Northeastern University, Khoury College of Computer Science

CS 6220 Data Mining | Assignment 4

Sibo Wu

1. derive the maximum likelihood estimate of the parameter Lamda

Answer： the PMF for a Poisson distribution:

$$p(X = x_i|\lambda) = (e^{-}\lambda\lambda_i^x)/(x_i!)$$

The Likelihood function (product)

$$L(\lambda) = \prod_{i=1}^{n} p(X = x_i|\lambda)\backslash$$

Take natural log(base e -> ln) likelihood function

$$\mathrm{l}(\lambda) = \ln L\,(\lambda) = \sum_{i=1}^{n} \ln\left(\frac{e^{-\lambda}\lambda^{x_i}}{x_i!}\right) = \sum_{i=1}^{n}(-\lambda + x_i \ln(\lambda) - \ln(x_i!))$$

Take differentiate the log-likelihood function with lambda

$$\frac{d\mathrm{l}(\lambda)}{d\lambda} = \frac{d}{d\lambda}\left(\sum_{i=1}^{n}(-\lambda + x_i \ln(\lambda) - \ln(x_i!))\right)$$

$$\frac{d\mathrm{l}(\lambda)}{d\lambda} = \sum_{i=1}^{n}\left(-1 + \frac{x_i}{\lambda}\right)$$

Setting this derivative equal to zero

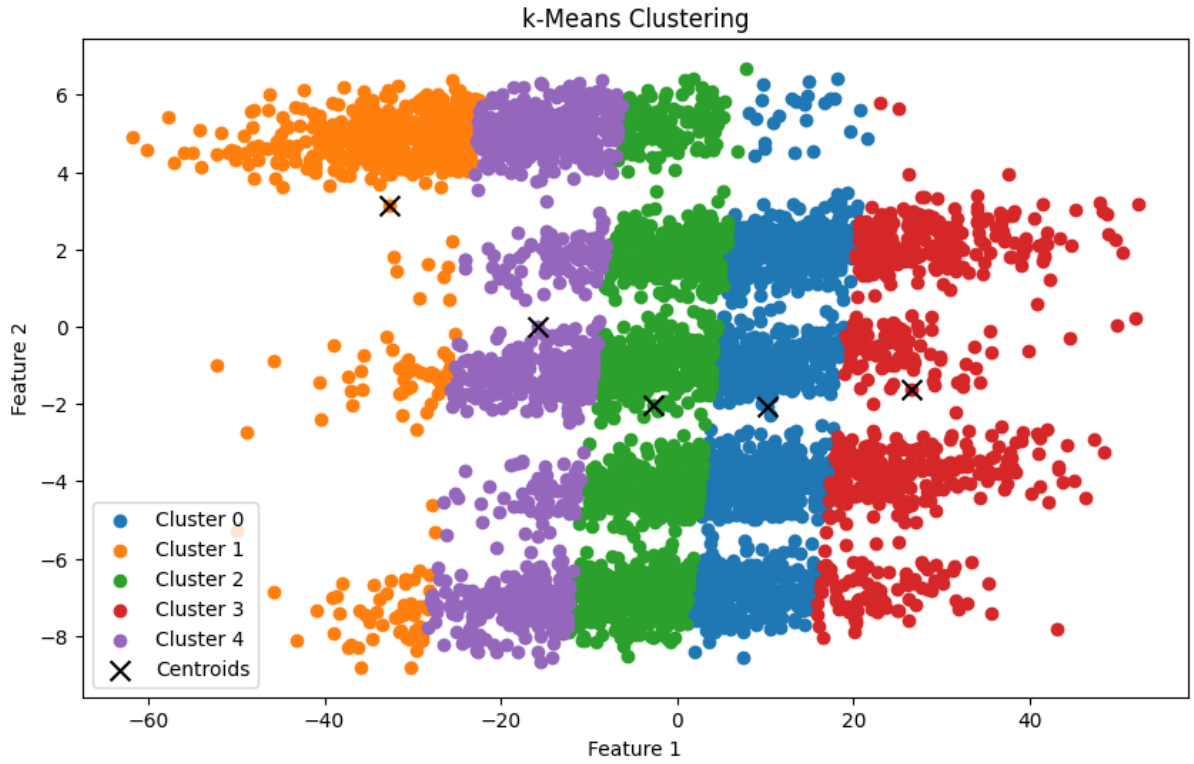
$$-n + \sum_{i=1}^{n} \frac{x_i}{\lambda} = 0$$
$$\lambda = \frac{1}{n}\sum_{i=1}^{n} x_i$$

