CSDS 391 HW8

Maximilian Schulten

November 2024

Exercise 1

a) By Bayes Theorem we establish the following:

$$P(\theta|y,n) = \frac{P(y|\theta,n)P(\theta)}{P(y|n)} \tag{1}$$

Now, we assume a uniform prior so $P(\theta) = 1$. So from equation (1) we find that:

$$P(\theta|y,n) = \frac{P(y|\theta,n)}{\int_0^1 P(y|\theta,n)d\theta} = P(y|\theta,n)(n+1) \tag{2}$$

Finally, given the known probability density function of a binomial distribution, inserting into (2) we conclude:

$$P(\theta|y,n) = \binom{n}{y}\theta^y(1-\theta)^{n-y}(n+1) \tag{3}$$

b)

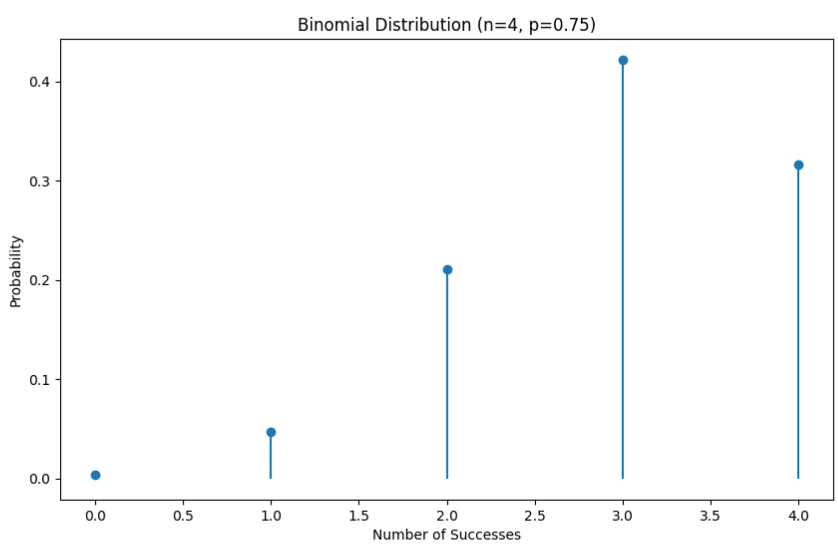


Figure 1: Likelihood of Binomial Distribution

c)

