k-Means
Daniel van Eijk-Bos en Rick van Mourik

Code uitleg:

...load_data()

...find_min_max()

...normalize()

...normalize()

...calculate_optimal_k()

.....pinpoint_season()

.....calculate_final_centroids

.....create_starting_centroids()

.....calculate_clusters()

.....calculate_centroid_diff()

.....calculate_distance_to_centroids()

.....calculate_centroid_location()

.....get_centroid_seasons()

.....calculate_distance_to_centroids()

.....inter_cluster_distance()

.....success_rate_calculation()

.....plot_k

In de **main()** functie wordt als eerste de dataset ingelezen (samen met de bijbehorende dates en labels) en opgeslagen in variabelen voor later gebruik. Daarna kan je kiezen tussen het inladen van de dagen uit validatieset (samen met de bijbehorende dates en labels), of het inladen van de test data (zonder bekende dates en labels). De dates die eventueel ingelezen worden, worden rechtstreeks uit de data gehaald en de bijbehorende labels worden berekend in de **load_data()** functie.
Wanneer de data is ingelezen en opgeslagen, wordt deze genormaliseerd door middel van de **normalize()** functie. Deze functie gebruikt de minimale en maximale data die uit de originele data wordt gehaald met behulp van de **find_min_max()** functie. De **normalize()** functie gebruikt de **var – var_min / var_max – var_min** formule om voor elk datapunt in een dag de genormaliseerde versie hiervan te berekenen en op te slaan.
Afhankelijk van de wat je wilt doen (het algoritme testen met **validation1.csv** of seizoenen berekenen voor nieuwe dagen uit **days.csv**) gebeuren er nu verschillende dingen. (Ook is er de optie om de

