

Hogeschool Gent

Samenvatting ML

Curriculum 2021-2022

Dylan Cluyse
19-3-2022

Inhoud

H1:	ML landscape	2
H2:	End to End machine learning	11
H3:	Classificatie	23
H4:	Trainingsmodellen	40
H5:	Support Vector Machines	46
H6:	Decision trees	48
H7:	Ensemble learning	50
H10:	Deep-learning en neurale netwerken	52

H1: ML landscape

ML leert uit ervaring en data:

- Computers die zelfstandig kunnen leren.

ML is niets nieuws. Bestaat al sinds 1960

- Golven/evoluties meegemaakt. Soms hype en soms niets hype.

Deze keer geen hype.

- We beschikken over gigantische hoeveelheden data nu.
- We beschikken over de juiste algoritmen nu.
- We hebben hardware dat krachtig genoeg is in combinatie met Python.

Verskillende stappen:

1. Probleem bestuderen. Wat moet er precies gebeuren?
2. ML-algoritme trainen
 - a. Data aanbieden en het systeem laten leren wat de patronen zijn.
 - b. Zorgen dat de machine weet wat de grenzen en scheidingen zijn.
 - c. Vb: wat is spam en wat is geen spam? Iets wat door de gebruiker wordt geclassificeerd.
3. Oplossing evalueren.
 - a. Nog te veel fouten? Fouten analyseren en terug naar de eerste stap.
 - b. Oplossing: data toevoegen/updaten
4. ML-programma lanceren.

Data mining:


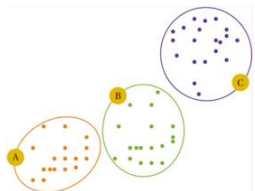

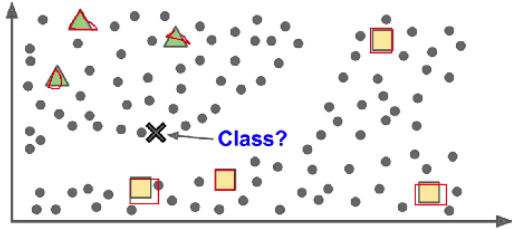
- Omgekeerde aanpak.
- We trainen eerst het algoritme.
- We bekijken de oplossing en leiden af of iets inderdaad spam is (=patronen ontdekken)

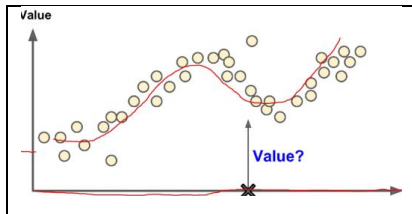
Voorbeelden van toepassingen:

- Tumoren detecteren uit hersenscans
 - Afleiden op basis van hersenscans van mensen die wel of geen diagnose hebben gekregen.
- Sport en financieel nieuws gaan classificeren.
 - Afleiden op basis van onderwerpen, namen van atleten, evenementen.
- Intelligente bot voor een game maken / chatbot
 - Afleiden op feedback van de speler.
- Kredietkaartfraude detecteren
 - Afleiden op basis van **anomalieën**
 - Groot verschil tussen bepaalde waarden.
 - Vb: iemand die uit Duitsland een product koopt op een Amerikaanse webshop met een IJslandse kredietkaart/bank.
- Spam-filter:
 - Je geeft voorbeelden van mails die geflagged zijn door de gebruiker en voorbeelden van gewone of niet spam-mails mee.

Verschillende typen:

We kiezen een type van machinaal leren op basis van de nodige toezicht, de snelheid waaraan data wordt toegevoegd en de mate waarop data wordt gegeneraliseerd.

Supervised	Unsupervised	Reinforcement	Semisupervised learning
<p>= vertrekken van een gelabelde dataset.</p> <p>Dataset labelen is véél werk en grote kans op fouten!</p> <p>Vb: elke e-mail is voorzien van een label (spam of geen spam).</p> <p><u>Twee subklassen:</u></p> <p>Classificatie:</p> <p>Vastgelegde klassen in dataset</p>  <p>Regressie:</p> <p>Geen vastgelegde klassen, maar waarden/getallen.</p> <p>Vb: maximale dagtemperatuur voorspellen a.d.h.v vorige data</p>	<p>= vertrekken van een ongelabelde dataset.</p> <p>= clustering of het samen groeperen van dichtgeplaatste waarden</p> <p>Vb: clusters opstellen van het inkomen en hierop groepen gaan opmaken.</p>  <p>Vb: anomaliedetectie voor kredietkaarten. Fouten uit de dataset halen.</p> 	<p>= leren op basis van fouten en belonen.</p> <p>Water aanraken:</p> <p>Heet/kokend water is niet goed dus in de toekomst niet aanraken</p> <p>Baby dat leert stappen:</p> <p>Vooruitgaan is goed.</p> <p>Vallen is niet goed dus recht blijven.</p>	<p>= tussenvorm van (un)supervised:</p> <ul style="list-style-type: none"> Vb Facebook bij fotoherkenning: persoon aanduiden op foto zonder input van de gebruiker. Baseert zich op andere foto's. <p>Gelabelde dataset: de foto's van de gebruiker</p> <p>Ongelabelde dataset: nieuwe foto's van andere gebruikers</p> <p>Ligt de data dicht bij gelabelde dataset? Match!</p> 

 <p><u>Voorbeelden van algo's:</u></p> <p>K-gemiddelden, lineaire en logistieke regressie, SVM's, neurale netwerken.</p>	<p>Associatie rule learning: vb analyse van het winkelmandje of 'welke producten worden samen gekocht'?</p> <p>Vb controller bij gameconsole of batterijen bij elektronische apparaten.</p>		
--	--	--	--

Batch leren

= dataset wordt geaccumuleerd of constant data toevoegen

- De modellen moeten regelmatig re-trainen o.b.v. de nieuwe data

→ Neemt veel tijd in beslag om te re-trainen.

Het systeem kan niet leren over een stream van data.

Mogelijks te automatiseren.

Ook 'offline learning' genoemd

Online leren

= de training gebeurt incrementeel

Data wordt continu in kleine groepen toegevoegd.

+ Iedere stap is snel en goedkoop

Het systeem leert na iedere data toevoeging.

+ voor ML-systemen die continu data krijgen en data die snel moet aangepast worden zoals vb de beursprijzen

+ wanneer je een beperkt aantal resources hebt voor het systeem

Learning rate: de rate waarop jouw systeem gaat aanpassen over nieuwe data

- Hoge rate: leren worddt snel vergeten
- Lage rate: gelijkaardig aan hoe batch learning werkt

ML-systemen worden gecategoriseerd o.b.v. hoe ze worden gegeneraliseerd. Data generaliseren is kijken hoe het model zich met nieuwe data gedraagt. 'Hoe wordt de voorspelling gemaakt?'

Instantie-gebaseerd

= kijken naar gelijkaardige instanties, welke instanties liggen er het dichtste bij

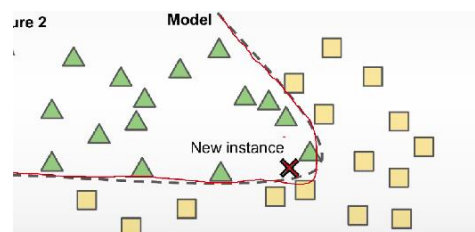
Vb: welke e-mails hebben de meeste gelijkenissen?

Model-gebaseerd

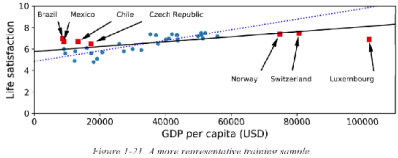
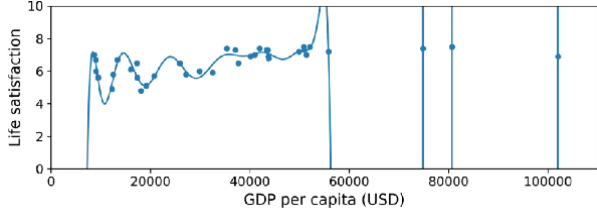
= wiskundig model opstellen en dat gebruiken om de voorspelling te maken

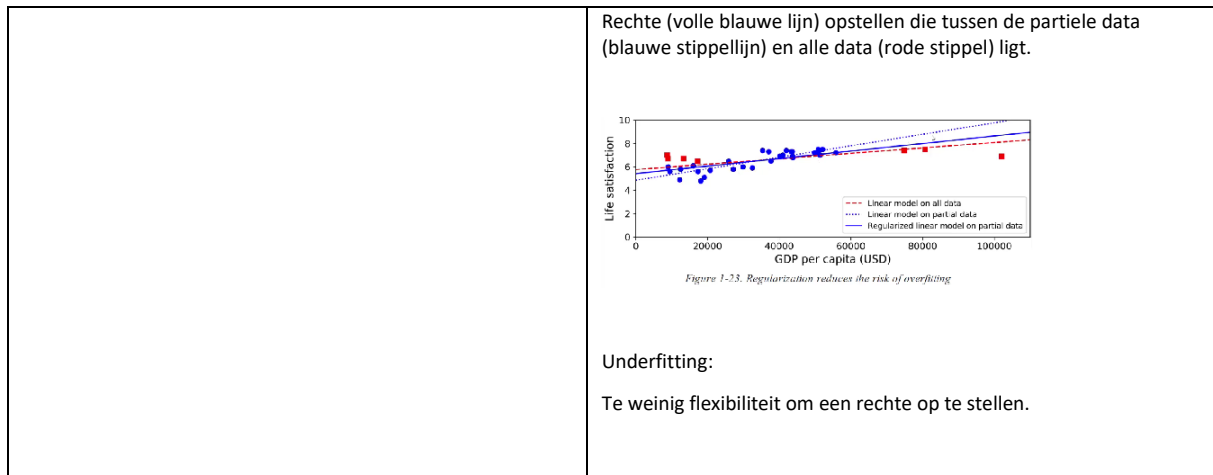
Vb: kijken of er een verband is met het gemiddeld inkomen in een land en het gemiddelde geluk/blijdschap van een land

Life satisfaction berekenen op basis van het gemiddeld inkomen met een trainingsdataset.



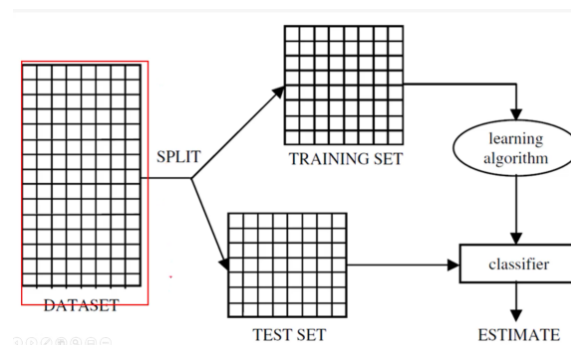
Grootste uitdagingen van ML

<p><u>Onvoldoende hoeveelheid trainingsdata</u></p> <p>= Je hebt een grote hoeveelheid data nodig om conclusies te trekken.</p> <p>Vb: 20 landen vergeleken met het totaal aantal landen wat boven 200 is.</p>	<p><u>Niet-representatieve data</u></p> <p>= Er ontbreekt data wat heel mogelijk de regressie (drastisch) gaat veranderen.</p> <p>Vb: enkel rijke landen gaan gebruiken en zo de arme landen niet betrekken in de dataset.</p> <p>Rode bolletjes werden niet betrokken maar zorgen hier voor een andere helling op de regressie.</p>
<p><u>Slechte kwaliteit van de data</u></p> <p>Bijvoorbeeld menselijke fouten bij de classificatie.</p> <p>Vb: E-mails die géén spam zijn zitten bij de klasse spam-mails.</p> <p>OPL → Zorg voor 'clean' of schone data:</p> <ul style="list-style-type: none"> Vermijd uitschieters. Vermijd instanties die eigenschappen ontbreken, vb gebruiker vulde leeftijd niet in survey. <p>OPL → train een model met de eigenschap en een model zonder die eigenschap.</p>	 <p>→ Leidt tot sampling noise</p> <p>Sampling bias of vooroordelen:</p> <ul style="list-style-type: none"> Belastingsdetectie waar je bijvoorbeeld bij het doen van vorige fraudulente praktijken gaan sneller in een steekproef worden opgenomen. Het profileren van criminele activiteiten waar er wordt gekeken naar etniciteit of achtergrond → over-representatie
<p><u>Irrelevante features</u></p> <p>= zaken die in de data zitten, maar geen belang hebben</p> <p>Wat betekent 'garbage in, garbage out'?</p> <p>= onnodige data wordt ingevuld wat</p> <p>OPL → feature selection of enkel focussen op de meest nuttige features waarop je het model gaat trainen.</p> <p>OPL → feature extraction of bestaande features gaan combineren met andere features om zo een nuttige feature te ontwerpen.</p> <p>OPL → features ophalen vanuit nieuwe data</p>	<p><u>Underfitting en overfitting</u></p> <p>= trainingsdata wordt correct verwerkt, maar de generalisatie gaat verkeerd</p> <p>Trainingsdata wordt 'vanbuiten geleerd'</p> <p>→ Heel goed voor trainingsdata, maar helemaal niet goed voor nieuwe data.</p> <p>Hieronder wordt er opeens een 15^e graadsveelterm gebruikt bij de laatste vier punten:</p>  <p>OPL → meer parameters geven, maar niet te veel want anders heb je 'overfitting'</p> <p>OPM → regularisatie</p>



Hoe evalueren we het model?

- Vertrekken vanuit dataset en splitsen in twee sets:
 - Trainingset: 80%
 - Leeralgoritme wordt hierop gebaseerd
 - Testset: 20%
 - Predict-functie
 - Gekend label per feature
 - Label laten voorspellen door de tabel en kijken hoe goed ze overeenkomen
 - Toont de nauwkeurigheid van het model aan
- Testen en valideren:
 - Testset mag je enkel gebruiken op het einde.
 - Als je testset meeneemt bij het tunen van het model:
 - Niet goed want je zoekt dan een model dat zo goed mogelijk is voor de testset.
 - Zo heb je underfitting. Goede uitslag voor gekende data, geen goede voor ongekende.
 - Dus → Deel van trainingset apart zetten en gebruiken als 'validatieset'.



Cross-validatie:

Hierbij wordt de trainingset in drie delen gesplitst.

Trainingset in drie delen verdelen → drie keer trainen:

- Drie kleinere validatiesets
- Ieder model wordt geëvalueerd op basis van de set

- training duurt langer

+ precisie

+ idealere vergelijkingen

Quiz:

Twee vragen:

Welke types van ML ga je gebruiken?

Welke uitdagingen ga je hebben bij het project?

Bij project Musti:

Types ML:

Supervised learning omdat je vertrekt vanuit een gelabelde dataset. Alle foto's duiden aan of de kat weggaat, stationair staat of binnenkomt. Hier wordt er gewerkt met classificatie waar er dus ook drie klassen zijn.

We gaan werken met batch learning. We passen het model pas aan wanneer we het zelf willen aanpassen, moest het model constant on-the-fly aangepast worden na iedere foto dan zouden we hier moeten gebruik maken van online learning.

Hier zitten we met model-based learning. We moeten zelf een model gaan opstellen.

