PRINCIPLES OF MACHINE LEARNING

Exercise sheet 4 - fun with Frank-Wolfe

14.12.2023

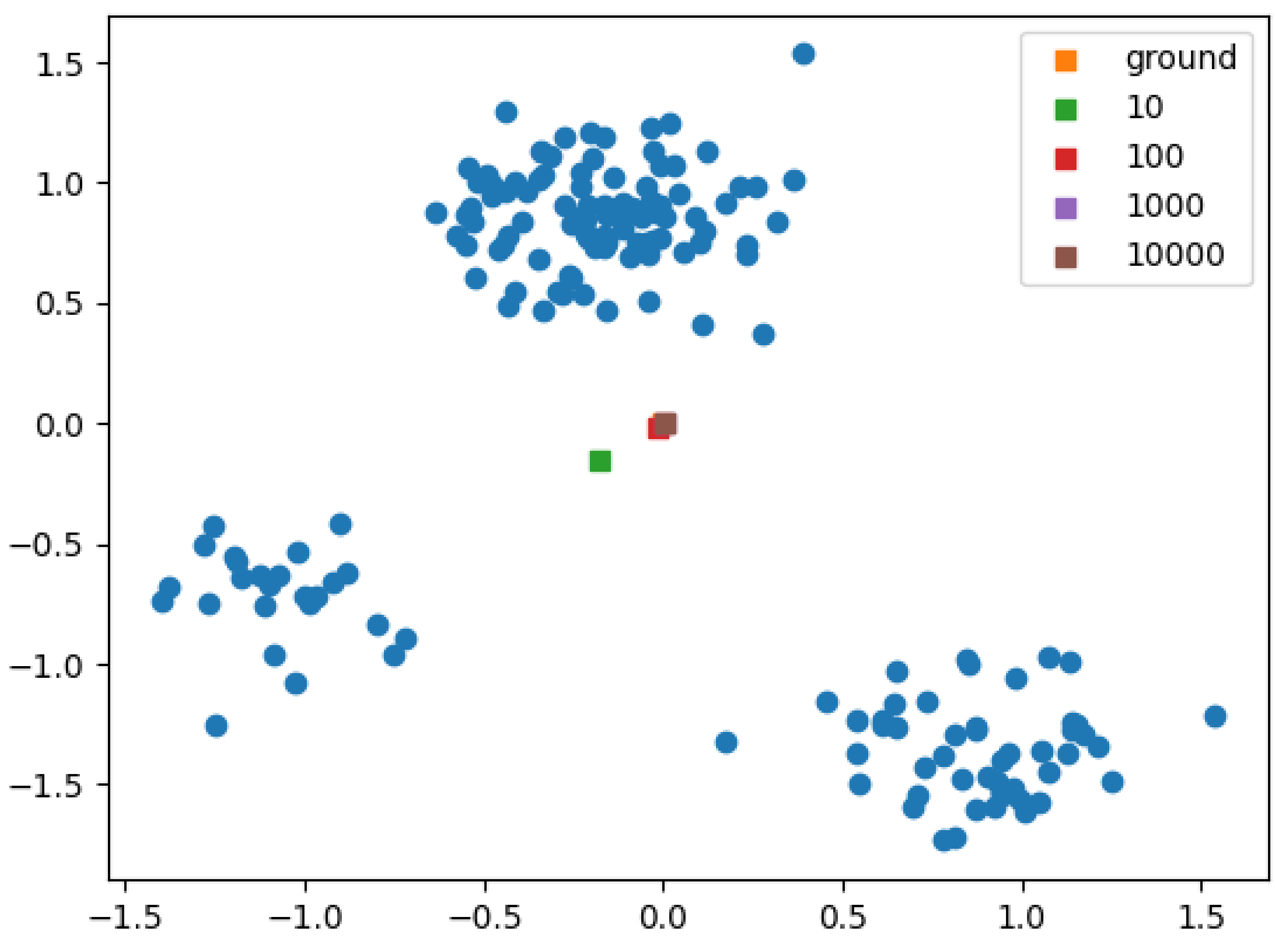
Richard Restetzki	2973740
Akmalkhon Khashimov	50178353
Valdrin Smakaj	50138041
Suraj Giri	50190564
Nijat Sadikhov	50186266
Nikita Morev	50134788
Aleksandr Semenikhin	50118777
Vibhor Sharma	50010826
Muslimbek Abduvaliev	50136555
Suyash Thapa	50205756
Mohammad Mehdi Deylamipour	50009389
Nicolás López Funes	50151598
Andrii Shevliakov	50198078
Nurmukhammad Aberkulov	50102727

