1. Het perceptron

Vooruitblik

In dit hoofdstuk maken we kennis met een van de meest eenvoudige machine learning methoden die lineair separabele datasets met nominale uitkomsten perfect kan classificeren: het perceptron. We formuleren het model waarmee instances van een klasselabel worden voorzien, en leiden eveneens een algoritme af dat in staat is de parameters van het model te fitten. Tenslotte generaliseren we deze methode naar instances met numerieke uitkomsten, waarna zal blijken dat we met een soortgelijk model eveneens lineaire regressie kunnen uitvoeren.

1.1. Inleiding

Machine learning algoritmen leren op grond van beschikbare data aangaande een aantal *attributen* om een gewenste uitkomst toe te kennen aan een onbekende *instance*. Hierbij kun je onderscheid maken tussen *nominale* en *numerieke* uitkomsten (d.w.z. categorische klasselabels danwel continue getalwaarden). Algoritmen kunnen daarnaast worden onderverdeeld in twee typen: *supervised* en *unsupervised*.

Supervised algoritmen worden getraind met instances die reeds voorzien zijn van de gewenste uitkomst, het zogenaamde *target*. Het doel van supervised algoritmen is om te leren hoe je deze gewenste uitkomst kan afleiden uit de waarden van de verschillende attributen. *Classificatie* en *regressie* zijn typische voorbeelden van supervised algoritmen met respectievelijk nominale en numerieke uitkomsten. In de *trainingsfase* worden door het algoritme een aantal *parameters* bepaald om tot een optimaal model te komen: het *concept* (bv. een beslisboom of een lineaire vergelijking). Daarna dient het algoritme het gefitte model in de *testfase* toe te passen om een zo goed mogelijke inschatting te geven van de uitkomst van een nieuwe set onbekende instances waarvan de juiste uitkomst niet tevoren gegeven is. Vaak wordt er ook nog een *validatiefase* tussengevoegd waarin de onderzoeker een aantal *hyperparameters* van het algoritme varieert om zo tot een optimaal model te komen (bv. de hoeveelheid pruning van de beslisboom of de graad van het polynoom in de vergelijking).

Unsupervised algoritmen krijgen een hoeveelheid data voorgeschoteld en moeten zelf een relevante structuur in de data zien te ontdekken. *Clustering* en *dimensiereductie* zijn typische categorieën van unsupervised algoritmen met respectievelijk nominale en numerieke uitkomsten. In het geval van clustering wordt aan elke instance een label toegekend behorende bij het cluster waartoe het instance op grond van diens attributen behoort; in het geval van dimensiereductie wordt aan elke instance een beperkt aantal getalm-

atige scores toegekend. Opnieuw leidt dit tot een geoptimaliseerd model, het concept (bijvoorbeeld de cluster centroïden in het geval van k-means clustering, of de loadings bij principale componenten analyse). Kenmerkend is dat unsupervised algoritmen noch tijdens de trainings-, de validatie-, of de testfase ooit te horen hoeven te krijgen wat een ware gewenste uitkomst is. Het nadeel wat hier tegenover staat is dat de uitkomsten van unsupervised algoritmen achteraf vaak lastig te interpreteren zijn.

Trainingsdata bestaan uit een (groot) aantal instances met attributen en een bijbehorende uitkomst: een nominaal klasselabel voor classificatie of een numerieke getalwaarde voor regressie. Gewoonlijk worden de attributen gerepresenteerd door de kolommen van een datatabel, waarbij één speciale kolom (meestal de laatste kolom) de te voorspellen uitkomsten bevat. De waarden van de attributen vormen input voor het algoritme en duiden we aan als $x_1, x_2, \ldots, x_d$; d geeft hierbij het aantal attributen aan, d.w.z. de *dimensionaliteit*. We zullen er in het vervolg van uit gaan dat alle attributen x_i numeriek worden gerepresenteerd als getalwaarden; nominale invoerwaarden zijn op zich wel mogelijk, maar zullen desgewenst in numerieke waarden worden omgezet. De bijbehorende te voorspellen nominale of numerieke uitkomst duiden we aan met y. Deze wordt toegekend door een *expert*, soms ook wel een *orakel* genoemd. Omdat het model in de praktijk niet perfect is, zal de voorspelde uitkomst, aangeduid met $\hat{y}$, kunnen afwijken van de werkelijke waarde, y. We gebruiken hier de gebruikelijke notatie uit de statistiek waarbij een dakje grootheden aanduidt die een schatting zijn van de echte waarde.

Het model kunnen we zien als een abstracte wiskundige functie f die afhankelijk van de attributen een voorspelling genereert: $\hat{y} = f(x_1, x_2, \ldots, x_d)$. De precieze vorm van die functie is bij aanvang onbekend, maar je stopt er de attributen van een instance in en krijgt er de voorspelde uitkomst uit. De *fout* die hierbij gemaakt wordt is dan gelijk aan $e = \hat{y} - y$; deze wordt ook wel de *afwijking*, het *residu* of de *error* genoemd. Het doel van machine learning is natuurlijk om het model zo te optimaliseren dat fouten zo min mogelijk voorkomen (bij classificatie) of zo klein mogelijk zijn (bij regressie). Hiertoe kunnen gewoonlijk een aantal parameters van het model worden gevarieerd; de aard hiervan hangt sterk af van het type model.