Uitdagingen:

- Te weinig data? – Niet al te veel foto's maar net genoeg om een conclusie te kunnen trekken. Stel dat er meer dan één kat was met hetzelfde aantal foto's, dan zitten we met een probleem tenzij we voor iedere kat bijvoorbeeld minstens 100 foto's hebben.
- Niet representatieve data? – Lichtinval kan lastig worden, maar dit is in staat om ons model te gaan hertrainen.
- Irrelevante features? – Niet van toepassing hier. Er wordt hier enkel gewerkt met pixels.
- Overfitting? Underfitting? – Bij een te simpel model heb je wel hier een kans op overfitting.

Wat is machine learning?	Het maken van systemen die taken beter en/of sneller kunnen uitvoeren met behulp van data.
Benoem de vier problemen die machinaal leren kan oplossen:	<ul style="list-style-type: none">• Het oplossen van complexe problemen waarvoor we geen algo's hebben.• Het vervangen van lange lijsten met handgeschreven regels.• Systemen bouwen die aanpassen aan snel veranderende omgevingen.• Data mining
Wat is een gelabelde trainingset?	Een dataset waarbij iedere instantie gelabeld is.
Wat zijn de twee vaak voorkomende gesuperviseerde taken?	Regressie en classificatie.
Benoem vier vaak voorkomende ongesuperviseerde taken:	Clustering, visualisatie, dimensionaal reduceren en association rule learning
Welk soort ML-algo heb je nodig om een robot te doen stappen in een ongekende omgeving?	Reinforcement learning is het meest gepaste algo voor deze taak. Dit omdat we te maken hebben met ongekend terrein wat eerder binnen de scope van reinforcement learning valt.
Welk soort ML-algo heb je nodig om je klanten te onderverdelen in verschillende groepen?	Als we de groepen niet kennen, dan kunnen we werken met clustering om zo clusters op te bouwen van klanten. Als we juist wel de verschillende groepen kennen dan kunnen we werken met een classificatie algoritme om zo de klanten in de juiste groep te classificeren.
Behoort spam-detectie eerder tot gesuperviseerd of ongesuperviseerd leren?	Supervising want je gaat het algoritme voeren met emails die al een label hebben.
Wat is een online learning systeem?	Een systeem dat incrementeel of pas op bepaalde momenten gaat leren.
Wat is 'out-of-core' leren?	Hierbij wordt de data in mini-batches verdeeld. Hierop wordt er dan met online learning gewerkt.
Welk soort ML-algoritme maakt voorspellingen op basis van een similarity measure?	Instance-based want hier wordt de data uit het hoofd geleerd. Bij een nieuwe instantie wordt er een voorspelling gemaakt o.b.v. de meest gelijkaardige instantie.

Wat is het verschil tussen een model parameter en een hyperparameter?	<p>Model-parameter bepaalt wat er voorspelt moet worden gegeven een nieuwe instantie.</p> <p>Hyperparameter is een parameter van het leeralgoritme en niet het model. Deze parameter kan je beïnvloeden.</p>
Wat is de strategie van model-based leeralgoritmes?	Er wordt gezocht naar optimale waardes zodat het model gepaste generalisaties kan maken voor nieuwe instanties.
Welke vier uitdagingen zijn er bij ML?	<ul style="list-style-type: none"> • Gebrek aan data, • mindere datakwaliteit, • niet-representatieve data, • niet informatieve features, • te simpele modellen die buiten de dataset vallen • onnodig complexe modellen die buiten de dataset vallen.
<p>Stelling: het model draait goed op trainingsdata, maar het genereert ondermaatse nieuwe instanties</p> <p>Wat gebeurt er? Hoe kan dit opgelost worden?</p>	<p>Het model valt buiten de trainingsdata (=overfitten).</p> <p>We kunnen dit oplossen door:</p> <ul style="list-style-type: none"> • het model te vereenvoudigen • Het aantal parameters te verminderen • Het model te regulariseren
Wat is een testset?	Een testset wordt gebruikt om de fouten te voorspellen dat een model kan maken voor nieuwe instanties.
Wat is het nut van een validatieset?	Een validatieset wordt gebruikt om modellen te vergelijken. Dit maakt het mogelijk om het beste model te kiezen én ook om de hyperparameters te tunen.

Python

```
# Libraries installeren. De belangrijkste libraries voor data-verzameling zijn Numpy en Pandas.
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Data gaan ophalen vanuit een csv-bestand.
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv", thousands=',',
                             delimiter='\\t',
                             encoding='latin1', na_values="n/a")

# Data voorbereiden.
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
country_stats
```

Python

```
# X stelt de data voor. Dit is een matrix van feature-waarden. Elke kolom is een feature.
# y stelt een vector van de waarden die je wilt bepalen voor. Dit is één kolom.
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

# Twee variabelen: life satisfaction en het GDP per capita.
# Data visualiseren met een scatterplot Het GDP (x-as) wordt tegenover de satisfaction (y-as) geplaatst.
# Het GDP wordt op de X-as geplaatst. De satisfaction
country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
plt.show()

# Model voor lineaire regressie kiezen en opslaan onder een variabele
# Dit model gaat een rechte opstellen.
model = sklearn.linear_model.LinearRegression()

# model trainen, geef de inputvariabelen (X) en de nieuwe variabelen (y) mee
model.fit(X, y)

# Een eerste voorspelling maken voor het GDP van Cyprus.
# Gebruik de predictfunctie om met een meegegeven waarde een andere waarde te gaan zoeken/bereken.
X_new = [[22587]] # Cyprus' GDP per capita
print(model.predict(X_new)) # output = [[ 5.96242338]]
```

✓ 0.2s

Python

```
# Een model voor K-nearest neighbours gaan kiezen en opslaan onder een variabele.
# Er werd gegeven dat K gelijk moet staan aan 3. Dit geef je als parameter mee.
import sklearn.neighbors
model1 = sklearn.neighbors.KNeighborsRegressor(n_neighbors=3)

# Model trainen.
model1.fit(X,y)

# Zelfde voorspelling maken hier.
print(model1.predict(X_new)) # output = [[5.76666667]]
```

✓ 0.1s

Python

H2: End to End machine learning

Kadering van het probleem, welke vragen moeten we ons afstellen:

1. Wat is de business objective?
2. Wat is de huidige oplossing?

De business objective

- Model bouwen != einddoel
- Belangrijk: Hoe wilt het bedrijf het model gaan gebruiken?
- Welke meettechnieken wil je gebruiken om het model te evalueren?
Hoeveel inspanning wil je leveren om het te tweakken?

Data pipeline

- Een opeenvolging van data processing components.
- Gewoonlijke zaak in ML – veel data die bijgewerkt en getransformeerd moet worden
- Component:
 - Asynchroon en self-contained (=interface er tussen is enkel de data-opslag)
 - Neemt grote hoeveelheid data op en verwerkt het
- Als een component breekt
 - Laatste output wordt gebruikt en zo blijven de downstream componenten werken
 - Wordt vaak niet opgemerkt – data is niet meer ‘vers’ + snelheid van systeem zakt

De oplossing

- Referentie voor de performantie + geeft inzicht op hoe je het probleem moet oplossen
- Vb: Welke regels worden er nu gebruikt om de prijzen te schatten? (kan manueel zijn vb)
- Eigen model ontwerpen:
 1. Welke soort learning? Supervised, unsupervised, reinforcement, etc.
 2. Welk soort taak? Classificatie, regressie, etc.
 3. Batch of online learning?
- Model ontwerpen voor de housing prices:
 1. Supervised learning want je hebt hier gelabelde voorbeelden. Elke instantie heeft een verwachte output.
 2. Regressie want je moet een waarde voorspellen.
 - a. ‘Multiple regression’ – het systeem moet meerdere kenmerken gebruiken om een waarde te voorspellen.
 - b. ‘Univariate regression’ – er moet één waarde voorspeld worden voor iedere district
 3. Batch learning is voldoende want hier heb je geen continue toevoer van data.

Het (regressie)model evalueren:

RMSE of gemiddelde kwadratische afwijking	MAE of de absolute waarde van de error
<ul style="list-style-type: none"> • Meet de afstand tussen twee vectoren • Hoe lager de waarde, hoe beter de dataset getraind kan worden met het model. • De vierkantswortel van de gemiddelde kwadratische afwijking. • Toont aan hoeveel fouten of afwijking het systeem maakt in voorspellingen. • Het verschil tussen de werkelijkheid en wat wij voorspellen. • Maatstaf om te achterhalen hoe goed ons model is • Vierkantswortel zorgt voor een realistische afwijking. • Zoekt naar het kleinste getal. Het omgekeerde hiervan is de negative mean square error. $RMSE(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$	<ul style="list-style-type: none"> • Absolute waarde van de afwijking • Geeft aan hoe groot de fout is die we kunnen verwachten op het voorspelde gemiddelde. • Gebruik je bij veel outliers • Hoe lager het percentage, hoe beter het model. $MAE = \frac{1}{n} \sum_{j=1}^n y_j - \hat{y}_j $

m: het aantal instanties in de dataset waarop je de RMSE gaat berekenen	y(i): label van de instantie
X: Een matrix met alle waarden van alle instanties in de dataset. Per rij is er één instantie. De i' de rij is x(i)	h: de hypothese of voorspellende functie $\hat{y} = h(x(i))$

Waarom veronderstellingen maken?

- Lijst en verifieer alle veronderstellingen die je tot nu toe hebt gemaakt
 - Zo problemen early-on vermijden
 - Vb: onderscheid maken tussen 'klein', 'middel' en 'groot' bij opp.
 - Eerst regressie (waarde in cm) → classificatie
 - Model maken voor regressie en aanpassen naar een model voor classificatie = niet evident!

Stappenplan

1. Data ophalen en workspace maken

Vereiste modules:

- Numpy: wiskundige methoden
- Pandas: het behandelen en bewerken van data
- Matplotlib: visualisaties maken in python
- Scikit: machine learning en statistisch modelleren

2. Data downloaden

- Geëncrypteerd bestand downloaden -- .tgz bestand.
- Bij een .csv-bestand – kijk welke delimiter er is
 - Tab-toets, puntkomma, komma, etc.
- Gebruik functies bij voorkeur in plaats van zelf uitpakken!
 - Draagt bij tot de automatisatie van het model.

3. Data visualiseren

- Scatterplot gebruiken om districten te visualiseren

Kind	Soort plot
X,Y	Wat moet er op de as getoond worden,
Alpha	Hoe groot is de drukte?
S=housing["population"]	Hoe groot is de radius van de bol? – hier gebaseerd op populatie
S="median_house_val"	Wat toont de kleur aan? – hier gebaseerd op prijs
Cmap	Pre-made colormap (blauw – rood)

4. Kijk voor overeenkomsten of correlaties:

De dataset is nog matig groot. Je kan Pearson's R of de standaardovereenkomstcoëfficiënt berekenen met de 'corr()' methode.

- Coëfficiënt varieert van -1 t.e.m. 1
- Hoe dichter bij 1 – hoe sterk de overeenkomst
 - De gemiddelde huisprijs gaat op wanneer het gemiddeld inkomen naar beneden gaat
- Hoe dichter bij -1 – hoe sterk dat er juist géén overeenkomst is
- Hoe dichter bij 0 – toont aan dat er géén lineaire overeenkomst is
- OPM – enkel lineaire overeenkomsten worden gemeten

```
#matrix opbouwen
corr_matrix = housing.corr()

#attributen tegenover de gemiddelde huisprijs zetten, hoe zijn de overeenkomsten tussen de twee? Sorteer van 'zeer gelijk' naar 'helemaal niet gelijk'.
Corr_matrix["median_house_value"].sort_values(ascending=False)
```

Een andere manier om overeenkomsten te checken is met de '**scatter_matrix()**' functie van pandas.

- **Plot ieder numerieke waarde tegenover elke andere numeriek attribuut.**
- Bij 11 numerieke attributen – 121 mogelijke uitkomsten wat niet haalbaar is op een page.
- Toont een histogram van ieder attribuut, andere opties zijn mogelijk

```
# scatter_matrix importeren
from pandas.plotting import scatter_matrix

# attributen meegeven waarvan de correlaties getoond moeten worden
# ieder attribuut wordt tegenover elkaar geplaatst
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))

#figuur opslaan
save_fig("scatter_matrix_plot")

# scattermatrix gebruiken om een attribuut tegenover een gegeven attribuut te plaatsen
housing.plot(kind="scatter", x="median_income", y="median_house_value",
              alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```

Experimenteren met attribuutcombinaties:

- Tail-heavy data oplossen door te transformeren met bijvoorbeeld een algoritme.
- Eerst experimenteren vooraleer je de data van het ML-algo klaarzet
 - Hoeveel slaapkamers zijn er per huishouden?
 - Hoeveel slaapkamers zijn er ten opzichte van het aantal kamers?

```
# verhouding aantal kamers / aantal huishoudens
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]

# verhouding aantal slaapkamers / aantal kamers
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
```

5. Data klaarzetten voor ML-algo's

Waarom functies gebruiken?

- Goede stap richting automatiseren en bij het krijgen van een nieuwe dataset
- Bouw een 'bibliotheek' aan transformatie functies die je kan gebruiken bij latere projecten
- Gebruik deze functies in een live-systeem om de nieuwe data te transformeren voordat je het geeft aan algo's.
- Laat je toe om verschillende transformaties te maken en te zien welke combo's het beste werken.

