exercise 4

fun with Frank-Wolfe

solutions due

until **December 9, 2025** at **23:30** via **ecampus**

general remarks

The following sheets contain a lot of text with a lot of “unconventional” information not found in the books. All this text is to provide you with the theoretical background of the tasks you are supposed to solve.

These tasks will all be about applications of the Frank-Wolfe algorithm for convex optimization over convex sets. In fact, they are (almost) all about computing global or local means of data sets. While you may be familiar with 1-mean or k-means computation, it is likely that you will not yet have seen the methods we employ here.

If you work carefully through this exercise, you will gain incredibly deep insights into the nature of k-means clustering. This should be fun and will come in handy later in the course . . .

task 4.1 [5 points]

sample means (part 1)

Consider a sample $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ of n data points $\boldsymbol{x}_j \in \mathbb{R}^m$ which we may gather in a data matrix

$$\boldsymbol{X} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{m \times n}$$

In this task, we are interested in estimating the mean $\hat{\boldsymbol{\mu}} \in \mathbb{R}^m$ of such a sample. To begin with, we recall a lesser known fact, namely that the familiar maximum likelihood estimator

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j$$

is the minimizer of the following convex function

$$f(\boldsymbol{\mu}) = \frac{1}{n} \sum_{j=1}^{n} \left\| \boldsymbol{x}_j - \boldsymbol{\mu} \right\|^2 = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j^\intercal \boldsymbol{x}_j - 2\, \boldsymbol{x}_j^\intercal \boldsymbol{\mu} + \boldsymbol{\mu}^\intercal \boldsymbol{\mu}$$

In other words, the sample mean of $\mathcal{X} \subset \mathbb{R}^m$ is the particular point $\hat{\boldsymbol{\mu}} \in \mathbb{R}^m$ that has the smallest average distance to all the points contained in $\mathcal{X}$.

background info

Let's quickly verify that

$$\hat{\boldsymbol{\mu}} = \operatorname*{argmin}_{\boldsymbol{\mu}} f(\boldsymbol{\mu})$$

We have the gradient

$$\nabla f = \frac{df}{d\boldsymbol{\mu}} = \frac{1}{n} \sum_{j=1}^{n} 2\, \boldsymbol{\mu} - 2\, \boldsymbol{x}_j$$

which, when equated to $\mathbf{0}$ (and then divide by 2), yields the equivalencies

$$\frac{1}{n} \sum_{j=1}^{n} \boldsymbol{\mu} = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \quad \Leftrightarrow \quad \frac{1}{n}\, n\, \boldsymbol{\mu} = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \quad \Leftrightarrow \quad \boldsymbol{\mu} = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j$$

Now, given what we saw in earlier exercises, we realize that we can also write the sample mean like this

$$\hat{\boldsymbol{\mu}} = \tfrac{1}{n}\,\boldsymbol{X}\mathbf{1} \equiv \boldsymbol{X}\hat{\boldsymbol{w}}$$

where the entries of vector $\hat{\boldsymbol{w}} = \frac{1}{n}\mathbf{1} \in \mathbb{R}^n$ are non-negative and sum to one

$$\begin{aligned} \hat{\boldsymbol{w}} &\geq \mathbf{0} \\ \mathbf{1}^\intercal\hat{\boldsymbol{w}} &= 1 \end{aligned}$$

But this is then to say that we may also compute the sample mean using a rather elaborate two-step procedure, namely

Step 1:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \;\; \frac{1}{n}\sum_{j=1}^{n}\big\|\boldsymbol{x}_j - \boldsymbol{X}\boldsymbol{w}\big\|^2 \qquad (1)$$

$$\text{s.t.} \quad \begin{aligned} \mathbf{1}^\intercal\boldsymbol{w} &= 1 \\ \boldsymbol{w} &\geq \mathbf{0} \end{aligned}$$

Step 2:

$$\hat{\boldsymbol{\mu}} = \boldsymbol{X}\hat{\boldsymbol{w}} \qquad (2)$$

Below, your task will be to implement this procedure. Indeed, based on what we did in earlier exercises, everybody should be able to solve the problem in (1) using `scipy.otimize` methods. However, there also is another, arguably better way ...

