Geometric Methods - Final Project

António Loison, Florian Le Bronnec

April 13, 2022

Contents

1 Introduction

Finding clusters within data is an extremely common and useful task. Generally in clustering we want to find group of samples that are similar to each others. And once such an arrangement is found, it is very tempting to represent each cluster by its barycenter. We can even generalize this idea and to get a more powerful representation: we may want to represent our data by a linear combination of several atoms. This is the purpose of dictionary learning, that can be seen, under some considerations, as a generalization of clustering.

In this project we are going to study in detail the KMeans algorithm and the Non-Negative Matrix Factorization task. We will discuss how they are related and visualize their performances on some datasets.

2 Coding policy

2.1 Reproducibility

We designed all the experiments with the desire to make them reproducible. A special attention has been ported to the following points:

- All the experiments are results of Python scripts' outputs, provided along with this report.
- A precise list of requirements is given.
- All the scripts parameters have coherent defaults, corresponding to the outputs presented in the report, and most of them can be manually modified.
- Usage of predefined seed for pseudo-random generation (that can be manually changed).

2.2 Use of external code

We relied as much as possible on classical open-source libraries (Scikit-Learn and Numpy for the scientific aspects and Plotly for the visualization) for our implementations. And we coded ourselves what is left (we will give in this report a precise description of the code work).

3 Generic dataset

To illustrate our experiments, we made available a function called `gaussian_blob` that can construct a mixture of gaussian dataset, with some outliers. This is described in algorithm 1 and an illustrations is provided on fig. 1.

Algorithm 1: Gaussian Blob

Data: d the relative distance between blobs ; $n_{clusters}$: the number of clusters ; $n_{samples}$: the number of samples ; $\Sigma_1, \dots, \Sigma_{n_{clusters}}$: 2-tuples representing the diagonal covariance matrices of the gaussians ; $\theta_1, \dots, \theta_{n_{samples}}$: the rotations of each gaussian.

Result: $\mathbf{X} \in \mathbb{R}^{N \times 2}$, the drawn samples

Sample the $n_{clusters}$ centers uniformly on a grid of step d of size $3d \times 3d$ (this parameter can also be changed).

Sample $n_{samples}$ points from a Multinomial distribution $\sim \mathcal{M}(n_{samples}, \pi)$ to get the repartition within each gaussian, with $\pi = \begin{pmatrix} 1/n_{clusters} \\ \vdots \\ 1/n_{clusters} \end{pmatrix} \in \mathbb{R}^{n_{clusters}}$.

Sample the number of points belonging to each gaussian using the rotated corresponding covariance matrices Σ_i.

Add a small number of uniform noise over a grid of size $5d \times 5d$.

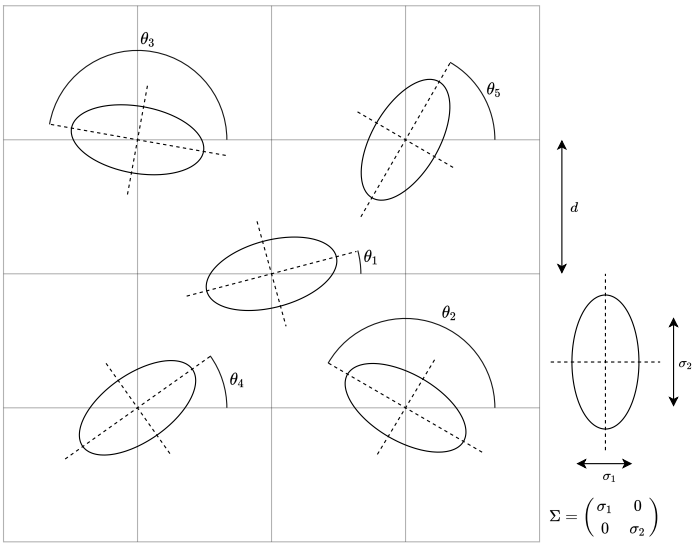


Figure 1: Illustration of the parametrized dataset.