```
# Terugzetten naar een clean training set.
# We 'droppen' de labels en predictors -
# - dit om geen transformaties toe te passen op de predictors en doelwaarden
# .drop() gaat maakt een kopie aan van de data
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

6. Data cleaning:

Wat doe je bij missende waarden? Drie opties:

- Overeenkomstige districten verwijderen
- De attribuut verwijderen
- De waarden met een nulwaarde – het gemiddelde, mediaan, etc – vullen.
 - OPM: bereken het gemiddelde op de trainingset!
 - Vergeet niet om het gemiddelde op te slaan!

```
# maak een kopie van de housing dataset, maar behoud enkel de
rijen met minstens één null
# zo kan je beter zien wat iedere optie doet
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
# optie 1
sample_incomplete_rows.dropna(subset=["total_bedrooms"])
# optie 2
sample_incomplete_rows.drop("total_bedrooms", axis=1)
# optie 3
median = housing["total_bedrooms"].median()
# overall waar er geen waarde is --> vul m.b.v. de berekende mediaan
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True)
```

Extra optie: gebruik SimpleImputer om de waarden op te vullen.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
# maak een kopie van de data ZONDER het text-attribuut 'ocean-proximity'
housing_num = housing.drop("ocean_proximity", axis=1)
# alternatively: housing_num = housing.select_dtypes(include=[np.number])
# maak een kopie van de data ZONDER het text-attribuut 'ocean-proximity'
housing_num = housing.drop("ocean_proximity", axis=1)
# alternatively: housing_num = housing.select_dtypes(include=[np.number])
# gebruik de library om de trainingdata te 'fitten' met de data zonder ocean_prox → wordt
opgeslagen onder 'statistics_'
imputer.fit(housing_num)
```

```

imputer.statistics_
# ter controle: dit gaat de mediaan weergeven van ieder attribuut
housing_num.median().values
# vervang nu alle missende waarden met de 'geleerde' gemiddelden
X = imputer.transform(housing_num)
# het resultaat (zonder null-waarden) terug in de dataframe plaatsen
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                           index=housing.index)

```

7. Overweg gaan met tekst en categorische attributen:

Niet-numerieke attributen vereisen een andere aanpak. Hieronder zijn er een beperkt aantal mogelijkheden mogelijk – omzetten naar getallen/waarden

```

# toon de eerste tien instanties en focus enkel op de ocean proximity
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
#library importen
from sklearn.preprocessing import OrdinalEncoder
# encoder aanmaken waar we een getal linken met een categorie
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
# voor tien waarden
housing_cat_encoded[:10]
# toon alle mogelijke categoriën --> 1H ocean, inland, island, etc.
ordinal_encoder.categories_

```

8. Nominale variabelen omzetten naar ordinale variabelen (indien nodig)

Probleemstelling: We willen extra belang hechten aan de afstand. Nu hebben we los de verschillende afstanden staan zonder enige ordening. Bijvoorbeeld 'dichtbij zee' naar 'binnenland', maar er wordt hier geen orde bijgehouden.

- Sparse naar dense– dit doen we om geheugen te besparen.
 - Sparse – houdt veel nullen bij, enkel een 1 per rij
 - Dense – houdt enkel de niet-nullen bij – bespaart zo geheugen
- Alternatieve optie – verander de waarden naar een numerieke waarde.
 - Bijvoorbeeld de afstand tot de kust bijhouden in plaats van een categorie

```

# andere encoder importeren
from sklearn.preprocessing import OneHotEncoder
# encoder aanmaken die rekening houdt met de ordening
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
# converteren van een sparse array naar een dense array
housing_cat_1hot.toarray()
# categoriën weergeven
cat_encoder.categories_

```


9. Custom transformers:

- Data transformeren – soms speciale attributen mixen waar je een eigen transformer moet maken
- Klasse maken + drie methoden: fit(), transform() en fit_transform()

```
from sklearn.base import BaseEstimator, TransformerMixin

# indexen van de attributen
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

# klasse om extra kolommen toevoegen
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    # constructor -- aantal slaapkamers per kamer, willen we deze kolom of niet?
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kwargs
        self.add_bedrooms_per_room = add_bedrooms_per_room

    # methode die je meekrijgt, geen aanpassing nodig
    def fit(self, X, y=None):
        return self # nothing else to do

    # X = dataset, alle rijen uit X kiezen
    def transform(self, X):
        # kamers per huishouden berekenen uit de dataset
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        # populatie per huishouden berekenen uit de dataset
        population_per_household = X[:, population_ix] / X[:, households_ix]
        # hebben we slaapkamers meegekregen?
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            # alles 'aan elkaar hangen'
            return np.c_[X, rooms_per_household, population_per_household,
                          bedrooms_per_room]
        # no bedrooms?
        else:
            return np.c_[X, rooms_per_household, population_per_household]

# initieren ZONDER slaapkamers
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)

# housing.values moeten we meegeven -- switchen naar numpy array
housing_extra_attribs = attr_adder.transform(housing.values)
```

10. Feature scaling:

- ML algo's draaien niet goed wanneer numerieke attributen verschillende schalen hebben
- Bij housing data—het aantal kamers varieert van 6 tot 39320 en gemiddeld inkomen is van 0 tot 15
- Opmerking: gebruik feature scaling enkel (!) op de trainingsdata

Er zijn twee manieren om alle attributen op eenzelfde schaal te krijgen.

<u>Min-max scaling</u>	<u>Standardization</u>
Normalisatie van de waarden. Waarden worden opnieuw geschaald zodat ze tussen 0 en 1 liggen. De min-waarde verminderen en delen door (max-min). → <code>MinMaxScaler()</code>	Verminder de gemiddelde waarde en deel door de standaardafwijking. De verdeling gaat binnen een specifieke range horen. Uitschieters gaan de nauwkeurigheid van het model minder gaan bepalen. → <code>StandardScaler()</code>

11. Transformation Pipelines:

- Volgorde van de transformers is belangrijk – pipeline voorzien

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# constructor vereist -- lijst van paren die de volgorde tonen
# alles behalve de laatste moet een transformer zijn (bevat dus 'fit_transform()')
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

# alle fit() methodes van elke transformer wordt aangesproken
# gebeurt opeenvolgend of sequentieel
# begint bij SimpleImputer -> CombinedAttr.. -> StandardScaler
housing_num_tr = num_pipeline.fit_transform(housing_num)

# array weergeven
housing_num_tr
```

Een transformer die de pipeline van alle kolommen gaat behandelen rekening houdend met de verschillende soorten kolommen:

```
# lib importeren
from sklearn.compose import ColumnTransformer

# lijst van alle numerieke kolommen en alle categorische kolommen (ocean_prox)
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

# één transformer die kolommen gaat behandelen
# vraagt een lijst van tupels op -
# - bevat naam, transf. en lijst van kolommen waar transf. moet bezig houden
full_pipeline = ColumnTransformer([
    # alle numerieke kolommen -- numerieke pipeline
    # num_pipe -- geeft een dense matrix
    ("num", num_pipeline, num_attribs),
    # alle categorische kolommen -- OneHotEncoder
    # OHE -- geeft een sparsematrix
    ("cat", OneHotEncoder(), cat_attribs),
])

# transformeren
housing_prepared = full_pipeline.fit_transform(housing)
```

12. Een model kiezen en trainen:

```
from sklearn.linear_model import LinearRegression

# object aanmaken en hierop de fit-functie aanspreken
# dataset en labels meegeven
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

# voorspelling maken
# haal data en de labels op uit de dataset
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
# data die je wilt gaan voorspellen
some_data_prepared = full_pipeline.transform(some_data)

# gebruik de predict()-functie van het lineaire regressie-object
print("Predictions:", lin_reg.predict(some_data_prepared))

# bereken de RMSE op de volledige trainingsset met de functie van Scikit
from sklearn.metrics import mean_squared_error
```

```

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse # 68627.87390018745 als uitkomst wat veel te weinig is, dit zou tussen 120k en 260
k moeten liggen

# alternatief op RMSE
# krachtig model om complexe niet-lineaire relaties in de data te vinden
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)

# model is getrained nu
# evalueer het model op de trainingsset
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse # geslaagd model ==> result: 0.0

```

13. Cross-validatie:

- Betere vorm van evaluatie
- Scikit's K-fold cross-validation feature
 - Splitst de trainingset in tien verschillende subsets (=folds)
 - Traint en evalueert het model tien maal
 - Voor iedere fold wordt een andere evaluatie gebruikt, op de andere negen folds wordt getrained
 - Resultaat = tien evaluatiescores
- Cross-validatie op het lineair model, tree-model en randomforest
 - Randomforest = 'ensemble learning' ofwel een model bovenop meerdere modellen

```
from sklearn.model_selection import cross_val_score

# functie aanroepen
# cv: bepaalt welke splits-strategie we moeten gebruiken
# scoring: de manier van scoring meegeven
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                          scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)

# scores tonen waaronder gemiddelde en std (= hoe precies is het model?)
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
display_scores(tree_rmse_scores)

# dezelfde scores berekenen voor het lineair regressiemodel
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                              scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)

#RandomForest of een model dat zoveel mogelijk decision trees gaat trainen op verschillende,
willekeurige subsets → hiervan dan het gemiddelde nemen
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

14. Overfitten oplossen:

- Model versimpelen
- Grenzen zetten op model
- Meer trainingsdata verkrijgen
- Hou het aantal modellen beperkt (hoogstens vijf)
- Gebruik de pickle of joblib module om modellen op te slaan

15. Model fine-tunen:

De beste parameters achterhalen kan je doen aan de hand van een 'search'. Hiervoor zien we in de cursus randomized en grid search.

Grid-search:

- Meegeven welke hyperparameters je mee wilt experimenteren en welke waarden je wilt uitproberen
- Gebruikt cross-validatie om alle mogelijke combinaties te evalueren
- Sommige data-preparation stappen kan je instellen als hyperparameters
- Niet handig bij veel te grote hyperparameter search space.

```
from sklearn.model_selection import GridSearchCV
# zoeken naar de beste combo's van hyperparameters voor de RandomForestRegressor
param_grid = [
    # probeer 3 x 4 combinaties met bootstrap op True (standaard)
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # probeer 2 x 3 combo's met de bootstrap op False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]
forest_reg = RandomForestRegressor(random_state=42)
# voer 18 (12 + 6) combinaties uitvan de RFR hyperparameter waarden
# train iedere model vijf maal (cv=...) --> 18 x 5 dus 90 keer trainen
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
# haal de beste combinatie van parameters op
grid_search.best_params_
# vraag de beste estimator op
grid_search.best_estimator_
# toon de evaluatiescores
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

Randomized search

- Handiger bij een grote hyperparameter zoekklasse.
- Een gegeven aan verschillende, willekeurige combo's wordt geëvalueerd
- Door het aantal iteraties vast te zetten ga je meer controle hebben over de computing budget

```

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distributions = {
    'n_estimators': randint(low=1, high=200),
    'max_features': randint(low=1, high=8),
}

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distributions,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error',
                                random_state=42)
rnd_search.fit(housing_prepared, housing_labels)

```

16. Ensemble methods:

- Combineer methoden die het best draaien
- De groep gaat beter performen dan het beste individuele model
 - De individuele modellen maken verschillende type fouten.

17. Analyseer de beste modellen en hun fouten:

- Je krijgt een goed inzicht door de beste modellen te bekijken
- RFR – toont relatieve belang van ieder attribuut aan bij het maken van nauwkeurige voorspellingen.
- Kijk naar de specifieke fouten die je systeem maakt.
 - Waarom maken ze het? Wat doen we om het op te lossen? Vb nieuwe features toevoegen of niet-informatieve features verwijderen.

18. Evalueer het systeem op basis van de testset:

- Haal de predictors en labels uit de testset
- Draai de 'full_pipeline' om de data te transformeren
- Evalueer het finale model op de testset

```

final_model = grid_search.best_estimator_
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

```

Vergeleken met cross-validatie → mindere snelheid

- Dit omdat jouw systeem gefinetuned is om goed te draaien op de validatiedata
- Gaat niet goed performen op ongekende datasets
- Geen moeite doen om hyperparameters te tweaken – aanpassingen gaan geen nieuwe data maken.

19. Launch, monitor en maintain your system

- Sla de Scikit-Learn model op inbegrepen de volledige preprocessing en prediction pipeline
- Laad het getrainde model op in de production environment en gebruik de 'predict' functie
- Vb: je model staat op een website en een gebruiker wilt deze raadplegen:
 - De user wilt een prijs schatten – query wordt gestuurd naar de webserver
 - Predict-methode wordt dan aangeroepen.
 - Resultaat wordt weer teruggestuurd naar de web applicatie.

H3: Classificatie

Twee soorten gesuperviseerde taken:

- Regressie: H2 met het voorspellen van huisprijzen
- Classificatie: H3

Mnist-dataset:

- 70k afbeeldingen van handgeschreven cijfers
- Ieder img heeft een label met het getal

```
# Dataset ophalen -- staat openbaar beschikbaar
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
```

Iedere dataset van Scikit heeft een gelijke dictionary-opdeling bestaande uit:

- DESCR-key
- Data key met per rij een instantie en een kolom per feature
- Target key met een array van alle labels

```
# twee arrays, de data en de meegegeven labels
X, y = mnist["data"], mnist["target"]
X.shape # 70 000 afbeeldingen
y.shape # afbeelding is 28 x 28 --> 784 features
```

Reshape nu alle instantie feature vectoren naar een 28x28 array

- Toon het met de 'imshow()' methode

```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

# eerste getal nemen uit de array met alle foto's
some_digit = X[0]

# reshape() functie aanspreken
some_digit_image = some_digit.reshape(28, 28)

# afbeelding tonen
plt.imshow(some_digit_image, cmap=mpl.cm.binary)

# standaard wordt er een x,y-as getoond, die willen we niet weergeven
plt.axis("off")
```


Kijken naar de labels en het omzetten van de labels naar de juiste datatype.

Hieronder gaan we de afbeeldingen gaan hervormen naar een 'grid' of dubbele array van 28 op 28.

```
# label tonen van de afbeelding
y[0]

# staat nu momenteel als een 'string' ingesteld
# belangrijk dat je de label cast naar een integer
y = y.astype(np.uint8)

def plot_digits(instances, images_per_row=10, **options):
    # grootte van de afbeelding
    size = 28

    # aantal afbeeldingen per rij instellen, wordt nu 10x10
    images_per_row = min(len(instances), images_per_row)
    # This is equivalent to n_rows = ceil(len(instances) / images_per_row):
    n_rows = (len(instances) - 1) // images_per_row + 1

    # Voeg lege afbeeldingen toe op het einde van het grid als dit nodig is:
    n_empty = n_rows * images_per_row - len(instances)
    padded_instances = np.concatenate([instances, np.zeros((n_empty, size * size))], axis=0)

    # Reshape de array zodat het gestructureerd is als een grid met 28x28 afbeeldingen:
    image_grid = padded_instances.reshape((n_rows, images_per_row, size, size))

    # combineer de verticale grid as (=0) en de verticale image-as (=2)
    # en combineer de twee horizontale assen (=1, 3)
    # we moeten de assen verplaatsen zodat we ze naast elkaar kunnen combineren
    # combineren --> transpose()
    big_image = image_grid.transpose(0, 2, 1, 3).reshape(n_rows * size,
                                                         images_per_row * size)

    # afbeelding gemaakt --> nu tonen:
    plt.imshow(big_image, cmap = mpl.cm.binary, **options)
    plt.axis("off")

# toon een 9x9 figure
plt.figure(figsize=(9,9))

# haal 100 voorbeeldafbeeldingen uit de dataset
example_images = X[:100]

# gebruik de vbafbeeldingen en geef mee hoeveel afbeeldingen per rij
plot_digits(example_images, images_per_row=10)

# sla figure op en toon met plt()
save_fig("more_digits_plot")
plt.show()
```

Testset moet eerst aangemaakt worden!

- Mnist dataset is vooraf al gesplitst in een trainingset (60k afbeeldingen) en een testset (10k afbeeldingen)
- Trainingset is vooraf al *geshuffled* -> garandeert cross-validatie folds die gelijkaardig zijn

- Sommige leeralgo's zijn gevoelig aan de rangschikking van de trainingsinstanties
 - Draaien slecht als ze meerdere gelijkaardige instanties krijgen na elkaar

```
#afbeeldingen toekennen aan de datasets (X) en de labels (y)
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Een binaire classifier testen

- Probleem simpeler maken – enkel één getal identificeren
 - Binary classifier: twee klassen → 5 en niet-5

```
y_train_5 = (y_train == 5) # True voor alle 5'en, False voor alle andere getallen
y_test_5 = (y_test == 5)
```

SGD of Stochastic Gradient Descent

- Classifier die lineaire classifiers en regressors gaat trainen.
- Op zich geen model, maar een **manier om een model te trainen**.
 - SGDClassifier en SGDRegressor zijn wél modellen.
- Plus: kan grote datasets aan omdat SGD de trainingsinstanties apart behandelt
- Geschikt voor online learning
- Gevoelig aan feature scaling & verrgt hyperparameters zoals het aantal iteraties en de regularisatieparameter.

