

Machine Learning & AI

Tuur Vanhoutte

15 april 2021

Inhoudsopgave

1 Inleiding	1
1.1 AI in context	1
1.1.1 Vormen van AI	1
1.1.2 Sectoren die de planeet verbeteren	2
1.1.3 Waarom nu?	2
2 Hoe leren uit data?	2
2.1 Leeralgoritmes	2
2.2 Supervised Learning	3
2.2.1 Regressie vs Classificatie	4
2.2.2 Voorbeeld	4
2.3 Unsupervised learning	5
2.4 Reinforcement learning	6
2.5 Overzicht leeralgoritmes	6
2.6 Werkwijze van een ML Project	7
2.6.1 Tijdverdeling	7
3 Enkelvoudige Lineaire regressie	8
3.1 Voorbeeld	8
3.1.1 Scatterplot	8
3.2 De hypothese	9
3.3 De kostenfunctie	9
3.4 Gradient Descent (GDS)	9
3.4.1 Learning rate	10
4 Meervoudige lineaire regressie	11
4.1 Statistische vooranalyse	12
4.1.1 Consistentie van de dataset	12
4.1.2 Uitschieters	12
4.1.3 Onderlinge correlatie	13
4.2 Features en targets	13
4.3 Trainen van het model	14
4.3.1 Initialiseren en trainen van het regressiemodel	14
4.4 Testen van het model	14
4.4.1 Voorspellingen maken	14
4.5 Performantie en scores van het model	15
4.5.1 Mean Absolute Error (MAE)	15
4.5.2 Mean Squared Error (MSE)	15
4.5.3 Determinatiecoëfficiënt	16
5 Feature engineering	16
5.1 Normalisatie / Scaling	16
5.1.1 Voordelen	17
5.1.2 MIN-MAX-scaling	17
5.1.3 Standard scaling (normalisatie)	18
5.1.4 Robust scaling	20
5.2 Feature expansion	20
5.2.1 Nieuwe features	20
5.2.2 Hogere-orde features	21
5.3 One-hot encoding	22

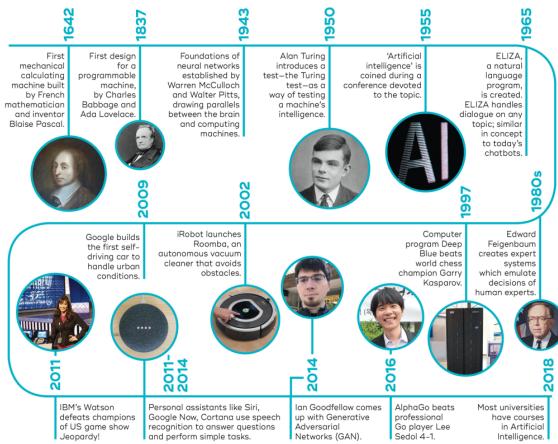
6 Underfitting & overfitting	23
6.1 Underfitting	23
6.2 Overfitting	23
6.2.1 Impact van de grootte van de dataset	24
6.3 Regularisatie (regularisation)	24
6.3.1 Voorbeeld regularisatie	24
6.3.2 Regularisatie met L2 norm	25
6.3.3 Regularisatie met L1 norm	25
6.3.4 Voorbeeld regularisatie op huizenprijzen	25
7 Classificatie	26
7.1 Wat is classificatie?	26
7.1.1 Voorbeelden	26
7.2 Types van classifiers	26
7.2.1 Binary (binomial) classifier	26
7.2.2 Multiclass classifier	27
7.2.3 Multilabel classifier	27
7.3 Voorbeeld van een classificatie: appel herkennen	28
7.3.1 Waarom lineaire regressie geen goede optie is	28
8 Logistische regressie	29
8.1 Het model	29
8.1.1 De logistische functie	30
8.1.2 Interpretatie via voorbeeld appels	30
8.1.3 Grafische interpretatie via voorbeeld appels	31
8.1.4 Wat als het model niet lineair is?	31
8.2 De kostenfunctie	32
8.3 Voorbeeld logistic regression met Sklearn (lineair model)	32
8.3.1 Preprocessing van de data	32
8.3.2 Trainen van het logistic regression model	33
8.3.3 Classificeren van een nieuwe sample	33
8.3.4 Visualiseer de decision boundary	34
8.4 Voorbeeld logistic regression met Sklearn - niet-lineair	35
8.4.1 Feature engineering: automatisch toevoegen van hogere orde features	35
8.4.2 Oplossing: regularisatie	35
8.5 Multi-class classification	36
8.5.1 One-vs-All	36
8.5.2 One-vs-One	37
9 Evaluatie van een classifier	37
9.1 Accuracy	38
9.2 True Positive Rate (TPR)	38
9.3 Positive Predictive Value (PPV)	38
9.4 F1 Score	38
9.5 Receiver Operating Characteristic (ROC)	38
9.5.1 ROC curve en AUC (Area Under ROC Curve)	39
10 Support Vector Machines (SVM)	41
10.1 Wat is een SVM?	41
10.1.1 Welke classifier zou je verkiezen?	41
10.2 Hoe een SVM classificeert	41
10.2.1 Wat als een perfecte lineaire scheiding niet mogelijk is?	42

10.2.2 Wat bij niet-lineair scheidbare gegevens	43
10.3 Kernels	44
10.3.1 Meest gebruikte kernels	44
10.3.2 Voorbeeld	44
10.3.3 Parameter gamma	45
10.3.4 Hyperparameters	45
10.4 Motivatie voor het gebruik van een SVM	46
10.5 Logistische regressie vs SVM	46
10.5.1 Wanneer welke classifier kiezen?	46
11 Cross-validatie	46
11.1 Verschillende types cross-validatie	46
11.1.1 Hold out cross-validation	46
11.1.2 K-fold cross-validation	47
11.1.3 Leave one out cross-validation	47
11.1.4 Bootstrap cross-validation	48
11.2 Hyperparameter tuning via cross-validatie	48
11.2.1 Grid search	48
11.2.2 Random search	49
11.2.3 Bayes optimization	49
12 Niet-gebalanceerde data	50
12.1 Problematiek	50
12.2 Omgaan met niet-gebalanceerde data	50
12.2.1 Undersampling & Oversampling	50
12.2.2 Andere scoring parameter/metric kiezen	51
12.2.3 Class-weight balancing	51
12.2.4 Data augmentation - SMOTE (Synthetic Minority Over-sampling Technique)	52
12.2.5 Data augmentation - image augmentation	52
13 Naive Bayes & Natural Language Processing	53
13.1 Discriminative vs generative classification	53
13.2 Bayes rule	53
13.2.1 Voorbeeld: kanker	53
13.2.2 Uitwerking via Bayes Rule	53
13.3 Naive Bayes - tekstclassificatie	54
13.3.1 Spamdetectie	54
13.3.2 Met Naive Bayes	55
13.3.3 Voorbeeld	55
13.4 Laplacian smoothing	56
13.4.1 Invloed van de hyperparameter alpha	56
13.5 Log likelihood	57
13.6 Tekstclassificatie in de praktijk	57
13.6.1 Preprocessing - opkuisen van de tekst	57
13.6.2 Preprocessing - herleiden van woorden tot de stam	58
13.6.3 Preprocessing - verwijder te korte woorden	58
13.6.4 Opbouwen van feature vectors - bag of words	58
14 Decision Trees	59
14.1 Decision Trees voor classificatie	59
14.1.1 Voorbeeld met rondheid & groenheid van appels	59
14.1.2 Entropy	59

14.1.3 Information Gain	60
14.1.4 Gini impurity	61
14.2 Decision Trees voor regressie	62
14.2.1 Voorbeeld 1: aantal golfspelers bij verschillende weersomstandigheden	62
14.2.2 Voorbeeld 2: Autoprijs op basis van 2 eigenschappen van de auto	63
15 Ensemble learning	63
15.1 Problematiek van decision trees	63
15.2 Overzicht Ensemble learning methodes	64
15.3 Bagging	64
15.3.1 Concept van bagging - Majority voting	64
15.3.2 De techniek van bagging	65
15.3.3 Random Forest Trees	66
15.4 Boosting	68
15.4.1 Adaboost	68
15.4.2 Gradient Boosting	70
15.5 Stacking	71
15.5.1 Folding	71
15.6 Overzicht Ensemble learning	72
16 Unsupervised learning	72
16.1 Introductie	72
16.2 Clustering	73
16.2.1 Voorbeelden	73
16.3 K-means clustering	74
16.3.1 Hoe gelijkenis uitdrukken?	75
16.3.2 Werking	75
16.3.3 Eigenschappen van K-means clustering	77
16.3.4 Strategie om het aantal clusters te bepalen	78
16.4 Hierarchical clustering	79
16.4.1 Sterktes van hierarchical clustering	79
16.4.2 Hoe ga je nu de afstand tussen 2 clusters bepalen?	79
16.4.3 Complexiteit van hierarchical clustering	80
16.5 Guassian mixture models	80
16.6 Spectral clustering	81
16.7 Self organizing maps	82
17 Dimensionality reduction	83
17.1 Wat?	83
17.2 Waarom?	83
17.3 Toepassingen	84
17.4 Principle Component Analysis (PCA)	84
17.4.1 Voorbeeld aan de hand van de MNIST dataset	85
17.4.2 Kiezen van het aantal principle components: verklaarde variantie	86
17.4.3 Face recognition	87

1 Inleiding

1.1 AI in context



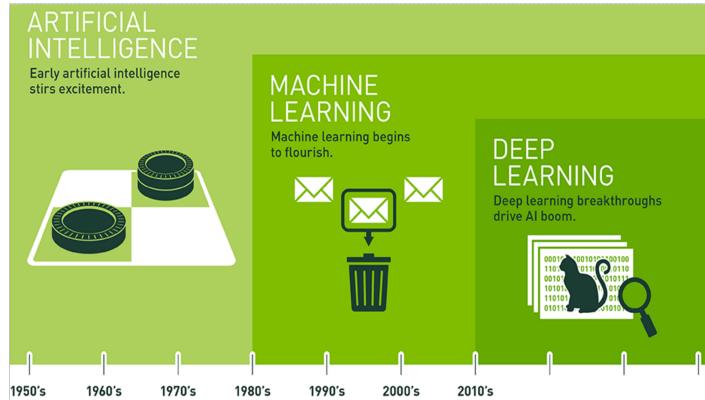
Figuur 1: Geschiedenis van AI

Belangrijkste gebeurtenissen:

- **1943:** McCulloch - Pitts: fundering van neurale netwerken
- **1950:** Alan Turing: de Turing test
- **1956:** Dartmouth workshop: bijeenkomst voor breinstorm AI
- **1997:** Garry Kasparov vs Deep Blue (IBM)
- **2011:** IBM Watson
- **2016:** AlphaGo
- **2021-:** toekomst

1.1.1 Vormen van AI

- Zwakke AI (weak AI / Artificial Narrow Intelligence)
 - Goed in een bepaalde taak maar alleen in die taak
 - **Voorbeelden:** spamfilters, schaakcomputers, gezichtsherkenning
- Sterke AI (strong AI / Artificial General Intelligence)
 - Intelligentie op menselijk niveau
 - In staat om zich aan te passen en problemen te leren oplossen in verschillende contexten
- Superintelligentie (Artificial Super Intelligence)
 - Als AI zelfbewust wordt en de mens op alle vlakken voorbij steekt



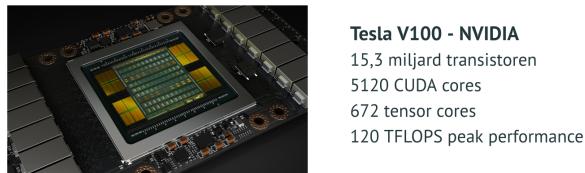
Figuur 2: AI vs ML vs DL

1.1.2 Sectoren die de planeet verbeteren

- Klimaatsverandering
- Biodiversiteit en conservatie
- Water
- Hernieuwbare energie
- Medische sector
- Weer- en rampenvoorspelling

1.1.3 Waarom nu?

- Snellere hardware
- Betere algoritmes
- Meer data
- (Open source) frameworks



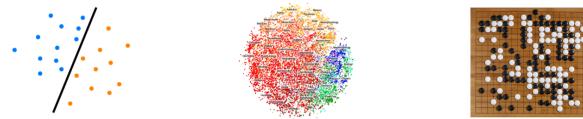
Figuur 3: Voorbeeld huidige hardware: de Tesla V100 van Nvidia

2 Hoe leren uit data?

2.1 Leeralgoritmes

- Supervised
 - Inputs met gewenste outputs zijn gegeven

- Task driven
- Unsupervised
 - De gewenste outputs zijn niet gegeven
 - Data driven (clustering)
- Reinforcement
 - Beslissingsproces op basis van beloningen
 - Algoritme leert te reageren op zijn omgeving



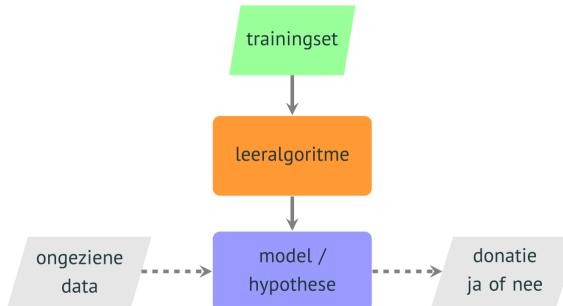
Figuur 4: Supervised / Unsupervised / Reinforcement learning

2.2 Supervised Learning

Leren uit een gelabelde dataset. Vind het verband tussen de features en de labels

		sentiment	text
0	1	1	I am going to start reading the Harry Potter series again because that is one awesome story.
1	1	1	the story of Harry Potter is a deep and profound one, and I love Harry Potter.
2	1	1	Mission Impossible 3 was excellent.
3	0	0	The Da Vinci Code sucked, but the night was great.
4	1	1	The Da Vinci Code was absolutely AWESOME!
5	0	0	Then snuck into Brokeback Mountain, which is the most depressing movie I have ever seen..
6	1	1	I love Harry Potter.
7	0	0	Ok brokeback mountain is such a horrible movie.
8	1	1	He's like,"YEAH I GOT ACNE AND I LOVE BROKEBACK MOUNTAIN".
9	0	0	Da Vinci Code sucks.

Figuur 5: Leren uit een dataset

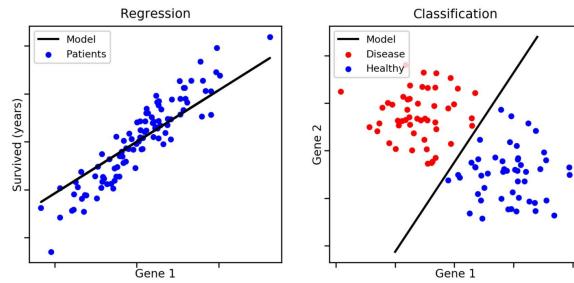


Figuur 6: Supervised learning kan uit ongeziene data een resultaat berekenen

2.2.1 Regressie vs Classificatie



Figuur 7: Regressie vs classificatie

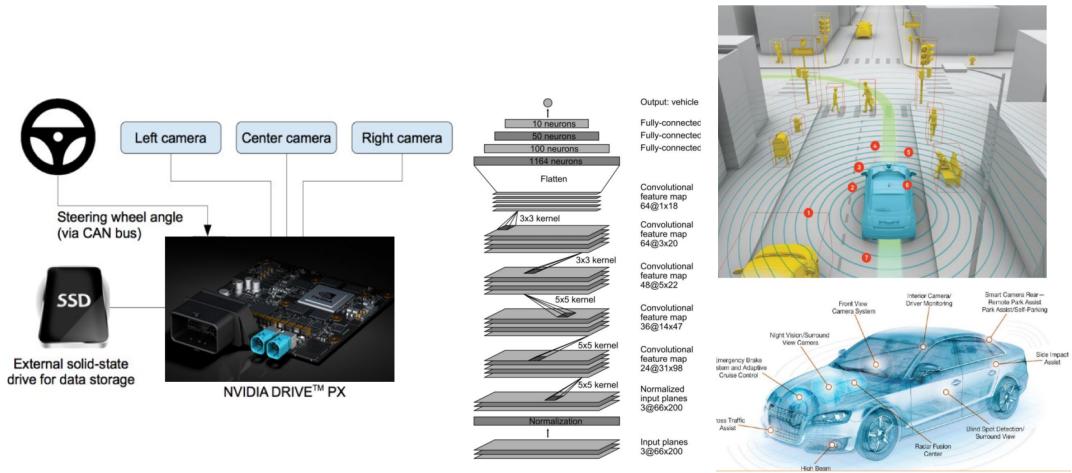


Figuur 8: Regressie vs classificatie

2.2.2 Voorbeeld

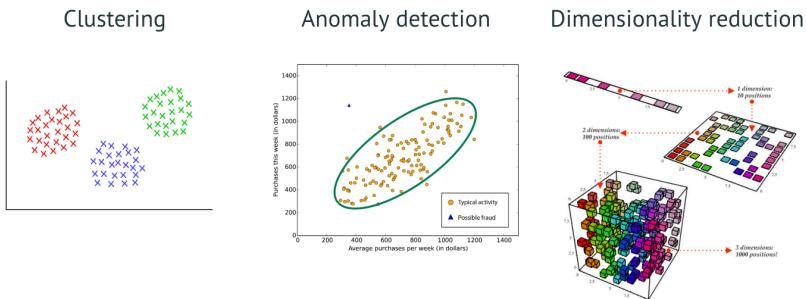
Hoe stuurhoek bepalen bij een self-driving car?

- (infrarood) camera's
- Stereo vision
- Radar
- LIDAR
- GPS
- Audio

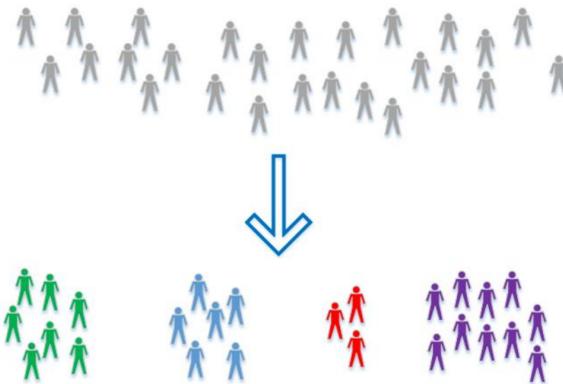


Figuur 9: Via sensoren weet de auto

2.3 Unsupervised learning

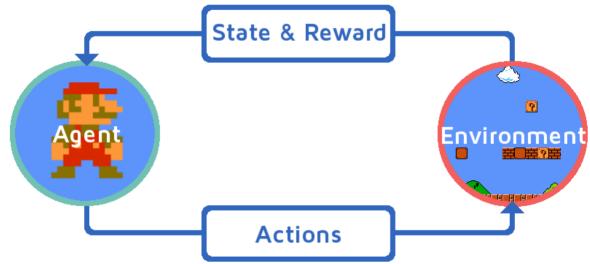


Figuur 10: Unsupervised Learning



Figuur 11: Voorbeeld Clustering: de data in groepen verdelen

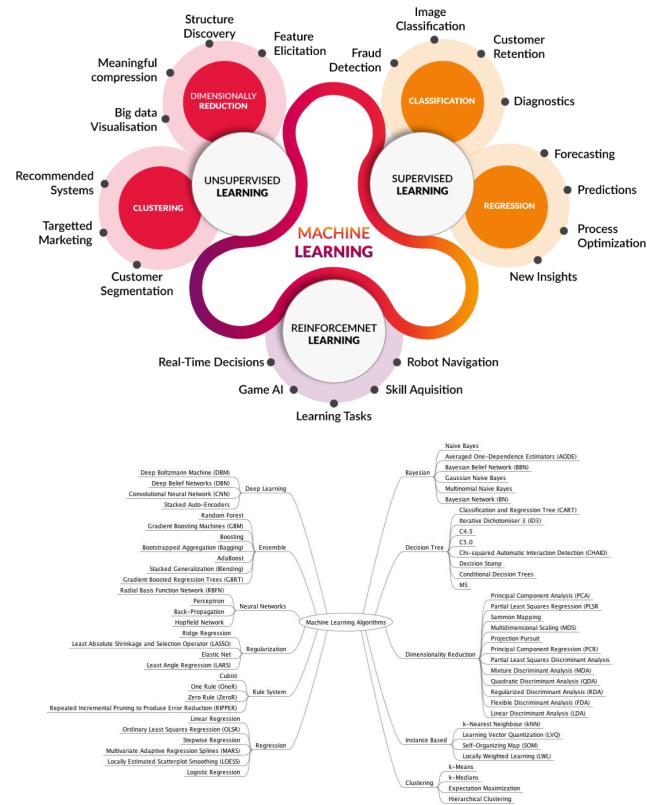
2.4 Reinforcement learning



Figuur 12: Reinforcement learning

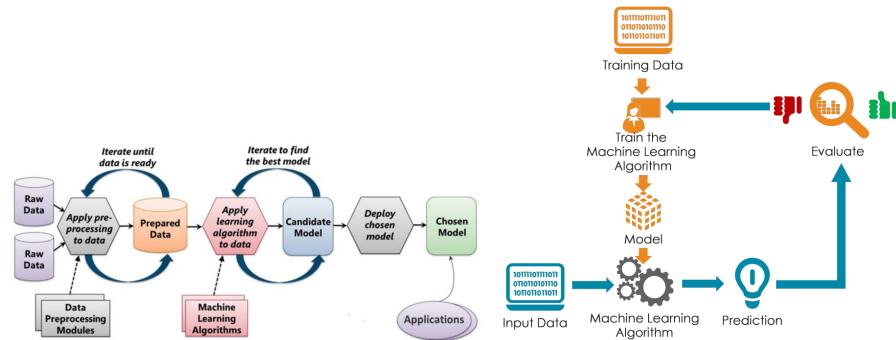
- Voor elke actie krijgt de AI feedback
- De AI leert uit de feedback
- In het begin zijn de acties heel willekeurig

2.5 Overzicht leeralgoritmes



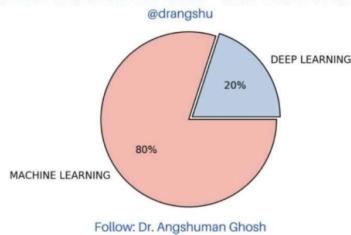
Figuur 13: Overzicht

2.6 Werkwijze van een ML Project



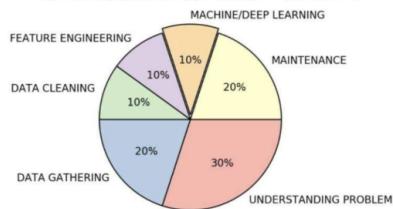
2.6.1 Tijdverdeling

DATA SCIENTIST JOB - EXPECTATION



Follow: Dr. Angshuman Ghosh

DATA SCIENTIST JOB - REALITY



Figuur 14: Tijdverdeling: verwachting vs realiteit

3 Enkelvoudige Lineaire regressie

3.1 Voorbeeld

	leeftijd	gewicht	bloeddruk
0	52	78	132
1	59	83	143
2	67	88	153
3	73	96	162
4	64	89	154
5	74	100	168
6	54	85	137
7	61	85	149
8	65	94	159
9	46	76	128
10	72	98	166

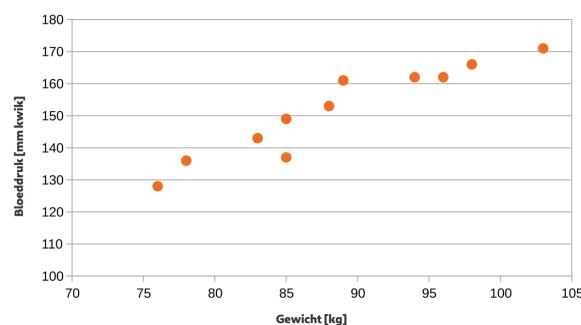
Figuur 15: Voorspel de bloeddruk op basis van leeftijd en gewicht

- **features:** leeftijd en gewicht
- **target:** bloeddruk (=wat je wil voorspellen = output = label)
- **trainingset:** 11 training examples (=samples)

Definitie 3.1 (Regressie-analyse) *Regressie-analyse is het modelleren van of het zoeken naar een verband op basis van één of meerdere variabelen.*

Bij regressie is de output/target een (continue) variabele

3.1.1 Scatterplot



Figuur 16: Scatterplot: de grafiek toont een positieve correlatie ⇒ een sterk verband

3.2 De hypothese

Definitie 3.2 (De hypothese) Het verband (model of hypothese) $h_\theta(x)$ is van de vorm:

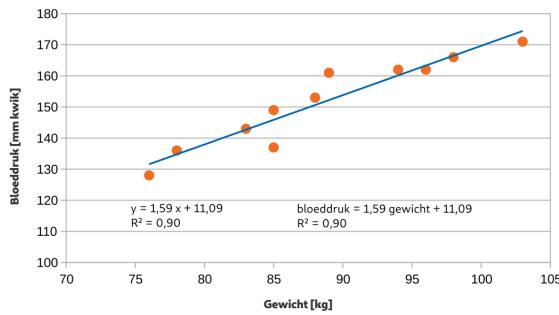
$$h_\theta(x) = \theta_0 + \theta_1 x \quad (1)$$

Bepalen van de optimale waarden voor θ_0 en θ_1 :

- θ_0 = snijpunt van de y-as (= noemen we ook de **bias**)
- θ_1 = helling van de rechte (rico)

De parameters θ_i = **weights**

Het zoeken van het model / hypothese = **training / learning**



Figuur 17: Lineaire trnedlijn met model $h_\theta(x)$

R²-waarde: determinatiecoëfficiënt

3.3 De kostenfunctie

We minimaliseren de kostenfunctie $J(\theta)$ via de **Least Mean Squared** methode (LMS).

