Support Vector Clustering: Theory and application to the wine dataset

Gabriel Buffet **Nils Cazemier** **Meilame Tayebjee**

Abstract

Support Vector Clustering, designed by Ben-Hur et al. [2001], is a *non-parametric* (*i.e.* that does not require a target number of clusters) clustering algorithm, that can yield clusters of arbitrary shapes. In this work, we remind some background on clustering, the main key features of the algorithm, as well as other classic clustering algorithms' that we will use as baselines. We define metrics, re-implement SVC and conduct experiments on both synthetic and real data, eventually aiming at having a clear overview of the strengths and weaknesses of the algorithm, *in concreto*.

1 Background

1.1 Introduction

Clustering Clustering in Machine Learning consists in splitting a given datasets of points into different *groups* (or clusters) - regrouping data points *that look the same* together, and two individuals of different groups should be *as different as possible*. The clustering algorithms are said to be part of the *unsupervised learning* realm, as the data we use are not labeled.

To decide if a data point *look like* another, we use a distance metric, the most classic one being the Euclidean norm on $\mathbb{R}^d$. More specific distances can be brought about, as well as use of kernels (Filippone et al. [2008]) or graphs (the weight of an edge being the value of a similarity function between two nodes) (Liu and Barahona [2020]).

The range of applications is huge. We mention for instance: community detection on social media (Fraisier et al. [2018] use Twitter data to group users that retweet each other a lot, showing the polarization of the public debate), biomedical (Zhang et al. [2023] propose a new clustering algorithm to split patients with different types of gastric cancers) or computer vision (Dhanachandra et al. [2015] use K-Means for image segmentation - that is to locate and target objects and boundaries).

Support Vector Machines Before the take-off rise of computational resources that enabled the use of deep neural network models to cope with complex data-oriented issues, **Support Vector** based algorithms have proven to be very effective for regression or classification tasks, being based on statistical learning frameworks or VC theory proposed by Vapnik and Chervonenkis (see Cortes and Vapnik [1995]. These performances have been enhanced by the use of non-linear kernels to get rid of non-linearities. This supervised learning technique can be adapted to unsupervised learning techniques such as clustering.

Notations Notations are quite simple. We assume we have a dataset of n points $\mathcal{X} = (X_i)_{i=1...n}$ in $\mathbb{R}^d$. We call $\mathcal{C} = (C_1, ..., C_k)$ a (hard) **clustering** (or a **partition**) with k clusters, if $\forall i,\ C_i \subset \mathcal{X}$, for $i \neq j$, $C_i \cap C_j = \emptyset$ and $\bigcup_{i=1...k} C_i = \mathcal{X}$. $|\cdot|$ classicly denotes the cardinality of a set. The *entropy function* will also be useful $H(\mathcal{C}) = -\sum_{C\in\mathcal{C}} P(C)\log(P(C))$, where $P(\cdot) = \frac{|\cdot|}{n}$.

For a given cluster C, we define its *centroid* by $\mu(C) = \frac{1}{|C|}\sum_{i/X_i\in C} X_i$.

Our contribution Our work's main focus is the Support Vector Clustering (SVC) algorithm, introduced by Ben-Hur et al. [2001]. We remind some important facts about it in 2, as well as other classic clustering algorithms (1.2) that we will use as baselines on our experiments on synthetic (3.1) and real (3.2) datasets. We end up drawing the main benefits and limitations of SVC.

Code We re-implemented SVC on Python, and all of our experiments are reproducible and available here : `https://github.com/meilame-tayebjee/Support_Vector_Clustering`. Supplemental experiments (on the Iris dataset for instance) are also available.

