also to nearest-neighbor classification, as we would then predict the correct class. Ding and He [10] also proposed two clustering algorithms to optimize for this criterion. In this article, we propose a fast variant of the consistency-preserving K-means algorithm, that provably yields the same results.

We first introduce nearest-neighbor consistency in Section 2 and clustering with consistency in Section 3. We then propose a faster algorithm in Section 4. We show experimental evidence of the performance benefits in Section 5 and conclude the paper in Section 6.

2. k-Nearest-Neighbor Consistency

The k-Nearest-Neighbor Consistency is motivated by k-Nearest-Neighbor classification, but it can also be used to evaluate clustering validity, as proposed by Ding and He [10]. In the following, we only consider clustering that form a total partitioning of the data set, encoded as a cluster label l_i for each sample i. Overlapping, hierarchical, or incomplete clustering (e.g., noise in DBSCAN) solutions are not considered. For the given samples $X = \{x_1, \dots, x_n\}$, and the cluster labels $L = \{l_1, \dots, l_n\}$, the $k\mathrm{NN}(x_i)$ are the k nearest neighbors of x_i. One single sample i is considered to be k-nearest-neighbor consistent if all neighbors $p \in k\mathrm{NN}(x_i)$ have the same cluster label as the sample i:

$$kc_i(X, L) = \begin{cases} 1 & \text{if } \forall x_j \in k\mathrm{NN}(x_i) \,:\, l_j = l_i \\ 0 & \text{otherwise} \end{cases}$$

To measure the kNN consistency of a clustering where not all points are consistent, Ding and He [10]also use a fractional kNN consistency

$$KC(X, L) = \frac{1}{n} \sum_{i=1}^{n} kc_i(X, L).$$

To use the consistency as a quality measure, Handl and Knowles [11] developed a connectivity-based measure out of the kNN consistency, which gives more emphasis to the nearest neighbors. The definition of the kNN consistency can also be applied to other neighborhood concepts, for example, k-Mutual-Nearest-Neighbors [10], which symmetrizes k-Nearest-Neighbors and is denoted by $k\mathrm{MN}(x)$:

$$x_j \in k\mathrm{MN}(x_i) \iff x_j \in k\mathrm{NN}(x_i) \wedge x_i \in k\mathrm{NN}(x_j).$$

Algorithm 1: ENFORCE

```
l ← cluster assignments by input clustering algorithm;
S ← calculate sets of closed neighborhoods;
N ←null;
foreach S_i ∈ S = {S_1, ..., S_h} do                    // closed neighborhoods
    (N) ←null;
    foreach x_j ∈ S_i do                                // count cluster assignments
        N_{l_j} + +;
    o ← arg max N;
    foreach x_j ∈ S_i do l_j ← o;                       // assign closed neighborhoods
return L;
```

Mutual nearest neighbors have the benefit of being a symmetric relation, but also being less sensitive to outliers and more likely to contain multiple components. Intuitively, we remove all unidirectional edges from the k-nearest-neighbor graph. Consider a data set where we have one dense cluster and a far away outlier. The outlier's neighbors will be points of the cluster, but it will not be in the nearest neighbors of the cluster points. Both the kNN and the kMN are often used in the context of spectral clustering, which in turn is related to DBSCAN [12, 13].

Ding and He [10] aim at enforcing kNN consistency in their algorithms. We can interpret this as a form of constrained clustering, where we add must-link constraints for neighbors. Such constraints have been previously used, e.g., for model selection [14] and for integrating external constraints into K-means [15, 16].