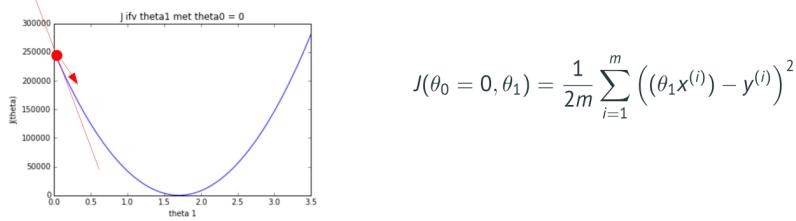
$$J(\theta) = \frac{1}{2 \cdot m} \cdot \sum_{i=1}^m (h_\theta(x_i) - y_i)^2 \quad (2)$$

- m = de bias = snijpunt met de y-as
- De kostenfunctie berekent de gemiddelde fout door alle fouten op te tellen
- Elke fout wordt gekwadrateerd om:
 - negatieve waardes positief te maken
 - de fout uit te groten

3.4 Gradient Descent (GDS)

$$J(\theta_0, \theta_1) = \frac{1}{2 \cdot m} \cdot \sum_{i=1}^m ((\theta_1 \cdot x_i + \theta_0) - y_i)^2 \quad (3)$$

Stel de parameters θ_0 en θ_1 voortdurend bij in een iteratief proces tot je de waarden voor θ_0 en θ_1 hebt gevonden die de kleinst mogelijke waarde. Start met willekeurige waarden.



Figuur 18: De GDS als $\theta_0 = 0$

- Je krijgt een dalparabool als uitkomst
- Je kan aflezen wat de parameters moeten zijn om de kleinst mogelijke waarde te vinden
- In de realiteit heb je vaak veel meer dan 2 gewichten
 - Voorbeeld: de textgenererende AI GPT-3 heeft rond de 175 miljard gewichten
 - ⇒ veel rekenkracht nodig om beste uitkomst te vinden

3.4.1 Learning rate

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m ((\theta_1 x^{(i)} + \theta_0) - y^{(i)})^2$$

Bepaal de gradient naar θ_0 en θ_1

$$\frac{dJ(\theta)}{d\theta_0} = \frac{2}{2m} \sum_{i=1}^m ((\theta_1 x^{(i)} + \theta_0) - y^{(i)})$$

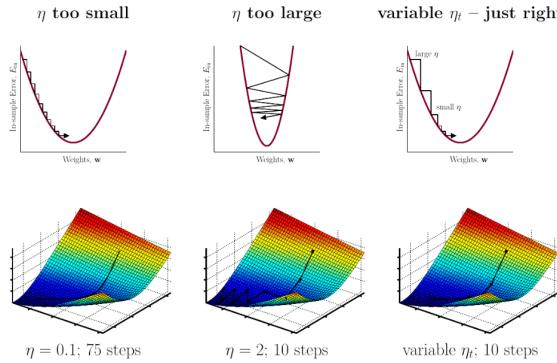
$$\frac{dJ(\theta)}{d\theta_1} = \frac{2}{2m} \sum_{i=1}^m ((\theta_1 x^{(i)} + \theta_0) - y^{(i)}) x^{(i)}$$

Update de parameters θ_0 en θ_1 volgens:

$$\theta_0 := \theta_0 - \eta \frac{dJ(\theta)}{d\theta_0} \quad \text{en} \quad \theta_1 := \theta_1 - \eta \frac{dJ(\theta)}{d\theta_1}$$

Figuur 19: De parameters θ_0 en θ_1 stellen we constant bij (formules niet te kennen)

- We bepalen de afgeleide (= de gradient, de helling) van θ_0 en θ_1
- We gebruiken die afgeleiden om een betere θ_0 en θ_1 te vinden.
- We vermenigvuldigen de gradient met een variable η (=de learning rate)
- Onze nieuwe θ wordt berekend met behulp van de oude θ en de afgeleide maal de learning rate.



Figuur 20: Learning rate η : bij een te kleine/te grote η hebben we te veel stappen

De learning rate η stellen we constant bij om met zo weinig aantal stappen het optimum te bereiken.

4 Meervoudige lineaire regressie

Definitie 4.1 (Meervoudige lineaire regressie) Bij meervoudige lineaire regressie (*multiple regression*) wordt het model/hypothese bepaald aan de hand van een trainingset met **meerdere features**.

Voorbeelden:

- Bloeddruk wordt bepaald a.d.h.v. gewicht en leeftijd
- De kwaliteit van wijn wordt voorspeld op basis van: zuurtegraad, suikergehalte, chloriden, dichtheid, sulfaten, hoeveelheid alcohol, ...
- Het warmeverlies van een huis wordt voorspeld op basis van: het type glas, muurisolatie, oriëntatie van het huis, ...

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \quad (4)$$

- CRIM - per capita crime rate by town.
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise).
- NOX - nitric oxides concentration (parts per 10 million).
- RM - average number of rooms per dwelling.
- AGE - proportion of owner-occupied units built prior to 1940.
- DIS - weighted distances to five Boston employment centres.
- RAD - index of accessibility to radial highways.
- TAX - full-value property-tax rate per \$10000.
- PTRATIO - pupil-teacher ratio by town.
- B - $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town.
- LSTAT % lower status of the population.
- Price - Median value of owner-occupied homes in 1000's.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PT	B	LSTAT	Price
0	0.00632	18.0	2.31	0	0.538	6.575	65.199997	4.0900	1	296	15.300000	396.899994	4.98	504.000000
1	0.02731	0.0	7.07	0	0.469	6.421	78.900002	4.9671	2	242	17.799999	396.899994	9.14	453.600008
2	0.02729	0.0	7.07	0	0.469	7.185	61.099998	4.9671	2	242	17.799999	392.829987	4.03	728.700016
3	0.03237	0.0	2.18	0	0.458	6.998	45.799999	6.0622	3	222	18.700001	394.630005	2.94	701.400032
4	0.06905	0.0	2.18	0	0.458	7.147	54.200001	6.0622	3	222	18.700001	396.899994	5.33	760.200016

Figuur 21: **Voorbeeld:** voorwerp de huisprijs op basis van deze features

4.1 Statistische vooranalyse

4.1.1 Consistentie van de dataset

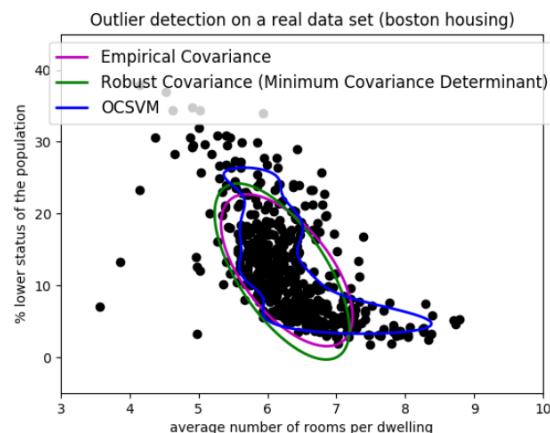
- Volledigheid van de dataset
- Inconsistenties
- Spreiding van de gegevens

Verwijderen van een onnodige kolom:

```
1 dataset.drop('CHAS', axis=1, inplace=True)
```

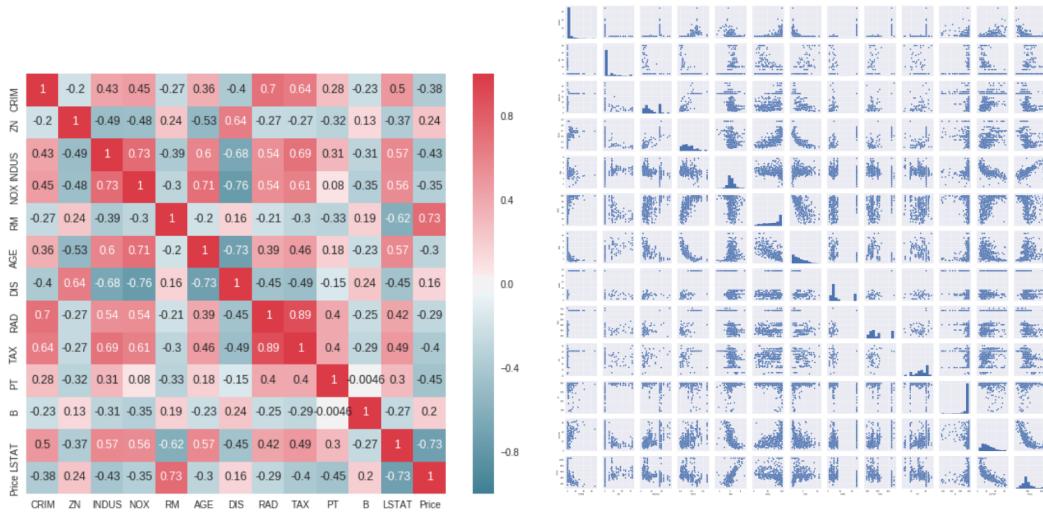
4.1.2 Uitschieters

- Vinden en verwijderen van extreme waarden/samples
- Geavanceerde technieken: zie later bij clustering



Figuur 22: Uitschieters

4.1.3 Onderlinge correlatie



Figuur 23: Heatmap en pairplot van de onderlinge correlatie tussen de features

4.2 Features en targets

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PT	B	LSTAT	Price
features X	0.00632	18.0	2.31	0.538	6.575	65.199997	4.0900	1	298	15.300000	396.899994	4.98	504.000000
targets y	0.02731	0.0	7.07	0.469	6.421	78.90002	4.9671	2	242	17.799999	396.899994	9.14	453.600008
	0.02729	0.0	7.07	0.469	7.185	61.099999	4.9671	2	242	17.799999	392.829987	4.03	728.700016
	0.03237	0.0	2.18	0.458	6.998	45.799999	6.0622	3	222	18.700001	394.630005	2.94	701.400032
	0.06905	0.0	2.18	0.458	7.147	54.200001	6.0622	3	222	18.700001	396.899994	5.33	760.200016

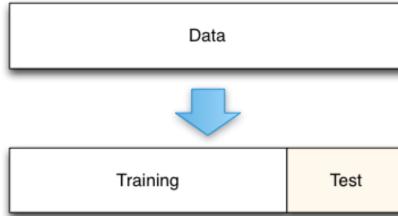
Figuur 24: Dataset opsplitsen in features en targets

```

1 # opsplitsen in features en targets. axis=1 == column
2 y = dataset['target_kolom'].values
3 X = dataset.drop('target_kolom',axis=1).values
4 # Alternatief: als bvb de laatste kolom het target is
5 features=list(dataset.columns[:dataset.columns.size-1])
6 X = dataset[features].values
7 y = dataset['Price'].values

```

4.3 Trainen van het model



Figuur 25: Dataset opsplitsen in training set en test set

- Belangrijk om eerst de data te randomiseren: te data zou gesorteerd kunnen zijn, dat willen we vermijden
- Stel dat huizenprijzen van laag naar hoog gesorteerd is, en je traaint de data op de eerste 75%, en test de laatste 25%. Resultaat: Het model zal niet getraind zijn op dure huizen.
- Ander voorbeeld: stel dat je een self-driving AI alleen traaint op de autosnelweg, en dan test in een zone-30 straat bij een school...

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
2 random_state=0)
```

4.3.1 Initialiseren en trainen van het regressiemodel

```
1 lregmodel = linear_model.LinearRegression()
2 lregmodel.fit(X_train, y_train)
```

Model:

```
1 print('coeffs: ', lregmodel.coef_)
2 print('intercept', lregmodel.intercept_)

3

4 >> coeffs: [ -3.56141289e+00, 4.05479295 e-01, 8.14080284 e-01,
5   8.96450415 e+01, -3.02997261e-01, -2.77339444e+01,
6   7.47151897 e+00, -2.92233040e-01, -1.61741146e+01,
7   -1.17962045e +01]

8

9 >> intercept: 650.652022517
```

Price = - 3.56 × CRIM + 0.41 × ZN + 0.81 × INDUS - 270.51 × NOX + 89.65 × RM - 0.30 × AGE - 27.74 × DIS + 7.47 × RAD - 0.29 × TAX - 16,17 × PT + 0.08 × B - 11.80 × LSTAT + 650.65

4.4 Testen van het model

4.4.1 Voorspellingen maken

CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PT	B	LSTAT
0.11	0	12.03	0.57	6.80	89.30	2.39	1	273	21.00	393.45	6.48

Figuur 26: Voorspel de prijs van een huis met deze features

```

1 house = np.array([0.11, 0, 12.03, 0.57, 6.80, 89.30, 2.39, 1,
2 273, 21.00, 393.45, 6.48])
3
4 house = house.reshape(1, -1)
5
6 # met reshape wordt house:
7 # house = np.array([[0.11, 0, 12.03, 0.57, 6.80, 89.30, 2.39, 1,
8 # 273, 21.00, 393.45, 6.48]])
9
10 price = lregmodel.predict(house)
11
12 print('De prijs van het huis bedraagt: ', price)
13
14 >> De prijs van het huis bedraagt: 563.68335073

```

- `reshape(1, -1)` maakt een rijvector
- Werkelijke prijs: 562.00

4.5 Performantie en scores van het model

4.5.1 Mean Absolute Error (MAE)

Definitie 4.2 (MAE) *De Mean Absolute Error (MAE) is het gemiddelde van de absolute waarden van het verschil tussen de werkelijke waarden y_i en de voorspelde waarden \hat{y}_i .*

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^n |y_i - \hat{y}_i| \quad (5)$$

```

1 from sklearn.metrics import mean_absolute_error
2
3 y_predicted = lregmodel.predict(X_test)
4 MAE = mean_absolute_error(y_test, y_predicted)
5
6 print('MAE= ', MAE)
7
8 >> MAE = 64.0090867586

```

4.5.2 Mean Squared Error (MSE)

Definitie 4.3 (MSE) *De Mean Squared Error (MSE) is het gemiddelde van de gekwadrateerde waarden van het verschil tussen de werklijke waarden y_i en de voorspelde waarden \hat{y}_i .*

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (6)$$

```

1 from sklearn.metrics import mean_squared_error
2
3 y_predicted = lregmodel.predict(X_test)
4 MSE = mean_squared_error(y_test, y_predicted)

```

```

5   print('MSE = ' MSE)
6
7 >> MSE = 7803.89332739

```

4.5.3 Determinatiecoëfficiënt

Definitie 4.4 (De determinatiecoëfficiënt R^2) De determinatiecoëfficiënt (R^2) is de variabiliteit van het model

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (7)$$

Bij perfecte voorspellingen is $R^2 = 1$

Een negatieve waarde voor R^2 betekent dat het model slechter scoort dan een horizontale lijn (= slechter dan het gemiddelde te nemen) ($y_i = \bar{y}$, \bar{y} is het gemiddelde van y)

```

1 from sklearn.metrics import r2_score
2
3 y_predicted = lregmodel.predict(X_test)
4 r2 = r2_score(y_test, y_predicted)
5
6 print('r2_score = ', r2)
7
8 # alternatieve manier voor het bepalen van de r2 score:
9 r2 = lregmodel.score(X_test, y_test)
10
11 >> r2 score = 0.754254234917

```

5 Feature engineering

Om een beter model te verkrijgen (en zo een betere R^2 score), kunnen we verschillende dingen doen:

- Meer data toevoegen
- Ander model kiezen, hyperparameter tuning
- Feature engineering: het aanmaken van extra features gebaseerd op de bestaande features

5.1 Normalisatie / Scaling

Definitie 5.1 (Normalisatie / Scaling) Normalisatie of Scaling zorgt ervoor dat de features op dezelfde schaalverdeling staan

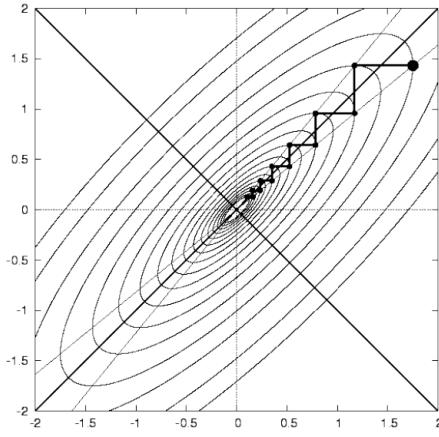
In ons voorbeeld van de huurprijs:

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PT	B	LSTAT
count	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000	441.000000
mean	2.649013	9.988662	10.759479	0.547575	6.265785	67.632200	3.851895	8.478458	389.045351	18.387302	375.393853	12.165329
std	6.273766	19.941189	6.749778	0.112896	0.685393	27.997824	2.054024	8.000859	158.293650	2.164533	49.296266	6.632866
min	0.006320	0.000000	1.250000	0.385000	3.561000	2.900000	1.169100	1.000000	188.000000	12.600000	83.449997	1.730000
25%	0.081870	0.000000	5.190000	0.449000	5.877000	45.000000	2.122200	4.000000	277.000000	17.000000	377.730011	6.920000
50%	0.217190	0.000000	8.140000	0.524000	6.172000	74.500000	3.375100	5.000000	311.000000	18.700001	392.200012	10.740000
75%	1.656600	12.500000	18.100000	0.609000	6.590000	93.599998	5.231100	8.000000	432.000000	20.200001	396.899994	15.940000
max	67.920799	80.000000	27.740000	0.871000	8.780000	100.000000	10.710300	24.000000	711.000000	22.000000	396.899994	31.990000

NOX: 0.385 \rightarrow 0.871 terwijl TAX: 188 \rightarrow 711

Figuur 27: NOX: min = 0.385, max = 0.871; TAX: min = 188, max = 711

5.1.1 Voordelen

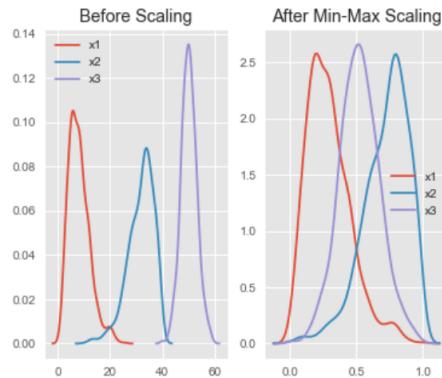


Figuur 28: Gradient Descent convergeert minder snel als features op een verschillende schaalgrootte staan. Normalisatie zorgt er dus voor dat het model sneller zal trainen.

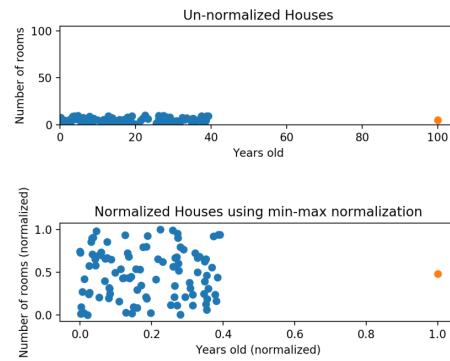
5.1.2 MIN-MAX-scaling

$$x_{s_i} = \frac{x_i - \text{Min}(x)}{\text{Max}(x) - \text{Min}(x)} \quad (8)$$

- Schaalt alle features tussen 0 en 1
- Werkt goed bij niet-Gaussiaanse distributies en bij kleine variantie
- De scheefheid (skew) blijft bewaard
- Gevoelig voor uitschieters



Figuur 29: Voor en na scaling



Figuur 30: Bij het voorbeeld van de huizenprijzen

```

1  from sklearn.preprocessing import MinMaxScaler
2
3  scaler = MinMaxScaler().fit(X_train)
4  X_train = scaler.transform(X_train)
5  X_test = scaler.transform(X_test)
6
7  # alternatief
8  scaler = MinMaxScaler()
9  X_train = scaler.fit_transform(X_train)
10 X_test = scaler.transform(X_test)

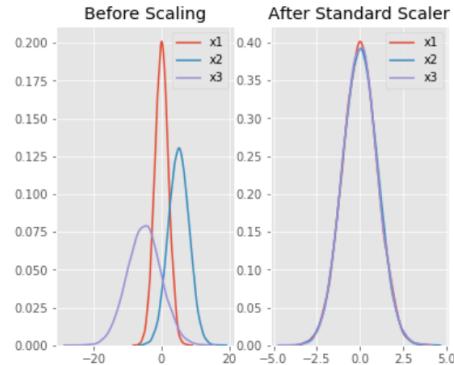
```

5.1.3 Standard scaling (normalisatie)

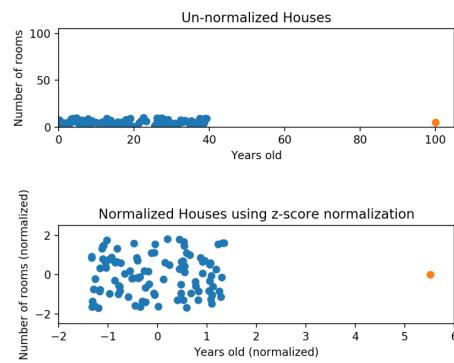
$$x_{s_i} = \frac{x_i - \text{mean}(x)}{\text{stddev}(x)} \quad (9)$$

- Geschaalde features;
 - Gemiddelde = 0
 - Standaardafwijking = 1

- Geschaalde features schommelen rond 0 (soms nodig bij deep learning)
- Vervormt geen relatieve afstanden tussen de feature waarden
- Kan beter overweg met uitschieters
- Garandeert geen genormaliseerde data op exact dezelfde schaal



Figuur 31: Voor en na standard scaling



Figuur 32: Bij het voorbeeld van de huizenprijzen

