Methoden en Technieken: Supervised Machine Learning

Hogeschool van Amsterdam - Master Applied Artificial Intelligence

Kristo Wind

Amsterdam
November 2022

Methoden en Technieken: Supervised ML

Contents

Data-analyse stappenplan

1. Dataset altijd eerst ruwe data, zoals fotos of punten op assenstelsel. Eerst gevoel krijgen voor de data.
2. Hoeveel verschillende soorten data. Mist er data?
3. Willekeurige plots maken.
4. logaritme van inkomen vertalen het beter.
5. Afmetingen bekijken.
6. Min-max waarden.
7. Aantal fotos per persoon op volgorde (df.value_counts(subset='Name')).
8. Histogram plotten (plt.hist(counts[:,1], bins=30)).
9. Voor balans in data, augmenteren.

Week 1

Aandachtspunten:

- Leerdoelen 7, 8, 9, 12
- $E(\bar{X}) = \mu$
- $Y = f(X) + \epsilon$
- Flexibiliteit van een model
- Statistical Learning hoofdstuk 2.
  - prediction & inference (regressie)
  - parametric & non-parametric methods (regressie)
  - The Bayes Classifier (classificatie)

1 Basiskennis statistiek en kansrekening

$$E(\bar{X}) = \mu$$

$\bar{X}$ = steekproefgemiddelde (unbiased estimator)
μ = populatiegemiddelde
E = verwachtingswaarde

Waarom statistiek en kansrekening?

- Wat is het percentage minderjarige bezoekers per dag?
- Is dat elke dag hetzelfde?
- Wat is het gemiddelde percentage minderjarige bezoekers per dag?
- Wat is, gemeten over een maand, het gemiddelde percentage minderjarige bezoekers per dag?
- Wat zegt dit over het werkelijke gemiddelde percentage minderjarige bezoekers per dag?
- Wat zegt dit over het werkelijke percentage minderjarige bezoekers op een dag?

Voorbeeld ML-vraagstuk (regressie)
Het bepalen van de leeftijd van de klant adhv de variabele gewicht komt wiskundig neer op het bepalen van de f in de volgende vergelijking:

$$Y = f(X) + \epsilon$$

Met Y de leeftijd van de klant, X het gewicht van de klant en ϵ de fout-term.
f: de informatie in X over Y (deterministisch: temaken met een kans)
ϵ: dat wat we niet kunnen weten over Y, gegeven X (stochastisch). Het verschil voorspelde leeftijd en echte leeftijd

Leren: het schatten van functie f. Data gebruiken is essentieel. Als we f op een andere manier zouden bepalen (bijv. uit theoretische overwegingen) zal het niet leren worden genoemd.

Diverse ontwerpkeuzes die behandelt dienen te worden. Onderstaande bepaald adhv de opdracht en is de ontwerpvraag.

- Classificatie of regressie?
- Welke foutmaat? Wat wil je minimaliseren?
- Voorspellen of samenhang? Blackbox al goed of wil je weten hoe?

Onderstaand wordt het model genoemd (hypothesis set en algoritme samen)

- Welke functies om f te schatten (hypothesis set)?
- Welk algoritme om de beste $\hat{f}$ te vinden?

2 Flexibiliteit van het model (om f proberen te schatten)

Kans is aanwezig dat f niet lineair van X afhangt en dat de foutmaat kleiner wordt als we een flexibeler model kiezen (hypotheseruimte groter maken). Hoe meer termen, hoe meer kans op overfitting. $\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X$ is lineair bijvoorbeeld. De onderstaande polynoom niet.

$$\hat{f}X) = \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 + \hat{\beta}_4 X^4$$

2.1 De bias-variance tradeoff

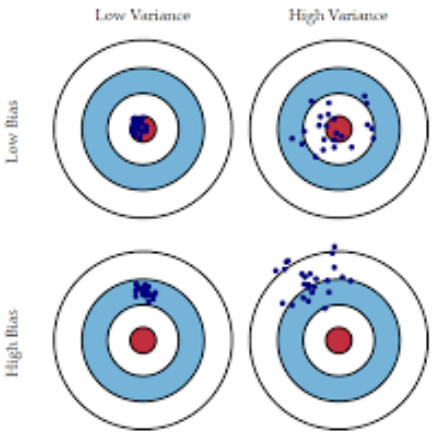


Figure 1: bias-variance

Als data wordt weergegeven door:

$$(x_1, y_1), (x_2, y_2), .., (x_n, y_n)$$

dan wordt de foutmaat gegeven door de mean squared error (MSE):

$$MSE = E[(Y - \hat{f}(x_0))^2] = Var(\hat{f}(x_0)) + (f(x_0) - E[\hat{f}(x_0))^2 + Var(\epsilon) = Var(\hat{f}(x_0)) + B(\hat{f}(x_0))^2 + Var(\epsilon)$$

Waarbij:
1e term: de spreiding van de steekproef (hoe groot is het wolkje?)
2e term: als 0 = geen gemiddelde afwijking = geen bias (hoeveel zit je gemiddeld af van de roos (μ)
3e term: de spreiding van de error-term
Met aannames:

1. $Var(X) = E(X^2) - E(X)^2$
2. $E(\epsilon) = 0$
3. $Cov(X, \epsilon) = 0$

3 Statistical Learning

inputs $(X_1, X_2, ..., X_p)$ = predictors, independent variables, features, variables
output(Y) = response, dependent variable

Prediction: When function f is not neccessary to determine (blackbox).

- *predict the response using predictors.*

Inference: Need to know function f.

- *Which media are associated with sales?*
- *Which media generate the biggest boost in sales?*
- *To what extent is the product's price associated with sales?*

Estimate f:

- Parametric methods
- Non-parametric methods

Parametric methods (inflexible):

1. Make an assumption about the functional form of shape of f (linear).
2. Need an approach to fit the model.

Answers:

1. $Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$.
2. (ordinary) least squares is most common.

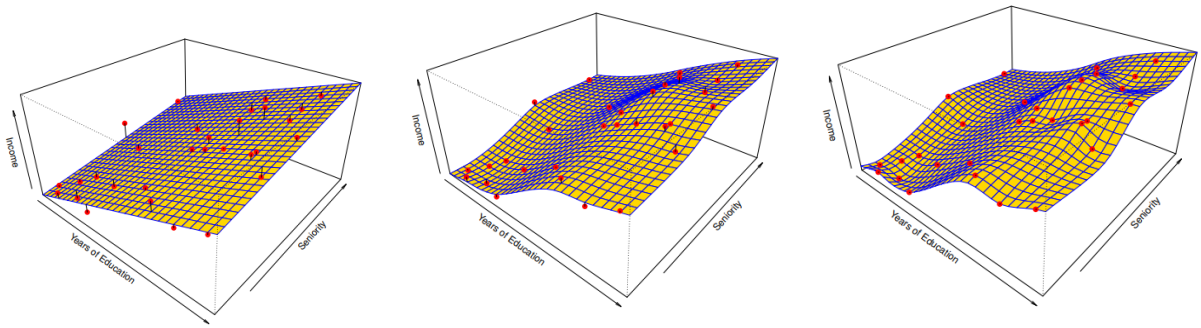
3.1 Non-parametrische methoden (flexibel)

Non-parametric methods do not make explicit assumptions about the functional form of f. Instead they seek an estimate of f that gets as close to the data point as possible without being too rough of wiggly.
Regressie: kwantitatief
Classificatie: kwalitatief

Cross-validatie (zie week 4): estimation method for estimating test MSE using the training data (MSE = optimal combination of bias & variance).
Good performance: low squared bias & low variance.

4 Classificatie

- The Bayes Classifier
- KNN



(a) Linear (parametric) (b) Flexible (non-parametric) (c) Too flexible (overfitting)

4.1 The Bayes Classifier

The line where $Pr(Y = j|X = x_0)$ is exactly 0.5, with error $1 - E(\max_j Pr(Y = j|X))$.

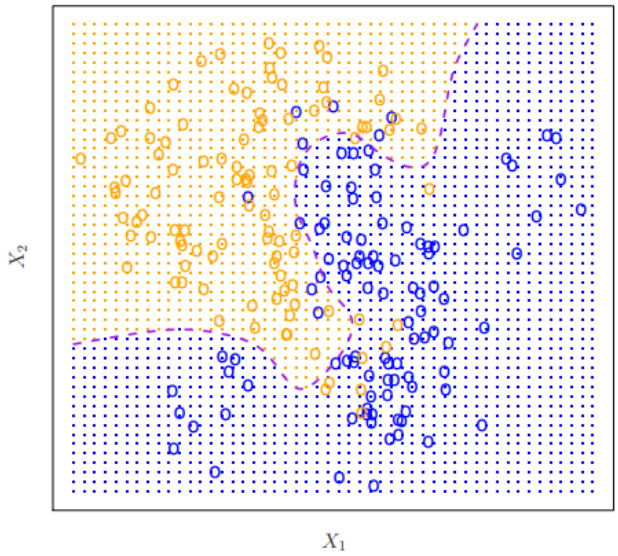


Figure 3: Bayes Classifier

Week 2

Aandachtspunten:

- KNN (classificatie)
- Learning from Data
- Regularisatie (l_1 en l_2)

4.2 K-Nearest Neighbors (KNN)

The line where $Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j)$.
$\mathcal{N}_0$ represents the training data closest to x_0.