Een ideaal model is in principe in staat om elke denkbare vorm voor de functie f te fitten die het beste past bij de data, en vormt daarmee een *universele approximator*. In de praktijk dient een goede balans gevonden te worden tussen *underfitting* omdat het model niet flexibel genoeg is enerzijds, en *overfitting* omdat toevallige details in de trainingsdata geleerd worden die niet generaliseren naar nieuwe testdata anderzijds.

Neurale netwerken zijn voorbeelden van supervised machine learning. Ze worden getraind met voorbeelden die van een uitkomst zijn voorzien. Zoals we later zullen zien zijn neurale netwerken voldoende flexibel om nominale óf numerieke uitkomsten te kunnen produceren. Ze kunnen daarmee gebruikt worden om zowel classificatie- als regressieproblemen op te lossen. Met wat extra kunstgrepen die wij niet zullen verkennen kunnen ze zelfs unsupervised problemen aan. Het zijn dus zeer universeel toepasbare algoritmen.

Er is al werk gedaan aan neurale netwerken sinds halverwege de vorige eeuw. Een van de oorspronkelijke motivaties om onderzoek te doen naar neurale netwerken was om beter inzicht te krijgen in de werking van de hersenen. Een aantal van de vroegste algoritmen waren bedoeld als biologische modellen. Sommige moderne algoritmen zoals *spiking neu-*

ral networks hebben nog steeds een nauw verband met de werking van hersencellen. De meeste kunnen echter beter niet gezien worden als realistische modellen van hersenfunctie en zijn zo ook al lang niet meer bedoeld.

In de jaren negentig werden een aantal serieuze beperkingen van neurale netwerken duidelijk. Hoewel neurale netwerken potentieel zeer krachtig zijn, is het niet altijd eenvoudig om complexe modellen te trainen zodat ze ook goede prestaties behalen. Hiervoor is veel rekenkracht en veel beschikbare trainingsdata nodig. Met andere woorden, neurale netwerken kunnen met de juiste parameters de meest ingewikkelde voorspellingen correct maken, maar het bepalen wát die juiste parameters precies zouden moeten zijn bleek zeer moeilijk en soms onmogelijk. Het feit dat neurale netwerken vaak miljoenen parameters hebben die allemaal tegelijkertijd geöptimaliseerd moeten worden maakt dit probleem niet eenvoudiger. Deze beperkingen zorgden ervoor dat neurale netwerken rond de eeuwwisseling minder populair werden. In de laatste tien tot twintig jaar zijn echter nieuwe technieken ontwikkeld die het mogelijk maken om zogenaamde diepe neurale netwerken te trainen middels *deep learning*, hierbij geholpen door het beschikbaar komen van enorme datasets en krachtige hardware. Dit heeft ertoe geleid dat de huidige neurale netwerken vaak betere prestaties behalen dan vele andere algoritmen, en het regelmatig beter doen dan menselijke deskundigen, wat hun populariteit ten goede is gekomen.

Het doel van dit vak is om inzicht te krijgen in de werking van neurale netwerken, en als het ware een kijkje onder de motorkap te nemen. Daartoe gaan we o.a. van de grond af aan een eigen neuraal netwerk implementeren in Python. Hoewel we natuurlijk niet zullen komen tot een implementatie die zich qua efficiëntie en mogelijkheden kan meten met bestaande bibliotheken met kant-en-klare routines (zoals keras, tensorflow, theano, pytorch, of scikit-learn), zullen we eigen neurale netwerken kunnen opzetten die alle essentiële kenmerken in zich bergen.

Opgave 1. *

Maak een kruistabel met twee rijen voor "supervised" en "unsupervised" algoritmen, en twee kolommen voor "nominale" en "numerieke" uitkomsten. Vul elke cel van deze tabel met een voorbeeld van een bijpassend algoritme, beschrijf in woorden de vorm van hun concepten, en geef een mogelijke toepassing.

Opgave 2. *

Is overfitting het beste te onderscheiden van underfitting aan de hand van de resubstitution error (op de trainingsdata) of aan de hand van de cross-validation error (op de testdata)? Motiveer je antwoord.

Opgave 3. *

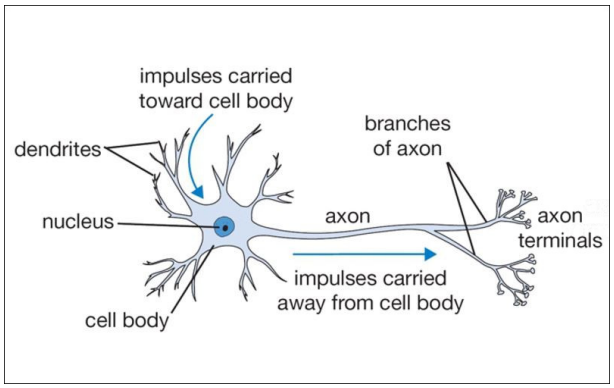
Naast supervised en unsupervised learning wordt vaak ook nog een categorie algoritmen onderscheiden onder de noemer *reinforcement learning*. Nog weer andere algoritmen heten *semi-supervised*. Wat wordt met deze termen bedoeld? Zoek hierover zonodig informatie op.

Opgave 4. **

Stel dat er onbeperkt veel trainingsdata beschikbaar zou zijn, zou het One-R algoritme dan gezien kunnen worden als een universele approximator? En hoe zit dat voor k-nearest neighbor, een (J48-)tree, Naive Bayes, of logistische regressie?

