exercise 6

density modeling and vector quantization

solutions due

until **January 25, 2024** at **23:59** via **ecampus**

general remarks

This exercise sheet covers material we study in lectures 11, 12, 13, and 14. In these lectures, we present bits and pieces of code which may come in handy. We also suggest revisiting what was implemented in exercises 4 and 5 . . .

task 6.1 [no points]

EM for GMMs

Implement the EM algorithm for fitting a general GMM of k components

$$p\Big(\boldsymbol{x} \;\Big|\; \big\{w_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\big\}_{i=1}^{k} \Big) = \sum_{i=1}^{k} w_i \, \mathcal{N}\big(\boldsymbol{x} \;\big|\; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i \big)$$

to a set of n Euclidean data points

$$\mathcal{D} = \big\{\boldsymbol{x}_j\big\}_{j=1}^{n} \subset \mathbb{R}^m$$

Recall that, for this specific but common practical setting, the EM algorithm simply reads

for $i = 1, \ldots, k$
  initialize w_i, $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$

for $t = 1, \ldots, t_{\text{max}}$

  for $i = 1, \ldots, k$ // E-step
    for $j = 1, \ldots, n$

$$\zeta_{ij} = \frac{w_i \, \mathcal{N}\big(\boldsymbol{x}_j \;\big|\; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\big)}{\sum\limits_{i=1}^{k} w_i \, \mathcal{N}\big(\boldsymbol{x}_j \;\big|\; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\big)}$$

  for $i = 1, \ldots, k$ // M-step

$$w_i = \frac{\sum\limits_{j=1}^{n} \zeta_{ij}}{n}$$

$$\boldsymbol{\mu}_i = \frac{\sum\limits_{j=1}^{n} \zeta_{ij} \, \boldsymbol{x}_j}{\sum\limits_{j=1}^{n} \zeta_{ij}}$$

$$\boldsymbol{\Sigma}_i = \frac{\sum\limits_{j=1}^{n} \zeta_{ij} \, \big[\boldsymbol{x}_j - \boldsymbol{\mu}_i\big] \big[\boldsymbol{x}_j - \boldsymbol{\mu}_i\big]^{\mathsf{T}}}{\sum\limits_{j=1}^{n} \zeta_{ij}}$$

Note: Please do *not* use `sklearn` or other fancy libraries which you may find on the Web. Rather, implement your solution using only plain vanilla `numpy`, `scipy`, `matplotlib`, ... functionalities.

Hints: It is common practice to initialize all mixture weights and covariance matrices to

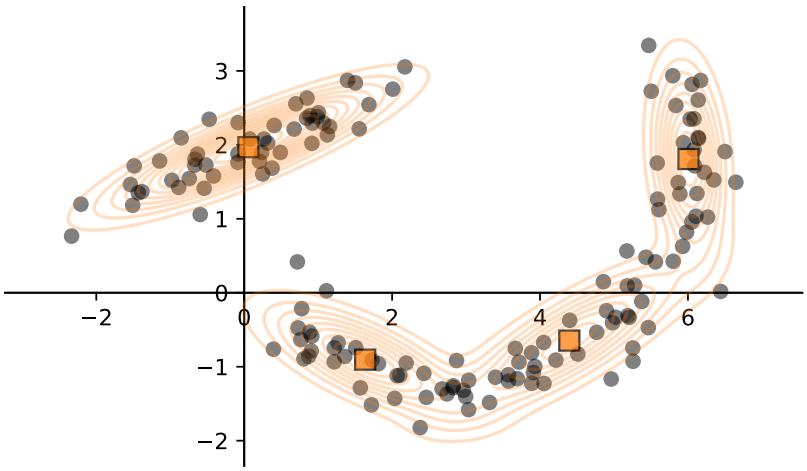
$$w_i = \frac{1}{k} \in \mathbb{R}$$
$$\boldsymbol{\Sigma}_i = \boldsymbol{I} \in \mathbb{R}^{m \times m}$$

and to run k-means clustering on the data in $\mathcal{D}$ in order to initialize the mixture means $\boldsymbol{\mu}_i$.

