

# Machine Learning – Opgaven

## week 1

### Inleiding

Deze week werken we aan de lineaire regressie van data met één variabele. De data die we daarvoor gebruiken is de winst van een vervoerder gerelateerd aan de grootte van een stad waar die vervoerder werkzaam is. Je kunt je voorstellen dat het nuttig is om deze verhouding te weten, omdat je op basis hiervan geïnformeerd een besluit kunt nemen of je in een nieuwe stad (met een bepaalde grootte) een nieuw filiaal wilt openen.

De startcode van deze opgave [kun je hier downloaden](#). Het bestand `week1_data.pkl` bevat de data waar we mee gaan werken. Dit is een  $97 \times 2$ -NumPy-vector: de eerste kolom bevat de grootte van de steden, de tweede kolom bevat de winst van de vervoerder.

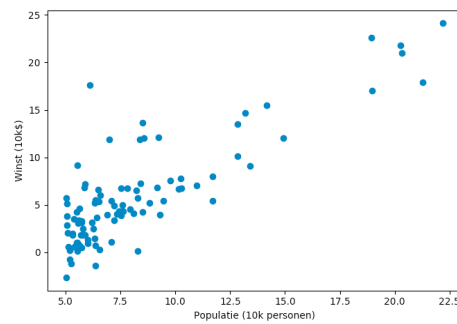
Het is de bedoeling dat je het bestand `uitwerkingen.py` afmaakt. Dit bestand wordt gebruikt door `exercise1.py` waarin de opgaven sequentieel worden doorlopen. Bestudeer beide bestanden om te weten hoe ze werken en wat de bedoeling ervan is.

### 1. Het visualiseren van de data

Een eerste stap in elk machine-learning project is een beeld creëren van de data. Het eenvoudigst om dit in dit geval voor elkaar te krijgen is door gebruik te maken van [de library matplotlib](#). Hierin vind je [een API pyplot](#) waarmee je vrij eenvoudig een *scatter plot* kunt maken.

Bestudeer de documentatie van `pyplot` en implementeer aan de hand hiervan de functie `drawGraph` in `uitwerkingen.py`, zodat je een afbeelding zoals de onderstaande krijgt. Hoewel de

data niet echt normaal verdeeld is, is er wel een zekere verhouding waar te nemen tussen de grootte van de stad en de winst die de vervoerder maakt. In de rest van deze opgave gaan we deze verhouding bepalen.




Als het tekenen van de scatter plot gelukt is, kun je deze opgave verder overslaan door aan het aanroepen van `exercisel.py` de command-line optie `skip` mee te geven: dit tekenen wordt dan overgeslagen.

## 2. Het bepalen van de **COST** function (de kostenfunctie)

Zoals in de theorieles besproken is, is het bepalen van de verhouding tussen twee (of meer) grootheden een kwestie van het minimaliseren van de kostenfunctie: de som van de gekwadrateerde fouten (SSE). Dit minimaliseren doe je door middel van *gradient descent* en het is dikwijls nuttig om die kostenfunctie gedurende de iteraties bij te houden, zodat je de verschillende uitkomsten door je data heen kunt plotten – zo kun je bijvoorbeeld controleren of je niet in een lokaal minimum terecht bent gekomen. In deze opgave programmeren we de kostenfunctie verder uit; de volgende opgave gaat verder in op de *gradient descent*.

De kostenfunctie is gegeven door de volgende formule:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Hierin is  $J(\theta)$  de totale kost die berekend wordt met de huidige waarden van  $\theta$  ;  is de hypothese van de waarde (de voorspelling) en  $y$  is de actuele waarde. Door het verschil tussen deze twee waarden voor elk datapunt op te tellen en uiteindelijk uit te middelen, komen we op de voorspellende waarde die de formule heeft met de huidige waarden van  $\theta$ .

De algemene formule voor de hypothese (de voorspelling) is als volgt:

$$h_{\theta}(x) = \theta^T x = \theta_0 + \theta_1 x$$

Omdat we op zoek zijn naar een lijn, hebben we feitelijk te maken met één parameter (een lijn is immers  $y = b + ax$ ). Om de dimensionaliteit van de trainingsdata te laten corresponderen met  $\theta$ , voegen we een rij van enen toe.

```
m,n = data.shape
X = np.c_[np.ones(m), data[:, [0]]]
y = data[:, [1]]
theta = np.zeros( (2, 1) )
```

De voorspelde waarden in  $X$ , de actuele waarden in  $y$  en een  $\theta$  worden aan de methode `computeCost` meegegeven; deze functie moet de waarde van  $J(\theta)$  teruggeven. Implementeer deze functie op basis van de beschrijving hierboven in het bestand `uitwerkingen.py` (zie ook de aanwijzingen in het bestand zelf); maak hem zo, dat hij werkt voor elke grootte van  $\theta$  (een vectoriële implementatie).

Als je klaar bent, kun je het bestand `opgaven.py` runnen. Dit bestand roept `computeCost` aan en print de gevonden waarde uit. Als het goed is, krijg je uiteindelijk een waarde van rond de 32.07.

### 3. Gradient descent

Als laatste maak je de methode `gradientDescent` in het bestand `uitwerkingen.py` af. In deze methode wordt een aantal stappen uitgevoerd, waarbij in elke stap de vector  $\theta$  volgens de onderstaande formule wordt aangepast.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

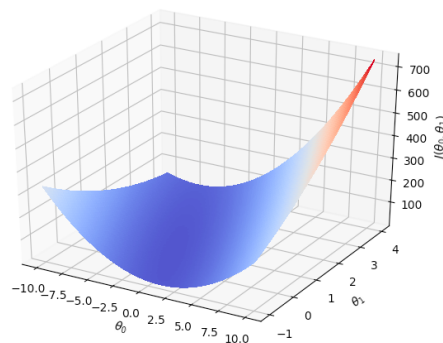
Als het goed is, zorgt elke stap in deze methode ervoor dat

**Invalid Equation** lager wordt. Let er wel op dat je alle  $\theta_j$  tegelijkertijd aanpast (in dit geval is de grootte van  $\theta$  2, dus elke iteratie moeten er twee parameters worden aangepast). Let er verder op dat je *alleen* de  $\theta$  aanpast:  $X$  en  $y$  zijn constante waarden die niet hoeven te worden aangepast.

De algemene structuur in het bestand is al gegeven. Als je implementatie klaar is, kun je opnieuw het bestand `opgaven.py` aanroepen; deze roept de functie `gradientDescent` zodat de update 1500 keer wordt gedaan. Als het goed is, is  $\theta$  uiteindelijk rond de  $(-3.63, 1.16)$ .

## 4. Contour plot

In deze laatste opgave gebruik je de methode `computeCost` die je in de eerste opgave hebt gemaakt om een contour-plot van de kosten te tekenen. Hierdoor kun je inzicht krijgen in hoe deze waarde zich ontwikkelt bij verschillende waarden van  $\theta$ . Het grootste deel van deze opgave is al in de methode `showContour()` in het bestand `uitwerkingen.py` gegeven; je hoeft alleen maar de waarden van de matrix `J_val` te vullen. Bestudeer het commentaar in het bestand voor meer toelichting. Als je klaar bent, roept het bestand `exercise1.py` de methode `showContour()` aan om de plot te tekenen. Als het goed is, ziet deze er ongeveer als hieronder uit.



## Beoordeling

Opgave	Percentage
data-visualisatie	15

cost function	30
gradient decent	40
contour plot	15