Towards an Optimal Subspace for K-Means

Dominik Mautz[†], Wei Ye[†], Claudia Plant[§], Christian Böhm[†]
[†]Ludwig-Maximilians-Universität München, Munich, Germany
{mautz,ye,boehm}@dbs.ifi.lmu.de
[§]University of Vienna, Vienna, Austria
claudia.plant@univie.ac.at

ABSTRACT

Is there an optimal dimensionality reduction for *k*-means, revealing the prominent cluster structure hidden in the data? We propose SubKmeans, which extends the classic *k*-means algorithm. The goal of this algorithm is twofold: find a sufficient *k*-means-style clustering partition and transform the clusters onto a common subspace, which is optimal for the cluster structure. Our solution is able to pursue these two goals simultaneously. The dimensionality of this subspace is found automatically and therefore the algorithm comes without the burden of additional parameters. At the same time this subspace helps to mitigate the *curse of dimensionality*. The SubKmeans optimization algorithm is intriguingly simple and efficient. It is easy to implement and can readily be adopted to the current situation. Furthermore, it is compatible to many existing extensions and improvements of *k*-means.

KEYWORDS

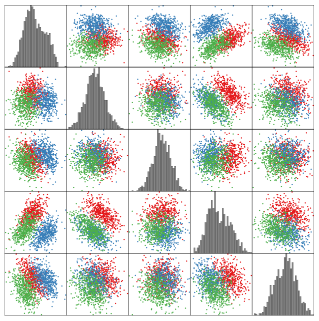
clustering; k-means; subspace; dimensionality reduction

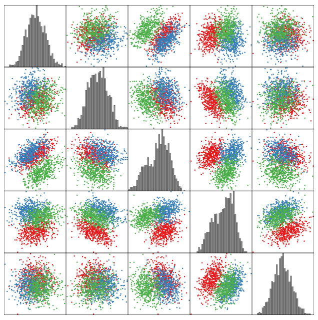
1 INTRODUCTION

The *k*-means clustering algorithm is one of, if not the, most often used partition based clustering algorithm. Our own community voted it second of the top 10 data mining algorithms [31]. Its popularity is largely a result of several intriguing properties: it is simple, it can readily be adopted to different situations, it is fast and can scale to very large datasets. Last but not least, its results are mostly sufficient for a first analysis.

However, the interpretation of what the algorithm finds becomes increasingly difficult with growing number of dimensions. Even in lower dimensional spaces it is sometimes hard to tell what structure the algorithm has found. In the following we propose SubKmeans, a technique, which extends *k*-means. It finds a clustering partition and simultaneously a transformation which highlights the structure found in the dataset.

Let us consider a small, five-dimensional synthetic dataset containing three clusters, as shown in Figure 1 (a). Let us assume for


(a) Synthetic Data


(b) PCA

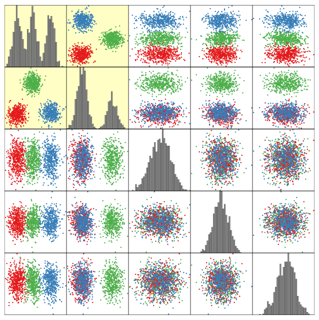

(c) SubKmeans

Figure 1: Diagram (a) shows the scatter plots of artificial data set with color-coded ground truth. Diagram (b) shows the PCA transformation with ground truth. Diagram (c) shows the data transformed by SubKmeans. This method finds the correct cluster partition and automatically identifies the first two features in the transformed space—highlighted in yellow—as important for the cluster structure.

a moment that the ground truth would be unknown and the scatter plot monochrome. A data scientist inspecting these plots may assume that the data contains some structure as it shines through in some scatter plots. Yet, a clear structure does not present itself. Driven by this idea the data scientist now may try out a Principal Component Analysis (PCA) transformation, which reveals the directions of most variance within this dataset. Figure 1 (b) shows the result of this action. However, the result is somewhat disappointing. The new perspective onto the data does not reveal a clear structure. The next step may be to apply a simple clustering algorithm like *k*-means, to shed some light into the data. Let us further assume that this algorithm also finds the color-coded ground truth—which *k*-means indeed does for this simple dataset.

KDD '17, August 13-17, 2017, Halifax, NS, Canada
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3097989

At this point, our scientist may have a second look on both diagrams. But still, clear cluster boundaries are hardly visible and a clear cluster structure does not present itself. It is hard to tell what k-means actually found and if it is meaningful.

As a consequence, our researcher might turn to other, more sophisticated methods like Independent Component Analysis (ICA) or variants of non-linear manifold learning. These other techniques may even be able to find transformations in such a dataset, revealing more of the hidden structures. But, to the best of our knowledge, none of these techniques tries to find an optimal dimensionality reduction for k-means clustering. Therefore, in general, our data scientist will have a hard time identifying the feature combinations, which are most relevant for the overall found cluster structure.

The situation is completely different for the scatter plots shown in Figure 1 (c). The data shown in the figure was partitioned and transformed by our proposed algorithm SubKmeans. We can readily identify three clusters supported by the first two features. Further, we can see that only those two features provide relevant information for the overall cluster structure. The other three features only present unimodal structures and do not provide any crucial information. This algorithm takes—in contrast to a combination of PCA and k-means—the local structure into account and is not only able to partition the data, but also to simultaneously transform it such that its inner structure is revealed to the data scientist. Further, it identifies which of these newly revealed features are important to the clustering by splitting the transformed data into two subspaces. A *clustered* space, which contains all useful structure, and a *noise* space, which contains the irrelevant (unimodal) structures. Performing the partitioning in the *clustered* space has the additional advantage that it mitigates the problem of the meaningfulness of proximity in high-dimensional spaces.

Our **main contributions** are as follows:

(1) We extend k-means such that it incorporates a simultaneous dimensionality reduction step.
(2) The dimensionality of the *clustered* space is determined automatically.
(3) We do not introduce an additional parameter. The only parameter needed is the number of clusters k.
(4) We propose an optimization algorithm, SubKmeans, which is very easy to implement and can readily be adopted to different situations. It is fast, even without sophisticated performance optimizations.
(5) SubKmeans is compatible to many proposed extensions of k-means.

2 SUBSPACE *K*-MEANS

In the following we describe our proposed method SubKmeans.[1] We describe our algorithm, for the sake of brevity, as a simple extension of the well-known Lloyd's algorithm [21] with its EM-style alternating assignment and update steps. Though, it is possible to incorporate many other k-means extensions and modifications for performance optimization. A non-exhaustive list of these can be found in Section 4. Table 1 shows the used symbols and definitions.