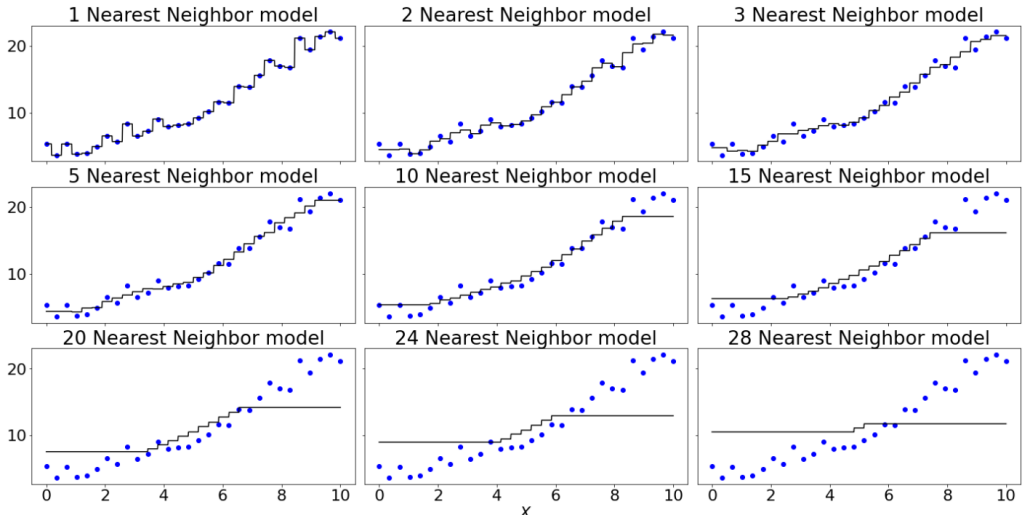


Figure 4: K-Nearest Neighbors algorithm

expected test-MSE $= E[Y - g(x_0))^2] = Var(Y) + (E(Y) - g(x_0))^2 = Var(\epsilon) + (f(x_0) - g(x_0))^2$

.

$$Var(X) = E(X^2) - E(X)^2$$
$$E(X^2) = Var(X) + E(X)^2$$
$$E((Y - g(x_0))^2) = Var(Y - g(x_0)) + E(Y - g(x_0))^2 =$$
$$= Var(Y) + (E(Y) - g(x_0))^2 = Var(\epsilon) + (E(Y) - g(x_0))^2$$

. Waarbij $g^*(x_0)$ is het minimum en wordt gegeven door $g^*(x_0) = f(x_0) = E(Y|X = x_0)$.

$Var(g(x_0))$ is 0, want is een vaste waarde en $E(g(x_0)) = g(x_0)$ want verwachtingswaarde van vaste waarde is de vaste waarde.
$g(x_0) = E(Y|X = x_0)$ geeft bijvoorbeeld voor iemand van 80kg een gemiddelde van de k mensen die het dicht bij de 80kg liggen, dus KNN benadering.

4.3 Foutmaten

Regressie:

- MSE (mean squared error)

- MAE (mean absolute error): de optimale $f(X)$ gegeven worden door de voorwaardelijke mediaan van Y gegeven X.

Classificatie:

- Accuracy
- Bayes classifier

4.3.1 Accuracy

Train error rate= $\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$ met:

$$I(y_i \neq \hat{y}_i) = \begin{cases} 1, & \text{if } \ y_i \neq \hat{y}_i \\ 0, & \text{if } \ y_i = \hat{y}_i \end{cases}$$

4.3.2 Bayes classifier

De functie $f^*(x_0)$ met laagste test error rate is de Bayes classifier gegeven door: $f^*(x_0) = C_j$ met $P(Y = C_j|X = x_0) = max_i(P(Y = C_i|X = x_0))$. De klasse C_i met de hoogste kans $(max_i(P(Y = C_i|X = x_0)))$ als x_0 gegeven, wordt klasse C_j de uitkomst van de optimale functie $f^*(x_0)$.

4.4 Verschil lineaire regressie en KNN-regressie

$f(x) = E(y|x)$ is lineaire regressie aanname. KNN probeert $E(y|x)$ zonder aannames te schatten door omringende waarden te analyseren. Stel data hangt lineair samen, kan je $f(x) = E(y|x)$ wel aannemen. Meer datapunten nodig voor KNN tov van lineaire regressie om zekerder te zijn dat er minder fouten op de testdata voorkomen. Het aantal waarnemingen dat je nodig hebt, hoe groter de functieruimte is. Minder waarnemingen, minder flexibel model. Overweging tussen lineaire regressie en KNN hangt af van overfitting. KNN heeft mogelijkheid tot overfitting waar lineaire regressie dat niet heeft.

5 Lineaire regressie

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

Lineaire regressie slaat niet op de macht van X, maar op de β. Want X en Y zijn gegeven. In termen van de coëfficiënten is de vergelijking lineair. Dus onderstaande vergelijking is ook lineair:

$$Y = \beta_0 + \beta_1 X_1 + \epsilon$$

Eigenschappen van $\hat{\beta}_j$:

- schatters $\hat{\beta}_j$ zijn zuiver: $E(\hat{\beta}_j) = \beta_j$
- de verdeling van de schatters is bekend

95% betrouwbaarheidsinterval: als je oneindig keer steekproef doet valt $\hat{Y}$ in 95% van de gevallen in Y.

Voordelen lineaire regressie:

- Hoge interpreteerbaarheid
- Bij beperkt aantal waarnemingen goed bruikbaar
- Grote storingen goed bruikbaar
- Bij sparse data goed bruikbaar
- Door gebruik te maken van transformaties toch redelijk flexibel

Wat kan er misgaan?

- Y hangt niet lineair van $X_1, X_2, ..., X_p$ af
- Correlatie tussen storingstermen
- Niet-constante variantie van de storingen
- Uitschieters
- Punten met veel invloed
- Collineaire variabelen

Vuistregel Robert: 'Aantal parameters mag niet zelfde orde van grote hebben als het aantal waarnemingen. Ten minste 10x zo veel waarnemingen voor een goede voorspelling. Als die waarnemingen niet aanwezig zijn kan ridge-regressie gebruikt worden om een goede voorspelling (i.e. variantie verlagen) te genereren.'

6 Regularisatie

6.1 Ridge-regressie

Laat de n waarnemingen van de train-data gegeven zijn door

$$(x_{i1}, x_{i2}, ..., x_{ip}, y_i), \text{ met } i \in 1, 2..., n$$

Dan wordt de foutmaat in ridge-regressie gegeven door

$$\text{ridge-fout} = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j X_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2$$

met λ een waarde die de tuning-parameter wordt genoemd. Bij kleine λ mag beta-schijf groter en bij grote λ ben je streng en mag beta-schijf niet groter. Ridge-regressie maakt gebruik van l_2-regularisatie ($\beta_1^2 + \beta_2^2 < 1$).Kleinere coëfficiënten impliceert eenvoudigere functies om overfitting tegen te gaan. Door middel van **cross-validatie** kan je de λ bepalen. Een juiste λ zal ervoor kunnen zorgen dat de variantie meer daalt dan de onzuiverheid toeneemt, waardoor de test-MSE kleiner wordt.

6.2 Lasso-regressie

Lasso: least absolute shrinkage and selection operator

$$\text{lasso-fout} = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j X_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$$

In plaats van de l_2-penalty van ridge-regressie, maakt lasso-regressie gebruik van de l_1-penalty ($|\beta_1| + |\beta_2| < 2$). β_0 bestraf je niet en zit dus niet in de fouten.
Kleine λ: kleinste kwadraten methode
Grote λ: niet ver zoeken, dicht bij oorsprong blijven
Gebruik CV om om λ te zoeken om de ruimte te bepalen.