To see this, we first of all note that the objective function of the constrained minimization problem in (1) is

$$g(\boldsymbol{w}) = \frac{1}{n}\sum_{j=1}^{n}\big\|\boldsymbol{x}_j - \boldsymbol{X}\boldsymbol{w}\big\|^2 = \frac{1}{n}\sum_{j=1}^{n}\left(\boldsymbol{x}_j^\intercal\boldsymbol{x}_j - 2\,\boldsymbol{x}_j^\intercal\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w}\right)$$

for which we have the following gradient

$$\begin{aligned} \nabla g = \frac{dg}{d\boldsymbol{w}} &= \frac{1}{n}\sum_{j=1}^{n}\left(2\,\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} - 2\,\boldsymbol{X}^\intercal\boldsymbol{x}_j\right) \\ &= \frac{2}{n}\,n\,\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} - \frac{2}{n}\,\boldsymbol{X}^\intercal\sum_{j=1}^{n}\boldsymbol{x}_j \\ &= 2\,\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} - \frac{2}{n}\,\boldsymbol{X}^\intercal\boldsymbol{X}\mathbf{1} \\ &= 2\,\boldsymbol{X}^\intercal\boldsymbol{X}\left[\boldsymbol{w} - \tfrac{1}{n}\,\mathbf{1}\right] \end{aligned} \qquad (3)$$

Second of all, we note that any $\boldsymbol{w} \in \mathbb{R}^n$ which obeys the constraints $\boldsymbol{w} \geq \mathbf{0}$ and $\mathbf{1}^\intercal \boldsymbol{w} = 1$ is an element of the standard simplex

$$\Delta^{n-1} = \left\{ \sum_{i=1}^{n} w_i \, \boldsymbol{e}_i \,\middle|\, \boldsymbol{w} \in \mathbb{R}^n,\ \boldsymbol{w} \geq \mathbf{0},\ \mathbf{1}^\intercal \boldsymbol{w} = 1 \right\}$$

where $\boldsymbol{e}_i$ denotes the i-th standard basis vector of $\mathbb{R}^n$.

Third of all, since Δ^{n-1} is a *compact convex set* and $g(\boldsymbol{w})$ is a convex objective function, we realize that (1) constitutes a convex minimization problem over a compact convex set and can thus be solved by running the following (problem specific) version of the Frank-Wolfe algorithm

procedure FW_SAMPLE_MEAN_WEIGHTS($\boldsymbol{X} \in \mathbb{R}^{m \times n}, t_{\text{max}} \in \mathbb{N}$)

guess a feasible $\boldsymbol{w}_0 \in \Delta^{n-1}$, say
$\boldsymbol{w}_0 = \boldsymbol{e}_1$

for $t = 0, \ldots, t_{\text{max}} - 1$

determine step direction
$\boldsymbol{s}_t = \underset{\boldsymbol{s} \in \Delta^{n-1}}{\operatorname{argmin}} \ \boldsymbol{s}^\intercal \nabla g(\boldsymbol{w}_t)$

determine step size
$\eta_t = \frac{2}{t+2}$

update current guess
$\boldsymbol{w}_{t+1} = \big(1 - \eta_t\big) \, \boldsymbol{w}_t + \eta_t \, \boldsymbol{s}_t$

return $\boldsymbol{w}_{t+1}$

Note: $\boldsymbol{s}^\intercal \nabla g(\boldsymbol{w}_t)$ is a linear function over Δ^{n-1}. Since Jensen's inequality tells us that he minimum of a convex function over a convex set is attained at a of said vertex, we have

$$\boldsymbol{s}_t = \underset{\boldsymbol{s} \in \Delta^{n-1}}{\operatorname{argmin}} \ \boldsymbol{s}^\intercal \nabla g(\boldsymbol{w}_t) \quad \Leftrightarrow \quad \boldsymbol{s}_t = \underset{\boldsymbol{e}_i \in \mathbb{R}^n}{\operatorname{argmin}} \ \boldsymbol{e}_i^\intercal \nabla g(\boldsymbol{w}_t)$$

an you should use the very convenient fact in your solutions for this task.

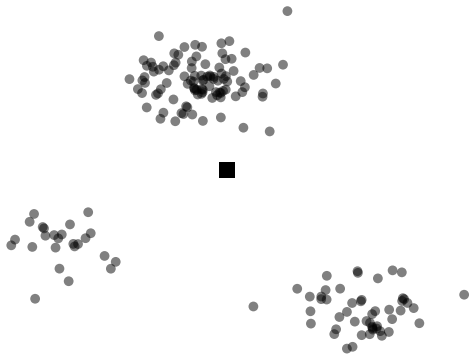
here is your task

The file

```
threeBlobs.csv
```

contains a data matrix $\boldsymbol{X} \in \mathbb{R}^{2\times 175}$ of 175 data points $\boldsymbol{x}_j \in \mathbb{R}^2$. Read this matrix into memory and —to get a ground truth result— (properly) use the *numpy* function *mean* to compute the sample mean $\bar{\boldsymbol{x}}$ of the $\boldsymbol{x}_j$. Have a look at the result.

Next, using the gradient in (3), implement the Frank-Wolfe algorithm for solving the problem in (1). Run the algorithm for $T = 10000$ iterations to get an estimate of $\hat{\boldsymbol{w}} \in \mathbb{R}^{175}$. Given $\hat{\boldsymbol{w}}$, compute $\hat{\boldsymbol{\mu}} = \boldsymbol{X}\hat{\boldsymbol{w}}$ and plot the data (say as round dots) together with your estimated mean (say as a square). Your plot should look something like this



Finally, here is a statement w/o proof: "After T iterations, the Frank-Wolfe algorithm will produce a solution that is $O\big(\frac{1}{T}\big)$ away from an optimum".