4 KMeans++

4.1 KMeans

KMeans++ is a smart initialization of KMeans. To fully understand how KMeans++ works, let's first recall the KMeans algorithm as it will also be useful in the following parts of this report.

Algorithm 2: KMeans

Data: $\mathbf{X} \in \mathbb{R}^{d \times n}$: the samples ; $n_{clusters}$: the number of clusters.

Result: $\mathbf{Y} \in \mathbb{R}^{N}$, the cluster label corresponding to each sample.

Initialize the $n_{clusters}$ centers by choosing $n_{clusters}$ points among $\mathbf{X}$, and store them into a matrix $\mathbf{C} \in \mathbb{R}^{d \times n_{clusters}}$.

while *not convergence* **do**

Assign to each point to the cluster which has the nearest center.

$$\mathbf{Y}_i = \underset{k \in [\![1, \dots n_{clusters}]\!]}{\operatorname{argmin}} \|\mathbf{X}_i - \mathbf{C}_k\|_2^2. \tag{1}$$

Update $\mathbf{C}$ by computing the centroid of each new cluster:

$$\mathbf{C}_k = \frac{1}{n_k} \sum_{i \,|\, \mathbf{Y}_i = k} \mathbf{X}_i. \tag{2}$$

end

The Python code corresponding to algorithm 2 is implemented in the Python class `KMeans`.

KMeans converges but only to a local minima, which will hence depend on the initialization. This is where KMeans++ comes in.

4.2 KMeans++

4.3 Theoretical analysis

Kmeans++ is based on the simple idea that we can do better than pure random choice of initial points. Empirically, it seems coherent to think that choosing points that are more spread out can give better

results if the clusters are a bit separated. KMeans++ is described in algorithm 3.

Algorithm 3: KMeans

Data: $\mathbf{X} \in \mathbb{R}^{d \times n}$: the samples ; $n_{clusters}$: the number of clusters.

Result: $\mathbf{C} \in \mathbb{R}^{d \times n_{clusters}}$, the initial clusters' centers

Sample a first center $\mathbf{c}_1$ uniformly among $\mathbf{X}$.
for $k \in [\![2, \dots, n_{clusters}]\!]$ **do**
  Sample $\mathbf{c}_k$ from $\mathbf{X}$ with probabilities p_i:

$$\forall i \in [\![1, \dots, n]\!],\ p_i = \frac{D_i^2}{\sum_j D_j^2}, \tag{3}$$

  where $D_i = \min_{l \in [\![1,k]\!]} \|\mathbf{X}_i - \mathbf{c}_l\|_2^2$ is the distance from a sample to the closest already defined center.
end

Let's now introduce the potential function φ, which is in fact the function that KMeans try to minimize.

$$\varphi(\mathbf{C}) = \sum_{i=1}^{n} \min_{1 \leq k \leq K} \|\mathbf{X}_i - \mathbf{C}_k\|, \tag{4}$$

and let's call φ_{OPT} the optimal value of φ on $\mathbf{X}$ (computationally intractable). In [2] the authors showed that we have a lower bound on the expectation on a clustering ran with KMeans++ initialization, (5).

$$\mathbb{E}\left[\varphi(\mathbf{C})\right] \leq 8(\ln k + 2)\varphi_{OPT}. \tag{5}$$

But we should stay aware that this lower is not extremelly tight, and the variance may be high.

4.4 Implementation and results

4.4.1 Implementation

KMeans++ initialization has been implemented in the function `kmean_pp_init`. It makes use of Scikit-Learn KDTree to compute the distance of each point to the nearest centers.

In this function we can also change the exponent used in the probability computation in eq. (3), which is 2 by default.

4.4.2 Experiments

To illustrate different properties of KMeans++ we conducted a series of experiment.

Visual comparison The first experiment is a simple comparison of KMeans++ initialization and a random one, result is displayed on fig. 2. We see that centers are more likely to be drawn far from each other with KMeans++ and hence to be initially more coherent with the underlying clusters.

