universität wien

BACHELORARBEIT / BACHELOR’S THESIS

Titel der Bachelorarbeit / Title of the Bachelor’s Thesis

„Kernel k-means clustering framework in Python“

verfasst von / submitted by

Johannes Oster

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Bachelor of Science (BSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 033 526
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Bachelor Business Informatics
Betreut von / Supervisor:	Dipl.-Ing. Marian Lux

Abstract

This thesis outlines and evaluates a custom implementation of the Kernel k-means clustering algorithm for multiple dimensions in Python. It benchmarks its performance against well-established clustering algorithms from scikit-learn[1] including Agglomerative Clustering, Spectral Clustering and Gaussian Mixture. The quality of the clustering outcomes is evaluated using several metrics, including the Silhouette Score, Calinski-Harabasz Index, Davies-Bouldin Index, and Sum of Variances. The custom implementation demonstrated competitive performance in low-dimensional small and large data sets but higher sensitivity to growing data dimensionality ($n_{features} \geq 100$) than the other algorithms.

[1]https://scikit-learn.org/stable/modules/clustering.html

Kurzfassung

Diese Arbeit beschreibt und evaluiert eine Implementierung des Kernel k-means clustering Algorithmus für ein- und mehrdimensionale Datensätze in Python. Sie misst dessen Leistung im Vergleich zu etablierten Clustering-Algorithmen aus scikit-learn[1], einschließlich Agglomerative Clustering, Spectral Clustering and Gaussian Mixture. Die Qualität der Clustering-Ergebnisse wird anhand mehrerer Metriken bewertet, einschließlich des Silhouette Scores, des Calinski-Harabasz-Index, des Davies-Bouldin-Index und der Summe der Varianzen. Die Implementierung zeigte eine konkurrenzfähige Leistung bei kleinen und großen Datensätzen mit geringer Dimension, wies jedoch eine höhere Sensibilität für hochdimensionale Datensätze auf ($n_{\text{features}} \geq 100$) als die anderen Algorithmen.

[1]https://scikit-learn.org/stable/modules/clustering.html

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

1.1 Problem Statement

Clustering is a fundamental task in data analysis, with applications spanning numerous fields. It is a process that groups similar data points based on characteristics such as density, centroid proximity, or hierarchical relationship, enabling us to understand the structure of the underlying data sets [SK21].

Traditional clustering algorithms, such as k-means, rely on distance-based similarity measures like Euclidean or Manhattan distance [AP22]. They excel when the data points are linearly separable, i.e., they can be separated by a line in two-dimensional space, a plane in three-dimensional space, and like manner. However, real-world data is often complex, with multiple factors contributing to its formation, leading to irregularly shaped, non-linear structures. Hence, drawing linear boundaries to separate the data points or clusters can become challenging.

To address this limitation, kernel methods have been proposed. The idea behind kernel methods is to implicitly map the data to a higher-dimensional space where the data can potentially become linearly separable [Gir02] - this is often referred to as the "kernel trick". Kernel k-means, in particular, applies this method to the k-means algorithm, making it better suited for non-linearly separable data [DGK04][DGK05].

Despite these advantages, Kernel k-means implementations are less prominent than their distance-based counterparts and could often benefit from more evaluation and comparison with other clustering techniques. Therefore, this thesis aims to distinguish itself by revisiting the basics and providing a clear, complete practical explanation of the original Kernel k-means algorithm.

1.2 Goals of Work

The primary goals of this work are to:

1. Develop a well-documented and efficient implementation of the Kernel k-means algorithm in Python usable as a framework.

2. Assess the efficacy of the proposed Kernel k-means implementation qualitatively and by quantitatively benchmarking it against popular clustering algorithms, such as Agglomerative Clustering and Spectral Clustering, with various synthetic data sets exhibiting different properties such as size, dimension and degree of separability.

2 Related Work

The mathematical foundations of kernel methods have their roots in the early 20th century. Still, their integration into data analysis is a development of the mid to late 20th century and early 21st century [Fre03][Hil89][Aro50][Aiz64][BGV92][SSM98][DGK04][FCMR08]. Since then, the field of kernel-based clustering algorithms has seen significant advancements and applications in various domains.

Kernel methods are not restricted to k-means clustering. Several researchers have focused on enhancing traditional algorithms by integrating a kernel function. For instance, the paper "A novel kernel self-organizing map algorithm for clustering" by Ning Chen, Hongyi Zhang, and Jiexin Pu introduces a kernel-enhanced Self-Organizing Maps (SOM) algorithm. It proposes the Extended Kernel SOM (EKSOM) method that significantly reduces errors and increases accuracy in clustering tasks compared to the original SOM algorithm [CZP09].

Some works specifically introduce adaptations of the Kernel k-means to tackle its limitations. Zhang and Rudnicky propose a new clustering strategy in their paper, "A large scale clustering scheme for Kernel k-means", working with one "kernel" at a time which enables efficient handling of large data volumes [ZR].

The recent paper "Statistical and Computational Trade-Offs in Kernel K-Means" by Calandriello and Rosasco studies the computational requirements and proposes a Nyström approach to Kernel k-means, demonstrating the same accuracy as the exact Kernel k-means but requiring only a fraction of the computations [CR19].

In the context of large data sets, the paper "Approximate Kernel k-means: solution to large scale kernel clustering" presents an efficient approximation for the Kernel k-means algorithm. The authors propose avoiding the computation of the full kernel matrix by restricting the cluster centres to a small subspace spanned by randomly sampled data points [CJHJ11].

Shi Yu, Tranchevent L., Xinhai Liu et al. propose an optimized Kernel k-means algorithm, leveraging an alternating minimization framework to optimize the cluster membership and kernel coefficients for a nonconvex problem. They claim this efficiency makes it particularly suitable for large-scale data sets [YTL$^+$12].

While these works offer valuable adaptations and advancements on the Kernel k-means algorithm, a comprehensive explanation and benchmarking of the raw, unmodified Kernel k-means algorithm often seems less emphasized, which is what this thesis will focus on.

3 Framework Design

3.1 Theoretical Background

The Kernel k-means algorithm is an iterative method that clusters data points $x_i \in \mathbb{R}^N$ into k distinct groups π_l. The grouping is achieved by implicitly mapping the data points to a higher-dimensional feature space H through a nonlinear transformation Φ and minimizing the within-cluster scatter (3.2) [MRT18]. Figure 3.1 visualises this transformation explicitly.