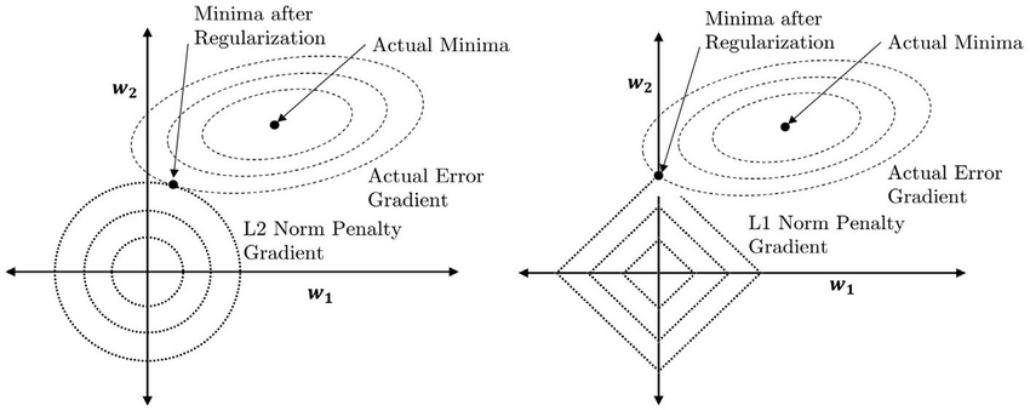


Figure 5: l_2-regularisatie (links) en l_1-regularisatie (rechts)

Week 3

Aandachtspunten:

- classificatie: foutmaat voor een kansmodel
- Maximum likelihood estimator (MLE)
- Lineaire discriminantanalyse (LDA) en principal component analyse (PCA)

7 Classificatie: foutmaat voor een kansmodel

informatie-entropy (loss functie):

$$\text{logistic loss} = -\sum_{i=1}^{n}(y_i)ln(\hat{y_i}) + (1 - Y_i)ln(1 - \hat{y_i})$$

$$(y_i)ln(\hat{y_i}) + (1 - Y_i)ln(1 - \hat{y_i}) = \begin{cases} ln(1 - \hat{y_i}), & \text{if } \ y_i = 0 \\ ln\hat{y_i}, & \text{if } \ y_i = 1 \end{cases}$$

Logistische regressie:

$$ln(\frac{\hat{y}}{1 - \hat{y}}) = \hat{\beta}_0 + \hat{\beta}_1 x$$

sigmoid-activatiefunctie (i.e. sigma-transformatie) differentieert makkelijk. SGD afname is makkelijk te berekenen:

$$\frac{e^x}{1 + e^x} = \sigma(x)$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Least squared gebruiken we niet, omdat gemiddelde gekwadrateerde fout geen realistische schatters geven. Willen foutmaat kiezen zodat we schatters krijgen die zich netjes gedragen en met het probleem te maken hebben. Voor regressie is LS wel goede schatter.

8 Maximum likelihood estimator

Maximum likelihood bij logistische regressie:

$$P(Y|X = x) = P(Y = 1|X = x)^Y (1 - P(Y = 1|X = x))^{1-Y}$$

MVUE: Minimale variantie voor zuivere schatter
Logistische regressie is een uitbreiding van sigmoid met meerdere voorspellende variabelen:

$$\hat{y}(x_1, x_2, ..., x_p) = \frac{e^{\hat{\beta}_0 + \sum_{i=1}^{p} \hat{\beta}_i x_i}}{1 + e^{\hat{\beta}_0 + \sum_{i=1}^{p} \hat{\beta}_i x_i}}$$

Logistische regressie voor meer klassen (softmax):

$$ln(\frac{P(Y = C_1|X = x)}{P(Y = C_k|X = x)}) = \beta_{10} + \beta_1 x$$

$$ln(\frac{P(Y = C_2|X = x)}{P(Y = C_k|X = x)}) = \beta_{20} + \beta_2 x$$

9 LDA

In de situatie dat Y uit meer klasses $K > 2$ bestaat en dat er meer ($p > 1$) voorspellende variabelen zijn, maken we een vergelijkbare aanname. Namelijk dat, gegeven de klasse k voor Y, de voorspellende variabelen een **multivariate normale verdeling** volgen met een covariantiematrix die onafhankelijk is van de klasse k.
Ook nu zal de klasse bepaald worden door discriminantfuncties die lineair zijn in de p voorspellende variabelen.

LDA met $p = 1$:
Stel dat Y een lineaire variabele is die we willen schatten middels een continue X. Neem aan dat voor gegeven klasse $k \in 0, 1$ van Y,

$$X \sim N(\mu_k, \sigma^2)$$

Als we P(Y) weten, kunnen we via de stelling van Bayes P(Y—X) uitrekenen.

Als men aanneemt dat de covariantiematrix van de verdeling van de voorspellende variabelen voor elke klasse k van Y verschillend mag zijn, dan worden de discriminantfuncties kwadratisch (quadratic discriminant analysis).
LDA is dus een speciaal geval van QDA.
QDA ten opzichte van LDA heeft voordelen (het is immers algemener) en nadelen (het is immers algemener).

Er ontstaat een kwadratisch probleem als variantie van twee klassen dermate verschilt (zie foto). Als nog meer variabelen worden toegevoegd, welke mogelijk gecorreleerd zijn (diagonaalmatrix naar $\frac{1}{Cov(X_1,X_2)}$ etc. op de plekken van de 0), functieruimte wordt veel groter (grotere vrijheidsgraden). Model flexibeler maken? Dus PCA gebruiken (demensiereductie en haalt correlaties weg tussen data).

10 PCA

PC_1 sla je dicht op de langste richting, op hoek van 90 graden maak je PC_2 (matrix diagonaliseren, programma vind een rotatie). Zodoende vallen de correlaties weg.

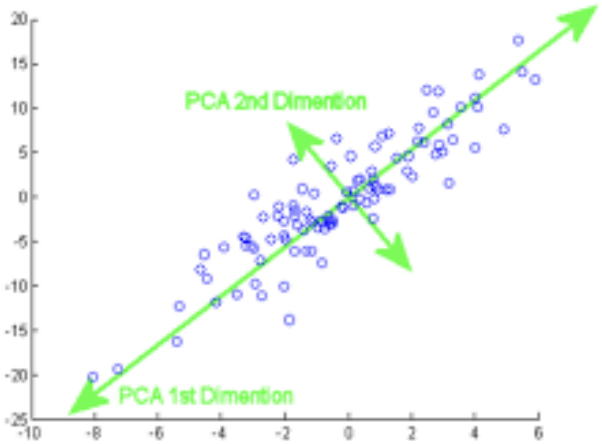


Figure 6: Principal component analysis

PC_1 beschikt over 90% van de variantie, dus andere PCAs zijn ruis.
Foto's zijn eigenlijk zeer hoog dimensionale waarnemingen. Een PCA uitgevoerd op een dataset van foto's van gezichten word ook wel Eigenfaces genoemd.

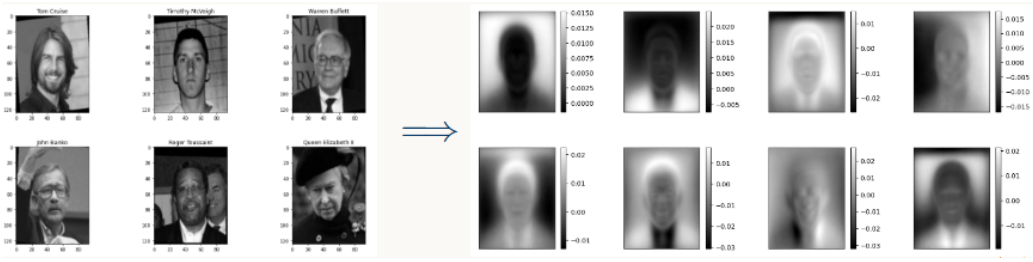


Figure 7: PCA's van gezichten

Het idee is dat iedere foto geschreven kan worden als een lineaire combinatie van Eigenfaces:

Figure 8: PCA samenhang op een gezicht

10.1 PCR

Principal components regression (PCR) of hoofdcomponenten-regressie bestaat uit een regressie-analyse op de eerste M hoofdcomponenten. De aanname van PCR is dat de richtingen waarin de voorspellende variabelen het meest variëren ook de richting zijn die het meest de variatie van Y kunnen verklaren.

Eerst PCA uitvoeren (als correlatie), een aantal PC's waar je regressieanalyse op uitvoert. Dus niet regressie op X_1 en X_2, maar op PC_1 en PC_2.

Week 4

Aandachtspunten:

- Modelbeoordeling en -selectie: Resampling methods
- Data snooping, data leakage (Leakage and the Reproducibility Crisis in ML-based Science)
  - zie taxonomy template in Data_leakage_taxonomy.docx in ../Vakken/MenT

11 Resampling methods

Manieren om iets te kunnen zeggen over de variantie van de schatting. Onderstaande methoden worden gebruikt om een **model te beoordelen** op testdata en het **selecteren van een model** (welk niveau van flexibiliteit):

1. Cross-validatie
2. Bootstrap

11.1 Cross-validatie

Train-test split kan ook gesplitst worden in train-val-test split (train set nog een keer splitsen). Train met verschillende waarden voor hyperparameters op de trainset en beoordeel de prestatie van de modellen op de validatieset. Nadelen zijn dat de fout-maat op de validatie afhankelijk is van de splitsing en je hebt een kleinere trainset waardoor de schatting van de fout-maat minder goed is (mogelijk een bias). Oplossing is **Leave-one-out cross-validation (LOOCV)**:

11.1.1 LOOVC

Trainset van n waarnemingen. Splits de trainset n keer in een:

- nieuwe trainset van $n-1$ waarnemingen, en
- een validatieset van overgebleven waarneming

Met als fout-maat de gemiddelde foutmaat op de n verschillende validatiesets. Zodoende wordt de trainset nauwelijks kleiner en is de foutmaat niet afhankelijk van de splitsing.

11.1.2 k-fold CV

Moet n keer getraind worden op $n-1$ waarnemingen. Dus, **k-fold cross-validation (k-fold CV)** gebruiken:
Stel dat we de hyperparameter λ willen tunen (bijvoorbeeld KNN om k te tunen):

- Definieer M verschillende waarden $\lambda_1, \lambda_2, ..., \lambda_M$ voor λ
- Bepaald voor elke waarde $m \in 1, ..., M$ middels k-fold CV de gemiddelde foutmaat $E_{CV}(m)$
- kies die waarde m^* voor m waarvoor $E_{CV}(m)$ het laagst is
- train opnieuw, maar nu op de gehele dataset en λ_{m^*}

Vuistregel: gebruik k=10 en gebruik honderd waarnemingen om slechts enkele parameters te tunen.