```
from sklearn.linear_model import SGDClassifier
# max-iter - het maximal getal van passes over de trainingsdata
# tol - tolerantie-niveau
sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
# fitten
sgd_clf.fit(X_train, y_train_5)
# sgd gebruiken om nu een afbeelding van het getal 5 te herkennen
sgd_clf.predict([some_digit]) #geeft in dit geval True
```

Performance measures

- Evalueren van een classifier is moeilijker dan het evalueren van een regressor
- Meerdere mogelijkheden beschikbaar

Nauwkeurigheid meten met **cross-validatie**

- Cross-validation implementeren
 - Bij K-fold cross validation gaan we de trainingset in K-aantal folds splitten (->3)
 - Hierop een voorspelling/evaluatie maken voor iedere fold

```
# cross-validatiescore gaan berekenen
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

```
# optioneel - dit moet je gebruiken om Cross-Val unieker en toepasselijker
# te maken in andere cases
# doet hetzelfde als de 'cross_val_score()' en print hetzelfde resultaat
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
```

```
# voert stratified sampling uit om folds aan te maken met een gelijkwaardige/representatieve
ratio tussen de verschillende klassen
# m.a.w. even veel van elke klasse
skfolds = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# bij iedere iteratie -- kloon trainen + voorspelling maken op de test-fold
# daarna het aantal juiste voorspellingen optellen
# geeft de ratio terug van het juiste aantal voorspellingen
for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))
```

Nauwkeurigheidpercentages zijn niet de meest ideale meetmethode voor classifiers.

- Zeker bij skewed datasets (waar sommige klassen meer frequent voorkomen dan andere)

```
# klasse maken die iedere afbeelding (niet-5 imgs) gaat classificeren
from sklearn.base import BaseEstimator
class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
# nieuw object aanmaken en gebruiken in de cross_val_score
never_5_clf = Never5Classifier()
cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
# uitkomst is een array met daarin drie acc. percentages
```

Confusion matrix

- Een betere manier om de performnce van een classifier te evalueren
- Tel het aantal instanties wanneer een klasse wordt herkend als een andere klasse
 - Vb: wanneer app een '5' als een '3' ziet – dan moet je in de 5^{de} en 3^{de} kolom kijken
- Vooraf:
 - Set van voorspellingen nodig
- Cross_val_predict geeft de voorspellingen terug gemaakt op iedere test fold.
 - Cleane voorspelling voor iedere instantie in de trainingset
- Iedere rij in de matrix = eigenlijke klasse | iedere kolom = voorspelde klasse
 - Perfecte matrix = enkel niet-nul getallen op de diagonale zijde van de matrix
 - (Van links boven naar rechts onder)

```
# voorspellingen maken
from sklearn.model_selection import cross_val_predict
```

```

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)

# (links) bouw de confusiematrix op en
# geef de trainingset van 5 en de voorspellingen mee

from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5, y_train_pred)

# (rechts) stel dat we een 100% perfecte voorspelling hebben
y_train_perfect_predictions = y_train_5
confusion_matrix(y_train_5, y_train_perfect_predictions)

```

```

array([[53892,   687],
       [ 1891, 3530]])

```

```

array([[54579,    0],
       [    0, 5421]])

```

Formules binnen de confusiematrix:

- Precision wordt samen gebruikt met recall
- Recall = de ratio van het aantal positieve instanties

$\frac{TP}{TP + FP}$	Precisie <ul style="list-style-type: none"> TP = het aantal ware positieve getallen FP = het aantal valse positieve getallen
$= \frac{TP}{TP + FN} = \frac{TP}{P}$	Recall / sensitivity of true positive rate <ul style="list-style-type: none"> TP = het aantal ware positieve getallen FN = het aantal valse negatieven

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

Confusion matrices met een dimensie groter dan 2x2

- Rijen stellen hier de klassen voor terwijl de kolommen de voorspelde klassen voorstellen.
- Confusiematrix analyseren geeft je inzicht in hoe je de classifier kan verbeteren.
 - Hier moet je rekening houden met de data van 8. Verzamel meer data of verminder het aantal valse 8'en.

```

# confusiematrix opbouwen en hierop de precisiescore berekenen
cm = confusion_matrix(y_train_5, y_train_pred)
cm[1, 1] / (cm[0, 1] + cm[1, 1])

# precisiescore berekenen o.b.v. trainingset en voorspelde set
from sklearn.metrics import precision_score, recall_score
precision_score(y_train_5, y_train_pred)

```

```
# Wat is de recall? Hoeveel positieve instanties zijn er?
recall_score(y_train_5, y_train_pred)

# via de confusiematrix berekenen
cm[1, 1] / (cm[1, 0] + cm[1, 1])
```

F1-score:

- 'Harmonisch gemiddelde van de precisie en recall-score'
 - Meer impact van de lagere waarden
 - Enkel een hoge score als de recall en precisiescore hoog zijn
- Voordeelt classifiers die een gleijke precisie en recall hebben
 - Niet altijd wat je nodig hebt
 - Bvb: een app die videos herkent die veilig zijn voor kinderen
 - Hier wil je juist de goede videos vermijden (=lage recall) en enkel de veilige behouden (hoge precisie).
 - Bvb: classifier die shoplifters herkent
 - Ok als je een lage precisie hebt zolang dat je een hoge recall hebt
 - Hoe hoger de recall, hoe meer shoplifters je zal kunnen pakken.

```
from sklearn.metrics import f1_score
# f1-score berekenen volgens de functie
f1_score(y_train_5, y_train_pred)
# f1-score berekenen volgens de confusiematrix
cm[1, 1] / (cm[1, 1] + (cm[1, 0] + cm[0, 1]) / 2)
```

Trade-off tussen precisie en recall:

- Voor iedere instantie wordt er een score gemaakt op een beslissingsfunctie
- Als de score groter is dan de threshold – instantie toekennen aan de positieve klasse
 - Anders naar een negatieve klasse
- Decision threshold naar rechts verplaatsen:
 - Kan precisie verhogen -- maar mogelijks recall omlaag halen
 - Omgekeerd ook mogelijk (threshold naar links)
 - Niet mogelijk om deze te verplaatsen binnen SciKit

```
# spreek de decision fnctie aan
# score voor iedere instantie wordt teruggegeven
y_scores = sgd_clf.decision_function([some_digit])
y_scores
# threshold instellen op nul en hierop voorspellingen maken
threshold = 0
y_some_digit_pred = (y_scores > threshold)
# resultaat is True want de voorspellingen blijven hetzelfde
# alles is boven nul dus geen verschil
y_some_digit_pred
# threshold verhogen naar 8000
threshold = 8000
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred # gaat False zijn -- recall is verlaagd
```

Hoe bepaal je de juiste threshold?

- Alle scores ophalen van de instanties uit de trainingset
- Bereken de precisie en recall voor alle mogelijke thresholds
- Plotten en visueel aanschouwelijk maken

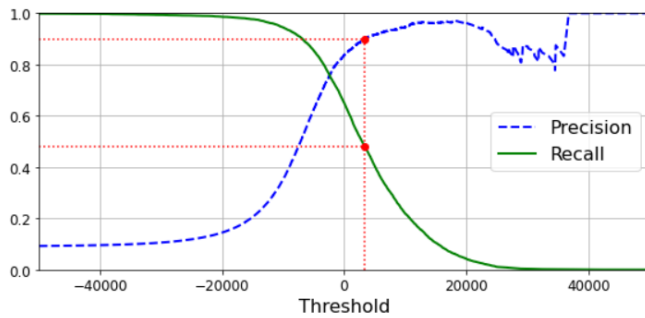
```
# hoe bepaal je de juiste threshold?
# cross_val_predict om alle scores te krijgen van instanties in de trainingset
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")

# bereken de juiste precisie en recall voor alle mogelijke threshold
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)

# deze functie gaat de precisie en threshold tegenover elkaar gaan plaatsen
# er wordt een punt aangeduid waar de ideale threshold zich bevindt
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
    plt.legend(loc="center right", fontsize=16) # Not shown in the book
    plt.xlabel("Threshold", fontsize=16)      # Not shown
    plt.grid(True)                           # Not shown
    plt.axis([-50000, 50000, 0, 1])          # Not shown

# threshold bepalen waarbij de precisie boven 90% ligt
recall_90_precision = recalls[np.argmax(precisions >= 0.90)]
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]

# visuele details
plt.figure(figsize=(8, 4))
# Not shown
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.plot([threshold_90_precision, threshold_90_precision], [0., 0.9], "r:")
# Not shown
plt.plot([-50000, threshold_90_precision], [0.9, 0.9], "r:") # Not shown
plt.plot([-50000, threshold_90_precision], [recall_90_precision, recall_90_precision], "r:") # Not shown
plt.plot([threshold_90_precision], [0.9], "ro")
# Not shown
plt.plot([threshold_90_precision], [recall_90_precision], "ro")
# Not shown
save_fig("precision_recall_vs_threshold_plot")
# Not shown
plt.show()
```

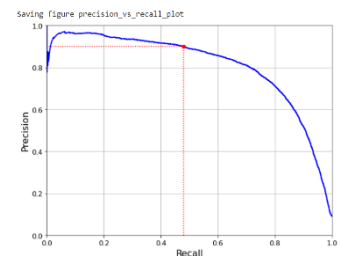


Waarom is de precisiecurve opeens 'bumpy' vergeleken met de recall-curve?

- Precisie gaat soms omlaag wanneer je de threshold doet stijgen
- Kijk wat er gebeurt als je bijvoorbeeld begint vanuit de centrale threshold en met één cijfer naar rechts verplaatst.
 - Precisie gaat van 80 -> 75
 - Recall gaat enkel dalen wanneer de threshold stijgt

Je kan de precisie t.o.v. de recall bekijken:

```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])
    plt.grid(True)
```



```
plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.plot([recall_90_precision, recall_90_precision], [0., 0.9], "r:")
plt.plot([0.0, recall_90_precision], [0.9, 0.9], "r:")
plt.plot([recall_90_precision], [0.9], "ro")
save_fig("precision_vs_recall_plot")
plt.show()
```

- Je ziet dat de precisie fors begint te dalen vanaf een recall van 80%.
 - Trade-off vinden voordat het sterk begint te dalen – hier op ~40%
- Op deze manier kan je prioriseren waar je vooral op wilt focussen.
 - 90% precisie – 42% recall
 - 90% recall – 45 % precisie

```
# np.argmax - geeft je de eerste index van de maximale waarde ofwel de eerste 'True' waarde
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
threshold_90_precision # resultaat = 3370.019
# voorspelling maken o.b.v. de trainingset
y_train_pred_90 = (y_scores >= threshold_90_precision)
# wat zijn de scores van de precisie en recall?
precision_score(y_train_5, y_train_pred_90)
recall_score(y_train_5, y_train_pred_90)
```

ROC-curve

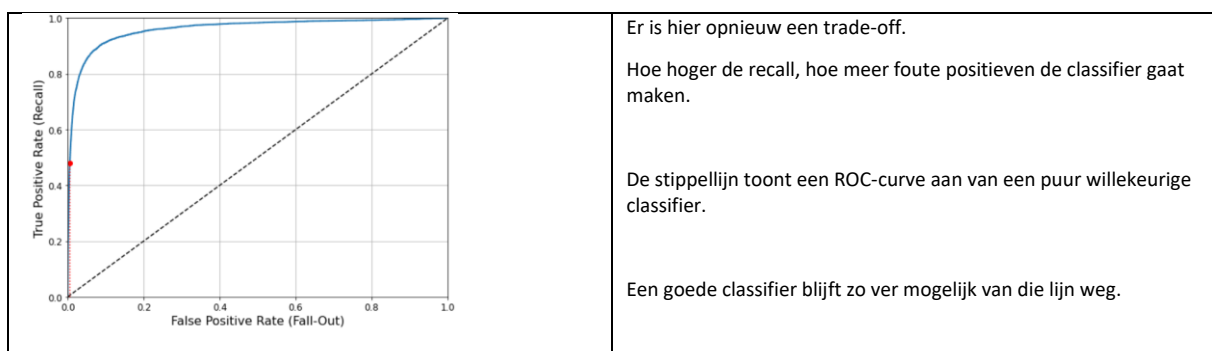
- Tool om te werken met binaire classifiers

- Zeer gelijkaardig aan de precisie/recall curve
- Plot de recall/TPR tegenover de FPR
 - FPR = ratio van negatieve instanties die incorrect zijn geclassificeerd als juiste instanties
 - Staat gelijk aan 1
 - TNR = ratio van negatieve instanties die correct geclassificeerd staan als negatieve.
 - Hoe correct hebben wij mensen zonder een hartziekte voorspelt?
 - Wordt ook 'specificity' genoemd
- Bereken TPR en FPR voor verschillende threshold-waarden

```
# bereken de TPR en FPR
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

# plot een curve die FPR tegenover TPR toont
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
    plt.axis([0, 1, 0, 1]) # Not shown in the book
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16) # Not shown
    plt.ylabel('True Positive Rate (Recall)', fontsize=16) # Not shown
    plt.grid(True) # Not shown

plt.figure(figsize=(8, 6)) # Not shown
plot_roc_curve(fpr, tpr)
fpr_90 = fpr[np.argmax(tpr >= recall_90_precision)] # Not shown
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:") # Not shown
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:") # Not shown
plt.plot([fpr_90], [recall_90_precision], "ro") # Not shown
save_fig("roc_curve_plot") # Not shown
plt.show()
```



Classifiers vergelijken:

- Ruimte onder de curve meten
 - Een perfecte classifier zal een score hebben van 1
 - Een willekeurige classifier zal een score hebben van 0.5
- Train een classifier en vergelijk de roc-curve met de auc-score

- Haal de scores op van de trainingset
- Gebruik de predict_proba methode
 - Geeft een array met daarin rij per instantie en kolom per klasse
 - Elk bevat de kans dat de instantie behoort tot de gegeven klasse

```

from sklearn.metrics import roc_auc_score
# auc-score berekenen
roc_auc_score(y_train_5, y_scores)
# uitkomst is 0.967..

# maak een classifier aan met een gegeven aantal estimators
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
# voorspel op basis van de classifier een forest
y_probab_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                   method="predict_proba")

# roc_curve verwacht labels en scores, maar je kan hier ook class probabilities meegeven
y_scores_forest = y_probab_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)
# plot nu de ROC curve
plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right")
plt.show()
# wat is de auc-score? Hoe groot is de ruimte onder de curve?
roc_auc_score(y_train_5, y_scores_forest) #result = 0.998

```

ROC of PR?

- Verkiez de PR-curve wanneer de positieve klasse zeldzaam is
 - Of wanneer je meer geeft om de valse positieven dan de valse negatieven
- Gebruik in het andere geval de ROC curve

Multiclass classification

- Meer dan twee klassen (<> binaire classification)
- Sommige klassen doen dit nativly
- Verschillende strategieën om multi-class classification uit te voeren met meerdere binaire classifiers
- Systeem maken voor de tien getallen – twee manieren:

One versus the rest strategie	One versus one strategie
10 binaire classifiers trainen, voor elk getal 1	Binaire classifier trainen voor elk paar digits :
Soort van detector (is dit een nul? Ja of nee; is dit een één? Ja of nee; etc.)	Is dit nul of één? Is dit nul of twee? Is dit één of twee?
Wordt gebruikt bij het merendeel van binaire classification algo's.	Bij n-aantal klassen $\Rightarrow N \times (N-1)$ aantal classifiers <ul style="list-style-type: none"> • In dit geval: tien afbeeldingen

	<ul style="list-style-type: none"> • $10 \times (10 - 1) / 2 = 90 / 2 = 45$ classifiers • Te veel <p>Voordeel: iedere classifier moet enkel getraind worden op het deel van de trainingset voor de twee klassen die het moet onderscheiden.</p>
--	--

Scikit detecteert snel wanneer je een binaire classification algo voor een multiclass classification taak gebruikt.