1.2 Baseline algorithms

K-Means K-Means is a special case of the Expectation-Maximization algorithm (Dempster et al. [1977]), alternating between computation of *centroids* (*M step*) and re-labelling of the points (*E step*). Algorithm 2 (see Appendix) - designed by Lloyd [1982] - details the process. The aim of K-Means is to yield a clustering that minimizes - as much as possible, as it is a NP-hard problem - the following quantity, called the *intra-class inertia*:

$$\mathrm{W} = \frac{1}{n}\sum_{k=1}^{K}\sum_{X_i \in C_k} \|X_i - \mu_k\|^2$$

where K is the target number of clusters (hyperparameter) and $\mu_k = \frac{1}{|C_k|}\sum_{X_i \in C_k} X_i$ the empirical mean of the observations in cluster C_k (the centroid).

It is **parametric** in the sense that K, the target number of clusters, is an input.

K-Means - Highlights

- K-Means yields **convex**, approx. **same size** and **same shape** clusters (basically, "*soccer balls*"). It quite heavily depends on the **initialisation**.
- Complexity: $O(nK)$
- Parametric : needs a target K, that needs to be fine-tuned.

Hierarchical Clustering Hierarchical (or **Agglomerative**) Clustering starts by considering each data point as a separate cluster and then iteratively merges the two closest clusters (the distance between clusters - the *linkage criterion* - has to be fine-tuned and has an impact on the clustering - see Appendix), until we only have one cluster. The result is a tree-like structure, called a **dendrogram**.

Eventually, a target number of clusters K has to be chosen to yield the final clustering (we need to know where to cut the tree), even though it is not needed in the training and the dendrogram enables to have a better view to fine-tune that K. Hence, we will say here that it is **parametric**.

Hierarchical Clustering - Highlights

- Hierarchical Clustering is able to yield clusters of **arbitrary shape**, **arbitrary size**, making it versatile in capturing the inherent structure of the data.
- Complexity: $O(n^3)$
- Sensitivity to the **linkage criterion** (e.g., complete, single, average...).
- Parametric : needs a target K, that needs to be fine-tuned.

DBSCAN Density-based spatial clustering of applications with noise (DBSCAN) (Ester et al. [1996]) is a **density-based clustering**, that is looks for areas with higher density of points to define a cluster. The pseudo-code is provided in Algorithm 4 (see Appendix).

Basically, we start with an arbitrary point and retrieve the ϵ-neighborhood (all the data points that lie within a distance ϵ of the considered point, ϵ being a parameter). If this ϵ-neighborhood contains at least *minPts* points (*minPts* being the other parameter), this neighborhood is said to be *dense*, a

cluster starts and we iterate through the neighbors, adding their ϵ-neighborhood provided they have enough points too. If the the ϵ-neighborhood do not contain enough points, the point is classified as a noise (but still could be part of a cluster later if it is *reached* from another point), and we move to another point.

At the end of the day, each point is classified as : a core point (dense neighborhood), a border point (no dense neighborhood but in the neighborhood of a core point), or noise. This is intuitively quite close to what is done in SVC (see 2).

DBSCAN - Highlights

- DBSCAN is able to yield clusters of **arbitrary shape**, **arbitrary size**. It is particularly robust to **outliers** and **noise**.
- Complexity: $O(n^2)$, $O(nlogn)$ if we use an indexing structure that limits the number of neighborhood queries.
- **Non-Parametric** in the sense that it doesn't comply to a target K, just like SVC.
- However, very **sensitive** to the hyper-parameters ϵ and *minPts* that needs to be carefully fine-tuned not to obtain aberrant results.

1.3 Metrics

Silhouette Score:

$$S(\mathcal{C}) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|C_k|}\sum_{i\in C_k} s(i)$$

where $s(i) = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}$, $a(i) = \frac{1}{|C_i|-1}\sum_{X_j\in C_i, j\neq i}\|X_i - X_j\|$ (the mean distance between X_i and all other data points in the same cluster) and $b(i) = \min_{k\neq i}\frac{1}{|C_k|}\sum_{X_j\in C_k}\|X_i - X_j\|$ (the smallest mean distance of X_i to all points in any other cluster). The Silhouette Score measures how well-separated clusters are, ranging from -1 to 1, where higher values indicate better-defined clusters.