1.2. Het perceptron model

Het *perceptron* is in 1958 door Rosenblatt ontworpen als opvolger van het *McCulloch-Pitts neuron* en is de simpelste vorm van een neuraal netwerk. Dit zal door ons als uitgangspunt genomen worden om uiteindelijk een volledig neuraal netwerk op te bouwen. Om het te onderscheiden van multi-layer perceptrons (die we later zullen bekijken) wordt dit eerste eenvoudige model ook wel het *single-layer* perceptron genoemd. Een dergelijk perceptron bestaat uit één enkele rekeneenheid. Zoals een biologisch neuron typisch duizenden dendrieten heeft die invoer verzamelen vanuit andere neuronen, een cellichaam dat deze inputs combineert, en één axon dat vervolgens de uitkomst doorgeeft aan andere neuronen, zo kent ook het perceptron een meervoudig aantal inputs (de attributen $x_1, x_2, \ldots, x_d$), een bewerking die hierop toe wordt gepast (de functie f), en één uitkomst die de voorspelde klasse representeert (de waarde $\hat{y}$).



Het doel van het perceptron algoritme is om *binaire* (of *dichotome* of *binomiale*) classificatie uit te voeren op twee klassen. Het is gebruikelijk om een getalwaarde $y = -1$ te kiezen om de ene klasse aan te duiden en een waarde $y = +1$ voor de andere klasse. (Soms worden de waarden 0 en 1 gebruikt; de wiskundige formulering verandert dan licht maar de principes blijven hetzelfde.) Het perceptron begint ermee de attributen x_i te wegen met een serie gewichten w_i, en ze vervolgens op te tellen. Dit wordt een *lineaire combinatie* genoemd. Je verkrijgt hiermee een uitkomst

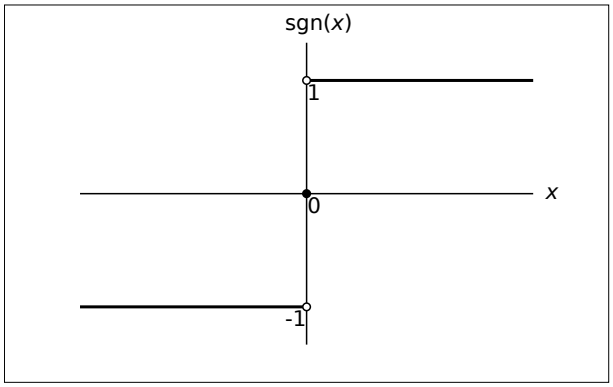
$$a = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_d \cdot x_d$$

Omdat zowel de attributen x_i als de gewichten w_i numerieke getalwaarden bevatten, is ook de uitkomst a numeriek. Aangezien het hier echter niet om regressie maar classificatie gaat, moet deze nog worden omgezet naar een voorspeld klasselabel $\hat{y}$. Het perceptron doet dit door het ene klasselabel $\hat{y} = -1$ toe te kennen als $a < 0$, of het andere klasselabel $\hat{y} = +1$ als $a > 0$. (Voor een uitkomst $a = 0$ die exact op de grens ligt kan een willekeurig

klasselabel worden toegekend, of wordt een uitkomst $\hat{y} = 0$ gekozen om aan te geven dat de voorspelling onzeker is; dit theoretische grensgeval is voor het perceptron echter nooit van praktische betekenis.)

De gewichten w_i geven aan wat de invloed van de waarde van een attribuut is op de uitkomst. Een gewicht nabij nul duidt erop dat het attribuut x_i niet of nauwelijks invloed heeft op de uitkomst en dus van weinig voorspellende waarde is. Een positief gewicht w_i geeft aan dat de waarde van het attribuut x_i typisch hoger is in de klasse $y = +1$ dan in de klasse $y = -1$; een negatief gewicht geeft het omgekeerde aan.

We kunnen het model summier neerschrijven door gebruik te maken van de *signum-functie* $\operatorname{sgn}(x)$ die een waarde ± 1 geeft (of eventueel 0) als het argument negatief of positief is (of eventueel nul).

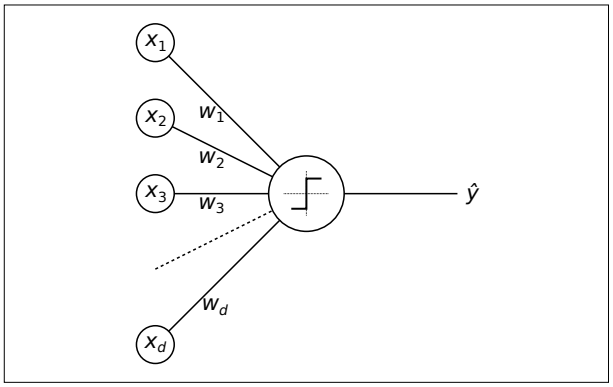


We krijgen dan de volgende formule voor het perceptron:

$$\hat{y} = \operatorname{sgn}\left(\sum_i w_i \cdot x_i\right)$$

We zullen later naast de signum-functie ook nog andere functies tegenkomen. In de context van neurale netwerken worden dit *activatiefuncties* genoemd. Dit grijpt weer terug op de analogie met een biologisch neuron waar het cellichaam bepaalt of het neuron actief wordt door een actiepotentiaal af te voeren langs het axon, of niet. De waarde a noemen we dan ook wel de *pre-activatiewaarde* (de waarde "voordat" de activatiefunctie wordt toegepast), en de waarde $\hat{y}$ is daarmee de *post-activatiewaarde* (de waarde "na-dat" de activatiefunctie wordt toegepast). Als we de activatiefunctie in het algemeen aanduiden met φ zijn de pre- en postactivatiewaarden aan elkaar gerelateerd volgens $\hat{y} = \varphi(a)$.