3. Clustering with Nearest Neighbor Consistency

The k-Nearest-Neighbor consistency was originally proposed by Ding and He [10] along with two algorithms to create k-Nearest-Neighbor-Consistent clustering, ENFORCE and consistency-preserving K-means (K-means-CP). The ENFORCE algorithm modifies an existing clustering, which can be created by any cluster analysis method, to improve its consistency by reassigning neighborhoods to the most common cluster. To achieve this, ENFORCE uses closed neighborhoods. A set $S \subset X$ is a closed neighborhood if for all $x \in S$ holds $k\text{NN}(x) \subset S$ and for every two elements $x_i, x_j \in S$ there is a path of elements from $k\text{NN}(x_i)$ or $k\text{NN}(x_j)$. The definition can be easily extended to any neighborhood function such as kMN. It is easy to see that all points in a closed neighborhood must be in the same cluster for the clustering to be consistent and that every point can only belong to one closed neighborhood. Figure 1 shows the closed neighbor sets on an example data set using mutual nearest neighbors with $k = 3$. To enforce a kNN-consistent clustering, ENFORCE iterates through all sets of closed neighborhoods and counts the respective cluster labels. Then it assigns the entire set to the label that occurs most often. There is also an interesting parallel to DBSCAN clustering here, which computes a similar transitive closure, but with a density-based notion of the neighborhood. DBSCAN clusters are closed neighborhoods with respect to density reachability.

In contrast to ENFORCE, consistency-preserving K-means (K-means-CP) does not work on an

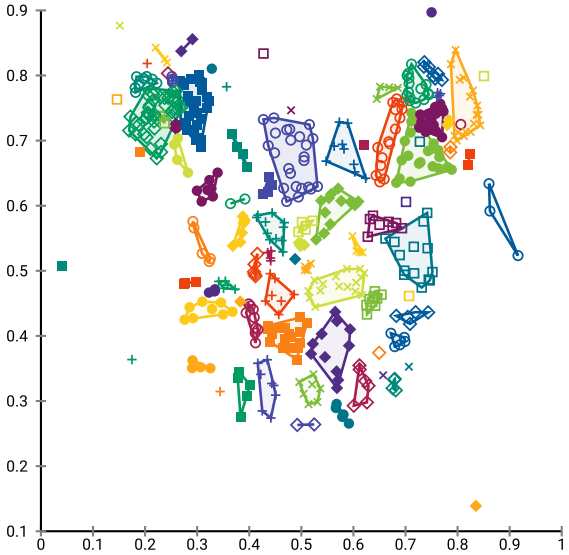


Figure 1: Closed neighbor sets for mutual nearest neighbors with $k = 3$.
For neighbor-consistent clustering, these sets must be assigned to the same cluster.

existing clustering, but integrates consistency into the standard K-means procedure. K-means uses the quadratic Euclidean distance, although there are other approaches, such as spherical K-means [17, 18], they will not be considered here. First, initial cluster centers $\mathrm{M} = \{\mu_1, ..., \mu_k\}$ are chosen with any of the standard heuristics. Then an alternating optimization as in the standard algorithm is performed. But in order to obtain a kNN-consistent result, K-means-CP, like ENFORCE, always assigns entire closed neighborhoods to the same cluster. Thus, Ding and He [10] define the nearest cluster center $\mathrm{nearest}(S_i)$ of a closed neighborhood S_i by the sum of squared euclidean distances

$$\mathrm{nearest}(S_i) = \arg\min_j \sum_{x \in S_i} \left\| x - \mu_j \right\|^2 .$$

All points in the closed neighborhood S_i are then assigned to this cluster. The assignment step is alternated with a recalculation of the cluster centers. A new cluster center μ_j of the cluster C_j is determined as usual in K-means using the arithmetic average of the assigned data

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x \,.$$

These steps are repeated until no point is reassigned (and hence the cluster centers do not change anymore). Convergence guarantees of the standard algorithm still apply. This algorithm computes exactly $N \cdot K$ distances in each step to determine the nearest clusters, and these distance computations make up a major part of the algorithm's run time. We can also optimize the recomputation of the cluster centers by using an incremental computation, but this has much less effect. In the following, we discuss why many of the distance computations performed above are unnecessary, and we can hence improve the run time of this algorithm.