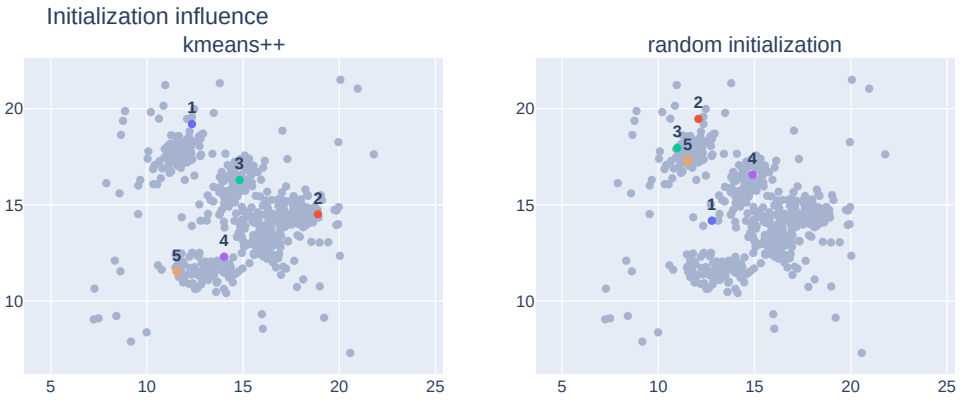


Figure 2: Numbers are the order in which the centers are sampled.

Influence of p on initialization Here p is the exponent used in the probabilities computation $p_i = \frac{D_i^p}{\sum_j D_j^p}$. Indeed, we can wonder why did the authors of [2] chose 2 and not something else. For that, let's see on fig. 3 some plots of the the initializations we can have when p increases. When p increases it seems we are selecting more outliers, which is in fact coherent with the expression of p_i. Figure 4 is very informative and gives the average number of selected outliers in the initialization as p increases. We see that using $p = 1$ seems to give an even lower number of outliers than 2 (which does not necessary implies a better result after the clustering has been ran, because for example selecting K times the same point will give a low number of outliers but probably a poor clustering result).

Remark 4.1 (Outliers)**.** *On fig. 4 we counted the number of outliers. This is done thanks to the Python function* `get_n_outliers`*. This function compute the Mahalanobis distance between the samples and each center using their associated covariance matrices. Then if the minimum distance of a sample from the centers is higher than a given threshold (4 by default, that correspond roughly to a probabilty of 0.1 of not being an outlier when drawn from the corresponding gaussian), the sample is considered as an outlier.*

Figure 3: Illustration of some selected centers with different values of p.

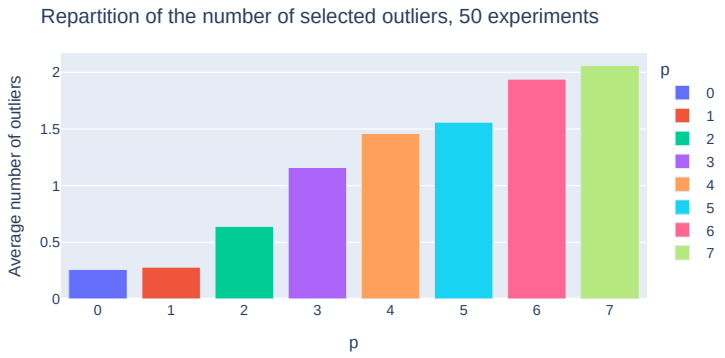


Figure 4: Numbers of selected outliers as p increases. $p = 0$ corresponds to random initialization.

Influence of p on φ We carried a similar experiment but this time we monitored the final value of φ, after we ran KMeans clustering on the dataset $\mathbf{X}$. This time we used a violin plot which show with more details the repartition of the values of φ along our experiments, on fig. 5. We see that on this kind of dataset smaller values of p leads to better results in term of φ on average. However, it is interesting to observe that we can have some "failure" cases where the obtained value of φ is very high, even with $p = 2$. This justifies the idea of launching several time the algorithm and selecting the run which corresponds to the better φ, as with what is done in Scikit-Learn. It is also interesting to see that small values of p (*i.e.,* 1 or 2 here) seem to give very similar results. There is maybe a very small advantage for $p = 2$ because the mass of the distribution seems to be a little more concentrated around the low values. We also have to acknowledge that on this particularly simple example the gain of using KMeans++ is not high.