2. Implement a simple k-means algorithm in Python on Colab with the following initialization:

```
def euclidean_distance(x, y, P=None):
    if P is None:
        return np.sqrt(np.sum((x - y) ** 2, axis=1))
    else:
        diff = x - y
        return np.sqrt(np.diag(diff @ P @ diff.T))
def kmeans_cluster(data,
                   centroids = np.random.randn(2,5),
                   P = np.eye(2),
                   num_iterations = 100):
  centroids = data.T[:, :5].T
  classes = np.random.choice(5, len(data))
  # <YOUR-CODE-HERE>
  for _ in range(num_iterations):
        distances = np.array([euclidean_distance(data, centroid, P) for
centroid in centroids])
        classes = np.argmin(distances.T, axis=1)

        for i in range(len(centroids)):
            centroids[i] = np.mean(data[classes == i], axis=0)

  return centroids, classes
```

3. Scatter the results in two dimensions with different clusters as different colors. You can use matplotlib's pyplot functionality:

```
def plot_data(data, centroids, classes):
  # <YOUR-CODE-HERE>
  plt.figure(figsize=(10, 6))
  for i in range(len(centroids)):
      # Plot data points assigned to each centroid
      plt.scatter(data[classes == i, 0], data[classes == i, 1],
label=f'Cluster {i}')
  plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='x',
s=100, label='Centroids')
  plt.title('k-Means Clustering')
  plt.xlabel('Feature 1')
  plt.ylabel('Feature 2')
  plt.legend()
  plt.show()

  return
```



4. Why is k = 5 a logical choice for this dataset? After plotting your resulting clusters and. What do you notice?

Answer:

Choosing K= 5 likely corresponds to the five distinct production years of the F-150 trucks, suggesting inherent groupings within the data.

The resulting clusters after plotting show clear segmentation, leading that k = 5 effectively captures the variability.

