OPDRACHT 5: AAI - K-NEAREST NEIGHBORS

Alexander N. V.

HU Kennis route

Code

Normalisatie van de attributen bij iedere dataset

```
def get_min_max_dataset(dataset):
    """
    Berekent de minimale en maximale waarde voor elke kolom (attribuut) in een
dataset.

    Args:
        dataset (numpy.ndarray): Een 2D NumPy array met numerieke waarden.

    Returns:
        tuple: Een tuple van twee lijsten. De eerste lijst bevat de minimale
waarden
               per kolom, de tweede lijst de maximale waarden.
    """
    max_atr = []
    min_atr = []
    for i in range(0, len(dataset[0])):
        max_atr.append(max(dataset[:, i]))
        min_atr.append(min(dataset[:, i]))
    return min_atr, max_atr
```

```
def normalize_datasets(data_1, data_2):
    """
    Normaliseert 2 datasets door gebruik te maken van de min-max normalisatie

    Args:
        data_1 (numpy.ndarray): Eerste dataset.
        data_2 (numpy.ndarray): Tweede dataset.

    Returns:
        tuple: A tuple of the normalized datasets.
    """
    # min en max per kolom
    min_max_data1_ = get_min_max_dataset(data_1)
    min_max_data2_ = get_min_max_dataset(data_2)

    # globale min en max van iedere feature wordt in een np array opgeslagen
    min_ = np.minimum(min_max_data1_[0], min_max_data2_[0])
    max_ = np.maximum(min_max_data1_[1], min_max_data2_[1])
    # Hieronder nemen we voordeel van de volgende numpy array eigenschap
    range_ = max_ - min_

    # dit stukje code is delen door nul te voorkomen
    range_[range_ == 0] = 1
```

```
    # Normalize datasets
    data_1_normalized = (data_1 - min_) / range_
    data_2_normalized = (data_2 - min_) / range_

    return data_1_normalized, data_2_normalized
```

Het K-Nearest Neighbour Algoritme

```
def k_NN(X_train, y_train, x_test, k=3):
    """
     Voer k-Nearest Neighbors uit en verminder k bij ties zonder recursie.

    Args:
        X_train(np.array): dataset zonder labels
        y_train(np.array): labels van X_train
        x_test(np.array): Test vector

    Returns:
        tuple: A tuple of the normalized datasets.
    """
    if k < 1:
        raise ValueError("k moet minimaal 1 zijn.")

    # Bereken de afstanden tussen de testvector en elke trainingsvector
    distances = [(np.linalg.norm(x_test - x_train), label)
                 for x_train, label in zip(X_train, y_train)]

    # Sorteer de afstanden
    sorted_distances = sorted(distances, key=lambda x: x[0])

    """
    We hoeven niet opnieuw naar de vectoren in X_train te kijken,
    omdat de afstand van een vector slechts wordt gebruikt om de
dichtstbijzijnde buren te sorteren.
    Zodra   we de dichtstbijzijnde k-buren hebben, zijn alleen hun labels van
belang
    voor de classificatie, ongeacht hoe dicht ze bij het testpunt liggen.
    """
    # Itereer en probeer k-1 als er een tie is
    while k >= 1:
        k_closest_labels = [label for _, label in sorted_distances[:k]]
        label_counts = Counter(k_closest_labels).most_common()

        if len(label_counts) > 1 and label_counts[0][1] == label_counts[1][1]:
            k -= 1  #verminder k en probeer opnieuw
        else:
            #geen tie -> retourneer het label met de hoogste frequentie
```

```
            return label_counts[0][0]

    # Als k is verminderd tot 1, neem het label van de dichtstbijzijnde buur
    k_closest_labels = [label for _, label in sorted_distances[:k]]
    label_counts = Counter(k_closest_labels).most_common()
    return label_counts[0][0]
```

