k-Nearest Neighbours

Door Joost van Bussum en Niels Risseeuw

Uitleg van de code:

Call tree

findOptimalK
kNN
CalculatePlotDistance
decideLabel

totaal-uitleg

Om een optimale K te vinden voor het k-Nearest-Neighbours algortime wordt de functie **findOptimalK** aangeroepen. Deze functie test waardes van k van 1 tot en met een zelf aangegeven getal om daar uiteindelijk het percentage goed voorspelde seizoenen te berekenen.

Vanuit **findOptimalK** wordt de functie **kNN** aangeroepen, deze functie geeft een lijst terug met de voorspelde seizoenslabels en de juiste seizoenslabel. Deze functie berekent de afstand van één plot (een feature vector van 7 elementen) met zijn k aantal nearest neighbours om daar vervolgens een seizoenslabel uit te halen. Dit wordt gedaan met onder andere de functie **calculatePlotDistance.**

In **calculatePlotDistance** wordt de afstand tussen twee plots berekent met de stelling van Pythagoras. Deze afstand wordt verder gebruikt in de **kNN** functie.

Vervolgens wordt in de **kNN** functie deze afstand met de voorspelde label in een 2-dimensionale array(**distance** genaamd) gestopt. Dit gaat door tot alle afstanden berekent zijn. Vervolgens wordt de **distance** array gesorteerd en gesliced tot een **k** aantal elementen. Vervolgens wordt met deze array de functie **decideLabel** aangeroepen om een definitief seizoenslabel aan de distance toe te voegen. Deze wordt in een array gestopt, genaamd **validatedLabels,** deze array bestaat uit de definitief gekozen label en de daadwerkelijke label.

Vervolgens komt deze data weer terug in de **findOptimalK** functie om een correctheidspercentage uit te rekenen voor de specifieke **k.**

Resultaten

De resultaten van de **findOptimalK** functie zijn als volgt(er wordt gekeken naar waardes van 1 tot 100 van **k**), de 3 beste waardes van **k** worden hieronder getoond(de waarde van **k** links en het percentage correctheid rechts) met de beste waarde als eerst. Hieruit is de halen dat met onze implementatie van het kNN algoritme met een **k** van 61 een correctheidspercentage halen van 62%.

```
[Running] python -u "c:\Users\Joost\Documents\C++ shizzle\AppliedAI\opdracht_1.py"
[[61, 62.0], [21, 61.0], [47, 61.0]]
```

Antwoord op de opdracht

Door te kijken naar de resultaten in het vorige kopje kunnen we concluderen dat de beste waarde voor k in onze implementatie van het kNN algoritme 61 is. Dit aangezien het het hoogste slagingspercentage had van alle waardes tussen 1 en 100. Hieruit kunnen wij ook concluderen dat onze implementatie een errorpercentage heeft van 38%.