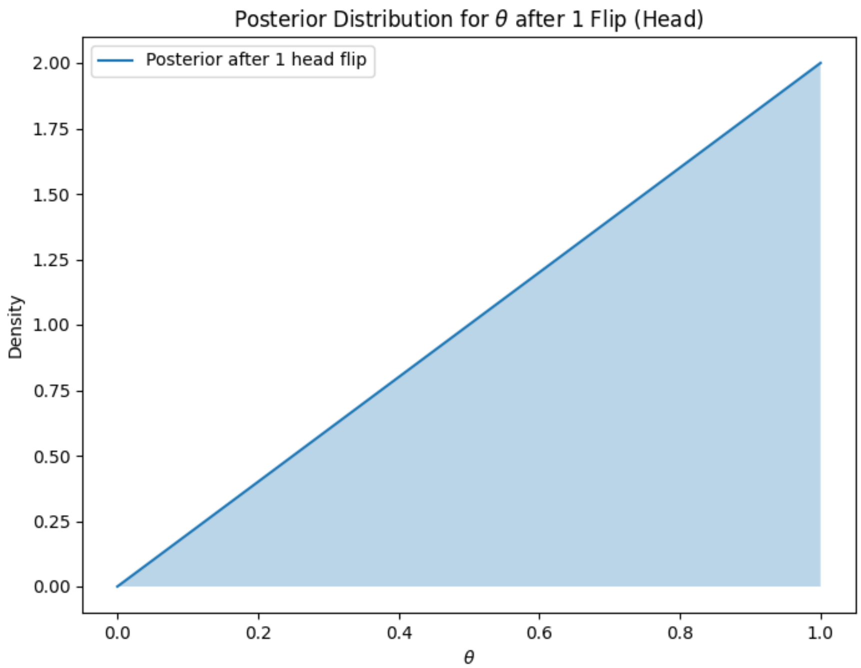


Figure 2: Posterior Distribution After 1 Heads

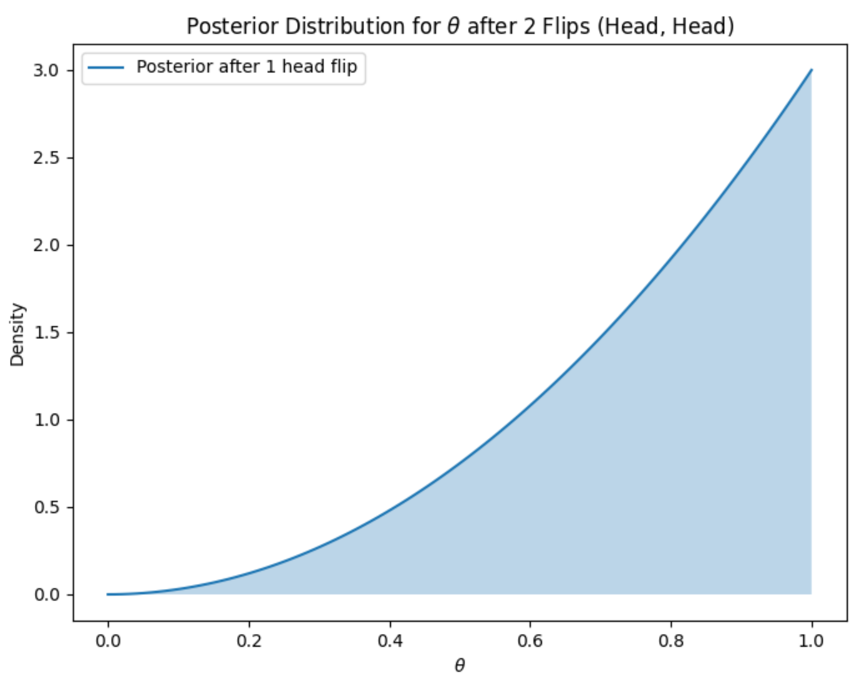


Figure 3: Posterior Distribution After 2 Heads

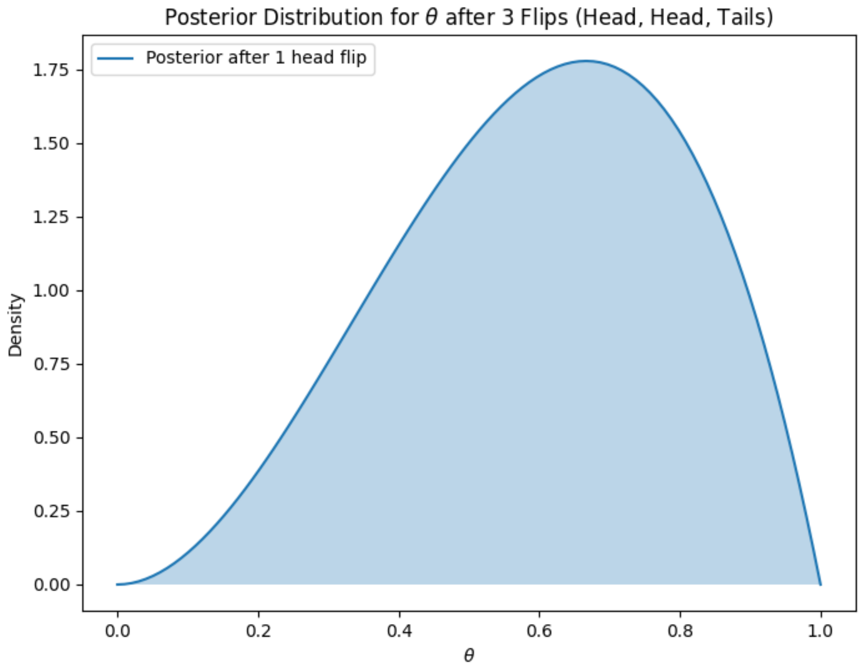


Figure 4: Posterior Distribution After 2 Heads one Tails

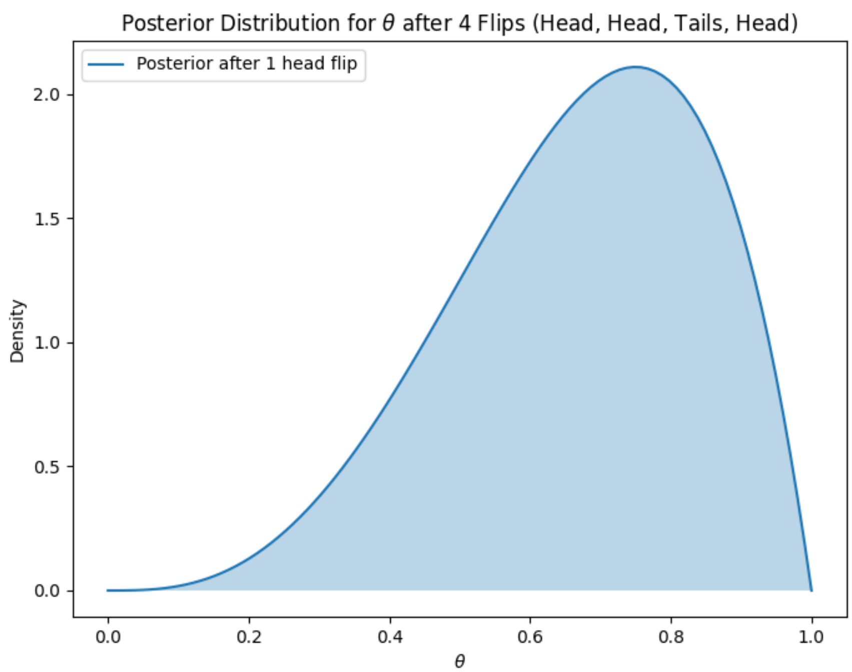


Figure 5: Posterior Distribution After 3 Heads one Tails

Exercise 2

Suppose we have $x_1, \ldots, x_n$ independent, identically distributed observations. We establish the probability density function of the Gaussian distribution as:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma_3^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4}$$

Given our assumptions, we apply the likelihood function for μ and θ:

$$L(\mu, \sigma^2) = \prod_{i=1}^{n} f(x_i|\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right) \tag{5}$$

From here we can optimize the likely hood function with respect to θ and μ to find their respective maximum likelihood estimates. Let us begin by simplifying our expression as much possible.

$$\begin{aligned}
L(\mu, \sigma^2) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right) \\
\ln L(\mu, \sigma^2) &= \ln \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right) = \sum_{i=1}^{n} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)\right) \\
\ln L(\mu, \sigma^2) &= \sum_{i=1}^{n} \left(\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(x_i-\mu)^2}{2\sigma^2}\right) = \sum_{i=1}^{n} \left(-\frac{1}{2}\ln\left(2\pi\sigma^2\right) - \frac{(x_i-\mu)^2}{2\sigma^2}\right) \\