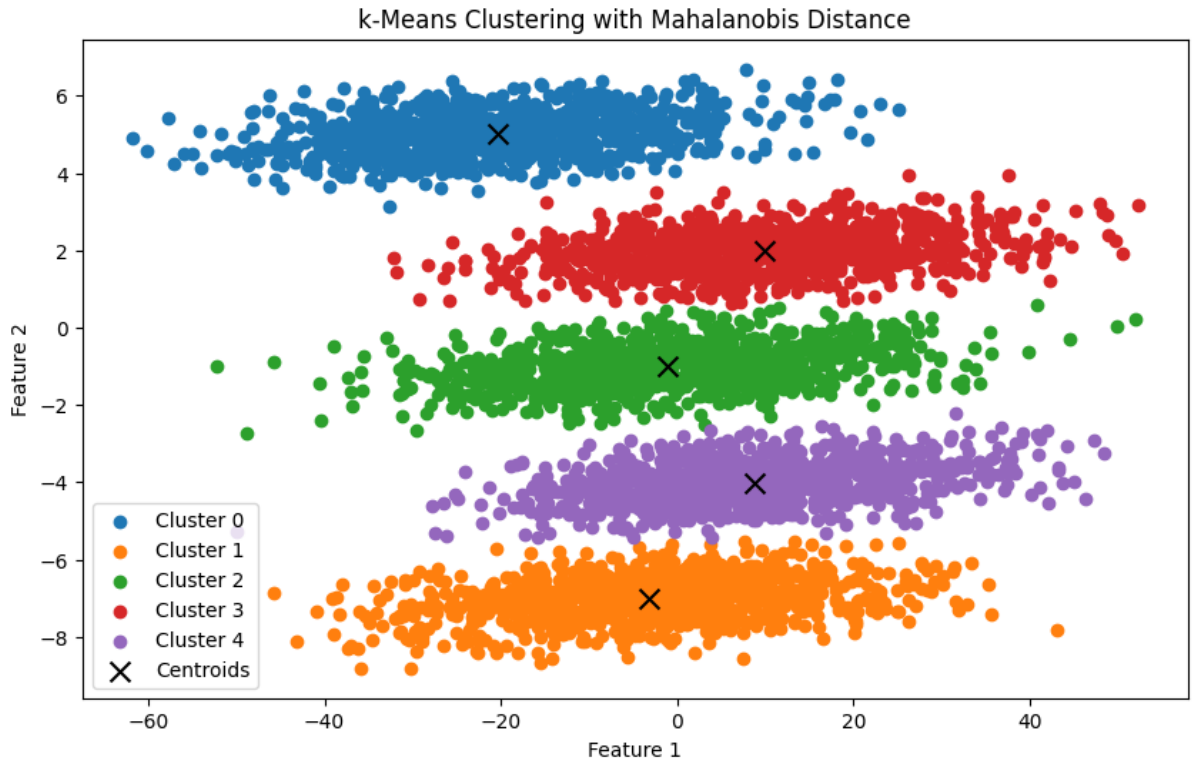
5. Implement a specialized k-means with the above Mahalanobis Distance. Scatter the results with the different clusters as different colors. What do you notice?

Answer: The clusters formed using Mahalanobis Distance reflects a more accurate grouping by considering the relationships between features, as opposed to just their individual values. The data is clearly grouped into separate clusters, suggesting distinct categories within the dataset.

```
def mahalanobis_distance(x, y, P_inv):
    diff = x - y
    return np.sqrt(np.dot(np.dot(diff.T, P_inv), diff))
```

```
def kmeans_cluster(data, centroids, P, num_iterations=100):
    # Precompute the inverse of (P^T P)
    P_inv = np.linalg.inv(np.dot(P.T, P))

    for _ in range(num_iterations):
        # Calculate Mahalanobis distances to centroids
```

```
        distances = np.array([[mahalanobis_distance(x, centroid, P_inv)
for centroid in centroids] for x in data])

        # Assign data points to the nearest centroid
        classes = np.argmin(distances, axis=1)

        # Update centroids
        for i in range(len(centroids)):
            centroids[i] = np.mean(data[classes == i], axis=0) if
len(data[classes == i]) > 0 else centroids[i]

    return centroids, classes

def plot_data(data, centroids, classes):
    plt.figure(figsize=(10, 6))
    for i in range(len(centroids)):
        # Plot data points assigned to each centroid
        plt.scatter(data[classes == i, 0], data[classes == i, 1],
label=f'Cluster {i}')
    plt.scatter(centroids[:, 0], centroids[:, 1], c='black',
marker='x', s=100, label='Centroids')
    plt.title('k-Means Clustering with Mahalanobis Distance')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()
```



6. Calculate and print out the first principle component of the aggregate data.

Answer: First Principal Component - Aggregate Data: [-0.70710678 0.70710678]

7. Calculate and print out the first principle components of each cluster. Are they the same as the aggregate data? Are they the same as each other?

First Principal Component - Cluster 0: [0.70710678 0.70710678]
First Principal Component - Cluster 1: [-0.70710678 -0.70710678]
First Principal Component - Cluster 2: [-0.70710678 -0.70710678]
First Principal Component - Cluster 3: [0.70710678 0.70710678]
First Principal Component - Cluster 4: [-0.70710678 -0.70710678]

The first principal components for each cluster and the aggregate data are similar in magnitude but differ in sign.

This similarity occurs because the direction of maximum variance is consistent across clusters and the aggregate data, indicating that all data subsets—including the entire dataset—have a similar shape in their spread.