Content:

Section I: Introduction to the concept of clustering and possible problems that may arise.
Section II: Overview of the *clustering* package, available on GitHub at fcasola/Clustering-Algorithms and comparison of the pros and cons of each algorithm.
Section III: Extension to the higher dimensional case.

Section I: The clustering problem

In *unsupervised learning*[1] one wants to obtain (or "learn") a function or an underlying law describing the dataset without having been previously exposed to labeled examples. The idea is the one of being capable of spotting general patterns directly into the raw data. Related techniques make for instance use of concepts like similarity within the data, in which case we talk about *clustering*[1], or try to discover latent variables within the input set by projecting data from a higher to a lower dimensional space, known as *dimensionality reduction*.

The problem of clustering has several extremely relevant practical applications, e.g. in image segmentation and compression[1], but its theoretical study is notoriously hampered by the lack of the very same formal definition of what a cluster is[2]. Historically, the problem was first addresses in the 50s with the introduction of an objective cost function, considering the sum of the squared Euclidean distances between each point and its respective cluster centre[3]. These centre locations would then be iteratively reassigned in order to minimize the cost function. The need to average over these k intracluster set of distances gave the name *k-mean* (KM) to the algorithm. It is customary to group under the name of *partitional* all algorithms making use of an objective function optimization[2].
Since the locus of points equidistant from two points (e.g. two cluster centres) is a straight line, one of the problems related with the KM algorithm is that it works only for linearly separable datasets. A mapping of each of the N data points $\mathbf{x}_i$ via $\mathbf{x}_i \to \phi(\mathbf{x}_i)$ into a *feature space* where the mapped points might be linearly separable was therefore introduced. The approach, called *Kernel k-mean* (KKM), owes its name to the fact that quadratic distances in feature space can be computed via the sole knowledge of the N^2 ij-pairwise products $K_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, with K_{ij} being the Kernel matrix.
Two major limitations are related with the KM and KKM algorithms: on one hand, number and initial centres of the clusters have to be specified[3], while on the other hand the choice of a Kernel matrix still defines a shape for the detectable clusters. In order to solve the problem of initial centres assignment, and therefore make the minimization of the objective function a deterministic process, the authors in Ref. 3 introduced the *Global Kernel k-mean* (GKKM) algorithm. In GKKM, each point in the dataset is progressively considered as a candidate cluster centre by evaluating the KKM clustering error (CE) resulting from the cost function. The one clustering solution with k clusters and the lowest CE, out of the total N possible, is then considered as the optimal one. The GKKM algorithm is deterministic, as the solution for k clusters is built progressively starting from the $k-1$ one. Although GKKM solves cluster initialization, it still requires to input the total number of clusters and it still makes use of a specific kernel; furthermore, the algorithm complexity is kN times larger than KKM[3]. Importantly, methods have been devised in order to reduce the GKKM complexity. A very effective one runs GKKM only over a set of $P \ll N$ initial cluster centres candidates or *exemplars*. These candidates are selected via likelihood maximization of a Gaussian-like mixture[1] in feature space, with the P exemplars being the best estimates for the mixture centres. The resulting algorithm, called *Global Kernel k-mean with Convex Mixture Models*[3] (GKKM-CMM), reduces the complexity by a factor P/N with respect to GKKM.

Even when provided with the GKKM-CMM algorithm, the user still has to specify the total number of clusters and the shape of a detectable cluster is still dependent on the Kernel choice. In order to overcome these limitations, the authors in Ref. 2 propose a scheme for cluster detection that aims at starting from the very same definition of what a cluster is. The inclusion (exclusion) of two points $\mathbf{x}_i, \mathbf{x}_j$ into (from) the same cluster is described by the ij-weights of a connectivity matrix w_{ij} being 1 (or 0). Such weights are *adaptive*, meaning that they get updated by progressively looking at bigger length scales in the vicinity of $\mathbf{x}_i$ and $\mathbf{x}_j$ and get changed to zero ($w_{ij} = 0$) if the number of points in between the pair is below a threshold value. The threshold is controlled by an hyperparameter λ for the model, the only free parameter of the analysis. The algorithm gets the name of *Adaptive Weights Clustering* (AWC), with the iterative search technique called *propagation-separation*[2].

Section II: Overview of the *clustering* package and analysis

The *clustering* Python package available at fcasola/Clustering-Algorithms contains implementations of 4 partitional (the KM, KKM, GKKM, GKKM-CMM) algorithms and one propagation-separation (the AWC) algorithm. Whereas a technical guide to the use of the package is present in the README file, an extensive testing of all the implemented algorithms is present in the analysis folder, in the form of interactive