\ln L(\mu, \sigma^2) &= -\frac{n}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2
\end{aligned}$$

Now we optimize with respect to μ.

$$\begin{aligned}
&\frac{\partial}{\partial\mu}\left(\ln L(\mu, \sigma^2) = -\frac{n}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right) \\
&\frac{\partial}{\partial\mu}\ln L(\mu, \sigma^2) = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu) \\
&\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu) = 0 \\
&\sum_{i=1}^{n}(x_i-\mu) = 0 \\
&(x_1-\mu) + (x_2-\mu) + \cdots + (x_n-\mu) = 0 \\
&\sum_{i=1}^{n} x_i = n\mu \\
&\mu = \frac{1}{n}\sum_{i=1}^{n} x_i
\end{aligned}$$

We conclude that $\hat{\mu}_{\text{MLE}} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Now for σ.

$$\begin{aligned}
&\frac{\partial}{\partial\sigma}\left(\ln L(\mu, \sigma^2) = -\frac{n}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right) \\
&\frac{\partial}{\partial\sigma}\ln L(\mu, \sigma^2) = -\frac{n}{2}\frac{4\pi\sigma}{2\pi\sigma^2} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i-\mu)^2 = 0 \\
&-\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i-\mu)^2 = 0 \\
&\frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i-\mu)^2 = \frac{n}{\sigma} \\
&\sum_{i=1}^{n}(x_i-\mu)^2 = n\sigma^2 \\
&\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2
\end{aligned}$$

We conclude that $\hat{\sigma}^2_{\text{MLE}} = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2$.

