Alexander N. V.

HU  Kennis route

OPDRACHT 6: AAI - k-Means

# Code

## Normalisatie van de attributen bij iedere dataset

def normalize\_data(data):

    min\_values = data.min(axis=0)

    max\_values = data.max(axis=0)

    return (data - min\_values) / (max\_values - min\_values)

## Het K-Means Algoritme

def k\_means\_clustering(X, k, max\_iters=100, tol=1e-4):

    """

    Voer k-Means clustering uit.

    Args:

        X (numpy.ndarray): De dataset, een 2D-array met vorm (n\_samples, n\_features).

        k (int): Aantal clusters.

        max\_iters (int): Maximum aantal iteraties.

        tol (float): Tolerantie voor veranderingen in centroiden.

    Returns:

        tuple: (clusters, centroids)

            clusters (list): Lijst van clusterindices voor elke datapunten.

            centroids (numpy.ndarray): De k-centrumwaarden.

    """

    n\_samples, n\_features = X.shape

    centroids = X[np.random.choice(n\_samples, k, replace=False)]

    for iteration in range(max\_iters):

        #toewijzing van punten aan dichtstbijzijnde centroid

        clusters = []

        for point in X:

            distances = np.linalg.norm(point - centroids, axis=1)

            cluster\_index = np.argmin(distances)

            clusters.append(cluster\_index)

        clusters = np.array(clusters)

        # update centroiden als het gemiddelde van de punten in elke cluster

        new\_centroids = np.array([X[clusters == i].mean(axis=0) for i in range(k)])

        if np.all(np.linalg.norm(new\_centroids - centroids, axis=1) < tol):

            break  # stop als de verandering in centroiden onder de tolerantie is

        centroids = new\_centroids

    return clusters, centroids

## Call Tree

## Root

## --- get\_season\_label

## --- normalize\_data

## --- k\_means\_clustering

## --- np.random.choice

## --- np.linalg.norm

## --- np.argmi

## --- determine\_optimal\_k

## --- k\_means\_clustering

## --- np.linalg.norm

## --- assign\_season\_to\_clusters

## --- Counter

## Uitleg

De code begint met het inladen van de dataset via **np.genfromtxt** voor de gegevens en datums. De functie **get\_season\_labe**l wordt gebruikt om het seizoen (bijv. 'winter', 'zomer') te bepalen op basis van de datums.

De data wordt vervolgens genormaliseerd met de functie **normalize\_data**. In de **k\_means**\_**clustering** functie worden de gegevens verdeeld in **k** clusters. Het algoritme kiest willekeurig centroiden, wijst elk datapunt toe aan de dichtstbijzijnde centroid

Vervolgens worden de clusters gelabeld met de functie **assign\_season\_to\_clusters** op basis van het meest voorkomende seizoenlabel binnen elk cluster.

Ten slotte berekent de functie **determine\_optimal\_k** de "distortion" voor verschillende waarden van k en toont de resultaten vervolgens in een scree plot om het optimale aantal clusters te bepalen.

# Resultaten

## Beste K-waarde berekenen

Hiermee bepalen we een optimale k.

def determine\_optimal\_k(X):

    """

    Bepaal het optimale aantal clusters (k) met een scree plot.

    Args:

        X (numpy.ndarray): De dataset, een 2D-array met vorm (n\_samples, n\_features).

    Returns:

        None (maakt een scree-plot).

    """

    distortions = []

    K = range(1, 11)  # Test k van 1 tot 10

    for k in K:

        \_, centroids = k\_means\_clustering(X, k)

        distortions.append(np.sum(np.min(np.linalg.norm(X[:, None] - centroids, axis=2), axis=1)))

    plt.figure(figsize=(8, 5))

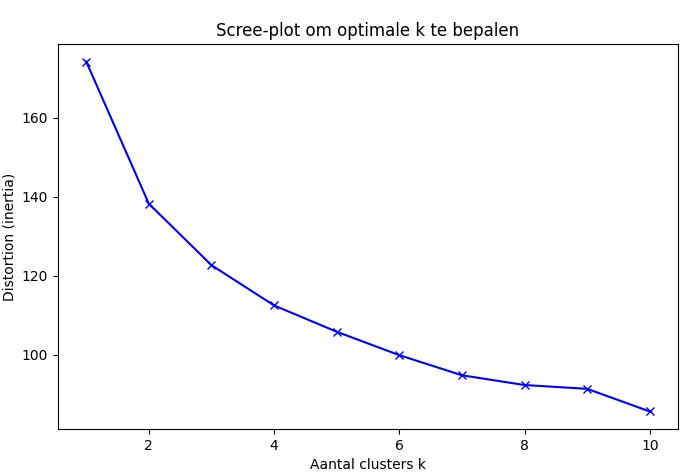
    plt.plot(K, distortions, 'bx-')

    plt.xlabel('Aantal clusters k')

    plt.ylabel('Distortion (inertia)')

    plt.title('Scree-plot om optimale k te bepalen')

    plt.show()



## Toepassing van de Maximum vote

def assign\_season\_to\_clusters(clusters, labels):

    """Deze functie bepaalt welk label het beste de betekenis van elk cluster.

    Args:

        clusters (np.numpy): Array die clusterindeling voor elk datapunt bevat

        labels (list): seizoenlabels in dit geval

    Returns:

        Dictionary: Dictionary waarbij de sleutel het cluster is en de waarde het meest voorkomende label

    """

    cluster\_labels = {}

    for cluster in range(k):

        cluster\_points = labels[clusters == cluster]

        most\_common = Counter(cluster\_points).most\_common(1)[0][0]

        cluster\_labels[cluster] = most\_common

    return cluster\_labels

## Gebruik van de bovenstaande functie:

clusters, centroids = k\_means\_clustering(train\_data\_normalized, k)

cluster\_labels = assign\_season\_to\_clusters(np.array(clusters), y\_train)

# Resultaten

for cluster, season in cluster\_labels.items():

    print(f"Cluster {cluster} is toegewezen aan seizoen: {season}")

# Optimal k bepalen met scree plot

determine\_optimal\_k(train\_data\_normalized)

OUTPUT:  
Cluster 0 is toegewezen aan seizoen: zomer

Cluster 1 is toegewezen aan seizoen: winter

Cluster 2 is toegewezen aan seizoen: herfst

Cluster 3 is toegewezen aan seizoen: lente