Test your code for $k = 4$ on the $m = 2$-dimensional data in file

`oneBlobOneMoon.csv`

If your code works correctly and if you use what you learned in exercise 5 to visualize your results, they could/should look something like this



task 6.2 [no points]

ITVQ

Recall that information theoretic vector quantization (ITVQ) reduces a given data matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$$

to a codebook

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_k \end{bmatrix} \in \mathbb{R}^{m \times k}$$

of $k \ll n$ prototypes.

Implement the ITVQ procedure

initialize

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_k \end{bmatrix}$$

for $t = 1, \ldots, t_{\text{max}}$

compute

$$\boldsymbol{\Phi}_{xw} \text{ with entries } \left[\boldsymbol{\Phi}_{xw}\right]_{ij} = \phi_\sigma\left(\boldsymbol{x}_i, \boldsymbol{w}_j\right)$$

$$\boldsymbol{\Phi}_{ww} \text{ with entries } \left[\boldsymbol{\Phi}_{ww}\right]_{ij} = \phi_\sigma\left(\boldsymbol{w}_i, \boldsymbol{w}_j\right)$$

$$\boldsymbol{D}_{xw} = \operatorname{diag}\left[\mathbf{1}_n^\intercal \boldsymbol{\Phi}_{xw}\right]$$

$$\boldsymbol{D}_{ww} = \operatorname{diag}\left[\mathbf{1}_k^\intercal \boldsymbol{\Phi}_{ww}\right]$$

$$c = \frac{n\, \mathbf{1}_n^\intercal \boldsymbol{\Phi}_{xw}\, \mathbf{1}_n}{k\, \mathbf{1}_k^\intercal \boldsymbol{\Phi}_{ww}\, \mathbf{1}_k}$$

update

$$\boldsymbol{W} = \left[\boldsymbol{X}\, \boldsymbol{\Phi}_{xw} + c \cdot \boldsymbol{W}\, \left[\boldsymbol{D}_{ww} - \boldsymbol{\Phi}_{ww}\right]\right] \boldsymbol{D}_{xw}^{-1}$$

where $\phi_\sigma\left(\boldsymbol{a}, \boldsymbol{b}\right)$ denotes an isotropic Gaussian density kernel, namely

$$\phi_\sigma\left(\boldsymbol{u}, \boldsymbol{v}\right) = \frac{1}{\left(2\,\pi\,\sigma^2\right)^{\frac{m}{2}}} \exp\left(-\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{2\,\sigma^2}\right)$$

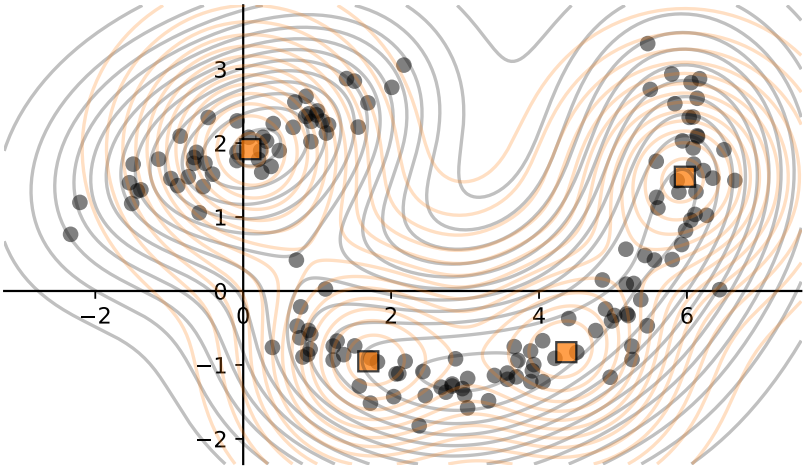
Hints: It is generally a good idea to initialize the codebook $\boldsymbol{W}$ by randomly sampling k columns from the data matrix $\boldsymbol{X}$.

W.r.t. code efficiency, you might want to think about the following. Do you really have to implement the two matrices $\boldsymbol{D}_{xw}$ and $\boldsymbol{D}_{ww}$? After all they are but diagonal matrices and there may be a faster way to practically compute their effects?