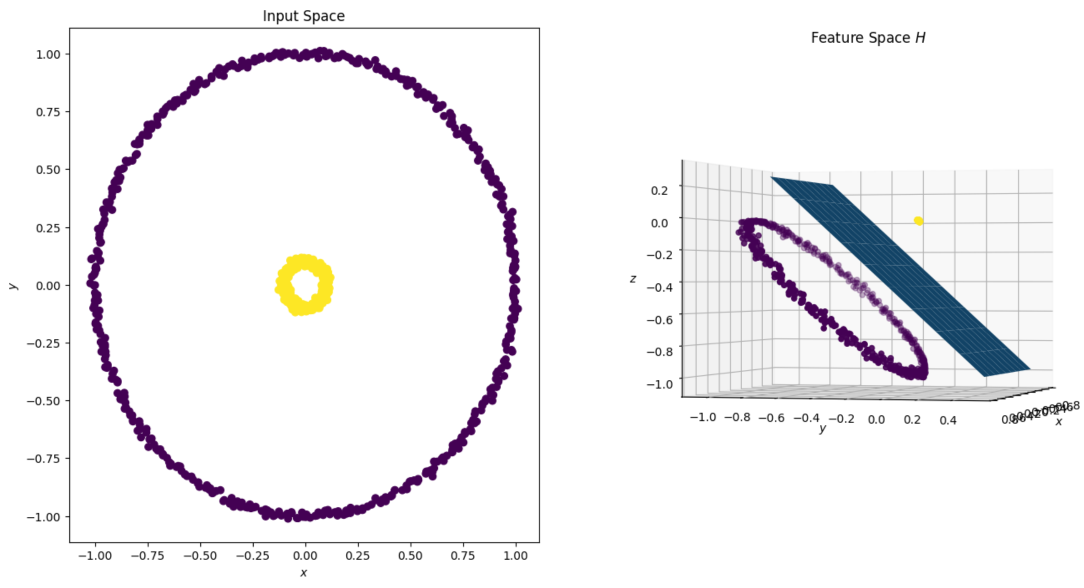

Figure 3.1: Visualizing how an explicit transformation $\phi(x_i)$ leads to linear separability.

Every clustering algorithm relies on some definition of similarity or distance between two data points. In the Kernel k-means algorithm, the squared Euclidian distance is used to measure the similarity between two points $x_i, x_j \in \mathbb{R}^N$ in the higher dimensional feature space H:

$$E(x_i, x_j) = \|\Phi(x_i) - \Phi(x_j)\|^2 \tag{3.1}$$

The algorithm aims to minimize equation 3.1 between each data point and its corresponding cluster centre in the higher-dimensional space. We will denote the centre of a cluster π_l by c_l and a partitioning of points as $\{\pi_{l=1}^k\}$. Our objective function, defined as the sum of the squared distances between data points and their respective cluster centres in the feature space, is:

$$D(\{\pi_{l=1}^k\}) = \sum_{l=1}^{k} \sum_{x_i \in \pi_l} \|\Phi(x_i) - c_l\|^2 \tag{3.2}$$

with

$$c_l = \frac{\sum_{x_j \in \pi_l} \Phi(x_j)}{|\pi_l|}$$

[DGK05]

While theoretically, the dot product of the feature vectors $\langle \Phi(x_i), \Phi(x_j) \rangle$ in the higher-dimensional space could be explicitly calculated, in practice, this is often computationally infeasible. To circumvent this issue, the "kernel trick" is applied. We will define a matrix K where $K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$. This addition lets us rewrite the Euclidian distance 3.1 in the following way:

$$\begin{aligned} E(x_i, x_j) &= \|\Phi(x_i) - \Phi(x_j)\|^2 \\ &= \langle \Phi(x_i) - \Phi(x_j), \Phi(x_i) - \Phi(x_j) \rangle \\ &= \langle \Phi(x_i), \Phi(x_i) \rangle - 2\langle \Phi(x_i), \Phi(x_j) \rangle + \langle \Phi(x_j), \Phi(x_j) \rangle \\ &= K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j) \end{aligned} \tag{3.3}$$

and our current objective function 3.2 as:

$$\begin{aligned} D(\{\pi_{l=1}^k\}) &= \sum_{l=1}^{k} \sum_{x_i \in \pi_l} \|\Phi(x_i) - c_l\|^2 \\ &= \sum_{l=1}^{k} \sum_{x_i \in \pi_l} \left(\langle \Phi(x_i), \Phi(x_i) \rangle - 2\frac{\sum_{x_j \in \pi_l} \langle \Phi(x_i), \Phi(x_j) \rangle}{|\pi_l|} + \frac{\sum_{x_j, x_h \in \pi_l} \langle \Phi(x_j), \Phi(x_h) \rangle}{|\pi_l|^2} \right) \\ &= \sum_{l=1}^{k} \sum_{x_i \in \pi_l} \left(K(x_i, x_i) - 2\frac{\sum_{x_j \in \pi_l} K(x_i, x_j)}{|\pi_l|} + \frac{\sum_{x_j, x_h \in \pi_l} K(x_j, x_h)}{|\pi_l|^2} \right) \end{aligned} \tag{3.4}$$

[KLLL05]

With this knowledge, we know that at each iteration for a given x_i we try to find a cluster π_l that fulfils:

$$\arg \min_{l \in \{1,2,\ldots,k\}} \left(K(x_i, x_i) - 2\frac{\sum_{x_j \in \pi_l} K(x_i, x_j)}{|\pi_l|} + \frac{\sum_{x_j, x_h \in \pi_l} K(x_j, x_h)}{|\pi_l|^2} \right) \tag{3.5}$$

[TL09]

3.2 The Matrix K

We previously introduced the matrix K as an indirect substitute for the inner product of two points in the feature space H. This so-called "kernel matrix" K results from applying a kernel function to all pairs (x_i, x_j) of a data set $X \subseteq \mathbb{R}^N$ and is an $n \times n$ matrix, where n is the number of data points ($n = |X|$).

A kernel function k computes the distance between two data points $x_i, x_j \in X$ in a transformed space without explicitly mapping the data points to that space - i.e. without explicit knowledge and application of the transformation Φ. As previously mentioned, this is sometimes referred to as the "kernel trick" [SS18] and significantly reduces the computational effort of the algorithm.

3.3 Kernel Functions

A kernel function is central to the Kernel k-means algorithm by defining a similarity measure in the transformed feature space.

Definition 3.3.1. A function $k : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ is a valid kernel function if, and only if, it is symmetric and positive semidefinite, that is:

1. $k(x_i, x_j) = k(x_j, x_i)$
2. $\forall c \in \mathbb{R}^N, c^T K c \geqslant 0$

The necessity for these properties for the algorithm to work is based on the fact that we want to use an inner product as a measure of similarity between two points in a higher dimensional space H. The proof that all symmetric positive semidefinite kernels induce such an inner product is out of the scope of this thesis but can be found in the book "Foundations of Machine Learning" by M. Mohri, A. Rostamizadeh, and A. Talwalkar [MRT18].

There are several common kernel functions, including the Gaussian kernel (3.6), Polynomial kernel (3.7), or Sigmoid kernel (3.8). [LFYZ08]

$$K_{Gaussian}(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \tag{3.6}$$

$$K_{Polynomial}(x_i, x_j) = (x_i \cdot x_j + \theta)^d \tag{3.7}$$

$$K_{Sigmoid}(x_i, x_j) = \tanh(\kappa(x_i \cdot x_j) + \theta) \tag{3.8}$$

κ, θ, σ and d are parameters of the kernel functions. Due to its simplicity and low computational cost [SD19], the framework includes a radial basis function (rbf) kernel implementation as a default. However, the algorithm is kernel-agnostic and takes a kernel function as an argument.