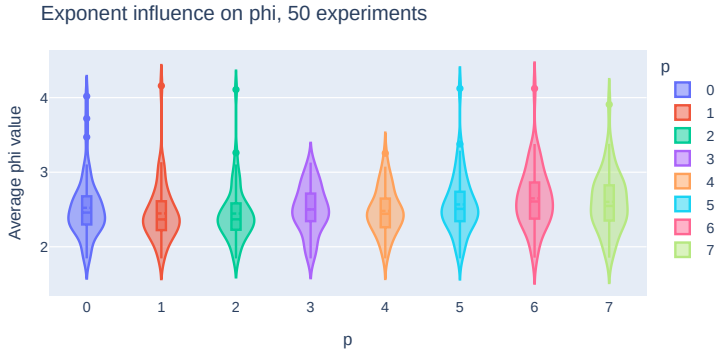


Figure 5: Evolution of φ with p. We encourage the reader to launch himself the experiment and to investigate the interactive figures if he is interested on having more precise values for this plot.

5 NMF and KMeans

Non-negative matrix factorization is an instance of the more general dictionary problem, with the specificity that we are only allowed to work with matrix with positive coefficients. This is in fact not restrictive, because in real life a lot of data are positive: distances, prices, images, counts. Moreover, NMF leads to better interepretable results.

5.1 Problem definition

To explain this remark on interpretability, let's first formalize the problem. NMF aims at finding a factorization $\mathbf{X} \approx \mathbf{F}\mathbf{G}^T$, or more formally, it aims at solving the following optimization problem:

$$\underset{\mathbf{F}\in\mathbb{R}_+^{p\times K},\mathbf{G}\in\mathbb{R}_+^{n\times K}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|_2^2. \tag{6}$$

And in fact, $\mathbf{F}$ can be interpreted as a dictionary of atoms used to reconstruct $\mathbf{X}$ with each row $\mathbf{G}^i$ with

$1 \leq i \leq n$ of $\mathbf{G}$ corresponding to the coefficients of the linear combination of the atoms of $\mathbf{F}_1, \ldots, \mathbf{F}_K$ in the reconstruction of $\mathbf{X}_i$. And because this combination only have positive coefficients, each used atoms must bring something useful in the reconstruction, as we won't be able to substract it later.

5.2 Orthogonality constraints and links with KMeans

In the limiting case where $\mathbf{G}$ only has one coefficient per column, each $\mathbf{X}_i$ is reconstructed with only one atom. This is very similar to what is done with clustering, where points are represented by their centroids.

Indeed, if $\mathbf{G}^T\mathbf{G} = \mathbf{I}$, it implies that necessarily each row of $\mathbf{G}$ has a unique non-zero coefficient. Hence, each $\mathbf{X}_i$ is reconstructed with only one atom of $\mathbf{F}$.

$$\underset{\mathbf{F}\in\mathbb{R}_+^{p\times K},\mathbf{G}\in\mathbb{R}_+^{n\times K}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|_2^2, \quad \text{such that} \quad \mathbf{G}^T\mathbf{G} = \mathbf{I}_K. \tag{7}$$

This is basicall what made the authors of [4] claimed the equivalence between KMeans and NMF (the interested reader can have a look at the article to get a more complete description of their reasoning).