Exercise 3

a) We establish that $C_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $C_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Moreover, we are given that, from the priors: $P(C_1) = 2P(C_2)$. By the axioms of probability:

$$
\begin{aligned}
P(C_1) + P(C_2) &= 1 \\
2P(C_2) + P(C_2) &= 1 \\
P(C_2) &= \frac{1}{3} \\
P(C_1) &= \frac{2}{3}
\end{aligned}
$$

We can also derive the probability of $P(x)$ as:

$$
\begin{aligned}
P(x) &= P(x|C_1)P(C_1) + P(x|C_2)P(C_2) \\
P(x) &= P(x|C_1)\frac{2}{3} + P(x|C_2)\frac{1}{3} \\
P(x) &= \frac{2}{3}\frac{1}{\sqrt{2\pi\left(\sigma_1\right)^2}}\exp\left(-\frac{(x-\mu_1)^2}{2\left(\sigma_1\right)^2}\right) + \frac{1}{3}\frac{1}{\sqrt{2\pi\left(\sigma_2\right)^2}}\exp\left(-\frac{(x-\mu_2)^2}{2\left(\sigma_2\right)^2}\right)
\end{aligned}
$$

b) Denote the probability of error as $P(\mathbf{E}) \coloneqq P(\text{Classify } x \in C_1 | x \in C_2)P(C_2) + P(\text{Classify } x \in C_2 | x \in C_1)P(C_1)$ in general terms. We are given $\mu_1 < \mu_2$. Interpreting this visually, we expect the probability distribution of $P(x|C_1)$ to lie "below", if you will, that of $P(x|C_2)$ by virtue of them following a Gaussian distribution. As defined in the problem, let $x = \theta$ be the decision boundary, and let us say that $x \in C_1 \iff x < \theta$ and $x \in C_2 \iff x \geq \theta$. So we can derive the probability of misclassifying an $x \in C_1$ as a C_2 element as $P(x \geq \theta | x \in C_1)$ and $P(x < \theta | x \in C_2)$ for the converse. Here we can apply the Cumulative Distribution Function for the Gaussian distribution. Namely:

$$
P(x \geq \theta | C_1) = 1 - \int_{-\infty}^{\theta} P(x|C_1)\,dx = P(\text{Classify } x \in C_2 | x \in C_1) \tag{6}
$$

$$
P(x < \theta | C_2) = \int_{-\infty}^{\theta} P(x|C_2)\,dx = P(\text{Classify } x \in C_1 | x \in C_2) \tag{7}
$$

Finally, we can establish from (6) and (7) that:

$$
P(\mathbf{E}) = \frac{2}{3}\left(1 - \int_{-\infty}^{\theta} P(x|C_1)\,dx\right) + \frac{1}{3}\left(\int_{-\infty}^{\theta} P(x|C_2)\,dx\right)
$$

$$
P(\mathbf{E}) = \frac{2}{3}\left(1 - \int_{-\infty}^{\theta} \frac{1}{\sqrt{2\pi\left(\sigma_1\right)^2}}\exp\left(-\frac{(x-\mu_1)^2}{2\left(\sigma_1\right)^2}\right)dx\right) + \frac{1}{3}\left(\int_{-\infty}^{\theta} \frac{1}{\sqrt{2\pi\left(\sigma_2\right)^2}}\exp\left(-\frac{(x-\mu_2)^2}{2\left(\sigma_2\right)^2}\right)dx\right)
$$