- Draait automatisch OvR of OvO

```

# SVC classifier importeren
from sklearn.svm import SVC
# nieuwe classifier aanmaken (opties spelen weinig rol)
svm_clf = SVC(gamma="auto", random_state=42)
# classifier trainen
# trainen op de originele doelklassen van 0 tot en met 9
svm_clf.fit(X_train[:1000], y_train[:1000]) # y_train, niet y_train_5
# voorspelling maken
svm_clf.predict([some_digit])
# decision_function() -- geef tien scores terug per instantie
# één score per klasse
some_digit_scores = svm_clf.decision_function([some_digit])
some_digit_scores
# hoogste score is die van de overeenkomstige klasse -- hier 5
np.argmax(some_digit_scores)
# alle klassen die momenteel worden bijgehouden
svm_clf.classes_
# wat is de waarde van 5 hier?
svm_clf.classes_[5] # uitkomst = 5

```

Getrainde classifier:

- Bewaart een lijst van alle doelklassen in de 'classes_' attribuut.
- In dit geval staat de index van iedere klasse gelijk aan het cijfer van de klasse zelf

SciKit kiest automatisch – maar kan je forceren.

- Maak een instantie en geef de classifier door naar de constructor

```

# klasse forceren -- hier wordt OvR strict gebruikt
from sklearn.multiclass import OneVsRestClassifier
ovr_clf = OneVsRestClassifier(SVC(gamma="auto", random_state=42))
ovr_clf.fit(X_train[:1000], y_train[:1000])
ovr_clf.predict([some_digit])

# aantal estimators
len(ovr_clf.estimators_)

# SGDclassifier gaan trainen
sgd_clf.fit(X_train, y_train)
sgd_clf.predict([some_digit])

# tien binaire classifiers worden getraind
# per klasse wordt er één waarde gegeven
sgd_clf.decision_function([some_digit])

```

De voorspelling decision function interpreteren:

- Functie geeft een array terug met daarin alle voorspellingen
- Hoe harder onder nul de score is, hoe correcter de voorspelling dat iets fout is

- Hoe hoger boven nul de score is, hoe correcter de voorspelling is

Classifier evalueren:

- Cross-validatie gebruiken:

```
# SGDClassifier meegeven als dataset
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")

# bedoeld voor hogere nauwkeurigheid te behalen
# de inputs worden hier beter geschaald
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```

Het model verbeteren met behulp van de confusion matrix.

- Gebruik de 'cross_val_predict()' en daarna 'confusion_matrix()'

```
# confusiematrix tonen
y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
conf_mx = confusion_matrix(y_train, y_train_pred)
conf_mx

# met sklearn 0.22 kan je sklearn.metrics.plot_confusion_matrix() gebruiken
# matrix plotten
def plot_confusion_matrix(matrix):
    """If you prefer color and a colorbar"""
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)

# toon de matrix en sla deze op
plt.matshow(conf_mx, cmap=plt.cm.gray)
save_fig("confusion_matrix_plot", tight_layout=False)
plt.show()
```

```
# splits iedere waarde in de confusiematrix met het aantal images in de bijhorende klasse
row_sums = conf_mx.sum(axis=1, keepdims=True)
# o.b.v. het aantal kan je het aantal error-
# rates gaan vergelijken met het totaal aantal errors
norm_conf_mx = conf_mx / row_sums

# behoud enkel de fouten --> vul de diagonale met nullen
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
save_fig("confusion_matrix_errors_plot", tight_layout=False)
plt.show()
```

Alleenstaande fouten analyseren:

- Goede manier om inzicht te krijgen in wat je classifier doet
- Moeilijker en time-consuming

```
# plot voorbeelden van de cijfers 3 en 5
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]
plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
save_fig("error_analysis_digits_plot")
plt.show()
```

De twee 5x5 blokken boven tonen alle cijfers aan die geclassificeerd zijn als een 3 terwijl de twee blokken onderaan alle cijfers tonen die geclassificeerd zijn als een 5.

- Sommige cijfers zijn verkeerd ingeschat door de classifier
- Vb: SGD is een lineair model – het weegt per klasse tegenover de pixel en bouwt zo scores op
- 3 en 5 zijn redelijk gelijkaardig qua pixels, vandaar de vergissing
- Classifier is gevoelig aan image shifting – oplossing is preprocessing

Multilabel classification:

- Meerdere klassen tonen per instantie of meerdere ‘binaire tags’ geven

```
# we maken een array aan met twee target-labels voor ieder cijfer (afbeelding)
from sklearn.neighbors import KNeighborsClassifier
y_train_large = (y_train >= 7) # hoe groot is de afbeelding?
y_train_odd = (y_train % 2 == 1) # hoe zeldzaam/raar is de afb?
y_multilabel = np.c_[y_train_large, y_train_odd]

# classifier aanmaken
knn_clf = KNeighborsClassifier()

# train het met de multiple targets array
knn_clf.fit(X_train, y_multilabel)

# voorspelling maken --> geeft je een array met twee waarden terug
knn_clf.predict([some_digit])
```

Het resultaat zal hier een array geven van [True en False].

- True = hoe 'raar' is de afbeelding?
- False = de afbeelding voldoet niet aan de grootte

Meerdere manieren om een multilabel classifier te evalueren.

- Vb F1 score berekenen voor ieder label en daarna de gemiddelde score berekenen

```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
# verander het 'gewicht'/impact dat een label krijgt bij 'average='
f1_score(y_multilabel, y_train_knn_pred, average="macro")
```

- Gaat er wel van uit dat alle labels even belangrijk zijn
 - Meer gewicht geven aan de score op afbeeldingen met 'average=weighted'

Multioutput classification:

- Meerdere outputs worden simultaan berekend
- Een generalisatie van multilabel classification waar iedere label een multiclass kan zijn
- Illustratie: een systeem maken dat ruis van afbeeldingen gaat halen.
 - Image met noise/ruis → clean image
- De output is multilabel (één label per pixel) en per label kan je meerdere waarden hebben.

Verskil tussen classificatie en regressie is soms *blurry*:

- Pixel intensiteit voorspellen is meer zoals regressie dan classificatie
- Multioutput is niet limited tot classification taken

```
# trainingset en testset maken door de MNIST afbeeldingen
# maak noise aan met de randint-functie
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test

# noise toevoegen
some_index = 0
plt.subplot(121); plot_digit(X_test_mod[some_index])
plt.subplot(122); plot_digit(y_test_mod[some_index])
```

```
save_fig("noisy_digit_example_plot")
plt.show()

# classifier gebruiken om het cijfer schoon te maken
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
save_fig("cleaned_digit_example_plot")
```


H4: Trainingsmodellen

Dot product	Inwendig product
Transpose van een matrix	A' of A^t Schrijf de rijen als de kolommen van A^T Schrijf de kolommen als de rijen van A^T
Pseudoinverse	Voorgesteld als X^+ Wordt berekend met SVD – splitst de trainingset in een matrix multiplication van drie matrices. <ol style="list-style-type: none">1. Algoritme neemt de sigma en zet alle waarden kleiner dan de threshold op nul.2. Alle niet-nulwaarden worden geïnverteerd.3. Matrix wordt getransponeerd teruggegeven

Weten wat er achter de schermen gebeurt:

- Efficiënter bugfixen

Modellen:

Gesloten lineaire regressie	Lineaire regressie met <i>gradient descent</i>	Polynomiale regressie
Geregulariseerde binaire modellen	Logistische regressie	<i>Softmax</i> regressie

```
import numpy as np

# willekeurig set van punten genereren, we genereren 100 random woorden -
# - vector maken, X = uniform verdeeld (evenveel tussen ieder interval)
X = 2 * np.random.rand(100, 1)

# Y = bijhorende y-
# waarde berekenen voor de X, 'ruis' of willekeurige afwijking toevoegen,
# .randn() --> willekeurige normaalverdeling
y = 4 + 3 * X + np.random.randn(100, 1)

# Honderd punten gaan plotten
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 2, 0, 15])
save_fig("generated_data_plot")
plt.show()

# maat van de helling / rico is 3
# opdracht nu -- rechte opstellen die het minste MSE heeft op alle Y-waarden

# de theta-waarde met de minste kostfunctie | theta hoedje berekenen
X_b = np.c_[np.ones((100, 1)), X] # add x0 = 1 to each instance
# .inv() --> matrix transponeren of inverteren
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```

# theta hoedje tonen
theta_best

# voor 0 en 2 maken we nieuwe X-waarden
X_new = np.array([[0], [2]])
# aan iedere instantie 'x0 = 1' toevoegen
X_new_b = np.c_[np.ones((2, 1)), X_new]

# voorspelling maken met gebruik van theta hoedje en de dot-functie
# dot-functie = matrix verdubbelen
y_predict = X_new_b.dot(theta_best)
y_predict

# de voorspellende lineaire regressie plotten
plt.plot(X_new, y_predict, "r-")
# alle 'punten' plotten
plt.plot(X, y, "b.")
# assen maken
plt.axis([0, 2, 0, 15])
plt.show()

```

Lineaire regressie uitvoeren:

```

# rechte opstellen
from sklearn.linear_model import LinearRegression
# instantie maken
lin_reg = LinearRegression()
# model trainen op input-waarden en y-waarden
lin_reg.fit(X, y)
# intercept -- waar gaat die door de y-as
# coef_ -- rico
lin_reg.intercept_, lin_reg.coef_
# voorspellen
lin_reg.predict(X_new)