Remark 5.1 (KMeans - NMF Equivalence)**.** *One has to notice that in KMeans the class indicator vectors that are interpreted as the rows of* $\mathbf{G}$ *have a more restrictive form than just being orthogonal to each other: each row must have equal non-zero coefficients. And with only the orthogonality constraint of* (7)*, nothing ensures that in the end we will end up with class indicator vectors in* $\mathbf{G}$*.*

Remark 5.2 (KMeans - NMF Equivalence other proof)**.** *This idea to proof an equivalence between KMeans and NMF has been investigated a lot in the literature and [1] claimed also to provide another proof of the equivalence between KMeans and NMF, in Theorem 1. In the proof of this theorem they use the zero gradient condition but stay a bit vague on the justification for using it, especially because we are solving an optimisation problem under constraints. In the general case the Lagrange condition will give use something like:*

$$\nabla J + \sum \lambda_i \nabla F_i = 0, \tag{8}$$

where $J(\mathbf{F}, \mathbf{G}) = \|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|_2^2$ *is the objective function, the* F_i *are the constraint functions and* λ_i *are the Lagrange's multipliers. Which shows that the zero-gradient condition is not obvious.*

We will now give some detail on the implementation and on the experiments we ran.

5.3 Implementation

To solve the NMF problem with orthogonality constraints (7), [1] derive a complete algorithm based on the Lagrangian of the problem. We refer to [1] for a complete proof of the correctness of the algorithm. Algorithm 4 gives iterative update rules to use.

Algorithm 4: 2 factors NMF

Data: $\mathbf{X} \in \mathbb{R}^{d\times n}$: the samples ; K: the rank of the matrices.

Result: $\mathbf{F} \in \mathbb{R}^{d\times K}, \mathbf{G} \in \mathbb{R}^{n\times K}$ the approximate decomposition of $\mathbf{X}$.

Initialize $\mathbf{F}$ and $\mathbf{G}$ at random with positive values.
while *not convergence* **do**

$$\begin{cases} \mathbf{G} \leftarrow \mathbf{G} \odot \sqrt{\dfrac{\mathbf{X}^T\mathbf{F}}{\mathbf{G}\mathbf{G}^T\mathbf{X}^T\mathbf{F}}}, \\ \mathbf{F} \leftarrow \mathbf{F} \odot \dfrac{\mathbf{X}\mathbf{G}}{\mathbf{F}\mathbf{G}^T\mathbf{G}}, \end{cases} \tag{9}$$

where $\odot$ means element-wise multiplication and / the element-wise division.
end

Remark 5.3 (Link with KMeans updates)**.** *The proof of these updates in [1] uses similar arguments than for the EM algorithm, with the use of an auxiliary function and a guarantee in the decrease of the error at each step. But the other argument to derive such update rules are a bit more involved than classical*

EM algorithm (like for GMM). Hence, it introduces another difference with KMeans algorithm in the interpretability of the update rules. We will give further remarks on the resolution of 2 factors NMF in the results section.

5.4 Non-Negative Matrix Tri-Factorisation

In [1] the authors introduces also the Non-Negative Matrix Tri-Factorisation problem, with orthogonality constraints, which amounts at solving:

$$\underset{\mathbf{F}\in\mathbb{R}_+^{p\times K},\mathbf{S}\in\mathbb{R}_+^{K\times K},\mathbf{G}\in\mathbb{R}_+^{n\times K}}{\operatorname{argmin}} \|\mathbf{X}-\mathbf{F}\mathbf{G}^T\|_2^2, \quad \text{such that} \quad \mathbf{F}^T\mathbf{F}=\mathbf{I}_K,\ \mathbf{G}^T\mathbf{G}=\mathbf{I}_K. \tag{10}$$

Three factors NMF is useful because it can be interpreted as a clustering of both the rows and columns of $\mathbf{X}$ (with the same ideas than in section 5.2), when $\mathbf{S}$ is diagonal. In fact, introducing a double orthogonality constraint is very restrictive but using $\mathbf{S}$ introduce another degree of freedom which is beneficial.

5.5 Implementation and results for 2 factors NMF

5.5.1 Implementation

2-Factors NMF has been implemented in the Python class `NMF`. Several initializations are possible:

- `"random"`: the coefficients are initialized uniformly between 0 and 1.
- `"kmeans++"`: the atoms of $\mathbf{F}$ are interpreted as cluster centers and drawn using KMeans++ initialization $\mathbf{G}$ is drawn at random on $[0, 1]$.

