En kort introduktion til Maskinlæring; Størrelse af gletsjere

STEMAcademyML

November 2025

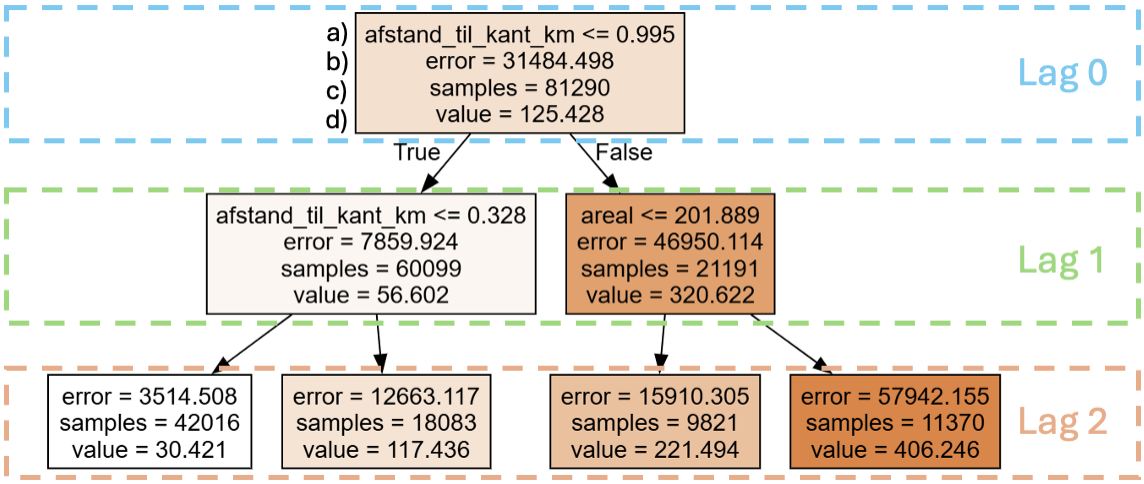
I fysik vil vi ofte gerne bruge eksisterende data til at forudsige ting. Hvis vi har rigtig meget forskelligt data, kan det være svært for os som mennesker at overskue. Her kan vi bruge maskinlæring og kunstig intelligens til at hjælpe os. Det kan ske både inden for klassifikation af himmellegemer, partikler i partikelfysik og gletsjerstørrelser, som er det vi vil kigge på her. Dataen som vi har adgang til har nogle måske kryptiske navne, men det betyder ikke noget. Vi behøver ikke forstå hvad de betyder, for at få en Machine Learning algoritme til at lære af det, og fortælle os hvilke variabler der er vigtige.

1 Introduktion

På Københavns Universitet er der lige blevet oprettet et gletsjervidenskabsfakultet hvor man vil prøve at bestemme dybden på alle gletsjere i hele verden. I praksis er dette rigtig svært at måle, da man ville skulle bore adskillige hundrede meter ned i iskapper verden over. Til gengæld har du fået adgang til et datasæt hvor dette er blevet gjort på en lille andel af verdens gletsjere samt andre målinger på gletsjere såsom deres beliggenhed, skråning, bevægelseshastighed og lignende. Opgaven ligger i at kunne sige noget om dybden på en gletsjer, hvis vi ved nogle andre ting om den. Du kunne nu godt tænke dig et skrive en ligning for dybden, men hvordan skulle sådan et udtryk se ud? Hvordan ved vi hvad sammenhængen er for eksempel mellem dens bevægelseshastighed og dens dybde? At skrive sådan en funktion direkte ville være ekstremt svært (eller rettere sagt, umuligt). I stedet kan vi lade computeren selv lære sammenhængene ud fra eksempler dvs. data. I fysik bruges machine learning ofte til 2 forskellige formål. Det ene, **Classification**, bruges til at afgøre, hvilken kategori noget tilhører, fx om en gletsjer ligger i europa eller ej. Den anden, **Regression**, bruges til at forudsige en konkret værdi, fx dybden på en gletsjer. I dette eksempel vil vi kigge på regression.