Algorithm 2: K-means-CP

```
C ← initialize k cluster;
S ← calculate sets of closed neighborhoods;
l ← dummy assignments to non-existent cluster;
repeat
    foreach S_i ∈ S = {S_1, ..., S_h} do                    // closed neighborhoods
        N ←null;
        foreach μ_j ∈ M = {μ_1, ..., μ_k} do                // determine nearest cluster
            foreach x_u ∈ S_i do
                N_j ← N_j + ‖x_u − μ_j‖²;
        o ← arg min N;
        foreach x_j ∈ S_i do l_j ← o;                       // assign closed neighborhoods
    if l is unchanged then break;
    M ← calculate cluster centers for C;
return L;
```

4. Fast k-Nearest-Neighbor-Consistent Clustering

The K-means-CP algorithm determines the nearest cluster of a closed neighborhood set by the sum of the squared Euclidean distances of all elements of a closed neighborhood set to the different cluster centers. Using the parallel axis theorem of König, Huygens and Steiner, we can prove that instead of considering the distances of all samples in a closed neighborhood set S_i, it is sufficient to consider only the mean vector $\bar{s}_i = \frac{1}{|S_i|}\sum_{x\in S_i} x$. We can also trivially use the mean vector for updating the cluster centers, if we weight it by the number of points in the closed neighborhood set. The same property has been previously used by Lee et al. [19] to reduce uncertain UK-means clustering to regular K-means clustering. Our improved Neighbor-consistent K-Means (NCK-means) hence uses the mean vectors of the closed neighborhood sets. This yields a faster variant of consistency-preserving K-means with a significantly lower run time and the guaranteed same result.

We will reference the set which contains all closed neighborhood sets by S. Because all elements of a closed neighborhood set will be assigned to the same cluster, we will reference the set which contains all closed neighborhood sets whose elements have cluster label i by C_i.

4.1. Proof of correctness

To prove the equivalence of the results of NCK-means and K-means-CP, we show that the substeps of the algorithm always produce the same results. We first show that for given cluster centers, NCK-means and K-means-CP assign the same cluster labels to a closed neighborhood set. We start by considering how the original K-means-CP assigns cluster labels:

$$S_j \in C_i \iff i = \arg\min_i \sum_{x\in S_j} \|x-\mu_i\|^2 \ .$$

Algorithm 3: NCK-means

```
C ← initialize k cluster;
S̄ ← calculate representatives of closed neighborhood sets;
repeat
    foreach s̄_i ∈ S̄ = {s̄_1, ..., s̄_h} do   // representatives of closed neighborhoods
        N ←null;
        foreach μ_j ∈ M = {μ_1, ..., μ_k} do          // determine nearest cluster
            N_j ← N_j + ‖s̄_i − μ_j‖²;
        o ← arg min N;
        foreach x_j ∈ S_i do l_j ← o;                 // assign closed neighborhoods
    if l is unchanged then break;
    M ← calculate cluster centers for C;
return L;
```

NCK-means assigns its cluster labels by the equivalent optimization:

$$S_j \in C_i \iff i = \arg\min_i \left\|\bar{s}_j - \mu_i\right\|^2 .$$

This equivalence easily follows from the following version of the parallel axis theorem:

$$\sum_{x\in X} |x-a|^2 = \sum_{x\in X} \left(|x-\bar{X}|^2\right) + N|a-\bar{X}|^2$$

by using the vector form (a sum over all components) and then $a = \mu_i$ and $X = S_j$:

$$\sum_{x\in S_j} \|x-\mu_i\|^2 = \sum_{x\in S_j} \left(\|x-\bar{s}_j\|^2\right) + |S_j| \cdot \|\mu_i - \bar{s}_j\|^2 .$$

Because only the last term depends on i, we can omit the others for the minimization over i:

$$\arg\min_i \sum_{x\in S_j} \|x-\mu_i\|^2 = \arg\min_i \left\|\bar{s}_j - \mu_i\right\|^2 .$$

We can prove the above version of the parallel axis theorem as follows:

$$\begin{aligned}\sum_{x\in X} |x-a|^2 &= \sum_{x\in X} \left|(x-\bar{X}) - (a-\bar{X})\right|^2 = \sum_{x\in X} \left((x-\bar{X})^2 - 2(x-\bar{X})(a-\bar{X}) + (a-\bar{X})^2\right) \\ &= \sum_{x\in X} (x-\bar{X})^2 - 2(a-\bar{X}) \underbrace{\sum_{x\in X} (x-\bar{X})}_{=0} + N|a-\bar{X}|^2 = \sum_{x\in X} |x-\bar{X}|^2 + N|a-\bar{X}|^2\end{aligned}$$

The classic parallel axis theorem is obtained for $a = 0$.