[1]Implementation and supplementary available under: http://dmm.dbs.ifi.lmu.de/downloads/

Table 1: Symbols and Definitions

Symbol	Interpretation
$d \in \mathbb{N}$	Dimensionality of original space
$m \in \mathbb{N}$	Dimensionality of the *clustered* space
$k \in \mathbb{N}$	Number of Clusters
$\mathcal{D}$	Set of all objects
C_i	Set of objects assigned to cluster i
$\mathbf{x} \in \mathbb{R}^d$	A data point or object
$\boldsymbol{\mu}_{\mathcal{D}} \in \mathbb{R}^d$	Dataset mean in the original space
$\boldsymbol{\mu}_i \in \mathbb{R}^m$	Mean of cluster i in the original space
$S_{\mathcal{D}} \in \mathbb{R}^{d \times d}$	Scatter matrix of the dataset in the original space
$S_i \in \mathbb{R}^{d \times d}$	Scatter matrix of cluster i in the original space
$\Sigma \in \mathbb{R}^{d \times d}$	See definition in Formula 3
$P_{\mathrm{C}} \in \mathbb{R}^{m \times d}$	Projection onto the first m attributes
$P_{\mathrm{N}} \in \mathbb{R}^{d-m \times d}$	Projection onto the last $d - m$ attributes
$V \in \mathbb{R}^{d \times d}$	(orthonormal) matrix of a rigid transformation
$\mathbf{I}_l$	$l \times l$ identity matrix
$\mathbf{0}_{l,r}$	$l \times r$ zero matrix

2.1 Cost Function

In the classic version of the k-means algorithm, we want to find a set of k clusters C_i, which partitions the data such that the sum of squared errors is minimized,

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 ,$$

where $\| \cdot \|$ represents the Euclidean norm.

We extend this basic idea in SubKmeans by the assumption that the cluster structure is contained in a lower dimensional subspace. That is, we assume that the data space can be split into two arbitrarily oriented subspaces. The first m-dimensional, subspace contains all structural information of the k clusters. We call this subspace the *clustered* space.

The orthogonal complement of this subspace builds a second, $(d - m)$-dimensional, subspace. It does not contain any structural information important for the clustering. The expectation within this subspace is that the data point values for each feature are drawn from the same unimodal symmetric distribution (See also Section 2.5). We call this the *noise* space, since its feature values do not help to distinguish between the different clusters in the *clustered* space. As we are not interested in the actual distribution of the *noise*, we assume that this space can be described by a single cluster.

Further, we assume that we can find an orthonormal (rigid) transformation matrix V, which rotates (and reflects) the original space in such a way that the first m features in the transformed space correspond to the *clustered* space and the last $(d - m)$ features correspond to the *noise* space. We can use two simple projections to split these features into their respective subspaces:

$$P_{\mathrm{C}} = \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0}_{d-m,m} \end{bmatrix}, \quad P_{\mathrm{N}} = \begin{bmatrix} \mathbf{0}_{m,d-m} \\ \mathbf{I}_{d-m} \end{bmatrix}.$$

A data point $\mathbf{x}$ can then be projected onto the *clustered* space by $P_{\mathrm{C}}^{\mathsf{T}} V^{\mathsf{T}} \mathbf{x}$. The projection onto the *noise* space is performed similarly using P_{N}.

Algorithm 1: SubKmeans

```
 1 Input: dataset $\mathcal{D}$; number of clusters $k$
 2 Output: clusters $\{C_1, \dots, C_k\}$;
 3     rotation matrix $V$;
 4     dimensionality of clustered space $m$
   // Initialization:
 5     $V \leftarrow$ random orthonormal matrix
 6     $m \leftarrow$ some initial value, e.g. $\frac{d}{2}$ or $\sqrt{d}$
 7     $\boldsymbol{\mu}_\mathcal{D} \leftarrow \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}\in\mathcal{D}} \mathbf{x}$
 8     $S_\mathcal{D} \leftarrow \sum_{\mathbf{x}\in\mathcal{D}} (\mathbf{x} - \boldsymbol{\mu}_\mathcal{D}) (\mathbf{x} - \boldsymbol{\mu}_\mathcal{D})^\mathsf{T}$
 9     $\forall i \in [1, k] : \boldsymbol{\mu}_i \leftarrow$ random data point of $\mathcal{D}$
10 repeat
       // Assignment step
11     $C_j \leftarrow \emptyset$
12     $\forall \mathbf{x} \in \mathcal{D}$:
13         $j \leftarrow \underset{i\in[1,k]}{\arg\min} \left\| P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \mathbf{x} - P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \boldsymbol{\mu}_i \right\|^2$
14         $C_j \leftarrow C_j \cup \{\mathbf{x}\}$
       // Update step
15     $\forall i \in [1, k]$:
16         $\boldsymbol{\mu}_i \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}\in C_i} \mathbf{x}$
17         $S_i \leftarrow \sum_{\mathbf{x}\in C_i} (\mathbf{x} - \boldsymbol{\mu}_i) (\mathbf{x} - \boldsymbol{\mu}_i)^\mathsf{T}$
       // Eigendecomposition:
       // $\mathcal{E}$: list of eigenvalues in ascending order
       // $V$: corresponding eigenvectors
18     $V, \mathcal{E} \leftarrow \mathrm{eig}\left(\left(\sum_{i=1}^{k} S_i\right) - S_\mathcal{D}\right)$
19     $m \leftarrow |\{e | e \in \mathcal{E} \wedge e < 0\}|$
20 until convergence;
```

Combining these assumption we get the following cost function, which we want to minimize:

$$\mathcal{J} = \left[\sum_{i=1}^{k} \sum_{\mathbf{x}\in C_i} \left\| P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \mathbf{x} - P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \boldsymbol{\mu}_i \right\|^2 \right] + \sum_{\mathbf{x}\in\mathcal{D}} \left\| P_\mathrm{N}^\mathsf{T} V^\mathsf{T} \mathbf{x} - P_\mathrm{N}^\mathsf{T} V^\mathsf{T} \boldsymbol{\mu}_\mathcal{D} \right\|^2 . \quad (1)$$

This cost function builds a trade-off between the two subspaces. Our expectation is that this trade-off 'squeezes' all structural information out of the *noise* space into the *clustered* space. Combinations of features of the original space, which contain information regarding the clusters, are better represented by the first term. Yet, combinations of features that do not contain the structural information are better represented by the second term. Thus by optimizing this objective function, we can find the optimal subspace for *k*-means.

2.2 Optimization Algorithm

We can optimize this objective function with a modified version of Lloyd algorithm. The full algorithm is shown in Algorithm 1. Like the classic version, it is only able to find local minima and therefore has to be run multiple times—with different initializations—for a sufficient clustering solution.