jupyter notebooks.
We have implemented KM as a subcase of KKM with a linear Kernel. Ref. 3 proposed 2 algorithms to reduce complexity and improve performance of GKKM. We have chosen to implement GKKM-CMM as the latter was shown in the same reference to provide the best performance. We will here discuss clustering issues, pros and cons, comparisons between algorithms by referring to the demo analyses of the Shape and the UCI sets from the Clustering benchmark datasets (with the notation that, e.g., S2a and U2b are section 2a and 2b of the Shape and UCI notebooks, respectively). We will leave to the next section a discussion on the effects of high-dimensional data, mostly through an analysis of the UCI set.
After importing the *clustering* package and loading the datasets (sections S1 and U1), we first compared the baseline KM case with the case of KKM with a Gaussian Kernel. Section S2b clearly shows that KM wants to linearly separate data into 2 clusters, contrary to KKM. One of the assignment requests was to automatically infer the number of clusters from data instead of providing this expected number to the partitional algorithms. This problem has to be considered in connection with a proper choice of the model hyperparameter σ, the Gaussian Kernel variance. In order to tackle both issues, in section S3a-3d we performed a calculation of the so-called *Elbow plot* for different values of the parameter σ. The Elbow plots wants to compute the CE as a function of the number of input clusters, trying to essentially detect a change in the first derivative of the CE curve coinciding with the optimal cluster number. We have perfomed the Elbow analysis both for the raw and the normalized[4] data. Z-score normalization[4] does not have a major impact on the S dataset, meaning that the CE is not substantially reduced, and was henceforth discarded. The Elbow plot provides promising results (see e.g. section S3d), but overall a clear identification of the optimal number of clusters seems arduous and alternative methods should be tested. In section S4a-4c a systematic test of the partitional algorithms was performed on each element of the S dataset. Sections S4b and S4c plot the predicted labels, normalized CE and the value of the *Normalized Mutual Information* (NMI) for each element in S. The NMI (bounded between 0 and 1) is a measure of how much the true and predicted clustering labels match (NMI=1) or differ (NMI=0). We took the NMI definition from Ref. 2. We observe similar good results for both the KKM and GKKM-CMM algorithms, with $0.77 \leq \text{NMI} \leq 0.99$, except for the "spiral" and the "jam" set that have a more complicated topology and might require a different Kernel choice. The analysis of the S set terminates with a testing of the AWC algorithm. AWC is a nonparametric clustering algorithm, meaning that it is supposed to natively identify proper number of clusters and predict clustering labels. The authors in Ref. 2 propose two schemes, one called *sum of heuristic weights*, for the correct estimate of the hyperparameter λ, the only free parameter of the analysis. Even though in our present analysis of the S set we limited ourselves to an heuristic choice of the λ value, we note overall, in section S5b, very good results upon the application of the AWC routine. Proper number of clusters and labels were predicted for most of the elements in the S set (i.e. "aggregation", "D31", "R15", "flame"), with small deviations from the results published in Ref. 2 for the "compound" (scattered points are often identified as singleton clusters) and "pathbased" set. Our implementation of the AWC algorithm seems to suffer more for the "spiral" and the "jam" set, but again a proper estimate of λ could substantially improve the results.

Section III: Extension to the higher dimensional case

High dimensional spaces are notoriously difficult to deal with when it comes to learning algorithms. This is mostly related with the *curse of dimensionality*[1], a geometrical problem inherent to the fact that in order to keep a constant density, as the dimensionality D increases, one would need an exponentially larger number of points. As a result, the number of samples N must be very quickly large as D grows in order for a proper clustering to be possible. In addition, it is known[5] that in general as D increases, the distance to the nearest neighbour point approaches the one to the farthest one[5]. We tested high dimensional ($D > 2$) spaces via the U set. Four elements in the U set were considered (section U1) and dimensionality reduction to $D = 2$ was performed in order to attempt a meaningful visualization. We projected data to $D = 2$ using $t-$SNE, which has the advantage of grasping non-linearities but the problem of not being a deterministic operation. The "iris" set provides a nice way to compare KM and KKM in $D > 2$, since it has one class linearly separable from the other two. We indeed observe a fairly good result for both KM and KKM in section U2b. We keep the "iris" set and move on to test AWC algorithm as a function of the parameter λ, similarly to what suggested by the *sum of heuristic weights* method[2]. We observe in section U3b a plateau in the detected number of clusters as a function of the λ-value. Nicely, such value corresponds to the expected $k = 3$ value for the "iris" set. We picked the smallest λ value corresponding to the onset of a plateau in order to plot the expected labels. For this value of λ, the cluster structure looks stabilized and not affected by a further increase of this parameter[2]. Such good results could not be replicated for, e.g., the "yeast" set (see section U3c). In order to test whether this was related with the dimensionality issue or with an algorithm problem, the same test in U3b-3c was run using a GKKM-CMM algorithm in U4a. Once more, we observe good results for the "iris" set (U4b) and bad ones for the "yeast" set, pointing towards dimensionality and dataset-related problems.

IV: Bibliography and References:

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer Science (2006).

[2] K. Efimov and L. Adamyan, *Adaptive Nonparametric Clustering*, arXiv:1709.09102v1 (2017).

[3] G. F. Tzortzis and A. C. Likas, *The Global Kernel k-Means Algorithm for Clustering in Feature Space*, IEEE Trans. on Neur. Net. **20**, 7 (2009).

[4] V. R. Patel and R. G. Mehta, *Performance Analysis of MK-means Clustering Algorithm with Normalization Approach*, IEEE Inf. and Comm. Tech. - WICT, (2011).

[5] K. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, *When is "Nearest Neighbor" Meaningful?*, Int. Conf. on Dat. Theory, 217-235 (1999).