The rbf kernel incorporates a parameter σ that controls the width of the kernel. By default, σ is set to the square root of the median of the squared distances between all pairs of points, as this typically provides a good balance between capturing the structure in the data without overfitting [M.T19].

$$\sqrt{\text{median}\left(\{d^2(x_i, x_j)\}_{1\leq i<j\leq n}\right)} \tag{3.9}$$

where

$$d(x_i, x_j) = \|x_i - x_j\|^2$$

Adjusting σ impacts the model's sensitivity to the pairwise distances between data points, with a smaller σ focusing more on closer points, potentially leading to overfitting. In comparison, a larger σ would put more emphasis on distant points, possibly resulting in underfitting.

Figure 3.2 shows the impact of different σ values on scikit-learn's support vector classification algorithm (svc) using the rbf kernel[1]. Like Kernel k-means, svc leverages kernel functions to transform the data into a higher-dimensional space. This algorithm has been chosen over the custom Kernel k-means algorithm for this demonstration since it calculates a decision boundary that visualizes the impact of σ more effectively than merely observing colour-coded data points.

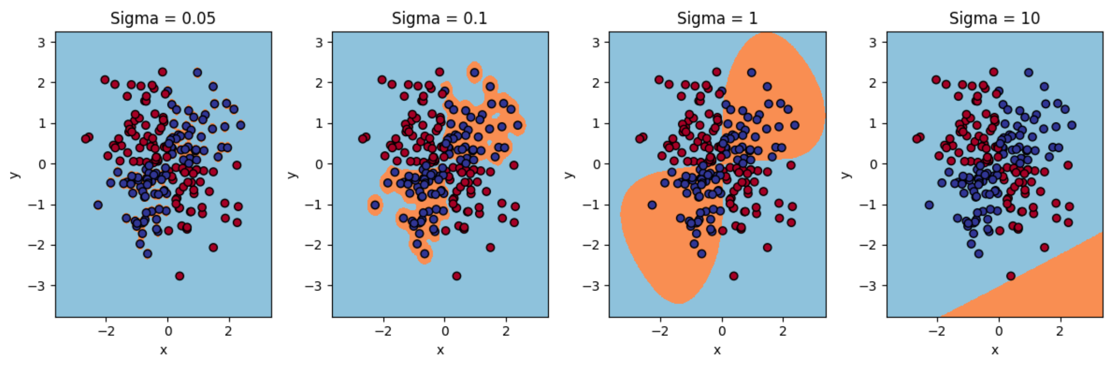

Figure 3.2: Impact of different σ (Sigma) values on the clustering result of scikit-learn's support vector classification algorithm

When dealing with dense datasets, a smaller value of σ is preferred to delineate clear boundaries between clusters. In contrast, a larger σ is advisable to prevent overfitting for sparser datasets.

Figure 3.3 illustrates the consequences of choosing σ incorrectly in the custom Kernel k-means implementation on sparse and dense data sets.

[1]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

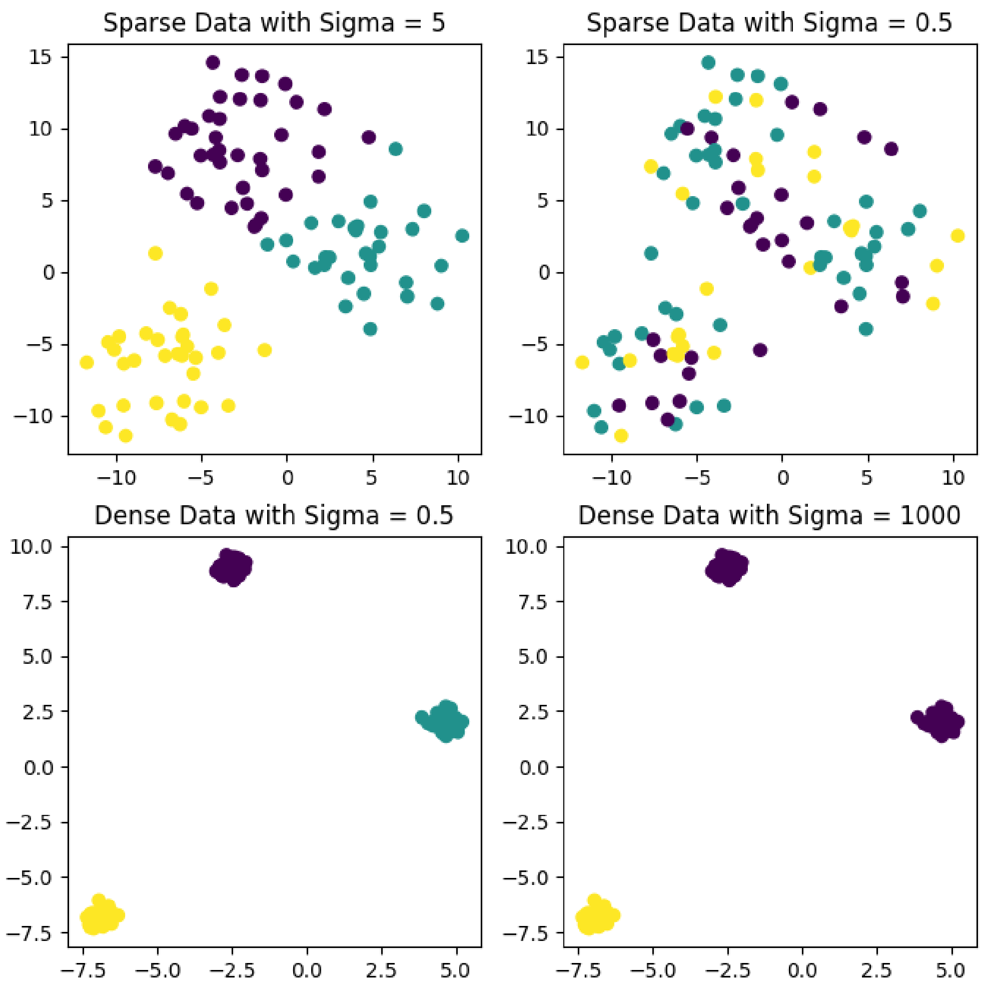

Figure 3.3: Impact of wrong σ (Sigma) on Kernel k-means

3.4 The Algorithm

The Kernel k-means algorithm can be distilled into a series of systematic steps. The key steps are the computation of the kernel matrix K using the provided kernel function, initialization of the clusters (which will be discussed in the following section 3.5), and iterative assignments of data points to clusters. The cluster a point is assigned to is based on which cluster minimizes the within-cluster scatter. The algorithm iteratively reassigns points to clusters until a convergence criterion is met, typically when the change in cluster assignments is no longer significant or there is no change (see 3.6).