As a first step we have to initialize V with an random orthonormal matrix. For instance, we could perform a QR decomposition of a random matrix and use the resulting orthonormal matrix Q as the initial value. Further, we need the initial dimensionality of the *clustered* space m. It should be noted that this value may not be chosen too big or too small, as it balances the probability that the initial *clustered* space contains parts of the cluster structure we want to find and the expressiveness of the similarity measure. For simplicity, we use in our implementation $d/2$ as it depends on d. Yet, it is also possible to let the user choose an initial value. The optimal value for m is subsequently found during the optimization. Last but not least, the initial cluster centers could for example be picked randomly from the dataset or set by *k*-means++.

The **assignment step** is nearly equivalent to the classic version. We keep the parameters of V, m, and $\boldsymbol{\mu}_i$ fixed and assign each data point to the cluster for which the squared distance to the mean in the *clustered* space is minimized,

$$\underset{i}{\arg\min} \left\| P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \mathbf{x} - P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \boldsymbol{\mu}_i \right\|^2 .$$

This minimizes the cost function with respect to the fixed parameters.

In the **update step** we keep the data point assignment fixed and determine the parameter values, for which the cost function is minimized. The following sections discuss in detail the steps taken and the correctness of each step.

2.2.1 *Estimation of the cluster centers $\boldsymbol{\mu}_i$.*

We can determine the estimator for the cluster centers by setting the partial derivative with respect to $\boldsymbol{\mu}_i$ equal to zero:

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} \mathcal{J} = \frac{\partial}{\partial \boldsymbol{\mu}_i} \sum_{\mathbf{x}\in C_i} \left\| P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \mathbf{x} - P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \boldsymbol{\mu}_i \right\|^2 \overset{!}{=} 0$$

We shorten this simple prove at this point for brevity. The result of this calculation shows what one would intuitively expect: the cluster mean $P_\mathrm{C}^\mathsf{T} V^\mathsf{T} \boldsymbol{\mu}_i$ in the *clustered* space is simply the mean value of all data points assigned to this cluster (in the original space) and can be calculated with the well-known formula:

$$\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{x}\in C_i} \mathbf{x}.$$

We would like to emphasize that this formula is independent of V and m.

2.2.2 *Estimation of the transformation matrix V.*

Next we need to estimate the value of V. For now, we assume that m is a fixed but arbitrary integer value in the range $[0 : d]$. Instead of taking the endeavor using a partial derivative, we prove that with some tricks we can rewrite the cost function such that it yields an eigen decomposition problem. Solving this eigenvalue problem also minimizes the objective function:

$$\begin{aligned}\mathcal{J} &= \left[\sum_{i=1}^{k}\sum_{x\in C_i}\left\|P_C^\top V^\top \mathbf{x} - P_C^\top V^\top \boldsymbol{\mu}_i\right\|^2\right] \\ &\quad + \sum_{\mathbf{x}\in\mathcal{D}}\left\|P_N^\top V^\top \mathbf{x} - P_N^\top V^\top \boldsymbol{\mu}_{\mathcal{D}}\right\|^2 \\ &= \left[\sum_{i=1}^{k}\sum_{\mathbf{x}\in C_i}\left(P_C^\top V^\top \mathbf{x} - P_C^\top V^\top \boldsymbol{\mu}_i\right)^\top\left(P_C^\top V^\top \mathbf{x} - P_C^\top V^\top \boldsymbol{\mu}_i\right)\right] \\ &\quad + \sum_{\mathbf{x}\in\mathcal{D}}\left(P_N^\top V^\top \mathbf{x} - P_N^\top V^\top \boldsymbol{\mu}_{\mathcal{D}}\right)^\top\left(P_N^\top V^\top \mathbf{x} - P_N^\top V^\top \boldsymbol{\mu}_{\mathcal{D}}\right) \\ &= \left[\sum_{i=1}^{k}\sum_{\mathbf{x}\in C_i}\left(\mathbf{x}-\boldsymbol{\mu}_i\right)^\top V P_C P_C^\top V^\top\left(\mathbf{x}-\boldsymbol{\mu}_i\right)\right] \\ &\quad + \sum_{\mathbf{x}\in\mathcal{D}}\left(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{D}}\right)^\top V P_N P_N^\top V^\top\left(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{D}}\right) \\ &= \sum_{i=1}^{k}\sum_{\mathbf{x}\in C_i}\operatorname{Tr}\left(\left(\mathbf{x}-\boldsymbol{\mu}_i\right)^\top V P_C P_C^\top V^\top\left(\mathbf{x}-\boldsymbol{\mu}_i\right)\right) \\ &\quad + \sum_{\mathbf{x}\in\mathcal{D}}\operatorname{Tr}\left(\left(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{D}}\right)^\top V P_N P_N^\top V^\top\left(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{D}}\right)\right), \end{aligned} \tag{2}$$

where we apply in the last part of the equation the trace-trick. It exploits the fact that a scalar is also a 1×1 matrix. We can utilize the addition and cyclic permutation properties of trace to further rewrite the function:

$$\begin{aligned}&= \operatorname{Tr}\left(P_C P_C^\top V^\top\left[\underbrace{\sum_{i=1}^{k}\sum_{\mathbf{x}\in C_i}\left(\mathbf{x}-\boldsymbol{\mu}_i\right)\left(\mathbf{x}-\boldsymbol{\mu}_i\right)^\top}_{=:S_i}\right]V\right) \\ &\quad + \operatorname{Tr}\left(P_N P_N^\top V^\top\left[\underbrace{\sum_{\mathbf{x}\in\mathcal{D}}\left(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{D}}\right)\left(\mathbf{x}-\boldsymbol{\mu}_{\mathcal{D}}\right)^\top}_{=:S_{\mathcal{D}}}\right]V\right) \\ &= \operatorname{Tr}\left(P_C P_C^\top V^\top\left[\sum_{i=1}^{k} S_i\right]V\right) + \operatorname{Tr}\left(P_N P_N^\top V^\top S_{\mathcal{D}} V\right)\end{aligned}$$

Next we have to apply a second trick. First we should note that $P_C P_C^\top$ is a diagonal matrix where the first m entries are ones and the rest are zeros:

$$P_C P_C^\top = \operatorname{diag}(\underbrace{1,\cdots 1}_{m-\text{times}}, \underbrace{0,\cdots 0}_{(d-m)-\text{times}})$$

and equivalently for the noise projection:

$$P_N P_N^\top = \operatorname{diag}(\underbrace{0,\cdots 0}_{m-\text{times}}, \underbrace{1,\cdots 1}_{(d-m)-\text{times}})$$

Since the trace of a matrix is the sum of its diagonal elements, we can use it to bring the two diagonal matrices $P_C P_C^\top$ and $P_N P_N^\top$ into a relationship:

$$\operatorname{Tr}(P_N P_N^\top A) = \operatorname{Tr}(A) - \operatorname{Tr}(P_C P_C^\top A),$$

where A is an arbitrary $d \times d$ matrix.