4.1: sample means (part 1)

Increase in the number of iterations results in increase of proximity/correctness of the estimated mean.

```
def gradientF(vector: np.ndarray, n: int, data: np.ndarray) -> np.ndarray:
    return 2 * data.transpose() @ data @ (vector - np.ones(n) / n)
def frank_wolfe(T:int, n: int, data: np.ndarray) -> np.ndarray:
    vecW = np.ones(n) / n  # center of the standard simplex
    for t in tqdm(range(T)):
        beta = 2 / (t + 2)
        vecG = gradientF(vecW, n, data) imin = np.argmin(vecG)
        vecW *= (1 - beta)
        vecW[imin] += beta
    return data @ vecW
```

4.1: result



4.2: sample means (part 2)

Given that: $\boldsymbol{X} \in \mathbb{R}^{m\times n}, \boldsymbol{X^T} \in \mathbb{R}^{n\times m}, z \in \mathbb{R}^{n\times 1}, z^T \in \mathbb{R}^{1\times n}, \omega \in \mathbb{R}^{n\times 1}, \omega^T \in \mathbb{R}^{1\times n}$

$\boldsymbol{Inferring}$: $\boldsymbol{X}\omega \in \mathbb{R}^{n\times 1}, z\omega^T \in \mathbb{R}^{n\times n}, z\omega^T\boldsymbol{X^T}\boldsymbol{X}\omega \in \mathbb{R}^{n\times 1}, \boldsymbol{X}\omega z^T \in \mathbb{R}^{n\times n}, \mathrm{tr}[\boldsymbol{X^T}\boldsymbol{X}\omega z^T] \in \mathbb{R}^1, z^T\boldsymbol{X^T}\boldsymbol{X}\omega \in \mathbb{R}^1$

Proof: Let $v_1 = \boldsymbol{X^T}\boldsymbol{X}\omega$ and $v_2 = z^T$ in case 1 and $v_1 = z\omega^T\boldsymbol{X^T}\boldsymbol{X}\omega$ and $v_2 = z^T$ in case 2. Therefore:

The trace of the product $\mathbf{v}_2\mathbf{v}_1$ is given by:

$$\mathrm{Tr}(\mathbf{v}_2\mathbf{v}_1)$$

The product $\mathbf{v}_2\mathbf{v}_1$ results in an $n \times n$ matrix. Let's compute the product:

$$\mathbf{v}_2\mathbf{v}_1 = \begin{bmatrix} v_{21} \\ v_{22} \\ \vdots \\ v_{2n} \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \end{bmatrix}$$

The resulting matrix $\mathbf{v}_2\mathbf{v}_1$ is:

$$\begin{bmatrix} v_{21}v_{11} & v_{21}v_{12} & \cdots & v_{21}v_{1n} \\ v_{22}v_{11} & v_{22}v_{12} & \cdots & v_{22}v_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{2n}v_{11} & v_{2n}v_{12} & \cdots & v_{2n}v_{1n} \end{bmatrix}$$

4.2: sample means (part 2)

Now, the trace is the sum of the diagonal elements:

$$\mathrm{Tr}(\mathbf{v}_2\mathbf{v}_1) = v_{21}v_{11} + v_{22}v_{12} + \cdots + v_{2n}v_{1n}$$