Practically investigate what this means! Proceed as follows: Run your Frank-Wolfe implementation for $T \in \{10, 100, 1000, 10000\}$ iterations. This should result in different $\hat{\boldsymbol{w}}$ and therefore in different mean estimates $\hat{\boldsymbol{\mu}} = \boldsymbol{X}\hat{\boldsymbol{w}}$. Compare each of these mean estimates to the ground truth $\bar{\boldsymbol{x}}$ which you computed above. Discuss what you observe.

task 4.2 [5 points]

sample means (part 2)

Let $\boldsymbol{x}_j$, $\boldsymbol{X}$, $\boldsymbol{\mu}$, $\hat{\boldsymbol{\mu}}$, $\boldsymbol{w}$, $\hat{\boldsymbol{w}}$, and $\mathbf{1}$ be as in the previous task and (try to) convince yourself that the following equivalencies hold true

$$\sum_{j=1}^{n} \big\|\boldsymbol{x}_j - \boldsymbol{\mu}\big\|^2 \quad \Leftrightarrow \quad \big\|\boldsymbol{X} - \boldsymbol{\mu}\mathbf{1}^\intercal\big\|^2 \quad \Leftrightarrow \quad \big\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big\|^2$$

Working with the rightmost expression, we can thus compute the mean $\hat{\boldsymbol{\mu}}$ of the columns $\boldsymbol{x}_j$ of $\boldsymbol{X}$ using this elaborate two-step procedure

Step 1:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \big\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big\|^2 \quad \text{s.t.} \quad \begin{aligned} \mathbf{1}^\intercal\boldsymbol{w} &= 1 \\ \boldsymbol{w} &\geq \mathbf{0} \end{aligned} \tag{4}$$

Step 2:

$$\hat{\boldsymbol{\mu}} = \boldsymbol{X}\hat{\boldsymbol{w}} \tag{5}$$

Indeed, the objective of the minimization problem in (4) can be written as

$$\begin{aligned} h(\boldsymbol{w}) &= \big\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big\|^2 \\ &= \operatorname{tr}\Big[\big[\boldsymbol{X} - \boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big]^\intercal\big[\boldsymbol{X} - \boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big]\Big] \\ &= \operatorname{tr}\big[\boldsymbol{X}^\intercal\boldsymbol{X} - 2\,\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal + \mathbf{1}\boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big] \\ &= cnst - 2\,\operatorname{tr}\big[\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big] + \operatorname{tr}\big[\mathbf{1}\boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w}\mathbf{1}^\intercal\big] \end{aligned}$$

At this point, we can resort to two results which you are supposed to prove below. The two traces can be rewritten such that our objective function reads

$$\begin{aligned} h(\boldsymbol{w}) &= cnst - 2\,\mathbf{1}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} + \mathbf{1}^\intercal\mathbf{1}\boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} \\ &= n\,\boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} - 2\,\mathbf{1}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} + cnst \end{aligned}$$

This, in turn, gives us the following gradient

$$\nabla h = \frac{dh}{d\boldsymbol{w}} = 2\,n\,\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} - 2\,\boldsymbol{X}^\intercal\boldsymbol{X}\mathbf{1}$$

Convince yourself that equating ∇h to $\mathbf{0}$ and then solving for $\boldsymbol{w}$ leads to the familiar result

$$\hat{\boldsymbol{w}} = \tfrac{1}{n}\left[\boldsymbol{X}^\intercal \boldsymbol{X}\right]^{-1} \boldsymbol{X}^\intercal \boldsymbol{X}\mathbf{1} = \tfrac{1}{n}\,\mathbf{1}$$

If you are wondering why we derived this result yet again, then note that it was to familiarize ourselves with the kind of linear algebra needed in the subsequent tasks ...

here is your task

Let $\boldsymbol{X}$ and $\boldsymbol{w}$ be as above and $\boldsymbol{z} \in \mathbb{R}^n$ be any n-dimensional vector. Prove the following two identities[1]

$$\operatorname{tr}\left[\boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w}\boldsymbol{z}^\intercal\right] = \boldsymbol{z}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w}$$

$$\operatorname{tr}\left[\boldsymbol{z}\boldsymbol{w}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w}\boldsymbol{z}^\intercal\right] = \boldsymbol{z}^\intercal \boldsymbol{z} \cdot \boldsymbol{w}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w}$$

[1] In case you are tackling this task with an AIs please tell us about your experiences! Did it work? Did the AI reason correctly?

task 4.3 [10 points]

finding maximally different data points

In the previous tasks, you merely had to do what the problem specifications asked you to do. In this task, you need to get creative!