A typical run of the algorithm can be described in the following high-level pseudocode:

Algorithm 1: Kernel k-means

```
input  : Data set $X$, Number of clusters $k$, Kernel function $k\_func$, Convergence
         tolerance $tol$
output : Cluster assignments A

1 Compute kernel matrix $K = k\_func(X)$
2 Initialize $k$ clusters (see 3.5)
3 while change in assignments $A$ or change in objective function $> tol$ do
4     for each data point $x_i \in X$ do
5         Assign it to the cluster $\pi_l$ that minimizes:
          $\arg\min_{l \in \{1,2,\dots,k\}} \left( K(x_i, x_i) - 2\frac{\sum_{x_j \in \pi_l} K(x_i,x_j)}{|\pi_l|} + \frac{\sum_{x_j,x_h \in \pi_l} K(x_j,x_h)}{|\pi_l|^2} \right)$
6     end
7 end
8 return cluster assignments $A$
```

[TL09][DGK04]

3.5 Initialization of Clusters

As mentioned in 1, the algorithm starts with an initial assignment of data points to clusters. This assignment plays a pivotal role in the accuracy of the overall algorithm because k-means and its variations, like Kernel k-means, are known to get stuck in local minima [PCDX20]. One option is randomly assigning each data point to one of the clusters.

3.5.1 Random initialization

In the case of random initialization, we randomly select k points from the data set and assign them as the initial centroids. The major advantage of this strategy is its simplicity and speed, even for large data sets. However, it also comes with the drawback of inherent instability, meaning different runs of the same algorithm on the same data will most likely produce different clustering results. Moreover, it can sometimes lead to poor convergence speed and local optima. The latter is particularly pronounced in high-dimensional spaces. In the worst case, some clusters may stay empty after the initial assignment, causing further complications.

In summary, while random initial cluster assignment offers speed and simplicity, it can lead to inconsistent and potentially sub-optimal clustering results. Thus, an alternative initialization method has been implemented as well.

3.5.2 k-means++

An alternative approach would be to use a more sophisticated initialization method like k-means++, which can provide a better starting point and potentially lead to better final cluster assignments.

The fundamental idea behind k-means++ is to initiate the algorithm with 'k' centroids that are distant from each other, assuming that these distant points are likely to be in different clusters.

Algorithm 2: k-means++

```
input  : Data set $X$, Number of clusters $k$
output : Cluster assignments $A$

1 Randomly select the first centroid $c_1$ from $X$
2 for $l = 2$ to $k$ do
3     for each data point $x_i \in X$ do
4         Compute the distance $d$ between $x_i$ and the nearest center $c_m$ that has
            already been chosen
5     end
6     Select one new data point at random as a new centre, using a weighted
        probability distribution where a point $x_j$ is chosen with probability
        proportional to $d^2$
7 end
8 return $\left[ \underset{c_j \in \text{centroids}}{\arg\min} \|x_i - c_j\|^2 : x_i \in X \right]$
```

[AV07]

3.5.3 Comparison

To compare the two approaches 100 iterations with $n_{samples} = 1000$, $n_{features} = 3$ and $n_{clusters} = 5$ have been performed. The results reveal that the k-means++ initialization outperforms the random initialization, as demonstrated in tables 3.1 and 3.2, even though it comes with more variability. The used metrics are described in 5.1.1.

Initialization	Silhouette	Calinski	Davies	Sum of Vars
random	6.31e-01 ± 8.66e-02	3.32e+03 ± 2.13e+03	6.04e-01 ± 2.17e-01	3.15e+01 ± 2.05e+01
k-means++	6.69e-01 ± 9.28e-02	5.82e+03 ± 2.46e+03	5.75e-01 ± 2.55e-01	1.70e+01 ± 4.95e+00

Table 3.1: Means ± Standard Deviations of Kernel k-means scores with different initial clustering approaches

Initialization	Silhouette	Calinski	Davies	Sum of Vars
random	6.27e-01	2.63e+03	5.61e-01	2.41e+01
kmeans++	6.76e-01	5.93e+03	4.72e-01	1.50e+01

Table 3.2: Medians of Kernel k-means scores with different initial clustering approaches

3.6 Convergence Criterion

The algorithm stops iterating when the change of the normalized result of the objective function between two iterations falls below a certain threshold or there's no change in cluster assignments.

Defining this threshold is often a matter of balancing computational efficiency with the quality of the solution. A smaller threshold would lead to a more precise solution but might require more computational time. Conversely, a larger threshold could lead to quicker execution but at the risk of stopping prematurely and missing a better solution.

The second condition - no change in cluster assignments - is a more definitive stopping criterion. If the cluster assignments do not change from one iteration to the next, the algorithm has reached a state where further iteration would not change the clustering solution. In this case, the algorithm has reached a stable solution, and any further computation would be redundant. The algorithm stops at this point.

4 Implementation

The algorithm is developed in Python 3 and relies on NumPy[1] for numerical operations and efficient matrix computations. It is designed as a standalone function that takes as input the data tensor, the number of clusters, a kernel function, the maximum number of iterations and a tolerance level that determines the convergence criterion (3.6).

This modular approach ensures that while the main function lays out the skeletal structure of the algorithm, critical components, such as the kernel function and the initial clustering algorithm, can be seamlessly swapped out.

4.1 Installation and Testing

To install the package for local use, it is recommended to do so via `pip install python-kkmeans`[2] or directly from the repository by executing the command: `pip install git+ssh://git@git01lab.cs.univie.ac.at/osterj97/oster-bachelor-thesis.git`.

For further developments or to run the provided examples, it is necessary to download the repository[3] and install NumPy and other dependencies. This can be done using the command `pip install -r requirements.txt`[4].

Testing is set up using `pytest`[5]. Running all tests can be achieved by executing the command `pytest` in the root directory.

4.2 Benchmarking and Examples

The 'example' folder contains a benchmarking command-line interface (cli) and a Jupyter Notebook used to generate all the graphical illustrations and data tables featured in this thesis.

For guidelines on using the benchmarking cli, execute the command `python examples/benchmarking_cli.py -h`. The Jupyter Notebook is designed to be self-explanatory and can be used for further exploration.

[1]https://numpy.org/
[2]https://pypi.org/project/python-kkmeans/
[3]https://git01lab.cs.univie.ac.at/osterj97/oster-bachelor-thesis
[4]The algorithm only depends on Numpy, however for testing and the examples pytest, scikit-learn and matplotlib are required
[5]https://pytest.org/en/7.3.x/contents.html

4.3 Time and Memory Complexity

The time complexity of random cluster initialization is $O(n)$ and $O(n * k^2)$ for the k-means++ due to a nested list comprehension that executes $O(nk)$ operations for k clusters.

The Kernel k-means algorithm's first step, the kernel's calculation scales with $O(n^2 * d)$, where d represents the dimensionality of the data.

For a given iteration of the Kernel k-means, the computation involves creating a 'mask' to extract cluster-specific subsets of the kernel matrix efficiently. This mask creation process represents the association of data points with the current cluster, where 'mask' is a binary matrix of size $n * k$, with '1' indicating the association of a data point with a cluster and '0' otherwise.

Furthermore, a set K_X of all kernel values between all data points and the data points in the current cluster and a set K_{XX} of all kernel values for all pairs of data points belonging to the same cluster are computed. Assuming the worst-case scenario where the mask contains only ones, each usage of K_{XX} and K_X incurs a time complexity of $O(n^2)$, yielding an overall time complexity of $O(n^2k)$ for this loop.