Now, let's consider the product $\mathbf{v}_1\mathbf{v}_2$, which is a 1×1 matrix (a scalar):

$$\mathbf{v}_1\mathbf{v}_2 = \begin{bmatrix} v_{11} \\ v_{12} \\ \vdots \\ v_{1n} \end{bmatrix} \begin{bmatrix} v_{21} & v_{22} & \cdots & v_{2n} \end{bmatrix}$$

The result is:

$$\mathbf{v}_1\mathbf{v}_2 = v_{11}v_{21} + v_{12}v_{22} + \cdots + v_{1n}v_{2n}$$

Now, observe that these two expressions are the same:

$$\mathrm{Tr}(\mathbf{v}_2\mathbf{v}_1) = \mathbf{v}_1\mathbf{v}_2$$

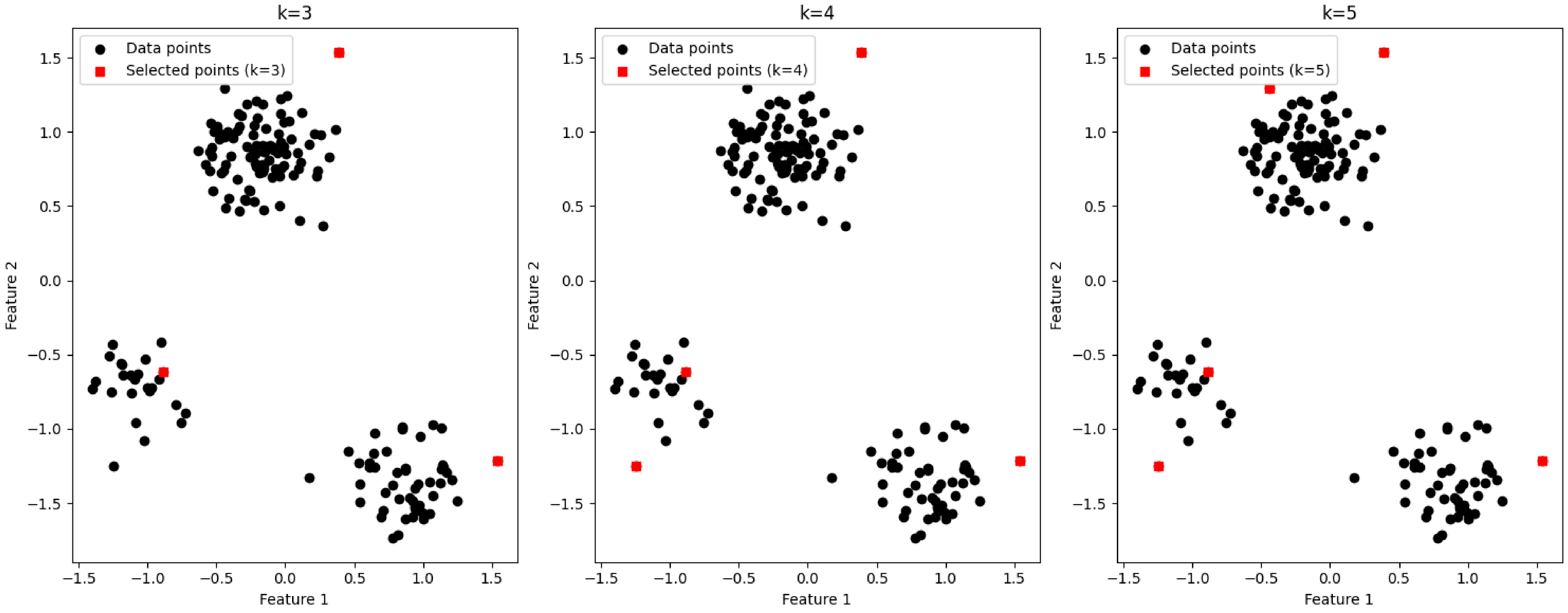
Therefore, we have shown that the trace of the product $\mathbf{v}_2\mathbf{v}_1$ is equal to the trace of the product $\mathbf{v}_1\mathbf{v}_2$.