Further, we note that the trace of a matrix is also the sum of its eigenvalues and V is defined as an orthonormal matrix. Therefore, the transformation of the scatter matrix, $V^\top S_{\mathcal{D}} V$, only changes the direction of the eigenvectors, but does not change the eigenvalues itself. This means, that the trace of such a transformation, $\operatorname{Tr}(V^\top S_{\mathcal{D}} V)$, has to be constant for all possible values of the orthonormal matrix V, and we can ignore this term for the estimation of V. We can use these properties to further rewrite the cost function:

$$\begin{aligned}&\operatorname{Tr}\left(P_C P_C^\top V^\top\left[\sum_{i=1}^{k} S_i\right]V\right) + \operatorname{Tr}\left(P_N P_N^\top V^\top S_{\mathcal{D}} V\right) \\ &= \operatorname{Tr}\left(P_C P_C^\top V^\top\left[\sum_{i=1}^{k} S_i\right]V\right) - \operatorname{Tr}\left(P_C P_C^\top V^\top S_{\mathcal{D}} V\right) + \operatorname{Tr}\left(V^\top S_{\mathcal{D}} V\right) \\ &= \operatorname{Tr}\left(P_C P_C^\top V^\top \underbrace{\left(\left[\sum_{i=1}^{k} S_i\right] - S_{\mathcal{D}}\right)}_{=:\Sigma} V\right) + \underbrace{\operatorname{Tr}\left(V^\top S_{\mathcal{D}} V\right)}_{\text{const. w.r.t. } V}.\end{aligned} \tag{3}$$

Since $P_C P_C^\top$ leaves the upper left $m \times m$ matrix untouched and sets all other values to zero, it is possible to minimize the function (1), for a fixed partition, fixed $\boldsymbol{\mu}_i$ and arbitrary but fixed m, by putting the eigenvectors of Σ into V's columns such that the m-eigenvectors corresponding to the m-smallest eigenvalues project the data into the *clustered* space and the other $(d - m)$ eigenvectors project the data onto the *noise* space. We therefore perform an eigenvalue decomposition of Σ and use the eigenvectors—sorted ascending to their corresponding eigenvalue—as columns in V. We should note that Σ is symmetric and therefore it is orthogonal diagonalizable and all its eigenvalues are real. Of course this only holds under the assumption that Σ has full rank, much like we assume for a Gaussian mixture model that its covariance matrices have full rank.

2.2.3 *Estimation of the dimensionality m of the* clustered *space.*

Since we sort the eigenvalues in V in ascending order and the constant term in Formula 3 does not depend on m, the estimation of V is independent of the actual m (As long as we sort V ascending to the corresponding eigenvalues). This gives us the ability to also optimize m within each update step.

We can use the rewritten cost function in Formula 3 also to minimize the costs with respect to m. We note that the equation depends on m only through P_C. Since the trace is the sum of all eigenvalues and we want to minimize this sum, we can only minimize it, if we sum up all negative eigenvalues (which Σ may has because in general it is indefinite). This means, we let all eigenvectors corresponding to a negative eigenvalue project onto the *clustered* space. If the eigenvalues are all positive, we can minimize the trace only by setting m equal to zero. That is we do not have a *clustered* space because we cannot find any structure in the data following the current partition and the dataset is better explained by a single cluster. Eigenvectors with corresponding eigenvalues greater than zero have to project onto the *noise* space. If the eigenvalue is zero we are indifferent with respect to the cost function. Yet, from a clustering perspective we prefer to have a smaller *clustered* space and therefore it makes sense to let those eigenvectors also project onto the *noise* space. Consequently, we can optimize the cost function for a given V by setting m to the number of negative eigenvalues of Σ. Though, in practice, we might also want to assign eigenvectors with a negative eigenvalue close to zero (e.g. $\geq -1^{-10}$) to the *noise* space for the same reasons as components with an eigenvalue equal to zero.

We would like to note that the eigenvalues of Σ also give rise to a notion of how important the corresponding eigenvector is for the clustering structure: the smaller it is, the better the cluster structure is represented by it. Sorting the eigenvectors in V ascending by their eigenvalue therefore means that we also sort them by importance with respect to the cluster structure.

2.3 Convergence

The convergence of our modified algorithm is quite simple to see. The cost function is decreasing in each update and assignment step and it is bounded from below. Thus, the algorithm has to converge towards a (local) minimum.

2.4 Complexity

The complexity of our algorithm depends on the complexity of the classic Lloyd algorithm $O(Idk|\mathcal{D}|)$, where I is the number of iterations. Additionally, it depends on the complexity of eigenvalue decomposition $O(d^3)$ and the calculation of the scatter matrices $O(d^2|\mathcal{D}|)$ for each iteration of the loop. Combining these yields a total complexity of $O(I(mk|\mathcal{D}| + d^2|\mathcal{D}| + d^3))$ and is therefore comparable to most classical subspace clustering algorithms with respect to the cubic runtime in the dimensionality.

Yet, the cubic runtime of the eigen decomposition is nearly a non-issue with modern technology. Currrent GPUs provide high-level APIs for such tasks and distributed computing frameworks allow the decomposition of matrices with millions of entries in mere seconds.[2]

2.5 Relationship to Mixture Models

A further justification for our cost function can be found by exploring the relationships to Gaussian mixture models. It is quite easy to show that the standard k-means cost function is a limit to a Gaussian mixture model with the probability distribution

$$p(\mathbf{x}) = \Pi_i^k \pi_i \mathcal{N}(\mathbf{x}|\mu_i, \sigma\mathbf{I}),$$

where one lets the overall variance σ go to zero [20].

Following this approach, it can be shown that SubKmeans is also a limit to the mixture model with probability distribution:

$$p(\mathbf{x}) = \left[\sum_i^k \pi_i \mathcal{N}(P_C^\top V^\top \mathbf{x}|P_C^\top V^\top \boldsymbol{\mu}_i, \sigma\mathbf{I})\right] \mathcal{N}(P_N^\top V^\top \mathbf{x}|P_N^\top V^\top \boldsymbol{\mu}_{\mathcal{D}}, \sigma\mathbf{I})$$

This can easily be seen by taking the limit of the log-likelihood function with respect to $\sigma \leftarrow 0$, which yields the cost function shown in Eq. 1 (with additional constant terms). We would like to point out that the *clustered* and *noise* space in this mixture model are statistically independent.