Test your code for $k = 4$ on the $m = 2$-dimensional data in file

`oneBlobOneMoon.csv`

If your code works correctly and if you use what you learned in exercise 5 to visualize your results, they could/should look something like this



exemplary ITVQ result for $\sigma^2 = 2$

Once you convinced yourself that your code works properly, see what it produces for different choices of k and σ^2.

task 6.3 [no points]

VQ via minimizing MD

When working with high-dimensional data ($m \gg 1$), ITVQ as in the previous task is numerically brittle! (You may explore this for yourself in task 6.5). However, in lecture 14, we will learn about a related, much faster, and much more robust method, namely vector quantization by means of *minimizing the mean discrepancy* between two Parzen density estimates.

Crucially and in contrast to ITVQ, mean discrepancy minimization works with a non-normalized Gaussian kernel

$$\phi_\sigma(\boldsymbol{u}, \boldsymbol{v}) = \exp\left(-\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{2\,\sigma^2}\right)$$

and proceeds as follows

initialize

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_k \end{bmatrix}$$

for $t = 1, \ldots, t_{\text{max}}$

compute

$\boldsymbol{\Phi}_{xw}$ with entries $\left[\boldsymbol{\Phi}_{xw}\right]_{ij} = \phi_\sigma(\boldsymbol{x}_i, \boldsymbol{w}_j)$

$\boldsymbol{\Phi}_{ww}$ with entries $\left[\boldsymbol{\Phi}_{ww}\right]_{ij} = \phi_\sigma(\boldsymbol{w}_i, \boldsymbol{w}_j)$

$$\boldsymbol{D}_{xw} = \operatorname{diag}\left[\mathbf{1}_n^\intercal \boldsymbol{\Phi}_{xw}\right]$$

$$\boldsymbol{D}_{ww} = \operatorname{diag}\left[\mathbf{1}_k^\intercal \boldsymbol{\Phi}_{ww}\right]$$

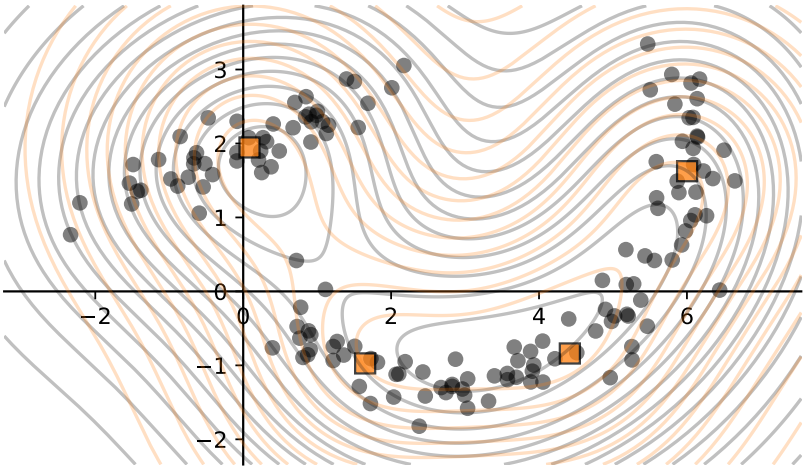
update

$$\boldsymbol{W} = \left[\boldsymbol{X}\,\boldsymbol{\Phi}_{xw} + \frac{n}{k} \cdot \boldsymbol{W}\left[\boldsymbol{D}_{ww} - \boldsymbol{\Phi}_{ww}\right]\right] \boldsymbol{D}_{xw}^{-1}$$

here is your task

Implement this procedure, run your code on the `oneBlobOneMoon.csv` data, and visualize your results.

If everything works properly, results, they could/should look something like this



exemplary MMD result for $k = 4$, $\sigma^2 = 2$

task 6.4 [no points]

prototypes within kernel MEBs

In exercise 5, you already computed (Gaussian) kernel minimum enclosing balls. For reference, we briefly recall the basic idea / procedure.

