Principles of Machine Learning: Exercise 4

Alina Pollehn (3197257), Julian Litz (3362592), Manuel Hinz (3334548)
Felix Göhde (3336445), Felix Lehmann (3177181), Caspar Wiswesser (3221493)
Adrian Köring (3347785), Greta Günther (3326765), Linus Mallwitz (3327653)
Niklas Mueller-Goldingen (3363219), Jennifer Kroppen (2783393)

18.12.2023

Exercise 4.1: Overview

1. Goal: Minimize $f(\mu) := \frac{1}{n} \sum_{j=1}^{n} \|x_j - \mu\|$
2. Alternative formulation: Find $w \geq 0$ s.t. $1^\intercal w = 1$ and

$$w = \underset{w}{\operatorname{argmin}} \underbrace{\frac{1}{n} \sum_{j=1}^{n} \|x_j - Xw\|^2}_{=:g(w)}$$

3. We want to use the Frank-Wolfe algorithm to solve our optimization problem, therefore we also need the gradient:

$$\nabla g = 2X^\intercal X[w - \frac{1}{n}1]$$

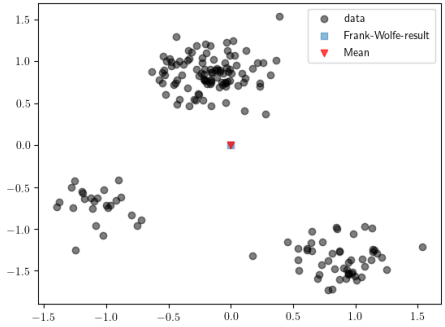
Exercise 4.1: Implementation

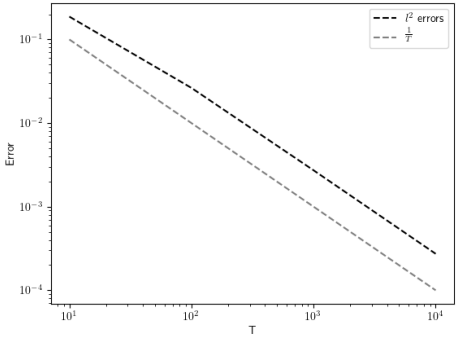
```
def grad_g(w,n):
    vecW=np.ones(n)/n
    return 2*X.T@X@(w-vecW)

def frank_wolfe_minimize_avg_dist(X,T=10000):
    n=X[0,:].shape[0]
    wt=np.zeros(n)
    wt[0]=1
    for t in range(T):
        gamma_t=2/(t+2)
        imin=np.argmin(grad_g(wt,n))
        wt *= gamma_t
        wt[imin] += gamma_t
    return X@wt
```

Exercise 4.1: Error analysis

Because we know the minimizer (which is just the mean of our vectors), we can calculate our error $\epsilon := \|\hat{\mu} - \mu\|$ as a function of the number of iterations. We also know that the error should be in $O(\frac{1}{T})$.





We also ran least squares to estimate the exponent similarly to Task 1-3. This yields $-1.05 \approx -1$, which also consistent with our theoretical error bound.

Exercise 4.2: Proving two identities

Let X be our data matrix and $w = \frac{1}{n}1_n$ and $z \in \mathbb{R}^n$.
Then the following identities hold:

$$\mathrm{tr}[\overbrace{X^\intercal X}^{A_1} wz^\intercal] = z^\intercal X^\intercal Xw$$

$$\mathrm{tr}[\underbrace{zw^\intercal X^\intercal X}_{A_2} wz^\intercal] = z^\intercal z \cdot w^\intercal X^\intercal Xw = z^\intercal (zw^\intercal X^\intercal X)w$$

It suffices to show the following lemma (using either $A = A_1$ or $A = A_2$):

Lemma 1

For a matrix $A \in \mathbb{R}^{n\times n}, w = \frac{1}{n}1_n \in \mathbb{R}^n$ and $z \in R^n$ the following equality holds:

$$\mathrm{tr}(Awz^\intercal) = z^\intercal Aw$$

Proof of Lemma 1

Proof.