Normalized Mutual Information (NMI):

$$NMI(\mathcal{C},\mathcal{P}) = \frac{2\cdot I(\mathcal{C},\mathcal{P})}{H(\mathcal{C}) + H(\mathcal{P})}$$

where $I(\mathcal{C},\mathcal{P}) = \sum_{C\in\mathcal{C}}\sum_{p\in\mathcal{P}} P(C\cap p)\log\left(\frac{P(C\cap p)}{P(C)P(p)}\right)$, H and P defined in 1.1. NMI measures the mutual dependence between two clustering results, normalized by the entropy of the individual clusters. Typically, $\mathcal{P}$ is a reference partition, or a ground truth: higher NMI indicates closeness to it.

Davies-Bouldin Index (DBI):

$$DBI(\mathcal{C}) = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{\text{avg\_distance}(C_i) + \text{avg\_distance}(C_j)}{\|\mu(C_i) - \mu(C_j)\|}\right)$$

where $\text{avg\_distance}(C) = \frac{1}{|C|}\sum_{X_i\in C}\|X_i - \mu(C)\|$. The Davies-Bouldin Index quantifies the compactness and separation of clusters, with lower values (close to 0) indicating better clustering.

Rand Index:

$$\text{RandIndex} = \frac{TP + TN}{TP + TN + FP + FN}$$

The Rand Index measures the similarity between true and predicted clusterings, considering true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Once again, it is used to compare a clustering $\mathcal{C}$ to another *reference* partition $\mathcal{P}$. It is comprised in $[0, 1]$, and the higher, the better.

Homogeneity, Completeness, and V-measure:

$$\text{Homogeneity} = 1 - \frac{H(\mathcal{P}|\mathcal{C})}{H(\mathcal{P})} \qquad \text{Completeness} = 1 - \frac{H(\mathcal{C}|\mathcal{P})}{H(\mathcal{C})}$$

where $H(\mathcal{P}|\mathcal{C}) = -\sum_{C\in\mathcal{C}}\sum_{p\in\mathcal{P}} P(C\cap p)\log\left(\frac{P(C\cap p)}{P(C)}\right)$, $\mathcal{P}$ being a reference partition. Homogeneity measures how well each cluster contains only members of a single class, Completeness measures how well all members of a given class are assigned to the same cluster. To be noted that their harmonic mean is exactly the NMI, defined above (also called the V-measure).

Fowlkes-Mallows Index:

$$\text{FMI} = \frac{\text{TP}}{\sqrt{(\text{TP}+\text{FP})\cdot(\text{TP}+\text{FN})}}$$

The Fowlkes-Mallows Index assesses the similarity between true and predicted clusterings, focusing on true positives, false positives, and false negatives. It is comprised in $[0, 1]$, and the higher, the better.

2 Theory of Support Vector Clustering

2.1 SVC as a Constrained Optimization Problem

Let $\{x_i\}_{1\leq i\leq N} \subseteq \mathcal{X} \subseteq \mathbb{R}^d$ be our dataset, $\psi : \mathbb{R}^d \to \mathbb{R}^{d'}$ the feature map and K the corresponding kernel (Gaussian in all our applications). **We want to found - in the feature space - the minimum radius circle enclosing all the data points**, allowing for some outliers as in Support Vector Machines.

We can formulate this problem as a constrained optimization problem :

$$\begin{aligned} \min R, \quad \text{s.t. } & \forall i,\ \|\psi(x_i) - \boldsymbol{a}\|^2 \leq R^2 + \xi_i \\ & \forall i,\ \xi_i \geq 0 \end{aligned} \tag{1}$$

It enables us to introduce the Lagrangian for this problem, although we choose to add a penalty C to control the ξ_i as following:

$$\mathcal{L}(R, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\beta}) = R^2\left(1 - \sum_{i=1}^{n}\beta_i\right) + \sum_{i=1}^{n}(\beta_i + \mu_i - C)\,\xi_i + \sum_{i=1}^{n}\|\psi(x_i) - \boldsymbol{a}\|^2\beta_i$$

where $\beta_i \geq 0$ and $\mu_i \geq 0$ are the Lagrangian multipliers.