```

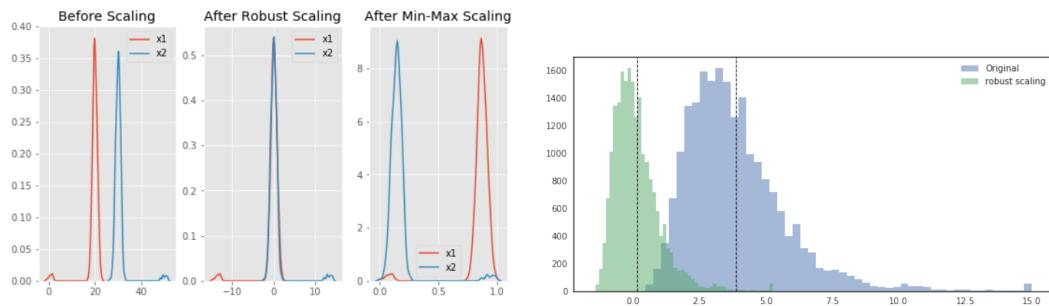
1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler().fit(X_train)
4 X_train = scaler.transform(X_train)
5 X_test = scaler.transform(X_test)
6
7 # alternatief
8 scaler = StandardScaler()
9 X_train = scaler.fit_transform(X_train)
10 X_test = scaler.transform(X_test)

```

5.1.4 Robust scaling

$$x_{s_i} = \frac{x_i - Q_2(x)}{Q_3(x) - Q_1(x)} \quad (10)$$

- Lijkt op MIN-MAX scaler maar gebruikt de interkwartielafstand ipv range
- Houdt geen rekening met uitschieters
- Gebruikt minder data bij het bepalen van de schaal
- Range van de genormaliseerde data is groter dan bij MIN-MAX scaling
- Garandeert geen genormaliseerde data op exact dezelfde schaal



Figuur 33: Voor en na robust scaling

```

1 from sklearn.preprocessing import RobustScaler
2
3 scaler = RobustScaler().fit(X_train)
4 X_train = scaler.transform(X_train)
5 X_test = scaler.transform(X_test)
6
7 # alternatief
8 scaler = RobustScaler()
9 X_train = scaler.fit_transform(X_train)
10 X_test = scaler.transform(X_test)

```

5.2 Feature expansion

5.2.1 Nieuwe features

Bedenken van nieuwe features

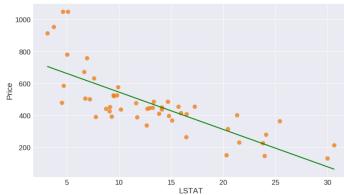
Voorbeelden:

- Uit de lengte en breedte van een huis de oppervlakte halen als nieuwe feature.
- Uit een start en eindpunt de afstand halen als nieuwe feature.
- Uit een datum afleiden welke dag van de week het is.
- Veranderingen in de features.
- Nieuwe opgemeten parameters.

5.2.2 Hogere-orde features

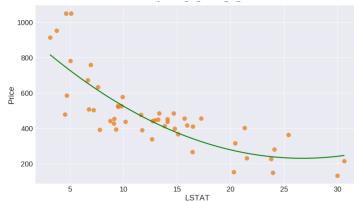
= Het verband tussen features en de target(s) is niet altijd lineair.

Voorbeeld: samenhang tussen LSTAT (x_1) en de huizenprijs (P)



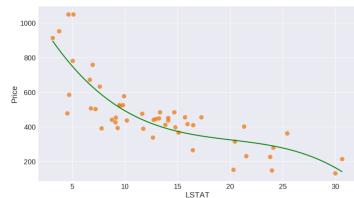
Figuur 34: $P = \theta_0 + \theta_1 x_1$

Toevoegen van een extra hogere-orde feature $x_2 = x_1^2$:

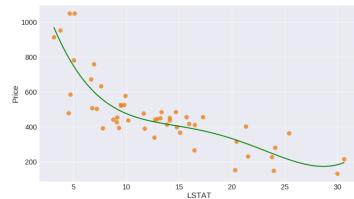


Figuur 35: $P = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

Toevoegen van een extra hogere-orde features $x_3 = x_1^3$ en $x_4 = x_1^4$:



Figuur 36: $P = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$



Figuur 37: $P = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

```

1 # toevoegen van een extra feature: LSTAT^2 LSTAT^3
2 dataset.insert(dataset.columns.size - 1, 'LSTAT^2', dataset.LSTAT**2)
3 dataset.insert(dataset.columns.size - 1, 'LSTAT^3', dataset.LSTAT**3)

```

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PT	B	LSTAT	LSTAT^2	LSTAT^3	Price
0	0.00632	18.0	2.31	0.538	6.575	65.199997	4.0900	1	296	15.300000	396.899994	4.98	24.800400	123.505993	504.000000
1	0.02731	0.0	7.07	0.469	6.421	78.900002	4.9671	2	242	17.799999	396.899994	9.14	83.539606	763.552030	453.600008
2	0.02729	0.0	7.07	0.469	7.185	61.099998	4.9671	2	242	17.799999	392.829987	4.03	16.240902	65.450837	728.700016
3	0.03237	0.0	2.18	0.458	6.998	45.799999	6.0622	3	222	18.700001	394.630005	2.94	8.643600	25.412185	701.400032
4	0.06905	0.0	2.18	0.458	7.147	54.200001	6.0622	3	222	18.700001	396.899994	5.33	28.408899	151.419431	760.200016

Figuur 38: Resultaat toevoegen hogere-orde features

- Nu model met extra features trainen en nadien testen op de test set.
- Opgepast:** bij de test set moet je ook dezelfde features toevoegen.

5.3 One-hot encoding

- = Omzetten van categorische variabelen naar meerdere aparte features
- categorische variabelen = variabelen zonder echte waarden, de waarden stellen een categorie voor (niet altijd een nummer)
- ⇒ voor elke categorie een nieuwe kolom
- ‘Dummy Variable Trap’
 - = als een rij maar tot 1 categorie kan behoren (zie onderstaand voorbeeld, een appel is geen kip), dan zou je in principe 1 kolom kunnen schrappen en dan kan je toch dezelfde informatie krijgen.
 - $x_1 + x_2 + x_3 = 1 \Leftrightarrow x_1 = 1 - x_2 - x_3$
 - In de praktijk laat men dit gewoon staan.
 - (niet te kennen op examen)

Label Encoding			One Hot Encoding			
Food Name	Categorical #	Calories	Apple	Chicken	Broccoli	Calories
Apple	1	95	1	0	0	95
Chicken	2	231	0	1	0	231
Broccoli	3	50	0	0	1	50

Figuur 39: Voorbeeld One-hot encoding

```

1 # voeg de categorieën toe als kolommen
2 dataset = pd.concat(
3     [dataset, pd.get_dummies(dataset['food_name'], prefix='food')], 
4     axis=1)
5 # verwijder de food_name kolom
6 dataset.drop(['food_name'], axis=1, inplace=True)
7 # toon de eerste 5 rijen:
8 dataset.head()

```

	food_name	Calories		Calories	food_Apple	food_Broccoli	food_Chicken	food_Chocolat
0	Apple	95	0	95	1	0	0	0
1	Chicken	231	1	231	0	0	1	0
2	Broccoli	50	2	50	0	1	0	0
3	Chocolat	549	3	549	0	0	0	1

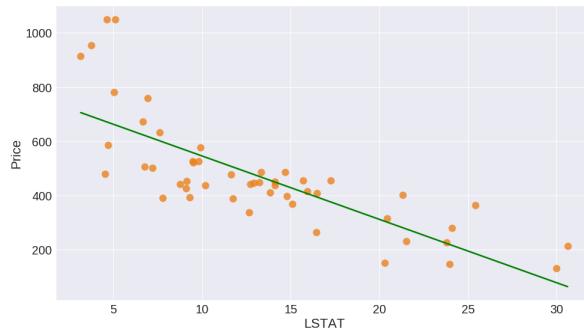
Figuur 40: Resultaat van dataset.head(), voor en na one-hot encoding

6 Underfitting & overfitting

6.1 Underfitting

Definitie 6.1 (Underfitting) *Underfitting treedt op wanneer een model de training data niet kan modeleren en ook niet kan generaliseren op nieuwe data.*

- *Het model is te 'simpel'*
- *Model met hoge bias*
- ⇒ *de score met de training data en de score met de test data zijn beiden laag.*

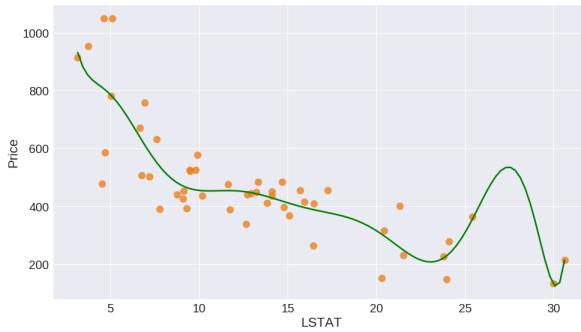


Figuur 41: Het model werkt voor sommige variabelen maar voor velen ook niet

6.2 Overfitting

Definitie 6.2 (Overfitting) *Overfitting treedt op wanneer een model de training data te goed modelleert en niet kan generaliseren op nieuwe data.*

- *Het model is te 'complex'*
- *De ruis van willekeurige fluctuaties in data worden opgepikt*
- *Model met een hoge variance*
- ⇒ *de score met de training data is groter dan de score met de test data*

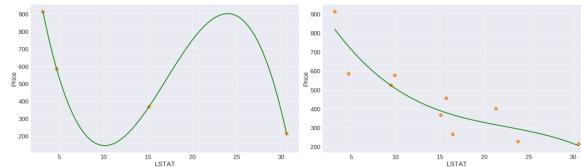


Figuur 42: Model met order ≈ 15

6.2.1 Impact van de grootte van de dataset

Afhankelijkheid van de grootte van de dataset (aantal observaties m):

- Bij weinig observaties: snel overfitting bij complexer model
- Bij veel observaties: minder snel overfitting bij complexer model



Figuur 43: Voorbeeld met 3de orde polynoom: links = overfitting, rechts niet

6.3 Regularisatie (regularisation)

Definitie 6.3 (Regularisatie) Methode om de mate van bias van een hypothese te regelen en een goed evenwicht te vinden tussen underfitting en overfitting.

We gebruikten tot nu een kostenfunctie die we moesten minimaliseren om het model te trainen. Daar voegen we nu $R(\theta)$ aan toe:

$$J(\theta) = \frac{1}{2 \cdot m} \sum_{i=1}^m (h_\theta(x_i) - y_i)^2 + R(\theta) \quad (11)$$

Definitie 6.4 ($R(\theta)$) $R(\theta)$ is de regularisatie-term.

Dit is een extra kostenterm die het gebruik van hogere orde features afstraft tenzij ze de globale kostenfunctie doen dalen. $R(\theta)$ is gelijk aan $\lambda \theta \theta^\top$.

6.3.1 Voorbeeld regularisatie

$$J(\theta) = \frac{1}{2 \cdot m} \sum_{i=1}^m (h_\theta(x_i) - y_i)^2 + \lambda \theta \theta^\top$$

- $\theta = \{\theta_1, \dots, \theta_n\}$
 - intercept θ_0 wordt meestal niet geregulariseerd
 - Als de θ 's vergroten, zal de $R(\theta)$ ook vergroten

- λ is een tuning parameter (hyper parameter), we moeten die zelf vinden
 - $\lambda = 0 \Rightarrow$ geen regularisatie
 - $\lambda = \inf \Rightarrow \theta = 0$
 - λ tussenin regelt de mate van regularisatie.

De tuningparameter λ regelt de complexiteit van de hypothese:

- Kleine waarde voor λ : lage bias, hoge variantie (overfitting)
- Grote waarde voor λ : hoge bias, lage variantie (underfitting)

Afhankelijk van hoe R gedefinieerd wordt is er een andere benaming voor de regularisatie:

- Ridge regression (L2 regularisatie)
- Lasso regression (L1 regularisatie)

Het is ook mogelijk om meerdere regularisatietermen aan de kostenfunctie toe te voegen, bijvoorbeeld 1 gebaseerd op L2 en 1 op L1.

6.3.2 Regularisatie met L2 norm

Bij L2 regularisatie wordt de regularisatieterm berekend met behulp van de som van de kwadraten van de θ 's

$$J_{L2} = J + \lambda_2 \sum_{j=1}^m \theta_j^2$$

$$J_{L2} = \sum_{i=1}^n (\text{target}_i - \text{output}_i) + \lambda_2 \sum_{j=1}^m \theta_j^2$$

6.3.3 Regularisatie met L1 norm

Bij L2 regularisatie wordt de regularisatieterm berekend met behulp van de som van de absolute waarden van de θ 's

$$J_{L1} = J + \lambda_1 \sum_{j=1}^m |\theta_j|$$

$$J_{L1} = \sum_{i=1}^n (\text{target}_i - \text{output}_i) + \lambda_1 \sum_{j=1}^m |\theta_j|$$

6.3.4 Voorbeeld regularisatie op huizenprijzen

Via Ridge of Lasso regressie met regularisatieparameter α

- Hoe groter α , hoe sterker de regularisatie en dus hoe simpler het model
- Hoe kleiner α , hoe zwakker de regularisatie en dus hoe complexer het model

```

1 regmodel = Ridge(alpha=0.14, tol=0.0001, fit_intercept=True)
2 regmodel.fit(X_train, y_train)
3 regmodel.score(X_test, y_test)
>> 0.79834480089914472
5
6 lregmodel = Lasso(alpha=0.5, tol=0.0001, fit_intercept=True)
7 lregmodel.fit(X_train, y_train)
8 lregmodel.score(X_test, y_test)
>> 0.8437113338085345

```

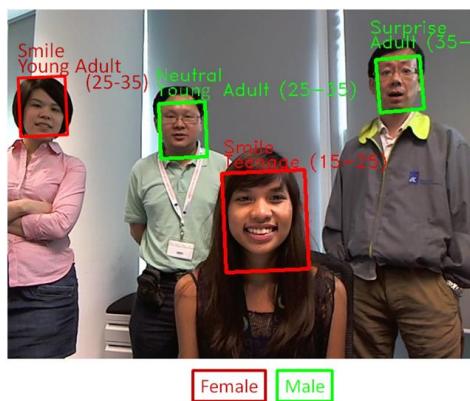
7 Classificatie

7.1 Wat is classificatie?

Definitie 7.1 (Classificatie) *Classificatie is een supervised learning techniek waarbij een getraind model niet-geziene inputs toewijst aan één of meerdere gelabelde categorieën (classes)*

7.1.1 Voorbeelden

- Gezichtsherkenning
- Nummerplaatherkenning
- Spam detectie
- Medische diagnoses
- Voorspelling of een klant op een advertentie zal klikken
- Kwaliteitscontrole
- ...

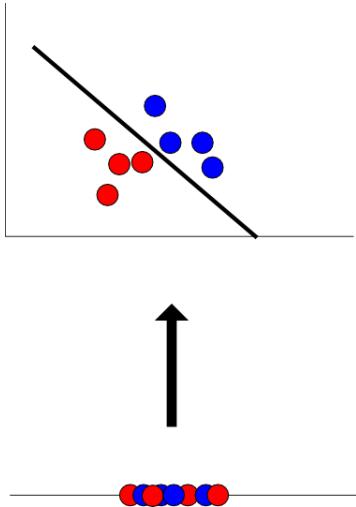


Figuur 44: Geslachtsherkenning + leeftijdsclassificatie

7.2 Types van classifiers

7.2.1 Binary (binomial) classifier

= verdeel de samples in **twee verschillende klassen**



Figuur 45: Voorbeeld: bepaal of een kanker goedaardig of kwaadaardig is

7.2.2 Multiclass classifier

Definitie 7.2 Een multiclass classifier zal samples in 3 of meer verschillende klasses verdelen



Figuur 46: Voorbeelden: gezichtsherkenning, sentiment analyse

7.2.3 Multilabel classifier

Definitie 7.3 Een multilabel classifier zal meerdere labels aan een sample toewijzen. Een sample kan dus tot meerdere klasses behoren



Figuur 47: Image content analysis, een film kan tot meerdere genres behoren

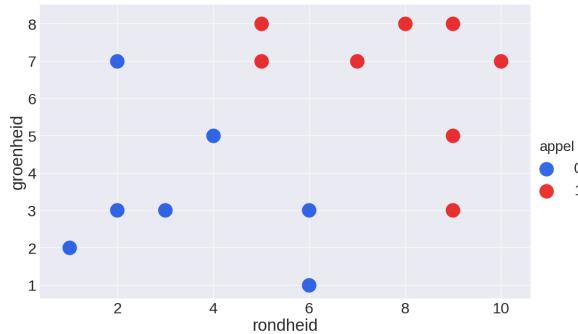
7.3 Voorbeeld van een classificatie: appel herkennen

	rondheid	groenheid	appel
0	9	8	1
1	10	7	1
2	2	3	0
3	1	2	0
4	5	8	1
5	7	7	1
6	6	3	0
7	3	3	0
8	9	5	1
9	9	3	1
10	4	5	0
11	6	1	0
12	5	7	1
13	8	8	1
14	2	7	0

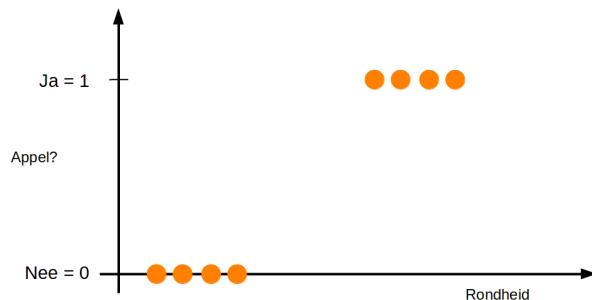
Figuur 48: Voorspel of een stuk fruit een appel is op basis van vorm en kleur

- features: rondheid en groenheid
- target: appel: ja/nee
- trainingset: 15 training samples

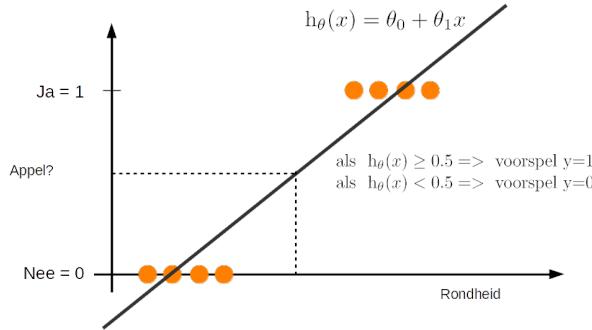
Bij classificatie is de output/target een (**discrete**) variabele/klasse



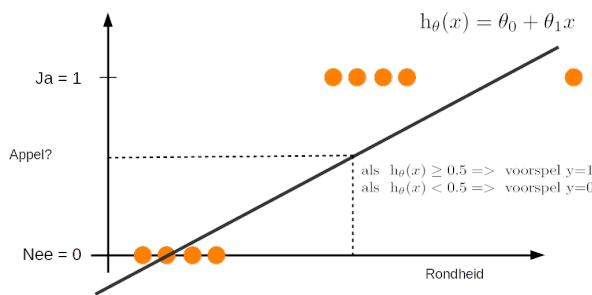
7.3.1 Waarom lineaire regressie geen goede optie is



Figuur 49: Voorbeeld: rondheid bepaalt of het een appel is of niet



Figuur 50: We proberen te classificeren met behulp van lineaire regressie



Figuur 51: Toevoegen van een nieuw punt (rechts)

- Stel dat we een nieuw punt toevoegen dan kan het zijn dat onze regressie veel minder nauwkeurig is
- Een betere manier: **logistische regressie**

8 Logistische regressie

Definitie 8.1 Logistische regressie wordt gebruikt om de kans te bepalen of een bepaalde sample tot één van twee klassen behoort. De uitkomst is dus een probabiliteit.

De naam is een beetje misleidend: Logistic Regression is eigenlijk een classificatietechniek. De reden voor deze naam is omdat regressie gebruikt wordt om de classificatie te berekenen.

8.1 Het model

Omdat de uitkomst een probabiliteit is, zoeken we een functie h_θ zodat het model $h_\theta(x)$ voldoet aan:

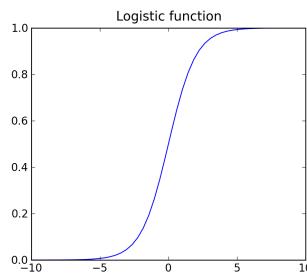
$$0 \leq h_\theta(x) \leq 1$$

- $h_\theta(x)$ = de geschatte kans dat $y = 1$ bij input x
- Voorbeeld: $h_\theta(x) = 0.80$
 - Het model is voor 80% overtuigd dat het om een appel gaat.
 - Als je dus als threshold 50% gebruikt, classificeert het model het object als een appel

8.1.1 De logistische functie

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \quad (12)$$

- Met e = het getal van Euler = ≈ 2.718
- Dit is een sigmoïdefunctie: de basisform voor zo'n functie is $\frac{1}{1+e^{-z}}$
- Bij onze logistische functie is z dus gelijk aan $\theta^T x$
 - Dit is gelijk aan de uitkomst van een lineaire regressiefunctie
 - Met x = de samplewaardes
 - Met θ = de gewichten die bij die waardes horen



Figuur 52: De logistische functie is duidelijk een sigmoïdefunctie of S-functie

- $y = 1$ als $h_\theta(x) \geq 0.5 \Rightarrow \theta^T x \geq 0$
- $y = 0$ als $h_\theta(x) < 0.5 \Rightarrow \theta^T x < 0$

8.1.2 Interpretatie via voorbeeld appels

Het model is van de vorm:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

met x_1 = rondheid en x_2 = groenheid

Veronderstel na training: $\theta_0 = -40, \theta_1 = 4, \theta_2 = 4$

- Voorspel $y = 1$ als $-40 + 4x_1 + 4x_2 \geq 0$
- Voorspel $y = 0$ als $-40 + 4x_1 + 4x_2 < 0$

Gegeven een rondheid van 8 en een groenheid van 6:

$$\begin{aligned} -40 + 4 \cdot 8 + 4 \cdot 6 &= 16 \Rightarrow \text{Appel} \\ h_\theta(x) &= \frac{1}{1+e^{-16}} = 0.999999887 \end{aligned}$$

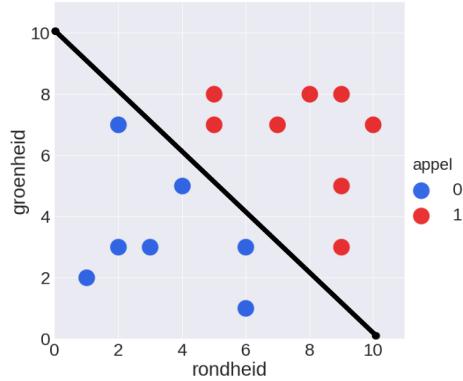
Gegeven een rondheid van 5 en een groenheid van 4.5:

$$\begin{aligned} -40 + 4 \cdot 5 + 4 \cdot 4.5 &= -2 \Rightarrow \text{Geen appel} \\ h_\theta(x) &= \frac{1}{1+e^{-2}} = 0.12 \end{aligned}$$

- Het model is maar voor 12% zeker dat het om een appel gaat
- \Rightarrow Met 88% zekerheid gaat het volgens het model niet om een appel

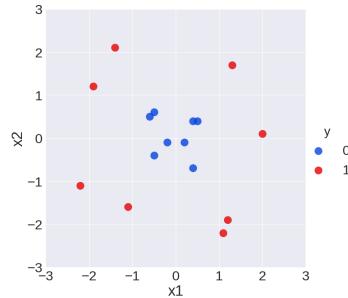
8.1.3 Grafische interpretatie via voorbeeld appels

- Op de scheidingslijn: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$
- In het voorbeeld: $-40 + 4x_1 + 4x_2 = 0$



Figuur 53

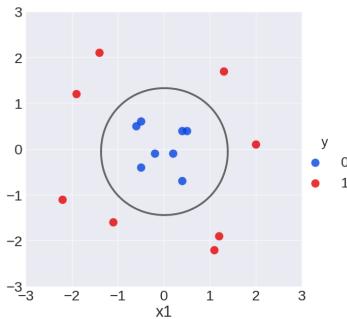
8.1.4 Wat als het model niet lineair is?



Figuur 54: Het model is niet lineair

Extra features: $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

- Veronderstel: $\theta_0 = -2, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$
- Voorspel $y = 1$ als $-2 + x_1^2 + x_2^2 \geq 0$



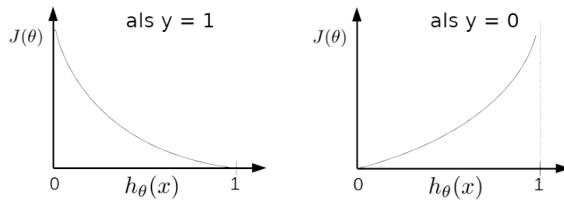
Figuur 55: $x_1^2 + x_2^2 \geq 2$ (vergelijking van een cirkel met straal $\sqrt{2}$)

8.2 De kostenfunctie

De kostenfunctie wordt:

$$J(\theta) = \begin{cases} -\ln(h_\theta(x)) & \text{als } y = 1 \\ -\ln(1 - h_\theta(x)) & \text{als } y = 0 \end{cases} \quad (13)$$

$$J(\theta) = -\frac{1}{m} \cdot \left[\sum_{i=1}^m y_i \ln(h_\theta(x_i)) + (1 - y_i) \cdot \ln(1 - h_\theta(x_i)) \right] \quad (14)$$



Figuur 56: Zoek de waarden voor θ die de kostenfunctie $J(\theta)$ minimaliseert via Gradient Descent (GDS)

8.3 Voorbeeld logistic regression met Sklearn (lineair model)

8.3.1 Preprocessing van de data

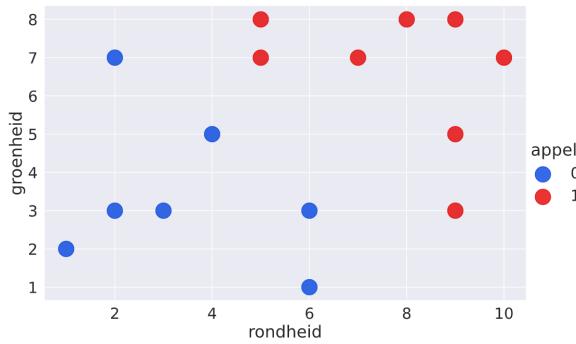
Analoog aan preprocessing bij lineaire regressie:

- Data inlezen
- Check op inconsistenties
- Check uitschieters
- Plot de data
- Splits op in features en targets
- Verdeel in een training en test set