We now consider the second step, updating the cluster centers. We start with the original calculation of the cluster centers μ_i. K-means-CP calculates this via the mean of the elements.

$$\mu_i = \frac{1}{|\{x_j \mid l_j = i\}|} \sum_{x_j \in X | l_j = i} x_j$$

Because the labels of each closed set are consistent, we can rewrite this to:

$$\mu_i = \frac{1}{\sum_{S_j \in C_i} |S_j|} \sum_{S_j \in C_i} \sum_{x_k \in S_j} x_k$$

and by using $\bar{s}_j = \frac{1}{|S_j|} \sum_{x_k \in S_j} x_k$, we obtain

$$\mu_i = \frac{1}{\sum_{S_j \in C_i} |S_j|} \sum_{S_j \in C_i} |S_j| \bar{s}_j \ .$$

This shows that for closed neighborhood sets with given cluster labels, NCK-means creates the same cluster centers as K-means-CP. Because both steps of the algorithms produce the same results, we have proven that the final result of the algorithms is the same.

4.2. Expected run time improvement

The run time improvement depends on the data reduction using closed neighbor sets. The standard K-means algorithm has a run time of $O(NKdi)$ where N is the number of points, K is the number of clusters, d is the dimensionality, and i is the number of iterations. Finding the closed neighbor sets additionally takes $O(N^2)$ time (although indexes for similarity search may yield a considerable speedup), and K-means-CP of Ding and He [10] hence has a complexity of $O(N^2 + Nkdi)$. NCK-means reduces this to $O(N^2 + |S|Kdi)$ where $|S|$ is the number of closed neighbor sets, and we must assume $|S| \in O(N)$. In a worst-case asymptotic analysis, the improvements hence are likely negligible because the cost of finding the closed neighbor sets dominates, but in practice it usually offers a decent run time improvement on the order of $|S|/N$ in the clustering step, at only the cost of little additional memory to store the means of each closed neighbor set. Hence, there is no reason not to use it. As it is a best practice to run K-means several times and keep the best outcome [20], the cost to find the closed neighbor sets can also be amortized over multiple restarts.

The choice of the neighborhood has an important effect on the run time. Finding the nearest neighbors becomes more expensive with a larger neighborhood size, but at the same time increasing the number of edges decreases the number of closed neighbor sets. For NCK-means, a lower number is beneficial, and both algorithms also benefit from an often lower number of iterations because reassignments are less likely to happen with the additional consistency constraints.

4.3. Further run-time improvements

Because our proof shows that it is sufficient to use the mean of each closed neighbor set, it is trivially possible to combine this with acceleration techniques of k-means such as the algorithms of Elkan [21], Hamerly [22], or the more recent Exponion [23] and Shallot [24] algorithms. The weight can also be used to build a BETULA tree [25]. We have not experimentally verified these additional speedups, as this would distract from the main objective of this paper. Furthermore, the differences between these algorithms will often be small compared to the cost of finding the nearest neighbors of all points to construct the closed neighborhoods in the beginning.