5.5.2 Results

Orthogonality constraints In order to investigate more deeply the relations between KMeans and NMF, it is important to know the final form of $\mathbf{G}$, especially whether its rows are orthogonal. Figure 6 shows that orthogonality has not been reached, despite the algorithm has converged. In [1] the authors explain indeed that in general the orthogonality is not reached, due to an approximation of the Lagrangian in algorithm 4.

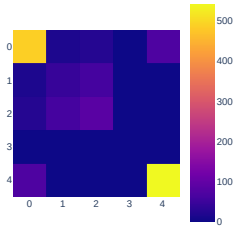
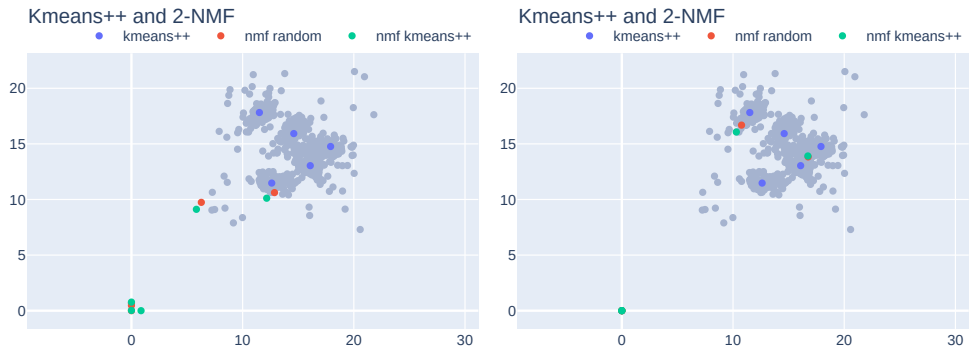


Figure 6: Visualization of $\mathbf{G}^T\mathbf{G}$ returned by algorithm 4.

Clustering performances Since the begining we are talking a lot about NMF for clustering. The previous paragraph showed that NMF in fact yields soft clustering assignment. In most cases the values of the factor matrices are initialized at random. As we are investigating the clustering capabilities of NMF, we had the idea to use KMeans++ initialization for $\mathbf{F}$. To get a better idea of the kind of clusters it can find, let's have a look at fig. 7a. We clearly see that the atoms of $\mathbf{F}$ cannot be interpreted as cluster centers. Even if we rescale these center, as shown on fig. 7b it is difficult to interpret it as a satisfying clustering result. Figure 7 shows the clustering performances with both random and KMeans++ initializations.

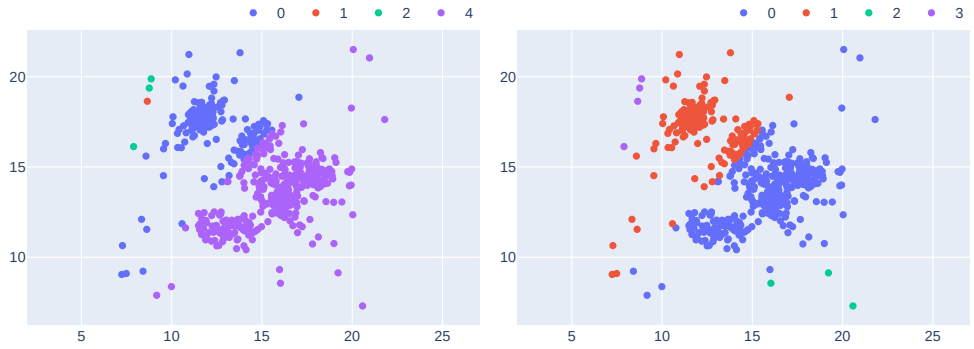


(a) Direct plot of the columns of $\mathbf{F}$.

(b) Rescaled columns of $\mathbf{F}$.

Figure 7: Visualization of the columns of $\mathbf{F}$.

Remark 5.4 (Rescaling)**.** *To rescale the columns of* $\mathbf{F}$ *we multiplied each column of* $\mathbf{F}$ *by the quantile of level* $1-r$ *of the corresponding coefficients in the colmuns of* $\mathbf{G}$*, where* $r = \frac{1}{n_{clusters}}$*. Indeed, if we assume that the result is analogous to a classical clustering output, on average a proportion* r *should be assigned to each cluster (balanced dataset) and therefore have the corresponding coefficients should have a high value.*



(a) Random initialization.