Wat kan λ zijn?:

- Wat is het type model dat wil ik gebruiken?
  - lineair of NN of SVM
  - polynomiaal model: tot hoeveelste orde
  - keuze voor tuning-parameter in regularisatie

11.2 Bootstrapping

(doen alsof er meer data is om achter de variantie van je steekproefgemiddelde te komen.)
Is een manier om de variantie van een schatter te schatten middels steekproeven uit een steekproef:

- oorspronkelijke steekproef S van n waarnemingen
- Trek met teruglegging B steekproeven van n waarnemingen
- Laat $\hat{\alpha}$ geschat zijn met S
- Het idee van bootstrapping is dat we de verdeling van $\hat{\alpha}$ kunnen schatten middels de geschatte parameters $\hat{\alpha}^{*b}$ $(b \in 1, 2, .., B)$ die geschat zijn op de B steekproeven.
- Een schatting voor de variantie van $\hat{}$ zal gegeven worden door de steekproefvariantie van de $\hat{\alpha}^{*b}$

Waarom zou je bootstrapping willen toepassen?:
Als de variantie heel hoog is, is de zekerheid laag. Je wilt met bepaalde zekerheid zeggen dat het steekproefgemiddelde een bepaalde waarde heeft.
Hyperparameter: parameters om leerproces aan te sturen
Parameter: gewichten/coëfficiënten = uitkomsten na leren

11.3 Data snooping/data leakage

- Geen correcte train-test-splitsing
  - Geen testset
  - Datapreparatie op train- en testset
  - Variabelenselectie op train- en testset
  - Dubbelingen in testset
- Verkeerde variabelen
  - Medicijngebruik om hoge bloeddruk te meten
- Testset is niet representatief
  - Lekkage in de tijd (om een voorspelling in de tijd te maken, hoort de train-set voor de testset hebben plaatsgevonden)
  - Geen oafhankelijkheid tussen train- en testset (voorbeeld: verschillende waarnemingen van dezelfde persoon in de train- en testset)
  - Bias in testset
    - Testset van specifieke regio, maar iets beweren over alle regio's
    - Ongewenste waarnemingen weglaten uit de testset

Week 5

Aandachtspunten:

- Beslisbomen regressie
- Beslisbomen classificatie
- Ensemble-technieken

12 Beslisbomen regressie

De foutmaat voor regressie met een beslisboom zal dan gegeven worden door

$$RSS = \sum_{j-1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Het doel van een beslisboom voor regressie zal zijn de J niet overlappende regio's $R_1, R_2, ..., R_J$ te vinden zo dat RSS geminimaliseerd wordt.

12.1 Het algoritme

1. Kies de variabele X_j en de waarde s zo dat de regio's

   $$\{X|X_j < s\} \text{ en } \{X|X_j \geq s\}$$

   de RSS minimaliseren.

2. Kies één van bovenstaande regio's en herhaal bovenstaande stap. De regio, variabele en waarde wordt zo gekozen dat het de RSS minimaliseert.

3. Herhaal bovenstaande stap tot een stop-criterium wordt bereikt. Bijvoorbeeld: geen enkele regio heeft meer dan vijf trainpunten.

12.1.1 Keuze voor algoritme

Dit algoritme zorgt voor rechthoekige gebieden.

Dit is een voorbeeld van een *greedy* algoritme. (greedy algoritme kijkt maar 1 stap vooruit, dat hoeft niet beter te zijn. Om op te lossen: pruning/snoeien)

De bomen die hier uit volgen kunnen behoorlijk complex zijn (erg veel gebieden).

We zouden als stop-criterium kunnen gebruiken dat elke stap de RSS met minstens een bepaalde waarde verlaagd. Een nadeel van deze methode is dat dit weer te rigide is. Er zou een zeer goede stap, na een vrij slechte stap kunnen komen.

Om overfitting tegen te gaan, kunnen we een deelboom van de oorspronkelijke boom gebruiken (snoeien, pruning).

We zouden die deelboom willen kiezen die de test-RSS minimaliseert. Cross-validation op alle deelbomen is niet altijd mogelijk.

Gebruik om te snoeien de volgende fout-maat

$$\text{cost-complexity-maat} = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

waar $|T|$ staat voor het aantal eindpunten van de boom T.

12.2 Cost complexity pruning (weakest link pruning)

1. Recursive binary splitting met stopcriterium op de traindata: volledige boom
2. Cost complexity pruning op de boom uit de vorige stap: rij van deelbomen afhankelijk van α
3. α bepaald je met cross-validatie met gemiddelde RSS. Voor elke $k \in \{1, 2, .., .K\}$
   (a) Herhaal stap 1 en 2 op de nieuwe traindata
   (b) Bepaal de RSS op de validatieset als functie van α

   De deelboom uit 2 met α uit 3

13 Beslisbomen classificatie

Ook nu verdelen we de ruimte van voorspellende variabelen in J niet overlappende regio's $R_1, R_2, ..., R_J$.

Stel dat een datapunt $(x_1, x_2, ..., x_p)$ in regio R_j valt, dan zal de voorspelling bij een beslisboom voor classificatie gegeven worden door:

$$\hat{y}(x_1, x_2, ..., x_p) = \hat{y}_{R_j}$$

waar $\hat{y}_{R_j}$ staat voor de klasse waar de meeste $y_i \in R_j$ vallen.

De foutmaten voor een classificatie beslisboom worden gegeven door twee maten. Het is gebruikelijk om de Gini-index of de entropie te gebruiken om de ruimte te splitsen. Dit zal zorgen voor regio's met hoge en lage fracties:

$$\text{Gini index} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

$$\text{entropy} = \sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$$

waarbij $\hat{p}_{mk}$ de fractie van traindata in regio R_m van klasse k zijn.

Daarnaast is het gebruikelijk om de error-rate te gebruiken als de accuraatheid van belang is voor het snoeien van de boom:

$$\text{error-rate} = 1 - max_k(\hat{p}_{mk})$$

14 Ensemble-technieken

Beslisbomen hebben een hoge variantie, door traindata aan te passen ontstaat meteen een hele andere boom. Bij ensemble methoden neem je het gemiddelde van uitkomsten van meerdere simpele modellen.

$$Var(x) = \sigma^2 \rightarrow Var(\bar{x}) = \frac{\sigma^2}{n}$$

Des te hoger n, des te lager je variantie. Door ieder model een andere steekproef te geven verhoog je n en zo verlaag je de variantie.

Moet je zorgen dat de modellen onafhankelijk zijn $\rightarrow$ ieder model eigen steekproef. Anders dan k-fold omdat je modellen daar niet onafhankelijk zijn.

De simpele modellen die je middelt noem je 'weak learners'. Die hebben misschien een hoge bias, maar die middel je weg.

- weak learner is iets beter dan een naïef-nul-model
  - die voorspelt altijd gemiddeld y of random gok bij binaire classificatie.

14.1 Bagging (regressie)

- Bootstrap aggregation
  - Bij bootstrap pak je B voorbeelden zonder terugleggen, zo maak je steekproeven en zeg je iets over de variantie.
- Bagging
  - trek B steekproeven en zoek voor elke $k \in B$ een model

Laat $f^{*b}(x)$ het model zijn uit steekproef b, dan geldt voor regressie:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{i-1}^{B} f^{*b}(x)$$

Voor classificatie geldt: meeste stemmen gelden.

Een waarneming die niet in de steekproef b zit noemen we out-of-bag. Deze data gebruik je om te testen. Dus om f(2) OOB te testen, gebruik ik 1 en 3, want voor hun is 2 de geschikte testdata.

Als b groot genoeg is, is de OOB-fout gelijk aan de LOOCV-fout. Bagging is nauwkeuriger

b	b {1,2,3}	OOB
1	1, 1, 1	2, 3
2	3, 2, 3	1
3	1, 3, 3	2

maar minder interpreteerbaar.

14.2 Random forest

Random forest wordt gemaakt uit bagged bomen, waarbij elke boom een gelimiteerd aantal features heeft met $m < p$. Bij random forest is het error landschap onbekend, je zoekt naar een minimum, maar de kans is groot dat je een lokaal minimum vindt. Als er een sterke voorspeller is $q \in p$ dan begint elke boom het zelfde en is er correlatie in modellen. Want een boom is greedy. Zo blijf je hangen in het lokale minimum. Door andere variabelen de-correleer je. m is de tuning parameter. Je geeft zo flexibiliteit om van A naar B te hupsen (waarbij A een lokaal minimum is en B het echte minimum).

m bepaal je met OOB-error (out-of-bag). De OOB-voorspelling zal een goede maat kunnen zijn voor een test-voorspelling. De OOB-error is de foutmaat volgend uit de OOB-voorspellingen en is een geldige schatting van de test-error op het bagged model. Als B groot genoeg is, is de))B-error equivalent met de LOOCV-fout. Er zijn vuistregels en deze zijn zelf te tunen. Bij sterk gecorreleerde waarden moet m klein zijn.

