OPDRACHT 6: AAI - K-MEANS

Alexander N. V.

HU Kennis route

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

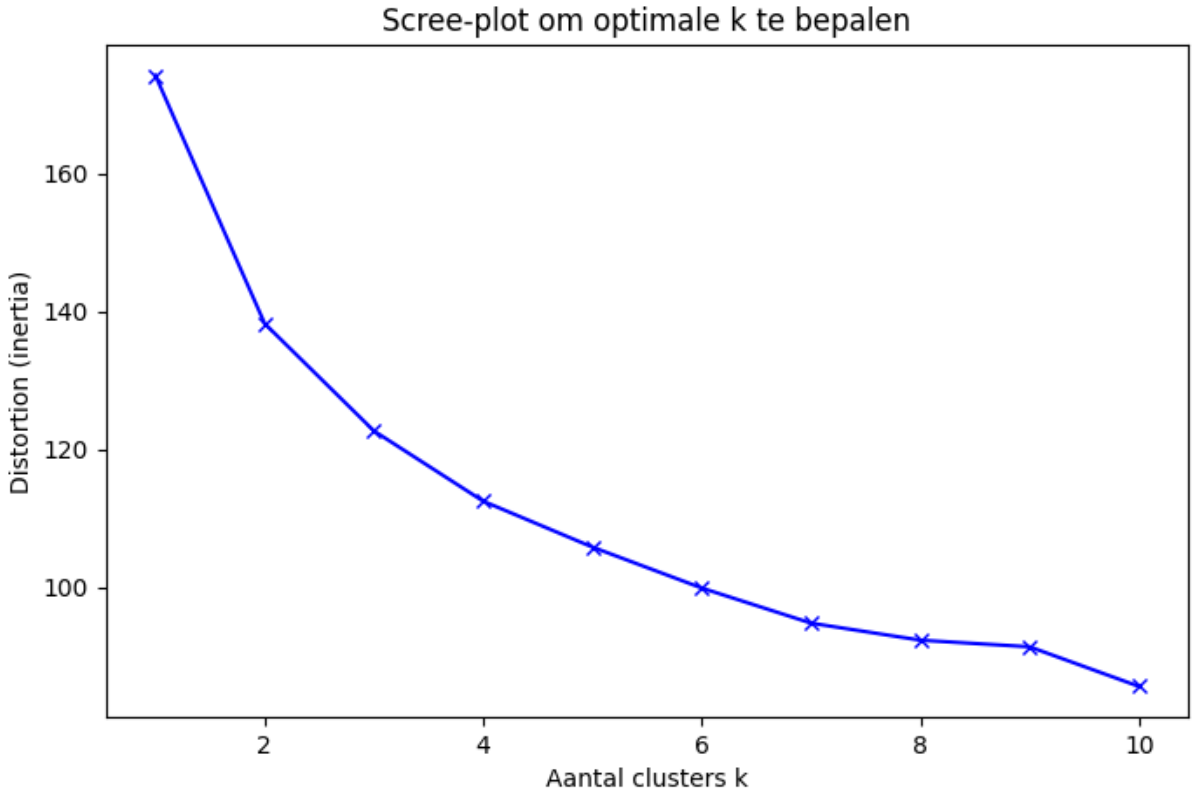
Hiermee bepalen we een optimale k.

```
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