Het is vrij gebruikelijk om dergelijke formules en modellen te visualiseren in de vorm van een diagram. Hierin wordt een cirkel gebruikt om een neuron te symboliseren, vaak voorzien van een symbool dat de activatiefunctie weergeeft. Aan de linkerkant geeft een aantal inkomende lijnen of pijlen de inputs x_i vanuit alle attributen weer, elk voorzien van een indicatie van het gewicht w_i; aan de rechterkant verlaat één lijn of pijl de cirkel met de voorspelde uitkomst $\hat{y}$.



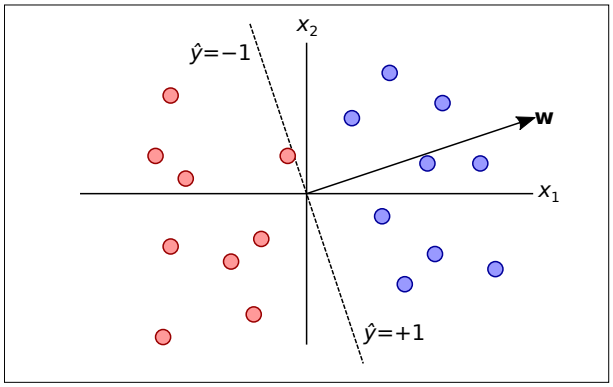
Een andere manier om de notatie te vereenvoudigen is door gebruik te maken van *vector* notatie. Vectoren worden vaak voorgesteld als pijlen met een zekere grootte en richting. Voor de discussie hier is het voldoende om een vector te zien als een reeks getalwaarden, vergelijkbaar met een *list* of *array* met een zekere vaste lengte. Zo kunnen we de attributen kortweg schrijven als een vector $\boldsymbol{x}$, waarbij $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$, en evenzo de bijbehorende gewichten $\boldsymbol{w}$ als $\boldsymbol{w} = [w_1, w_2, \ldots, w_d]$. De eerdere formule $a = \sum_i w_i \cdot x_i$ om de pre-activatiewaarde te berekenen kan dan worden vereenvoudigd tot

$$a = \boldsymbol{w} \cdot \boldsymbol{x}$$

Hierin staat de dot-notatie voor het *inproduct* van twee vectoren dat berekend wordt door de som te nemen van de producten van alle overeenkomstige elementen. Je kunt in de literatuur ook de matrixnotatie $a = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ voor kolomvectoren tegenkomen die op hetzelfde neerkomt. In het vervolg zullen wij de formules hier echter blijven uitschrijven met een sommatieteken.

Voor het perceptron ligt de grens tussen de twee klassen op die plek waarvoor de pre-activatiewaarde a gelijk is aan nul. Dat wil zeggen dat de grenslijn tussen de twee klassen wordt gedefinieerd door $w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_d \cdot x_d = 0$. Hierdoor is de grens tussen de twee klassen altijd een rechte lijn (of een recht vlak in drie dimensies, of een recht hypervlak in meerdere dimensies). Als de data daadwerkelijk op deze manier exact gescheiden kan worden dan noemen we deze data *lineair separabel*.

Voor het voorgestelde model is het niet moeilijk om in te zien dat deze rechte lijn precies door de oorsprong moet lopen. Immers, in de oorsprong zijn alle x_i aan nul. Als alle attributen gelijk zijn aan nul dan is ook elke gewogen combinatie van deze waarden gelijk aan nul, hetgeen $a = 0$ oplevert. Met andere woorden, de oorsprong van het coördinatenstelsel ligt precies op de scheidingslijn tussen de klassen. De richting van de scheidingslijn is op het eerste gezicht wat lastiger in te zien, maar kan worden begrepen door de gewichten w_i als een vector te bekijken. Het inproduct tussen twee vectoren is gelijk aan nul als de twee vectoren loodrecht op elkaar staan. Dat wil zeggen dat de scheidingslijn bestaat uit alle punten die ten opzichte van de oorsprong loodrecht staan op de vector $\boldsymbol{w}$.



Kortom, teken vanuit de oorsprong de vector $\boldsymbol{w}$, bepaal de hierop loodrechte lijn door de oorsprong, en je hebt de scheidingslijn volgens het model voor een gegeven set gewichten w_i gevonden. In de figuur hierboven is de grenslijn loodrecht op de vector $\boldsymbol{w}$ aangegeven met een stippellijn. Alle (blauwe) punten die aan de kant van de lijn liggen waar de vector $\boldsymbol{w}$ naartoe wijst krijgen het label $\hat{y} = +1$ toegewezen; de (rode) punten aan de andere kant van de lijn krijgen het label $\hat{y} = -1$.

Opgave 5. *
Teken een grafiek met daarin de negen punten $(\pm 2, \pm 1)$, $(\pm 1, \pm 2)$ en $(0, 0)$. Bepaal voor al deze punten tot welke klasse $\hat{y} = \pm 1$ deze punten behoren volgens een perceptron met gewichten $\boldsymbol{w} = [-3, 2]$. Schets in je figuur de gemodelleerde scheidingslijn tussen de twee klassen, en teken de vector $\boldsymbol{w}$.