We derive the dual problem:

$$\begin{aligned} \max_{\boldsymbol{\beta}\in\mathbb{R}} \left[\sum_j \beta_j - \sum_{i,j}\beta_i\beta_j K(x_i, x_j)\right] \quad & \text{s.t. } \forall j, 0 \leq \beta_j \leq C \\ & \sum_j \beta_j = 1 \end{aligned} \tag{2}$$

using the kernel trick ($\psi(x_i)\cdot\psi(x_j) = K(x_i, x_j)$) and the fact that $K(x_i, x_i) = 1$.

The KKT conditions are:

$$\xi_j\mu_j = 0 \tag{3}$$

$$(R^2 + \xi_j - \|\phi(x_j) - \mathbf{a}\|^2)\beta_j = 0 \tag{4}$$

2.2 Bounded Support Vector and Support Vector

The KKT conditions have us defining two types of vectors x_i.

If both $\xi_i > 0$ and $\beta_i > 0$, then $R < \|\psi(x_i) - \boldsymbol{a}\|$ which means that x_i is **out of the encapsulating sphere**. Moreover this implies that $\mu_i = 0$ and $\beta_i = C$ (based on first order conditions), and these "out-of-the-border" vectors will be referred as ***Bounded Support Vector*** **(BSV)**. They are considered as **outliers**.

On another hand, if $0 < \beta_i < C$ then $\mu_i \neq 0$ and necessarily: $\xi_i = 0$. The last KKT condition leads to : $R^2 = \|\psi(x_i) - \boldsymbol{a}\|$: such a vector is located on the boundary of the encapsulating sphere and will be referred as a ***Support Vector*** (SV).

2.3 The Radius function

We are actually going to work with a Gaussian kernel:

$$K(x_i, x_j) = e^{-q\|x_i - x_j\|^2}$$

where q is a width parameter to control.

For any vector x in $\mathcal{X}$, we can define its radius $R(x)$ in the projected space as following:

$$R^2(x) = \|\psi(x) - \boldsymbol{a}\|^2 = K(x,x) - 2\sum_{i=1}^{n} \beta_i K(x, x_i) + \sum_{i,j} \beta_i \beta_j K(x_i, x_j) \tag{5}$$

The identification of the support vectors leads us to define the radius R as: $R = \{R(x), x \in \mathcal{SV}\}$, where $\mathcal{SV}$ the subset of $\mathcal{X}$ containing the support vectors, $\{x \mid R(x) = R\}$ defining a notion of border in the data space, the BSVs remaining out of this border.

2.4 Cluster Assignment

The cluster assignment is finally defined as following: we compute an adjacency matrix where two vectors are connected if the transformation of the segment in the data space is contained in the encapsulating sphere - if so, they are likely to be part of the same cluster. Hence, the clusters are defined as the connected components induced by the adjacency matrix. Algorithm 1 shows the whole pseudo-code of SVC.

Algorithm 1: Support Vector Clustering

Input: dataset $\{x_i\}_{1 \leq i \leq N}$, hyperparameters p and q
$\beta \leftarrow$ Solutions of the dual optimization problem (2)
Compute the A matrix:

$$A_{ij} = \begin{cases} 1 & \text{if for all } y \text{ on the line segment connecting } x_i \text{ and } x_j,\ R(y) \leq R, \text{where R defined in (5)} \\ 0 & \text{otherwise.} \end{cases}$$

return Connected components of A

This procedure leaves the BSVw unclassified: here, we will leave them unclassfied, but they can also be assigned to the closest cluster.

2.5 Discussion

Parameters SVC has two parameters: 1) $p = \frac{1}{NC}$, which is an upper bound on the fraction of BSVs, as detailed by Ben-Hur et al. [2001], and 2) q, the width of the Gaussian Kernel.

When increasing q, the boundary fits more to the data, the number of SVs and **the number of clusters increase**. If the SVs are excessively numerous, one should increase p, allowing consequently some of these SVs to become BSVs and smoothening the border.