(b) KMeans++ initialization.

Figure 8: Clustering performances of NMF with different initializations. In both cases the result is not satisfying. The labels correspond to the argmax over the columns of $\mathbf{G}$.

5.6 Implementation and results for 3 factors NMF

5.6.1 Implementation

The implementation can be found in the Python class `TriNMF`. We refer to [1] for the update rules. There is an option to use $\mathbf{S}$ as a diagonal matrix, as it is with this form that the authors justified the clustering capabilities of 3 factors NMF.

5.6.2 Results

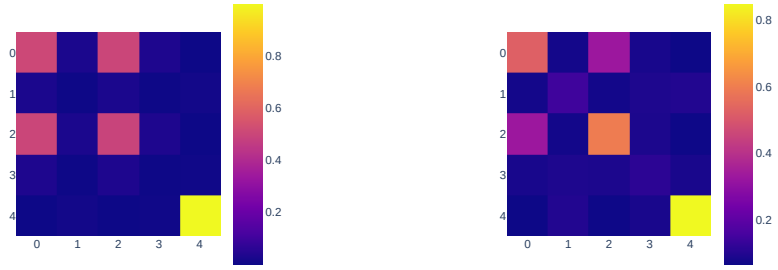
We conducted the same kind of experiments than with 2 factors NMF and compared the results.

S diagonal As explained above, when $\mathbf{S}$ is diagonal we can interpret $\mathbf{G}$ and $\mathbf{F}$ as the result of columns and rows of $\mathbf{X}$ clustering. However in practice we observed that using $\mathbf{S}$ diagonal is not giving good results in term of reconstruction, the loss converges to a local minimum quite far from zero, see fig. 9a, and the associated reconstruction, fig. 9b.

(a) Reconstruction loss $\|\mathbf{X} - \mathbf{FSG}^T\|_2^2$ (b) Reconstruction of $\mathbf{X}$.

Figure 9: 3-NMF reconstruction results.

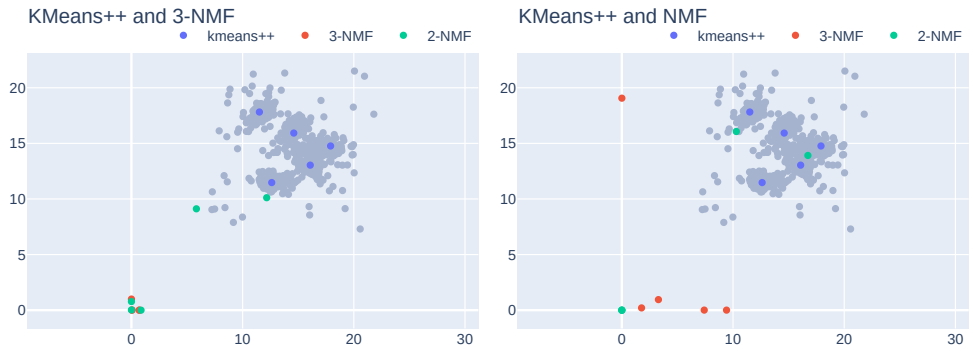
Orthogonality As for 2-NMF we can look at the orthogonality of both $\mathbf{F}$ and $\mathbf{G}$, displayed on fig. 10. We see again that here orthogonality is not reached.



(a) $\mathbf{F}^T\mathbf{F}$ (b) $\mathbf{G}^T\mathbf{G}$.

Figure 10: Orthogonality of the resulting matrices.

Clustering performances We can have a look at the repartition of the atoms and compare it with KMeans++ and 2-NMF once rescaled, see fig. 11. The clustering result is also plotted on fig. 12 (argmax of columns of $\mathbf{G}$). It is still not satisfying as a clustering result.



(a) Direct plot of the columns of $\mathbf{F}$. (b) Rescaled columns of $\mathbf{F}$.