3 EXPERIMENTS

We evaluated our method both with respect to its clustering results, as well as its dimensionality reduction properties based on eight real-world datasets of varying size and dimensionality. We got those datasets from the UCI[3] and UCR[4] repositories.

We compare SubKmeans to k-means in combination with two widely used preprocessing methods, which achieve a lower dimensional representation of the data, namely Principal Component Analysis (PCA)[19] and Independent Component Analysis (ICA) [17]. For PCA we used the widely used rule of thumb and kept components such that 90% of the total variance was retained. For ICA we used the method FastICA (with log cosh). As a sensible choice for the number of components in ICA we used k, since one might expect for each cluster that it should at least differ from the other clusters in one independent component. After this preprocessing we applied the classic k-means.

Further, we compare our method to four other proposed algorithms that perform some kind of dimensionality reduction during clustering, namely LDA-k-means, FOSSCLU, ORCLUS and 4C.

For ORCLUS and 4C we used the implementations of the ELKI[5] framework, for FOSSCLU we used the provided Java implementation. The LDA-k-means algorithm is a faithful translation of the repective Matlab version. The k-means and SubKmeans were implemented by the authors in Scala and Breeze. Thereby, we used simple implementations without any sophisticated performance optimization. All experiments were carried out on a computer with an Intel Core i7 3.40GHz, 32GB RAM, running Linux.

We applied standardization (zero mean and unit variance for all features) to all datasets as the sole common preprocessing step. For all algorithms (except for 4C), we set the number of clusters to the number of class labels of the respective dataset. We ran FOSSCLU, LDA-k-means, ICA-k-means, PCA-k-means and SubKmeans each in total 40 times. All of these algorithms are based on a cost function and may run into local minima. In order to account for those insufficient outcomes, we sorted the results by their costs of the respective cost function and removed the half with higher costs. Then we determined the average NMI-score for each algorithm.

ORCLUS has, besides k, the two additional parameters ϵ and a seeding parameter. For ϵ we ran a grid search over $[2; \min(20, d)]$ and tried this parameter 20 times with different seed values and selected the result that achieved the highest average NMI. 4C requires the user to set three parameters. We ran the algorithm for $\epsilon \in [2; 2 \times d]$, *minPts* $\in [1; 15]$ and $\lambda \in [2; \min(20, d)]$ and selected the overall best NMI.

Our *quantitative* results are presented in Table 2. We can see that SubKmeans is a promising clustering algorithm. It achieves with two exceptions the strongest NMI score. In the remaining lower-dimensional datasets, the advantages of the *clustered* space are not as visible as for the higher-dimensional datasets and SubKmeans is only ranked second.

Yet, even for lower dimensional datasets it may be of an advantage to use SubKmeans in terms of visualization. Figure 3 shows

[2]Compare http://docs.nvidia.com/cuda/ and https://databricks.com/blog/2014/07/21/distributing-the-singular-value-decomposition-with-spark.html

[3]https://archive.ics.uci.edu/ml/datasets.html

[4]http://www.timeseriesclassification.com/dataset.php

[5]https://elki-project.github.io/

Table 2: Real-world results (NMI-value). Results marked with † were either aborted after searching for 24 hours (per dataset) or failed due to memory constraints. In either case we show the best results we were able to achieve up to this point. The found dimensionality m of the *clustered* space for SubKmeans is shown in brackets.

	Wine ($\lvert\mathcal{D}\rvert$=178,k=3,d=9)	Pendigits ($\lvert\mathcal{D}\rvert$=7494,k=10,d=16)	Ecoli ($\lvert\mathcal{D}\rvert$=327,k=5,d=7)	Seeds ($\lvert\mathcal{D}\rvert$=210,k=3,d=7)	Soybean ($\lvert\mathcal{D}\rvert$=47,k=4,d=35)	Symbol ($\lvert\mathcal{D}\rvert$=995,k=6,d=398)	OliveOil ($\lvert\mathcal{D}\rvert$=60,k=4,d=570)	Plane ($\lvert\mathcal{D}\rvert$=210,k=7,d=144)
SubKmeans (m)	0.88 (2)	0.70 (9)	**0.68** (4)	**0.74** (2)	**1.00** (3)	**0.79** (5)	**0.75** (3)	**0.89** (6)
PCA-k-means	0.71	0.61	0.46	0.73	0.57	0.78	0.61	0.83
ICA-k-means	0.42	0.68	0.61	0.73	0.61	**0.79**	0.53	0.83
LDA-k-means	**0.93**	0.69	0.62	0.64	0.93	0.56	0.71	0.16
FOSSCLU	0.87	**0.77**	0.62	0.62	**1.00**	0.00†	0.00†	0.00†
ORCLUS	0.30	0.60	**0.68**	0.09	0.65	0.00†	0.40†	0.66
4C	0.52	0.55	0.09	0.53	0.64	0.21†	0.58†	0.78

a comparison between the transformation found by SubKmeans compared to PCA. We can see that the two most important features in the *clustered* space looks quite promising revealing relevant structures not only for synthetic datasets (like in Figure 1), but also for real-world applications. In contrast, the first two features of the *noise* space, do not reveal any major cluster structure to the naked eye and we come to the conclusion that the transformation is indeed able to separate important from unimportant features. If we compare these findings to the first two features of the PCA transformed data, we can see that this transformation reveals clear structures only in few cases (e.g. Wine and Symbol). Even for those datasets, the structural information revealed by SubKmeans often seems superior (e.g. Wine).

We performed additional experiments to reveal the effects of heavily overestimating the number of 'real' clusters in a dataset on the transformation matrix. The results show that the shape of the *clustered* space does not change greatly. Figure 4 shows two examples for Pendigits and Symbols, where k was set to twice the number of class labels. The first two dimensions of the clustered space look quite similar to the one shown in Figure 3—even with twice the number of clusters. This shows that SubKmeans indeed finds subspaces, in which the structural information of the clustering is most prominent and that our method also handles situation gracefully, in which the number of clusters is heavily overestimated.

This property may help identify the right number of clusters by visual inspection, even if the number of clusters for the first attempts were initially overestimated. Moreover, sometimes data scientists set a high value for k and process the found partition further to create non-spherical clusterings, for instance, by using the found partitions in combination with single-linkage agglomerative clustering [31]. Even in those situations the found transformation is very useful to verify these clusters visually.

Though, we do not want to conceal that the overestimation of k also has side effects on m. If we significantly overestimate the number of clusters, the algorithm will find more spurious structures within the noise space, with the consequence that m increases. Yet, since V is sorted by 'importance' this only has a rather small effect on the most significant components of V and by adjusting k accordingly or by inspecting the corresponding eigenvalues and choosing a threshold-value manually, this effect can be mitigated.