- Regressie: $m = \lfloor \sqrt{p} \rfloor$
- Classificatie: $m = \lfloor \frac{p}{3} \rfloor$

Hoe groot moet B zijn in bagging?
Een hogere B zorgt niet voor overfitting, maar kost alleen tijd. Plot))B-error en kies B waar die stabiliseert. Standaard is 100 of 500. Random forests worden niet gebruikt voor computer vision.

14.3 Boosting

14.3.1 Het idee:

1. we gebruiken eenvoudige modellen (beslisboom met d beslissingen).
2. We fitten een ander model op storingen van het vorige model.
3. Het nieuwe model wordt dan het vorige model plus het andere model (maal λ).
4. Dit doen we B keer en we middelen over alle uiteindelijke B modellen.

14.3.2 Het algoritme

1. Laat $\hat{f}(x) = 0$ en $r_i = y_i$ voor alle x_i in trainset
2. Voor $b \in \{1, 2, ..., B\}$ herhaal
   (a) Fit een boom $\hat{f}^b$ met d beslissingen op de trainset (X, r)
   (b) Update de voorspelling:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$
   (c) Update de storing:
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
3. Boosted model:
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

14.3.3 De tuning parameters

1. Het aantal bomen B. Boosting kan overfitten als B te groot is: cross-validation.
2. Shrinkage parameter λ (klein en positief). Vuistregel: 0,01 of 0,001.
3. Het aantal beslissingen d. Dit bepaalt de complexiteit. Meestal $d = 1$, een d van ongeveer 5 werkt meestal even goed als d uit cross-validation

14.4 Bayesian Additive Regression Tree (BART)

Een BART-model maakt gebruik van een bagging (middelen over bomen die op andere steekproeven dan de dataset gefit) en boosting (middelen bomen die fitten op de fout van vorige boom) idee:

- Elke boom is willekeurig
- Elke boom bouwt op de storing van de vorige

14.4.1 Het idee

1. Zodra we voor gegeven iteratie b en gegeven boom k de partiële storing hebben berekend, fitten we een nieuwe boom ($\hat{f}_k^b$) op de data (X, r).
2. We eisen dat de nieuwe boom een perturbatie (nieuwe takken snoeien en uitkomst wijzigen) van de oude boom uit de vorige iteratie zal zijn.

14.4.2 De notatie

- K: het aantal bomen
- B: het aantal iteraties
- $\hat{f}_k^b(x)$: de voorspelling van de k-de boom in de b-de iteratie

Na elke iteratie zullen de K bomen worden opgeteld

$$\hat{f}_k^b(x), \text{ voor } b \in \{1, 2, ..., B\}$$

Per iteratie zullen de K bomen 1 voor 1 worden aangepast door gebruik te maken van een partiële storing

$$r_i = y_i - \sum_{k'<k} \hat{f}_k'^b(x_i) - \sum_{k'>k} \hat{f}_k'^b - 1(x_i)$$

14.4.3 Het algoritme

1. Laat $\hat{f}_1^1(x) = \hat{f}_2^1(x) = ... = \hat{f}_K^1(x) = \frac{1}{nK} \sum_{i=1}^{n} y_i$
2. Bereken $\hat{f}^1(x) = \sum_{k=1}^{K} \hat{f}_K^1(x) = \frac{1}{n} \sum_{i=1}^{n} y_i$
3. Voor elke $b \in \{2, 3, ..., B\}$
   (a) Voor elke $k \in \{1, 2, ..., K\}$
      i. Bereken voor $i \in \{1, ..., n\}$ de huidige partiële storing r_i
      ii. Fit een nieuwe boom $\hat{f}_k^b(x)$ op r_i door de k-de boom uit de vorige iteratie $\hat{f}_k^b - 1(x)$ willekeurig te veranderen.

(b) Berekenen $\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x)$

4. Bereken het gemiddelde na L trekkingen (burn-in period):

$$\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$$

Week 6

Aandachtspunten:

- Support Vector Classifier
- Support Vector Machine

15 Support Vector Classifiers

De support vector classifier of soft margin classifier is een generalisatie van het optimaal scheidend hypervlak. De generalisatie bestaat uit het toelaten van verkeerd-geclassificeerde traindata ten koste van een "straf". Hiermee is het mogelijk om de support vector classifier te gebruiken bij data waarvan de klasses niet perfect te scheiden zijn door een hypervlak. Ook bij data waar de klasses wel perfect te scheiden zijn met een hypervlak, zou een support vector classifier betere (breedte van marge) resultaten kunnen geven.

- Bij binaire classificatie zoek je een hypervlak die klassen van elkaar scheiden (seperating hyperplane).
- Voor hetzelfde probleem kunnen meerdere hypervlakken bestaan, oneindig veel.
- We zoeken de maximal margin classifier. Zoek de kortste afstand tussen punt en vlak, die afstand maximaliseren we.
- In praktijk zoeken we een lijn precies tussen de 2 klassen. De punten het dichtst bij noemen we de *support vectors*. Als die veranderen, verandert het hele hypervlak.
- Meer support vectors geeft meer variantie, maar minder bias, je kan inzien welke datapunten de SV's zijn.

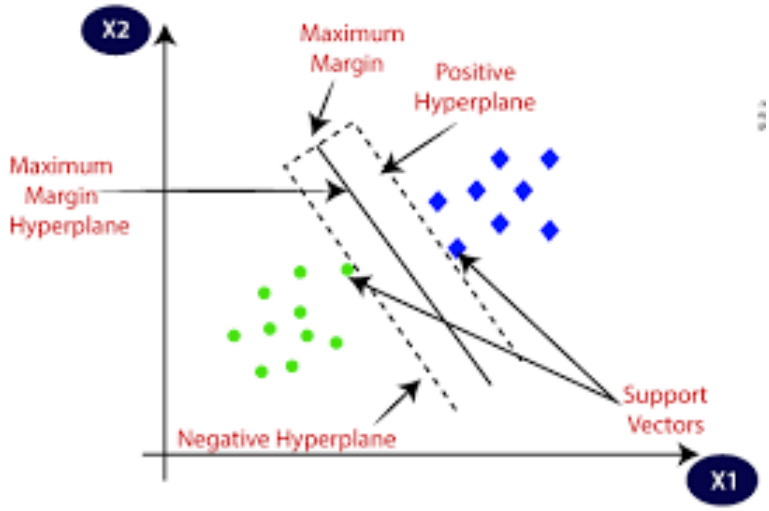


Figure 9: Support Vector Classifier (SVC)

16 Support Vector Machines

Support Vector Machines (SVM) zijn een generalisatie van support vector classifier (SVC).

Optimaal scheidende kromme
Een niet-lineair vlak om klasses te scheiden. (x_0: testdata en x_i: support vectors, $h(x_i)$: transformatie op de support vectors).

We zouden kunnen vermoeden dat met twee voorspellende variabelen (X_1 en X_2) de klasses gescheiden worden door

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \text{ (een parabool).}$$

Dit is een lineaire lijn in de nieuwe variabelen

$$h(x_i) = h(x_{i1}, x_{i2}) = \begin{pmatrix} h_1(x_{i1}, x_{i2}) \\ h_2(x_{i1}, x_{i2}) \end{pmatrix} = \begin{pmatrix} x_{i1}^2 \\ x_{i2} \end{pmatrix}$$

Transpose van $\beta = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$:

$$\beta^T = \begin{pmatrix} \beta_1, \beta_2 \end{pmatrix}$$

$$\beta^T X_0 = \begin{pmatrix} \beta_1, \beta_2 \end{pmatrix} \begin{pmatrix} X_{01} \\ X_{02} \end{pmatrix} = \beta_1 X_{01} + \beta_2 X_{02}$$

Kernel
Een kernel is een functie die de gelijkenis tussen twee punten meet. We definiëren

$$K(x, x') = \langle h(x)\ ,\ h(x') \rangle$$

en noemen K een kernel.

"Een support vector machine is een SVC met een kernel K die voor specifieke transformaties staan."

Twee standaard kernels zijn (1) polynomiale kernels met graad d en (2) radiaal (RBF).

$$(1) : K(x, x') = (1 + \langle x\ ,\ x' \rangle)^d$$
$$(2) : K(x, x') = exp)(-\gamma \langle x - x'\ ,\ x - x' \rangle)$$

Hoe kleiner de C, hoe minder marge om marge te overschrijden. De scheidende kromme kan dan zeer kronkelig zijn (te kronkelig = overfitting).