Opgave 6. *
Stel we beschouwen een dataset met drie attributen x_1, x_2 en x_3. We passen hierop een perceptron toe met gewichten $\boldsymbol{w} = [1, -2, 3]$. Welk label wordt toegekend aan een instance A met $\boldsymbol{x}_A = [0, 1, 2]$? En een instance B met $\boldsymbol{x}_B = \left[\frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right]$? Bedenk zelf een voorbeeld van een instance C met attributen die precies op de grenslijn van dit perceptron liggen (anders dan $\boldsymbol{x}_C = [0, 0, 0]$).

Opgave 7. **
Is het waar dat voor alle getallen a en b geldt dat $\operatorname{sgn}(a + b) = \operatorname{sgn}(a) + \operatorname{sgn}(b)$? En is $\operatorname{sgn}(ab) = \operatorname{sgn}(a) \cdot \operatorname{sgn}(b)$? Geef telkens ofwel een voorbeeld waaruit blijkt dat het niet geldt, of leg uit waarom het wel geldt.

Opgave 8. **
Leg uit dat de signum-functie geschreven kan worden als $\operatorname{sgn}(x) = \frac{|x|}{x}$ voor $x \neq 0$, waarbij $|x|$ voor de absolute waarde van x staat.

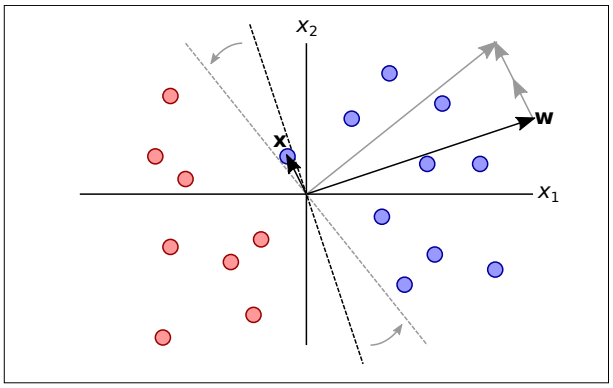
1.3. Optimalisatie

We weten nu dus hoe je de scheidingslijn kan bepalen voor een model met een zekere set gewichten, maar we weten nog niet hoe je geschikte waarden voor die gewichten kan

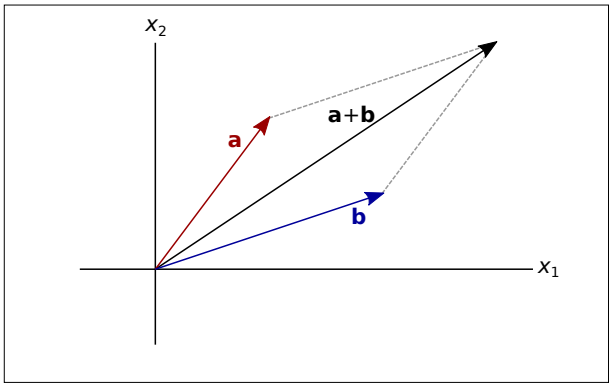
bepalen die de datapunten van twee lineair separabele klassen weten te scheiden. Het perceptron kent hiervoor gelukkig een buitengewoon eenvoudig recept.

Het idee is om voor alle instances één voor één te controleren of ze juist geclassificeerd worden. Als dat voor alle instances het geval is ben je klaar met de optimalisatie. Als er echter een instance gevonden wordt die verkeerd wordt geclassificeerd, dat wil zeggen het datapunt ligt aan de verkeerde kant van de scheidingslijn, dan wordt de scheidingslijn een stapje gedraaid in een zodanige richting dat het foutief geclassificeerde punt beter komt te liggen ten opzichte van de lijn.

Laten we een voorbeeld nemen van een punt $\boldsymbol{x}$ met als gewenst klasselabel $y = +1$, maar met als voorspelde uitkomst $\hat{y} = -1$. De error bedraagt dan $e = \hat{y} - y = (-1) - 1 = -2$. In de onderstaande figuur staat een voorbeeld van een dergelijke situatie geschetst. Er is één blauw punt $\boldsymbol{x}$ zichtbaar dat aan de verkeerde ("rode") kant van de lijn is komen te liggen.



Uit de figuur blijkt dat het wenselijk is om in dit geval de scheidingslijn tegen de klok in te roteren. Hiervoor is het noodzakelijk om ook de vector $\boldsymbol{w}$ tegen de klok in te roteren. Dit kunnen we bereiken door er een kleine vector bij op te tellen met een soortgelijke richting als de vector $\boldsymbol{x}$ zelf. Vectoren kun je optellen door alle overeenkomende elementen van de vector op te tellen (bv. $[1, 2] + [3, 1] = [4, 3]$). Of, in een plaatje kunnen we gebruik maken van de *kop-staart-regel*.



De precieze updateregel van het perceptron is iets algemener en luidt:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - (\hat{y} - y)\,\boldsymbol{x}$$

Door het meenemen van de fout $e = \hat{y} - y$ wordt de scheidingslijn de juiste kant op gedraaid. Voor het geval hierboven is $\hat{y} - y = -2$ negatief en wordt de vector $\boldsymbol{x}$ tweemaal bij de vector $\boldsymbol{w}$ opgeteld. Het resultaat is de in grijs afgebeelde gedraaide vector en scheidingslijn. In dit geval is het oorspronkelijk foutief geclassificeerde punt aan de juiste kant van de lijn komen te liggen. Dit hoeft overigens niet altijd meteen het geval te zijn, maar de lijn verschuift wel altijd in de goede richting. Merk tevens op dat wanneer het ene punt beter komt te liggen, het best zo kan zijn dat een ander punt juist slechter komt te liggen.