Given (Euclidean) data points in a matrix

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix} \in \mathbb{R}^{m \times n}$$

and a non-normalized Gaussian kernel

$$\phi_\sigma(\boldsymbol{u}, \boldsymbol{v}) = \exp\left(-\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{2\,\sigma^2}\right)$$

compute a kernel matrix and a kernel vector

$$\boldsymbol{\Phi} \text{ with entries } \big[\boldsymbol{\Phi}\big]_{ij} = \phi_\sigma(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$\boldsymbol{\phi} \text{ with entries } \big[\boldsymbol{\phi}\big]_{j} = \phi_\sigma(\boldsymbol{x}_j, \boldsymbol{x}_j)$$

and then solve the dual kernel MEB problem

$$\begin{aligned} \hat{\boldsymbol{\mu}} = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \;\; & \boldsymbol{\mu}^\intercal \boldsymbol{\Phi}\, \boldsymbol{\mu} - \boldsymbol{\mu}^\intercal \boldsymbol{\phi} \\ \text{s.t.} \;\; & \mathbf{1}^\intercal \boldsymbol{\mu} = 1 \\ & \boldsymbol{\mu} \geq \mathbf{0} \end{aligned}$$

Once $\hat{\boldsymbol{\mu}}$ is available, we can work with the ball's "characteristic function"

$$\chi_{\mathcal{B}}(\boldsymbol{x}) = \phi_\sigma(\boldsymbol{x}, \boldsymbol{x}) - 2\,\hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\varphi}(\boldsymbol{x}) + 2\,\hat{\boldsymbol{\mu}}^\intercal\, \boldsymbol{\Phi}\, \hat{\boldsymbol{\mu}} - \hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\phi}$$

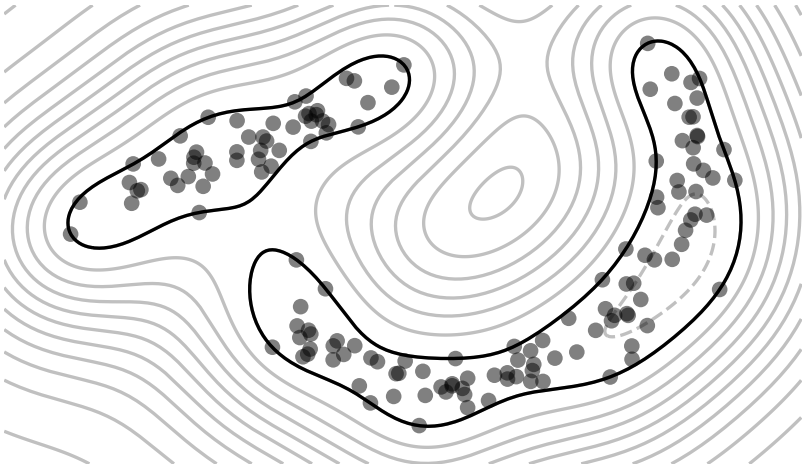
where the entries of the kernel vector $\boldsymbol{\varphi}(\boldsymbol{x})$ are given by

$$\big[\boldsymbol{\varphi}(\boldsymbol{x})\big]_j = \phi_\sigma(\boldsymbol{x}_j, \boldsymbol{x})$$

Function $\chi_{\mathcal{B}}(\boldsymbol{x})$ is interesting, because it *characterizes* the kernel MEB $\mathcal{B}$ in the following sense

$$\chi_{\mathcal{B}}(\boldsymbol{x}) = \begin{cases} < 0 & \text{if } \boldsymbol{x} \in \mathcal{B} \setminus \partial\mathcal{B} \\ = 0 & \text{if } \boldsymbol{x} \in \partial\mathcal{B} \\ > 0 & \text{if } \boldsymbol{x} \notin \mathcal{B} \end{cases}$$

For instance, if we compute a $\sigma^2 = 1.25$ kernel MEB for the exemplary data in `oneBlobOneMoon.csv` and visualize $\chi_{\mathcal{B}}(\boldsymbol{x})$, we obtain something like



In this plot, the bold level lines represent the ball's surface ($\chi_{\mathcal{B}}(\boldsymbol{x}) = 0$), the dashed level lines show its interior ($\chi_{\mathcal{B}}(\boldsymbol{x}) < 0$), and all other level lines show its exterior ($\chi_{\mathcal{B}}(\boldsymbol{x}) > 0$).