Let $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ be a sample of n data points $\boldsymbol{x}_j \in \mathbb{R}^m$ which we may gather in a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$. Now, *invent* an algorithm that selects $2 \leq k < n$ of the n columns of $\boldsymbol{X}$ such that the selected data vectors are as far apart as possible.

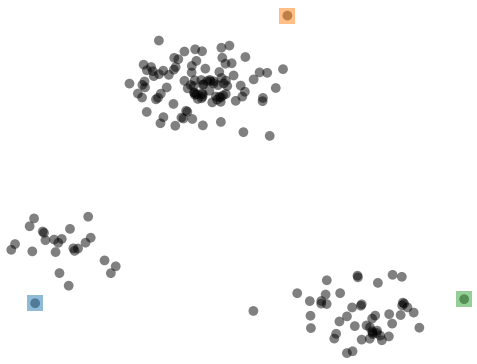
Mathematically, the problem considered in this task is a subset selection problem and we can formalize it as follows: given $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, solve

$$\hat{\mathcal{S}} = \operatorname*{argmax}_{\mathcal{S} \subset \mathcal{X}} \sum_{\boldsymbol{x}_i \in \mathcal{S}} \sum_{\boldsymbol{x}_j \in \mathcal{S}} \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2$$
$$\text{s.t.} \quad |\mathcal{S}| = k$$

Note: this problem is actually way more difficult than it may appear at first sight; in fact, it is NP-hard and an *efficient* algorithm that is *guaranteed* to find the optimal $\hat{\mathcal{S}}$ for very large sets $\mathcal{X}$ and arbitrary choices of k remains elusive to this date. Habving said this, one can definitely come up with efficient and well working heuristics and your instructors are looking forward to your ideas ...

task 4.3.1 [5 points]

Implement you algorithm and run it on the data in file `threeBlobs.csv`. Test your algorithm for several choices of k, say $k \in \{3, 4, 5\}$, and plot the data (say as black dots) as well as the selected columns (say as colored squares). For instance, for $k = 3$, your result could look like this:



task 4.3.2 [5 points]

Next, run your algorithm on some more interesting data. The file

`faceMatrix.npy`

contains a data matrix $\boldsymbol{X} \in \mathbb{R}^{361 \times 2429}$ whose columns $\boldsymbol{x}_j$ represent tiny face images of size 19×19 pixels. To read this matrix into memory and check its size, you may use

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

To have a look at one of the images it contains, say $\boldsymbol{x}_{15}$, you may use this

```
vecX = matX[:,14].reshape(19,19)
plt.imshow(vecX, cmap='gray')
plt.xticks([])
plt.yticks([])
plt.show()
```

Run your algorithm on this new data matrix $\boldsymbol{X}$ and consider several choices for k, say $k \in \{4, 9, 16, 25, 49, 100\}$. Visualize the extracted columns in terms of tiny images. For instance, for $k = 25$, your result could look like this:

Note: We ordered the extracted maximally diverse data points / images according to their overall brightness. You do not have to this. But if your results look completely different from ours, your algorithm / implementation may have issues.

task 4.4 [10 points]

unconventional k-means clustering (part 1)

Note: the following background material is for those who are not familiar with k-means clustering. If you know about it, you may safely skip ahead.

background

k-means clustering is without a doubt the most popular idea for clustering a data set $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^m$ into k clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k \subset \mathcal{X}$ which are disjoint, i.e. $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for $i \neq j$, and cover the data, i.e. $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \ldots \cup \mathcal{C}_k = \mathcal{X}$. The two defining properties of k-means clustering are

a) it represents each cluster $\mathcal{C}_i$ in terms of its *centroid* or mean $\boldsymbol{\mu}_i \in \mathbb{R}^m$

b) it clusters according to Euclidean distances between centroids and data points; that is, a point $\boldsymbol{x}_j$ will be assigned to cluster $\mathcal{C}_i$ if its distance to $\boldsymbol{\mu}_i$ is less than that to any other centroid.

A common formalization of the k-means clustering problem is as follows

$$\hat{\boldsymbol{\mu}}_1, \ldots, \hat{\boldsymbol{\mu}}_k = \underset{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in \mathcal{C}_i} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2 \tag{6}$$

Observe that this poses a chicken-and-egg problem: in order to compute cluster means, we need clusters and in order to compute clusters, we need cluster means.