Wanneer een omgekeerde classificatiefout wordt gemaakt, dat wil zeggen aan een instance met klasselabel $y = -1$ wordt $\hat{y} = +1$ toegekend, dan moet de scheidingslijn andersom gedraaid en moeten de vectoren van elkaar worden afgetrokken. De factor $\hat{y} - y$ in de formule hierboven zorgt voor het correcte teken. Daarnaast zorgt deze factor er ook voor dat instances die reeds juist worden geclassificeerd geen effect meer hebben op de gewichten w_i. Immers, in dat geval is de fout $\hat{y} - y = 0$ en wordt de vector $\boldsymbol{w}$ bijgewerkt met de *nulvector*, hetgeen niet leidt tot een andere waarde.

Op deze manier kunnen één voor één alle n instances behandeld worden. Alle fout geclassificeerde instances leiden tot het bijwerken van de gewichten w_i. Zodra er geen verkeerd geclassificeerde instances meer zijn wordt het algoritme beëindigd en is een optimale classifier gevonden. Mogelijk zijn hier meerdere gangen door alle instances voor nodig, genaamd *epochs*.

Resteert nog de vraag hoe je de waarde van $\boldsymbol{w}$ het beste kan initialiseren. Aannemende dat van tevoren niet bekend is in welke richting de scheidingslijn ongeveer zou moeten lopen is elke richting voor de vector $\boldsymbol{w}$ even geschikt. Het is daarom gebruikelijk om aanvankelijk eenvoudigweg alle $w_i = 0$ te kiezen.

Opgave 9. *

Maak zelf een soortgelijke schets als de figuur hierboven, maar dan voor een instance met als gewenste klasselabel $y = -1$ en voorspelde uitkomst $\hat{y} = +1$. Ga na dat ook hiervoor de updateregel van het perceptron de vector $\boldsymbol{w}$ aanpast in de juiste richting.

Opgave 10. *

Stel, een perceptron met gewichten $\boldsymbol{w} = [1, -4, 5]$ wordt toegepast op een instance $\boldsymbol{x} = \left[0, \frac{1}{2}, 1\right]$ en met het ware label $y = -1$. Wat worden de nieuwe gewichten nadat de update regel één keer is toegepast? Herhaal de updateregel totdat het perceptron deze instance correct classificeert: wat zijn de uiteindelijke gewichten $\boldsymbol{w}$?

Opgave 11. **

Als in plaats van labels $y = \pm 1$ de waarden $y = 0$ en $y = +1$ gebruikt zouden worden, samen met een model met een activatiefunctie die eveneens leidt tot voorspellingen $\hat{y} = 0$ of $\hat{y} = +1$ (de heaviside-stapfunctie), zou dan de updateregel $\boldsymbol{w} \leftarrow \boldsymbol{w} - (\hat{y} - y)\,\boldsymbol{x}$ nog correct blijven werken, of moet deze veranderd worden?

1.4. Bias

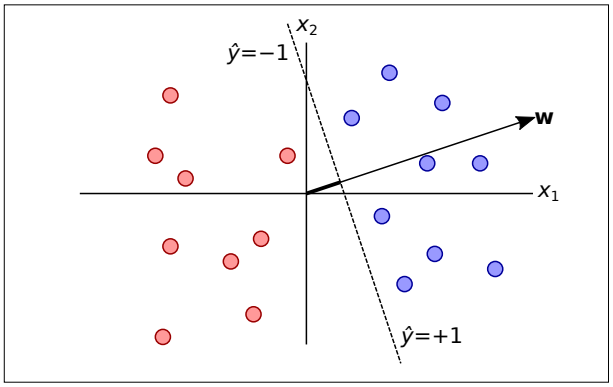
Tot dusverre hebben we als eis gesteld dat de data lineair separabel moeten zijn middels een rechte lijn door de oorsprong. In de praktijk is dat een onrealistisch zware eis. Het is echter vrij eenvoudig om dit te veralgemeniseren tot een rechte lijn die niet per se door de oorsprong hoeft te gaan. We kunnen dit bereiken door aan de pre-activatiewaarde een extra *bias* b toe te voegen, zodat we krijgen

$$a = b + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_d \cdot x_d$$

De classificatieregel wordt dan:

$$\hat{y} = \operatorname{sgn}\left(b + \sum_i w_i \cdot x_i\right)$$

Het is nu niet moeilijk in te zien dat de oorsprong waar alle attributen de waarde nul hebben niet langer op de scheidingslijn hoeft te liggen. In dat geval is immers alle $x_i = 0$ en wordt $\hat{y} = \operatorname{sgn}(b)$, oftewel de oorsprong wordt hetzelfde label toegekend als het teken van de bias b. De bias geeft aan hoe sterk het model een voorkeur heeft om een bepaald klasselabel toe te kennen als er verder geen inputs x_i zouden hoeven te worden meegewogen. Als de bias $b > 0$, dan heeft het model een ingebouwde voorkeur voor het label $\hat{y} = +1$; voor $b < 0$ neigt het eerder naar $\hat{y} = -1$. Hoe groter de absolute waarde van b, hoe verder de scheidingslijn van de oorsprong komt te liggen.