# .lstsq --> 'least squares', de minste afstand
theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
theta_best_svd

# alternatief op .lstsq() om de pseudoinverse te berekenen
np.linalg.pinv(X_b).dot(y)

```

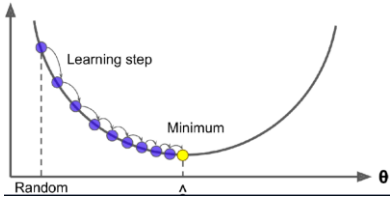
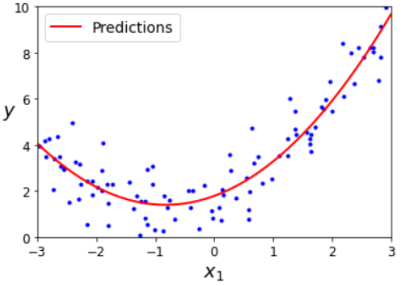
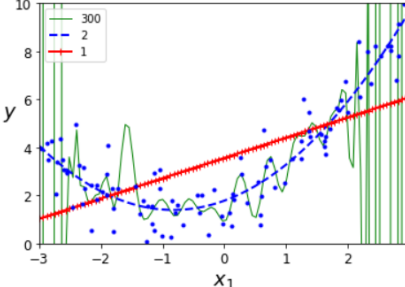
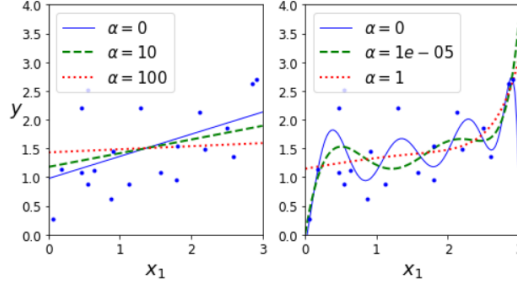
Tijdscomplexiteit

- Normaalverdeling berekent het omgekeerde van de $X^t X$ of $(n+1) \times (n+1)$
 - N = het aantal features
 - Tijdscomplexiteit ligt tussen $O(n^2)$ en $O(n^3)$
 - $SVD \rightarrow O(n^2)$
- Voorspellingen gaan veel sneller
 - Lineaire complexiteit vergeleken met het aantal instanties waarvan je een voorspelling wilt maken
 - Een voorspelling op twee keer zo veel instanties maken gaat minstens twee keer zo lang duren.
 - Hoe groot het aantal features \rightarrow performantie gaat traag zijn
 - Ander model nodig

Gradient descent: [Epoch vs Batch Size vs Iterations](#) | by SAGAR SHARMA | Towards Data Science

(skip dit maar, ik heb liever de uitleg op de schema's van de volgende pagina, site is ook goed uitgelegd 😊)

- Algo om de optimale oplossing aan een brede range van problemen vinden
 - Werking: Parameters iteratief tweakten om de kostfunctie te minimaliseren
 - Meet de lokale graad van het probleem en gaat verder totdat deze graad nul is
- Random initialization: Theta vullen met willekeurige waarden
 - Graduaal verbeteren om zo de kostfunctie te verminderen
 - Improven tot het algoritme convergeert tot een 'globaal minimum'
- Grootte van de stappen speelt belangrijke rol
 - Te klein – veel iteraties nodig
 - Te groot – 'heen en weer springen' om mogelijks op de andere kant te raken
- Niet alle kostfuncties zijn even 'mooi'
 - Mogelijks kan je een lokaal minimum tegenkomen \neq globaal min.
- MSE-kostfunctie bevat een convex-functie
 - Neem twee punten op de curve – het lijnsegment zal elkaar niet crossen
 - Geen lokaal minima, enkel een globaal minima
 - Continue functie met een helling die niet abrupt verandert
 - Geeft je garandeert het globaal minima mits wat wachten en als de learning rate niet te hoog ligt
- Kostfunctie heeft de vorm van een kom
 - Langwerpige kom als je de schaal aanpast – feature scaling
 - Je komt snel bij het minimum met feature scaling
- Gebruik StandardScaler om alles zo snel mogelijk te convergeren.
- Een model trainen == een combinatie van modelparameters zoeken die de kostfunctie minimaliseren
 - Hoe meer parameters een model heeft – hoe groter de dimensies dat deze ruimte heeft en hoe moeilijker het is om dit te vinden
 - In het geval van lineaire regressie – hetgeen wat je zoekt is vaak 'onderaan de kom'
- Implementeren:
 - Bereken de graad van de kostfunctie bekeken vanop iedere modelparameter Theta
 - Of hoeveel gaat de kostfunctie veranderen als je Theta aanpast
 - == partiële afgeleide

<p>Lineaire regressie (LinReg)</p> <p>Het verloop of de trend stelt een rechte voor.</p> <p>Formule:</p> $\hat{y} = h_0(\mathbf{x}) = \theta \cdot \mathbf{x}$ <p>0/ stelt hier een model parameter voor. x stelt een feature van de instantie voor. '0/ * X' stelt hier het inwendig product voor.</p> <p>Het model train je door de 0 te vinden met de kleinste RMSE. Het is beter om de MSE hierop te berekenen dan de RMSE. Het resultaat staat vaak gelijk.</p> <p>Lineaire modellen zijn niet in staat om verbanden of relaties terug te vinden tussen de features in.</p> <p>Directe closed-form vergelijking die de best passende parameters gaat berekenen naar de trainingset</p> <p>Maakt een voorspelling door de gewogen som van de input te berekenen (= bias term of intercept term).</p>	<p>Lineaire regressie met gradient descent (GD)</p> <p>GD = Iteratief parameters gaan aanpassen om zo de kostfunctie te verminderen.</p> <p>Je begint met het opvullen van de modelparameter 0/ met willekeurige parameters. (= random initialisatie)</p> <p>Je neemt stappen om zo de kostfunctie te verminderen. De stappen zijn niet statisch! Deze stappen zijn proportioneel op basis van de helling.</p> <p>Feature scaling is belangrijk hier! Zorgen voor waarden van bijvoorbeeld -1 tot 1 (of eender welke schaal) zorgt voor een komvormig verloop van de grafiek. (of schaalvormig no pun intended lmao)</p>  <p>Twee soorten: <u>batch</u> en <u>stochastische</u> GD.</p> <p>De twee soorten worden in het volgend schema verder uitgelegd.</p>	<p>Polynomiale regressie (PolyReg)</p> <p>Model dat niet-lineaire trends kan voorspellen. Het verloop of de trend stelt geen rechte voor, maar is opgebouwd uit een veelterm. Dit kan bijvoorbeeld een hyperbool, parabool, ellipsen, etc.</p> <p>Deze functie is in staat om verbanden te vinden tussen de features. Dit komt omdat alle combinaties van de features tot een bepaalde graad worden samengevoegd. Zo bekom je niet a^2, a^3 of b^2, maar ook ab of a^2b.</p>  <p>Veeltermregressie met hogere graden zullen trainingsdata véél beter trainen dan met lineaire regressie of eentermfuncties. Hieronder gaat PolyReg overfitten en LinReg underfitten.</p> 	<p>Geregulariseerde lineaire modellen. (RegReg)</p> <p>Dit model pakt overfitten aan. Dit kan je voorkomen door het model te gaan beperken.</p> <p>Beperken doe je door :</p> <ul style="list-style-type: none"> • het aantal vrijheidsgraden te verminderen • de gewichten van het model te beperken <p>Hoe lager het aantal vrijheidsgraden, hoe moeilijker het wordt voor het model om te overfitten.</p> <p>Ridge regressie:</p> <p>Bij de kostfunctie wordt er een regularisatieterm meegenomen. (alpha)</p> <p>Alpha = 0</p> <ul style="list-style-type: none"> • gewone lineaire regressie <p>Alpha = zeer groot</p> <ul style="list-style-type: none"> • alle gewichten gaan dicht bij nul liggen, het resultaat zal een platte lijn zijn die het gemiddelde snijdt <p>Links (LinReg) – rechts (RegReg)</p>  <p>Belangrijk dat je de data schaalt bij een geregulariseerd model!</p> <ul style="list-style-type: none"> • Model is gevoelig aan de schaal van de input features.
---	--	---	---

Logistische regressie

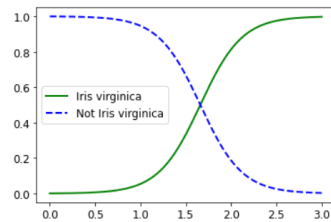
De **kans** berekenen waarbij een instantie **tot een klasse zal behoren**.

Vb: hoe groot is de kans dat een e-mail spam is?

Dit is een **binaire classifier** ofwel de kans wordt uitgedrukt in positieve en negatieve klassen.

- De e-mail is spam (positief of 1)
- De e-mail is geen spam (negatief of 0)

Stelt een sigmoïde functie voor:



Het midden stelt een ruimte van onzekerheid voor. We zijn niet volledig zeker of iets tot de positieve of negatieve klasse behoort.

Mogelijk om twee input features te gebruiken.

Per standaard One-Versus-Rest

Per standaard geregulariseerd.

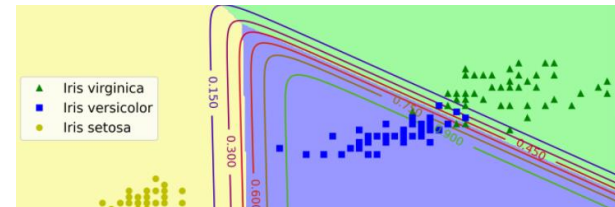
Softmax regressie

Dit is een uitbreiding op het logistische regressiemodel.

Bij dit model worden meerdere klassen ondersteund zonder dat je meerdere binaire *classifiers* nodig hebt.

Twee stappen:

1. Score op iedere klasse berekenen
2. De kans dat een instantie tot een klasse behoort berekenen.



Batch Gradient descent

De volledige dataset wordt gebruikt om de hellingsgraad te berekenen.

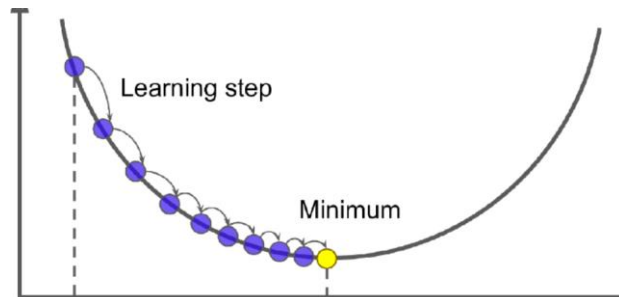
- Gevolg: De snelheid van het zal heel traag zijn op te grote datasets.

Hoe dichters naar het minimum, hoe kleiner de stap. Dit noemt de **partiële afgeleide**. Hiervoor gebruiken we wel een *learning rate* parameter.

- Hoe **hoger** de learning rate, hoe groter de stappen. Dit kan leiden tot het 'heen en weer' springen en eventueel dat er regelmatig over het minimum heen wordt 'gesprongen'.
- Hoe **later** de learning rate, hoe kleiner de stappen. Dit ten gevolge dat het model trager zal zijn omdat er meer stappen moeten worden afgelegd.

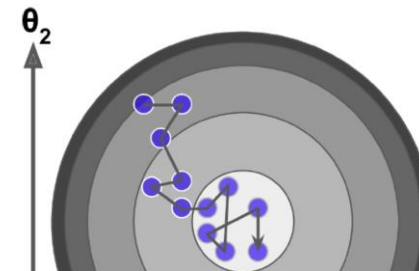
Het minimum zal, afhankelijk van de grootte van de learning rate, **altijd het minimum bereiken**.

- Te grote/kleine learning rate zorgt voor een inefficiënte snelheid van het model.



Stochastic gradient descent

- Eén instantie nemen: welke instanties gaan naar beneden/boven?
- Een **willekeurig deel** van de trainingset wordt genomen.
 - Enkel op dat deel worden de (hellings)graad of gradient berekend.
- Instanties worden *ad random* gekozen → '**dronkemanstaktiek**'
 - Je vermindert gradueel de kostfunctie via kromme wegen
 - Minder regelmaat: de kostfunctie gaat niet alleen omlaag gaan, maar heeft momenten van schommelingen tussen verhogen en verlagen.
- **Epoch** = het aantal keer dat je door jouw trainingset loopt
 - We doorlopen een willekeurig aantal instanties in een trainingset (= m)
 - We berekenen bij iedere iteratie de graad en proberen het minimum te benaderen.
 - Dit maakt het aantal epochs een **zeer belangrijke hyperparameter!!**
- SGDRegressor! Opties:
 - max_iter = aantal iteraties dat max. worden afgelegd
 - Tol = tolerantie, kan langer duren afhankelijk van de tolerantie
 - Eta (=grootte van de stappen)



H5: Support Vector Machines

SVM of een techniek die in staat is om lineaire of niet-lineaire classificatie en regressie uit te voeren.

- Kan ook uitschieters achterhalen.
- Ideaal voor kleine tot middelgrote datasets.
- Het doel is om een zo groot mogelijke 'straat' te vinden zonder enige instanties erop.
Geen enkele instantie mag op de straat liggen.

Verschillende soorten SVM's voor classificatie. (OPM: Naast classificatie kan je ook regressie uitvoeren met SVM's.)

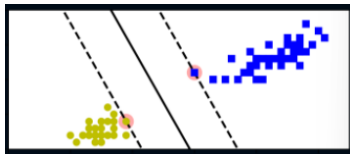
- Lineaire SVM classificatie
- Soft-margin classificatie
- Niet-lineaire SVM classificatie
- Polynomiale kernel

Lineaire SVM Class.

De klassen zijn duidelijk gescheiden met een volle, rechte lijn. Ze zijn lineair verdeelbaar of *linearly separable*.

De gestreepte lijn toont de *decision boundary* van het model aan. Hier kunnen de klassen niet meer goed onderscheiden worden.

Large margin classification of de breed mogelijke straat gaan trainen. De straat wordt aangeduid tussen twee gestreepte parallele lijnen. Supportvector is een instantie die op de boord van de straat ligt.

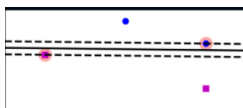


OPM: De *decision boundary (db)* zal niet beïnvloed worden als er meer instanties bijkomen. De db is volledig afhankelijk van de instanties die op de boord van de straat ligt ofwel de support-vectoren.

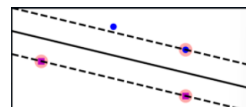
Gevoelig aan feature scaling:

- De twee support vectoren mogen er niet voor zorgen dat de 'straat te dun' is.
- Belangrijk dat de waarden goed geschaald zijn om zo een juiste verhouding te hebben tussen de twee SVM's.

Zonder scaling



Met scaling



Tijdscomplexiteit:

$O(m \times n)$ waarbij m het aantal trainingsinstanties voorstelt en n de nodige trainingstijd per instantie.

- Het duurt langer als je hogere precisie wilt, maar in de meeste gevallen is de standaardtolerantie perfect oke!

Niet-lineaire SVM class.

Sommige gevallen waarbij de horizontale elkaar overlappen of waar we niet duidelijk een lineaire scheiding kunnen zien.

Oplossing: polynomiale features toevoegen.

- op die manier bekommen we een dataset die wél lineair gescheiden kan worden

Werkwijze mbv een pipeline:

- polynomiale features toevoegen
- feature scaling met standardscaler
- linearsvc

Tijdscomplexiteit:

Ligt tussen $O(m^2 \times n)$ en $O(m^3 \times n)$.

- Hoe groter het aantal trainingsinstanties, hoe trager het model.
- Perfect voor kleine tot middel-grote datasets.
- Schaalt goed met *sparse features* (niet-nullen)

Polynomiale kernel

Problemen met polynomiale features toe te voegen bij SVM's:

- Bij een laag polynomiaal niveau is het model niet in staat om complexe datasets te behandelen. Bijvoorbeeld bij een hyper- of parabole grafiek. (tweedegraadsfunctie).
- Bij een hoog polynomiaal niveau gaat het model te traag zijn omdat er een groot aantal features zijn die behandeld moet worden.

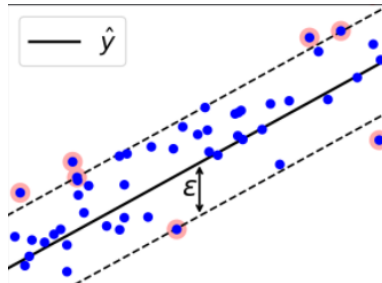
Oplossing met de *kernel trick*:

- Zelfde resultaat bekomen met eender welk aantal polynomen. Hoog en laag niveau doet er niet toe.
- Je voegt niet echt features toe dus geen problematische aanpak voor het aantal features.

OPM: niet ondersteund bij de Lineaire SVC's. Niet-lineaire SVM's ondersteunen dit wel.

SVM Regressie

- Omgekeerd doel dan SVM class.: deze techniek probeert om zo veel mogelijk instanties op een straat te krijgen én beperkt het aantal instanties buiten de marges.
 - Beperkte marges wilt zeggen dat je zo weinig mogelijk instanties niet op de straat wilt hebben.
- De breedte van de straat hangt af van de hyperparameter *epsilon* ϵ
- Voor niet-lineaire regressietaken gebruik je best een kernelized SVM model.

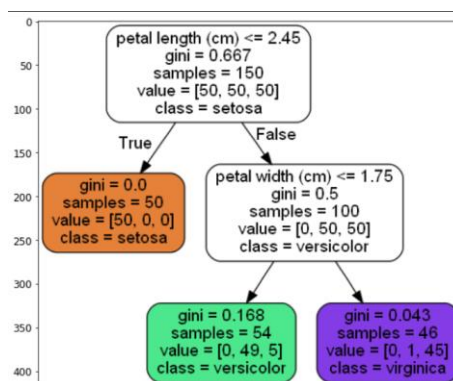


H6: Decision trees

Decision trees:

- **Bij classificatie en regressie (CART): enkel binaire bomen mogelijk!**
 - Voorbeeld groter of kleiner
- Ieder **blad** stelt een klasse voor.
 - Bijvoorbeeld klassen: peuter, kind, adolescent, volwassen, gepensioneerd
 - Afhankelijk van de paden dat gekozen werden.
- Diepte moet je als parameter meegeven (*max_depth*)
- Dit model vereist weinig datavoorbereiding. Je hebt **géén feature scaling nodig**.
- Vier parameters nodig:

<p>Gini</p> <p>Dit is de correctheid of juistheid van de splitsing.</p> <p>Als de node één klasse heeft dan spreken we van een pure gini.</p> <p>Als de node meerdere klassen heeft dan spreken we van niet-pure gini.</p> <p>Hoe lager de <i>gini</i>, hoe puur de node is. Een pure klasse heeft altijd waarde 0.</p> <p>Hieronder is de klasse setosa zeer puur want alle instanties behoren tot één klasse toe. Bij value zie je enkel op één plaats alle instanties staan.</p> <p>De node voor versicolor en virginica zijn niet puur. Versicolor heeft bijvoorbeeld vijf instanties staan die niet tot de klasse toebehoren. Dit zie je opnieuw bij values.</p>	<p>Class</p> <p>De klasse waartoe de instanties behoren.</p> <hr/> <p>Samples</p> <p>Het aantal samples die voldoen aan de criteria om tot deze klasse te behoren.</p> <hr/> <p>Value</p> <p>Het aantal trainingsinstanties die tot de klassenode toebehoren.</p>
---	--



Visueel:

- Flow-chart structuur waarbij iedere *node* een test voorstelt van een attribuut.
 - Bv.: Is de lengte van een persoon groter of kleiner dan 1 meter 60?
 - Een tak is het resultaat op een test. Bijvoorbeeld ja of nee. Dit zijn de pijlen.
 - De rechthoeken stellen de nodes voor. Die geven de gini, samples, value en klasse mee.
 - De bladeren vindt je helemaal onderaan iedere tak.
 - Verschil tussen white-box en black-box modellen.

<p>Whitebox modellen</p> <ul style="list-style-type: none"> • Geeft het gedrag aan • Maakt voorspellingen • Bv: software die een persoon herkent kan het model duidelijk zeggen wie het is. Het model weet dit omwille van oogkleur, haarkleur, gelaat, etc. 	<p>Blackbox modellen</p> <ul style="list-style-type: none"> • Maakt voorspellingen zonder duidelijk te zeggen hoe iets voorspelt is • Bv: software om een persoon te herkennen kan het model zeggen wie het is, maar het kan niet pinpointen waarom dat het deze persoon is.
--	---

De kansen van een klasse inschatten:

- Het aantal instanties van een klasse + de verhouding tussen de klassen wordt teruggegeven.
- Werkwijze:
 - De boom wordt afgegaan richting de node. Er wordt gezocht naar de node.
 - De ratio van het aantal instanties van een klasse in de node wordt teruggegeven.

Classification and Regression Tree of CART

- Algoritme dient om decision trees te trainen met een zo klein mogelijke kostfunctie.
- Doel: de *gini* zo klein mogelijk houden.
- **Het algoritme splitst de trainingsets recursief op in twee o.b.v. van een threshold en een feature.**
 - Voorbeeld: de lengte van een persoon mag niet langer zijn dan 2 meter
 - Het zoekt de waarden k en t voor een paar dat enkel uit pure subsets bestaat.
 - Recursief:
 - iedere node wordt opgesplitst tot je aan de maximale diepte komt
 - of wanneer er een split is waarbij de gini niet meer verminderd kan worden (gelijk aan 0)
- Tijdscomplexiteit: $O(\log(m))$

Vereiste hyperparameters:

- Model regulariseren: verhoog de minimumparameters en/of verminder de hyperparameters

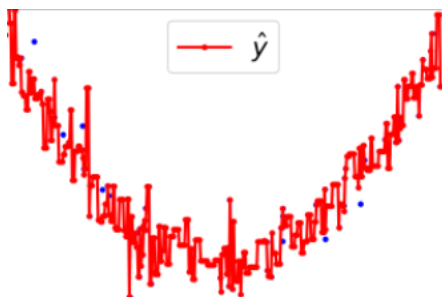
Max depth	Maximale diepte van de boom. Staat standaard op oneindig. (None)	Min samples split	Het minimum aantal samples dat een node moet hebben vooraleer het gesplitst mag worden.
Min samples leaf	Het minimum aantal samples dat een node moet hebben.	Min weight fraction leaf	
Max leaf nodes	Het hoogste aantal kinderen dat een node mag hebben.	Max features	

Opmerking: mijn intuïtie zegt dat ze hier nog een oefening op kunnen vragen waarbij ze een boom geven met de parameters en je moet zeggen of de boom juist of fout opgesteld is. Bijvoorbeeld een node met drie kinderen terwijl de parameter op twee staat. #not a prodigy