2 Decision trees

Et **Decision Tree** er bygget op af lag og grene. Ved hver gren stiller den et spørgsmål, og bevæger sig ned i det næste lag baseret på om spørgsmålet er sandt eller falsk. Og ved at lære af en masse data, kan den finde ud af hvilke spørgsmål der er bedst at stille, fordi den kender det rigtige svar.



Figur 1: Et decision tree med to lag. Hver boks repræsenterer en gruppe af gletsjere og viser: a) hvilket spørgsmål (også kaldet **split**) der bedst opdeler gletsjerne i gruppen, b) hvor stor fejl (**loss** eller squared error) der er, hvis man gætter gennemsnitsdybden for gruppen, c) hvor mange gletsjere der hører til gruppen, og d) hvad den gennemsnitlige dybde er for gletsjerne i gruppen.

Hvis algoritmen ikke får lov til at stille nogen spørgsmål, kan den kun gætte på gennemsnitsdybden for alle gletsjere i datasættet som i dette tilfælde er ca. 125m. Det er en meget simpel model, der altid vil gætte det samme tal.

Giver vi den lov til ét lag, kan den stille ét spørgsmål og dele data i to grupper. I vores tilfælde stiller algoritmen spørgsmålet: "Er afstanden til kanten mindre end 0,995 km?". Hvis afstanden er mindre, gætter den på 56.6m, og er den større, gætter den på 320.6m.

Med to lag kan træet stille et nyt spørgsmål i hver gruppe. I dette tilfælde spørger den igen om afstanden til kant men om areal i den anden gruppe. Spørgsmålet behøver ikke være det samme for hver gren. Algoritmen prøver sig frem med en masse forskellige spørgsmål, og vælger det der giver den bedste opdeling, det der gør at gættet er tættest på den sande dybde. For målinger tæt på kanten giver det mening at spørge det samme spørgsmål igen da der f.eks kunne være mange målinger i denne kategori. Mens for målinger længere væk fra kanten kunne det tænkes algoritmen ville deducere gletsjerens areal da det måske kunne give anledning til dybere gletsjere. Nu har træet fire mulige gennemsnitsdybder at gætte ud fra, og sådan kan vi blive ved til vi har tilstrækkeligt mange måder at inddele og gætte på.

2.1 Optimering: Boosted decision tree (BDT)

Data kan være meget komplekst, med mange variabler og stor variation i værdier. Det gør det svært at lave ét træ, der rammer godt. Man skal fx bestemme hvor dybt træet må være, hvor mange gletsjere der højest/mindst må være i en kasse, og hvor store fejl der er acceptable. På fagsprog kaldes det hyperparametre.

I stedet for at satse på ét perfekt træ, kan man lade algoritmen bygge mange små træer, hvor hvert nyt træ fokuserer på de fejl, de tidligere træer lavede. På den måde forbedres modellen trin for trin. Denne metode kaldes **Boosted Decision Trees**, og den giver typisk meget mere præcise forudsigelser end ét enkelt træ.

3 Neurale Netværk

En anden type algoritme, som man ofte hører om, er **Neurale Netværk (NN)**. Navnet kommer fra, at opbygningen minder om måden neuroner i hjernen sender signaler på. Ligesom et decision tree består af lag, har NN også lag af noget vi kalder **noder**. I stedet for ja/nej-spørgsmål fungerer de som justerbare knapper. Hver node har en **vægt**, der bestemmer, hvor meget et input betyder for resultatet.

3.1 Simpelt NN

Lad os tage et helt simpelt netværk, med kun 1 node. Som nævnt i indledningen, vil vi gerne skrive en funktion, der kan give os en dybde, og det kan et NN hjælpe os med. Hvis vi for eksempel har 3 variable, for eksempel afstand til kanten, areal og hastighed, bliver der i noden lavet ligningen:

$$a \cdot \text{afstand\_til\_kant} + b \cdot \text{areal} + c \cdot \text{hastighed} + d = \text{dybde}. \tag{1}$$

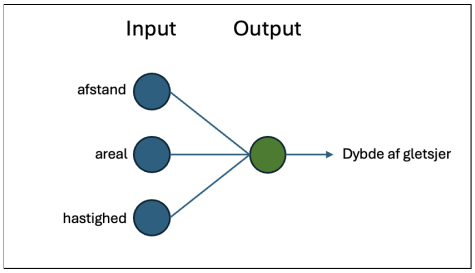
Her er a, b og c vægte, som modellen lærer automatisk, mens d er en **bias**, der fungerer som et startpunkt (ligesom b i $y = ax + b$). Bias gør, at modellen ikke behøver gå gennem nul og dermed kan lave mere realistiske forudsigelser.