remarks regarding code efficiency

Since efficient implementations are a reoccurring theme in our exercises, we note the following: **when working with Gaussian kernels, we have**

$$\phi_\sigma\big(\boldsymbol{x}_j, \boldsymbol{x}_j\big) = 1$$

(Convince yourself that this holds true.) But this means that $\boldsymbol{\phi} = \mathbf{1}$ so that $\boldsymbol{\mu}^\intercal \boldsymbol{\phi} = \boldsymbol{\mu}^\intercal \mathbf{1} = 1$ is a constant. The dual kernel MEB problem therefore simplifies to

$$\begin{aligned} \hat{\boldsymbol{\mu}} = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \;\; & \boldsymbol{\mu}^\intercal \boldsymbol{\Phi}\, \boldsymbol{\mu} \\ \text{s.t.} \;\; & \mathbf{1}^\intercal \boldsymbol{\mu} = 1 \\ & \boldsymbol{\mu} \geq \mathbf{0} \end{aligned} \tag{1}$$

Since we also have $\phi_\sigma\big(\boldsymbol{x}, \boldsymbol{x}\big) = 1$, the characteristic function of a Gaussian kernel MEB simplifies to

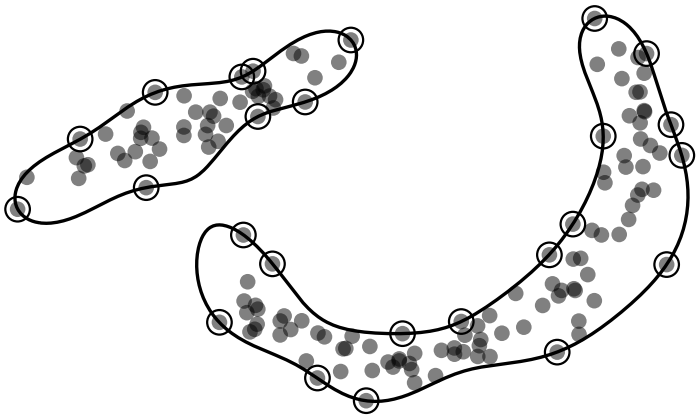
$$\chi_{\mathcal{B}}(\boldsymbol{x}) = 2\, \hat{\boldsymbol{\mu}}^\intercal\, \boldsymbol{\Phi}\, \hat{\boldsymbol{\mu}} - 2\, \hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\varphi}(\boldsymbol{x})$$

Moreover, we recall from lecture 08 that the solution $\hat{\boldsymbol{\mu}}$ to (1) is typically sparse because

$$\hat{\mu}_j = 0 \;\Leftrightarrow\; \boldsymbol{x}_j \text{ lies inside of } \mathcal{B}$$

$$\hat{\mu}_j > 0 \;\Leftrightarrow\; \boldsymbol{x}_j \text{ lies on the surface of } \mathcal{B}$$

and, as illustrated by the following figure, there are usually but a few points which support the surface of $\mathcal{B}$.



The fact that many of the $\hat{\mu}_j$ are zero can simplify the computation of $\chi_{\mathcal{B}}(\boldsymbol{x})$ even further ...

If we refer to the support vectors of $\mathcal{B}$ as $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k$ and to their respective Lagrange multipliers as $\hat{\nu}_1, \ldots, \hat{\nu}_k$, we may gather the latter in a vector $\hat{\boldsymbol{\nu}}$ and introduce

$$\boldsymbol{\Psi} \text{ with entries } \big[\boldsymbol{\Psi}\big]_{ij} = \phi_\sigma\big(\boldsymbol{s}_i, \boldsymbol{s}_j\big)$$

$$\boldsymbol{\psi}(\boldsymbol{x}) \text{ with entries } \big[\boldsymbol{\psi}\big]_{i} = \phi_\sigma\big(\boldsymbol{s}_i, \boldsymbol{x}\big)$$

In terms of these smaller vectors and matrices, the characteristic function becomes

$$\chi_{\mathcal{B}}(\boldsymbol{x}) = 2\,\hat{\boldsymbol{\nu}}^{\mathsf{T}}\,\boldsymbol{\Psi}\,\hat{\boldsymbol{\nu}} - 2\,\hat{\boldsymbol{\nu}}^{\mathsf{T}}\boldsymbol{\psi}(\boldsymbol{x}) \tag{2}$$