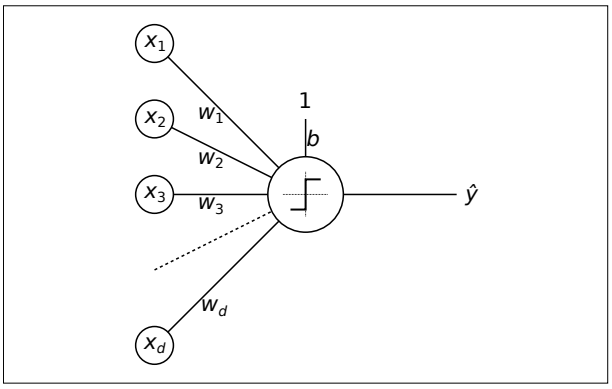


Hiermee is nog niet gezegd hoe de gewichten w_i en de bias b geoptimaliseerd kunnen worden aan de hand van een concrete lineair separabele dataset. Echter, we kunnen een slimme truc toepassen om de bovenstaande formule te reduceren tot het simpelere geval zonder bias. Beschouw hiertoe de bias b als onderdeel van de te optimaliseren gewichten w_i: laten we voor het gemak een tijdelijk extra gewicht w_0 toevoegen dat we identificeren met b.

$$a = \underbrace{w_0}_{=b} \cdot \underbrace{x_0}_{=1} + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_d \cdot x_d$$

In de formule wordt w_0 vermenigvuldigd met x_0. Als we elke instance een denkbeeldig attribuut $x_0 = 1$ meegeven, en de sommatie laten lopen vanaf $i = 0$, dan is het effect

dat er een extra term $w_0 \cdot x_0$ wordt opgeteld bestaande uit een gewicht dat altijd wordt vermenigvuldigd met 1. Dit heeft natuurlijk exact dezelfde uitwerking als het optellen van een constante bias, als we w_0 terug hernoemen naar b. In het diagram wordt dit soms expliciet aangegeven met een extra input met waarde 1 en gewicht b.

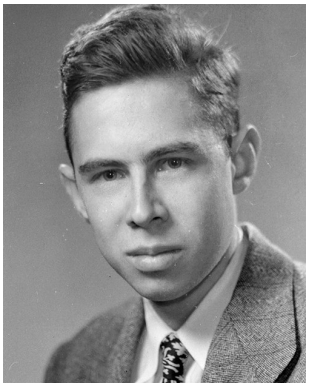


We kunnen precies dezelfde updateregel toepassen op het gewicht w_0 als hierboven op alle andere gewichten werd toegepast. Wanneer we ons realiseren dat x_0 altijd gelijk is aan 1, dan levert dit op $b \leftarrow b - (\hat{y} - y) \cdot 1$, oftewel samengevat:

$$\begin{cases} b \leftarrow b - (\hat{y} - y) \\ w_i \leftarrow w_i - (\hat{y} - y)\, x_i \end{cases}$$

Het algoritme is voltooid zodra alle instances juist worden geclassificeerd. Er worden daarna immers geen updates meer uitgevoerd.

Rosenblatt bewees rigoreus dat wanneer de data twee lineair separabele klassen vormen, het perceptron algoritme altijd convergeert en een scheidingslijn bepaalt die de klassen perfect van elkaar onderscheidt!



Opgave 12. *

Teken een grafiek met daarin de negen punten $(\pm 2, \pm 1)$, $(\pm 1, \pm 2)$ en $(0, 0)$, en bepaal tot welke klasse deze punten behoren volgens een perceptron met gewichten $\boldsymbol{w} = [-3, 2]$ en bias $b = -4$. Schets in je figuur de gemodelleerde scheidingslijn tussen de twee klassen, samen met de vector $\boldsymbol{w}$.

Opgave 13. *
Is het teken van de bias b in de eerste figuur van deze paragraaf hierboven positief of negatief? Leg je antwoord uit.

Opgave 14. **
Gegeven is het onderstaande datasetje met twee numerieke attributen x_1 en x_2 en één klasselabel $y = \pm 1$. Maak een schets van de ligging van deze datapunten in een coördinatenstelsel met twee assen. Is deze dataset lineair separabel? Voer handmatig het perceptron algoritme uit, geïnitialiseerd met $b = w_1 = w_2 = 0$, beginnend met het eerste datapunt, net zo lang door de opeenvolgende datapunten heen roulerend tot alle instances juist geclassificeerd worden. Hoe ligt de uiteindelijke scheidingslijn?

x_1	x_2	y
1	0	+1
0	1	+1
−1	0	−1
0	−1	−1

Opgave 15. **
Gegeven is het onderstaande datasetje met twee numerieke attributen x_1 en x_2 en één klasselabel $y = \pm 1$. Maak een schets van de ligging van deze datapunten in een coördinatenstelsel met twee assen. Is deze dataset lineair separabel? Voeg nu een extra attribuut toe dat je berekent uit de bestaande attributen volgens $x_3 = x_1 \cdot x_2$. Zal het perceptron algoritme convergeren als je het toepast op deze uitgebreide dataset?

x_1	x_2	y
1	1	+1
1	−1	−1
−1	1	−1
−1	−1	+1

1.5. Lineaire regressie

Met een paar aanpassingen kan het perceptron ook worden ingezet om lineaire regressie uit te voeren. Om te beginnen zal de uitkomst niet moeten worden omgezet naar waarden $\hat{y} = \pm 1$, maar dienen numerieke getalwaarden te worden geproduceerd. Dit kan eenvoudig worden bereikt door de signum-functie te verwijderen. We krijgen dan:

$$\hat{y} = b + \sum_i w_i \cdot x_i$$

Dit is eigenlijk equivalent aan het toepassen van een activatiefunctie $\varphi(a) = a$, dat wil zeggen de *identiteitsfunctie*, aangezien deze de post-activatiewaarde gewoon gelijk houdt aan de pre-activatiewaarde. Het resultaat is het hopelijk bekende (multi-)lineaire regressie model.