Gebruik van de normalisatie en KNN

```
X_validation =np.genfromtxt('validation1.csv', delimiter=';',
usecols=[1,2,3,4,5,6,7], converters={5: lambda s: 0 if s == b"-1" else
float(s), 7: lambda s: 0 if s == b"-1" else float(s)})

dates_validation = np.genfromtxt('validation1.csv', delimiter=';', usecols=0,
dtype=int)

# labels voor validation data
y_validation = np.array([get_season_label_v(date) for date in
dates_validation])

#Normalisatie van de waardes in iedere dataset
train_data_normalize, X_validation_normalized = normalize_datasets(train_data,
X_validation)
predictions_validation = [k_NN(train_data_normalize, y_train, x_val, k=2) for
x_val in X_validation_normalized]

# Compute validation accuracy
accuracy_validation = np.mean([pred == true_label for pred, true_label in
zip(predictions_validation, y_validation)])

print(f"Validation Accuracy: {(accuracy_validation * 100):.2f} %")
```

Call Tree

Root

--- get_season_label_v

--- normalize_datasets

--------- get_min_max_dataset

--- k_NN

        --- Counter

--- np.genfromtxt

Uitleg

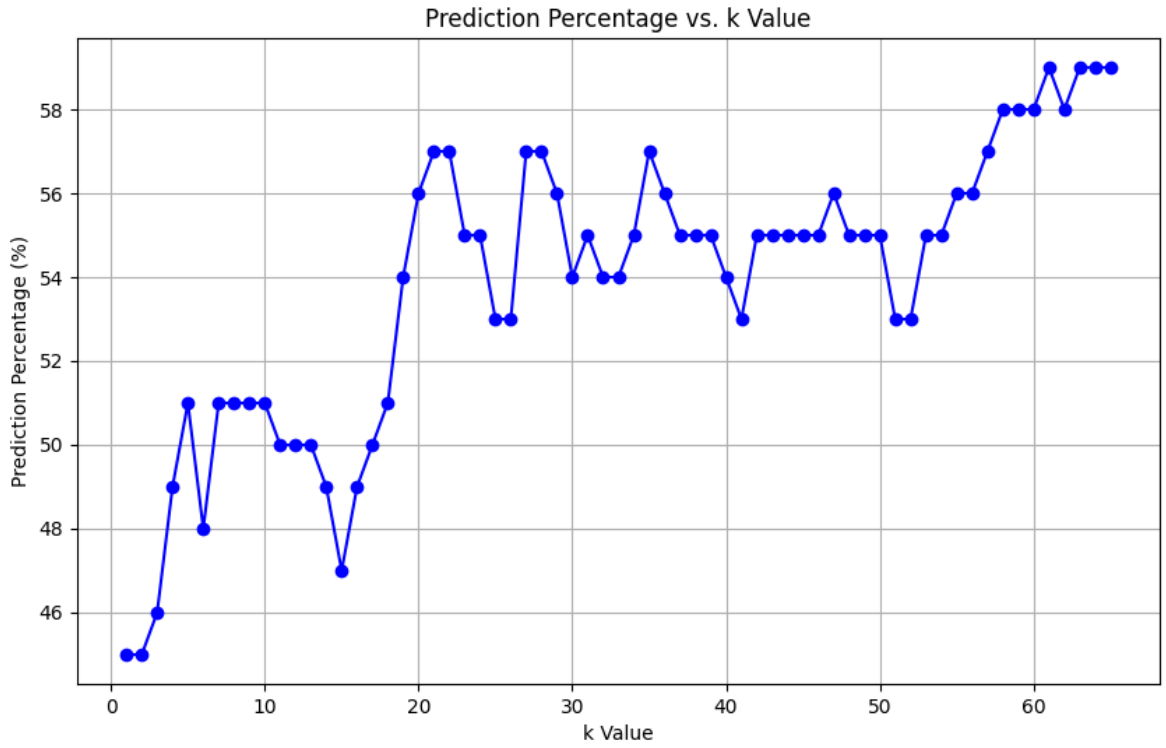
Voor een goede uitvoering van het algoritme moet ten eerste de dataset goed van tevoren genormaliseerd worden. Hiervoor wordt eerst de data (data + labels) natuurlijk goed ingeladen door **np.genfromtxt** en **get_season_label_y.** Vervolgens roept **normalize_datasets get_min_max_dataset** aan om de max en min uit iedere attribuut te berekenen. **normalize_datasets** functie gaat verder met de berekening van de globale min en max voor alle features in beide datasets.