Figure 11: Visualization of the columns of $\mathbf{F}$.

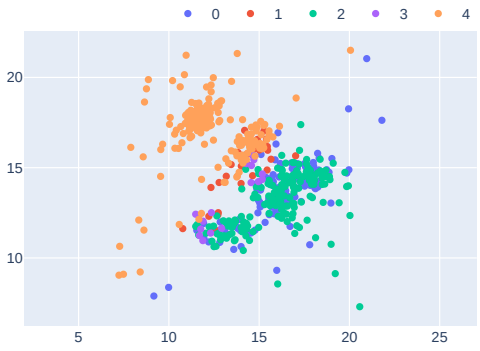
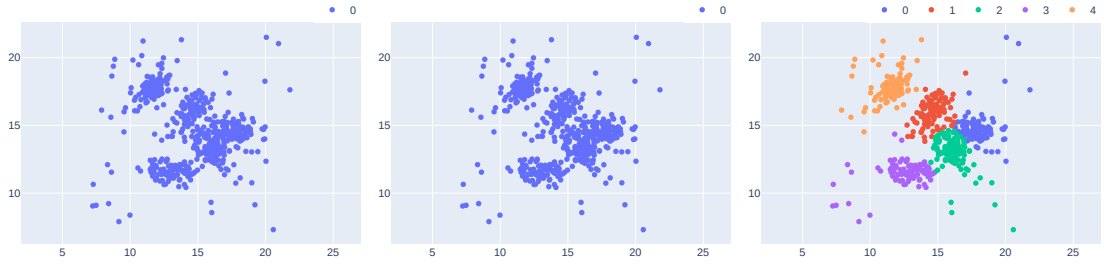


Figure 12: Clustering result with 3-NMF.

5.7 Data embedding

In this last section we decided to embed our 2D data in $\mathbb{R}^{50}$, and see what are the performances of the different algorithms we tried until now. Indeed, it can be interesting to see whether NMF can perform better than KMeans in a more complicated space.



(a) 2-NMF. (b) 3-NMF. (c) KMeans.

Figure 13: Clustering visualization on our projected data. Only KMeans yields a decent result.

6 Conclusion

In this project we have presented KMeans++ as well as 2 and 3 factors NMF. We have presented these algorithms under a clustering perspective, and we have to admit that the link between clustering and NMF is not as clear as what has been claimed in some papers. The formulation is undoubtedly very similar. But we have shown that similarity does not mean equivalence, and our experiments' results clearly showed that at least on our example NMF are not well suited for clustering.

This has however to be taken with caution, because other papers proved that on some text datasets NMF was performing really good [1, 3]. An explanation we can think of is that the examples shown by these article are from text data where the feature vectors are very sparse. In this situation KMeans may be indeed more limited because comparing word vectors with the euclidean distance is not really efficient. NMF can therefore be useful because it can finds a more efficient embedding.

To conclude, regarding clustering we encourage the reader to first work with KMeans, as the algorithm is simpler and more interpretable. However, if some limitations are encountered with KMeans, like with what can happen with text data, she/he could try to use NMF but with the limitations we mentioned in mind.

References

[1] Chris Ding et al. "Orthogonal Nonnegative Matrix T-Factorizations for Clustering". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '06. Philadelphia, PA, USA: Association for Computing Machinery, 2006, pp. 126–135. ISBN: 1595933395. DOI: 10.1145/1150402.1150420. URL: https://doi.org/10.1145/1150402.1150420.

[2] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. ISBN: 9780898716245.

[3] Del Nicoletta and Gianvito Pio. "Non-negative Matrix Tri-Factorization for co-clustering: An analysis of the block matrix". In: *Information Sciences* 301 (Apr. 2015), pp. 13–26. DOI: 10.1016/j.ins.2014.12.058.

[4] Chris Ding, Xiaofeng He, and Horst D. Simon. "On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering". In: *Proceedings of the 2005 SIAM International Conference on Data Mining (SDM)*, pp. 606–610. DOI: 10.1137/1.9781611972757.70. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611972757.70. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972757.70.