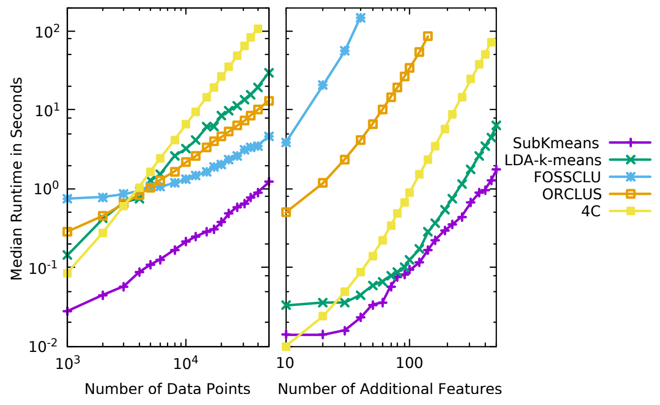


Figure 2: Runtime for varying d and $\lvert\mathcal{D}\rvert$

3.1 Runtime Experiments

We want to support our claim that, compared to other algorithms, incorporating some kind of dimensionality reduction, SubKmeans is indeed a fast algorithm. We compare it to FOSSCLU, LDA-k-means, ORCLUS and 4C. As basis we used a synthetic dataset containing data points, equally distributed over three well-separated Gaussian clusters. For all algorithms, we initially determined the parameter values yielding the best NMI score and kept them fixed.

We performed two experiments. In the first experiment, we wanted to evaluate the runtime behavior by sampling an increasing number of additional data points. In the second experiment, we wanted to evaluate how SubKmeans behaves for an increasing dimensionality. For this experiment we added additional *noise*-features drawn from a single Gaussian.

Figure 2 shows the results of both experiments. For each dataset and dimensionality configuration, we ran each algorithm ten times and present the median result. We can see that SubKmeans is indeed—among the tested—the fastest algorithm.

The closest competitor to SubKmeans for high-dimensional datasets is LDA-k-means. This can be expected, since this algorithm uses almost the same operations—assignment and update steps of k-means and the eigen-decomposition of scatter matrices—with an additional matrix inversion.

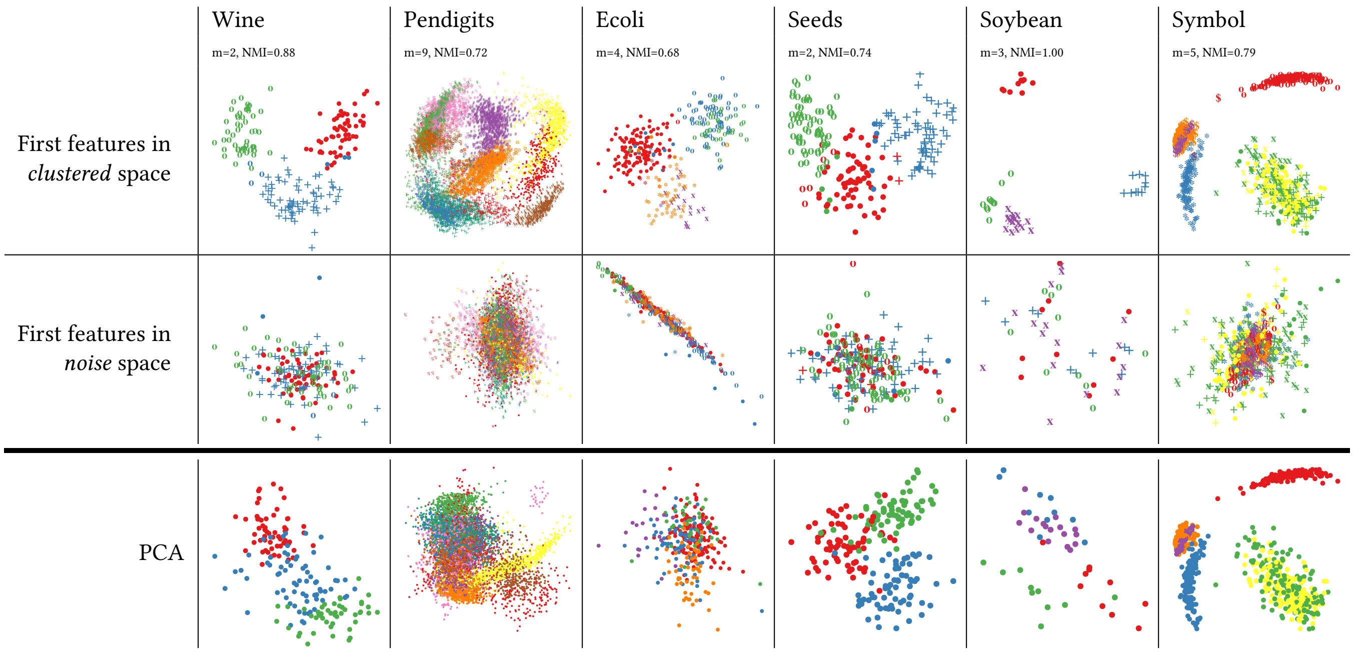


Figure 3: The diagrams show a comparison of PCA vs. the transformation by SubKmeans for the found solution with minimal costs for each dataset. The *clustered* space dimensionality m and the NMI of the model are given below the dataset name. Colors represent the class labels and symbol represent the predicted cluster labels. The first row shows the scatter plot of the first two dimensions of the *clustered* space. The second row shows the scatter plot of the first two dimensions of the *noise* space. The last row shows the first to two features found by PCA.

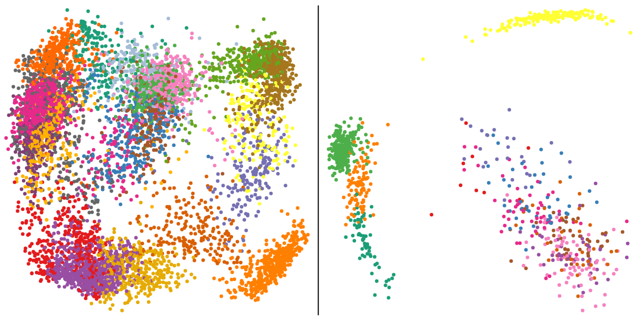


Figure 4: Diagrams show the two most important features of the clustered space of the Pendigits and Symbol datasets, where the number of clusters is two times the real cluster count.