```

1 # importeer de dataset
2 dataset = pd.read_csv('appels.csv')
3 # definieer de features
4 features = list(dataset.columns[:dataset.columns.size-1])
5 X = dataset[features].values
6 y = dataset['appel'].values
7 # lettergrootte voor de axis labels
8 sns.set(font_scale=2)
9 # definieer kleuren
10 colors = ['blue', 'red', 'greyish', 'faded_green', 'dusty_purple']
11 # plot
12 sns.lmplot(x='rondheid', y='groenheid', data=dataset,
13     fit_reg=False, hue='appel', palette=sns.xkcd_palette(colors),
14     scatter_kws={'s': 500}, size=7, aspect=1.5)

```



Figuur 57: Plot van de dataset

8.3.2 Trainen van het logistic regression model

```

1 # train een logistic regression classifier
2 logreg = linear_model.LogisticRegression(C=1e5)
3 # C = inverse of regularisation strength
4 # must be a positive float
5 # like in support vector machines: smaller values => stronger regularisation
6 logreg.fit(X, y)

8 print('coefficienten: ', logreg.coef_)
9 print('intercept: ', logreg.intercept_)

```

- Coefficienten: $\theta_1 = 4.287$ en $\theta_2 = 4.062$
- Intercept: $\theta_0 = -43.941$

8.3.3 Classificeren van een nieuwe sample

```

1 # voorspel de klasse met rondheid = 8 en groenheid = 6
2 print(logreg.predict(np.array([8,6]).reshape(1, -1)))
3
4 kans = logreg.predict_proba(np.array([8,6]).reshape(1, -1))

```

```

5 print("Kans op appel/geen appel = ", kans)
6
7 # voorspel de klasse met rondheid = 4 en groenheid = 4
8 print(logreg.predict(np.array([4,4]).reshape(1, -1)))
9
10 kans = logreg.predict_proba(np.array([4,4]).reshape(1, -1))
11 print("Kans op appel/geen appel = ", kans)

```

Output:

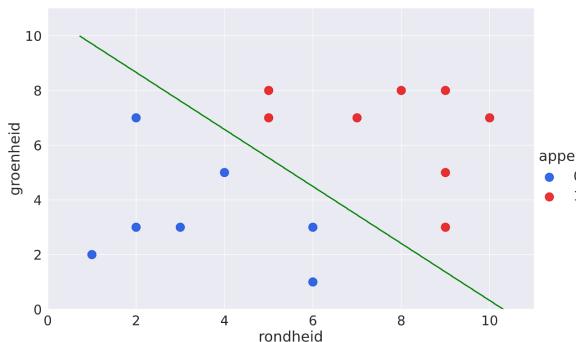
- kans op een appel/geen appel: 3.99395302e-07 | 9.99999601e-01
- kans op een appel/geen appel: 9.99973583e-01 | 2.64168196e-05

8.3.4 Visualiseer de decision boundary

```

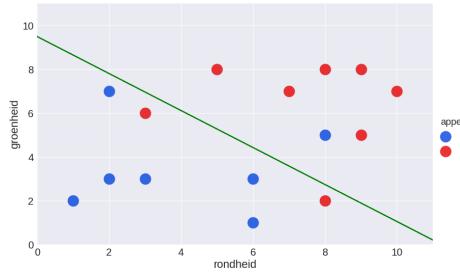
1 h = 0.01
2 rond_min = X[:,0].min()-2
3 rond_max = X[:,0].max()+2
4 groen_min = X[:,1].min()-2
5 groen_max = X[:,1].max()+2
6 xx, yy = np.meshgrid(np.arange(rond_min, rond_max, h),np.arange(groen_min, groen_max, h))
7 Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
8 Z = Z.reshape(xx.shape)
9 # font settings
10 sns.set(font_scale = 2)
11 # colors
12 colors = ["blue", "red", "greyish", "faded green", "dusty purple"]
13 sns.lmplot(x='rondheid',y='groenheid',data=dataset,
14     fit_reg=False,hue='appel',palette=sns.xkcd_palette(colors),
15     scatter_kws={'s':200}, height=8, aspect=1.5)
16 plt.ylim(0, 11)
17 plt.xlim(0, 11)
18 plt.contour(xx, yy, Z, colors='green')

```



Figuur 58: Visualisatie van de decision boundary

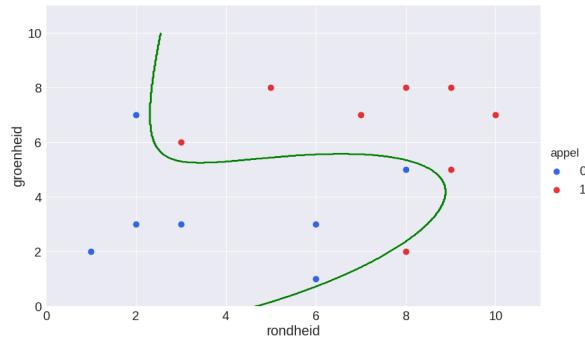
8.4 Voorbeeld logistic regression met Sklearn - niet-lineair



Figuur 59: Niet-lineair scheidbare dataset

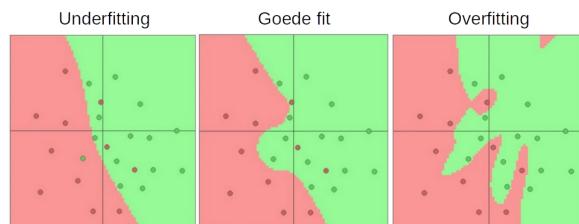
8.4.1 Feature engineering: automatisch toevoegen van hogere orde features

```
1 # Aanmaken van de hogere orde features
2 graad = 3
3
4 poly = PolynomialFeatures(graad)
5 Xp = poly.fit_transform(X)
6
7 # Train model op hogere orde features en visualiseer de decision boundary
8 logreg_poly = linear_model.LogisticRegression(C=1)
9 logreg_poly.fit(Xp, y)
```



Figuur 60: Resultaat: overfitting

8.4.2 Oplossing: regularisatie



Figuur 61: Regelen tussen underfitting en overfitting via regularisatie

- Via regularisatie een goed evenwicht zoeken tussen underfitting en overfitting
- In Scikit Learn `linear_model.LogisticRegression`:
 - C = inverse regularisatie sterkte
 - kleine waarden voor C \Rightarrow sterke regularisatie (overfitting)
 - grote waarden voor C \Rightarrow zwakke regularisatie (underfitting)