NN arbejder bedst med værdier mellem 0 og 1, så derfor er det vigtigt at dataen først bliver normaliseret. Hvis vi normaliserer alle værdier, kan et eksempel se sådan ud:

$$\text{neuron} = 0.9 \cdot 0.2 + 0.8 \cdot 0.6 - 0.1 \cdot 0.12 + 0.03 = 0.678 \tag{2}$$

c er for eksempel negativ fordi en gletsjer der bevæger sig meget hurtigt ikke kan være særligt dyb og høje gletsjere oftest er tungere og dermed langsommere. Til gengæld betyder dette mindre end arealet, der er en bedre indikator, og derfor har b en højere vægt.

Dette simple netværk ville se ud som i figur 2.



Figur 2: Et simpelt neuralt netværk, hvor vi har 3 input og en enkelt node. I noden laves ligning (1), hvor hver input variabel ganges med en vægt og lægges sammen plus en bias. Outputtet fra noden er modellens bud på dybden af gletsjeren.

3.2 NN med flere lag

Det er en god start, men i virkeligheden er der mange flere forhold, der spiller ind. For eksempel kan de samme variable betyde noget forskelligt afhængigt af, hvor på gletsjeren man måler. Tæt på midten bevæger isen sig typisk langsommere og kan derfor være dybere, mens isen nær kanten ofte bevæger sig hurtigere og er tyndere. Det betyder, at "hastighed" og "afstand til kanten" kan påvirke dybden på forskellige måder alt efter, hvor man befinder sig — og et neuralt netværk med flere lag kan lære at tage højde for netop sådan nogle sammenhænge.

Det er her vi introducerer det der kaldes "**Hidden layers**". Et hidden layer er et lag af noder, der ligger mellem input og output (se figur 3). Hver node i "hidden layer", laver samme ligning som (1), med de input den får, og sender så et tal videre til det næste lag. Men inden et tal sendes videre til det næste lag, bruges noget der kaldes en **activation function**.

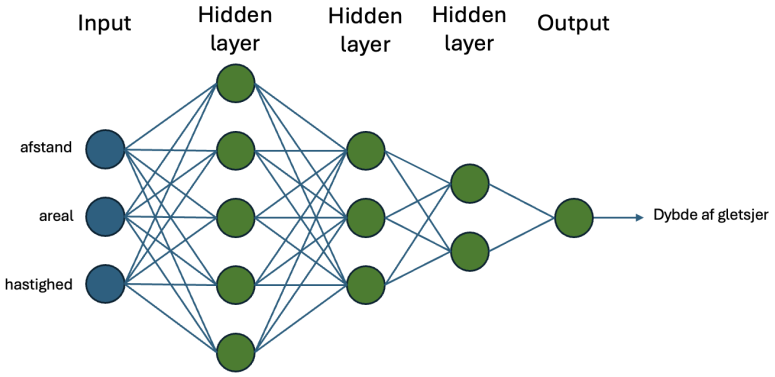
En af de mest brugte aktiveringsfunktioner hedder **ReLU** (Rectified Linear Unit):

$$f(x) = max(0, x) \tag{3}$$

Hvis ligning (1) giver en negativ værdi, vil $f(x)$ sende 0 videre til næste lag, mens positive værdier sendes videre som de er. Det betyder, at nogle noder "slukker", og andre "tænder", afhængigt af hvilken type gletsjer eller hvilket område vi ser på. På den måde kan forskellige noder lære at reagere på forskellige mønstre i dataen — for eksempel langsomt bevægende is nær midten af gletsjeren eller tynd, hurtig is tæt på kanten. På den måde kan netværket for eksempel have en høj vægt på "hastighed", når vi er tæt på kanten, men en lavere vægt, når vi er midt på gletsjeren, ved at tænde og slukke for de noder, der repræsenterer netop disse sammenhænge.