Further observe that k-means clustering is NP-hard so that any presently known algorithm for solving (6) is but a heuristic. A well known k-means heuristic is Lloyd's algorithm

"randomly" initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

repeat
  for $i = 1, \ldots, k$
    $\mathcal{C}_i = \left\{ \boldsymbol{x} \in \mathcal{X} \,\middle|\, \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \leq \|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2 \right\}$ // compute clusters

  for $i = 1, \ldots, k$
    $\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \boldsymbol{x}$ // update means

until convergence

which is used by the k-means functions available in `scipuy.cluster.vq`.

an alternative k-means formalization

The objective function in (6) can be expressed differently. Indeed, (try to) convince yourself that we have the following equivalence

$$\sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in \mathcal{C}_i} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2 \quad \Leftrightarrow \quad \sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} \cdot \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2$$

where the additional variables $z_{ij} \in \{0, 1\}$ are indicator variables such that

$$z_{ij} = \begin{cases} 1, & \text{if } \boldsymbol{x}_j \in \mathcal{C}_i \\ 0, & \text{otherwise} \end{cases}$$

The k-means clustering problem can therefore also be formalized as

$$\begin{aligned} \hat{\boldsymbol{\mu}}_1, \ldots, \hat{\boldsymbol{\mu}}_k = \underset{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k}{\operatorname{argmin}} & \sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2 \\ & z_{ij} \in \{0, 1\} \\ \text{s.t.} & \sum_{i=1}^{k} z_{ij} = 1 \end{aligned} \tag{7}$$

where the sum-to-one constraints enforce the goal of assigning each data point $\boldsymbol{x}_j$ to only one cluster $\mathcal{C}_i$.

an unconventional k-means formalization

Next, things will get more interesting. One can prove[2] the following equality

$$\sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2 = \left\| \boldsymbol{X} - \boldsymbol{M}\boldsymbol{Z} \right\|^2$$

where

$\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is a matrix of data vectors $\boldsymbol{x}_j \in \mathbb{R}^m$

$\boldsymbol{M} \in \mathbb{R}^{m \times k}$ is a matrix of cluster centroids $\boldsymbol{\mu}_i \in \mathbb{R}^m$

$\boldsymbol{Z} \in \mathbb{R}^{k \times n}$ is a matrix of binary indicator variables

$$Z_{ij} = \begin{cases} 1, & \text{if } \boldsymbol{x}_j \in \mathcal{C}_i \\ 0, & \text{otherwise.} \end{cases}$$

[2] C. Bauckhage, "k-Means Clustering Is Matrix Factorization", arXiv:1512.07548, 2015

Note that we have two crucial restrictions on matrix $\boldsymbol{Z}$. First of all, it must be binary, i.e. $\boldsymbol{Z} \in \{0,1\}^{k \times n}$. Second of all, since each data point can only belong to one cluster, each of its columns must sum to one. If we introduce the vectors $\mathbf{1}_k$ and $\mathbf{1}_n$ of all ones in $\mathbb{R}^k$ and $\mathbb{R}^n$, respectively, then this second constrained can be formalized as $\mathbf{1}_k^\intercal \boldsymbol{Z} = \mathbf{1}_n^\intercal$.

With all this in place, we can also formalize the k-means problem as

$$\begin{aligned} \hat{\boldsymbol{M}}, \hat{\boldsymbol{Z}} = \operatorname*{argmin}_{\boldsymbol{M},\boldsymbol{Z}} \; & \big\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{Z}\big\|^2 \\ & \boldsymbol{Z} \in \{0,1\}^{k \times n} \\ \text{s.t.} \quad & \boldsymbol{Z} \geq \mathbf{0}_{k \times n} \\ & \mathbf{1}_k^\intercal \boldsymbol{Z} = \mathbf{1}_n^\intercal \end{aligned} \tag{8}$$

where we write $\mathbf{0}_{k \times n}$ to denote the $k \times n$ matrix of all zeros.

Speaking of this matrix, we are well aware that the constraints $\boldsymbol{Z} \in \{0,1\}^{k \times n}$ and $\boldsymbol{Z} \geq \mathbf{0}_{k \times n}$ are redundant. Our reason for introducing this redundancy will become apparent later ...

For the objective function in (8), we have the following gradients w.r.t. $\boldsymbol{Z}$ and $\boldsymbol{M}$

$$\begin{aligned} \nabla_{\boldsymbol{Z}} \big\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{Z}\big\|^2 &= \boldsymbol{G}_{\boldsymbol{Z}} = 2\left[\boldsymbol{M}^\intercal \boldsymbol{M}\boldsymbol{Z} - \boldsymbol{M}^\intercal \boldsymbol{X}\right] \\ \nabla_{\boldsymbol{M}} \big\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{Z}\big\|^2 &= \boldsymbol{G}_{\boldsymbol{M}} = 2\left[\boldsymbol{M}\boldsymbol{Z}\boldsymbol{Z}^\intercal - \boldsymbol{X}\boldsymbol{Z}^\intercal\right] \end{aligned}$$

Equating these to zero and solving for $\boldsymbol{Z}$ and $\boldsymbol{M}$, respectively, we find

$$\begin{aligned} \boldsymbol{Z} &= \left[\boldsymbol{M}^\intercal \boldsymbol{M}\right]^{-1} \boldsymbol{M}^\intercal \boldsymbol{X} \\ \boldsymbol{M} &= \boldsymbol{X}\boldsymbol{Z}^\intercal \left[\boldsymbol{Z}\boldsymbol{Z}^\intercal\right]^{-1} \end{aligned}$$