Exercise 4

a) We are given the objective function:

$$
D = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{n,k}\|\mathbf{x_n} - \mu_\mathbf{k}\|^2
$$

Since we are taking the 2-norm of column vectors, we can consider the function represented by D a scalar valued function. Hence we take the partial derivative with respect to $\mu_{k,i}$ (the ith value of μ_k vector) and optimize. Moreover, by linearity of the derivative operation, we may (essentially) bypass the sums.

$$D = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{n,k}\|\mathbf{x_n} - \mu_{\mathbf{i}}\|^2 = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{n,k}\sum_{j=1}^{I}\left(x_{n,j} - \mu_{k,j}\right)^2$$

$$\frac{\partial D}{\partial \mu_{k,i}} = \frac{\partial}{\partial \mu_{k,i}}\left(-2\sum_{n=1}^{N}\sum_{k=1}^{K} r_{n,k}\sum_{j=1}^{I}\left(x_{n,j} - \mu_{k,j}\right)\right)$$

Now, since we are taking the partial derivative with respect to the ith scalar value of the kth mean vector we can ignore all terms where: a) $j \neq i$ and b) terms where we are not in the $k$the cluster. Hence we can drop those 2 sums and replace all instances of j with i as we are considering the case where $j = i$ only.

$$\frac{\partial D}{\partial \mu_{k,i}} = -2\sum_{n=1}^{N} r_{n,k}\left(x_{n,i} - \mu_{k,i}\right) = 0$$

$$\sum_{n=1}^{N} r_{n,k}\left(x_{n,i} - \mu_{k,i}\right) = 0$$

$$\sum_{n=1}^{N} r_{n,k}\left(x_{n,i} - \mu_{k,i}\right) = r_{1,k}x_{1,i} - r_{1,k}\mu_{k,i} + \cdots + r_{N,k}x_{N,i} - r_{N,k}\mu_{k,i} = 0$$

$$r_{1,k}x_{1,i} + \cdots + r_{N,k}x_{N,i} = r_{1,k}\mu_{k,i} + \cdots r_{N,k}\mu_{k,i}$$

$$\sum_{n=1}^{N} r_{n,k}x_{n,i} = \mu_{k,i}\sum_{n=1}^{N} r_{n,k}$$

$$\mu_{i,k} = \frac{\sum_{n=1}^{N} r_{n,k}x_{n,i}}{\sum_{n=1}^{N} r_{n,k}}$$

b) We can generalize this rule for all entries of μ_k as:

$$\mu_{\mathbf{k}} = \frac{\sum_{n=1}^{N} r_{n,k}\mathbf{x_n}}{\sum_{n=1}^{N} r_{n,k}}$$

Exercise 5

a) **Code Snippet**

```
import pandas as pd
import numpy as np


# Take euclidean distance of two points
def distance(p1, p2):

    p1, p2 = np.array(p1), np.array(p2)

    dist = np.sqrt(np.sum(np.square(p1 - p2)))

    return dist


# Assign each point to a cluster
```

```
def assign_clusters(X, centroids):
    clusters = []

    for x in X.values:
        min_dist = float('inf')
        closest_centroid = -1

        for idx, mu in enumerate(centroids):
            dist = distance(x, mu)
            # Update if this centroid is closer
            if dist < min_dist:
                min_dist = dist
                closest_centroid = idx

        # Append the index of the closest centroid for this data point
        clusters.append(closest_centroid)

    return clusters


def update_centroids(X, clusters, k):
    # Initialize a list to hold the new centroids
    new_centroids = []

    # Iterate over each cluster index
    for cluster_idx in range(k):
        # Points in cluster
        pts = []
        for idx, pt in enumerate(clusters):
            if pt == cluster_idx:
                pts.append(X.iloc[idx])
        # If there are points in the cluster, calculate the mean
        if pts:
            # Convert pts to a DataFrame to calculate the mean of each feature
            pts_df = pd.DataFrame(pts)
            new_centroid = pts_df.mean().values
            new_centroids.append(new_centroid)
        else:
            new_centroids.append(np.zeros(X.shape[1]))
    return np.array(new_centroids)


# Main procedure
def main(k, data, max_iters=999, err=1e-4):
    # Read csv
    data = pd.read_csv(data)

    # Set seed for reproducibility
    np.random.seed(123)

    # Find number of features
    # Assume the last column is the target column
    features = data.shape[1]-1

    # Get feature columns
    X = data.iloc[:, :features]
```

```
    # Select K random data points as the initial centroids
    centroids = np.array(X.loc[np.random.choice(X.shape[0], k, replace=False)])

    for _ in range(max_iters):
        clusters = assign_clusters(X, centroids)

        new_centroids = update_centroids(X, clusters, k)

        diff = np.abs(new_centroids - centroids)
        if np.all(diff <= err):
            print('Convergence Reached!')
            return centroids, clusters
        centroids = new_centroids

    print("No convergence...")
    return centroids, clusters
```