The computation of the distance matrix, which contains the distance of each data point to the centroids of the clusters, has a time complexity of $O(nk)$. Finding new cluster assignments using *argmin* and calculating the new objective value contribute additional time complexities of $O(nk)$ and $O(n)$, respectively. Summing these up for I iterations, where I represents the maximum number of iterations, the total time complexity of the Kernel k-means clustering algorithm is given by $O(I(n^2k + 2nk + n))$, which simplifies to $O(In^2k))$.

The space complexity is largely due to the storage of the kernel matrix ($O(n^2)$) and distance matrix ($O(n * k)$). The other variables take linear space leading to a space complexity of $O(n^2 + nk)$ (in case k is constant or grows slower than n, this simplifies to $O(n^2)$).

5 Evaluation and Discussion

5.1 Methods of evaluation

5.1.1 Quantitative Metrics

To evaluate the performance of the implementation, the following quantitative metrics were used: Silhouette Score (5.1.1), Calinski-Harabasz Index (5.1.1), Davies-Bouldin Index (5.1.1), Sum of Variances (5.1.1).

Silhouette Score

The Silhouette Score measures how similar a data point is to its own cluster compared to others. It ranges from -1 to 1, where a score close to 1 indicates good clustering, a score close to 0 indicates overlapping clusters and a score close to -1 indicates poor clustering. Good clustering is defined as a data point being well-matched to its own cluster and poorly matched or well-separated to neighbouring clusters.

If we denote $a(x_i) \in \pi_l$ as the average dissimilarity of x_i to all other data points of its own cluster π_l and $b(x_i)$ as the average dissimilarity of x_i to all data points of its nearest neighbouring cluster π_j, we can describe the Silhouette Score of x_i as:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{max(a(x_i), b(x_i))} \tag{5.1}$$

The average of all scores over all data points in all clusters is used as the final quality metric of our partitioning $\{\pi_{l=1}^{k}\}$ of the data set:

$$S(\{\pi_{l=1}^{k}\}) = \frac{1}{N} \sum_{l=1}^{k} \sum_{x_i \in \pi_l} s(x_i) \tag{5.2}$$

[Rou87]

Calinski-Harabasz Index

The Calinski-Harabasz Index evaluates the clusters based on the between-cluster dispersion mean and the within-cluster dispersion. As in 3.2, we denote c_l as the cluster's centre π_l and c as the overall centre. With this, the Calinski-Harabasz Index can be defined as:

$$CHI(\{\pi_{l=1}^{k}\}) = \frac{B_k/(k-1)}{W_k/(N-k)} \tag{5.3}$$

with the between-cluster dispersion $B_k = \sum_{l=1}^{k} |\pi_l| ||c_l - c||^2$ and the within-cluster dispersion $W_k = \sum_{l=1}^{k} \sum_{x_i \in \pi_l} ||x_i - c_l||^2$.

Although there is no given range of possible values, a higher index means the clusters are dense and well separated. [CH74]

Davies-Bouldin Index

The Davies-Bouldin Index is designed to identify dense and well-separated clusters. It is the average similarity measure of each cluster with its most similar one, where smaller scores represent better data partitioning. It is computed using intra-cluster and inter-cluster distances. Ranging from 0 to infinity a smaller value typically indicates better clustering.

For cluster π_l we define the average intra-cluster similarity $S(\pi_l)$ as:

$$S(\pi_l) = \frac{1}{|\pi_l|} \sum_{x_i \in \pi_l} ||x_i - c_l||^2 \tag{5.4}$$

The similarity measure $R_(\pi_l, \pi_m)$ between two clusters π_l and π_m is then defined as the ratio of the sum of their average intra-cluster distances and the distance between their centroids - capturing the inter-cluster dissimilarity:

$$R(\pi_l, \pi_m) = \frac{S(\pi_l) + S(\pi_m)}{||c_l - c_m||} \tag{5.5}$$

The Davies-Bouldin Index is the average over the maximum R_{lm} values for each cluster π_l:

$$DBI(\{\pi_{l=1}^k\}) = \frac{1}{k} \sum_{l=1}^{k} \max_{m \neq l} R_{lm} \tag{5.6}$$

[DB79]

Sum of Variances

The Sum of Variances calculates the variance (squared distance from the mean) for each point in its assigned cluster and then sums these variances across all clusters. The values range from 0 to infinity where a smaller sum indicates a more representative clustering where data points are closer to their respective cluster centres.

Let's take $x_i \in \pi_l$ and c_l as the centre of lth cluster π_l. The variance is then defined as the squared distance from the point to the cluster centre:

$$V(x_i \in \pi_l) = ||x_i - c_l||^2 \tag{5.7}$$

The Sum of Variances E for a partitioning $\{\pi_{l=1}^k\}$ of the data set is then the sum of all these variances:

$$E(\{\pi_{l=1}^{k}\}) = \sum_{l=1}^{k} \sum_{x_i \in \pi_l} V_{x_i} \tag{5.8}$$

[Fab94]

5.1.2 Evaluation Scenarios

Quantiative Benchmarking

The performance has been benchmarked against three standard scikit-learn clustering algorithms: Agglomerative, Spectral, and Gaussian Mixture clustering. Those algorithms have been chosen because scikit-learn provides a reasonably similar interface to all of them. This simplifies the benchmarking and reduces potential noise in the benchmarking results due to different parameter tunings. Additionally, the chosen algorithms are representative of different clustering paradigms to provide a broad comparison as Agglomerative clustering is hierarchical, Spectral clustering is graph-based, and Gaussian Mixture is probabilistic. Other clustering algorithms outside of scikit-learn could have been chosen. Still, the assumption was made that the quality within one framework is more similar than between frameworks, which again decreases the complexity of the benchmarking and the chance of additional noise in the evaluation. For a comprehensive performance review, however, it might also be beneficial to include top-performing algorithms from other frameworks.

To conquer the hierarchical structures within our data, making Agglomerative clustering a relevant choice. Given the non-convex clusters expected in the data distribution, Spectral clustering was included due to its ability to capture such structures based on graph representations. The Gaussian Mixture Model was chosen to explore the flexibility in capturing diverse cluster shapes, assuming data might be generated from mixed Gaussian distributions.

This assessment was conducted across two scenarios: increasing data set size with a fixed dimension and fixed data set size with increasing dimensionality. The performance was measured using the metrics described in section 5.1.1. The data sets for the benchmark were generated using scikit-learn.

For each scenario, the performance metric results were plotted against the changing parameter (either data set size or dimensionality), and a line was fitted to these points. The gradients of these lines were compared to evaluate how the performance of the algorithms changes with increasing data size or dimensionality. It is important to note that the interpretation of the gradient's steepness depends on the specific metric - steeper gradients indicate a faster decrease in performance for the Silhouette Score and Calinski-Harabasz Index but an increase in performance for the Davies-Bouldin Index and Sum of Variances. Each benchmarking iteration, defined by a specific data set size or dimension, was executed ten times to counteract the potential variability in the results of an individual algorithm run. The mean value of each metric across all ten runs has been chosen as the representative value for that metric and iteration.