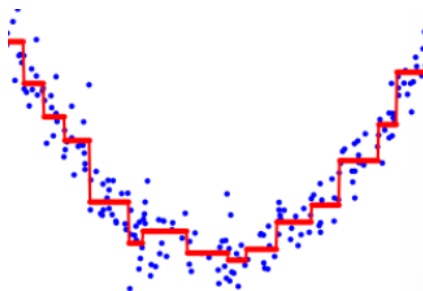
Regressietaken met decision trees

- Focus ligt hier op het minimaliseren van de MSE.
 - Bij het CART-algoritme lag de focus op het minimaliseren van de impureteit of het streven naar een gini van 0.
- Gevoelig aan overfitten → `min_samples_leaf` aanpassen naar een waarde die het overfitten tegen gaat.

Met overfitten



Na het aanpassen van het minimumaantal:



H7: Ensemble learning

Ensemble methode

- Meerdere predictormodellen bij elkaar nemen.
 - Bewijzend effect op verbeterde en nauwkeurige resultaten.
 - Random Forest is een vorm van ensemble learning.
- Werking:
 - Je traint eerst elke decision tree op een willekeurige subset van de trainingset.
 - Op alle aparte bomen ga je de voorspellingen gaan verzamelen, achteraf neem je de klasse met de meeste stemmen.

Trade-off tussen bias en variantie

Drie mogelijke fouten: bias, variantie en een niet verminderbare of *irreducible* fout.

<u>Bias</u>	<u>Variantie</u>	<u>Irreducible error</u>
Dit is een fout bij het generaliseren van data. Met eigen woorden gaat het model er van uit dat iets zo is. Bijvoorbeeld er van uit gaan dat een model kwadratisch zal zijn terwijl een model eigenlijk lineair is. Hoe groter de bias bij een model, hoe groter de kans op het underfitten van data.	Het model is overdreven gevoelig aan kleine variaties in de trainingsdata. Hoe hoger de vrijheidsgraad, hoe groter de variantie. Hoe groter de variantie, hoe groter de kans dat het model de data zal overfitten.	Dit probleem duidt aan dat er te veel ruis of <i>noise</i> is op de data. Je kan de noise verwijderen door de dataset eerst te <i>cleanen</i> . Bijvoorbeeld door uitschieters te verwijderen uit de dataset.

Voting classifiers

- Probleem: We hebben een manier nodig om de volgorde of het gebruik van de predictors terug te vinden.
- We hebben het object VotingClassifier waarbij we enkel de parameter moeten meegeven.
- We kunnen kiezen voor hard of soft voting.

<u>Hard voting</u>	<u>Soft voting</u>
<ul style="list-style-type: none">• De klasse met de meeste stemmen.• Iedere classifier geeft een stem uit aan een klasse.• Resultaat: een hogere nauwkeurigheid dan de beste classifier in de ensemble.	<ul style="list-style-type: none">• Iedere classifier gaat een p-waarde geven waar een specifiek datapunt zich ongeveer gaat op richten.• De kans wordt berekend op basis van predict proba.• De label met de grootste som van de gewichten krijgt de <i>winning vote</i>.• Deze manier is sneller want er wordt meer gewicht geplaatst in zeer zelfverzekerde modellen.

Bagging en pasting

- Je gaat dezelfde trainingsalgo voor iedere predictor opnieuw gebruiken.
- **Onderscheid tussen met (=bagging) of zonder replacement (= pasting) werken.**
- Eenmaal de predictors getraind worden zal de ensemble een voorspelling maken voor een nieuwe instantie.
- De **ensemble voorspelt** een nieuwe instantie door de **voorspelde waarden van de predictors op te tellen**
 - Optellen of aggregeren zorgt voor een verminderde bias en kleinere variantie.
- Iedere **aparte predictor heeft een grotere bias** dan als het op de originele dataset werd getraind.
- Deze twee technieken **zijn zeer goed schaalbaar**.
 - Predictors kunnen parallel getraind worden over verschillende cpu-kernen heen.

<u>Bagging</u>	<u>Pasting</u>
Trainingsinstanties worden meerdere keren gesampled over verschillende predictors <ul style="list-style-type: none">• Hier kan een sample meermaals getraind worden bij dezelfde predictor.• BaggingClassifier gebruikt hier soft voting in plaats van hard voting.	Trainingsinstanties worden meerdere keren gesampled over verschillende predictors. <ul style="list-style-type: none">• Hier wordt er gesampled zonder vervanging.

Out-of-bag evaluation

Nadeel van bagging:

- Sommige instanties werden meermaals gesampled, andere helemaal niet.
- Classifieer gaat per standaard de trainings-instanties met een vervanger gaan samplen.
 - ~ 63% zal gemiddeld gesampled worden in ieder geval, ~ 37% heeft kans om niet gesampled te worden
 - *Out-of-bag (oob)* is wanneer dat je trainingsinstantie niet wordt gebruikt bij het samplen. Je laat het in de verpakking
- De niet-gebruikte (of oob) instanties kan je gebruiken en evalueren, zonder de nood aan een aparte validatieset.

Random Forests

- Een ensemble of bundel van decision trees.
- Worden meestal via bagging getraind.
- De hyperparameters van decision tree classifiers worden overgenomen naar de random forest classifier.
 - Idem voor de bagging classifier hyperparameters.
- Feature importance
 - Het meten van de afhankelijke belangrijkheid van een feature.

Boosting

- Een ensemble methode die een zwakke learner gaat omzetten in een sterke learner.
- De predictors worden opeenvolgend of sequentieel aangeleerd. Elk probeert de predecessor te corrigeren.
- Er zijn verschillende boosting methoden beschikbaar. Voorbeelden: AdaBoost en Gradient Boost.

<u>Adaboost</u>	<u>Gradient Boost</u>
<p>Nieuwe predictors die gericht zijn op de moeilijkere gevallen.</p> <p>Werkwijze:</p> <ul style="list-style-type: none">• Eerst de base classifier trainen (bv. : decision tree) en hierop voorspellingen maken. Het gewicht van de verkeerde trainingsinstanties wordt verhoogd.• De tweede classifier, met nieuwe gewichten, wordt getraind en maakt voorspellingen o.b.v. trainingset. <p>SAMM-R:</p> <ul style="list-style-type: none">• De loss-functie is eerder gericht op waarschijnlijkheden dan voorspellingen. Snelheid is ook beter.• De leergraad hangt af van het aantal verkeerd geclassificeerde instanties.	<p>De methode probeert om een nieuwe predictor te trainen die zich aanpast aan de overgebleven fouten van de vorige predictor.</p> <ul style="list-style-type: none">• De methode gaat opkuisen wat de vorige predictor heeft achtergelaten.• In code: je vermindert het aantal fouten door de y-waarde van de dataset te verminderen met een voorspelling via predict(). vb: $y_2 = y - \text{tree.predict}(X)$ <p><i>Shrinkage</i> ofwel regulariseren door gebruik te maken van een heel lage (lager dan 0.1) <i>learning rate</i>.</p> <p><i>Early stopping</i></p> <ul style="list-style-type: none">• Geeft je een iterator terug over het aantal voorspellingen dat een ensemble heeft gemaakt op ieder stadium van de training (vb.: één boom, drie bomen, zevende boom, etc.)• In gevallen waarbij je het optimale aantal bomen wilt hebben.• Eenvoudige manier: staged_predict()

H10: Deep-learning en neurale netwerken.

ANN:

- Geïnspireerd op de structuur van het menselijk brein.
 - Hersenen bevatten neuronen die verbonden zijn met axonen.
 - Elektrische signalen worden verstuurd met de axonen naar de omringende neuronen.
- Schaalbaar en veelzijdig om complexe taken op zich te nemen
- Neurale netwerken (NN):
 - Modellen om complexe classificatie- en regressieproblemen op te lossen.

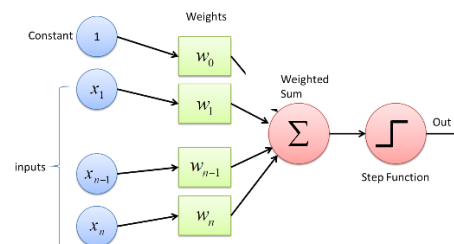
Uitleg perceptron

Basis:

- Algo om lineaire, binaire classifiers te trainen.
- Single-layer NN: er is maar één input-laag.
- Gebaseerd op Threshold Logical Units (TLU's)
- Gebruiken we om de data in twee delen te splitsen.
- Mogelijk om meerdere perceptronen te stapelen. Dit wordt dan een MLP of *Multilayer Perceptron*.

Proces:

- De inputsignalen worden vermenigvuldigd door het gewicht.
- Alle vermenigvuldigde waarden worden bij elkaar samengevoegd als een gewogen som.
- De gewogen som toepassen op de *step function*.

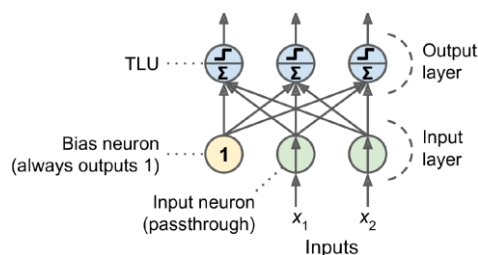


Fully connected layer:

- Dense layer
- Laag waarbij elke neuroon verbonden is met alle neuronen uit de vorige laag.

Multioutput classifier:

- Als een *classifier* meerdere instanties simultaan in minstens twee verschillende binaire klassen kan onderverdelen.



Waarom toch kiezen voor logistische regressie en niet voor een perceptron?

- Perceptronen maken voorspellingen op basis van een threshold. Ze geven géén *class probability* terug als output.
- Logistische regressie geeft wel de *class probability* terug.

Nadelen:

- Kan geen klassen voorspellen die niet lineair te scheiden zijn. Bv.: XOR-classificatieprobleem.

Vijf lagen:

Input:

- De data wordt aanvaard door het systeem. Elke input-node bevat een numerieke waarde.
- Input van een perceptron moet eerst doorheen de input neuronen.
- **Input neuronen** geven als resultaat altijd hetgeen wat ze gegeven werden terug. De output is gelijk aan input.
- **Bias term** wordt toegevoegd. De output van de bias term is altijd 1.

Gewicht:

- Het toont de sterkte aan van een verbinding tussen units. Deze proportie wordt bepaald op basis van de verwachte uitkomst.
- Op deze manier kunnen we rekening houden met bias-waarden. Dit zal bepalen of de *activation function* doen stijgen of dalen.
- Bijvoorbeeld je handpalm is gevoeliger dan de buitenkant van je hand. Je zal meer pijn voelen in je handpalm dan op de buitenkant. Het gewicht bij palm zal hier hoger liggen.

Weighted Sum: De som van alle gewichten.

Step function:

- Vaak *heavyside step function* of de *sign function*.
- *Heavyside* zal de waarden schalen tussen 0 en 1.
- *Sign* zal de waarden schalen tussen -1 en 1.

Multilayer Perceptron (MLP):

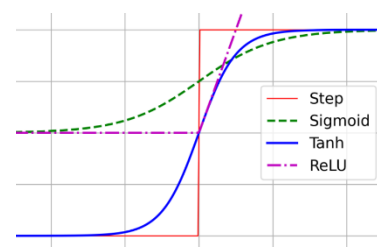
- Bestaat uit minstens één extra laag buiten de standaard inputlaag en outputlaag.
- Algoritme om te trainen:

<u>Backpropagation algoritme</u>	
<p>Werking:</p> <ul style="list-style-type: none"> • De verborgen laag krijgt willekeurige gewichten toegekend. • Je werkt met mini-batches. Je gaat de volledige trainingset in delen erdoor jagen <ul style="list-style-type: none"> ○ "Hoe goed is het al?" "Hoe goed is het deze keer?" • Epochs is het aantal keer dat de volledige trainingset doorlopen wordt. <p>Twee werkwijzen:</p>	
<p><u>Forward pass</u></p> <p>Werking:</p> <ol style="list-style-type: none"> 1. Elke mini-batch wordt doorgegeven aan de inputlaag. 2. De inputlaag stuurt die dan door naar de hidden layer. 3. De output voor iedere instantie in de mini-batch wordt berekend. 4. Het resultaat wordt doorgegeven aan de volgende laag. 5. Stap 3 en 4 worden herhaald tot het resultaat bij de inputlaag terechtgekomen is. <p>Opmerking: de tussentijdse resultaten, na afloop van stap 4 waarbij we nog niet bij de output layer zijn terechtgekomen, worden gehouden voor de backward pass.</p>	<p><u>Backward pass</u></p> <p>Waarde kiezen voor de gewichten van de hidden layer?</p> <ul style="list-style-type: none"> • Willekeurig is te riskant. Gerichte aanpak nodig. • We doorlopen het gehele netwerk achteruit. • We houden de foutenmarge van iedere node apart bij. • De gewichten gaan we baseren op de foutenmarge. <p>Werking:</p> <ol style="list-style-type: none"> 1. Loss-functie 2. Bijhouden hoeveel fouten iedere output-node heeft bijgedragen aan de totale foutenmarge. 3. Meten welk percentage er komt van de lagen onder de huidige laag. 4. Stap 1 t.e.m. herhalen tot je terug aan de inputlaag bent. 5. Een gradient descent step wordt uitgevoerd om de gewichten te gaan tweaken op basis van het foutenmarge. <p>De foutenmarge van de output wordt gemeten op basis van een loss-function.</p> <ul style="list-style-type: none"> • De functie vergelijkt de gewenste output met de echte output. De nauwkeurigheid wordt gemeten en doorgestuurd.

Activatiefuncties:

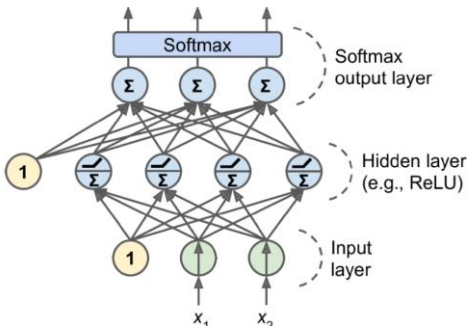
- Functie nodig om te bepalen hoe de gewogen som moet getransformeerd worden in output vanuit één of meerdere nodes.
- Meerdere lineaire transformaties → blijft een lineaire transformatie. We hebben een functie nodig die niet-lineair is.
- Step-functie moet vervangen worden. Drie alternatieven op de step-functie:

<p><u>Logistische of sigmoïde functie.</u></p> <p>Gebruik:</p> <ul style="list-style-type: none"> • Wanneer we geen graad nodig hebben om mee te werken. • We hebben enkel vlakke segmenten. • De waarden verschillen van 0 tot 1. 	<p><u>Hyperbolische tangent functie (Tanh)</u></p> <p>Gelijkaardig aan de logistische functie.</p> <ul style="list-style-type: none"> • De waarden van de output variëren van -1 tot 1. • De output van iedere laag zal schommelen rond 0 op het begin van de training.
<p><u>Rechtgezetete lineaire functie (ReLU)</u></p> <p>Gelijkaardig aan de logistische functie.</p> <ul style="list-style-type: none"> • Er is geen maximum output-waarde dus gunstigere resultaten bij <i>Gradient Descent</i> • Snel berekenbaar <p>De helling verandert abrupt op $z = 0$</p>	



MLP-regressie:

- Één outputneuron nodig om één waarde te voorspellen. (Univariate regression)
- Één outputneuron voor iedere waarde. Bijvoorbeeld temperatuur/snelheid voorspellen (multivariate regression).
- Geen activatiefunctie gebruiken voor de outputneuronen.
- Maatstaf van kwaliteit – MSE or MAE

Regressie MLP's	Classificatie MLP's
<p><u>Één waarde voorspellen:</u></p> <ul style="list-style-type: none"> • Één output neuron nodig. • Output = voorspelde waarde. <p><u>Meerdere waarden voorspellen:</u></p> <ul style="list-style-type: none"> • Één output neuron per outputdimensie • Bijvoorbeeld X- en Y-coördinaat. <p><u>Welk model kiezen?</u></p> <p><u>Positief resultaat?</u></p> <ul style="list-style-type: none"> • Gebruik ReLU • Gebruik de softplus activatiefunctie. Dit gaat nooit onder nul, maar een lage z-waarde zal dicht bij nul liggen. <p><u>Verwachting moet binnen een range van waarden liggen?</u></p> <ul style="list-style-type: none"> • Gebruik de logistische functie en schaal de labels van 0 tot 1. • Of gebruik de hyperbolic tangent functie en schaal de labels van -1 tot 1. <p><u>Waarom dient de loss-functie?</u></p> <ul style="list-style-type: none"> • Bij training met weinig outliers gebruik je best MSE. • Bij training met veel outliers gebruik je best MAE. 	<p><u>Classificatieprobleem:</u></p> <ul style="list-style-type: none"> • Één outputneuron met de logistische activatiefunctie nodig. Waarde zal tussen 0 en 1 liggen. • De omgekeerde waarde kan je berekenen door $1 -$ (verwachte kans dat iets tot een klasse behoort) <p>Multilabel binaire classificatie:</p> <ul style="list-style-type: none"> • Is een email ham of spam? Is een email urgent? • Twee output neuronen nodig, vb.: ham + urgent • Één output-neuron nodig per klasse en gebruik de softmax activatiefunctie voor de gehele output layer. <p>De softmax-functie zal ervoor zorgen dat de verwachte kansen tussen 0 en 1 liggen.</p> <ul style="list-style-type: none"> • Klassen die onafhankelijk zijn moeten hoogstens 1 hebben. 

Hyperparameters voor Regressie MLP's

Hyperparameter	Typical value
# input neurons	One per input feature (e.g. $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	mostly ReLU
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE

Hyperparameter	Binary Classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output activation	Logistic	Logistic	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy

Implementatie – stappen:

1. Dataset inladen
2. Trainingset opsplitsen in een validatie- en een trainingset.
3. Feature scaling
4. Keras Sequential API inladen:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

Model maken in Keras:

- We maken een nieuw object aan.
- De Flatten-laag stelt de eerste laag voor. Deze laag gaat instaan voor het omzetten van de inputlaag naar een eendimensionale array. Geen parameters nodig.

We willen drie hidden layers toevoegen:

- De eerste twee lagen zijn **rechtgezette lineaire functies** en de laatste gebruikt een softmax functie.
- De laatste is **softmax** omdat we klassen hebben die **onafhankelijk** zijn.

Drie parameters: loss, optimizer en metrics.

Loss:

- We kiezen deze waarde omdat we *sparse* inputlabels hebben en omdat de klassen onafhankelijk zijn.

Optimizer:

- Hier gebruiken we een SGD om het model te trainen. Keras zal hier een backpropagatie algo gebruiken.

Metrics:

- Wat willen we meten? In dit geval de nauwkeurigheid.

Keras gaat hier de loss berekenen en extra metrieken op het einde van iedere epoch gaan plaatsen. Zo weten we hoe goed het model draait.

Performance op validatieset > trainingset:

- Overfitting of een data mismatch

Je kan de epochs, params en keys opvragen.

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```


Implementatie Regression MLP:

1. Dataset ophalen, laden
2. Data splitsen
3. Data schalen

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()

X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data, housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

4. API inladen en model maken.

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.SGD(learning_rate=1e-3))
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
print(mse_test)
X_new = X_test[:3]
y_pred = model.predict(X_new)
```

