Documentatie KNN & KMeans

21-11-2023

Jippe Heijnen
AAI - Innovation
Leergang 23' - '24

Inhoud

Overzicht

Hieronder heb ik de beschrijvingen van de algoritmes met daarbij call trees van beide implementaties, uitleg van de functies en een korte reflectie op de resultaten.

K Nearest Neighbours

Ik heb verschillende onderdelen gemaakt om zo ook het effect van het normaliseren te ontdekken. Voor beide scenario's heb ik een plot gemaakt waarin de resultaten zijn te zien.

Call tree

```
├── PART ONE
│   ├── KNN
│   │   ├── calculate_distance
│   │   └── get_season
│   │       └── translate
│   ├── translate
│   └── Plot accuracy_points
├── PART TWO
│   ├── normalize_weather_data
│   ├── add_dates
│   ├── KNN
│   │   ├── calculate_distance
│   │   └── get_season
│   │       └── translate
│   ├── translate
│   └── Plot accuracy_points
├── PART THREE
│   └── KNN
│       ├── calculate_distance
│       └── get_season
│           └── translate
└── Print assigned seasons for both scenarios
```

Beschrijving implementaties

normalize_weather_data()

Parameters:

dset: np.ndarray[int, Any], other_sets: np.ndarray[[int, Any]]

Return type:

Union[np.ndarray[float, Any], Tuple[np.ndarray[float, Any], Any]]

Beschrijving:

Normaliseert de weergegevens door de minimum- en maximumwaarden van alle meegekregen datasets te gebruiken. Past $(X - X_{min}) / (X_{max} - X_{min})$ toe voor elke waarde in de gegeven datasets.

Werking:

Alle datasets worden samen genormaliseerd, zodat de waarden van de sets onderling niet scheef lopen tov elkaar.

add_dates()

Parameters:

dateless: list, dates

Return type:

Tuple

Beschrijving:

Bij het normaliseren van de datasets, moet de datum niet meegerekend worden, hierom wordt de datum weggehaald tijdens het normaliseren. Deze functie voegt de oorspronkelijke datums weer toe aan genormaliseerde datasets.

Werking:

Maakt een pandas DataFrame van genormaliseerde datasets en voegt de datums toe in kolom 0 van de datasets. Hierom is het belangrijk dat de volgorde van de parameters klopt.

calculate_distance()

Parameters:

a, b

Return type:

float

Beschrijving:

Berekent de Euclidische afstand tussen twee punten. In praktijk zijn dit de afstanden tussen een datapunt en een ander datapunt, of de afstand tussen een datapunt en een centroid.

Werking:

Gebruikt de formule voor de Euclidische afstand tussen twee punten in meerdere dimensies.

translate()

Parameters:

date

Return type:

String

Beschrijving:

Vertaalt een datum naar een seizoen.

Werking:

Converteert de gegeven datum naar een datetime-object en bepaalt het seizoen op basis van de maand.

get_season()

Parameters:

dataset, hvar_k

Return type:

String

Beschrijving:

Geeft het meest voorkomende seizoen terug voor een datapunt op basis van datapunten die dichtbij gelegen zijn.

Werking:

Telt het aantal voorkomende seizoenen in naburige datapunten en geeft het meest voorkomende seizoen terug.

KNN()

Parameters:

training_dataset: np.array, test_dataset: np.array, hvar_k

Return type:

List[str]

Beschrijving:

Voert het k-Nearest Neighbours-algoritme uit om seizoenen te voorspellen op basis van weerdata. Uitgebreide stappen van hoe dit algoritme werkt staat in de code.