```

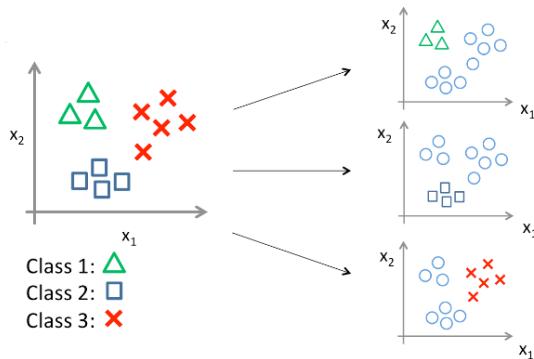
1 logreg = linear_model.LogisticRegression(C=100)
2 logreg.fit(Xf, y)

```

8.5 Multi-class classification

Data kan tot meerdere klassen behoren. We kunnen met zo'n data dus geen binaire classificatie doen.

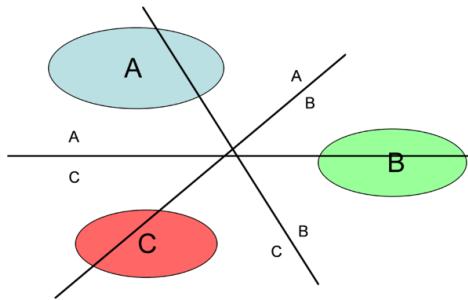
8.5.1 One-vs-All



Figuur 62: One-vs-All: we maken 3 binaire classifiers

- 'One-vs-Rest'
- Je classificeert 1 klasse tegen de rest
- Je hebt dus 1 classifier per klasse
- Totaal aantal classifiers: N (met N = aantal klassen)
- Gevoeliger voor niet-gebalanceerde data (als er in een bepaalde klasse weinig datasamples zijn)

8.5.2 One-vs-One



Figuur 63: One-vs-One

- Je hebt $\frac{N \cdot (N-1)}{2}$ classifiers (met N = aantal klasses) \Rightarrow rekenintensiever dan One-vs-All
- Minder gevoelig voor niet-gebalanceerde data

9 Evaluatie van een classifier

We moeten kunnen bepalen hoe goed een classifier is, om te weten hoe betrouwbaar de voorspelling is en om de classifier te vergelijken met andere classifiers. Bij regressie kunnen we gebruikmaken van onder andere MAE, MSE en de determinatiecoëfficiënt.

Bij classificatie zijn die metrics niet interpreerbaar. Daarom maken we gebruik van andere metrics die allemaal berekend worden met behulp van een waarheidstabellen:

	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Figuur 64: Confusion matrix

Stel: we hebben een binaire classifier die klasse 0 of 1 als output heeft. We vergelijken de verwachte waarden uit de classifier met de werkelijke waarden uit de testset.

- True Positive = de verwachte waarde en werkelijke waarde zijn beide 1
- False Positive = de verwachte waarde is 1, terwijl de werkelijke waarde 0 is
- False Negative = de verwachte waarde is 0, terwijl de werkelijke waarde 1 is
- True Negative = de verwachte waarde en werkelijke waarde zijn beide 0

We tellen het aantal TP, FP, FN, TN om te gebruiken in de volgende formules. Deze formules kunnen we gebruiken om een classifier te evalueren.

9.1 Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \quad (15)$$

Probleem: stel dat $TN = 990$, $FN = 10$, $TP = 0$, $FP = 0$

- Dan heb je een accuracy van 99%, terwijl je eigenlijk geen enkele positieve predictie hebt
- Accuracy is dus niet altijd veelzeggend \Rightarrow nood aan andere termen

9.2 True Positive Rate (TPR)

= Recall = Sensitivity = Hit rate

$$\text{Recall} = \frac{TP}{TP + FN} \quad (16)$$

- = Het aantal juiste positieve voorspellingen op het totaal aantal positieve waarden in werkelijkheid
- Wordt vaak in combinatie gebruikt met Precision (PPV)

9.3 Positive Predictive Value (PPV)

= Precision

$$\text{Precision} = \frac{TP}{TP + FP} \quad (17)$$

- = Het aantal juiste positieve voorspellingen op het totaal aantal positieve voorspellingen

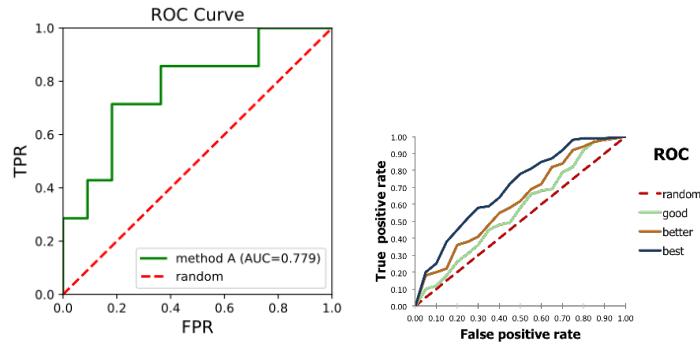
9.4 F1 Score

$$\text{F1 score} = \frac{2 \cdot (\text{Recall} \cdot \text{Precision})}{(\text{Recall} + \text{Precision})} \quad (18)$$

- Is een harmonisch gewogen gemiddelde van de recall en de precision
- Is de F_β -score met $\beta = 1$: $F_\beta = \frac{(1+\beta^2) * \text{Precision} * \text{Recall}}{(\beta^2 * \text{Precision}) + \text{Recall}}$

9.5 Receiver Operating Characteristic (ROC)

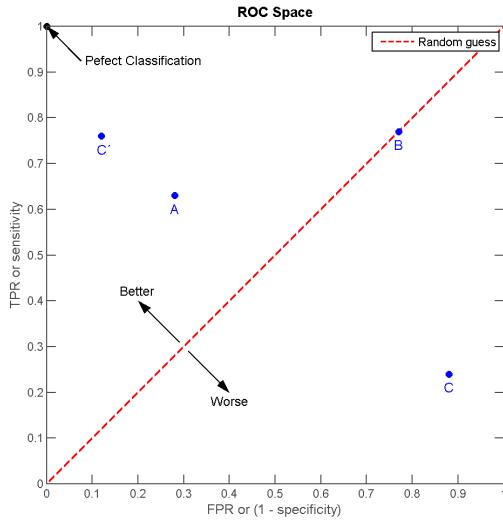
- Wordt gebruikt bij binaire classifiers (2 klassen)
- Wordt gebruikt om beter modellen te selecteren en minder goede te verwerpen



Figuur 65: ROC Curve (x-as = FPR, y-as = TPR)

We gebruiken voor de assen de TPR en FPR

- True Positive Rate = Sensitivity = Recall (zie hierboven)
- False Positive Rate = $1 - \text{Specificity} = \frac{FP}{FP+TN}$
- Specificity = $\frac{TN}{FP+TN}$



Figuur 66: Classifier C' is een betere classifier dan classifier A

- Bij classifier C' hebben we het omgekeerde van C genomen
- We noemen linksboven 'ROC heaven' en rechtsbeneden 'ROC Hell'.
- De rode stippelijn is een classifier die gewoon gokt.
- B is een classifier die even goed is als een gewone gok.

9.5.1 ROC curve en AUC (Area Under ROC Curve)

ROC bij verschillende threshold settings:

Example nr.	y_true	P(y=1 x)		Predicted (y=1)	Predicted (y=0)
1	1	0.93		4	1
2	0	0.55	Actual (y=1)	4	1
3	0	0.30	Actual (y=0)	3	2
4	0	0.53			
5	1	0.81			
6	1	0.69			
7	0	0.42			
8	1	0.28			
9	1	0.96			
10	0	0.51			

Figuur 67: $p(y = 1|x) - \text{threshold} = 0.5$

- Testset met 10 samples

$$TPR = \frac{TP}{TP+FN} = \frac{4}{4+1} = 0.8$$

$$FPR = \frac{FP}{FP+TN} = \frac{3}{3+2} = 0.6$$

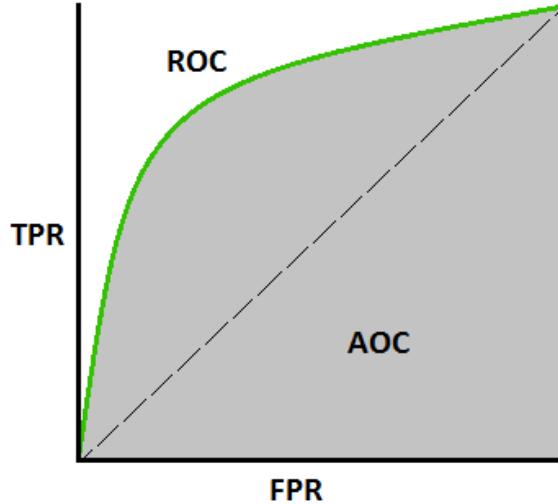
Example nr.	y_true	P(y=1 x)		Predicted (y=1)	Predicted (y=0)
1	1	0.93		4	1
2	0	0.55	Actual (y=1)	4	1
3	0	0.30	Actual (y=0)	0	5
4	0	0.53			
5	1	0.81			
6	1	0.69			
7	0	0.42			
8	1	0.28			
9	1	0.96			
10	0	0.51			

Figuur 68: $p(y = 1|x) - \text{threshold} = 0.6$

- Met $p(y = 1|x) - \text{threshold} = 0.6$ krijgen we dezelfde TPR, maar een betere FPR

$$TPR = \frac{TP}{TP+FN} = \frac{4}{4+1} = 0.8$$

$$FPR = \frac{FP}{FP+TN} = \frac{0}{0+5} = 0$$



Figuur 69: AUC of AuROC = Area under ROC

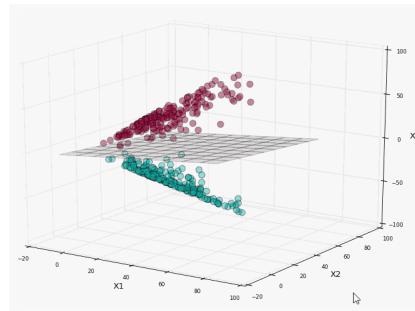
- Stel dat de threshold 0 is:

- Dan is TPR = 100%
- Maar FPR is ook 100%
- Stel dat de threshold dichter bij 1 komt (bv: 0.95):
 - Dan is TPR dichter bij 0%
 - En FPR ook dichter bij 0%
- Je zoekt de beste ROC \Rightarrow ROC met grootste AOC

10 Support Vector Machines (SVM)

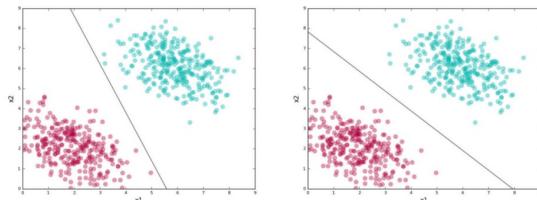
10.1 Wat is een SVM?

Definitie 10.1 (SVM) Een SVM is een supervised ML algoritme dat zowel voor classificatie als regressie gebruikt kan worden. Classificatie gebeurt door het vinden van een hyperplane die een optimale scheiding maakt tussen twee verschillende klassen.



Figuur 70: Grafiek: hyper-plane die een optimale scheiding maakt

10.1.1 Welke classifier zou je verkiezen?

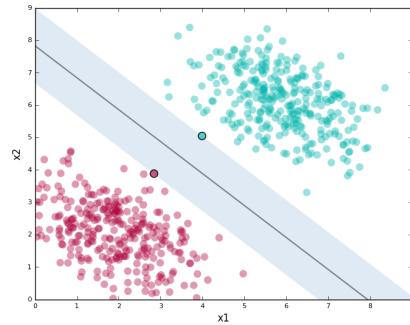


- Beide scheiden de data perfect
- Het gaat er niet om welke er best presteert op de training data, maar wel op de test data
- De rechtse classifier is meer robuust: de meeste punten liggen verder van de lijn in vergelijking met de linkse

10.2 Hoe een SVM classificeert

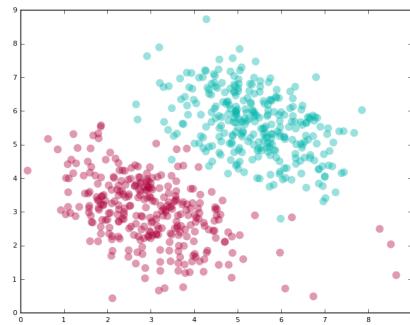
- Zoek scheidingslijn die de trainingset zo goed mogelijk scheiden

- Kies de scheidingslijn die de grootste afstand (margin) heeft tot de punten die er het dichtst bij gelegen zijn
- De dichtsbij gelegen punten noemen we de **support vectors**
- SVM = Large margin classifier



Figuur 71: Scheidingslijn kiezen

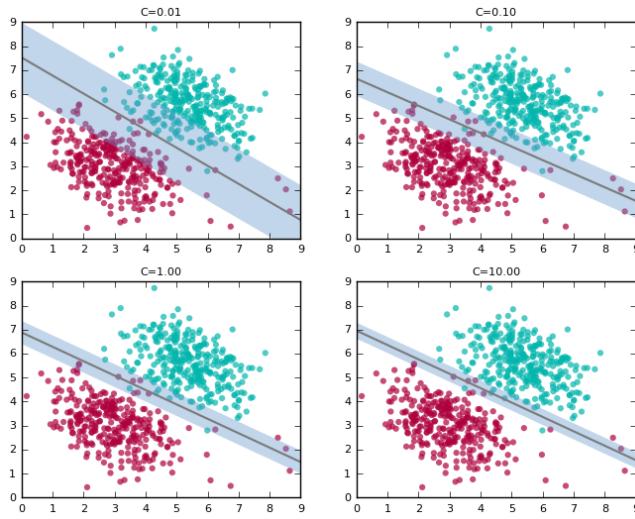
10.2.1 Wat als een perfecte lineaire scheiding niet mogelijk is?



Figuur 72

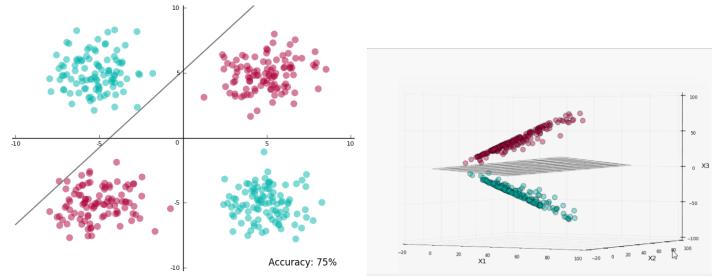
Antwoord: werken met een regularisatieparameter C

- Afweging tussen correcte classificatie op de training set en een grote marge tussen de klassen (large margin)
- Grote C -waarde: constraints zijn moeilijk te negeren \Rightarrow smalle marge
- Kleine C -waarde: constraints kunnen makkelijker genegeerd worden \Rightarrow brede marge



Figuur 73: Bij een grotere C-waarde krijgen we een smallere marge \Rightarrow op zoek gaan naar de optimale C

10.2.2 Wat bij niet-lineair scheidbare gegevens

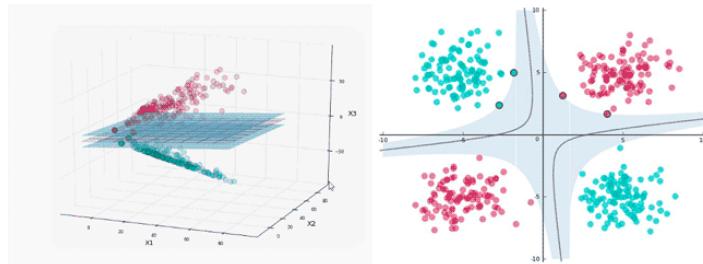


Figuur 74: Niet-lineair scheidbare gegevens

Oplossing: transformeer de data naar een hogere dimensie gevuld door lineaire scheiding

Bv:

- $X_1 = x_1^2$
- $X_2 = x_2^2$
- $X_3 = \sqrt{2} \cdot x_1 \cdot x_2$



Figuur 75: Resultaat na transformatie naar hogere ordes

10.3 Kernels

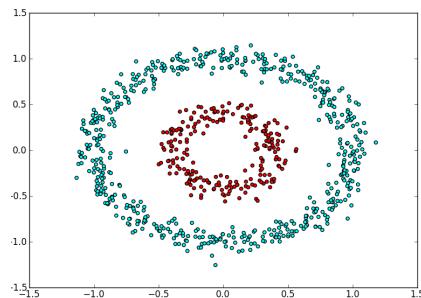
Projecteer de data in een hogere dimensie en probeer lineair te scheiden met behulp van een kernel:

10.3.1 Meest gebruikte kernels

- RBF - Radial Basis Function (Gaussiaanse kernel)
- Polynomial kernel
- Histogram kernel
- Lineaire kernel = SVM zonder kernel

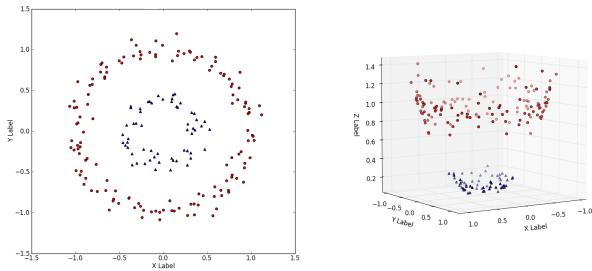
10.3.2 Voorbeeld

Hoe kan je onderstaande klassen scheiden?



Figuur 76: Voorbeeld

Oplossing: Gebruik een Gaussiaanse (RBF) kernel



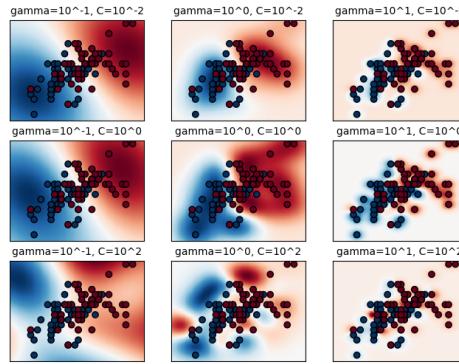
Figuur 77: Voorbeeld ('Donut problem')

- Visualisatie: <https://www.youtube.com/watch?v=3liCbRZPrZA>

10.3.3 Parameter gamma

De parameter gamma regelt de breedte van de RBF kernel

- Kleine gamma \Rightarrow brede RBF kernels
 - Te kleine gamma leidt ertoe dat het het model de complexiteit van het model niet kan capteren (**underfitting**)
- Grote gamma \Rightarrow smalle RBF kernels
 - Te grote gamma leidt tot **overfitting**
- Bij gebruik van een RBF kernel: **feature scaling** (=normalisatie) toepassen



Figuur 78: De parameter gamma regelt de breedte van de RBF kernels

10.3.4 Hyperparameters

Implementeren van een SVM:

- Test lineaire kernel (=geen kernel) en RBF kernel
- Tune de parameter C
- Bij gebruik van RBF kernel: tune zowel de parameters C als gamma

10.4 Motivatie voor het gebruik van een SVM

- Kan zowel gebruikt worden voor regressie als classificatie (en zelfs clustering).
- Werkt goed op kleine datasets (in tegenstelling tot neurale netwerken en deep learning).
- Is nog altijd effectief wanneer het aantal features groter is dan het aantal training samples.
- Het werkt goed bij een groot aantal features (high dimensional space).
- Gebruikt niet alle training examples tijdens training \Rightarrow geheugenefficiënt.
- Geen lokale minima/optima, maar globaal optimum.

10.5 Logistische regressie vs SVM

10.5.1 Wanneer welke classifier kiezen?

- Wanneer het aantal features groot is ten opzichte van het aantal training samples:
 - \Rightarrow Gebruik logistische regressie of SVM zonder kernel (=lineaire kernel)
- Wanneer het aantal features klein is en het aantal training samples behoorlijk
 - \Rightarrow Gebruik SVM met RBF kernel
- Bij een klein aantal features met een groot aantal training samples
 - \Rightarrow Creëer meer features
 - \Rightarrow Gebruik logistic regression of SVM zonder kernel (=lineaire kernel)

11 Cross-validatie

Cross-validatie gaat ons toelaten om automatisch goede parameters te vinden voor ons model

11.1 Verschillende types cross-validatie

- Hold Out cross-validation
- K-fold cross-validation
- Leave One Out cross-validation
- Bootstrap cross-validation

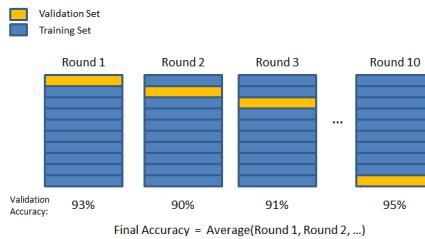
11.1.1 Hold out cross-validation

Training Data	Validation Data	Test Data

Figuur 79: Je splitst de data in training data en test data. Daarna splits je de training data nog eens op in validation data

- Training data: om model mee te trainen
- Validation data: tuning van hyper parameters en model selection
- Test data: uiteindelijke test van het best gevalideerde model op nog nooit gezien data

11.1.2 K-fold cross-validation

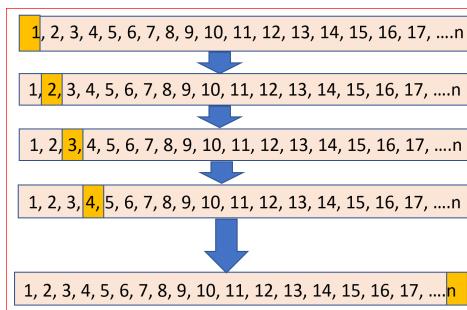


Figuur 80: Voorbeeld: 10-fold cross validation. De validation set is 1/10 van de trainingset

- We berekenen de validation accuracy K keer
- Met telkens een andere keuze trainingset en validation set
- De validation set is $1/K$ van de trainingset, de rest wordt gebruikt om te trainen
- We nemen van alle K validation accurancies het gemiddelde om zo een final accuracy te bekomen

11.1.3 Leave one out cross-validation

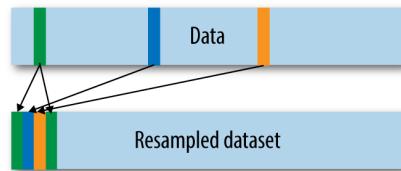
- Extreme vorm van K-fold cross-validation waarbij $K = N$ (aantal examples in de training set)
- We trainen op alle data behalve 1 sample
- De sample wordt elke ronde veranderd
- Evenveel rondes als aantal examples
- Heel veel rekenkracht nodig, maar vaak goede models



Figuur 81: Leave one out cross-validation

11.1.4 Bootstrap cross-validation

- Sampling met teruglegging (zie Bagging)
- Validatie op niet-geselecteerde examples
- We vertrekken vanaf een bestaande trainingset
- We maken een nieuwe ‘resampled’ trainingset:
 - Willekeurig samples uit de eerste trainingset kiezen
 - Sommige samples kunnen we dubbel kiezen (zie groene sample in afbeelding)
- Ongeveer 2/3 van de bestaande trainingset kopieren we naar de resampled trainingset
- Ongeveer 1/3 wordt niet gekozen
- We maken meerdere van zo’n resampled trainingsets
- Belangrijke manier om overfitting tegen te gaan

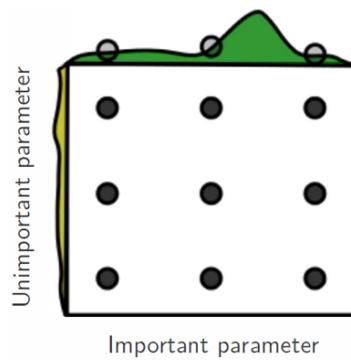


Figuur 82: Bootstrap cross-validation

11.2 Hyperparameter tuning via cross-validatie

11.2.1 Grid search

- We zetten de parameters op een grid
- We testen ons model met elke combinatie van parameters in de grid, bv:
 - x-waarden = regularisatieparameter C
 - y-waarden = gamma

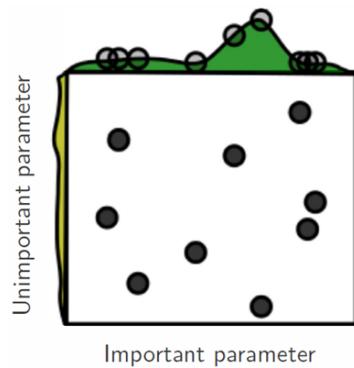


Figuur 83: Grid Search

- Stel: x-as = C , y-as = gamma

- Bij Grid Search ben je beperkt tot de combinaties die je zelf op voorhand kiest: je hangt vast aan een grid
- Het kan ook zijn dat je combinaties maakt van parameters die er eigenlijk niet toe doen
- In bovenstaande afbeelding is er weinig verschil tussen een hoge of een lage y-waarde
- Er zit wel een hoge piek bij de x-waarde (zie groene grafiek)
- Die piek wordt nooit gevonden omdat we vasthangen aan het grid.
- Oplossing: Random search

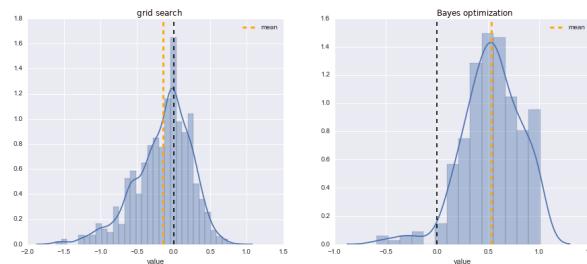
11.2.2 Random search



Figuur 84: Hetzelfde als grid search, maar met random punten

- Nu worden de punten willekeurig gekozen
- De piek wordt wel gevonden

11.2.3 Bayes optimization

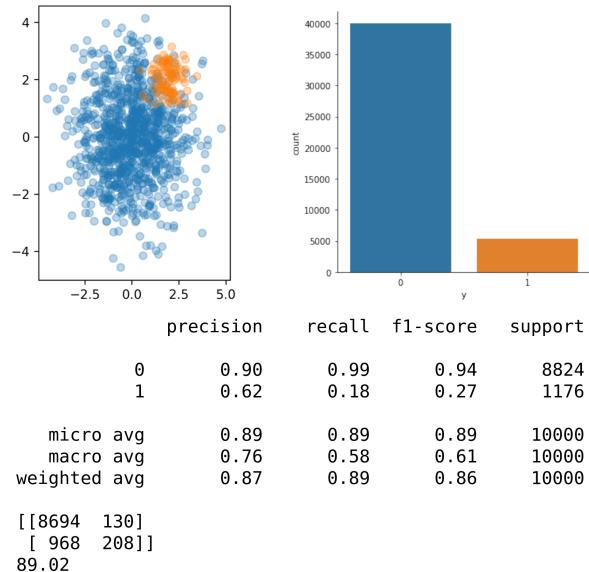


Figuur 85

- Je kan het Bayes optimization vergelijken met een randomized search
- Je test willekeurig een aantal modellen uit met verschillende gamma- en C -waarden
- Op basis van de resultaten die je haalt probeert Bayes optimization af te leiden waar het optimum ligt, zonder het expliciet te moeten uittesten
- Met weinig rekenkracht kan je toch goede resultaten bekomen

12 Niet-gebalanceerde data

12.1 Problematiek

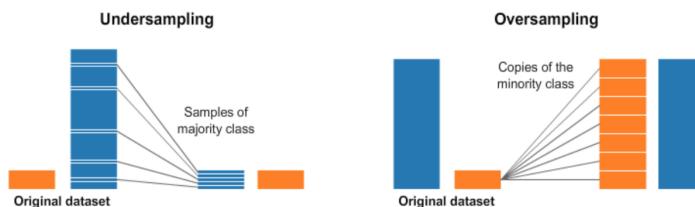


Figuur 86: De ene klasse (0, blauw) komt veel meer voor dan de andere klasse (1, oranje)

12.2 Omgaan met niet-gebalanceerde data

- Meer data verzamelen van de minderheidsklasse
- Meer features verzamelen

12.2.1 Undersampling & Oversampling



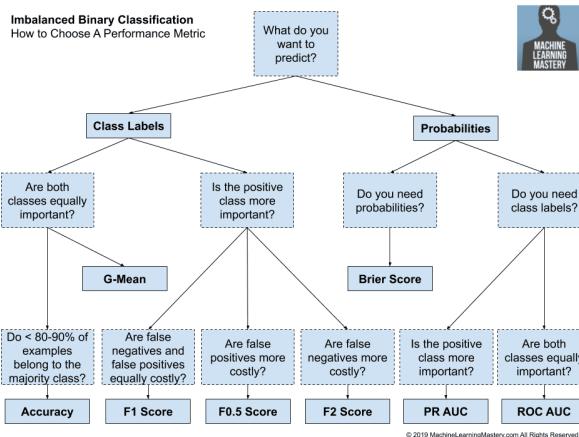
Figuur 87: Undersampling & oversampling

- We nemen enkel een deel van de samples tot we er evenveel hebben als de minderheidsklasse (undersampling)
 - Kies de samples waar de classifier het meeste fouten op maakt
- We nemen kopieën van de minderheidsklasse tot we er evenveel hebben als de meerderheidsklasse

12.2.2 Andere scoring parameter/metric kiezen

In plaats van het model te kiezen dat de hoogste accuracy oplevert, kies bijvoorbeeld het model dat de hoogste f1-score oplevert.

- https://scikit-learn.org/stable/modules/model_evaluation.html
- Om een globaal idee te krijgen van hoe goed het model scoort, kan je de verschillende gemiddeldes van f1 berekenen:
 - f1, f1_micro, f1_macro, f1_weighted, f1_samples



Figuur 88: Overzicht: Hoe een performance metric kiezen

12.2.3 Class-weight balancing

Misclassificatie van samples uit de minderheidsklasse leiden tijdens de training tot een hogere loss.

- Je geeft meer ‘gewicht’ aan een bepaalde klasse
- Ook al gaat de accuracy naar beneden, wil je er toch voor zorgen dat een bepaalde klasse belangrijker wordt

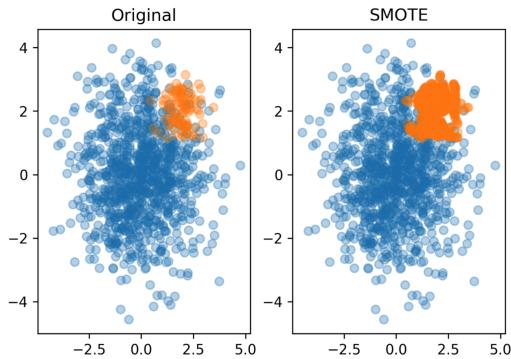
```