4 DISCUSSION AND RELATED WORK

k-means gets its unbroken popularity from its simplicity, adaptability and execute speed. It has inspired numerous extensions and adaptations, many of which can also be combined in SubKmeans. We present some of these extensions in the following.

Further, we discuss SubKmeans in the context of other methods and algorithms, which either are or incorporate some kind of dimensional reduction or feature selection steps.

k-means Extensions

The simplicity and effectiveness of the k-means algorithm has inspired many adaptations and extensions of the basic idea. In the following we provide only a small, non-exhaustive overview.

The basic idea has been adopted to use the median instead of the mean [18]. The hard cluster assignment is weakened in Fuzzy-c-means [11], in which—much like in mixture models—each data point is proportionally assigned to multiple clusters.

There have been proposed different ways to initialize the cluster centers. The widely used k-means++ algorithm [4] contains an seeding technique, which selects data points as initial centers based on their proportional distance to the previous selected centers. This method has been shown to reduce the number of iterations and achieves, on average, better local minima compared to a random initialization. However, it also has been criticized for its inherent sequential nature. A more recent proposal is k-means|| [5]. In this proposal the initialization of the cluster centers is parallelized, which better fits the growing size of current datasets.

The automated estimation of the number of clusters k has been tackled by different researchers. Either by extending the cost function by a complexity term, like in X-means[25], or by using statistical tests, like in Q-means [16], as a cluster-split criteria. The Linde-Buzo-Gray algorithm [14] doubles the number of clusters for each iteration, but, towards a different end: finding a sufficient codebook for vector quantization. Bisecting k-means [28] follows a hierarchical top-down approach and in each step splits the clusters according to the 2-means.

Some proposals for accelerating the update step are based on kd-trees [23, 24]. Other researchers propose to exploit the triangle inequality [12, 15, 26] to minimize the number of clusters each data point has to be compared to. Further proposals, like [7], suggest exploiting the instruction sets of modern CPUs to parallelize the computation within each core.

A Bayesian treatment of k-means caRn for instance be found in [30], which reverses the roles of expectation and maximization in the classical EM algorithm. In [20] a nonparametric method is presented, which adapts the Dirichlet process together with a modified version of the Gibbs sampling method.

k-means can be extended to non-linearly separable input data via the kernel-trick. This idea was unified with spectral clustering into a common framework [9] and there seems to be a connection to SubKmeans as well, which we are planning to explore more in the future.

Some researchers [8, 10, 32] propose to combine k-means with Linear Discriminant Analysis (LDA), in order to perform a simultaneous dimensional reduction. We discuss the LDA-k-means algorithm **??** in more detail below.

Dimensionality Reduction Techniques

The goal of dimensionality reduction is to find a lower dimensional representation of a high-dimensional dataset. The proposed methods, thereby, make quite different assumptions and introduce a range of different optimization goals.

Probably the single most famous method falling into this category is PCA, which minimizes the reconstruction error. An alternative formulation searches for the directions with the highest variance. Kernel PCA [27] combines this idea with the kernel-trick.

Independent Component Analysis (ICA) [17] is not directly a dimensionality reduction mechanism, but aims at identifying statistically independent sources within the data.

Isomap [29] is a method, which finds a nonlinear lower dimensional representation based on the estimated geometric properties of the data's manifold.

Subspace and Projected Clustering

Many different proposals for clustering in high-dimensional datasets can be grouped under the banner of subspace and projected clustering algorithms. Algorithms in this family assign each cluster its own subspace, in which some cluster-criteria is fulfilled. This distinguishes them from the approach taken in SubKmeans, which identifies one common subspace. Algorithms like CLIQUE [3] or PROCLUS [1] identify only axis-parallel subspaces. This approach has the benefit that clusters are easily explainable by its subspace-features. Algorithms that try to find clusters in arbitrarily oriented subspaces encompass for instance ORCLUS [2] or 4C [6].

Common Subspace Clustering

Besides our own proposal SubKmeans, we identified two other algorithms following the same theme of identifying a single, common subspace for all clusters. The advantage of this approach is that it allows a researcher to interpret the relationship between data points within a cluster, as well as the relationship between clusters within the common subspace.

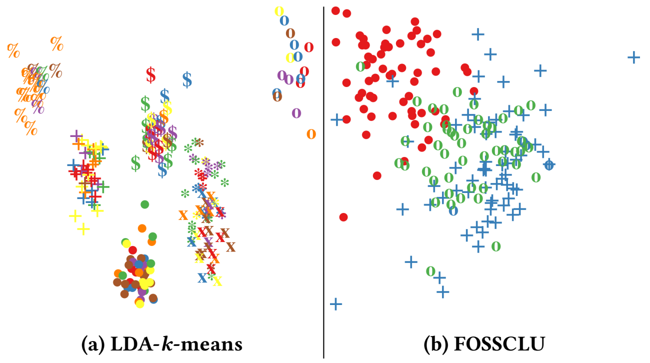


Figure 5: The left scatter plot shows an example of the overfitting behavior of LDA-k-means for the Plane dataset. The right plot shows an example of FOSSCLU for the Wine dataset. It shows the first two features of the *clustered* space. We can see that the cluster structures are occluded, because the features are not sorted by 'importance'. Colors represent class labels. Clusters are symbol-coded.

LDA-k-means [10] combines the supervised Linear Discriminant Analysis (LDA) with k-means. Thereby, it iterates between partitioning the data using k-means and transforming the data into a subspace via LDA, in which the within cluster variance is minimized and the between cluster variance is maximized. However, we identified several drawbacks of this approach compared to SubKmeans. First the subspace is fixed to $(k-1)$-dimensions. Secondly, LDA is known to overfit with growing number of dimensions [22]. We also experienced this behavior in our experiments. Especially for the high-dimensional data sets, the found partition in the transformed space look very confident. The clusters in the respective subspaces are well separated yielding summed squared distances near zero. Yet, at the same time these results yield very bad NMI scores. Figure 5 (a) shows this behavior for the Plane dataset, where we show the scatter plot of the first two components. In that sense, LDA-k-means ignores structures disagreeing with the current partitioning assumption. SubKmeans chooses a different path: the ignored structure must be better explainable by the single noise cluster or otherwise has to be overcome by the clusters in the *clustered* space. Moreover, we see the maximization of the inter-cluster variance can be problematic in a situation, where the number of clusters is overestimated. In such a scenario the LDA tries to push sub-clusters apart that are actually part of a proper bigger cluster. In combination with the overfitting issue, this may not be recognized in high-dimensional datasets. It is possible to use a regularized version of LDA [32], however, this introduces a further parameter, which the user has to specify. At the same time such a regularization parameter would also be possible for SubKmeans—a further research area we are planning to explore.