Werking:

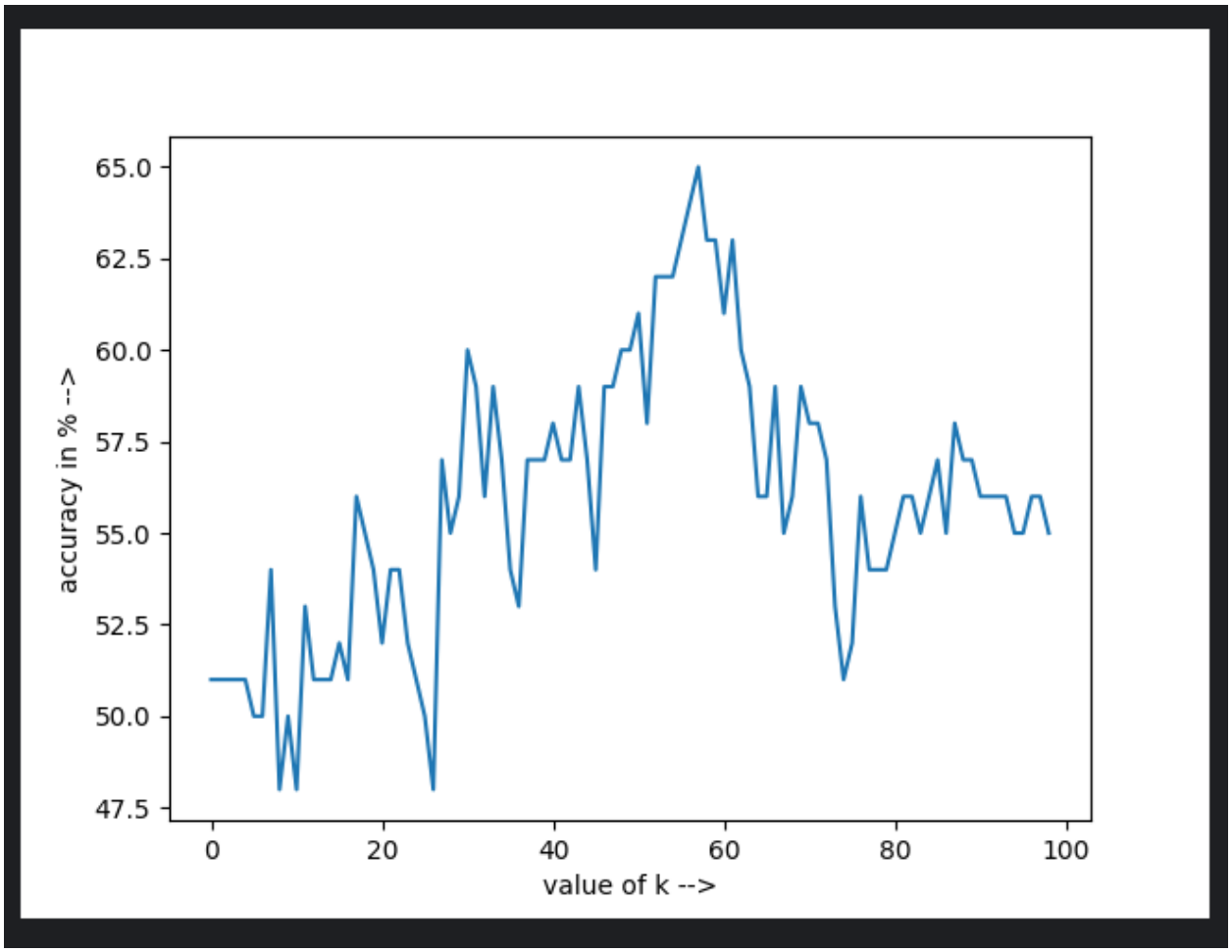
Berekent de afstanden tussen testdatapunten en trainingsdatapunten, sorteert op afstand en bepaalt het seizoen op basis van de meest voorkomende seizoenen bij de dichtstbijzijnde buren.

Resultaten

Bij het runnen van de onderdelen van K Nearest Neighbours heb ik de volgende resultaten behaald.

Deel 1

Dit was het stuk waar ik géén normalisatie heb toegepast op de datasets. Hieronder heb ik een grafiek van percentage juist ingedeelde datapunten, tegenover het aantal K neighbours. Wel leuk om op te merken is dat de optimale k hier ligt op ongeveer 55~60.