Naïvely thinking, this would thus suggest the following k-means algorithm

"randomly" initialize $\boldsymbol{M}$

for $T = 1, \ldots, T_{\text{max}}$

$\boldsymbol{Z} = \left[\boldsymbol{M}^\intercal \boldsymbol{M}\right]^{-1} \boldsymbol{M}^\intercal \boldsymbol{X}$ // compute clusters

$\boldsymbol{M} = \boldsymbol{X}\boldsymbol{Z}^\intercal \left[\boldsymbol{Z}\boldsymbol{Z}^\intercal\right]^{-1}$ // update means

Alas, this naïve algorithm will produce unreasonable results (try it) since it does not pay attention to the constraints on $\boldsymbol{Z}$. So how can we incorporate those?

Assuming we had reasonable guesses for $\boldsymbol{M}$ and $\boldsymbol{Z}$, we can generalize what we learned above and observe that

$$t_{\text{max}} = 1$$

iterations of the following Frank-Wolfe procedure

procedure FW_UPDATE_Z($\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n], t_{\text{max}}$)

for $t = 0, \ldots, t_{\text{max}} - 1$

$\boldsymbol{G}_{\boldsymbol{Z}} = 2\left[\boldsymbol{M}^\intercal \boldsymbol{M}\boldsymbol{Z} - \boldsymbol{M}^\intercal \boldsymbol{X}\right]$

for $j = 1, \ldots, n$

$o = \text{argmin}_i \left[\boldsymbol{G}_{\boldsymbol{Z}}\right]_{ij}$

$\boldsymbol{z}_j = \boldsymbol{z}_j + \frac{2}{t+2}\left[\boldsymbol{e}_o - \boldsymbol{z}_j\right]$

return $\boldsymbol{Z}$

will "naturally" produce a matrix $\boldsymbol{Z}$ that meets the constraints in problem 8. Based on this insight, we can then use the following procedure for k-means clustering

procedure FW_KMEANS_VERSION1($\boldsymbol{X} \in \mathbb{R}^{m \times n}, k, T_{\text{max}}$)

"randomly" initialize matrix $\boldsymbol{M} \in \mathbb{R}^{m \times k}$

for $T = 0, \ldots, T_{\text{max}} - 1$

$\boldsymbol{Z} = \frac{1}{k}\,\mathbf{1}_{k \times n}$

$\boldsymbol{Z} = \text{FW\_UPDATE\_Z}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{Z}, t_{\text{max}} = 1)$

$\boldsymbol{M} = \boldsymbol{X}\boldsymbol{Z}^\intercal \left[\boldsymbol{Z}\boldsymbol{Z}^\intercal\right]^{-1}$

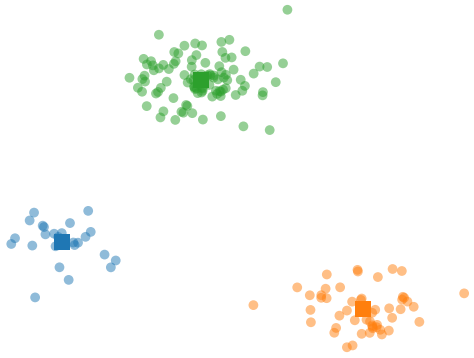
return $\boldsymbol{M}, \boldsymbol{Z}$

where we write $\mathbf{1}_{k \times n}$ to denote the $k \times n$ matrix of all ones.

Finally ...

task 4.4.1 [8 points]

Implement functions *FW_update_Z* and *FW_kMeans_Version1* and test this approach to k-means clustering on the data in file `threeBlobs.csv`. Visualize your clustering results. For example, for $k = 3$, you results could look like this



Note: if you are clever, you can avoid the *for* loop over the j in function *FW_update_Z*. Spend some time on thinking of how this could be achieved; your code will be much faster . . .

Note: k-means clustering is often very sensitive w.r.t. the initialization of the cluster centers $\boldsymbol{M} = [\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k]$. A reasonable idea is to randomly select k columns of the data matrix $\boldsymbol{X}$.

However . . . don't be surprised if your program crashes. What we are doing in procedure FW_KMEANS_VERSION1 involves matrix inversion which can sometimes lead to numerical instabilities. A more elaborate but generally safer idea for initializing $\boldsymbol{M}$ would be to use your code from task 4.3 to extract k very diverse columns from $\boldsymbol{X}$.

task 4.4.2 [2 points]

Try to run your code on the data in `faceMatrix.npy` and visualize the resulting mean faces. The numerical stability issues we mentioned above may be a headache here. Nevertheless, using initialization with maximally diverse data points, your instructors managed to get the following result for $k = 16$

task 4.5 [10 points]

unconventional k-means clustering (part 2)