Nu dat de datasets genormaliseerd zijn, voert de k-Nearest Neighbors de classificatie uit. De Euclidische afstand tussen de testvector en alle trainigsfactoren worden berekend door **Counter.** Tenslotte retourneer het algoritme het meest voorkomende label. Tie handeling wordt hier ook uitgevoerd. Omdat deze dataset meerdere klassen, als er een tie is, wordt de k-waarde verlaagd en opnieuw geprobeerd.

Resultaten

Beste K-waarde berekenen

Hieronder zien we een grafiek van de opgenomen resultaten tijdens het instellen van de k-waarde m.b.v. de validatie dataset. dsa



Algoritme testen met beste k-Waarde

De code voor de opzet van de test is het volgende:

```
from Op_5_AAI_k_nearest_neighbour import *
test_dataset = np.genfromtxt('days.csv', delimiter=';',
usecols=[1,2,3,4,5,6,7], converters={5: lambda s: 0 if s == b"-1" else
float(s), 7: lambda s: 0 if s == b"-1" else float(s)})
dates = np.genfromtxt( 'dataset1.csv', delimiter=';', usecols=[0])

output_file = "knn_results.txt"

test_dataset_normalized, train_data_norm = normalize_datasets(test_dataset,
train_data)

results = []
for x_test in test_dataset_normalized:
    prediction = k_NN(train_data_norm, y_train, x_test, k=61)
    results.append(prediction)

with open(output_file, "w") as file:
    for i, result in enumerate(results):
        file.write(f"Testvector {i+1}: Voorspeld label = {result}\n")
```

M.b.h. van de eerder berekende k-waarde, wordt het algoritme met bepaalde data uit willekeurige dagen getest. In dit geval heb ik 61 als k-waarde. Een overzicht van de tabel die hier werd gebruikt is het volgende:

G	TG	TN	TX	SQ	DR	RH				
40;52;2;102;103;0;0	4	5.2	0.2	10.2	10.3	0	0			
25;48;-18;105;72;6;1	2.5	4.8	-1.8	10.5	7.2	0.6	1			
23;121;56;150;25;18;18	2.3	12.1	5.6	15.0	2.5	1.8	1.8			
27;229;146;308;130;0;0	2.7	22.9	14.6	30.8	13.0	0	0			
41;65;27;123;95;0;0	4.1	6.5	2.7	12.3	9.5	0	0			
46;162;100;225;127;0;0	4.6	16.2	10.0	22.5	12.7	0	0			
23;-27;-41;-16;0;0;-1	2.3	-2.7	-4.1	-1.6	0	0	0.05			
28;-78;-106;-39;67;0;0	2.8	-7.8	-10.6	-3.9	6.7	0	0			
38;166;131;219;58;16;41	3.8	16.6	13.1	21.9	5.8	1.6	4.1			

Op basis van de data van iedere dag heeft het algoritme de volgende voorspellingen gemaakt:

- Testvector 1: Voorspeld label = lente

- Testvector 2: Voorspeld label = winter
- Testvector 3: Voorspeld label = lente
- Testvector 4: Voorspeld label = zomer
- Testvector 5: Voorspeld label = lente
- Testvector 6: Voorspeld label = zomer
- Testvector 7: Voorspeld label = winter
- Testvector 8: Voorspeld label = winter
- Testvector 9: Voorspeld label = zomer