Limitations of the quantitative benchmarking

While this benchmarking approach gives us valuable insights, it has certain limitations.

Firstly, the used 'make_blob' and 'make_circles' functions create relatively simple groupings that may not accurately represent the complexities of real-world data sets. Although incorporating additional data sets with varying shapes, densities, and noise levels could offer a more robust comparison, this has been deemed outside the scope of this thesis.

Furthermore, the benchmarking process has been conducted on a personal computer limiting the feasible data set characteristics since the memory requirement of the implementation is $O(n^2)$ for calculating the kernel matrix.

None of the hyperparameters of the algorithms have been modified. Especially the Gaussian Mixture Model and Spectral Clustering algorithms have hyperparameters that, if not optimized for each data set, could potentially lead to skewed conclusions about their performance.

Similarly, the impact of initialization on the Kernel k-means algorithm, known for its sensitivity to this aspect, has been omitted from consideration. To reduce the computational cost, only random initialization has been used.

Finally, the choice of metrics used for the evaluation may overlook some significant factors, such as outliers in the data.

While important for robust, comprehensive benchmarking, all the above aspects have been considered out of this thesis's scope.

Visual Evaluation

In addition to quantitative benchmarking, a qualitative assessment of the algorithm's performance has been performed. This evaluation was done using visualizations of the algorithm's output on various data sets. No comparison towards other algorithms has been made as the goal was to ensure the algorithm could produce reasonable results for differently shaped data sets. The results can be seen in figure 5.3.

5.2 Results

Figure 5.1 visualizes the impact of increasing data set sizes on the different algorithms (see 5.1.2). The individual coloured dots represent the mean result of one iteration (i.e. the mean of 10 iterations of one data set size). The lines represent the fitted regression that captures the trend of the data points, which has been computed using Numpy's *polyfit* function. Table 5.1 summarizes the gradients of those lines. In this and all following tables, a single asterisk * marks the best performance, and two asterisks ** the worst.

To gain additional insights, table 5.2 and 5.3 display the means, standard deviations and medians of the scores for each algorithm of the results of scenario one.

Figure 5.2 visualizes the impact of increasing data set dimension on the different algorithms (see 5.1.2). Table 5.4 summarizes the fitted regression's gradients that capture the trend of the scores.

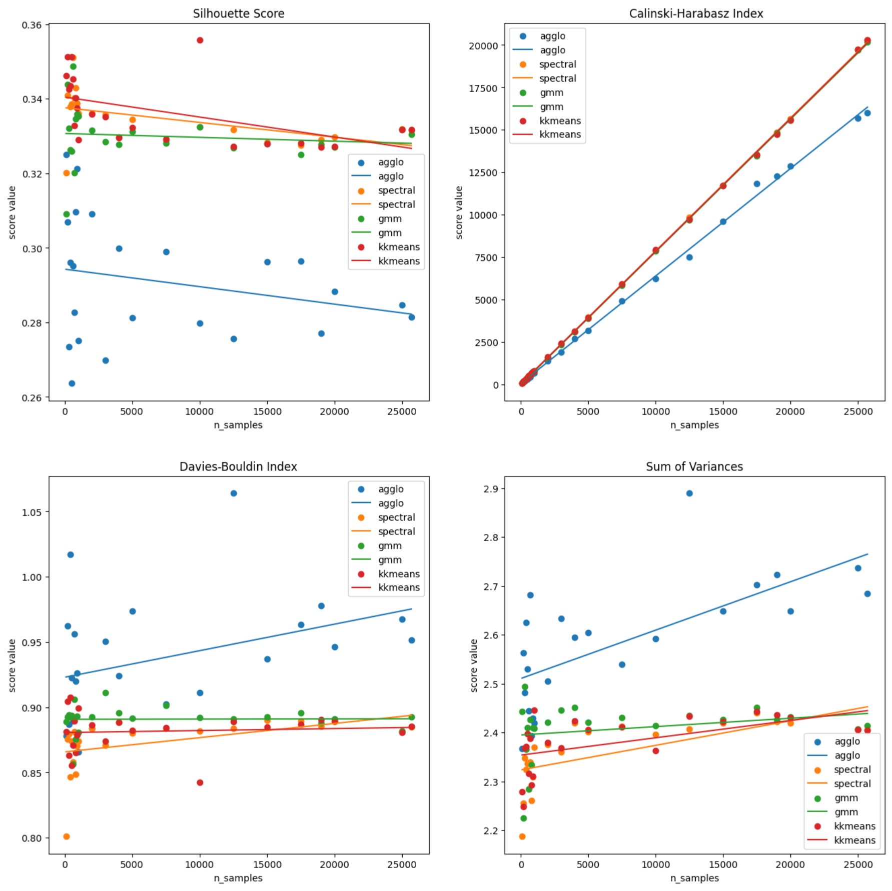

Figure 5.1: Benchmarking with growing data set size

Again, to gain additional insights, table 5.5 and 5.6 display the means, standard deviations and medians of the scores for each algorithm of the results of scenario two.

Figure 5.3 displays the results of clustering data sets of various shapes with the custom implementation as described in 5.1.2.

5.3 Results discussion

Figure 5.1 shows that all algorithms demonstrated a reduction of quality as the size of the data set increased in scenario one. Considering the similar medians and means throughout all algorithms and across all scores, one can argue that the custom Kernel k-means performed comparably to the other algorithms in this scenario.

Scenario two observed the effect of increasing data set dimensions on the performance of the clustering algorithms (figure 5.2). While the Calinski-Harabasz Index, the Davies Bouldin Index and the Sum of Variances, typically sensitive to high dimensionality, indicated a decrease in clustering quality, the Silhouette score increased.

Algorithm	**Silhouette**	**Calinski**	**Davies**	**Sum of Vars**
Agglomerative	-4.70e-07	6.34e-01**	2.04e-06	9.91e-06**
Spectral	-3.96e-07	7.84e-01*	1.09e-06*	5.03e-06
Gaussian Mixture	-1.03e-07*	7.82e-01	1.55e-08**	1.71e-06*
KKMeans	-5.36e-07**	7.83e-01	1.53e-07	3.54e-06

Table 5.1: Gradients algorithm scores with growing data set size

Algorithm	**Silhouette**	**Calinski**	**Davies**	**Sum of Vars**
Agglomerative	2.91e-01 ± 1.63e-02**	4.77e+03 ± 5.52e+03**	9.38e-01 ± 4.65e-02**	2.58e+00 ± 1.28e-01**
Spectral	3.35e-01 ± 6.69e-03	5.86e+03 ± 6.83e+03*	8.74e-01 ± 1.98e-02*	2.36e+00 ± 6.55e-02*
Gaussian Mixture	3.30e-01 ± 7.61e-03	5.82e+03 ± 6.81e+03	8.91e-01 ± 1.06e-02	2.41e+00 ± 5.76e-02
KKMeans	3.36e-01 ± 8.85e-03*	5.85e+03 ± 6.82e+03	8.82e-01 ± 1.50e-02	2.38e+00 ± 5.57e-02

Table 5.2: Means ± Standard Deviations of algorithm scores with growing data set size

While the graphs indicate a decrement in quality with increased data set size and dimension overall, it is worth noting that the Agglomerative, Spectral, and Gaussian Mixture algorithms exhibit small standard deviations across all scores for both scenarios, indicating a consistent and relatively stable performance. The custom Kernel k-means algorithm demonstrates similar stability with growing data set size (Scenario 1). Still, it shows signs of instability when dealing with higher dimensional data (Scenario 2), as evidenced by the larger standard deviations, especially looking at the Sum of Variances in table 5.5. Therefore, while the custom Kernel k-means performs comparably to the other algorithms, its robustness in higher dimensions could be further improved.