Let's dial things up a notch! We just considered the objective $\left\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{Z}\right\|^2$ and found that $\boldsymbol{M}$ can be written as $\boldsymbol{M} = \boldsymbol{X}\boldsymbol{Z}^\intercal\left[\boldsymbol{Z}\boldsymbol{Z}^\intercal\right]^{-1}$. But this is to say that we may also formalize the k-means clustering problem as

$$\begin{aligned}\hat{\boldsymbol{Z}} = \underset{\boldsymbol{Z}}{\operatorname{argmin}}\ &\left\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{Z}^\intercal\left[\boldsymbol{Z}\boldsymbol{Z}^\intercal\right]^{-1}\boldsymbol{Z}\right\|^2 \\ &\boldsymbol{Z} \in \{0,1\}^{k\times n} \\ \text{s.t.}\quad &\boldsymbol{Z} \geq \boldsymbol{0}_{k\times n} \\ &\boldsymbol{1}_k^\intercal\boldsymbol{Z} = \boldsymbol{1}_n^\intercal\end{aligned} \tag{9}$$

If we could solve this problem, we could finally obtain $\hat{\boldsymbol{M}} = \boldsymbol{X}\hat{\boldsymbol{Z}}^\intercal\left[\hat{\boldsymbol{Z}}\hat{\boldsymbol{Z}}^\intercal\right]^{-1}$. Alas, the problem in (10) is a truly formidable problem (to see why, just try to compute the gradient w.r.t. $\boldsymbol{Z}$).

There are complicated ways for addressing this issue, but there also is a relatively simple one. We may introduce the $n \times k$ matrix

$$\boldsymbol{Y} = \boldsymbol{Z}^\intercal\left[\boldsymbol{Z}\boldsymbol{Z}^\intercal\right]^{-1}$$

for which one can show the following: If $\boldsymbol{Z}$ is stochastic ($\boldsymbol{Z} \geq \boldsymbol{0}_{k\times n}$ and $\boldsymbol{1}_k^\intercal\boldsymbol{Z} = \boldsymbol{1}_n^\intercal$), then $\boldsymbol{Y}$ is stochastic ($\boldsymbol{Y} \geq \boldsymbol{0}_{n\times k}$ and $\boldsymbol{1}_n^\intercal\boldsymbol{Y} = \boldsymbol{1}_k^\intercal$).

But this is to say that there is yet another formalization of the k-means problem, namely

$$\begin{aligned}\hat{\boldsymbol{Y}}, \hat{\boldsymbol{Z}} = \underset{\boldsymbol{Y},\boldsymbol{Z}}{\operatorname{argmin}}\ &\left\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{Y}\boldsymbol{Z}\right\|^2 \\ &\boldsymbol{Z} \in \{0,1\}^{k\times n} \\ &\boldsymbol{Z} \geq \boldsymbol{0}_{k\times n} \\ \text{s.t.}\quad &\boldsymbol{1}_k^\intercal\boldsymbol{Z} = \boldsymbol{1}_n^\intercal \\ &\boldsymbol{Y} \geq \boldsymbol{0}_{n\times k} \\ &\boldsymbol{1}_n^\intercal\boldsymbol{Y} = \boldsymbol{1}_k^\intercal\end{aligned} \tag{10}$$

For the objective function in (10), we have the following gradient w.r.t. $\boldsymbol{Y}$

$$\nabla_{\boldsymbol{Y}}\left\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{Y}\boldsymbol{Z}\right\|^2 = \boldsymbol{G}_{\boldsymbol{Y}} = 2\left[\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{Y}\boldsymbol{Z}\boldsymbol{Z}^\intercal - \boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{Z}^\intercal\right]$$

This allows us to design the following Frank-Wolfe procedure for updating $\boldsymbol{Y}$ based on reasonable guesses for $\boldsymbol{Y}$ and $\boldsymbol{Z}$ ...

procedure FW_UPDATE_Y($\boldsymbol{X}, \boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k], \boldsymbol{Z}, t_{\text{max}}$)

for $t = 0, \ldots, t_{\text{max}} - 1$

$\boldsymbol{G_Y} = 2\left[\boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{Y}\boldsymbol{Z}\boldsymbol{Z}^\intercal - \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{Z}^\intercal\right]$

for $i = 1, \ldots, k$

$o = \text{argmin}_j \left[\boldsymbol{G_Y}\right]_{ji}$

$\boldsymbol{y}_i = \boldsymbol{y}_i + \frac{2}{t+2}\left[\boldsymbol{e}_o - \boldsymbol{y}_i\right]$

return $\boldsymbol{Y}$

This, in turn, can be used in the following procedure for k-means clustering

procedure FW_KMEANS_VERSION2($\boldsymbol{X} \in \mathbb{R}^{m \times n}, k, T_{\text{max}}$)

"randomly" initialize matrix $\boldsymbol{M} \in \mathbb{R}^{m \times k}$

for $T = 0, \ldots, T_{\text{max}} - 1$

$\boldsymbol{Z} = \frac{1}{k}\,\mathbf{1}_{k \times n}$

$\boldsymbol{Z} = \text{FW\_UPDATE\_Z}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{Z}, t_{\text{max}} = 1)$

$\boldsymbol{Y} = \frac{1}{n}\,\mathbf{1}_{n \times k}$

$\boldsymbol{Y} = \text{FW\_UPDATE\_Y}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, t_{\text{max}} = 100)$

$\boldsymbol{M} = \boldsymbol{X}\boldsymbol{Y}$

return $\boldsymbol{M}, \boldsymbol{Y}, \boldsymbol{Z}$

where we observe that the t_{max} in the call of FW_UPDATE_Z differs from the t_{max} in the call of FW_UPDATE_Y.