1 model = LogisticRegression(C=1, solver='liblinear', class_weight='balanced')
2
3 # voorbeeld: klasse 1 moet 20 keer belangrijker zijn dan klasse 0
4 # stel dat klasse 0 goede en klasse 1 kwaadaardige tumors zijn
5 class_weight = {0: 1, 1:20}
  
```

	precision	recall	f1-score	support
0	0.94	0.77	0.85	8824
1	0.27	0.62	0.38	1176
micro avg	0.76	0.76	0.76	10000
macro avg	0.60	0.70	0.61	10000
weighted avg	0.86	0.76	0.79	10000
	[[6830 1994] [444 732]] 75.62			

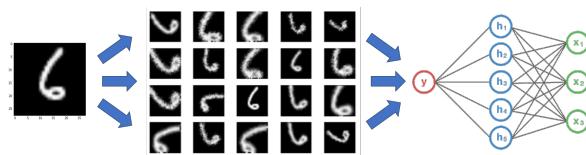
Figuur 89: Omdat klasse 1 veel minder samples heeft, scoort het model slechter bij die samples

12.2.4 Data augmentation - SMOTE (Synthetic Minority Over-sampling Technique)



- Men neemt een willekeurige sample van de minderheidsklasse
- Men kijkt naar een aantal buren ervan
- Tussen de sample en zijn buren maak je nieuwe samples aan
- Beter dan willekeurig nieuwe data aanmaken

12.2.5 Data augmentation - image augmentation

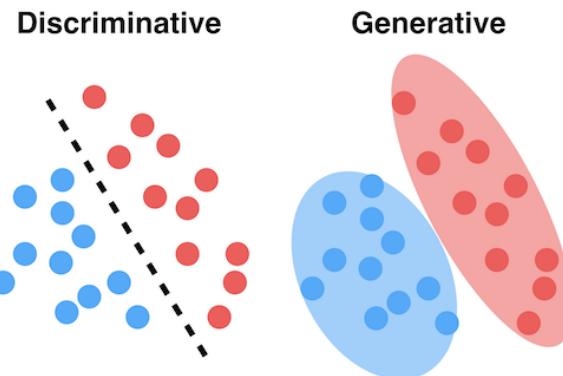


Figuur 90: Image augmentation

- We maken nieuwe images aan
- We bewerken de nieuwe images: roteren, scalen, filters op toepassen
- De nieuwe images voegen we toe aan onze dataset

13 Naive Bayes & Natural Language Processing

13.1 Discriminative vs generative classification



Figuur 91: Discriminative vs generative

- Discriminative classifiers leren de **grens tussen twee klassen**
 - leert $p(y|x)$
 - p = probability, kans
 - = kans dat een punt tot de klasse y behoort, gegeven de eigenschappen van de features x
- Generative classifiers leren de **distributie van de verschillende klassen**
 - leert $p(x|y)$
 - = kans gegeven dat iets tot een bepaalde klasse y behoort, hoe zien de features x er uit?
 - leert ook $p(y)$ = de prior = de totale kans dat iets tot een bepaalde klasse y behoort
 - Zoekt naar hoe features van een bepaalde klasse er uit zien
 - Wat hebben alle punten in een klasse gemeenschappelijk?

13.2 Bayes rule

13.2.1 Voorbeeld: kanker

De kans dat iemand kanker heeft bedraagt $1\% \Rightarrow P(\text{kanker}) = 0.01$

Van een test is geweten dat:

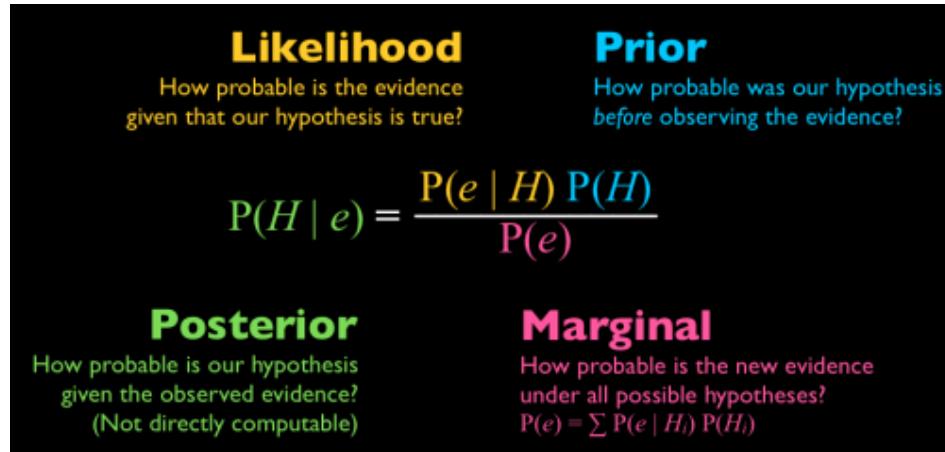
- In 90% van de gevallen is de test positief als je effectief kanker hebt (sensitiviteit)
- In 90% van de gevallen is de test negatief als je geen kanker hebt (specificiteit)

VRAAG: Als de test positief blijkt, wat is de kans dat je kanker hebt?

13.2.2 Uitwerking via Bayes Rule

$$P(\text{kanker}|\text{positief}) = \frac{P(\text{positief}|\text{kanker}) \cdot P(\text{kanker})}{P(\text{positief})}$$
$$P(\text{kanker}|\text{positief}) = \frac{0.9 \cdot 0.1}{0.01 \cdot 0.9 + 0.1 \cdot 0.99} = 0.0833 \approx 8\%$$

- **Prior:** $P(\text{kanker})$: Zonder de testresultaten te kennen, hoe waarschijnlijk is het dat iemand kanker heeft
- **Likelihood:** $P(\text{positief}|\text{kanker})$: Hoe waarschijnlijk is het dat de test positief heeft, wanneer de persoon effectief kanker heeft?
- **Marginal:** $P(\text{positief})$: Hoe waarschijnlijk is het dat de test positief is
- **Posterior:** $P(\text{kanker})$: Wat is de kans dat iemand effectief kanker heeft als de test positief blijkt?



Figuur 92

13.3 Naive Bayes - tekstclassificatie

= een classificatietechniek

13.3.1 Spamdetectie

$$p(H|e) = \frac{p(e|H) \cdot p(H)}{p(e)}$$

- H = hypothese (het bericht is spam)
- e = evidence (de tekst in het bericht)
- $p(H|e)$: de kans dat een bericht spam is gegeven de tekst van een bericht
- $p(e|H)$: de kans dat we deze tekst vinden in een spam bericht
- $p(H)$: de kans dat een willekeurig bericht spam is
- $p(e)$: de kans dat we deze tekst tegenkomen/waarnemen. De tekst moet worden weergegeven als meerdere stukken evidence: de woorden w_1, w_2, \dots, w_n

$$\begin{aligned} p(\text{Spam}|w_1, \dots, w_n) &= \frac{p(w_1, \dots, w_n|\text{spam}) \cdot p(\text{spam})}{p(w_1, \dots, w_n)} \\ &= \frac{p(w_1|w_2, \dots, w_n, \text{spam}) \cdot p(w_2|w_3, \dots, w_n, \text{spam}) \cdot p(\text{spam})}{p(w_1, \dots, w_n)} \end{aligned}$$

- $p(w_1|w_2, \dots, w_n, \text{spam})$ is de kans om het eerste woord te vinden, gegeven alle woorden en gegeven een bericht spam is

- Lastig te berekenen \Rightarrow Naive Bayes veronderstelling (Naive = vereenvoudigd)
- **Naive Bayes** = beschouw elk woord onafhankelijk van de andere woorden

13.3.2 Met Naive Bayes

$$p(spam|w_1, \dots, w_n) = \frac{p(w_1|spam) \cdot p(w_2|spam) \cdots p(w_n|spam) \cdot p(spam)}{p(w_1, \dots, w_n)}$$

$p(w_i|spam)$ = de kans dat w_i in een spambericht voorkomt

$$p(spam|w_1, \dots, w_n) = \frac{p(spam) \cdot \prod_{i=1}^n p(w_i|spam)}{p(w_1, \dots, w_n)}$$

Vereenvoudigen

Kies spam of ham (=niet spam) op basis van welke deze klassen de grootste kans heeft. We hebben geen kansen nodig \Rightarrow laat de noemer (=Marginal) vallen

$$p(spam|w_1, \dots, w_n) = p(spam) \cdot \prod_{i=1}^n p(w_i|spam)$$

13.3.3 Voorbeeld

WOORD	SPAM		HAM	
	frequentie	kans	frequentie	kans
free	90	0,32	200	0,21
Viagra	60	0,21	55	0,059
buy	120	0,43	290	0,31
advice	50	0,18	380	0,41
get	180	0,64	510	0,55

Figuur 93: Aantal spam = 280, aantal ham = 930

$$\bullet p(spam) = \frac{280}{280+930} = \frac{280}{1210} = 0.23 = 23\%$$

$$\bullet p(ham) = \frac{930}{280+930} = \frac{930}{1210} = 0.77 = 77\%$$

Classificeer de zin: 'Get free advice'

Kans op spam:

$$\begin{aligned} p(spam|get\ free\ advice) &= p(spam) \cdot p(get|spam) \cdot p(free|spam) \cdot p(advice|spam) \\ &= 0.23 \cdot 0.64 \cdot 0.32 \cdot 0.18 = 0.0085 \end{aligned}$$

Kans op ham:

$$\begin{aligned} p(ham|get\ free\ advice) &= p(ham) \cdot p(get|ham) \cdot p(free|ham) \cdot p(advice|ham) \\ &= 0.77 \cdot 0.55 \cdot 0.21 \cdot 0.41 = 0.0365 \end{aligned}$$

Conclusie: $0.0365 > 0.0085 \Rightarrow$ bericht is ham

13.4 Laplacian smoothing

Classificeer de zin: 'Get free valium'

- Valium behoort niet tot de trainingset $\Rightarrow p(valium|spam) = 0$ en $p(valium|ham) = 0$
- Het probleem: $p(spam|get free valium) = 0$ en $p(ham|get free valium)$

Oplossing: Laplacian smoothing (= add-one smoothing): kent niet geziene woorden en toch een zekere kans van voorkomen toe

$$P(w_s) = \frac{C(w_s) + \alpha}{N + \alpha \cdot V} \quad (19)$$

- $P(w)$ = de kans op het woord w
- $C(w)$ = het aantal keer dat het woord w voorkomt
- N = het totaal aantal woorden
- V = het aantal verschillende woorden (=vocabulaire grootte)
- α = een hyperparameter die dus getuned moet worden

WOORD	SPAM		HAM	
	freq.	Laplacian smoothing	freq.	Laplacian smoothing
free	90	$\frac{90+1}{280+1 \times 6} = \frac{91}{286} = 0.318$	200	$\frac{200+1}{930+1 \times 6} = \frac{201}{936} = 0.215$
Viagra	60	$\frac{60+1}{280+1 \times 6} = \frac{61}{286} = 0.213$	55	$\frac{55+1}{930+1 \times 6} = \frac{56}{936} = 0.060$
buy	120	$\frac{120+1}{280+1 \times 6} = \frac{121}{286} = 0.423$	290	$\frac{290+1}{930+1 \times 6} = \frac{291}{936} = 0.311$
advice	50	$\frac{50+1}{280+1 \times 6} = \frac{51}{286} = 0.0178$	380	$\frac{380+1}{930+1 \times 6} = \frac{381}{936} = 0.407$
get	180	$\frac{180+1}{280+1 \times 6} = \frac{181}{286} = 0.633$	510	$\frac{510+1}{930+1 \times 6} = \frac{511}{936} = 0.546$
Valium	0	$\frac{0+1}{280+1 \times 6} = \frac{1}{286} = 0.003$	0	$\frac{0+1}{930+1 \times 6} = \frac{1}{936} = 0.001$

Figuur 94: Laplacian smoothing met $\alpha = 1$

Kans op spam:

$$\begin{aligned} p(spam|get free valium) &= p(spam) \cdot p(get|spam) \cdot p(free|spam) \cdot p(valium|spam) \\ &= 0.23 \cdot 0.633 \cdot 0.318 \cdot 0.003 = 0.0001389 \end{aligned}$$

Kans op ham:

$$\begin{aligned} p(ham|textgetfreevalium) &= p(ham) \cdot p(get|ham) \cdot p(free|ham) \cdot p(valium|ham) \\ &= 0.77 \cdot 0.546 \cdot 0.215 \cdot 0.001 = 0.00009 \end{aligned}$$

$0.0001389 > 0.00009 \Rightarrow$ bericht is spam

13.4.1 Invloed van de hyperparameter alpha

$$P(w_s) = \frac{C(w_s) + \alpha}{N + \alpha \cdot V}$$

- Kleine α : neiging tot overfitting, model wordt te complex
- Grote α : neiging tot underfitting, model wordt te simpel

13.5 Log likelihood

Vermenigvuldigen van veel kansen kan resulteren in een floating-point underflow = de uitgekomen waarde is te klein, kleiner dan de computer in memory kan bewaren.

Oplossing: gebruik van log likelihood:

$$\log(xy) = \log(x) + \log(y) \quad (20)$$

$$\log(p(\text{spam}|w_1, \dots, w_n)) = \log(p(\text{spam})) + \sum_{i=1}^n \log(p(w_i|\text{spam}))$$

Dit wordt meestal intern door de Python libraries gedaan

$$\begin{aligned} p(\text{spam}|\text{get free Valium}) &\propto p(\text{spam}).p(\text{get}|\text{spam}).p(\text{free}|\text{spam}).p(\text{Valium}|\text{spam}) \\ &= 0,23 \cdot 0,633 \cdot 0,318 \cdot 0,003 = 0,0001389 \\ \\ \log(p(\text{spam}|\text{get free Valium})) &\propto \log(p(\text{spam})) + \log(p(\text{get}|\text{spam})) + \log(p(\text{free}|\text{spam})) + \log(p(\text{Valium}|\text{spam})) \\ &= \log(0,23) + \log(0,633) + \log(0,318) + \log(0,003) = -3,857 \\ \\ p(\text{ham}|\text{get free advice}) &\propto p(\text{ham}).p(\text{get}|\text{ham}).p(\text{free}|\text{ham}).p(\text{Valium}|\text{ham}) \\ &= 0,77 \cdot 0,546 \cdot 0,215 \cdot 0,001 = 0,00009 \\ \\ \log(p(\text{ham}|\text{get free Valium})) &\propto \log(p(\text{ham})) + \log(p(\text{get}|\text{ham})) + \log(p(\text{free}|\text{ham})) + \log(p(\text{Valium}|\text{ham})) \\ &= \log(0,77) + \log(0,546) + \log(0,215) + \log(0,001) = -4,044 \\ \\ -3,857 > -4,044 &\Rightarrow \text{bericht is spam} \end{aligned}$$

Figuur 95: Log likelihood met tekstclassificatie

13.6 Tekstclassificatie in de praktijk

13.6.1 Preprocessing - opkuisen van de tekst

```
1 # verwijder html
2 from bs4 import BeautifulSoup
3 text_no_html = BeautifulSoup(str(text), "html.parser").get_text()
4
5 # verwijder niet-letters
6 import re # regular expressions
7 text_alpha_chars = re.sub("[^a-zA-Z]", " ", str(text_no_html))
8
9 # converteer naar lowercase
10 text_lower = text_alpha_chars.lower()
11
12 # verwijder stopwoorden
13 from nltk.corpus import stopwords
14 stops = set(stopwords.words(language))
15 text_no_stop_words = ""
16 for w in text_lower.split():
17     if w not in stops:
18         text_no_stop_words = text_no_stop_words + w + " "
```

13.6.2 Preprocessing - herleiden van woorden tot de stam

= Stemming

```
1 from nltk.stem.snowball import SnowballStemmer
2
3 text_stemmer = " "
4 stemmer = SnowballStemmer(language)
5 for w in text_no_stop_words.split():
6     text_stemmer = text_stemmer + stemmer.stem(w) + " "
```

13.6.3 Preprocessing - verwijder te korte woorden

```
1 text_no_short_words = " "
2 for w in text_stemmer.split():
3     if len(w) >= minWordSize:
4         text_no_short_words = text_no_short_words + w + " "
```

13.6.4 Opbouwen van feature vectors - bag of words

- **Bag of words** = collectie van unieke woorden die voorkomen in de volledige trainingset. De uiteindelijke feature vector heeft dezelfde dimensie als de bag of words
- **Multi-variate Bernoulli Naive Bayes**: er wordt een document bijgehouden of een woord in de bag of words al dan niet voorkomt in een document (0 of 1) en dus niet de frequentie van voorkomen
- **Multinomial Naive Bayes** = de frequentie (aantal keer) waarmee een woord uit de bag of words voorkomt in het document wordt bijgehouden.
- In scikit-learn: CountVectorizer

tf-idf transformer: term frequency-inverse document frequency transformer

Sommige woorden die veel voorkomen in documenten zijn minder belangrijk. Met tf-idf transforming gaan we het gewicht van die woorden verminderen.

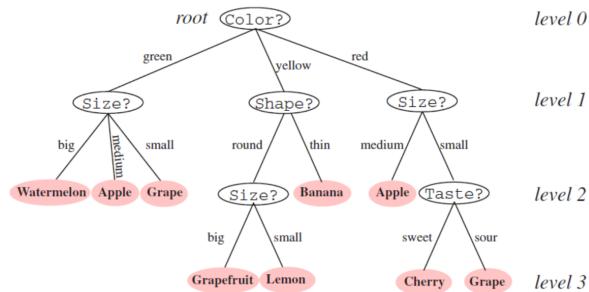
$$tfidf_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

- $tf_{i,j}$ = het aantal keer dat woord i voorkomt in document j
- df_i = het aantal woorden waar woord i in voorkomt
- N = totaal aantal documenten
- In scikit-learn: TfidfTransformer