3.3 Opbygning af NN

Et neuralt netværk kan foreksempel se sådan ud:



Figur 3: Et eksempel på et neuralt netværk med 3 input parametre og tre skjulte lag og et output. I hver node laves ligning (1), men i noderne i "hidden layer"bliver resultatet også pakket ind i activation function, før det sendes videre.

Her har vi 3 input parametre. Herefter tre hidden layers, hvor ligninger som (1) og (2) bliver udført med en activation function, og til sidst et output lag, der giver det endelige bud på dybden.

Hvis et hidden layer har n input (noder i det foregående lag) og m noder i sit eget lag, har det

$$n \times m + m$$

parametre.

Så for første skjulte lag har vi $3 \times 5 + 5 = 20$ parametre, for andet skjulte lag $5 \times 3 + 3 = 18$, tredje skjulte har $3 \times 2 + 2 = 8$ og for output laget $3 \times 1 + 1 = 4$. I alt altså 50 parametre. Vi kan også sige, at noderne svarer til antallet af **bias**, mens stregerne imellem svarer til antallet af **vægte**.

Til sidst justerer algoritmen alle vægte og biaser, så forskellen mellem forudsagte og sande værdier bliver så lille som muligt. På den måde kan neurale netværk finde mønstre, der er alt for komplekse til, at vi selv kunne skrive dem som en enkelt formel.

4 Hvordan ved algoritmen hvilket og hvor mange spørgsmål den skal stille?

En algoritme skal bruge noget til at måle, hvor godt den gør det, så den ved hvilke spørgsmål den skal stille, eller hvilke vægte den skal tildele de forskellige parametre. I Machine Learning kalder vi det en **Loss function**.

For et decision tree prøver algoritmen mange mulige spørgsmål og beregner, hvor meget hvert spørgsmål kan reducere fejlen. Det spørgsmål, der giver den største forbedring, bliver valgt. Processen gentages i hver ny gren, indtil træet er "dybt nok"eller ikke længere forbedres.

For et neuralt netværk fungerer princippet på samme måde, men i stedet for at vælge spørgsmål prøver algoritmen forskellige vægtkombinationer i de mange noder. Den beregner herefter sin loss (fejl), og justerer vægtene lidt ad gangen, så fejlen bliver mindre for hver iteration. Denne proces kaldes træning, og den metode, der bruges til at ændre vægtene, kaldes **gradient descent**.

De to mest brugte loss functions for regression er:

- **Absolute error (MAE)**: Her bruges den absolutte forskel mellem forudsagt og sand værdi.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|\text{sande værdi}_i - \text{forudsagte værdi}_i\right|$$

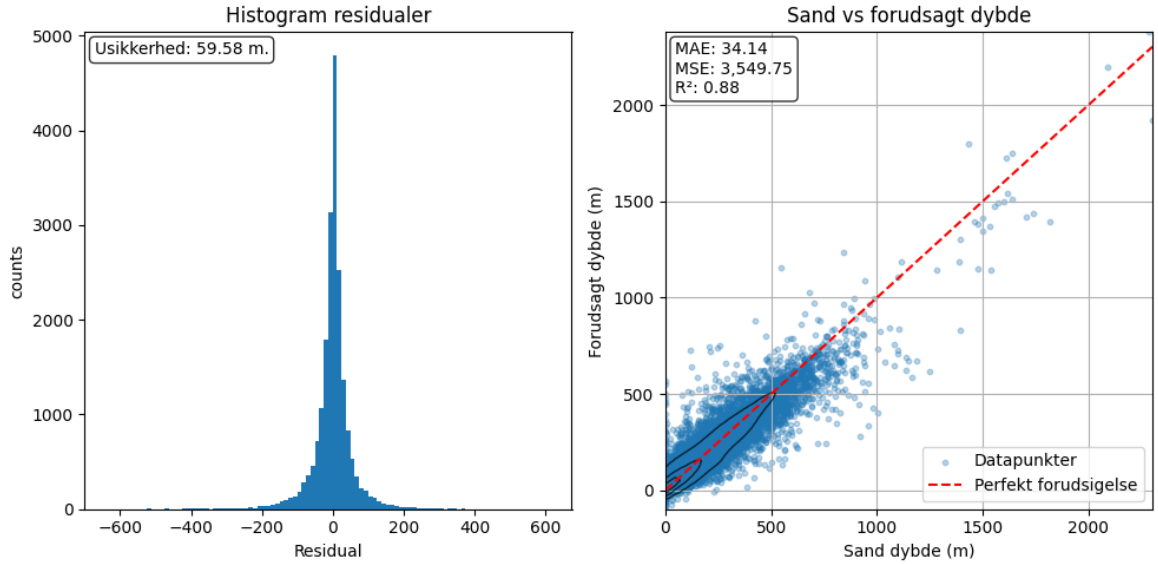
- **Squared error (MSE)**: Her kvadreres forskellen mellem forudsagt og sand værdi. Punkter der ligger langt fra normalen (outliers) bliver straffet hårdere, og derfor bruges MSE ofte når vi har mange outliers.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\text{sande værdi}_i - \text{forudsagte værdi}_i\right)^2$$