1. First we use $\operatorname{tr}(A \overbrace{wz^\intercal}^{\in \mathbb{R}^{n\times n}}) = \operatorname{tr}(wz^\intercal A)$ (because we apply a cyclic permutation)
2. Now $a := z^\intercal A$ is just a row vector, therefore

$$wa = \begin{pmatrix} \frac{a_1}{n} & \cdots & \frac{a_n}{n} \\ \vdots & \vdots & \vdots \\ \frac{a_1}{n} & \cdots & \frac{a_n}{n} \end{pmatrix} \in \mathbb{R}^{n\times n}$$

$$\implies \operatorname{tr}(wa) = \sum_{i=1}^{n} \frac{a_i}{n} = \sum_{i=1}^{n} \frac{a_i}{n} \cdot 1 = \sum_{i=1}^{n} a_i \frac{(1_n)_i}{n}$$

$$= (z^T A)w$$

□

Exercise 4.3: Overview

1. Given n data points $(x_j)_{j=1}^n, x_j \in \mathbb{R}^m$ find k points which are maximally different:

$$\hat{S} = \underset{S \subset X,\ |S|=k}{\operatorname{argmax}} \sum_{x_i \in S} \sum_{x_j \in S} \|x_i - x_j\|^2$$

2. Two approaches
   - Farthest first: Iteratively select points, s.t. the minimum of the distances to the previously selected points is maximized
   - Repeatedly drawing random subsets of size k and only keeping the maximal subset w.r.t the objective function
3. Results
   - The first and last algorithm give similar results w.r.t to the objective function, but look very different.
   - Converges in probability for fixed n, k

Comparing our two main approaches

- Greedy:
  - Rather fast
  - Harder to justify why it works
- Repeated drawing:
  - Run time determines expected error
  - Almost surely converges for fixed n, k
  - Randomized algorithm $\implies$ outliers are unlikely, but can happen
  - Theory: Order statistics are well understood
- The greedy approach is stable and fast
- Repeated drawing is random, slower, but comes with a strong theoretical foundation and convergence

Farthest first approach: Code

```python
import random
def farthest_first_traversal(X, num_points=3):
    data = X.T
    subset = [random.choice(data)] # Select the first point randomly

    while len(subset) < num_points:
        # Calculate vectorized distance matrix between X and subset S
        diff = data[np.newaxis, ...] - np.array(subset)[:, np.newaxis, :]  # N, S, 2
        dist = np.linalg.norm(diff, axis=-1)                               # N, S

        min_distances = dist.min(axis=0) # distance determined via nearest subset point
        subset.append(data[np.argmax(min_distances)])  # point furthest away

    return np.array(subset).T
```

Randomized drawing: Code

```
def get_random_sample(X,k):
    tmp=X[:,np.random.choice(X.shape[1], k, replace=False)]
    return tmp, calc_norm(tmp)

def get_pred(X,k,runs):
    max_sample,max_norm = get_random_sample(X,k)
    for _ in range(runs-1):
        sample, sample_norm=get_random_sample(X,k)
        if sample_norm>max_norm:
            max_sample,max_norm =  sample, sample_norm
    return max_sample, max_norm
```