4.3: finding maximally different data points

```
def select_diverse_points(X, k):
    if k >= X.shape[0]:
        raise ValueError("k must be smaller than the number of data
points")
    selected_indices = []
    for _ in range(k):
        max_dist = 0
        max_idx = -1
        for i in range(X.shape[0]):
            if i not in selected_indices:
                dist_sum = sum(np.linalg.norm(X[i] - X[j]) for j in
selected_indices)
                if dist_sum > max_dist:
                    max_dist = dist_sum
                    max_idx = i
        selected_indices.append(max_idx)

    return selected_indices
```

4.3: finding maximally different data points



4.3: finding maximally different data points (4,9,16,25)

4.3: finding maximally different data points (k=49)

4.3: finding maximally different data points (k=100)

4.4.1: unconventional k-means clustering (part 1)

```
def FW_update_Z(X, M, tmax=1):
    n = X.shape[1]  # Number of data points
    k = M.shape[1]  # Number of clusters
    Z = np.zeros((k, n))  # Initialize Z matrix

    for t in range(tmax):
        GZ = 2 * np.dot(np.dot(M.T, M), Z) - 2 * np.dot(M.T, X)
        row_indices = np.argmin(GZ, axis=0)
        Z = np.zeros_like(Z)  # Reset Z
        Z[row_indices, np.arange(n)] = 1  # Assign each data
point to the closest centroid

    return Z
```

4.4.1: unconventional k-means clustering (part 1)

```
def FW_kMeans_Version1(X, k, Tmax=100):
    m, n = X.shape
    # Randomly initialize centroids by selecting k data points
    indices = np.random.choice(n, k, replace=False)
    M = X[:, indices]

    for T in range(Tmax):
        Z = FW_update_Z(X, M)
        M = np.dot(X, Z.T) @ np.linalg.pinv(Z @ Z.T)  # Update centroids

    return M, Z
```