Observe that the gradient of $\chi_{\mathcal{B}}(\boldsymbol{x})$ in (2) is given by

$$\nabla\chi_{\mathcal{B}}(\boldsymbol{x}) = -\nabla 2\,\hat{\boldsymbol{\nu}}^{\intercal}\boldsymbol{\psi}(\boldsymbol{x}) = -2\,\nabla\sum_i \hat{\nu}_i\,\phi_\sigma\big(\boldsymbol{s}_i,\boldsymbol{x}\big)$$

$$= \frac{2}{\sigma^2}\sum_i \hat{\nu}_i\,\big[\boldsymbol{s}_i - \boldsymbol{x}\big]\,\phi_\sigma\big(\boldsymbol{s}_i,\boldsymbol{x}\big)$$

Equating this to zero and solving for $\boldsymbol{x}$ thus results in

$$\boldsymbol{x} = \frac{\sum_i \hat{\nu}_i\,\phi_\sigma\big(\boldsymbol{s}_i,\boldsymbol{x}\big)\,\boldsymbol{s}_i}{\sum_i \hat{\nu}_i\,\phi_\sigma\big(\boldsymbol{s}_i,\boldsymbol{x}\big)} \tag{3}$$

Note: We have already seen a similar expression when we studied the mean shift procedure in lecture 12.

here is your task

Implement code that computes the fixed point iteration in (3) for all points in a data matrix $\boldsymbol{X}$. Run your code on the data in file

`oneBlobOneMoon.csv`

and visualize your results. That is, visualize the data together with the points or *prototypes* the above iterations converge to.

task 6.5 [no points]

working with real data

Use something like the following to read the data in file `faceMatrix.npy` into a *numpy* array `matX`.

```
matX = np.load('faceMatrix.npy').astype(float)
m, n = matX.shape
```

Below, your tasks will be to compress the data in `matX` into a codebook `matW`. First, however, not the following ...

The data point we are dealing with in this task are $m = 361$ dimensional. By modern standards, this is not really that high-dimensional but will already lead to numerical issues when not addressed. Indeed, when working with high-dimensional data, it is always a good idea to *normalize* them.

A common approach is normalization to zero mean and unit variance. To make a long story short, for our current data this can be achieved like so

```
# normalize (column) data in matX to zero mean and unit variance
vecM = np.mean(matX, axis=1).reshape(m,1)
vecS = np.std (matX, axis=1).reshape(m,1)
matX = (matX - vecM) / vecS
```

Once you have normalized the data, you may quantize it using something along these lines

```
matW = vqMethod(matX, k, ...)
```

Note: If you now want to visualize the prototypes in the resulting codebook, you first need to de-normalize it

```
# de-normalize (column) data in matW
matW = matW * vecS + vecM
```

here are your tasks

Implement MacQueen's online k-means clustering as a reference method for extracting k prototypes from the data in `matX`. Experiment with different choices of k and visualize your results.

Use your MMD code from task 6.3 to extract k prototypes from the data in `matX`. Experiment with different choices of k and σ^2 and visualize your results.

As a reference, here is what your instructors got when working with $k = 16$

initial codebook `matW=matX[:,:k]`

MacQueen result

initial codebook `matW=matX[:,:k]`

MMD result, $\sigma^2 = \frac{m}{2 \cdot 5}, t_{\text{max}} = 100$

learning opportunity

Looking at the above results, there are a few things for you to think about:

1. In either case (for MacQueen and MMD), some initial prototypes have barely changed, i.e. moved to a different position in data space. Why or how or under which circumstances would this happen?

2. Can you think of a reason for why we have chosen the MMD σ^2 to depend on the data dimensionality m?

   Hint: Think about distances in high dimensional spaces. For instance, consider the corners of the hypercue $\{0,1\}^m$. The two corners which are farthest apart in this cube are $\mathbf{0}_m$ and $\mathbf{1}_m$. What happens to their distance $\|\mathbf{0}_m - \mathbf{1}_m\|$ if m is growing? What does that tell you about the value of a Gaussian kernel

   $$\phi_\sigma(\boldsymbol{u}, \boldsymbol{v}) = \exp\left(-\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{2\,\sigma^2}\right)$$

   where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$ and the dimensionality m is (fairly) large?

task 6.6

submission of presentation and code

As always, prepare a presentation / set of slides on your solutions for all the mandatory tasks and submit them to eCampus. Also as always, put your code into a ZIP file and submit it to eCampus.