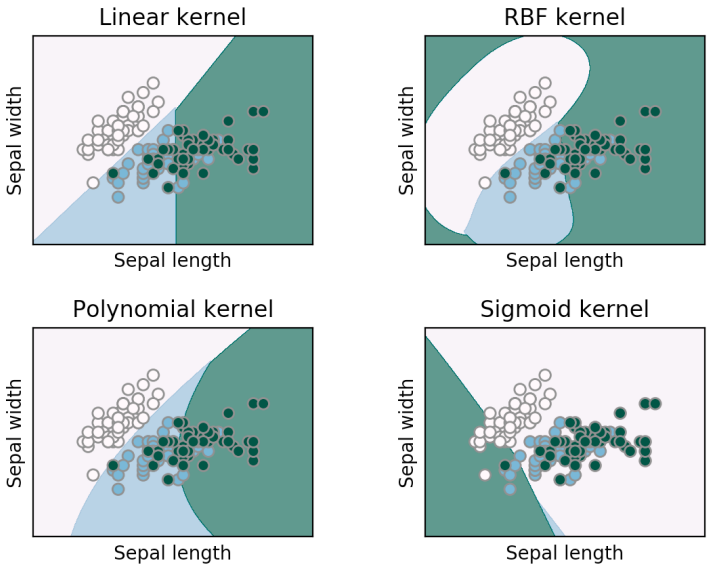


Figure 10: SVM kernels

Bijdrage van punten die ver van marge liggen neem je niet mee, alleen punten meenemen die marge overschrijden. Deze maximalisatie methode noemen we de Hinge loss.

$$\text{Hinge loss} = L(X, y, \beta) = \sum_{i-1}^{n} max(0, 1 - y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$

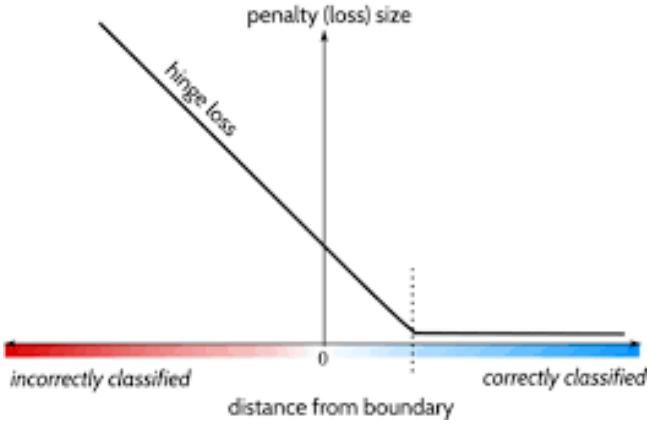


Figure 11: Hinge loss (scharnier loss)

Classificatie op meer dan twee klasses K

- One-versus-one (all pairs):
  Maak voor elk paar van klasses (k, k') een svm en kies voor een gegeven x0 de klasse die het vaakst wordt gekozen in de $\frac{1}{2}K(K-1)$ SVM's.

- One-vs-all:
  Maak voor elke klasse k een SVM (k tegen niet-k). Kies voor een gegeven x_0 die klasse k volgend uit een SVM waarbij x_0 het verst van het optimale hypervlak is. Als het datapunt op vlakken van meerdere klasses ligt, classificeer je het datapunt als de klasse waarvan de marge het verst van het datapunt afligt.

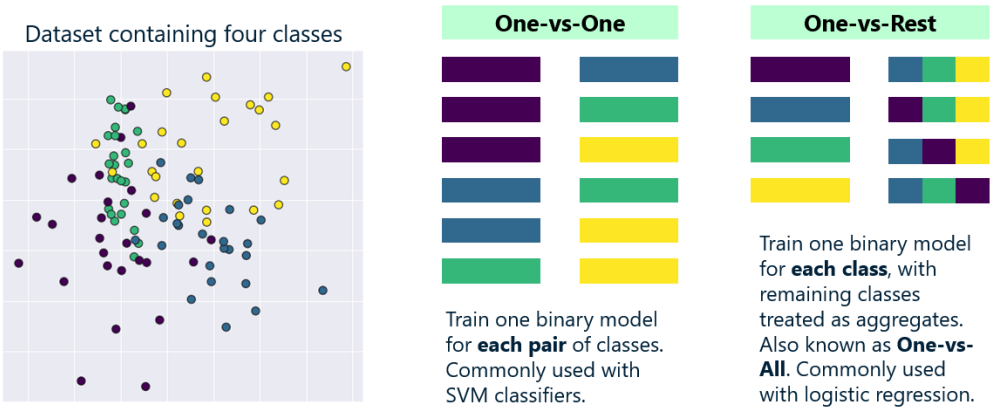


Figure 12: One-vs-one classificatie (links) en one-vs-rest classificatie (rechts)

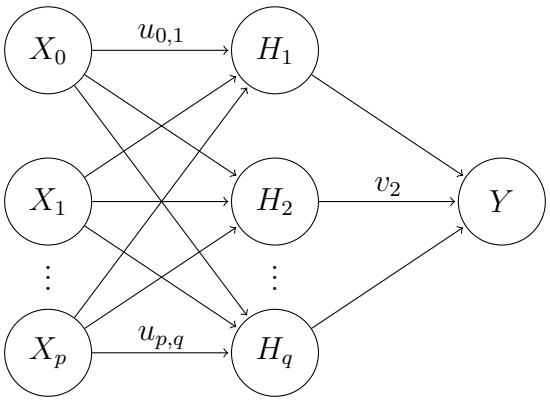
Week 8

17 Neurale Netwerken

17.1 Lineaire neurale netwerken

Lineaire netwerken zijn altijd terug te herleiden tot $Y = \beta X$. Zo is ook onderstaande neurale netwerk hiernaar terug te herleiden:

$$Y = \sum_{j=1}^{q} v_j (\sum_{i=0}^{p} u_{ij} X_i) = (\boldsymbol{U}\overrightarrow{v})\overrightarrow{X} = \overrightarrow{\beta}\overrightarrow{X}$$

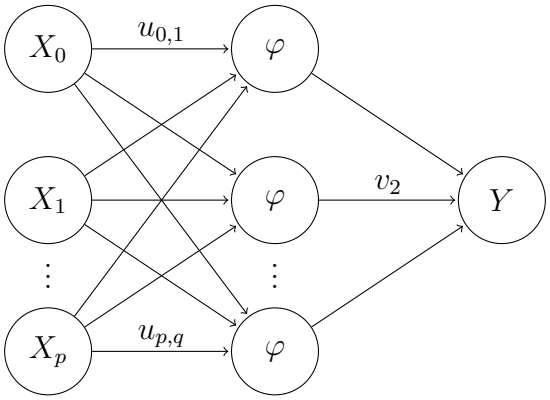


Een opeenstapeling van lineaire transformaties blijft een lineaire transformatie. Om uit het lineaire regime te ontsnappen moeten we een niet-lineaire transformatie op iedere laag toepassen, de zogenoemde activatiefunctie.

17.2 Logistische neurale netwerken (sigmoid)

De sigmoid-activatiefunctie zal toegepast worden, waardoor de volgende formule en netwerk ontstaan:

$$Y = \sum_{j=1}^{q} v_j \varphi(\sum_{i=0}^{p} u_{ij} X_i) = (\mathbf{U}\overrightarrow{v})\overrightarrow{X} = \overrightarrow{\beta}\overrightarrow{X}$$

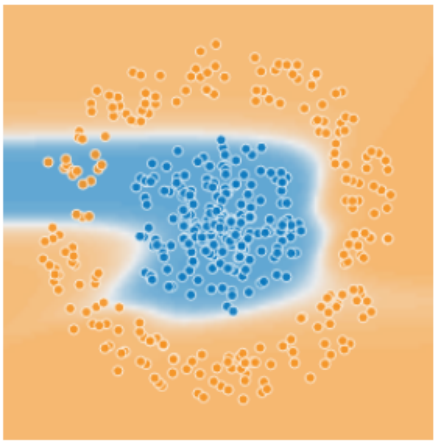


De structuur van het feed forward netwerk is:

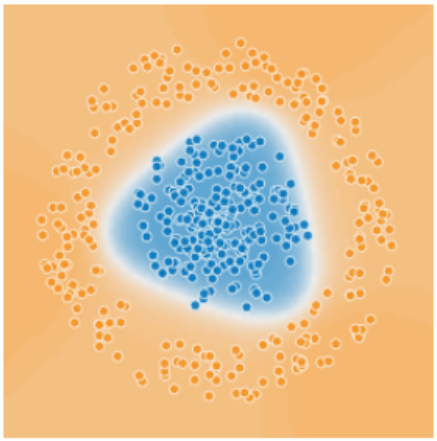
1. Invloer $\overrightarrow{X}$
2. Matrix vermenigvuldiging (weights) $\mathbf{U}\overrightarrow{X}$
3. niet-lin. transformatie (activatie) $\varphi(\mathbf{U}\overrightarrow{X})$
4. lin. combinatie $\overrightarrow{v}\varphi(\mathbf{U}\overrightarrow{X})$

Er kunnen ook meerdere lagen toegevoegd worden. Het is niet per definitie zo dat meer lagen voor een beter resultaat zorgen. Het aantal nodes is minstens net zo belangrijk. Voor meer informatie, zie *Deep, Skinny Neural Networks are not Universal Approximators.*

Twee nodes per laag kan mogelijk geen goede voorspelling maken, want er zitten maar twee grensvlakken aan verbonden als er maar twee nodes per layer aanwezig zijn. Zo kan er bijvoorbeeld geen cirkel gevormd worden bij 6, 2-dimensionale layers (a), terwijl dat met een enkele 3-node-layer wel gaat (b).



(a) The decision boundary learned with six, two-dimensional hidden layers is an unbounded curve that extends outside the region visible in the image.



(b) A network with a single three-dimensional hidden layer learns a bounded decision boundary relatively easily.