Exercise 4.3: Results: Blobs

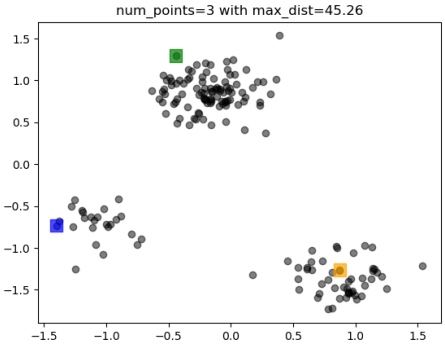


Figure: Farthest first result: Blobs

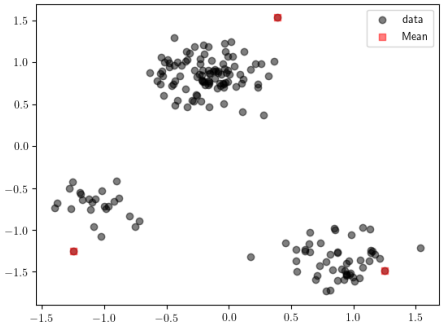


Figure: Randomized drawing result: Blobs

Exercise 4.3: Results: Faces

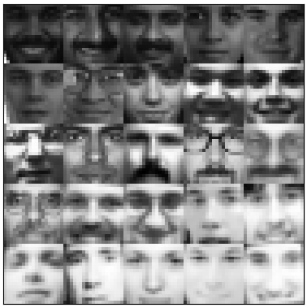


Figure: Randomized drawing results

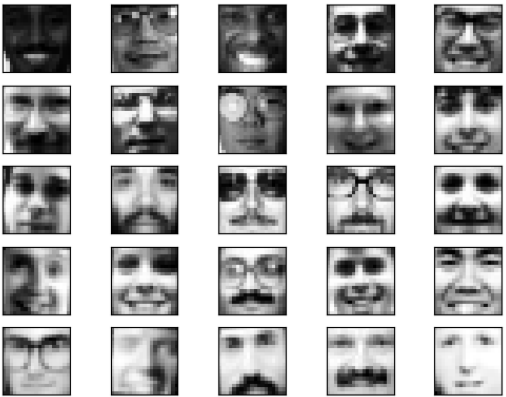


Figure: Farthest first results

Randomized drawing: A typical realisation

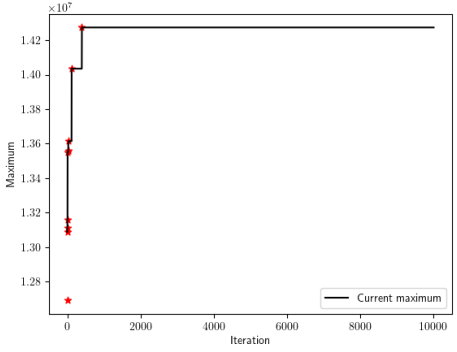


Figure: In practice randomized drawing reaches a stable maximum rather quickly

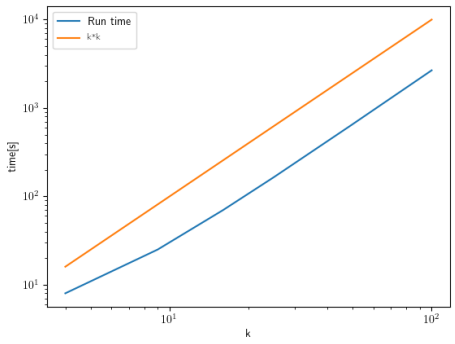


Figure: Randomized drawing: Run time

Exercise 4.3: Convergence and run time

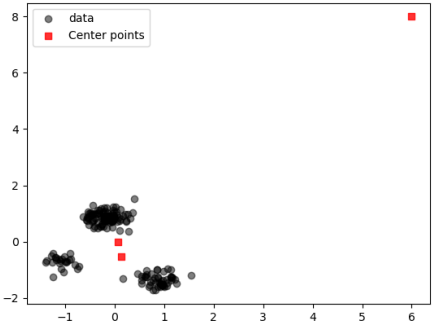


Figure: Result for the blobs

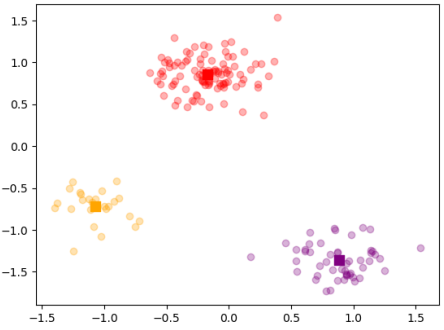


Figure: Result for the faces

k-means clustering: Two perspectives

1. well known formulation: Minimize:

$$\mu_1, \ldots, \mu_k = \underset{\mu_1,\ldots,\mu_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

   where $(C_i)_i$ are the clusters and μ_i are the centroids of each cluster.

2. Alternative formulation:

$$\mu_1, \ldots, \mu_k = \underset{\mu_1,\ldots,\mu_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in C_i} z_{ij} \|x_j - \mu_i\|^2$$

   where $z_{ij} \in \{0, 1\}$ and $\sum_{i=1}^{k} z_{ij} = 1$

3. The second formulation can be solved by a Frank-Wolfe-procedure:

```
def FW_KMEANS_VERSION1(X,k,T_max):

    idx=np.random.randint(0,X.shape[1],k)
    M=X[:,idx]
    for t in range(T_max):
        Z = 1/k *np.ones((k,X.shape[1]))
        Z = FW_UPDATE_Z(X,M,Z, 1)
        M = X @ Z.T @ np.linalg.inv(Z@Z.T)
    return M,Z
```

```
def FW_UPDATE_Z(X,M,Z, t_max):
    ei= np.identity(Z.shape[0])
    for t in range(t_max):
        G_z = 2* (M.T @ M @ Z - M.T @ X)
        #vectorize for loop
        o=np.argmin(G_z, axis=0)
        Z += 2/(t+2) * (ei[o].T- Z)
    return Z
```