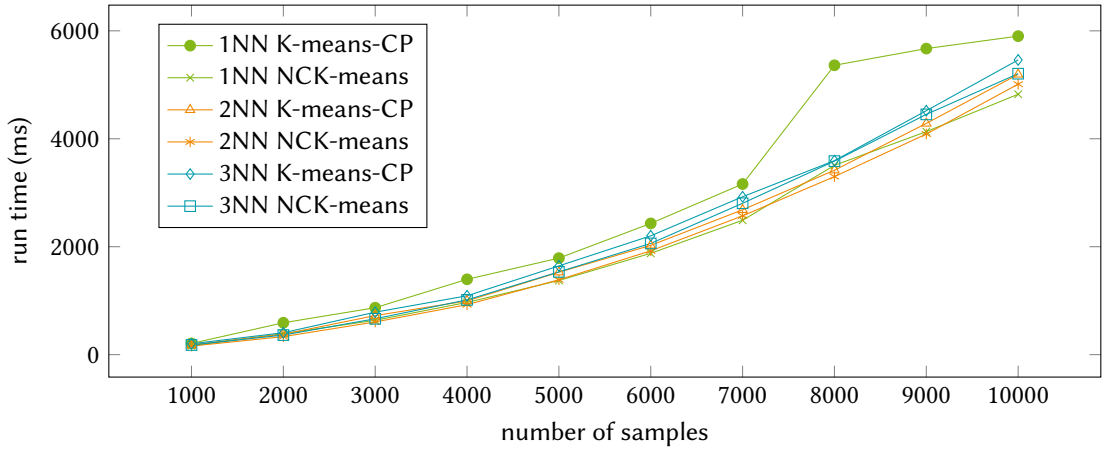
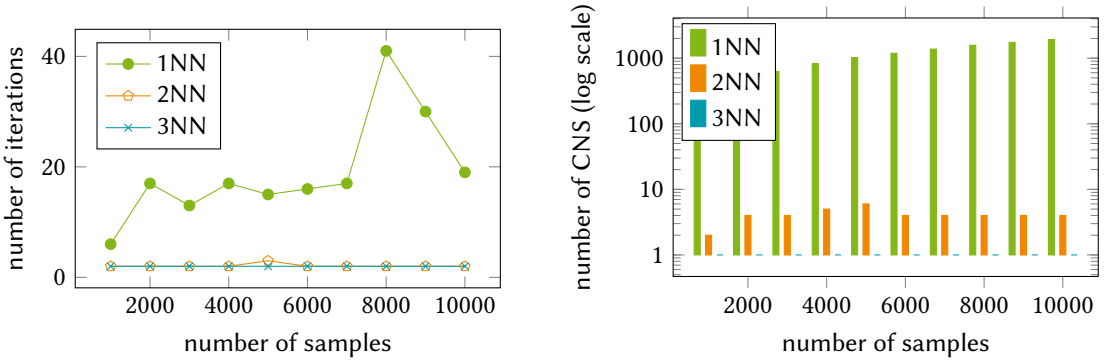


Figure 2: Run time of *K-means-CP* and *NCK-means* for different number of samples of MNIST, for *k-nearest neighbor* neighborhood with $k = 1, 2, 3$.



(a) Number of iterations required by the *K-means-CP* for different number of samples for *k-nearest neighbor* with $k = 1, 2, 3$.

(b) Number of *closed neighborhood sets* (CNS) for different number of samples for *k-nearest neighbor* neighborhood with $k = 1, 2, 3$.

Figure 3: Number of iterations and closed neighborhood sets on MNIST data

5. Experiments

To empirically verify the run time improvements, we implemented both versions with Java in the ELKI 0.8.0 data mining framework [26]. All implementations are within the same codebase to reduce confounding factors and improve comparability [27]. We perform 10 restarts on an Intel i4690K processor using a single thread, and evaluate the average values. We analyze the run time for the *MNIST 784* data set. *MNIST 784* contains grayscale pictures of handwritten digits with a picture size of 28×28 pixels, which is vectorized to a vector of length 784. We compared the algorithms with subsets of sizes $n = 1000, 2000, \ldots, 10000$ for the kNN and kMN neighborhood for $k = 1, 2, 3$ neighbors. The initial cluster centers were chosen with K-means++ [28]. The number of clusters is set to $K = 10$ because there are ten digits in the data set.