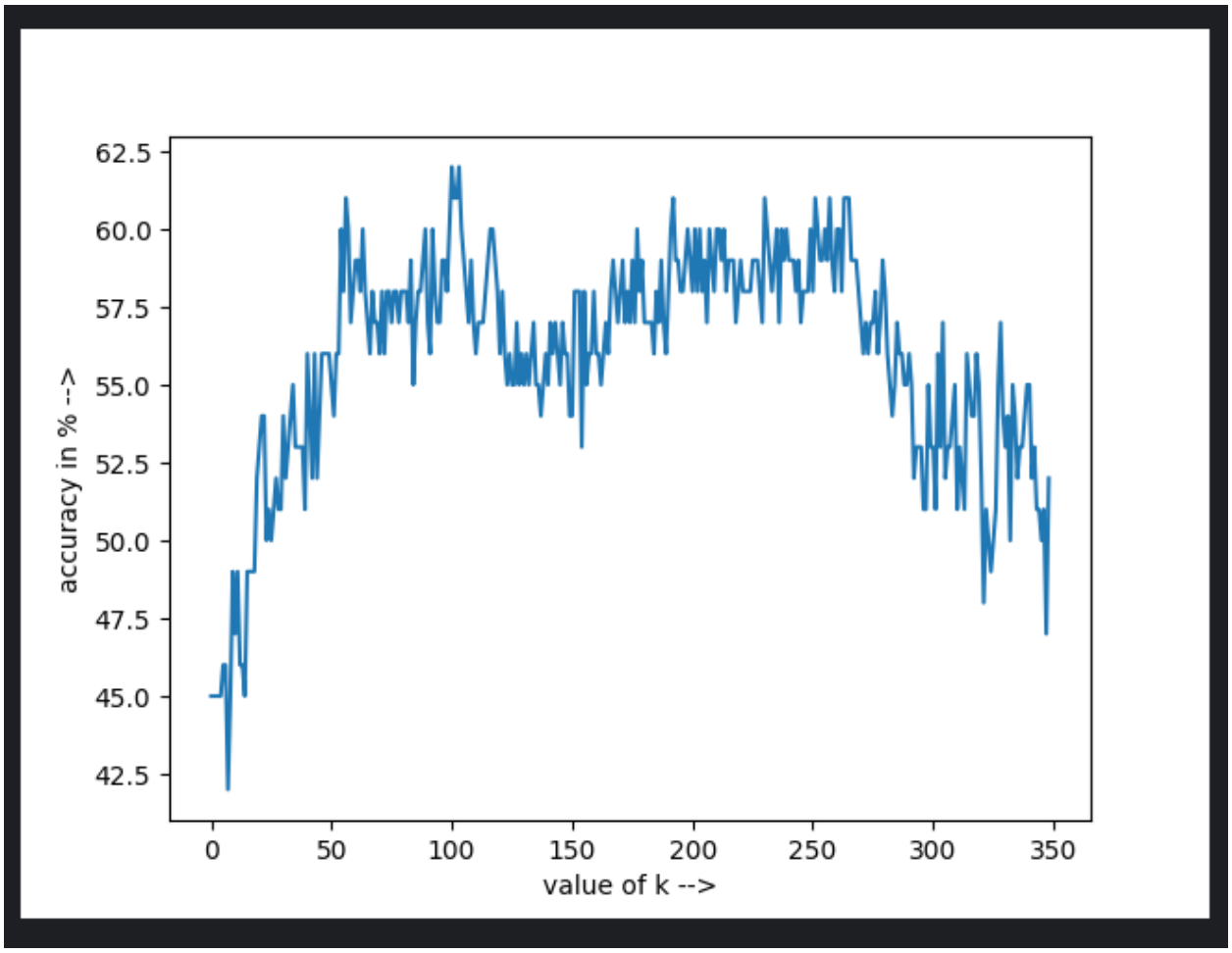


Deel 2

Dit was het stuk waarbij ik alle gegeven datasets heb gecombineerd, en heb genormaliseerd. Van de mins en maxes uit de verschillende datasets, heeft mijn functie de kleinste en grootste gepakt, zodat álle punten hierbinnen liggen. Hieronder zie je een zelfde grafiek als in deel 1. Ik heb er voor gekozen hier langer door te rekenen met:

K_{max} = len(dataset)

Dit kost weliswaar meer tijd, maar omdat ik zag dat de optimale k niet hetzelfde was als bij deel 1, wilde ik zien wat de trend deed.