Hyperparameter tuning:

- Neurale netwerken zijn flexibel, maar dat betekent dat er veel hyperparameters zijn die getweakt moeten worden.

<u>Aantal hidden layers</u> <ul style="list-style-type: none">• Bij één laag krijg je al degelijke resultaten. Hiërarchische structuur volgen: <ul style="list-style-type: none">• Laagste lagen gebruiken voor simpele objecten zoals lijnen en cirkels.• Hoogste lagen en de outputlaag gebruiken voor de complexe structuren te verwerken. Bv.: gezichten.• Voordeel van het hergebruik van de laagste lagen. Deze blijven hetzelfde bij verschillende soorten toepassingen.	<u>Aantal neuronen per hidden layer</u> <ul style="list-style-type: none">• Hangt af van het type input/output dat je taak nodig heeft.• Je mag hetzelfde aantal neuronen gebruiken.• Het aantal neuronen stelselmatig verhogen tot het netwerk begint te overfitten.	<u>Learning rate</u> <ul style="list-style-type: none">• Belangrijkste hyperparameter• Begin met een kleine learning rate en verhoog stelselmatig tot een grote waarde.• Vermenigvuldig de learning rate met een constante factor bij iedere iteratie.
	<u>Batch size</u> <ul style="list-style-type: none">• Heeft een impact op de snelheid van het model en de training time.• Voordeel is dat je grote sizes kan verwerken zodat je meer instanties per seconde ziet.• Grote sizes leidt tot grote training instabilites.	<u>Optimizer</u> <ul style="list-style-type: none">• Problemen tegengaan zoals het vanishing gradients probleem.• Non-saturating activation functions.• Meer in chapter 11.

H11: Diepe neurale netwerken (DNN)

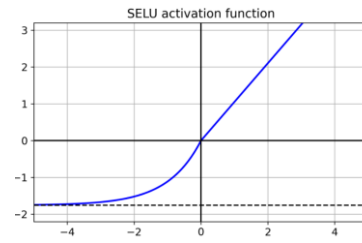
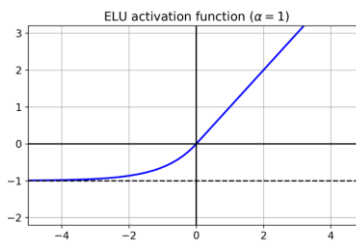
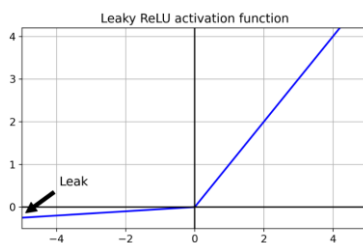
Problemen met het trainen van een DNN:

Vanishing gradients probleem	Niet genoeg trainingsdata of te kostelijk om te labelen	Veel parameters leidt tot overfitten en zeker wanneer er te veel noise is of wanneer er niet genoeg trainingsinstanties zijn.	Training is te traag
-------------------------------------	--	--	-----------------------------

Vanishing Gradients Problem:

- De graden worden alsmaar kleiner en kleiner of groter en groter wanneer je teruggaat in de DNN bij het trainen.
 - Graad die groter en groter wordt: **exploding gradients**
 - Graad die kleiner en kleiner wordt: **vanishing gradients**
- Gevolg: de laagste lagen zijn moeilijker te trainen
 - Oplossing: *Non saturating activation functions*
 - Oplossing: *Batch normalization*

Non-saturating activation functions:



Model maken:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, kernel_initializer="he_normal"),
    keras.layers.LeakyReLU(),
    keras.layers.Dense(100, kernel_initializer="he_normal"),
    keras.layers.LeakyReLU(),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(learning_rate=1e-3),
              metrics=["accuracy"])
```

Leaky ReLU:

- De Leaky laag toevoegen direct na de laag waarop je dit wilt toepassen.
- 'he_normal' wordt hier gebruikt als initialisator voor de gewichten.

Model aanpassen:

```
keras.layers.Dense(10, activation="elu")
```

ELU:

- Bij de dense laag pas je de activation-parameter aan.

Model maken:

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="selu",
                             kernel_initializer="lecun_normal"))
for layer in range(99):
    model.add(keras.layers.Dense(100, activation="selu",
                                  kernel_initializer="lecun_normal"))
model.add(keras.layers.Dense(10, activation="softmax"))

model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(learning_rate=1e-3),
              metrics=["accuracy"])
```

SELU:

- Als alle hidden layers een SELU activatiefunctie gebruikt dan zal het netwerk zichzelf normaliseren.
- De output zal dan een gemiddelde van 0 en standaardafwijking van 1 behouden. Dus een oplossing
- Dit noemt self-normalisatie.

Voorwaarden op SELU-lagen:

- De input-features moeten gestandaardiseerd zijn.
- Het gewicht van de hidden layers moet met een *LeCun normal initialisation* aangemaakt worden. (parameter)
- De architectuur moet opeenvolgend zijn. Geen loops.

Schalen:

```
pixel_means = X_train.mean(axis=0, keepdims=True)
pixel_stds = X_train.std(axis=0, keepdims=True)
X_train_scaled = (X_train - pixel_means) / pixel_stds
X_valid_scaled = (X_valid - pixel_means) / pixel_stds
X_test_scaled = (X_test - pixel_means) / pixel_stds
```

Batch normalization

Het vanishing/exploding gradient probleem tegengaan:

- De output van de laag normaliseren of batch normalisation.
- Operation toevoegen voor of na de activatiefunctie van iedere hidden layer.
- Iedere input wordt gecentreerd op nul en genormaliseerd.

<pre>model = keras.models.Sequential([keras.layers.Flatten(input_shape=[28, 28]), keras.layers.BatchNormalization(), keras.layers.Dense(300, activation="relu"), keras.layers.BatchNormalization(), keras.layers.Dense(100, activation="relu"), keras.layers.BatchNormalization(), keras.layers.Dense(10, activation="softmax")])</pre>	<p>We voegen een <i>batch normalisation</i> toe tussen de Dense-lagen.</p> <ul style="list-style-type: none">• Er zijn geen parameters nodig.
---	---

Getrainde lagen recyclen

Veel beter om bij een DNN lagen opnieuw te gebruiken.

- Het hergebruiken van de laagste lagen op een netwerk noemt **transfer learning**.
- Zorgt voor minder ruimte en een snellere training.

<p>Zeker veranderen:</p> <ul style="list-style-type: none">• De outputlaag. De kans is groot dat er een verschillend aantal outputs nodig zijn voor de taak.• De bovenste lagen zijn gebouwd voor specifieke taken. Het zijn high-level features die sterk kunnen verschillen. <p>Lagen hergebruiken:</p> <ul style="list-style-type: none">• De onderste lagen die noch high-level, noch intermediate-level zijn.	
---	--

<pre>model_A = keras.models.load_model("my_model_A.h5") model_B_on_A = keras.models.Sequential(model_A.layers[:-1]) # model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid")) model_A_clone = keras.models.clone_model(model_A) model_A_clone.set_weights(model_A.get_weights()) model_B_on_A = keras.models.Sequential(model_A_clone.layers[:-1]) model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))</pre>	<p>Eerst laad je het model op. Daarna haal je alles behalve de outputlaag op en sla je die onder een nieuwe variabele. Als laatste voeg je op het model (zonder outputlaag) een nieuwe dense laag toe.</p> <p>De lagen worden nu gedeeld dus beide modellen zullen getraind worden. Om dit te voorkomen moeten we een model bouwen op een kloon.</p>
---	--

Snellere optimizers:

GD optimizer vervangen door een van de volgende:

- Momentum optimization
- Nesterov
- AdaGrad
- RMSProp
- Adam and Nadam

Samenvatting:

Vijf manieren om een trainingset te versnellen:

- Initialisatiestrategie toepassen voor de gewichten
- Activatiefunctie aanpassen
- Batch normalization gebruiken
- Delen van een vooraf getraind netwerk gebruiken
- Snellere optimizer gebruiken

Overfitten vermijden:

Dropout:

- Bij iedere trainingsstap wordt er een neuroon, inclusief de inputneuronen, mogelijk uit het model gegooit.
- Tijdens de trainingstap wordt de neuroon genegeerd, maar de volgende stap kan deze weer ingezet worden.
- De dropout rate ligt vaak tussen 10 en 50%

H13: Feature Engineering in NLP

NLP:

- Teksten door machines laten interpreteren. Het laat een machine tekst lezen.
- Tekst mining:
 - Waardevolle tekst uit grote hoeveelheden tekstmateriaal te halen.
 - Bv.: achterhalen of een document over auto's gaat, bepalen of een bericht spam is
- Feature engineering:
 - Rauwe data nemen en die omvormen tot een voorspellend model voor ML of deep learning.
 - Machines moeten hier rekening houden met dubbelzinnigheden van de menselijke taal. Bijvoorbeeld homoniemen.

Taal achterhalen:

<pre>from langdetect import detect detect("Should I wear a mask when I'm exercising outside?")</pre>	
--	--

Tokenization:

- Strings in tokens opsplitsen om zo kleine structures of units te krijgen:
- String meegeven → word_tokenize(tekst)

Stop word removal:

- Stopwoorden verwijderen.
- Opm.: dit is taalspecifiek.
 - Hou een set bij van alle stopwoorden. Sets gaan sowieso geen dubbels hebben.
 - Je moet hiervoor eerst de taal meegeven → set(stopwords.words('english'))
- Kan je doen met een eigen functie.
 - Doorloop alle woorden (splits ze op met tokenize) en verwijder alles dat daarnet is opgeslaan.