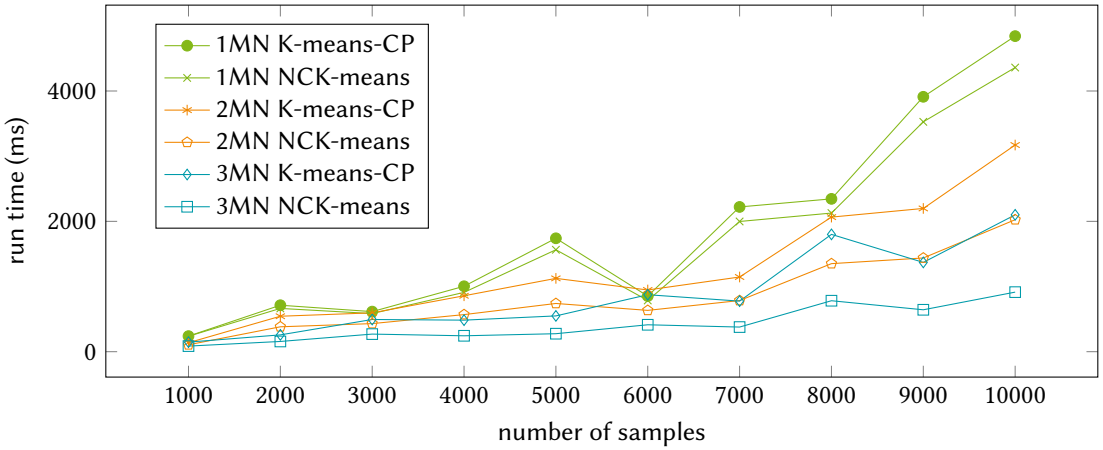
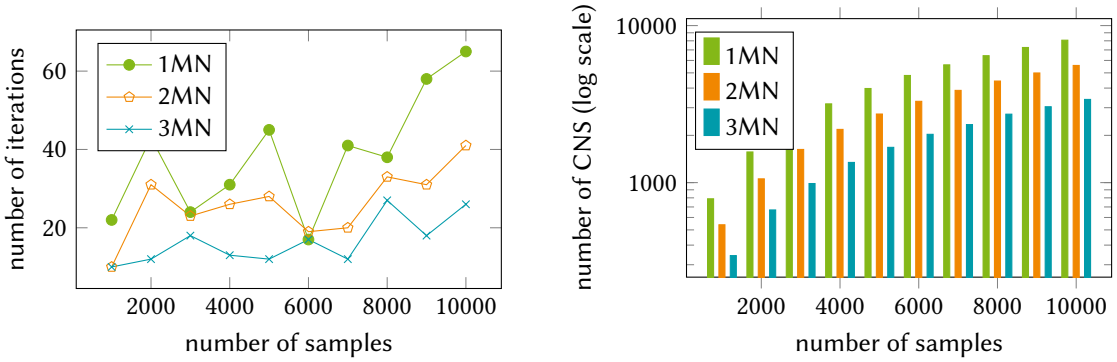


Figure 4: Run time of *K-means-CP* and *NCK-means* for different number of samples for *k-mutual-nearest neighbor* with $k = 1, 2, 3$.

The time measured is the time required for calculating the closed neighborhood sets and until the algorithm converges. The time to calculate the kNN of the data set is omitted because it is required for both algorithms and is the most time-consuming part. Therefore time variances in the calculation of the kNNs would overshadow the actual time difference of the algorithms. The ELKI framework automatically uses a k-d-tree or a vantage-point tree to accelerate nearest neighbor search [26], depending on data set characteristics and the distance function used. Because of the dimensionality, ELKI chose a VP-tree automatically for this data set. In Figure 6, we plot the run time of *1-mutual-nearest neighbor* computation and *NCK-means* for different number of samples of MNIST data. For MNIST with 784 dimensions, the kNN/kMN computation takes a large part compared to the clustering computation. For MNIST with 784 dimensions, the kNN/kMN computation takes a large share. We expect the kNN/kMN computation to be much more lightly weighted for lower dimensions.

In Figure 2, we plot the results for asymmetric kNN. While the run time improved in all cases, there are only noticeable differences for $k = 1$, which get larger at $n = 8000$. The little improvements for $k = 2, 3$ are caused by kNN inducing only very few closed neighborhood sets. For $k = 2$, the maximum amount of six closed neighborhood sets is reached with $n = 5000$. For $k = 3$ all data sets are covered by a single closed neighborhood set. This leads to algorithms only needing one or two iterations until convergence. The spike for $k = 1$ can also be explained by the iterations needed for the algorithms to converge, which reaches its clear maximum for $n = 8000$. In this case, the run time improves by over 34%.

In Figure 4, we plot the results for symmetric kMN. There are significant differences for all values of k used. This is because the amount of induced closed neighborhood sets seems to scale approximately linearly for kMN. The spikes in the run time differences can again be explained by the varying number of iterations required for convergence. The greatest relative improvement is reached by $k = 3$ for $n = 8000$, reducing the run time by over 56%. Across all k and n, we observe an average run time reduction of 30%.



(a) Number of iterations required by the *K-means-CP* for different number of samples for *k-mutual-nearest neighbor* with $k = 1, 2, 3$.

(b) Number of *closed neighborhood sets* for different number of samples for *k-mutual-nearest neighbor* with $k = 1, 2, 3$..