Complexity Benchmarks indicate convergence after roughly $O(N^2)$ kernel evaluations. The overall complexity is $O(N^2 d)$, being highly subject to the curse of dimensionality.

The computation of the A matrix is the main pain point, especially memory-wise. In practice, to verify the condition on the whole segment line, we use 10 equally spaced points between x_i and x_j.

We used the library *CVXPY* (Agrawal et al. [2018]) to solve the dual optimization problem. We used the library *Numba* on Python, which accelerated greatly the computation time. To retrieve the connected components given A, we used the library *networkx*.

3 Experiments and results

3.1 Synthetic datasets

In order to verify our implementation and to have a sense on how one should tune p and q, we first applied the algorithm to synthetic data.

3.1.1 Blobs

We start with a quite simple synthetic dataset, `make_blobs` from the *scikit-learn* library, that generates three convex ball-shaped clusters. We generate 200 points and use `cluster_std = 0.45`.

(a) A perfect clustering, after careful tuning

(b) An example of aberrant results

Figure 1: SVC clustering on the Blobs

Figure 1 shows the results. **We highlight here the needed careful tuning of the hyper-parameters p and q.** All of our clustering methods perform perfectly here, and we do not display the plots for the baselines for the sake of conciseness. They are available in the code.

3.1.2 Two moons

`make_moons`, again from the *scikit-learn* library, is a bit more challenging dataset, with non-convex shaped clusters. We generate 100 points, with `noise = 0.1`.

(a) SVC: red circles are the support vectors

(b) DBSCAN

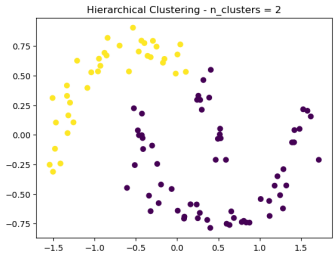
Figure 2: Success of non-parametric methods

Figure 2 shows how non-parametric methods fare better than parametric ones here (Figure 3). Especially, we highlight here that **SVC is able to generate clusters of arbitrary shape**, which is one of its main advantages.

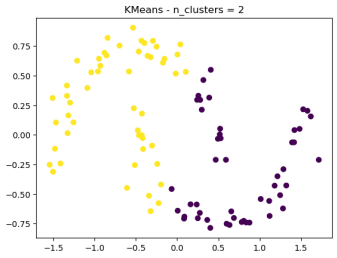
3.2 Real dataset

The dataset gathers the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Scaling our data and **using the first two principal components** to project our data we get the cluster representation shown on figure 4.



(a) Hierarchical clustering: *average* linkage



(b) K-Means

Figure 3: Failure of parametric methods

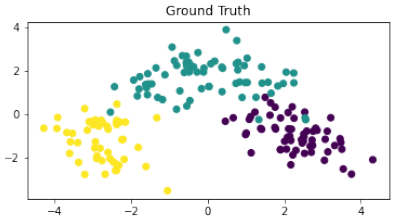


Figure 4: True clusters

We used the following parameters for our baselines:

- K-Means with a target of $K = 3$ clusters
- Hierarchical Clustering with a target of 3 clusters using the linkage `ward` since it gives the best clustering
- DBSCAN with $\epsilon = 2.4$ and `min_samples`=17 (manually tuned parameters)

To get the best values for p and q, we used a grid search technique plotting the results for each pair of values and updating the grid. We iterated until we obtained a satisfying clustering, that is to say a clustering giving three main clusters and compared the scores obtained to pick the parameters.