```
1 # je moet eerst de CountVectorizer toepassen voor je de TfidfTransformer kan gebruiken
2 count_vect = CountVectorizer()
3 X_train_bag_of_words = count_vect.fit(X_train)
4 X_train_bag_of_words = count_vect.transform(X_train)
5
6 tfidf_transformer = TfidfTransformer()
7 tf_transformer = TfidfTransformer().fit(X_train)
8 X_train_tf = tf_transformer.transform(X_train_bag_of_words)
```

14 Decision Trees

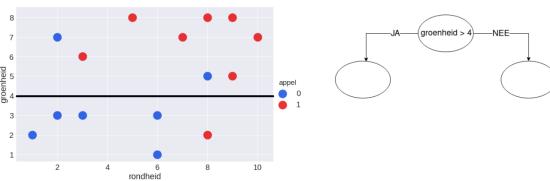
14.1 Decision Trees voor classificatie



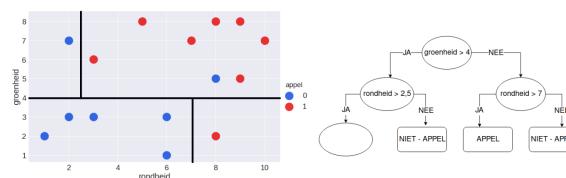
Figuur 96: Voorbeeld: een stuk fruit classificeren aan de hand van een decision tree

14.1.1 Voorbeeld met rondheid & groenheid van appels

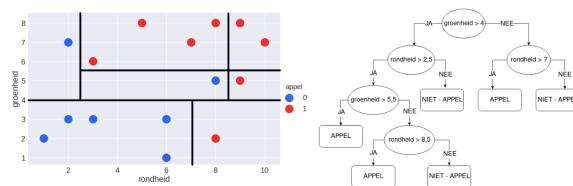
Bij ons voorbeeld van het classificeren van appels verdelen we nu de dataset herhaaldelijk in stukken:



Figuur 97: Is de groenheid > 4? We trekken een lijn bij groenheid == 4



Figuur 98: We voegen meerdere opsplitsingen toe



Figuur 99: Eindresultaat

14.1.2 Entropy

Hoe bepalen waar de boom te splitsen?

Een goede split is deze waarbij de wanorde afneemt en beide kanten ‘zuiverder (pure) worden’

Definitie 14.1 De entropy (H) is de maat voor de wanorde (gemiddelde informatieinhoud)

$$H = \sum_{i=1}^N p_i \cdot \log_2 \left(\frac{1}{p_i} \right) \quad (21)$$

$$H = - \sum_{i=1}^N p_i \cdot \log_2(p_i) \quad (22)$$

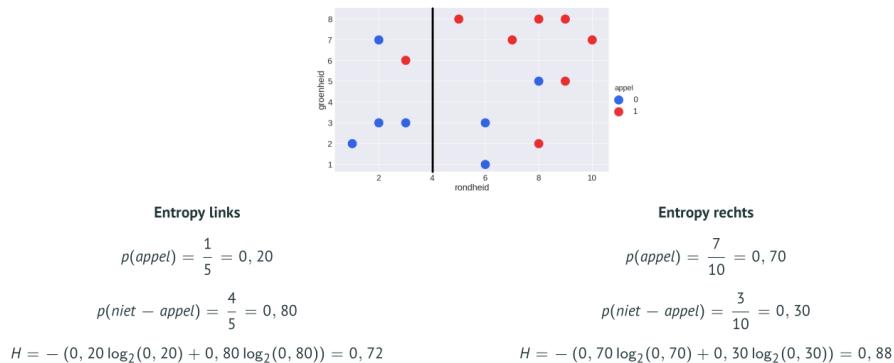
Voorbeeld

Als we terugkijken naar de plot van appels, zien we 15 datapunten, waarvan:

- Kans op een appel: $\frac{8}{15} = 0.53$
 - Kans op een niet-appel: $\frac{7}{15} = 0.47$
- $\Rightarrow H = -(0.53 \cdot \log_2(0.53) + 0.47 \cdot \log_2(0.47)) = -(-0.99740) = 0.9974$

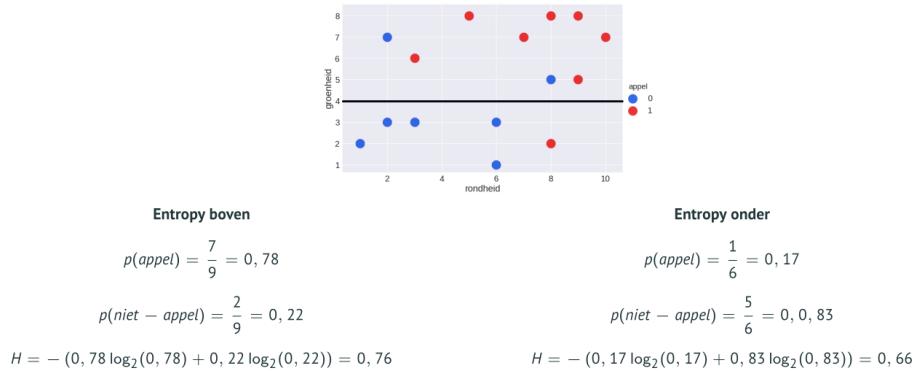
14.1.3 Information Gain

Definitie 14.2 Information Gain = (entropy voor de split) – (entropy na de split)



Figuur 100: Splitsen boven & onder

$$\text{Information gain} = 0.9974 - (\frac{5}{15} \cdot 0.72 + \frac{10}{15} \cdot 0.88) = 0.17073$$



Figuur 101: Splitsen links & rechts

$$\text{Information gain} = 0.9974 - (\frac{9}{15} \cdot 0.76 + \frac{6}{15} \cdot 0.66) = 0.27740$$

14.1.4 Gini impurity

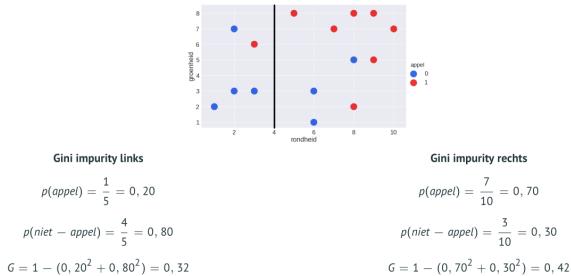
Alternatief voor het gebruik van de entropy.

Definitie 14.3 De *Gini Impurity* is een maat voor hoe vaak een willekeurig gekozen element van de set verkeerd gelabeld zou worden als het willekeurig werd gelabeld volgens de distributie van de labels in de subset

$$G = 1 - \sum_{i=1}^N p_i^2 \quad (23)$$

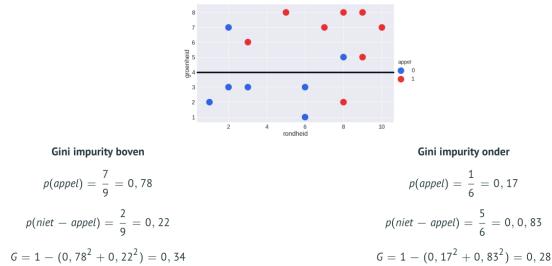
Voorbeeld

- Kans op een appel: $\frac{8}{15} = 0.53$
 - Kans op een niet-appel: $\frac{7}{15} = 0.47$
- $\Rightarrow G = 1 - (0.53^2 + 0.47^2) = 0.4982$



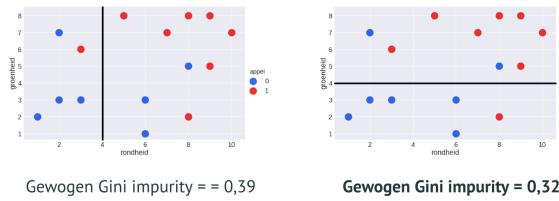
Figuur 102: Gini impurity links & rechts

$$\text{Gewogen Gini impurity} = \frac{5}{15} \cdot 0.32 + \frac{10}{15} \cdot 0.42 = 0.39$$



Figuur 103: Gini impurity boven & onder

$$\text{Gewogen Gini impurity} = \frac{9}{15} \cdot 0.34 + \frac{6}{15} \cdot 0.28 = 0.32$$



Figuur 104: Splits telkens bij de split die je de laagste Gini impurity oplevert

14.2 Decision Trees voor regressie

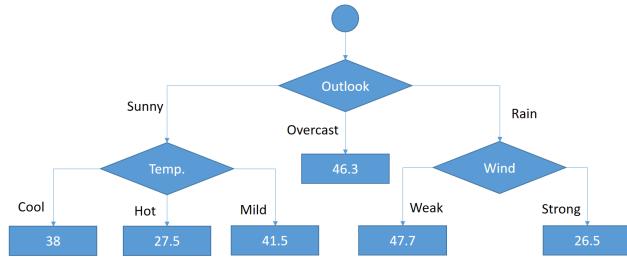
Decision trees kunnen ook gebruikt worden op continue waarden, om regressie mee te doen.

14.2.1 Voorbeeld 1: aantal golfspelers bij verschillende weersomstandigheden

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
3	Overcast	Hot	High	Weak	46
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
7	Overcast	Cool	Normal	Strong	43
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
10	Rain	Mild	Normal	Weak	46
11	Sunny	Mild	Normal	Strong	48
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44
14	Rain	Mild	High	Strong	30

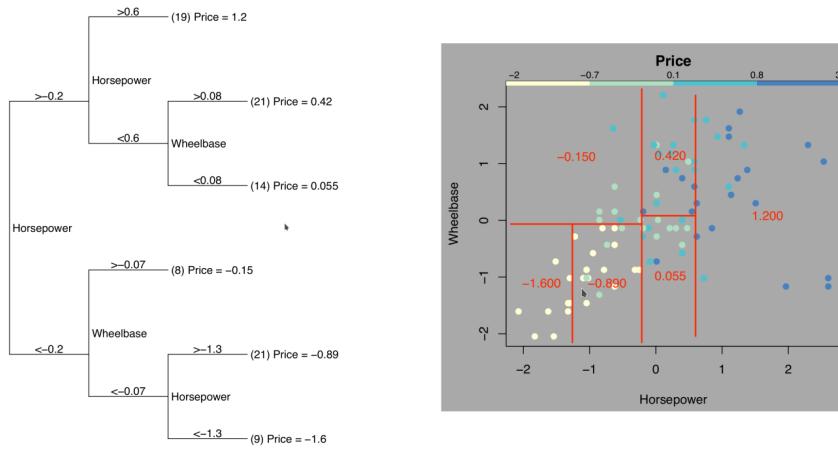
Figuur 105: De dataset

We gaan via gemiddeldes berekenen hoeveel elk 'blad' in de decision tree zal zijn:



Figuur 106: De decision tree

14.2.2 Voorbeeld 2: Autoprijs op basis van 2 eigenschappen van de auto



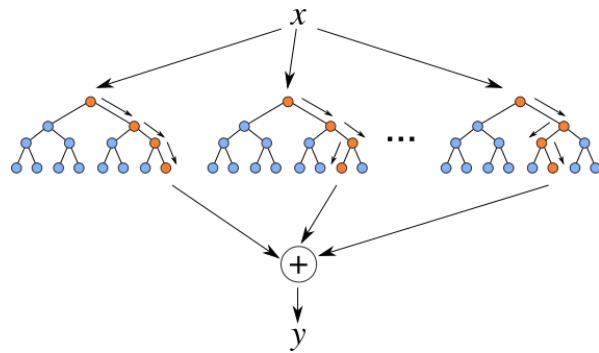
Figuur 107: We splitsen de data op, juist zoals bij classificatie

15 Ensemble learning

15.1 Problematiek van decision trees

Decision trees hebben de neiging tot overfitting

Oplossing: combineer de voorspellingen van verschillende gerandomiseerde modellen (bijvoorbeeld decision trees) tot één model = ensemble learning



Figuur 108: Combineren van meerdere modellen tot één model

Zo'n combinatie van verschillende decision trees (beslissingsbomen) noemen we een forest (bos)

15.2 Overzicht Ensemble learning methodes

- **Bootstrap aggregating (=bagging)**
- **Boosting**
- Bayes optimal classifier
- Bayesian model averaging/combination
- Bucket of models
- **Stacking**

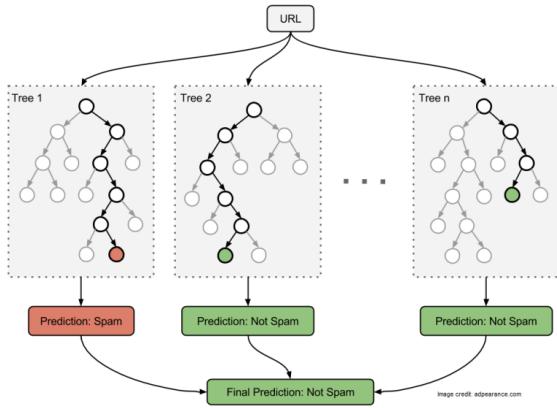
15.3 Bagging

15.3.1 Concept van bagging - Majority voting

Given a jury of voters and assuming independent errors. If the probability of each single person in the jury of being correct is above 50% then the probability of the jury being correct tends to 100% as the number of persons increase.

Nicolas de Condorcet (1743 - 1794)





Figuur 109: Verschillende decision trees samenvoegen tot één finaal model

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7	Sample 8	Sample 9	Sample 10	
Tree 1											7/10 correct
Tree 2											7/10 correct
Tree 3											6/10 correct
Majority voting											8/10 correct

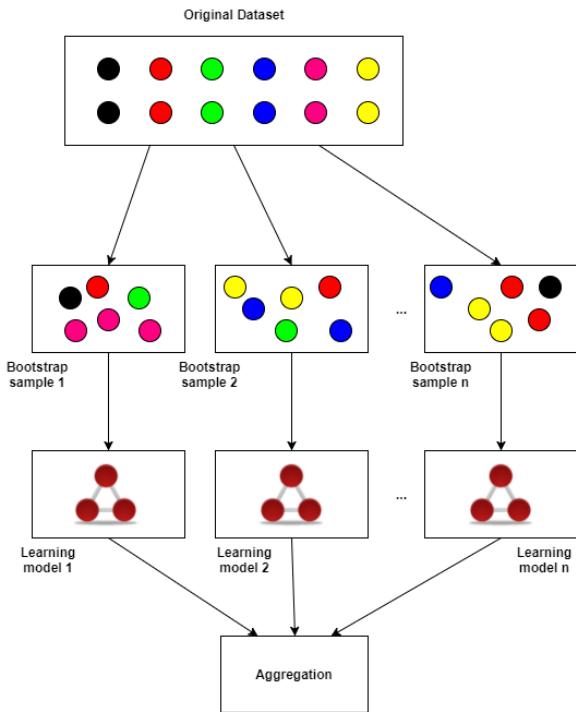
█ Correct
█ Foutief

Figuur 110: Het finale model keurt een sample foutief als het door de meerderheid van de bomen foutief wordt gekeurd

- We kiezen de klasse op basis van de meerderheid (=er wordt 'gestemd')
- Als er 1 model overfitting doet (bvb: ten gevolge van ruis), zullen andere modellen dat misschien niet doen
- Als je zo genoeg modellen traat haal je zo goed als alle ruis er uit
- Bagging is daarom goed bij uitschieters

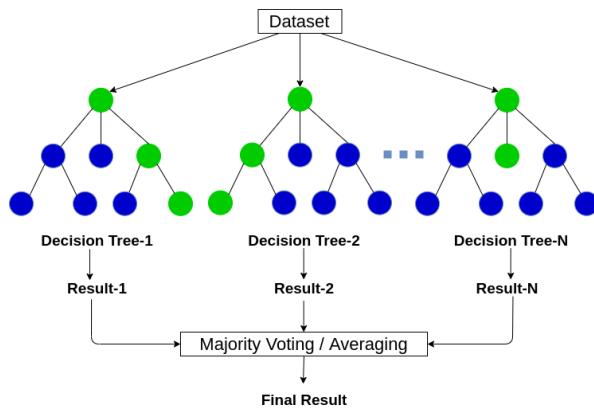
15.3.2 De techniek van bagging

Methode om de variantie te reduceren en overfitting te vermijden



1. Kies verschillende subsets (bags) uit de trainingset
2. Willekeurige selectie met teruglegging
3. Train op elke subset een classifier

15.3.3 Random Forest Trees

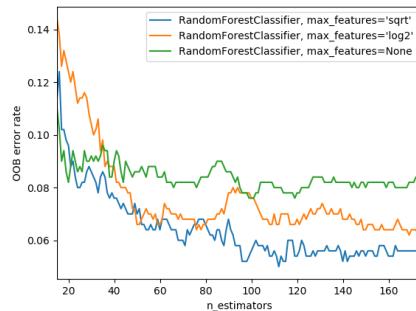


- = Verschillende decision trees combineren tot 1 bos = 1 groot model
- Dat moet niet noodzakelijk met pure enkel decision trees, je mag dat ook doen met andere classifiers zoals Logistic Regression
- ExtraTrees = aangepaste versie van de Random Forest Trees
 - = Extremely Randomized Trees

- Werken niet met bags, maar met de volledige trainingset waar samples uit gehaald worden
- Splits worden willekeurig gekozen
- \Rightarrow Entropy en Gini impurity hoeft je niet uit te rekenen \Rightarrow minder rekenwerk
- Deze methode werkt goed met groot aantal bomen
- Vaak minder goede accuracy dan andere methodes, maar werkt heel snel

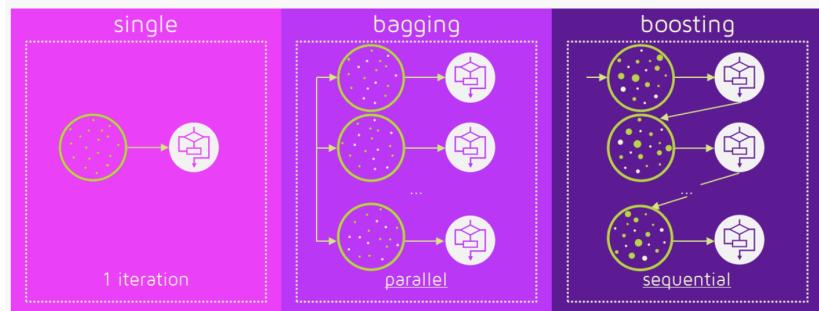
Hyperparameters:

- **n_estimators:** aantal bomen in het bos: meestal hoe meer hoe beter
- **criterion:** Gini (default) of Entropy
- **Maximum number of features:** het maximum aantal features per boom
 - int: aantal features
 - float: percentage
 - 'auto' max_features: vierkantswortel van totaal aantal features
 - 'sqrt': max_features: vierkantswortel van totaal aantal features
 - 'log2': log van het aantal features
 - Default worden alle features gebruikt
- **max_depth:** de maximale diepte van de boom. Als je te maken hebt met noisy data is het aan te raden de maximale diepte beperkt te houden
- **min_samples_split:** het minimum aantal samples nodig om binnen een boom te blijven splitsen
- **min_samples_leaf:** het minimum aantal samples dat zich aan een blad van de boom moet bevinden. Hoe groter deze waarde, hoe minder vatbaar voor overfitting
- **Bootstrap aggregating:** Bagging (zie verder). Staat default op True
- **oob_score:** 'out of bag'-score = de gemiddelde error bij het testen van een sample op bomen die niet op deze sample getraind zijn geweest



Figuur 111: number of estimators vs oob score

15.4 Boosting

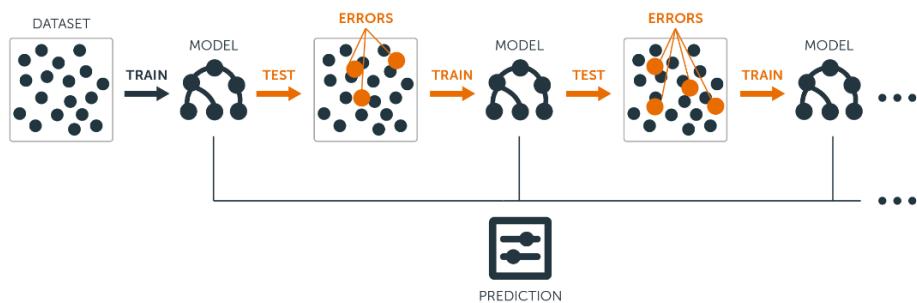


Figuur 112: Single vs bagging vs boosting

- Boosting zal zoals bagging meerdere decision trees overlopen
- Boosting zal dat in plaats van parallel, sequentieel doen
- Boosting modellen zijn heel gevoelig voor overfitting
- Modellen kunnen minder snel getraind worden dan bij Bagging, omdat het vorige model moet wachten op
- Er zijn meerdere vormen van Boosting
 - Adaboost
 - Gradient Boosting
 - ...

15.4.1 Adaboost

= Adaptive boosting



Figuur 113: Werking Adaboost

We beginnen juist zoals bij bagging:

- Je vertrekt opnieuw van de trainingsset, bv 1000 samples
- Eerste stap = je vult zoals bij bagging een ‘bag’ (bv opnieuw 1000 samples, via sampling en replacement: sommige samples gaan meerdere keren voorkomen, andere niet, ...)

- Je traint een model met de data uit die bag
- Dit model kan eender welk soort model zijn, maar meestal neemt men een decision tree met maar 1 splitsing (=stump)
- Je test dit model, eventueel op de data die niet werd geselecteerd uit de trainingset

Maar hier komt Adaboost aan te pas;

- Je traint een tweede model met opnieuw willekeurige data
- Maar: data die door het vorige model verkeerd werd geklassificeerd krijgt nu een grotere kans om in de nieuwe bag te komen
- Zo wordt model 2 meer getraind op de tekortkomingen van het eerste model
- Dit doe je herhaaldelijk voor x aantal modellen
- Via majority voting krijg je 1 finaal model

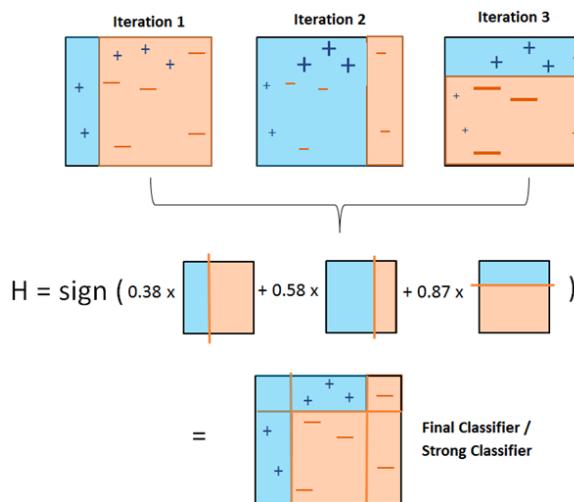
Je kan blijven modellen maken tot je een bepaalde accuracy bekomt

Maar: hoe meer modellen je traaint, hoe meer kans op overfitting. De bags van de latere modellen bevatten alsmaar meer ruis.

Learning rate

= voor welk percentage mogen de volgende modellen meetellen?

Stel dat we de learning rate op 0.9 zetten, dan telt het eerste model mee voor 0.9^0 , het tweede voor 0.9^1 , het derde voor 0.9^2 , het vierde voor 0.9^3 , ...

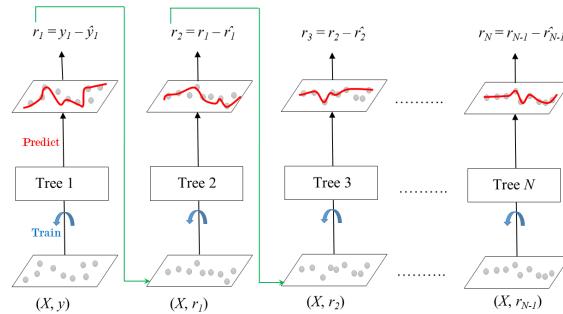


Figuur 114: Adaboost zal in zijn originele vorm met stumps (=decision trees met 1 splitsing) werken

- We splitsen bij elke iteratie de trainingsset in 2
- Bij de eerste iteratie zijn er 3 punten negatief geklassificeerd terwijl ze positief zijn
- Bij de tweede iteratie is er een nieuwe splitsing. Deze keer krijgen de verkeerd geklassificeerde punten uit het vorig model meer gewicht
- Maar nu zijn er 3 negatieve punten die positief geklassificeerd worden

- Daarom gaan we nu bij de derde iteratie opnieuw een splitsing doen.
- Deze keer krijgen die verkeerd geclasseerde punten meer gewicht
- Op basis van deze modellen met hun gewicht maken we een finaal model (=voting met gewicht)

15.4.2 Gradient Boosting

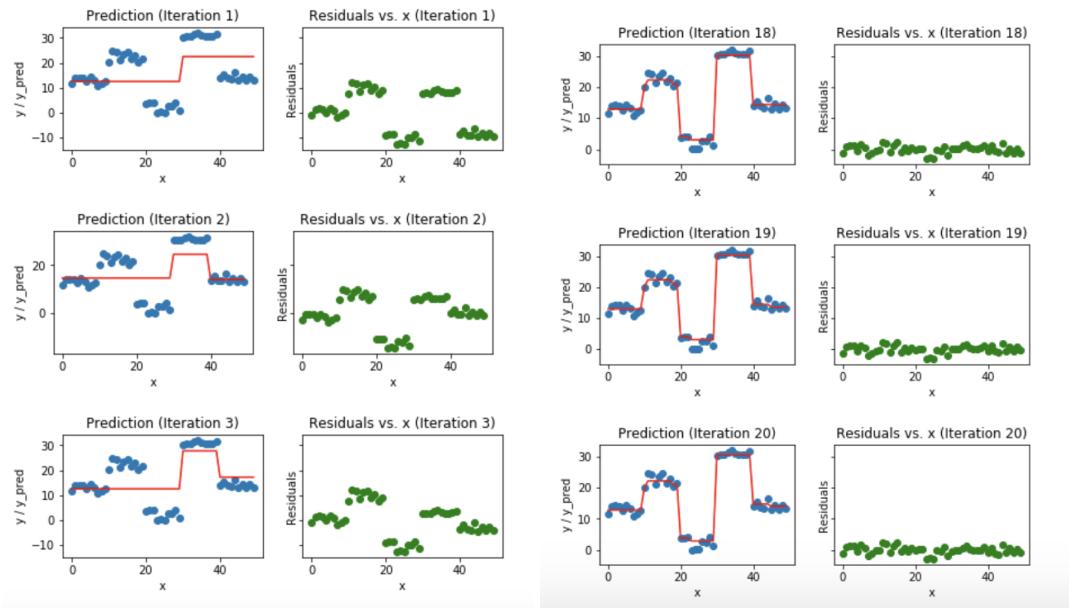


Figuur 115: Gradient Boosting

- Gelijkwaardig aan Adaboost: de volgende trees worden vooral getraind op de tekortkomingen van de vorige trees
- Model 1 maakt een ruwe schatting met een trainingset
- Model 2 zal met dezelfde trainingset voorspellen hoe fout het eerste model was
- Deze 2 modellen combineren we tot 1 model
- Model 3 zal voorspellen hoe fout de combinatie van model 1 en 2 was, om daaruit een nieuw model te halen
- Model 4 zal hetzelfde doen met model $1 + 2 + 3$
- ...
- Hier ook veel last van uitschieters en overfitting

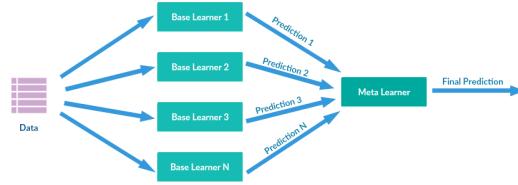
Bij regressie: de fouten worden bepaald door de residuen van

Bij classificatie: de fouten worden bepaald door de procentuele zekerheden van een voorspelling. Als een model niet zeker genoeg of te zeker was over een voorspelling, wordt die zekerheid aangepast door het volgende model.



Figuur 116: Bij elke iteratie worden de residuen berekend

15.5 Stacking

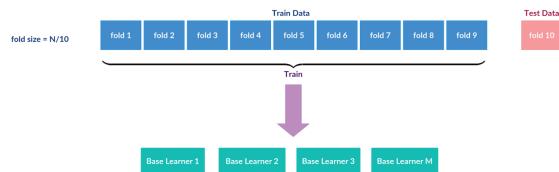


Figuur 117: De Meta Learner is een classificatiemodel die de voorspelligen zal combineren tot 1 finale voorspelling

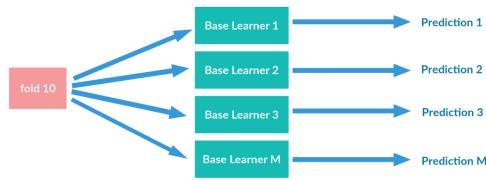
- Stacking zal alle predicties van een groep (meestal verschillende) base learners combineren tot een finale predictie
- Die base learners zijn bijvoorbeeld: SVM, LogisticRegression, Naive Bayes, RFT, ...
- Als meta learner wordt heel vaak Logistic Regression gebruikt

15.5.1 Folding

Intern wordt de data eerst geprocessed, de dataset wordt opgedeeld in folds:



Figuur 118: $K=10 \Rightarrow 9/10$ folds worden gebruikt als training data



Figuur 119: 1/10 wordt als test data gebruikt

- Die test data wordt ook door de getrainde base learning modellen gestuurd
- Met die resultaten zal de meta learner trainen

15.6 Overzicht Ensemble learning

Bagging	Boosting	Stacking
<ul style="list-style-type: none"> • Reduceert de variantie • Robuust tegen uitschieters en noisy data • Dikwijls via Random Forest Trees 	<ul style="list-style-type: none"> • Verhoogt de accuraatheid • Niet robuust tegen uitschieters of noisy data 	<ul style="list-style-type: none"> • Ensemble van strong learners • Metalearner die de optimale combinatie leert van de base learners

16 Unsupervised learning

16.1 Introductie

Unsupervised learning bevat een set aan machine learning technieken om in niet-gelabelde data structuren te zoeken en beschrijven

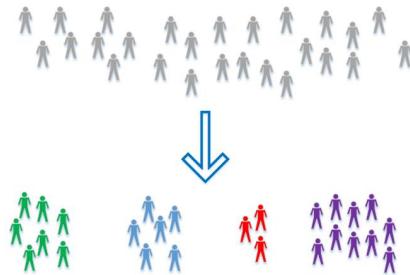
Belangrijke technieken omvatten:

- Clustering
- Anomaly / outlier detection

- Dimensionality reduction (=data comprimeren zodat je alleen relevante features overhoudt)
- Blind signal separation (= mix van signalen (bv WiFi-signalen) die uit elkaar getrokken moeten worden)

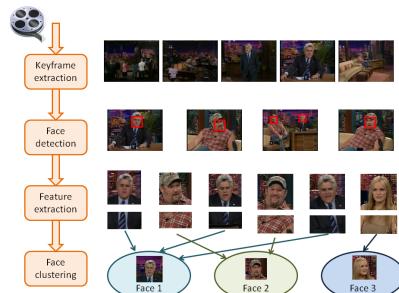
16.2 Clustering

Definitie 16.1 *Clustering is het groeperen van gegevens zodat binnen een bepaalde groep (=cluster) de gelijkenis groter is dan de gelijkenis met gegevens in een andere groep*

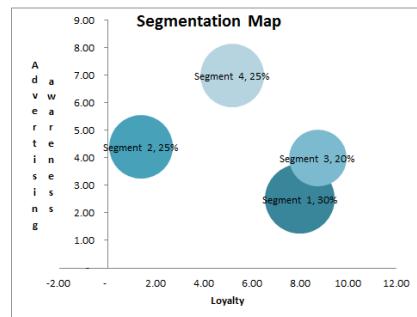


Figuur 120: Clustering: verdelen in groepen

16.2.1 Voorbeelden



Figuur 121: Voorbeeld: face clustering



Figuur 122: Voorbeeld: Marktsegmentatie



Figuur 123: Social media clustering



Figuur 124: Image segmentation

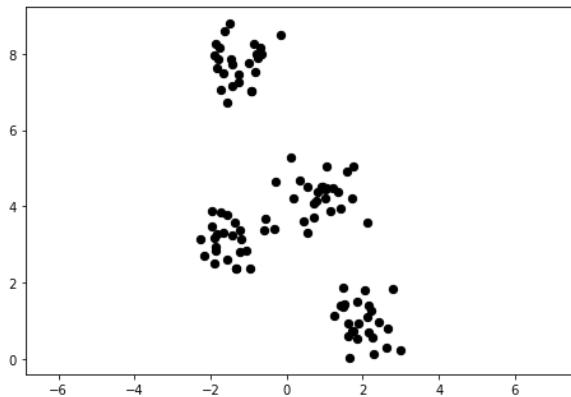
Andere voorbeelden:

- Gerelateerde resultaten bij een zoekopdracht van een zoekmachine tonen
- Gelijkwaardige artikels op nieuwsites

16.3 K-means clustering

K-means clustering is een zeer populaire clustering techniek door zijn eenvoud

- Simpel om te begrijpen
- Performant op grote dataset

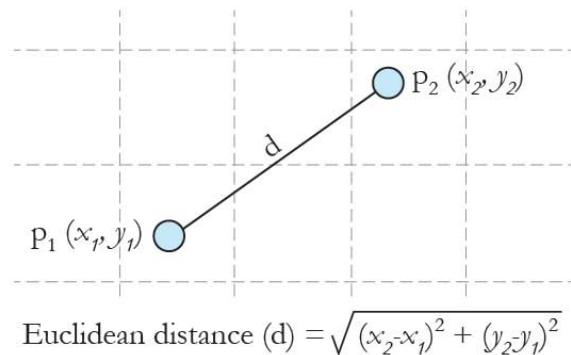


Figuur 125: Voorbeeld: zoek gelijksoortige clusters in deze data

16.3.1 Hoe gelijkenis uitdrukken?

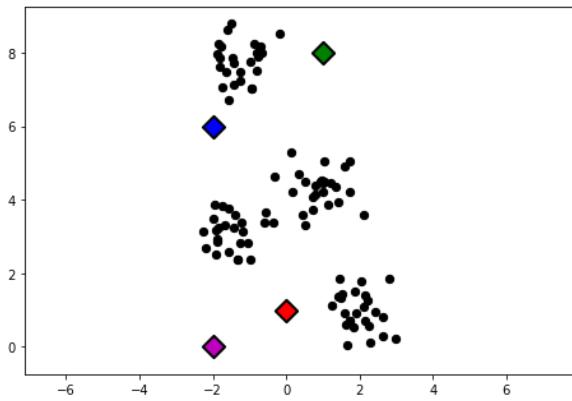
Via een afstandsfunctie:

- Hoe dicht liggen trainingssamples qua afstand van elkaar in de feature space?
- Veelgebruikte afstandsmaat is de **Euclidische afstand**
- Soms wordt ook de Manhattan Distance gebruikt: https://en.wikipedia.org/wiki/Taxicab_geometry
- Of de cosine distance: https://en.wikipedia.org/wiki/Cosine_similarity

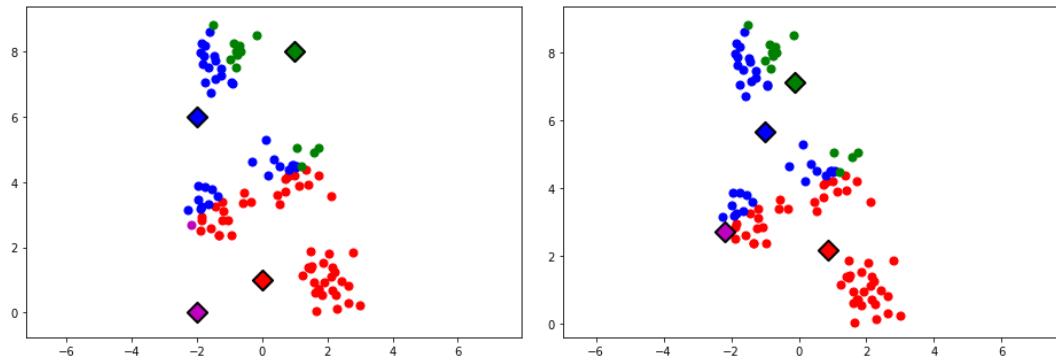


16.3.2 Werking

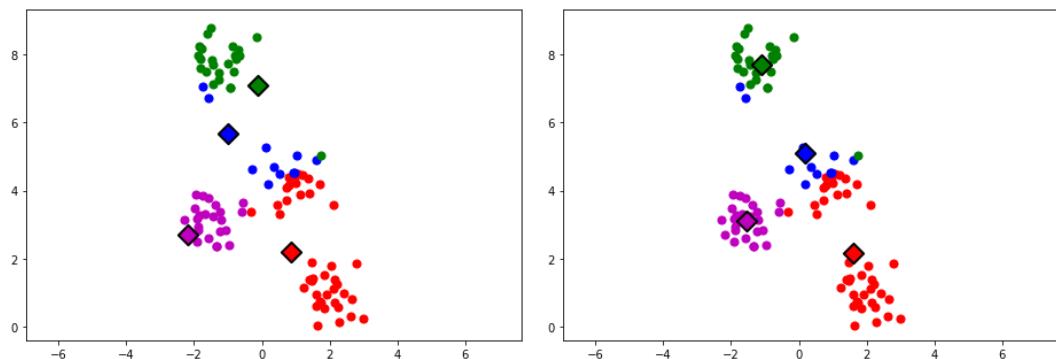
1. Initialiseer (al dan niet willekeurig) 4 **centoïden**
2. Zolang er datapunten veranderen van cluster:
 - (a) Wijs de datapunten toe aan de dichtsbij liggende centoïde
 - (b) Verplaats de centoïden naar het gemiddelde van de clusters



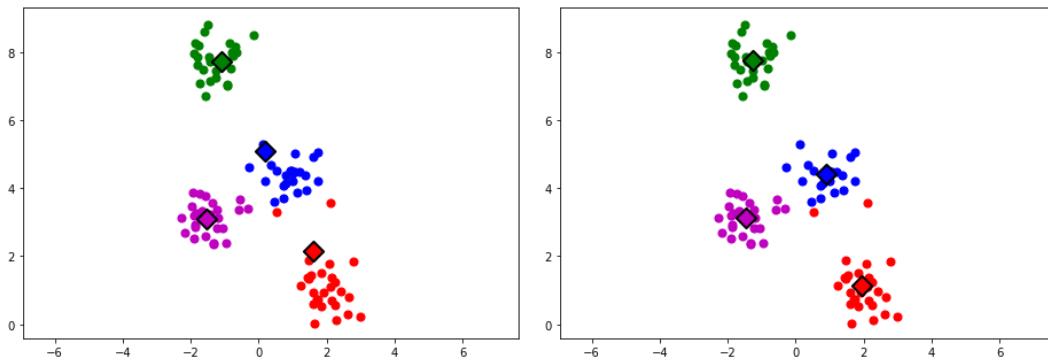
Figuur 126: Initialiseren van de centoïden



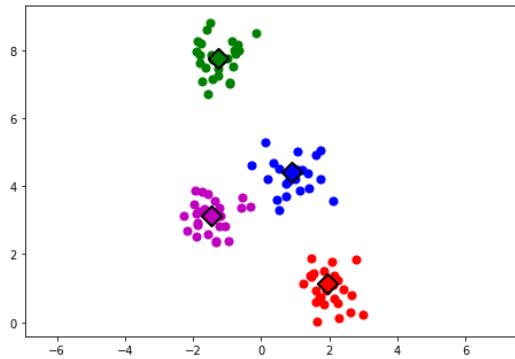
Figuur 127: Datapunten toewijzen aan dichtsbij liggende centoïde, dan de centoïden verplaatsen naar het gemiddelde van de clusters



Figuur 128: Opnieuw punten toewijzen en centoïden verplaatsen naar gemiddelde van de clusters



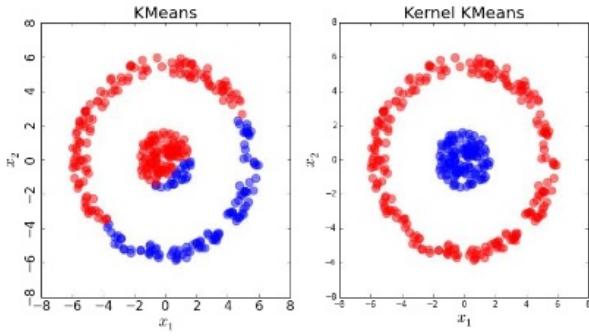
Figuur 129: Opnieuw punten toewijzen en centoïden verplaatsen naar gemiddelde van de clusters



Figuur 130: Wijs de datapunten opnieuw toe aan de dichtsbij liggende centoïde. We verplaatsen opnieuw de centoïden. Hier stoppen we omdat er geen datapunten meer van cluster veranderen.

16.3.3 Eigenschappen van K-means clustering

- Eenvoudig en resultaten gemakkelijk interpreerbaar
- K-means vindt niet altijd het globale optimum (=beste oplossing)
- Uiteindelijke resultaat hangt af van initialisatie van de centoïden
 - Probeer verschillende initialisaties
 - Initialiseer eerste centoïde op een willekeurig datapunt
 - Leg de tweede centoïde op een datapunt dat zover mogelijk van het eerste verwijderd is, de derde zo ver mogelijk van de twee eerste, enzoverder
- Zeer gevoelig aan uitschieters
- Problemen bij clusters met hetzelfde gemiddelde of niet-sferische clusters
 - Oplossing: kernel K-means of spectral clustering:



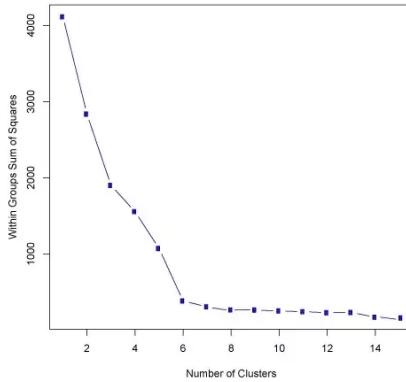
Figuur 131: K-means vs Kernel K-means

16.3.4 Strategie om het aantal clusters te bepalen

- Visuele inspectie van de datapunten (indien mogelijk)
- Silhouette clustering
- **Elbow method**

Definitie 16.2 Met de Elbow method kunnen we het aantal clusters vinden door voor een verschillend aantal clusters de ‘sum of squared error (SSE)’ te bepalen. De SSE is de som voor de gekwadrateerde afstanden tussen elk datapunt in een cluster en de centoïde van die cluster

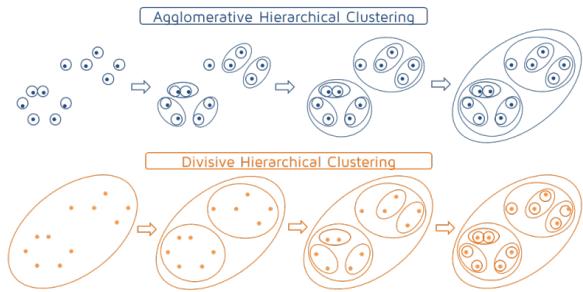
$$SSE = \sum_{i=1}^K \sum_{x \in c_i} dist(x, c_i)^2 \quad (24)$$



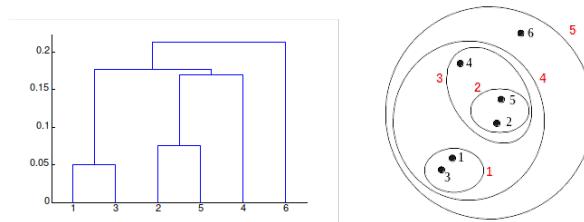
Figuur 132: Elbow method voor het vinden van het aantal clusters

Aantal clusters K is daar waar de fout niet noemenswaardig meer zakt. In het bovenstaande voorbeeld: $K = 6$

16.4 Hierarchical clustering



Figuur 133: Clusters kunnen worden voorgesteld via een **dendrogram**



Figuur 134: Dendrogram (= een soort boomstructuur-diagram)

- Je groepeert telkens 2 datapunten die het dichtst bij elkaar liggen
- Daarna groepeer je die clusters tot een cluster van clusters
- Nu kan je heel gemakkelijk het aantal clusters kiezen
- Het geeft ook een mooi inzicht over hoe de clusters tot stand zijn gekomen

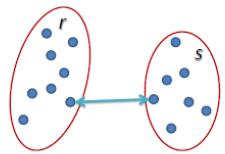
16.4.1 Sterktes van hierarchical clustering

- Geen vooropgesteld aantal clusters nodig. Elk gewenst aantal clusters kan bekomen worden door het dendrogram op de juiste plaats af te snijden
- De structuur van het dendrogram kan nuttig zijn. Bijvoorbeeld in biologie, productcategorieën, ...

16.4.2 Hoe ga je nu de afstand tussen 2 clusters bepalen?

3 manieren:

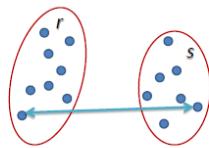
1. Met **single linkage**



$$L(r, s) = \min(D(x_{ri}, x_{sj}))$$

Figuur 135: Single linkage: je neemt de dichtste 2 punten

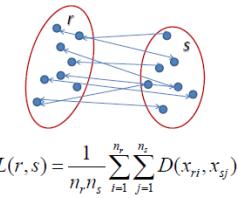
2. Met complete linkage



$$L(r, s) = \max(D(x_{ri}, x_{sj}))$$

Figuur 136: Complete linkage: je neemt de verste 2 punten

3. Met average linkage



$$L(r, s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

Figuur 137: Average linkage: je neemt de gemiddeldes van alle afstanden. Meer rekenwerk, maar minder gevoelig voor uitschieters maar wel een voorkeur voor globale clusters

16.4.3 Complexiteit van hierarchical clustering

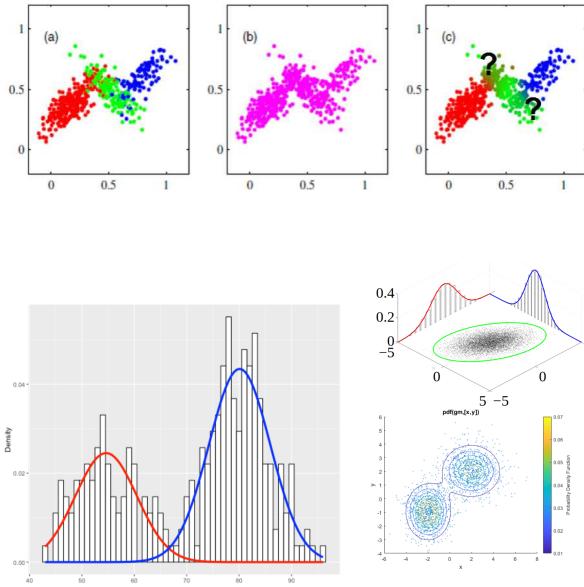
Zeer rekenintensief:

- Het aantal berekeningen is minstens kwadratisch afhankelijk van het aantal datapunten.
- Niet bruikbaar voor grote datasets.

16.5 Gaussian mixture models

Voorgaande modellen wijzen datapunten aan 1 enkele cluster

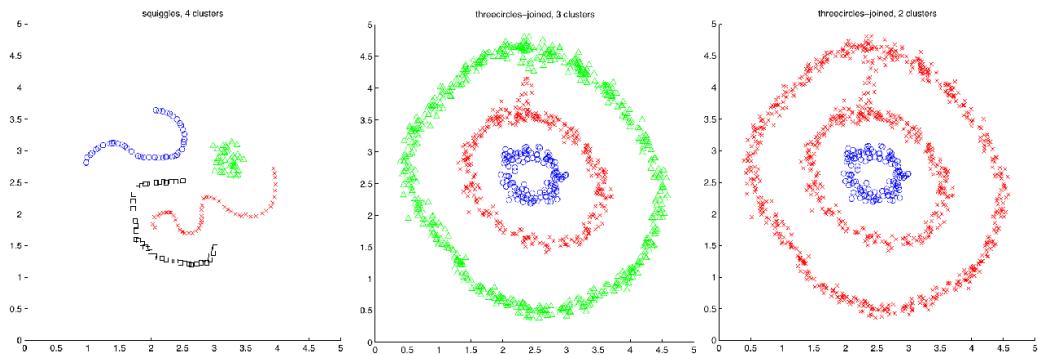
- Je probeert om Gaussiaanse curves te fitten op de data om zo clusters te vinden
- Je regelt de varianties en gemiddelden van die Gaussiaanse functies: je ‘verplaatst’ ze
- Je berekent zo een kans: er is x kans dat een punt tot een bepaalde cluster behoort



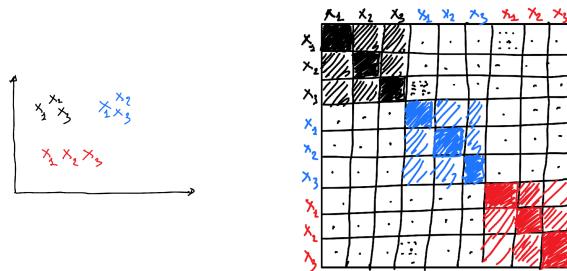
16.6 Spectral clustering

Hou rekening met de affiniteit in plaats van de absolute locatie.

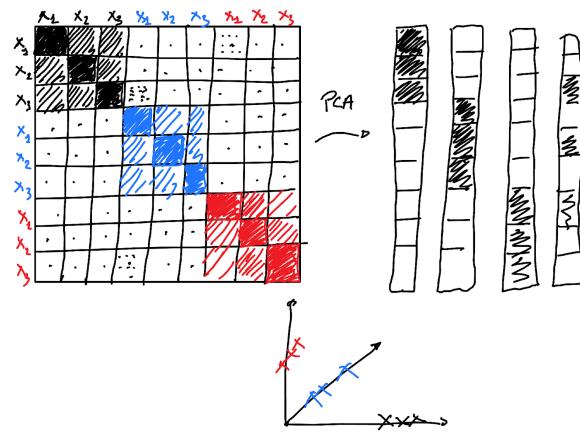
- Doet het vooral goed in specifieke gevallen: waar de cluster niet bepaald wordt door de densiteit, maar door de affiniteit
- Affiniteit = samenhang
- Spectral clustering zal werken met een affiniteitmatrix:
 - We starten met een bepaald punt
 - We kijken hoe dichtbij andere punten liggen
 - We noteren in een matrix voor elk punt hoe dichtbij elk ander punt ligt
- Handig voor datasets waar je patronen en specifieke lijnen hebt in de data, bijvoorbeeld image recognition



Figuur 138: Spectral clustering kan (zoals Kernel K-means) het ‘donut-probleem’ oplossen



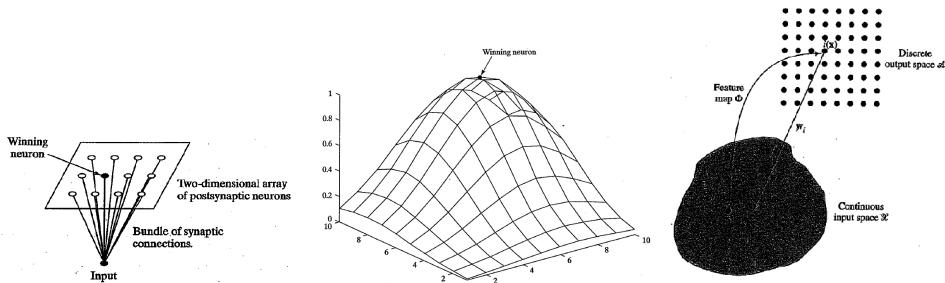
Figuur 139: Opstellen van de affiniteitmatrix



Figuur 140: Via Principle Component Analysis (PCA) bepalen we de ‘richting’ van de patronen

16.7 Self organizing maps

- Ook ‘Kohonen Maps’ genoemd (=genoemd naar een Finse professor)
- Competitie-gebaseerde clustering
- We spannen een net over (meestal) twee-dimensionale data
- Voor elk datapunt kijken we welk knooppunt van het net het dichtst bij ligt
- We verplaatsen elk datapunt naar dat dichtstbijzijnde knooppunt
- We doen dit herhaaldelijk \Rightarrow clustering
- Visualisatie:
https://en.wikipedia.org/wiki/Self-organizing_map#/media/File:TrainSOM.gif
https://en.wikipedia.org/wiki/Self-organizing_map



Figuur 141

17 Dimensionality reduction

17.1 Wat?

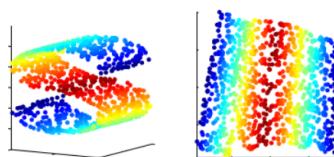
Binnen de context van machine learning (en statistiek) staat **dimensionality reduction** voor het proces waarbij het aantal variabelen (features in een dataset) gereduceerd wordt naar een kleinere set van 'principle variables'.

Meest gebruikte algoritmes voor dimensionality reduction:

- **Principle Component Analysis (PCA)** en Kernel PCA
- Linear Discriminant Analysis (LDA)
- Autoencoders
- Missing Values Ratio
- Low variance filter

17.2 Waarom?

- The curse of dimensionality
 - Wanneer het aantal feature zeer groot is, dan wordt het moeilijk om bepaalde algoritmes effectief te trainen
 - Bijvoorbeeld: clustering algoritmes
- Visualisaties in 2D of 3D \Rightarrow dimensie reduceren naar respectievelijk 2 of 3



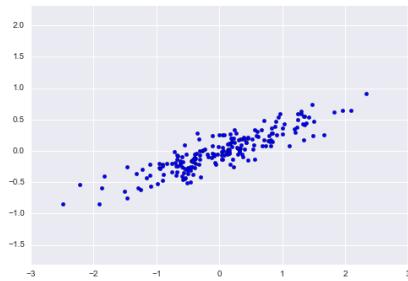
- Verwijderen van ruis
- Datacompressie

17.3 Toepassingen

- Document classification
- Classificatie van genen (veel genen, weinig samples)
- Gezichtsherkenning
- Spraakherkenning
- Handschriftherkenning
- Compressie
- Intrusion detection

17.4 Principle Component Analysis (PCA)

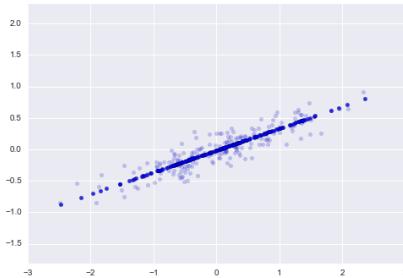
Principal component analysis (PCA) is een wiskundige techniek om het aantal (mogelijks) gecorreleerde features te transformeren (te combineren) naar een kleiner aantal niet-gecorreleerde features die de principle components worden genoemd.



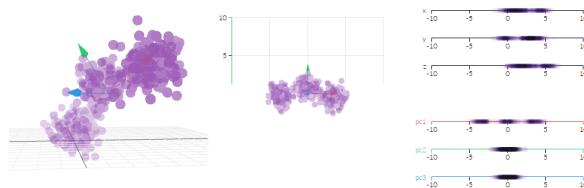
Figuur 142: Voorbeeld



Figuur 143: We identificeren de 2 principle components



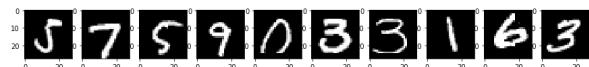
Figuur 144: We reduceren de data naar de grootste principle component



Figuur 145: <http://setosa.io/ev/principal-component-analysis/>

17.4.1 Voorbeeld aan de hand van de MNIST dataset

= dataset met zwart-wit afbeeldingen van getallen van 28x28 pixels



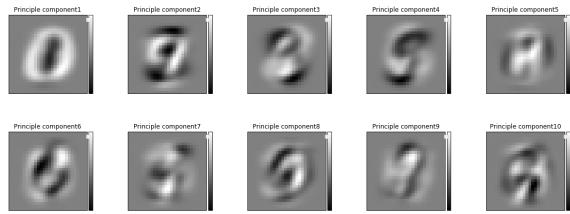
- De dimensie van 1 afbeelding (=aantal features) = $28 \cdot 28 = 784$
- We moeten zoeken naar manier om deze afbeeldingen in een lagere dimensie voor te stellen

In Sklearn:

```

1 # principle component analysis
2 from sklearn.decomposition import PCA
3
4 pca = PCA(n_components=10, svd_solver='full')
5 pca.fit(X_train)
6 # reduceren van de training- en testset
7 X_train_pca = pca.transform(X_train)
8 X_test_pca = pca.transform(X_test)
9
10 # elke afbeelding in zowel training- als de testset is gereduceerd tot 10 features
11 X_train_pca.shape
12 > (30000, 10)

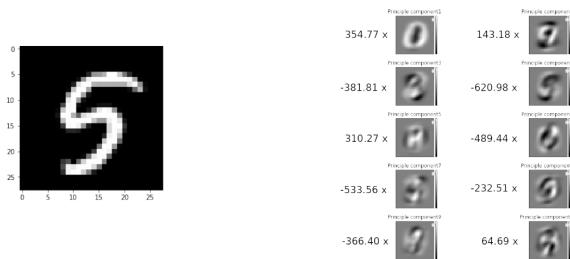
```



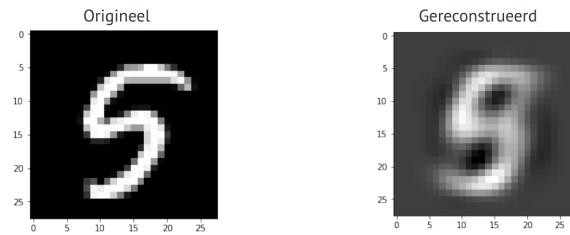
Figuur 146: Visualisatie van de eerste 10 principle components

Stel dat voor een bepaalde afbeelding we volgende PCA-scores berekend hebben:

1 354.77; 143.18; -381.81; -620.98; 310.27; -489.44; -533.56; -232.51; -366.40; 64.69



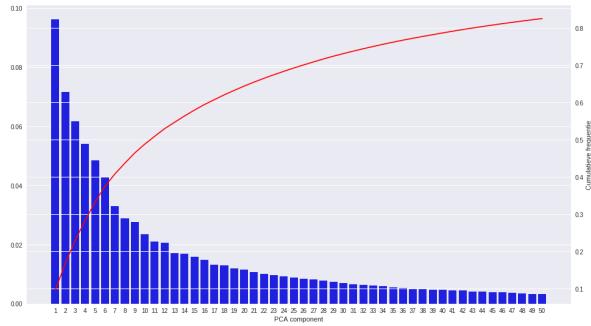
Figuur 147: PCA-scores van een bepaalde afbeelding. We vermenigvuldigen elke PC met zijn respectieve PCA-score



Figuur 148: Reconstructie van de digit aan de hand van 10 principle components (in plaats van de originele 784 features)

17.4.2 Kiezen van het aantal principle components: verklaarde variantie

PCA zoekt de componenten die een zo groot mogelijke variantie in de dataset verklaren. We willen zoveel mogelijk informatie overhouden na dimensionality reduction.



Figuur 149: Het belang van elke principle component: hoeveel data kan er door een PC verklaard worden

- Rode curve = cumulatieve waarde
- De eerste 16 components \Rightarrow goed voor 60% van de verklaarde variantie
- \Rightarrow 40% gaat verloren
- Makkelijk afleesbaar hoeveel PC's je nodig hebt voor bvb 90% verklaarde variantie

17.4.3 Face recognition

Bij face recognition noemt men de principle components 'Eigenfaces'



Figuur 150: We beschrijven een gezicht als een gewogen som van Eigenfaces