We also observe that procedure FW_kMeans_Version2 does not involve matrix inversion steps anymore!!!!!

here is your task

Reuse your above code for function *FW_update_Z* and implement two new functions *FW_update_Y* and *FW_kMeans_Version2*. Test your combined new code on the data in file `threeBlobs.csv`. Visualize your clustering results. They should look like those in task 4.4.1.

Also test your code on the data in `faceMatrix.npy` and visualize the resulting mean faces. These should look like hose in task 4.4.2.

Overall, we expect that you will find that this approach towards k-means works more robustly than the one in task 4.4.

task 4.6 [10 points]

archetypal analysis

Above, we expressed the k-means clustering as the following constrained optimization problem

$$\begin{aligned} \hat{\boldsymbol{Y}}, \hat{\boldsymbol{Z}} = \underset{\boldsymbol{Y},\boldsymbol{Z}}{\operatorname{argmin}} \left\| \boldsymbol{X} - \boldsymbol{X}\boldsymbol{Y}\boldsymbol{Z} \right\|^2 \\ \boldsymbol{Z} \in \{0,1\}^{k \times n} \\ \boldsymbol{Z} \geq \boldsymbol{0}_{k \times n} \\ \text{s.t.} \quad \boldsymbol{1}_k^\intercal \boldsymbol{Z} = \boldsymbol{1}_n^\intercal \\ \boldsymbol{Y} \geq \boldsymbol{0}_{n \times k} \\ \boldsymbol{1}_n^\intercal \boldsymbol{Y} = \boldsymbol{1}_k^\intercal \end{aligned}$$

Now, let us ever so slightly but very profoundly change the problem to

$$\begin{aligned} \hat{\boldsymbol{Y}}, \hat{\boldsymbol{Z}} = \underset{\boldsymbol{Y},\boldsymbol{Z}}{\operatorname{argmin}} \left\| \boldsymbol{X} - \boldsymbol{X}\boldsymbol{Y}\boldsymbol{Z} \right\|^2 \\ \boldsymbol{Z} \geq \boldsymbol{0}_{k \times n} \\ \text{s.t.} \quad \boldsymbol{1}_k^\intercal \boldsymbol{Z} = \boldsymbol{1}_n^\intercal \\ \boldsymbol{Y} \geq \boldsymbol{0}_{n \times k} \\ \boldsymbol{1}_n^\intercal \boldsymbol{Y} = \boldsymbol{1}_k^\intercal \end{aligned} \tag{11}$$

Try to spot the difference! The problem in (11) is called archetypal analysis and is a powerful tool in data exploration. To solve it, we suggest

procedure FW_ARCHETYPAL_ANALYSIS($\boldsymbol{X} \in \mathbb{R}^{m \times n}, k, T_{\text{max}} = 100$)

“randomly” initialize matrix $\boldsymbol{A} \in \mathbb{R}^{m \times k}$

for $T = 0, \ldots, T_{\text{max}} - 1$

$\boldsymbol{Z} = \text{FW\_UPDATE\_Z}(\boldsymbol{X}, \boldsymbol{A}, \frac{1}{k}\,\boldsymbol{1}_{k \times n}, t_{\text{max}} = 100)$

$\boldsymbol{Y} = \text{FW\_UPDATE\_Y}(\boldsymbol{X}, \frac{1}{n}\,\boldsymbol{1}_{n \times k}, \boldsymbol{Z}, t_{\text{max}} = 100)$

$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{Y}$

return $\boldsymbol{A}, \boldsymbol{Y}, \boldsymbol{Z}$

Observe that parameter t_{max} in the call of FW_UPDATE_Z has changed!

here is your task

Reuse your above code for *FW_update_Z* and *FW_update_Y* and implement *FW_Archetypal_Analysis*. Test your combined code on the data in file `threeBlobs.csv`. Plot the data in $\boldsymbol{X}$ (say as black dots) and the archetypes in $\boldsymbol{A}$ (say as colored squares). Also test your code on the data in `faceMatrix.npy` and visualize the resulting archetypal faces.

We won't spoil the fun and show how your results should look like. But we suggest you (visually) compare them against your results from tasks 4.3 and 4.5 . . . What do you observe?

task 4.7

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.