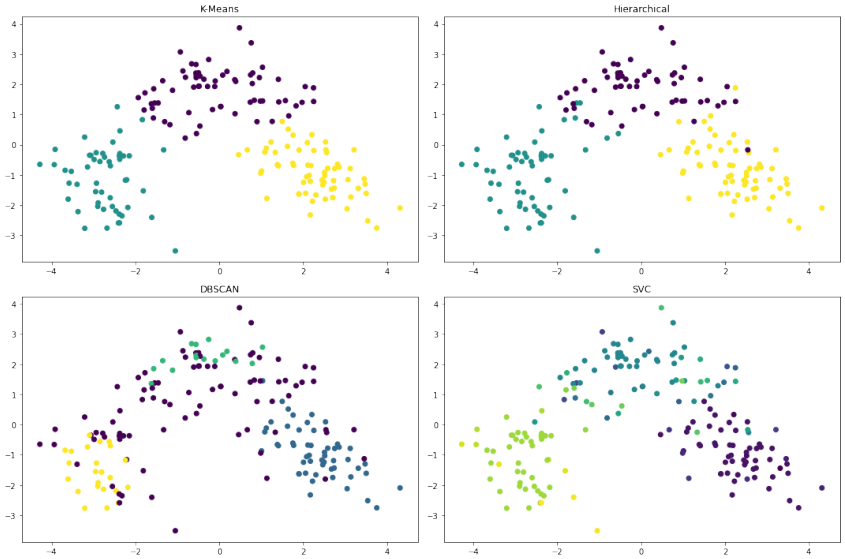


Figure 5: Result of clustering for the considered algorithms. The result for DBSCAN has been obtained using `eps=2.4` and `min_samples = 17`, the result for SVC was obtained with $p = 10^{-6}$ and $q = 0.65$

Figure 5 shows the result of each algorithm. The two parametric algorithms gives visually similar result with the three clusters separated fairly well. We see that the DBSCAN algorithm actually yielded four clusters, the three main ones correlated with the true clusters. Finally, the SVC algorithm gave a pretty good result here: we recognize the three main true clusters with some outliers (singletons).

Table 1 sums up the performance of the different algorithms using a first set of metrics. The silhouette metric indicates that here **both parametric algorithms** do much better than the non-parametric algorithms. It also establishes that, with regard to this metric, DBSCAN clusterizes in a better way. But this is understandable since here SVC left many singletons out of the big clusters which lowers its score. Analyzing the NMI score, we see that while K-Means outperforms the other algorithms, SVC holds up against DBSCAN. Our clustering algorithm even beats the K-Means when using the Davis Bouldin score, which is really satisfactory but may be caused by the presence of the singletons. Finally, the Rand index score shows that SVC provides a more significant clustering that the one obtained with DBSCAN, although the parametric algorithms are better according to this metric.

Table 1: Silhouette, Normalized Mutual Information, Davies Bouldin and Rand Index scores

	Silhouette	NMI	DBI	Rand
K-Means	0.28486	0.87589	1.38919	0.95429
Hierarchical	0.27744	0.78647	1.41859	0.90649
DBSCAN	0.08419	0.56584	2.23228	0.757
SVC	-0.09661	0.56901	0.64575	0.8095

To continue our analysis and understand in what aspect of the resulting clusterings differ, we gathered more scores in the table 2. First, SVC achieves a **perfect homogeneity** score, beating the parametric algorithms which can be interpreted in the following way: it prefers **to further split the clusters rather than to wrongly label the data points** as we can observe with the presence of the singletons. It's both good and bad at the same time since those singletons are the very cause of the low completeness score for SVC compared to the other algorithms. But both these aspects are important to get a good clustering, and they are merged in the NMI score since it corresponds to the harmonic mean of those two scores.

Combining the previous insight with the one given with the Fowlkes-Mallows score, we can see that SVC produced a fairly decent clustering while being non-parametric **slightly outperforming the other non-parametric algorithm considered here**- DBSCAN - the parametric algorithms producing better results with this dataset.