17.3 Hoe trainen we een neuraal netwerk?

Neem een netwerk met l lagen, van groottes n_1 t/m n_l, activatiefunctie φ^1 t/m φ^l en gewichten $\mathbf{U}^1$ t/m $\mathbf{U}^l$. Dus:

$$Y = g(\overrightarrow{X}) = \varphi^l(\mathbf{U}^l\varphi^{l-1}(\mathbf{U}^{l-1}\dots\varphi^2(\mathbf{U}^2\varphi(\mathbf{U}\overrightarrow{X}))))$$

We introduceren een loss functie $L(\hat{y}), y$.

$$L(\hat{y}), y = L(g(\overrightarrow{x}), y)$$

Het gewicht u_{ij}^k is het gewicht dat hoort bij de pijl van de node i in laag $k-1$ naar node j in laag k. We kunnen met de kettingregel de loss differentiëren naar u_{ij}^k.

$$\frac{\partial L}{\partial u_{ij}^k} = L'\varphi^{l'}\mathbf{U}^l\varphi^{l-1'}\dots\varphi^{k+1'}\mathbf{U}^{k+1}\varphi_i^{k-1}$$

waar φ_i^{k-1} de activatie van de i^{de} neuron in laag $k-1$ is.

We zien dat de afgeleide alleen afhangt van:

- de afgeleiden van de lagen rechts
- de activatie van de laag links

We kunnen dus redelijk efficiënt al deze afgeleiden van rechts naar links uitrekenen. Dit algoritme heet *back propagation.* Dit wordt gebruikt in het volgende neurale netwerk cyclus:

1. Bepalen van gewichten
2. De loss functie bepalen met de feedforward fase
3. Back propagation vanaf loss functie om gewichten te optimaliseren
4. Repeat

17.4 Stochastic Gradient Descent

Om onze gemiddelde loss $\bar{L}$ nu te minimaliseren gaan we als volgt te werk:

1. We splitsen de data op in kleine random batches
2. Voor iedere waarneming in een batch bepalen we $\frac{\partial L}{\partial u_{ij}^k}$
3. Deze afgeleiden middelen we tot $\frac{\partial \bar{L}}{\partial u_{ij}^k}$
4. We passen de gewichten als volgt aan:

$$u_{ij}^k(t+1) = u_{ij}^k(t) - r\frac{\partial \bar{L}}{\partial u_{ij}^k}$$

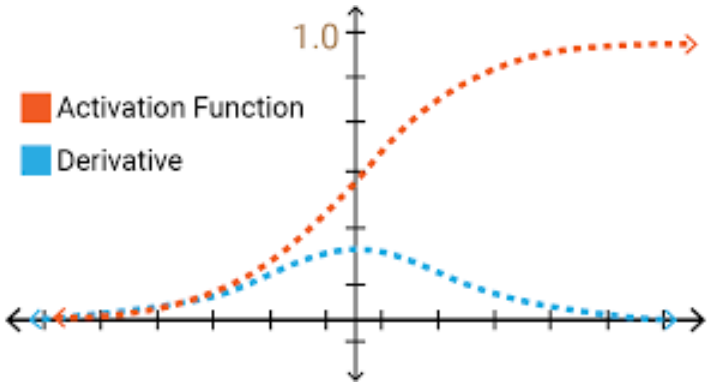
   hier is r de *learning rate*.
5. Als de volledige dataset op deze manier is doorlopen noemen we dit een epoch en herhalen we dit proces.

Batchsize

Een te kleine batchsize kan zorgen voor slecht trainen van het model, omdat er dan van enkele klassen data in een batch kan zitten. Als het model enkel daarop traint, traint het niet op alle klassen (traint te specifiek). Een batchsize van 32 is meestal groot genoeg, representatief voor data. Bij veel klassen kies je voor een grotere batchsize.

17.5 Vanishing Gradient Problem

Veel activatiefuncties vlakken af bij grote waardes van hun invoer. Hun afgeleides zijn dus heel klein en worden ook nog met elkaar vermenigvuldigd. Het gevolg is dat u_{ij}^k voor kleine k nauwelijks aangepast word en dat het netwerk nauwelijks leert. Om dat probleem (deels) op te lossen wordt tegenwoordig vaak de ReLU-activatiefunctie gebruikt.



17.6 Voorbereiding

Voorbereiding van het maken van een neuraal netwerk:

- Probleemanalyse
- Data verzameling
- Train/test split
- Data verkenning
- Keuze loss-functie
- Keuze metrieken

Feature selection
Model kan slechter scoren als er betekenisloze variabelen in het model verwerkt zijn. Je wilt de variabelen afgaan en analyseren welke belangrijk zijn.

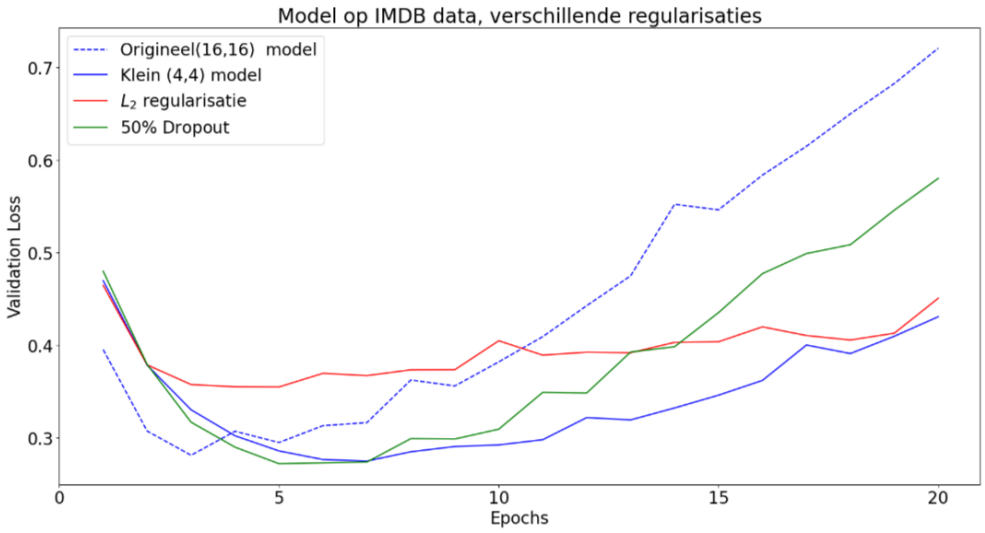
Problemen met modelverbetering

Probleem	**Oplossing**
Trainen doet niks, je loss veranderd niet of nauwelijks per epoch en het model leert niks.	Learning rate of batchsize aanpassen.
Het model traint wel, maar krijgt geen goede resultaten. Het verslaat simpele baselines niet of nauwelijks. Denk aan val_acc die niet omhoog gaat.	Data probleem/model: (1) te weinig data, (2) misclassificatie, (3) verkeerde aanpak.
Het model traint en krijgt redelijke resultaten, maar na verloop van tijd gaat het nooit overfitten en blijft het dus underfit. Denk aan stagneren van train_acc en val_acc.	Meer lagen toevoegen/lagen groter make.n
Duurt lang.	Geduld hebben.

Regularisatie

Na het behalen van een goede loss-epoch-grafiek willen we het moment van overfitten zo lang mogelijk uitstellen.

- Early stopping (stoppen bij laagste val_acc)
- Netwerk verkleinen als groot netwerk overfit, te klein blijft underfit. Probeer de juiste samenstelling te vinden)
- L_1 en/of L_2 regularisatie. (bij grote modellen werkt het niet goed)
- Dropout (willekeurige activaties op 0 zetten, bij grotere modellen werkt het wel goed)



Vuistregel: L_1 en/of L_2 regularisatie werkt bij kleinere modellen, dropout werkt bij grotere modellen.

Samenvatting en workflow

- Vooraf
  - Definieer het probleem
  - Verzamel de data
  - Bestudeer de data
  - Kies metrics en bepaal een baseline
  - Kies een *loss* functie
- Model ontwikkeling
  - Normaliseer data
  - Missende waardes?
  - Feature selectie
  - Maak een model dat je baseline verslaat
- Model verbetering
  - Voeg lagen toe
  - Maak lagen groter
  - Train voor meer epochs
- Regularisatie
  - Dropout of L_1, L_2 regularisatie
  - Experimenteer met hyperparameters (learning rate, grootte van lagen)

Week 9

17.7 Convolutionele Neurale Netwerken

Beeldherkenningstaken:

- Single-label classificatie
- Multi-label classificatie (meerdere dingen tegelijk herkennen)
- Segmentatie (classificatie op pixelniveau)
  - Semantic
  - Instance
- Object Detection

Waarom dense layers niet werken bij beeldmateriaal

- Globaliteit – iedere mogelijke combinatie van inputs zou een mogelijk signaal kunnen bevatten.
- Deze signalen verschillen allemaal van elkaar.

Voor beeldmateriaal gelden de volgende aannames:

- Lokaliteit – Alleen combinaties van inputs met zijn buren bevatten zinvolle signalen.
- Translatie-invariantie – Deze signalen blijven hetzelfde als de input met de buren verplaatst.