Figure 5: Number of iterations and closed neighborhood sets on MNIST data.

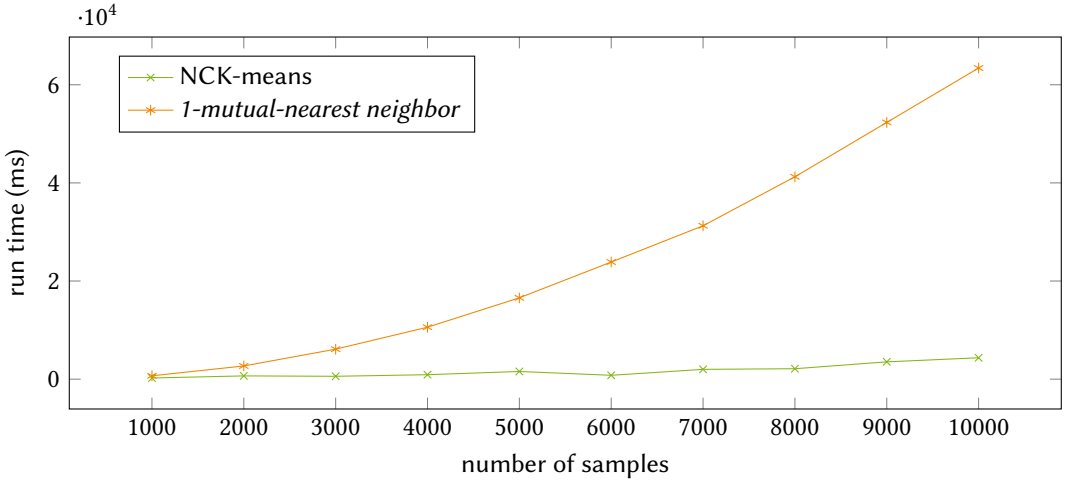


Figure 6: Run time of *1-mutual-nearest neighbor* computation and *NCK-means* for different number of samples of MNIST data.

The run time improvement per iteration would be the largest if there are very few closed neighborhood sets because, in this case we save a lot of time by using the representative. But if there are too few closed neighborhood sets, the algorithm converges very fast, e.g., in the case of 3NN, where the initial clustering is already converged. The clustering quality for very few closed neighborhood sets is also questionable but sometimes surprisingly good. This is probably because of the relationship to DBSCAN, which also builds closed neighborhoods, and spectral clustering, which partitions the nearest-neighbor graph.

6. Conclusion

We improve the K-means-CP algorithm for clustering with k-Nearest-Neighbor consistency by avoiding unnecessary distance computations. Using the parallel axis theorem, we prove that we can reduce all closed neighborhoods to a single representative, and use these in clustering instead of considering all samples individually. This allows clustering with nearest-neighbor consistency on larger data sets than before.

The technique could be integrated in further algorithms. In COP-K-Means [15] and PCK-means [16], for example, this allows us to reduce must-link constraints into using weighted means, too. But the speedup is only noticeable if we have many such constraints, and the typical scenario for constrained clustering is with only a few constraints given by the user to guide the clustering in a semi-supervised way.

References

[1] R. E. Bonner, On some clustering techniques, IBM Journal of Research and Development 8 (1964) 22–32. doi:10.1147/rd.81.0022.

[2] V. Estivill-Castro, Why so many clustering algorithms – a position paper, SIGKDD Explorations 4 (2002) 65–75. doi:10.1145/568574.568575.

[3] M. Ackerman, S. Ben-David, Measures of clustering quality: A working set of axioms for clustering, in: NIPS, 2008, pp. 121–128.

[4] P. J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (1987) 53–65.

[5] D. L. Davies, D. W. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. (1979) 224–227. doi:10.1109/TPAMI.1979.4766909.

[6] T. Calinski, J. Harabasz, A dendrite method for cluster analysis, Communications in Statistics 3 (1974) 1–27. doi:10.1080/03610927408827101.

[7] J. C. Dunn, Well-separated clusters and optimal fuzzy partitions, Journal of Cybernetics 4 (1974) 95–104. doi:10.1080/01969727408546059.