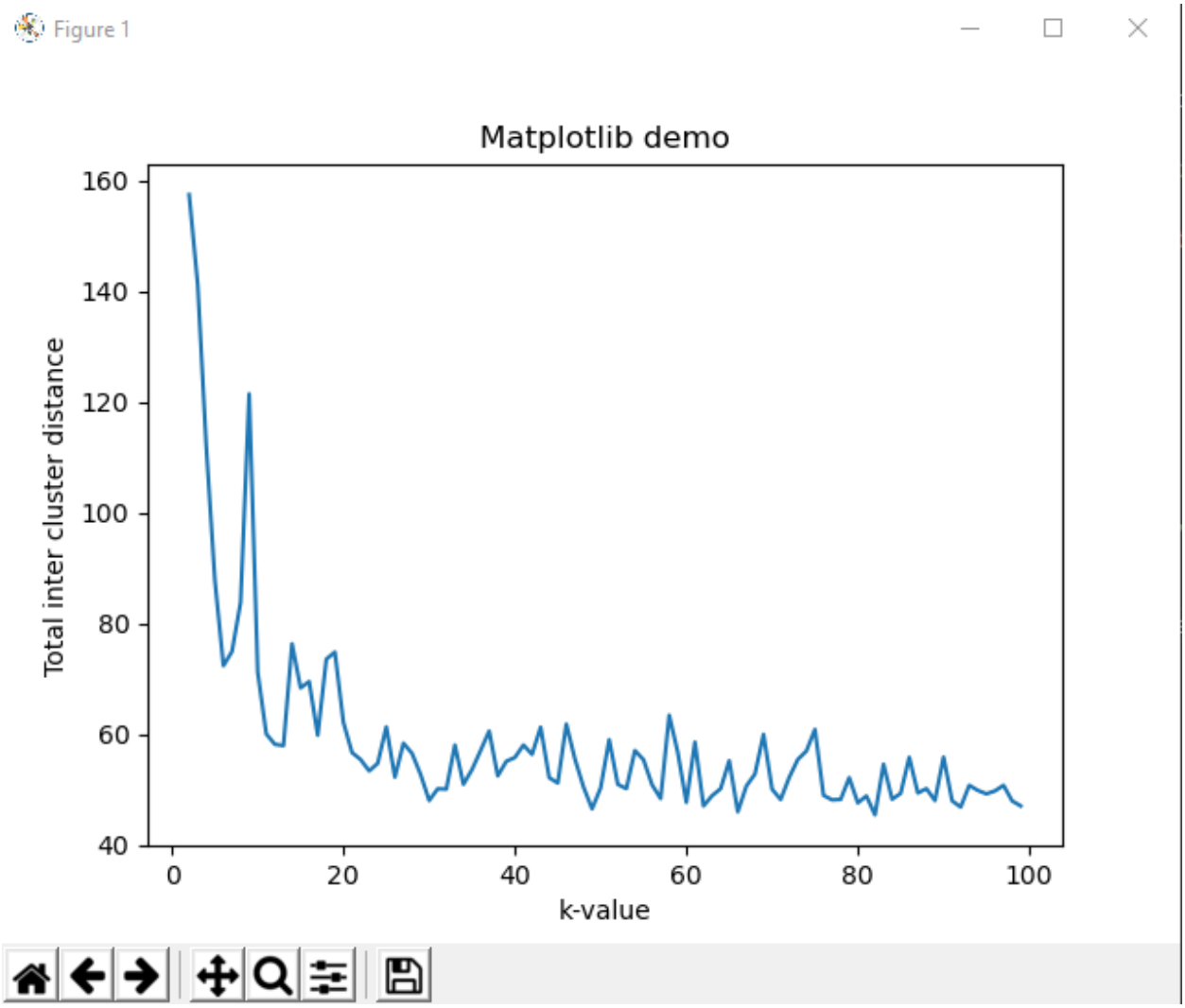
optimale k-value te berekenen, maar dit is onderdeel van het validatieproces) Bij het analyseren van nieuwe dagen wordt **pinpoint_season()** aangeroepen, deze geeft de uiteindelijke resultaten terug. In deze functie worden eerst alle bestaande clusters en centroids aangemaakt en opgeslagen aan de hand van **calculate_final_centroids()**. **Calculate_final_centroids()** maakt als allereerst gebruik van **create_starting_centroids()**, deze functie berekend k-aantal willekeurige centroids, en geeft deze terug. **Calculate_final_centroids()** gaat hierna verder met een while-loop die door blijft gaan zolang de huidige centroids en de vorige iteratie aan centroids niet hetzelfde zijn. In deze while-loop worden eerst de nieuwe clusters en distances berekent met **calculate_clusters()** en maakt daarna nieuwe centroids aan door middel van **calculate_centroid_location()**.
Calculate_clusters() maakt nieuwe clusters aan, hoe dit gebeurt hangt af van of het de eerste keer clusters maken is of niet. Als het de eerste keer clusters maken is, dan gebeurt er niks speciaals. Nieuwe clusters worden gemaakt in een list (met dezelfde volgorde als de centroids) en elke entry van de list is weer een list met alle indexes die verwijzen naar de dataset entries die in die cluster horen. Welke dataset entry in welke cluster hoort wordt bepaald door welke centroid het dichtst bij deze dataset entry is, dit wordt gedaan met behulp van de **calculate_distance_to_centroids()** functie. Ook wordt het afstandsverschil tussen de dichtstbijzijnde centroid en de 1 na dichtstbijzijnde centroid opgeslagen (op dezelfde manier als clusters). Als dit niet de eerste keer cluster maken is, dan wordt er allereerst berekend hoeveel de centroids van plek verschoven zijn met **calculate_centroid_diff()**. Voor elke entry in de oude clusters (de clusters die horen bij de oude, niet-verschoven centroids) wordt gecontroleerd of de corresponderende distance entry hoger of lager is dan de berekende centroid verschillen. Als deze entry lager is, gebeurt er niks speciaals en worden de dichtstbijzijnde centroid en afstand tussen de dichtstbijzijnde en 1 na dichtstbijzijnde centroid opnieuw berekend en opgeslagen. Als de entry hoger is dan de berekende centroid verschillen, dan kan ervan uitgegaan worden dat de huidige dataset entry nog steeds bij de huidige centroid hoort, en wordt dit zo opgeslagen. Dit scheelt veel rekentijd.
Calculate_centroid_location() berekent de nieuwe centroid locaties gebaseerd op deze nieuwe clusters. Hier wordt de gemiddelde locatie van alle entries in een cluster berekend en aan de corresponderende cluster toegewezen.
Als de centroid niet meer van locatie veranderen, worden deze en de bijbehorende clusters door **calculate_final_centroids()** terug gegeven.
In **pinpoint_season()** wordt na **calculate_final_centroids()**, **get_centroid_seasons()** aangeroepen. Deze functie wijst voor elke centroid aan de hand van de corresponderende cluster aan bij welk seizoen deze hoort.
Voor elke gegeven dag in **pinpoint_season()**, wordt **calculate_distance_to_centroids()** aangeroepen en wordt het seizoen van de dichtstbijzijnde centroid opgeslagen.
Als alle resultaten binnen zijn, geeft **pinpoint_season()** niet alleen de resultaten maar ook de clusters terug.
Wanneer er data gevalidate wordt, wordt **calculate_optimal_k()** aangeroepen, deze roept voor de geselecteerde k-waarden **pinpoint_season()** aan en berekend daaruit het hoogste slagingspercentage en de bijbehorende k-waarde, deze worden geprint. Daarnaast wordt er ook getest hoelang het hele proces duurt, voor vergelijking met een niet geoptimaliseerde versie. Er wordt ook een grafiek geplot van alle k-waardes tegenover de totale afstanden binnen een cluster, deze totale afstanden zijn het totaal van alle afstanden van de punten binnen het cluster tot de centroid van de cluster.

Resultaten:

De berekende seizoenen voor de onbekende dagen in days.csv (in beide main.py en old_main.py):
['winter', 'winter', 'herfst', 'zomer', 'winter', 'zomer', 'winter', 'winter', 'zomer']

De ideale berekende k-waarde is 74 met 60% slagingskans voor main.py.

Het verschil tussen main.py en old_main.py is dat main.py een snellere versie van het algoritme bevat en ook de scree grafiek plot.



K=6 is de optimale grootte van de dataset omdat hier de data betrouwbaar is.