5.3.1 Lessons learned and (unexpected) findings

It was unexpected to see the Silhouette score increasing with increased dimensionality, which goes against the typical 'curse of dimensionality' expectation. This may suggest that the Silhouette score might be less sensitive to high dimensionality due to its specific way of calculating cohesion and separation, but this observation warrants further research.

Despite having comparable median and mean scores to the other algorithms, the custom Kernel k-means algorithm showed signs of instability in higher dimensions, as evidenced by a larger standard deviation. This behaviour was not expected and provided insights into potential areas of improvement for the custom algorithm.

Algorithm	Silhouette	Calinski	Davies	Sum of Vars
Agglomerative	2.88e-01**	1.90e+03**	9.37e-01**	2.59e+00**
Spectral	3.34e-01*	2.40e+03	8.81e-01*	2.38e+00*
Gaussian Mixture	3.28e-01	2.31e+03	8.92e-01	2.42e+00
KKMeans	3.33e-01	2.41e+03*	8.85e-01	2.40e+00

Table 5.3: Medians of algorithm scores with growing data set size

Algorithm	Silhouette	Calinski	Davies	Sum of Vars
Agglomerative	1.19e-04**	-6.70e-01	-1.51e-04**	2.68e-01
Spectral	1.41e-04	-4.16e-01	-2.07e-04	2.66e-01*
Gaussian Mixture	1.19e-04**	-7.41e-01**	-2.16e-04	2.68e-01
KKMeans	1.47e-04*	-1.74e-01*	-2.46e-04*	4.80e-01**

Table 5.4: Gradients algorithm scores with growing data set dimension

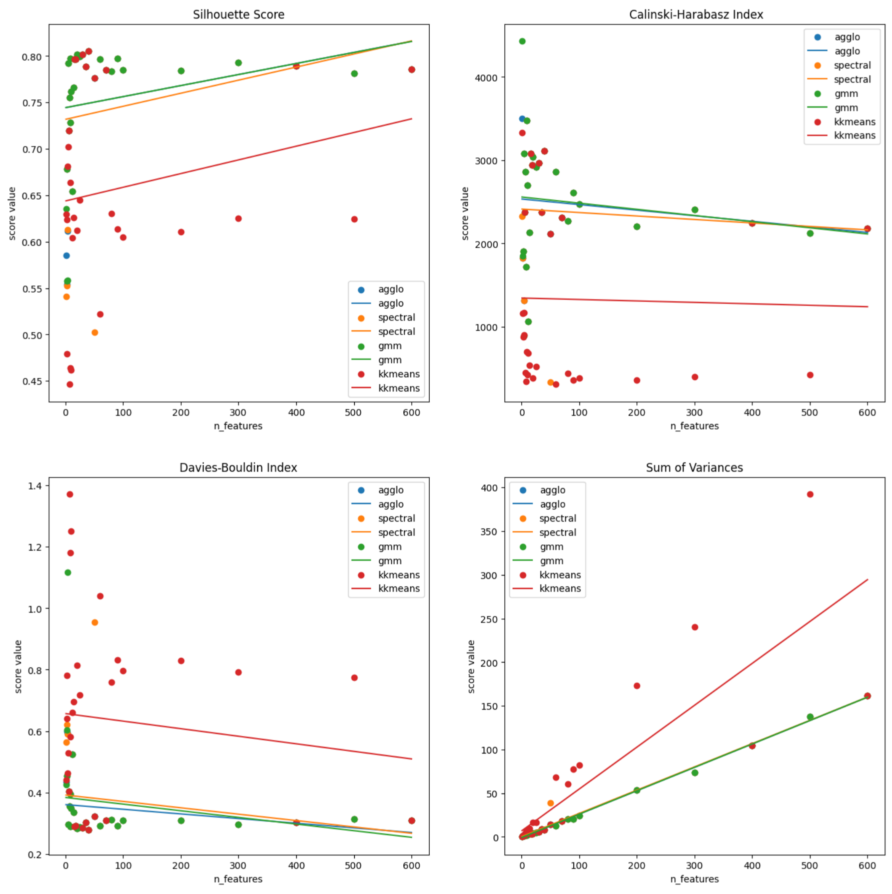

Figure 5.2: Benchmarking with growing data set dimension

Algorithm	Silhouette	Calinski	Davies	Sum of Vars
Agglomerative	7.55e-01 ± 6.85e-02*	2.47e+03 ± 5.84e+02	3.47e-01 ± 8.89e-02*	2.34e+01 ± 4.16e+01*
Spectral	7.44e-01 ± 8.58e-02	2.38e+03 ± 6.69e+02	3.74e-01 ± 1.47e-01	2.42e+01 ± 4.16e+01
Gaussian Mixture	7.55e-01 ± 6.91e-02*	2.49e+03 ± 6.89e+02*	3.65e-01 ± 1.61e-01	2.34e+01 ± 4.16e+01*
KKMeans	6.57e-01 ± 1.11e-01**	1.33e+03 ± 1.07e+03**	6.35e-01 ± 3.08e-01**	5.00e+01 ± 8.83e+01**

Table 5.5: Means ± Standard Deviations of algorithm scores with growing data set dimension

Algorithm	Silhouette	Calinski	Davies	Sum of Vars
Agglomerative	7.85e-01*	2.39e+03	3.10e-01*	5.54e+00*
Spectral	7.85e-01*	2.38e+03**	3.10e-01*	5.54e+00*
GaussianMixture	7.85e-01*	2.39e+03	3.10e-01*	5.54e+00*
KKMeans	6.30e-01**	7.88e+02*	6.51e-01**	8.18e+00**