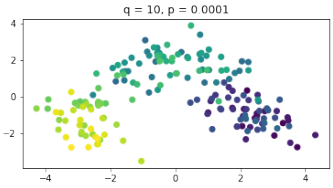
Table 2: Homogeneity, completeness and Fowlkes-Mallows scores

	Homogeneity	Completeness	Fowlkes-Mallows
K-Means	0.87884	0.87296	0.93191
Hierarchical	0.79043	0.78254	0.86021
DBSCAN	0.59374	0.54044	0.64326
SVC	1.0	0.39763	0.66055

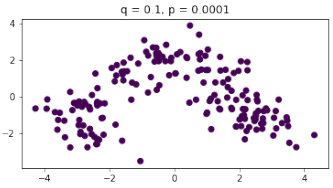
Sensitivity to hyper-parameters However, Figure 6 shows once again how sensitive SVC is to the hyper-parameter fine tuning, and how small changes can affect drastically the outcome. In Figure 7 we illustrate this instability. We can see that, for a fixed value of p, the result of the algorithm varies a lot with q. Indeed, there are not a lot of values that achieve both good NMI and Fowlkes-Mallows scores while giving a relatively low total number of clusters and having three big clusters.

Conclusion

To conclude, we summarize the main key points of SVC that we saw over these experiments:



(a) Case with N clusters



(b) Case with only 1 cluster

Figure 6: Extreme cases

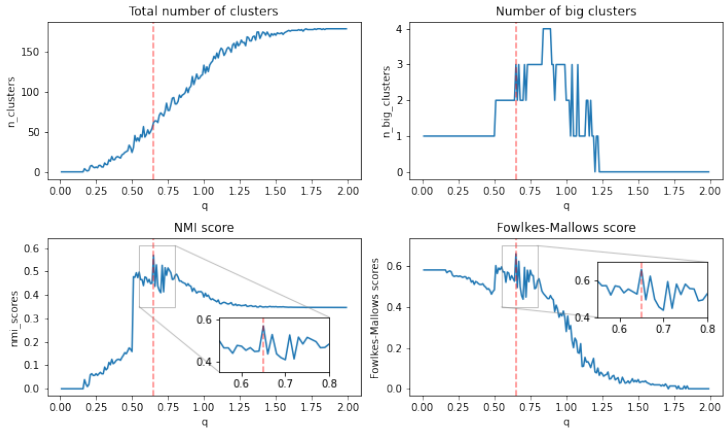


Figure 7: Variation of the NMI and the Fowlkes-Mallows scores, of the number of big clusters (clusters with at least 5 points) and of the total number of clusters with the parameter q

SVC - Highlights

- SVC is able to yield clusters of **arbitrary shape**, **arbitrary size**. It is **non-parametric** in the sense that it doesn't comply to a target K
- Its two parameters, p and q, enable to produce various type of clusterings, and to adapt to noise and overlaps (**increase q to split more the clusters, increase p to *allow* for more outliers**).
- On the other hand, high sensitivity to changes in these parameters, that require a careful fine-tuning.
- High complexity, be it in run-time or in memory. *Fast SVC* (Pham et al. [2016]) and *Scalable SVC* (Pham et al. [2017]) aim at solving these issues.
- Does not fare well in high-dimension, and requires the use of dimensionality reduction.

Next steps could include:

- Experiments on more "arbitrary-shaped" clusters, that might highlight in a better way the efficiency of SVC
- Experiments on high-dimensional data ($d > 50$) and analysis of the impact of a dimensionality increase, both on clustering quality and on run-time
- Implement the Fast and Scalable versions and produce a comprehensive run-time comparison and analysis

Appendix

Pseudo codes of the baseline algorithms (1.2)

Algorithm 2: K-Means

Data: Data points $\mathcal{X}$, Target number of clusters K
Result: Clustering

Initialize centroids randomly;
for $iteration \leftarrow 1$ **to** $maxIterations$ **do**
  Assign each data point to the nearest centroid:

$$\text{cluster}[X_i] = \arg\min_j \|X_i - \text{centroid}[j]\|^2$$

  Update centroids based on the mean of assigned data points:

$$\text{centroid}[j] = \frac{1}{|\text{cluster}[X_j]|} \sum_{i \in C_j} X_i$$

  if *centroids have converged* **then**
    break;
  end
end

Algorithm 3: Hierarchical Clustering

Data: Data points $\mathcal{X}$
Result: Dendrogram or Clusters

while *number of clusters is not 1* **do**
  foreach *pair of clusters* C_i, C_j **do**
    Compute the pairwise distance between C_i and C_j;
  end
  Merge the closest pair of clusters C_a and C_b;
  Update the distance matrix;
end