FOSSCLU [13] is the second algorithm and shares many similarities with SubKmeans. It also searches for an orthonormal transformation, which projects the data onto a subspace. Simultaneously, it uses the EM-algorithm to optimize a crisp GMM. It uses the Minimum Description Length principle to automatically determines the values of k and m. Theoretically, compared to SubKmeans, it should better adapt to the data, much in the same way a classic GMM does compared to k-means. Yet, we determined several speed-related aspects, which speak for the use of SubKmeans over FOSSCLU. First, the transformation matrix is quite difficult to optimize and the proposed algorithm—based on the Jacobi transformation—is very time-consuming, providing only very few opportunities for further optimizations. This is why the proposed algorithm only optimizes the transformation between the two subspaces for a fixed m, but not within them. This means that the transformation matrix in FOSSCLU projects the for the GMM essential structure onto the subspace, but in an arbitrary orientation. The features are therefore not sorted by importance and essential structures within the *clustered* space might still be occluded from the viewer. Figure 5 (b) shows this behavior for the Wine dataset. Here the found solution yields an NMI score of 0.92 and $m = 7$. However, when the user fixes $m = 2$, the transformation yields a *clustered* space similar to that of SubKmeans in Figure 3. In SubKmeans on the other hand, the components of V are sorted corresponding to the feature importance, highlighting the most prominent structures by the first components. Additionally, SubKmeans brings the inherent property to optimize m within each iteration, whereas FOSSCLU has to start a whole EM-optimization for each possible value of m. In summary, FOSSCLU has a major speed disadvantage for datasets with more than twenty dimensions and our experiments confirm that SubKmeans is several orders of magnitude faster than FOSSCLU.

However, we should point out that—as there is no free lunch–SubKmeans has, like all clustering algorithms, its *limitations*. Specifically, it fails at the same well known examples as the classic k-means version.

5 CONCLUSION

In this paper, we propose SubKmeans, an extension of the k-means algorithm, which performs clustering and dimensionality reduction simultaneously without the need for additional parameters (except for k). Thereby, it finds a k-partition together with a—for this clustering—optimal subspace. It has many desirable properties. The most basic, non-optimized version is easy to implement and only uses standard features provided by all data computing frameworks. Even in this basic implementation, it is fast, though this could most likely further increased, for instance by a more sophisticated way of calculating the scatter matrices. Also, one could easily incorporate many other extensions proposed for k-means, e.g. for further speed-ups or estimation of k.

Future efforts may be directed towards the approximation of the transformation matrix V. We are very confident that this can be achieved quite easily by using a randomized singular value decomposition, resulting in further performance improvements for very high-dimensional datasets.

REFERENCES

[1] Charu C Aggarwal, Joel L Wolf, Philip S Yu, Cecilia Procopiuc, and Jong Soo Park. 1999. Fast algorithms for projected clustering. In *ACM SIGMoD Record*, Vol. 28. ACM, 61–72.
[2] Charu C. Aggarwal and Philip S. Yu. 2000. Finding Generalized Projected Clusters in High Dimensional Spaces. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*. ACM, New York, NY, USA, 70–81. https://doi.org/10.1145/342009.335383
[3] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. *Automatic subspace clustering of high dimensional data for data mining applications*. Vol. 27. ACM.
[4] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1027–1035.
[5] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Scalable k-means++. *Proceedings of the VLDB Endowment* 5, 7 (2012), 622–633.
[6] Christian Böhm, Karin Kailing, Peer Kröger, and Arthur Zimek. 2004. Computing clusters of correlation connected objects. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 455–466.
[7] Christian Böhm and Claudia Plant. 2015. Mining Massive Vector Data on Single Instruction Multiple Data Microarchitectures. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, 597–606.
[8] Fernando De la Torre and Takeo Kanade. 2006. Discriminative cluster analysis. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 241–248.
[9] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. 551–556. https://doi.org/10.1145/1014052.1014118
[10] Chris Ding and Tao Li. 2007. Adaptive dimension reduction using discriminant analysis and k-means clustering. In *Proceedings of the 24th international conference on Machine learning*. ACM, 521–528.
[11] Joseph C Dunn. 1973. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. (1973).
[12] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *ICML*, Vol. 3. 147–153.
[13] Sebastian Goebl, Xiao He, Claudia Plant, and Christian Böhm. 2014. Finding the optimal subspace for clustering. In *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 130–139.
[14] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE transactions on information theory* 44, 6 (1998), 2325–2383.
[15] Greg Hamerly. 2010. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*. SIAM, 130–140.
[16] Greg Hamerly and Charles Elkan. 2004. Learning the k in k-means. (2004).
[17] Aapo Hyvärinen and Erkki Oja. 2000. Independent component analysis: algorithms and applications. *Neural networks* 13, 4 (2000), 411–430.
[18] Anil K Jain and Richard C Dubes. 1988. *Algorithms for clustering data*. Prentice-Hall, Inc.
[19] Ian Jolliffe. 2002. *Principal component analysis*. Wiley Online Library.
[20] Brian Kulis and Michael I Jordan. 2012. Revisiting k-means: New algorithms via Bayesian nonparametrics. *In Proceedings of the 23rd International Conference on Ma- chine Learning* (2012).
[21] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
[22] Dijun Luo, Chris HQ Ding, and Heng Huang. 2011. Linear Discriminant Analysis: New Formulations and Overfit Analysis.. In *AAAI*.
[23] Andrew W Moore. 1999. Very fast EM-based mixture model clustering using multiresolution kd-trees. *Advances in Neural information processing systems* (1999), 543–549.
[24] Dan Pelleg and Andrew Moore. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 277–281.
[25] Dan Pelleg, Andrew W Moore, and others. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters.. In *ICML*, Vol. 1.
[26] Steven J Phillips. 2002. Acceleration of k-means and related clustering algorithms. In *Workshop on Algorithm Engineering and Experimentation*. Springer, 166–177.
[27] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1997. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*. Springer, 583–588.
[28] Michael Steinbach, George Karypis, Vipin Kumar, and others. 2000. A comparison of document clustering techniques. In *KDD workshop on text mining*, Vol. 400. Boston, 525–526.
[29] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.
[30] Max Welling and Kenichi Kurihara. 2006. Bayesian K-Means as a" Maximization-Expectation" Algorithm.. In *SDM*. SIAM, 474–478.
[31] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, and others. 2008. Top 10 algorithms in data mining. *Knowledge and information systems* 14, 1 (2008), 1–37.
[32] Jieping Ye, Zheng Zhao, and Mingrui Wu. 2007. Discriminative K-means for Clustering. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*. 1649–1656. http://papers.nips.cc/paper/3176-discriminative-k-means-for-clustering