De updateregel blijkt ook vrijwel dezelfde vorm te krijgen. We moeten echter opletten dat voor het perceptron eigenlijk alleen de richting van de vector $\boldsymbol{w}$ relevant is. De

signum-functie kijkt namelijk niet naar de grootte van de pre-activatiewaarde a, alleen naar het teken. Nu de signum-functie is verwijderd gaat de grootte van $\boldsymbol{w}$ wel een belangrijke rol spelen. Immers, hoe groter de gewichten, hoe groter ook de uitkomsten. Het gevolg hiervan is dat we ook de grootte van de updates zorgvuldiger moeten kiezen om tot een goede oplossing te komen. We doen dit door een extra coëfficient α te introduceren, de *learning rate*. Hoe groter deze factor, hoe groter de updates van de waarden w_i en b. De updateregel voor lineaire regressie luidt:

$$\begin{cases} b \leftarrow b - \alpha\,(\hat{y} - y) \\ w_i \leftarrow w_i - \alpha\,(\hat{y} - y)\,x_i \end{cases}$$

Deze is nagenoeg identiek aan die van het perceptron. Merk wel op dat in dit geval zowel $\hat{y}$ als y numerieke getalwaarden zijn en de fout $e = \hat{y} - y$ daardoor allerlei mogelijke waarden kan aannemen. Omgekeerd kun je bovenstaande updateregel prima op het perceptron toepassen; het ligt voor de hand om dan $\alpha = 1$ te kiezen, hoewel je kan aantonen dat de waarde van α er voor het perceptron eigenlijk niet toe doet.

Als je de waarde van α bij lineaire regressie te klein neemt, dan zijn er een heleboel iteraties nodig om tot een optimale oplossing te komen omdat het model telkens maar een klein beetje wordt aangepast. Kies je α echter te groot, dan is het mogelijk dat je de vergelijking telkens te sterk aanpast, over de juiste oplossing heenschiet, en daardoor nooit convergeert. In complexere neurale netwerken wordt α vaak rond de 0.001 of 0.01 gekozen. Dit kan afhangen van de dataset en van het model en is veelal een kwestie van experimenteren: als het algoritme langzamer convergeert dan je zou verwachten heeft het soms zin om α te vergroten; als de parameters wild heen en weer springen en helemaal niet convergeren is dat een teken dat α wel eens te groot gekozen zou kunnen zijn. We zullen in latere hoofdstukken *adaptive learning* heuristieken bekijken die in staat zijn om de waarde van α automatisch te kiezen en gaandeweg aan te passen. Voor nu laten we het probleem van het kiezen van een geschikte learning rate aan de gebruiker van het perceptron om op te lossen.

Een belangrijk verschil tussen het perceptron en lineaire regressie is dat het perceptron voor lineair separabele data gegarandeerd in een eindig aantal stappen convergeert naar een optimaal model dat alle trainingsinstances correct classificeert. Voor lineaire regressie is dat meestal niet het geval: hierbij zie je dat de oplossing geleidelijk steeds beter wordt, maar de fout typisch nooit helemaal gelijk aan nul wordt.

Overigens bestaan er voor lineaire regressie exacte (matrix-)formules die in één keer de juiste oplossing geven. De hier behandelde numerieke oplossingsmethode is dus niet per se het meest praktische model, maar maakt wel de elegante parallel met het perceptron duidelijk en toont aan dat dit soort modellen zowel voor classificatie als voor regressie kunnen worden ingezet.

Opgave 16. *

Teken een grafiek met daarin de negen punten $(\pm 2, \pm 1)$, $(\pm 1, \pm 2)$ en $(0, 0)$, en bepaal de voorspellingen die aan deze punten worden toegekend door een lineair regressiemodel met gewichten $\boldsymbol{w} = [-3, 2]$ en bias $b = -4$.

Opgave 17. **

Gegeven is het onderstaande datasetje met slechts één numeriek attribuut x_1 en één numerieke uitkomst y. Maak een schets van de ligging van deze datapunten in een grafiek. Voer handmatig het lineaire regressie algoritme uit met learning rate $\alpha = \frac{1}{2}$, geïnitialiseerd met $b = w_1 = 0$, beginnend met het eerste datapunt, en stop na maximaal drie epochs. Hoe ligt de uiteindelijk gefitte lijn?

x_1	y
-1	-1
1	3

Opgave 18. **

Herhaal de bovenstaande opgave met learning rate $\alpha = 1$; wat valt je op?

Opgave 19. ***

Leg uit waarom het voor het perceptron eigenlijk niet uitmaakt hoe groot je de learning rate α kiest als je de updateregel van lineaire regressie zou toepassen.