[8] E. Schubert, P. J. Rousseeuw, Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms, Information Systems 101 (2021) 101804. doi:10.1016/j.is.2021.101804.

[9] L. Lenssen, E. Schubert, Clustering by direct optimization of the medoid silhouette, in: Similarity Search and Applications, 2022, pp. 190–204. doi:10.1007/978-3-031-17849-8_15.

[10] C. Ding, X. He, K-nearest-neighbor consistency in data clustering: Incorporating local information into global optimization, in: Symp. Applied Computing, SAC, 2004, p. 584–589. doi:10.1145/967900.968021.

[11] J. Handl, J. Knowles, An evolutionary approach to multiobjective clustering, IEEE Trans. Evol. Comput. 11 (2007) 56–76. doi:10.1109/TEVC.2006.877146.

[12] E. Schubert, S. Hess, K. Morik, The relationship of DBSCAN to matrix factorization and spectral clustering, in: Lernen, Wissen, Daten, Analysen, 2018, pp. 330–334.

[13] A. Beer, A. Draganov, E. Hohma, P. Jahn, C. M. Frey, I. Assent, Connecting the dots

– density-connectivity distance unifies DBSCAN, k-center and spectral clustering, in: Knowledge Discovery and Data Mining (KDD), 2023. doi:10.1145/3580305.3599283, to appear.
[14] M. Pourrajabi, D. Moulavi, R. J. G. B. Campello, A. Zimek, J. Sander, R. Goebel, Model selection for semi-supervised clustering, in: Extending Database Technology, EDBT, 2014, pp. 331–342. doi:10.5441/002/edbt.2014.31.
[15] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, Constrained k-means clustering with background knowledge, in: Int. Conf. Machine Learning (ICML), 2001, pp. 577–584.
[16] S. Basu, A. Banerjee, R. J. Mooney, Active semi-supervision for pairwise constrained clustering, in: SIAM Data Mining (SDM), 2004, pp. 333–344. doi:10.1137/1.9781611972740.31.
[17] I. S. Dhillon, D. S. Modha, Concept decompositions for large sparse text data using clustering, Mach. Learn. 42 (2001) 143–175. doi:10.1023/A:1007612920971.
[18] E. Schubert, A. Lang, G. Feher, Accelerating spherical k-means, in: Similarity Search and Applications (SISAP), 2021, pp. 217–231. doi:10.1007/978-3-030-89657-7_17.
[19] S. D. Lee, B. Kao, R. Cheng, Reducing UK-means to K-means, in: Workshops Int. Conf. Data Mining (ICDM), 2007, pp. 483–488. doi:10.1109/ICDMW.2007.40.
[20] E. Schubert, Stop using the elbow criterion for k-means and how to choose the number of clusters instead, SIGKDD Explorations 25 (2023) 36–42. doi:10.1145/3606274.3606278.
[21] C. Elkan, Using the triangle inequality to accelerate k-means, in: Int. Conf. Machine Learning (ICML), 2003, pp. 147–153.
[22] G. Hamerly, Making k-means even faster, in: SIAM Data Mining (SDM), 2010, pp. 130–140. doi:10.1137/1.9781611972801.12.
[23] J. Newling, F. Fleuret, Fast k-means with accurate bounds, in: Int. Conf. Machine Learning (ICML), 2016, pp. 936–944.
[24] C. Borgelt, Even faster exact k-means clustering, in: Symp. Intelligent Data Analysis (IDA), 2020, pp. 93–105. doi:10.1007/978-3-030-44584-3_8.
[25] A. Lang, E. Schubert, BETULA: fast clustering of large data with improved BIRCH CF-trees, Inf. Syst. 108 (2022) 101918. doi:10.1016/j.is.2021.101918.
[26] E. Schubert, Automatic indexing for similarity search in ELKI, in: Similarity Search and Applications (SISAP), 2022. doi:10.1007/978-3-031-17849-8_16.
[27] H. Kriegel, E. Schubert, A. Zimek, The (black) art of runtime evaluation: Are we comparing algorithms or implementations?, Knowl. Inf. Syst. 52 (2017) 341–378. doi:10.1007/s10115-016-1004-2.
[28] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: Symp. Discrete Algorithms, SODA, 2007, pp. 1027–1035.