Implementation Explanation

Most of this stuff is by the book, exactly as described in the slides. Upon testing, this seed converged after 5 iterations.

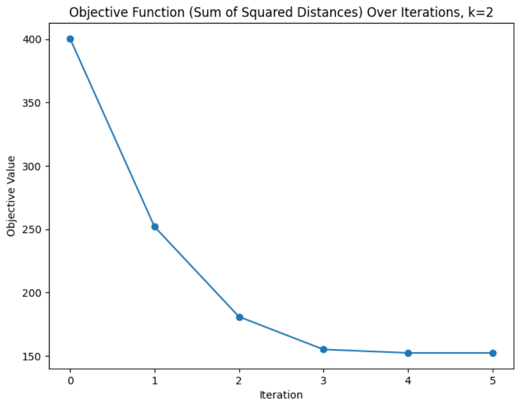
b) **Code Snippet**

```
# Find objective function for plotting
def calculate_objective(X, clusters, centroids):
    total_distance = 0
    for idx, point in enumerate(X.values):
        centroid = centroids[clusters[idx]]
        total_distance += distance(point, centroid) ** 2
    return total_distance

plt.figure(figsize=(8, 6))
plt.plot(objective_values, marker='o')
plt.title(f"Objective Function (Sum of Squared Distances) Over Iterations, k={k}")
plt.xlabel("Iteration")
plt.ylabel("Objective Value")
plt.show()
```

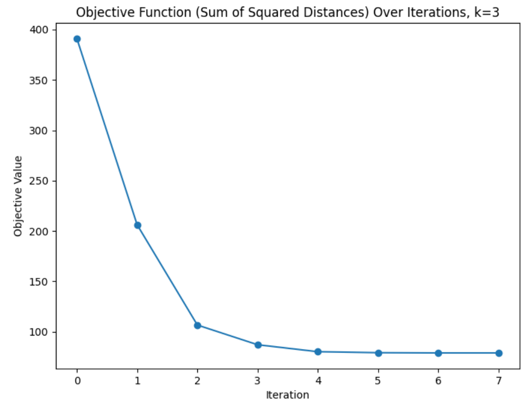
Plots

c) **Code Snippet**

```
# Plotting function for data points and centroids
def plot_clusters(X, centroids, clusters, title):
    plt.figure(figsize=(8, 6))
    for cluster_idx in range(len(centroids)):
        cluster_points = X.values[np.array(clusters) == cluster_idx]
        plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster_idx + 1}')
    centroids = np.array(centroids)
    plt.scatter(centroids[:, 0], centroids[:, 1], s=50, c='black', marker='x', label='Centroids')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.legend()
    plt.show()
```



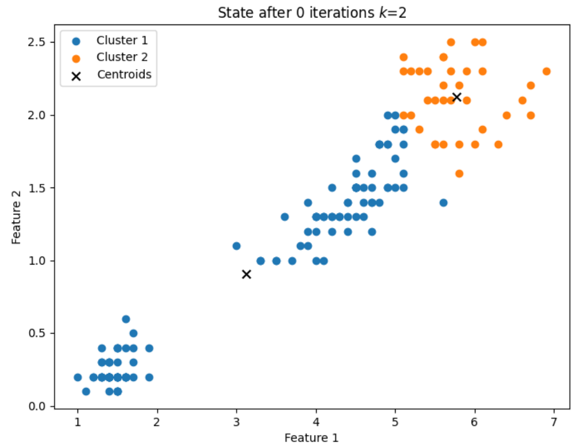
(a) D as a function of the no. of iterations, $k = 2$