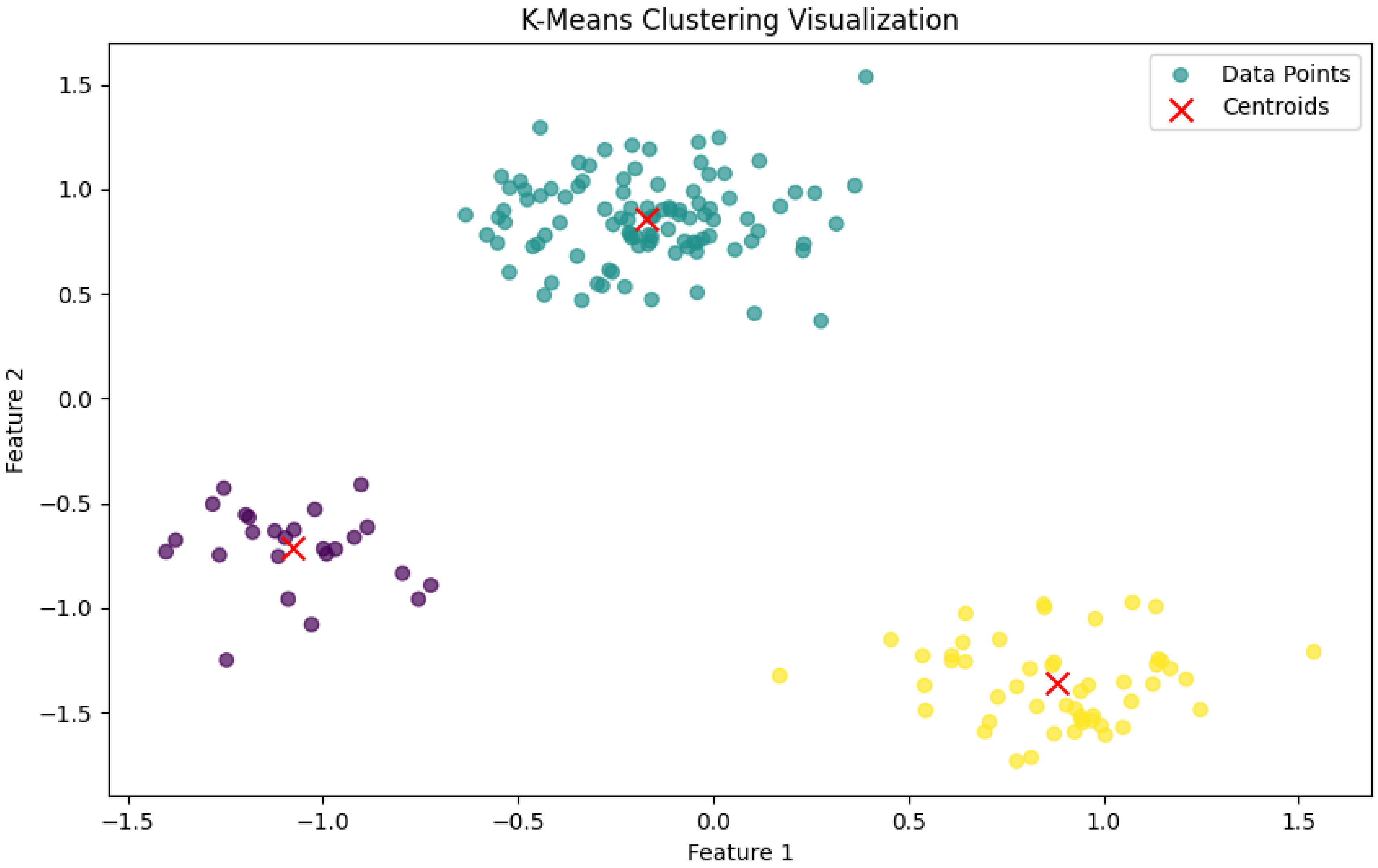
4.4.1: unconventional k-means clustering (part 1)

```
def FW_kMeans_Version1(X, k, Tmax=100):
    m, n = X.shape
    # Randomly initialize centroids by selecting k data points
    indices = np.random.choice(n, k, replace=False)
    M = X[:, indices]

    for T in range(Tmax):
        Z = FW_update_Z(X, M)
        M = np.dot(X, Z.T) @ np.linalg.pinv(Z @ Z.T)  # Update centroids

    return M, Z
```

4.4.1: unconventional k-means clustering (part 1)