Deel 3

Bij onderdeel 3 heb ik de dagen geprobeerd te klassificeren. Ook hier heb ik het getest *MET* én *ZONDER* normalisatie. Bij de *niet* genormaliseerde dataset heb ik k=56 gebruikt en bij de *wel* genormaliseerde dataset heb ik k=100 gebruikt. Het interessante hier is dat de resultaten maar één keer verschillen van elkaar.

```
The 9 assumed seasons for days.csv (No normalisation used)
['lente', 'lente', 'lente', 'zomer', 'lente', 'zomer', 'winter', 'winter', 'zomer']

The 9 assumed seasons for days.csv (Normalisation used)
['lente', 'winter', 'lente', 'zomer', 'lente', 'zomer', 'winter', 'winter', 'zomer']
```

K Means

Voor K Means heb ik gekozen een recursief algoritme toe te passen. een alternatief zou zijn dezelfde aanpak te gebruiken als bij mijn KNN, maar omdat ik twijfelde of de numpy slicing etc wel goed toepaste heb ik deze (volledig andere) methode toegepast.

Call tree

Enkele van onderstaande functies zijn hergebruikt van bovenstaande opdracht. Deze functies zijn:

- Normalize_weather_data
- calculate_distance
- add_dates

```
├── normalize_weather_data
├── add_dates
└── accuracy
    ├── pick_centroids
    ├── means
    │   ├── calculate_distance
    │   └── assign_labels
    └── validate_clusters
        └── calculate_distance
```

Beschrijving implementaties

accuracy()

Parameters:

dset, k, max_iter=10

Return type:

float

beschrijving:

Berekent de laagste foutmarge door het kMeans-algoritme meerdere keren uit te voeren. De foutmarge houdt hier in:

cumulatieve afstand van punten tot bijbehorende cluster centroid gedeeld door totaal aantal datapunten.

Werking:

Het voert het kMeans-algoritme uit met willekeurige centroids en returnt de laagste foutmarge.

pick_centroids()

Parameters:

dset, k

Return type:

np.ndarray

Beschrijving:

Kiest willekeurige centroids voor een gegeven aantal k en zorgt ervoor dat ze uniek zijn.

Werking:

Selecteert willekeurige punten als centroids, waarbij wordt gecontroleerd of ze uniek zijn.

assign_labels()

Parameters:

dset, centroids)

Return type:

np.array

Beschrijving:

Kent labels toe aan elk datapunt op basis van de afstand tot elk centroid.

Werking:

Berekent de afstanden tussen datapunten en centroids, wijst labels toe op basis van de kortste afstand.

means()

Parameters:

dset, centroids

Return type:

Union[List[List[np.ndarray]], np.ndarray]

Omschrijving:

Implementeert het kMeans-algoritme door datapunten toe te wijzen aan dichtstbijzijnde centroides en centroides bij te werken.

Werking:

Voert het kMeans-algoritme uit, waarbij het herhaaldelijk datapunten toewijst aan de dichtstbijzijnde centroides en de centroides bijwerkt.

validate_clusters()

Parameters:

clusters: List[List[np.ndarray]], centroids: np.ndarray

Return type:

float

Omschrijving:

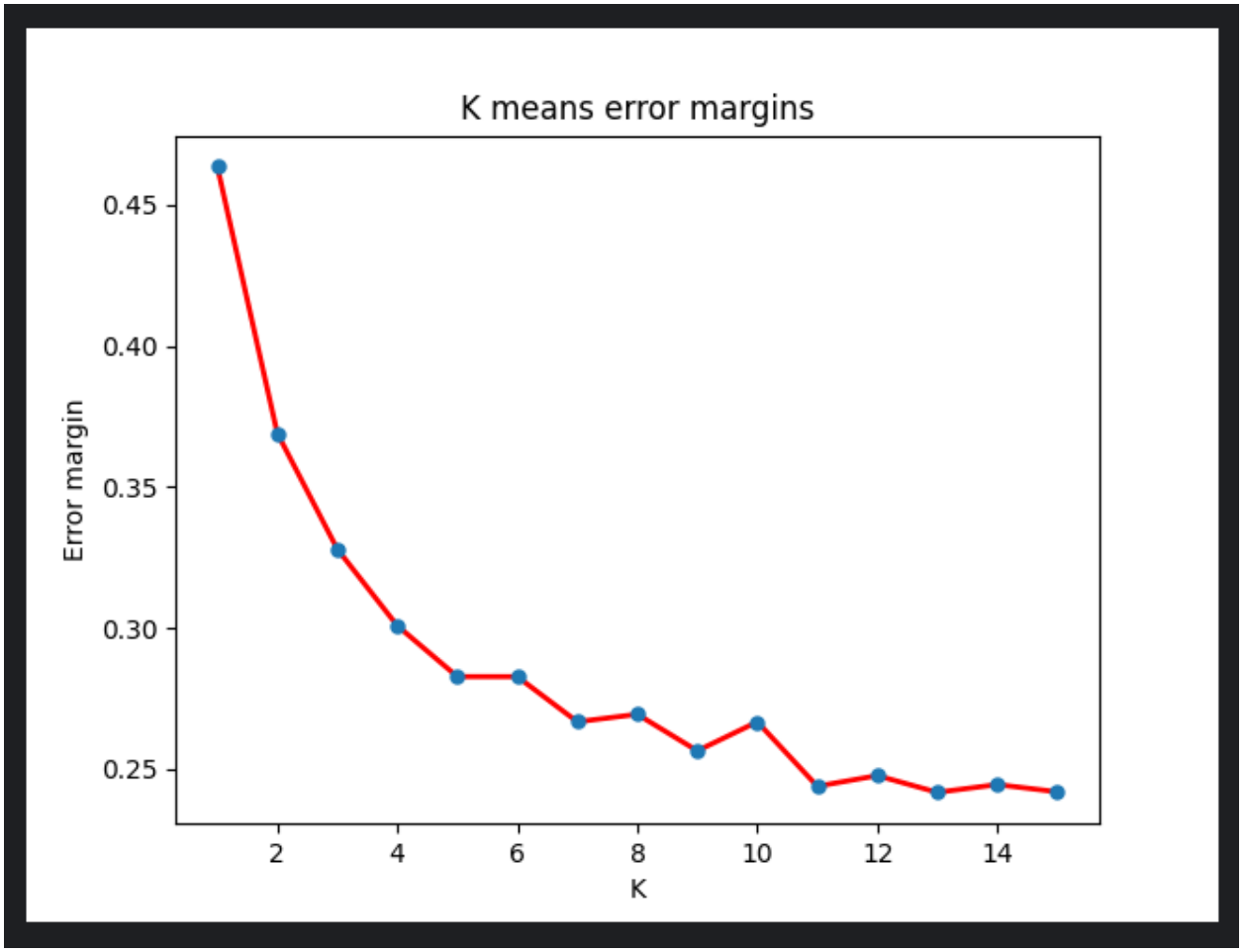
Berekent een foutmarge door de totale cumulatieve afstand van alle clusterpunten tot het clustercentrum te delen door het totale aantal punten.

Werking:

Berekent de foutmarge voor de gegeven clusters en centroids.

Resultaten

Bij de means opdracht was ik meteen tevreden met mijn resultaten, dus heb ik het gehouden op testen met enkel de genormaliseerde dataset. Hieruit kreeg ik de volgende Scree grafiek:



Dit is een mooi bewijs dat de data inderdaad waarschijnlijk 4 seizoenen bevat. Wij wisten dat natuurlijk aan de hand van de labels al, maar mijn algoritme is getest zonder labels!