Linkage in Hierarchical Clustering 1.2 In Hierarchical Clustering, each linkage method influences clustering differently. Single linkage ($d(C_i, C_j) = \min_{x \in C_i, y \in C_j} \text{distance}(x, y)$) tends to form elongated clusters sensitive to outliers. Complete linkage ($d(C_i, C_j) = \max_{x \in C_i, y \in C_j} \text{distance}(x, y)$) forms compact clusters less sensitive to outliers. Average linkage ($d(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \text{distance}(x, y)$) balances between single and complete linkage, creating more balanced clusters. Centroid linkage ($d(C_i, C_j) = \text{distance}(\text{centroid}(C_i), \text{centroid}(C_j))$) forms clusters based on centroids and is sensitive to outliers. Ward's linkage ($d(C_i, C_j) = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} \cdot \text{distance}(\text{centroid}(C_i), \text{centroid}(C_j))$) minimizes variance within clusters, producing compact clusters sensitive to outliers.

Algorithm 4: DBSCAN Algorithm

Data: Dataset D, Epsilon ε, MinPts
Result: Clusters

foreach *point p in D* **do**
  if *point p is not visited* **then**
    Mark point p as visited;
    $N \leftarrow$ regionQuery(p, ε);
    if $|N| <$ *MinPts* **then**
      Mark point p as noise;
    end
    else
      Create a new cluster C;
      ExpandCluster(p, N, C, ε, MinPts);
    end
  end
end

Function *ExpandCluster(p, N, C, ε, MinPts)*:
  Add point p to cluster C;
  foreach *neighbor q in N* **do**
    if *neighbor q is not visited* **then**
      Mark neighbor q as visited;
      $N' \leftarrow$ regionQuery(q, ε);
      if $|N'| \geq$ *MinPts* **then**
        $N \leftarrow N \cup N'$;
      end
    end
    if *neighbor q is not yet a member of any cluster* **then**
      Add neighbor q to cluster C;
    end
  end

Function *regionQuery(p, ε)*:
  return *all points in D within distance ε from point p*;

References

A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support Vector Clustering. *Journal of Machine Learning Research 2*, December 2001.

C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, sep 1995. ISSN 0885-6125.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

N. Dhanachandra, K. Manglem, and Y. J. Chanu. Image segmentation using k -means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India.

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, page 226–231. AAAI Press, 1996.

M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176–190, 2008. ISSN 0031-3203.

O. Fraisier, G. Cabanac, Y. Pitarch, R. Besançon, and M. Boughanem. #Élysée2017fr: The 2017 french presidential campaign on twitter. *Proceedings of the International AAAI Conference on Web and Social Media*, 12(1), Jun. 2018.

Z. Liu and M. Barahona. Graph-based data clustering via multiscale community detection. *Applied Network Science*, 5(1), Jan. 2020. ISSN 2364-8228. doi: 10.1007/s41109-019-0248-7. URL `http://dx.doi.org/10.1007/s41109-019-0248-7`.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982. doi: 10.1109/TIT.1982.1056489.

T. Pham, H. Dang, T. Le, and T. Le. Fast support vector clustering. *Vietnam Journal of Computer Science*, 4, 05 2016. doi: 10.1007/s40595-016-0068-y.

T. Pham, T. Le, and H. Dang. Scalable support vector clustering using budget, 2017.

X. Zhang, H. Zhang, Z. Wang, X. Ma, J. Luo, and Y. Zhu. Pwsc: a novel clustering method based on polynomial weight-adjusted sparse clustering for sparse biomedical data and its application in cancer subtyping. *BMC Bioinformatics*, 24, 12 2023.