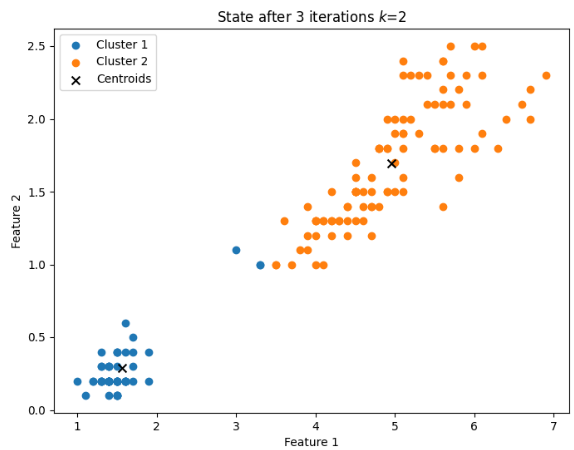
(b) D as a function of the no. of iterations, $k = 3$

Figure 6: Comparison of Objective Functions for Different k Values

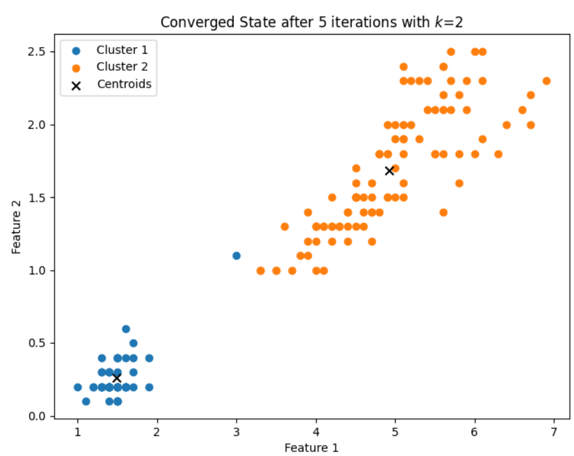
Plots



(a) Clusters and Centroids at initialization

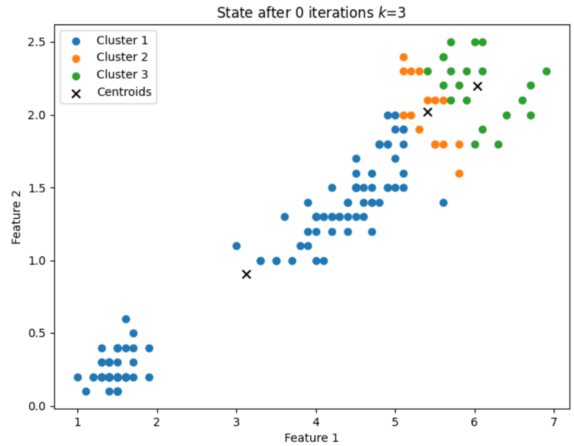


(b) Clusters and Centroids after 3 iterations

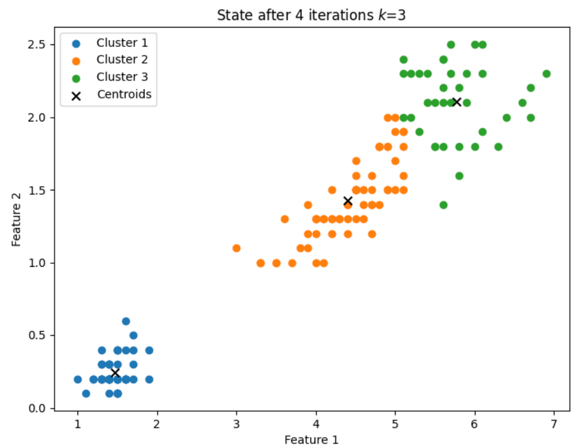


(c) Clusters and Centroids after convergence

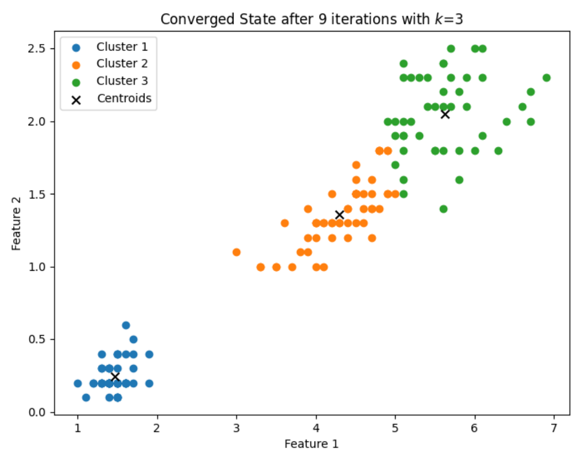
Figure 7: Centroids, Clusters, and Data plotted at different states of k-means clustering where $k = 2$



(a) Clusters and Centroids at initialization



(b) Clusters and Centroids after 3 iterations



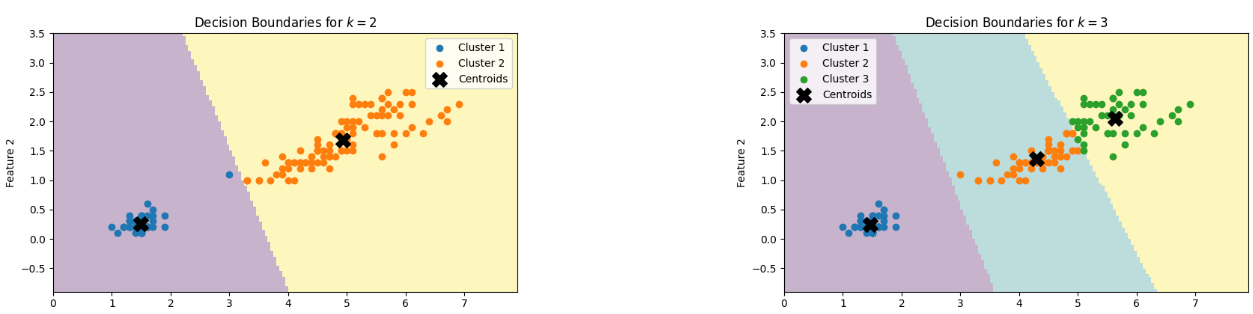
(c) Clusters and Centroids after convergence

Figure 8: Centroids, Clusters, and Data plotted at different states of k-means clustering where $k = 3$

d) The intuition I followed here is that: If we identify the points on the plot where said point is equidistant to one or more centroids then it is part of a decision boundary. This is what I implemented as:

```
def plot_decision_boundaries_approx(X, centroids, clusters, k, title):
    # Define the range of the grid based on the first two features
    x_min, x_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
    y_min, y_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.05),
                         np.arange(y_min, y_max, 0.05))

    # Prepare the grid points to assign clusters
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    grid_df = pd.DataFrame(grid_points, columns=[X.columns[0], X.columns[1]])

    # Assign each grid point to the nearest centroid
    grid_clusters = assign_clusters(grid_df, centroids)
    grid_clusters = np.array(grid_clusters).reshape(xx.shape)

    # Plot decision boundaries by coloring each region
    plt.figure(figsize=(8, 6))
    plt.imshow(grid_clusters, extent=(x_min, x_max, y_min, y_max),
               origin='lower', cmap='viridis', alpha=0.3, interpolation='nearest')

    # Plot the actual data points and centroids
    for cluster_idx in range(k):
        cluster_points = X.values[np.array(clusters) == cluster_idx]
        plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster_idx + 1}')

    centroids = np.array(centroids)
    plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='black', marker='X', label='Centroids')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.legend()
    plt.show()
```

Plots



(a) Decision Boundaries $k = 2$

(b) Decision Boundaries $k = 3$

Figure 9: Decision Boundaries for different k values