Een convolutielaag is een dense layer waar merendeel van de gewichten uit zijn gezet. *padding = 'same'* plaatst virtuele waarden aan rand die dezelfde waardes hebben als de pixels ernaast. Een aantal instellingen kunnen verder het aantal gewichten beïnvloeden:

- Kernel size
- Padding (vooral bij segmentatie, je wilt daar grootte behouden)
- Stride (stapgrootte kernel size)
- Pooling (max, global average, low pooling van de filtercombinatie)

Max pooling werkt beter dan het nemen van strides, maar verliest hiermee wel informatie over waar deze features zich bevinden (belangrijk voor segmentatie)

17.7.1 Typische architectuur

- Data augmentatie
- Meerdere convolutie blokken
  - Een of meer convolutielagen
  - Max pooling om het aantal gewichten laag te houden
- Flatten
- Een dense layer voor de uitvoer

17.7.2 Transfer Learning

De enorme voorgetrainde netwerken kunnen redelijk algemene patronen herkennen die toepasbaar zijn op heel veel andere beeldtaken. Daarom kan het bij weinig data handig zijn om een model te pakken die bijvoorbeeld een ImageNet-competitie heeft gewonnen. Pak de convolutielagen van deze netwerken en vervang de dense layer en loss-functie.

Werkwijze:

1. Voeg eigen dense layer toe aan voorgetrainde basis
2. Bevries gewichten van voorgetrainde basis
3. Train je netwerk
4. Finetunen
5. Ontvries gewichten met de laatste lagen (met factor 10 lagere learning rate)
6. Opnieuw trainen

17.7.3 Transpose Convolutie

- Onze invoer bestaat uit 200×200 pixel plaatjes.
- We willen deze plaatjes segmenteren, ofwel we kennen een klasse toe aan iedere pixel
- Onze uitvoer moet dus bestaan uit 200×200 geclassificeerde pixels

Echter:

- We nemen convoluties met een stride van 2
- Dus na deze convolutie halveert onze resolutie
- Uiteindelijk hebben we een 25×25 'plaatje' met 256 features

Een transpose convolutie probeert te leren hoe deze convoluties weer ongedaan gemaakt moeten worden. De (200,200,**3**) van de invoer geeft het aantal kleurenchannels weer (RGB), terwijl de (200,200,**3**) van de uitvoer het aantal klasses weergeeft.

17.7.4 1×1 convoluties

Pixels worden halveert, aantal filters/features verdubbelen. Kan tot een convolutielaag met 1 pixel en heel veel filters. Deze convoluties worden soms gebruikt om het aantal features te veranderen zonder ruimtelijke informatie te veranderen. Ze zijn een voorbeeld van een Network-in-network laag.

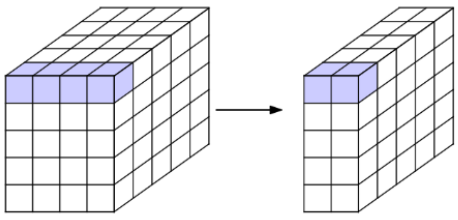


Figure 13: 1×1 convoluties

17.7.5 Residual connecties

Pixels worden halveert, aantal filters/features verdubbelen. Dan ontstaan er hele diepe netwerken. Kan niet notatie 'model.add()' gebruiken, want lopen 2 dingen parallel aan elkaar. Doen zodat het nog goed blijft trainen.

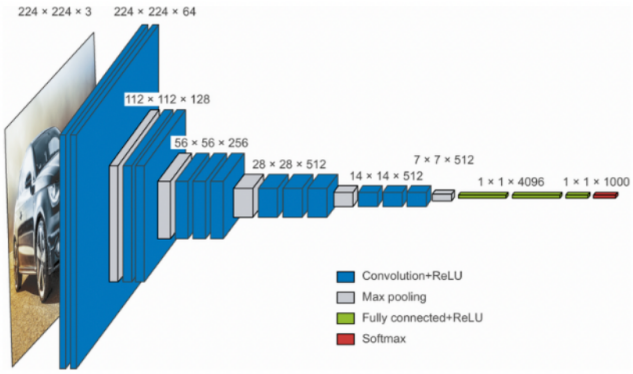


Figure 14: Residual netwerk

Het idee van residual connections is om de (gradient van de) fout dieper in het netwerk te laten propageren, zodat ook diepere lagen kunnen trainen. Dit doen we door de activatie van de diepere lagen bij nieuwe lagen op te tellen.

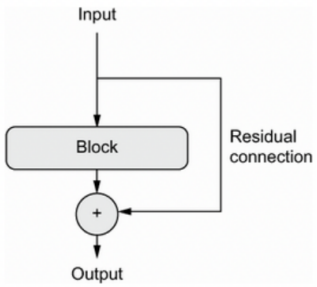


Figure 15: Residual connectie

- De residual bevat een 1×1 convolutie zonder activatie om te zorgen dat het aantal filters hetzelfde is als de uitvoer van het overgeslagen convolutieblok.
- Convoluties in het convolutieblok moeten met padding werken om te zorgen dat de resolutie niet veranderd.
- Als het convolutieblok gebruik maakt van pooling zal deze 1×1 convolutie met strides moeten werken zodat de resolutie hetzelfde blijft.
- Aan het eind wordt de residual connection bij de uitvoer van het convolutieblok opgeteld.

```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Conv2D(32, 3, activation="relu")(inputs)
residual = x
x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
x = layers.MaxPooling2D(2, padding="same")(x)
residual = layers.Conv2D(64, 1, strides=2)(residual)
x = layers.add([x, residual])
keras.Model(inputs=inputs, outputs=x).summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 32, 32, 3)]  0           []

 conv2d (Conv2D)                (None, 30, 30, 32)   896         ['input_1[0][0]']

 conv2d_1 (Conv2D)              (None, 30, 30, 64)   18496       ['conv2d[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 15, 15, 64)   0           ['conv2d_1[0][0]']

 conv2d_2 (Conv2D)              (None, 15, 15, 64)   2112        ['conv2d[0][0]']

 add (Add)                      (None, 15, 15, 64)   0           ['max_pooling2d[0][0]',
                                                                  'conv2d_2[0][0]']

==================================================================================================
Total params: 21,504
Trainable params: 21,504
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 16: Code van een simpel residual netwerk

17.7.6 Global Average Pooling

Convolutielaag (in plaats van dense en flatten layer) die evenveel filters als klasses heeft. Iedere filter komt overeen met iets dat je wilt herkennen. Gemiddelde van alle filters in de convolutielaag is de klasse die eruit komt.
Bij global average pooling wordt het gemiddelde van de hele filter genomen. De laatste convolutionele laag heeft even veel filters als er te voorspellen klassen zijn. Deze filters worden over de hele laag gemiddeld. Bron

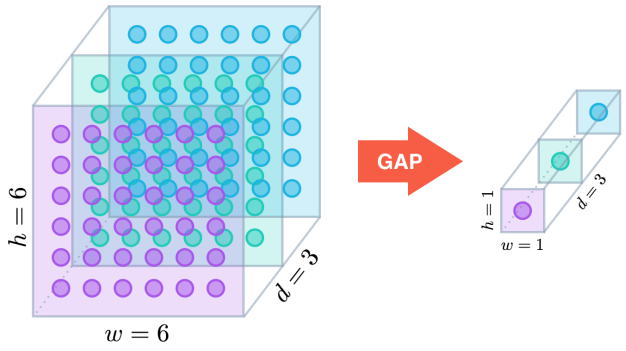


Figure 17: global average pooling

17.7.7 Batch Normalisation

Batch normalisatie wordt gebruikt om het aantal parameters te verkleinen. De uitvoer van een convolutionele laag wordt voor de activatiefunctie genormaliseerd.

- Gemiddelde 0
- Variantie 1

Omdat de activatiefunctie het stekt veranderd bij nul per filter, is er nu geen bias nodig in de convolutionele laag.

```
x = layers.Conv2D(32, 3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
```

Figure 18: Batch normalisatie

17.7.8 Separabele convoluties

In een gewone convolutielaag worden alle filters samen genomen, wat voor veel parameters zorgt. Dus splitsen we de filters en pas je toe op de diepere lagen (niet kleuren maar kleine randjes).

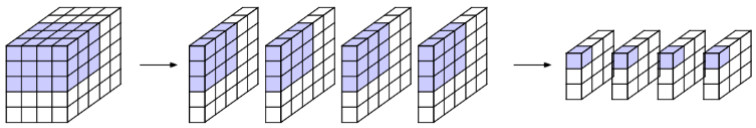


Figure 19: Separabele convoluties

17.7.9 Interpreteerbaarheid

In het algemeen word gezegd dat neurale netwerken black boxes zijn, het is niet echt duidelijk hoe ze tot hun voorspellingen komen. Echter, omdat CNN's visuele informatie verwerken is het hier mogelijk om toch redelijk veel inzicht te krijgen in hoe het model werkt.

- Je kan de activatie van je filters plotten
- Je kan achterhalen op welk patroon een filter maximaal reageert
- Je kan achterhalen welke pixels in een afbeelding de sterkste invloed op de voorspelling hadden

17.7.10 Gebruik pretrained model

```
model = keras.applications.xception.Xception(
    weights="imagenet",
    include_top=False)
layer_name = "block3_sepconv1"
layer = model.get_layer(name=layer_name)
feature_extractor = keras.Model(inputs=model.input, outputs=layer.out
```

Figure 20: Xception aanroepen