Table 5.6: Medians of algorithm scores with growing data set dimension

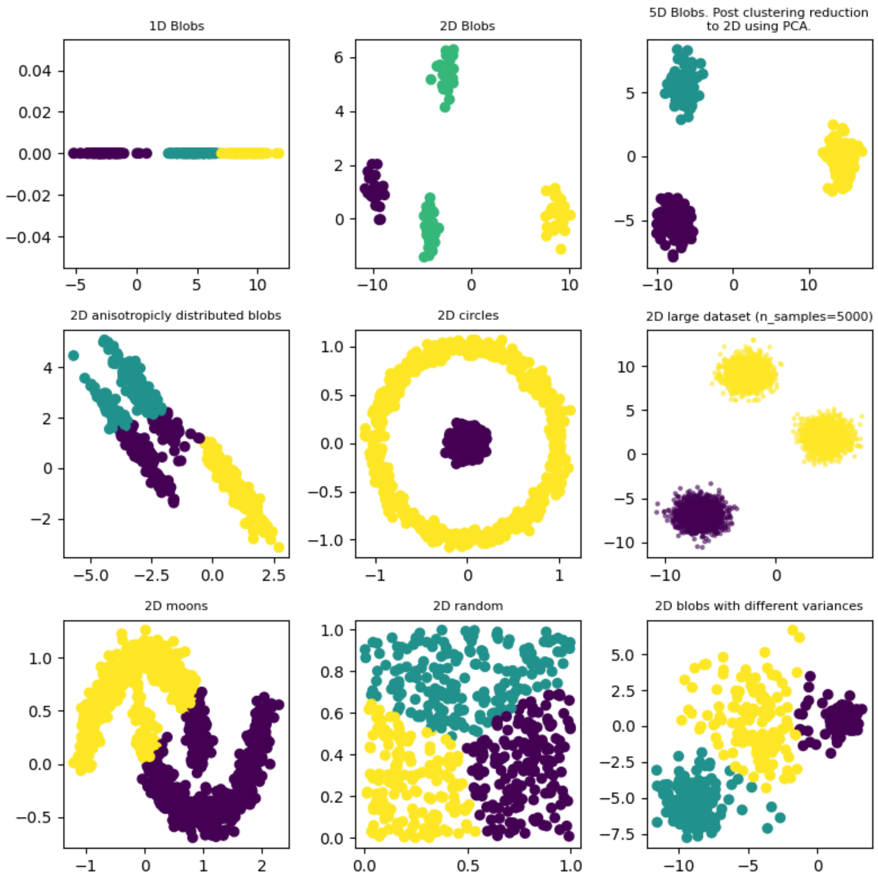

Figure 5.3: Kernel k-means on nine differently shaped data sets

6 Conclusion and Future Work

6.1 Summary of findings

The Kernel k-means implementation displayed comparable performance in low-dimensional small and large data sets.

However, when dealing with high-dimensional data ($n_{features} \geq 100$), it showed a noticeable dip in performance compared to other algorithms, such as Agglomerative Clustering and Spectral Clustering. This decrease can be largely attributed to the curse of dimensionality, which refers to the diminishing effectiveness of distance-based methods in high-dimensional spaces. Therefore, while the implementation is useful, its applicability might be limited in cases dealing with high-dimensional data.

6.2 Outline of future work

To address the limitations and further improve the performance of the implementation as well as the benchmarking, several advancements should be considered for future work:

1. Optimization for High-dimensional Data: Using dimensionality reduction techniques like PCA or feature selection methods can make the algorithm more efficient and effective in dealing with high-dimensional data.

2. Compare Kernel Functions: Extend the implementation by adding more kernel functions and compare their performance.

3. Performance Metrics: The computation time and resource intensity can be considered metrics when evaluating these clustering algorithms' performance, especially when handling large, high-dimensional data sets. Although it is critical to have accurate and meaningful clusters, it is equally important that the algorithm performs in a reasonable amount of time and doesn't require an excessive amount of computational resources. Given that the experiments have been conducted on a personal computer, incorporating cloud-based solutions enables a more consistent benchmarking and might produce different results, once those performance metrics are incorporated.

4. Different data sets: Extending the tests to other types of data sets, such as those with more complex data structures and real-world data with inherent noise, may produce valuable insights into the robustness of the implementation.

Despite its limitations, this work should provide a solid foundation for the further development and optimization of Kernel k-means algorithm.

Bibliography

[Aiz64] Mark A Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[AP22] Amali Amali and Gatot Tri Pranoto. Manhattan, euclidean and chebyshev methods in k-means algorithm for village status grouping in aceh province. *Journal of Applied Intelligent System*, 7:211–222, 12 2022.

[Aro50] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337, 5 1950.

[AV07] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. pages 144–152. Association for Computing Machinery, 1992.

[CH74] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods*, 3:1–27, 1974.

[CJHJ11] Radha Chitta, Rong Jin, Timothy C. Havens, and Anil K. Jain. Approximate kernel k-means. pages 895–903. ACM, 8 2011.

[CR19] Daniele Calandriello and Lorenzo Rosasco. Statistical and computational trade-offs in kernel k-means. 8 2019.

[CZP09] Ning Chen, Hongyi Zhang, and Jiexin Pu. A novel kernel self-organizing map algorithm for clustering. pages 2978–2982. IEEE, 8 2009.

[DB79] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:224–227, 4 1979.

[DGK04] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. pages 551–556. ACM, 8 2004.

[DGK05] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical report, University of Texas Dept. of Computer Science, 2005.

[Fab94] Vance Faber. Clustering and the continuous k-means algorithm. 1994.

[FCMR08] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41:176–190, 1 2008.

[Fre03] Ivar Fredholm. Sur une classe d'équations fonctionnelles. *Acta Mathematica*, 27:365–390, 1903.

[Gir02] M. Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13:780–784, 5 2002.

[Hil89] David Hilbert. *Grundzüge einer allgemeinen Theorie der linearen Integralgleichungen.* 1989.

[KLLL05] Dae-Won Kim, Ki Young Lee, Doheon Lee, and Kwang H. Lee. Evaluation of the performance of clustering algorithms in kernel-induced feature space. *Pattern Recognition*, 38:607–611, 4 2005.

[LFYZ08] Jinjiang Li, Hui Fan, Da Yuan, and Caiming Zhang. Kernel function clustering based on ant colony algorithm. pages 645–649. IEEE, 2008.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning.* MIT Press, 2 edition, 2018.

[M.T19] Amit M.Thombre. Effect of outlier removal on grid search and distance between two classes (the techniques to find hyperparameter, sigma of support vector machine). pages 1–8. IEEE, 12 2019.

[PCDX20] Debolina Paul, Saptarshi Chakraborty, Swagatam Das, and Jason Q Xu. Kernel k-means, by all means: Algorithms and strong consistency. *CoRR*, abs/2011.06461, 2020.

[Rou87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987.

[SD19] Caner Savas and Fabio Dovis. The impact of different kernel functions on the performance of scintillation detection based on support vector machines. *Sensors*, 19:5219, 11 2019.

[SK21] Meshal Shutaywi and Nezamoddin N. Kachouie. Silhouette analysis for performance evaluation in machine learning with applications to clustering. *Entropy*, 23:759, 6 2021.

[SS18] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels.* The MIT Press, 2018.

[SSM98] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[TL09] G.F. Tzortzis and A.C. Likas. The global kernel k-means algorithm for clustering in feature space. *IEEE Transactions on Neural Networks*, 20:1181–1194, 7 2009.

[YTL$^+$12] Shi Yu, Leon-Charles Tranchevent, Xinhai Liu, W. Glanzel, J. A. K. Suykens, B. De Moor, and Y. Moreau. Optimized data fusion for kernel k-means clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34:1031–1039, 5 2012.

[ZR] Rong Zhang and A.I. Rudnicky. A large scale clustering scheme for kernel k-means. pages 289–292. IEEE Comput. Soc.