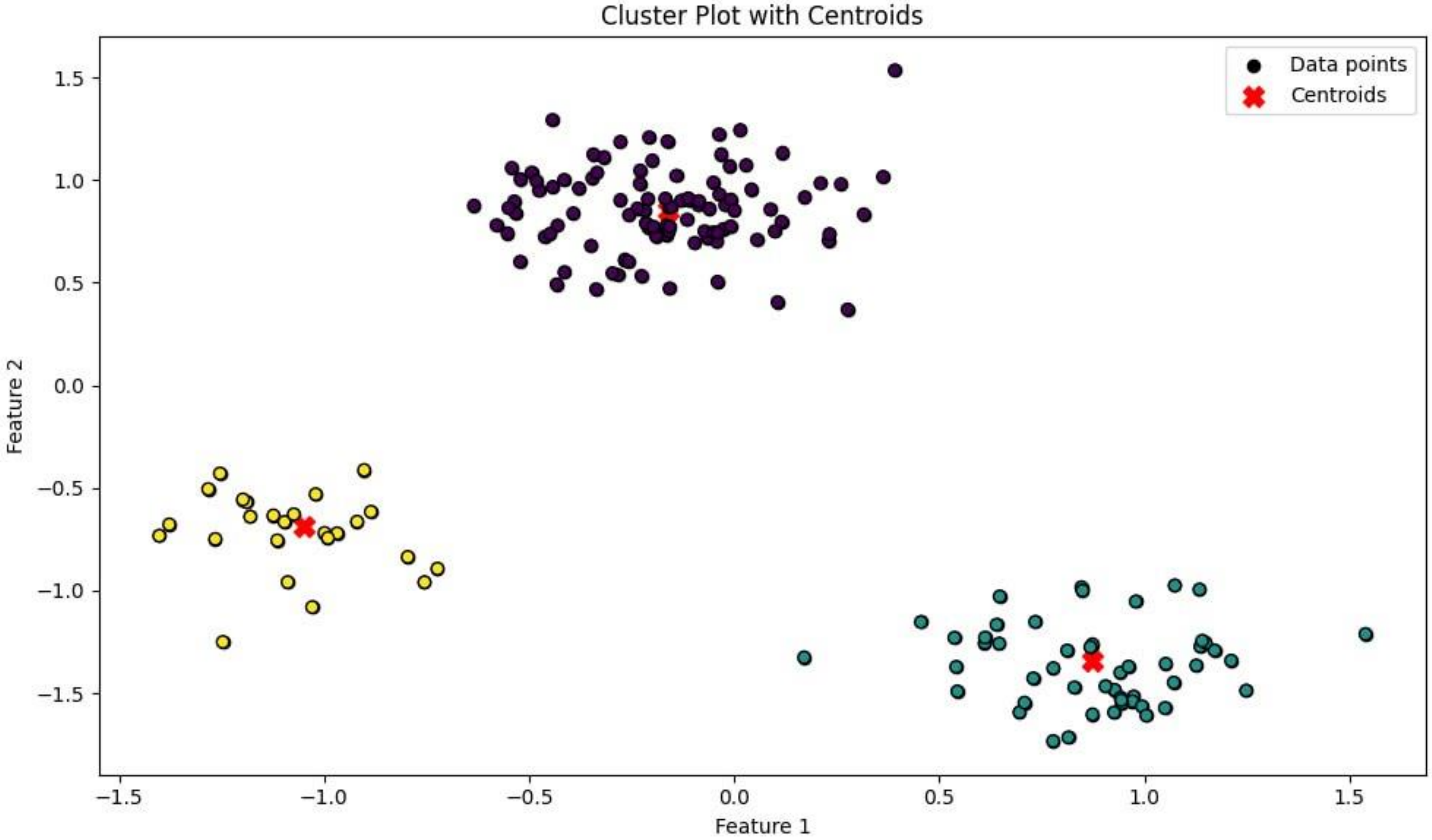
4.4.2: unconventional k-means clustering (part 1)

4.5: unconventional k-means clustering (part 2)

```
def FW_update_Y(X, Y, Z, t_max):
  for t in range(t_max):
    G_y = 2 * (X @ X.T @ Y @ Z @ Z.T - X @ X.T @ Z.T)
    for i in range(Y.shape[1]):
      o = np.argmin(G_y[:, i])
      Y[:, i] = Y[:, i] + (2 / (t + 2)) * (np.eye(Y.shape[0])[o] - Y[:, i])
  return Y

def FW_kMeans_Version2 (X, k, t_max):
  n, m = X.shape
  M = np.random.rand(m, k)
  for _ in range(t_max):
    Z = np.ones((k, n)) / k
    Z = FW_UPDATE_Z(X, M, Z, t_max=1)
    Y = np.ones((n, k)) / n
    Y = FW_UPDATE_Y(X, Y, Z, t_max=100)
    M = np.dot(X.T, Y)
  return M, Y, Z
```

4.5: unconventional k-means clustering (part 2)



4.5: unconventional k-means clustering (part 2)

☹

4.6: archetypal analysis

☹