Exercise 4.4.1: Results

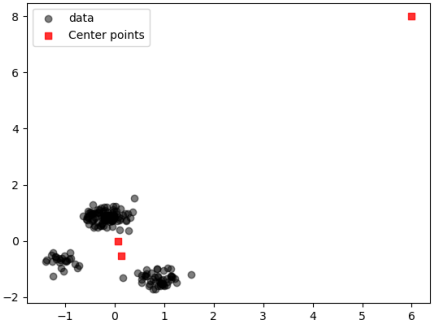


Figure: Result of the naive algorithm

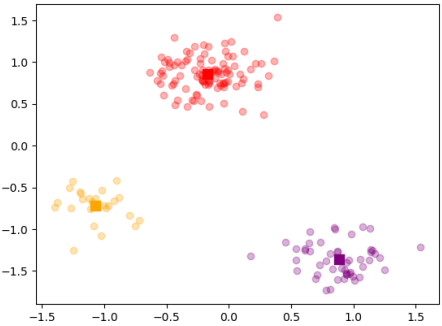


Figure: Result of FW-k-means Version 1

Clearly the results of the naive algorithm are worthless, while also crashing considerably more!

Exercise 4.4.2: Results

1. While using random data points to construct M causes stability issues,because of two matrix inversions, these were manageable
2. Given that our solution for Exercise 4.3 is still rather slow for large k, it was still faster to use random, rather than maximally different, data points.

Figure: Result of FW-k-means Version 1

Exercise 4.5:

1. Code is very similar to exercise 4.4:
   - uses the same vectorization to get rid of the for-loop
   - uses random data points to generate M before the first iteration
2. Main difference: no matrix inversion $\implies$ more robust
3. Similar results: comparable difference when comparing both, or two results of the same version

Exercise 4.5: Results

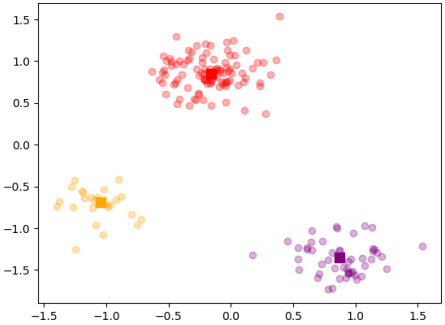


Figure: Result of FW-k-means Version 2 for threeBlobs.csv

Figure: Result of FW-k-means Version 2 for faceMatrix.npy

Exercise 4.6: Comparisson

1. Similar optimization problem compared to exercise 4.5: We only drop the constraint

$$Z \in \{0,1\}^{k \times n}$$

2. Similar code to exercise 4.5
3. Result: We get a set of "exaggerated" representations for each cluster, rather than the mean of each cluster

```
def FW_ARCHETYPAL_ANALYSIS(X,k, T_max=100):

    idx=np.random.randint(0,X.shape[1],k)
    A=X[:,idx]
    for t in range(T_max):
        Z = 1/k *np.ones((k,X.shape[1]))
        Z = FW_UPDATE_Z(X,A,Z, t_max=100)

        Y = 1/X.shape[1] *np.ones((X.shape[1],k))
        Y = FW_update_Y(X, Y,Z, t_max=100)

        A = X@Y
    return A,Y,Z
```

Exercise 4.6: Results

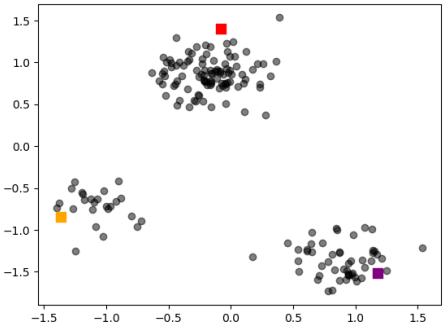


Figure: Archetypes of threeBlobs.csv

Figure: Archetypes of faceMatrix.npy