4.1 Hvordan kan vi se vi hvor godt algoritmen klarer sig?

Når nu vi har fået opbygget vores decision tree eller neurale netværk, vil vi gerne finde ud af hvor god den er til at forudsige. Det gør vi ved at give modellen ny data, som den ikke har trænet på, og bede den om at forudsige, og så sammenligne med de rigtige værdier. Typisk deler man datasættet i to dele: en del til **træning**, hvor modellen får lov at lære sammenhængene, og en mindre del til **test**, som gemmes væk, indtil modellen skal vurderes. På den måde sikrer vi, at modellen ikke bare husker de præcise svar, men faktisk har lært et mønster, der også virker på ny data.

For regression vil man typisk visualisere performance som vist nedenfor: Til venstre er plottet forskellen mellem forudsagte og sande værdier (residualer). Her vil vi gerne have, at punkterne ligger tæt på nul, så forudsigelserne ikke systematisk rammer for højt eller for lavt og at spredningen er så lav som muligt. Til højre er plottet forudsagte værdier mod sande værdier. Her vil vi gerne have, at punkterne ligger tæt på linjen $y = x$, som betyder, at forudsigelsen og den sande værdi er den samme.



Figur 4: Til venstre ses et histogram over residualerne (forskellen mellem forudsagt og sand dybde). De fleste residualer ligger tæt på nul, hvilket betyder, at modellen oftest rammer rigtigt, men der findes også enkelte store afvigelser. Til højre ses de forudsagte dybder mod de sande dybder. Den røde linje viser en perfekt forudsigelse ($y = x$). Punkterne ligger fordelt rundt om den for alle værdier af den sande dybde.

5 Er alle informationer lige vigtige?

Når vi har trænet vores model, er det interessant at finde ud af, hvilke variable den lægger mest vægt på. I et **decision tree** kan vi ofte allerede se det ud fra de spørgsmål, træet stiller først, for eksempel at afstand_til_kant og areal har stor betydning for dybden. I et **neuralt netværk** er det sværere at se direkte, men her kan vi kigge på, hvor meget de enkelte vægte påvirker resultatet.

En mere generel metode, der virker for næsten alle typer modeller, kaldes **Permutation Importance**. Her tager man den trænede model og "roder" med én variabel ad gangen. Det vil sige, man blander dens værdier tilfældigt, så den mister sin sammenhæng med resten af dataen. Hvis modellens præcision falder meget, betyder det, at den variabel var vigtig for forudsigelsen. Hvis vi fx bytter rundt på afstanden til kanten, og modellens præcision falder markant, betyder det, at afstand_til_kant er vigtig. Hvis der næsten ingen forskel er, betyder det, at modellen ikke brugte den variabel særlig meget.

Det fantastiske ved den her metode er, at vi ikke behøver at vide noget som helst om systemet på forhånd. Modellen finder helt selv ud af, hvilke ting der betyder mest, og det kan være mønstre, som folk normalt kun opdager efter mange års erfaring. Pludselig kan vi se, hvilke variable der faktisk er væsentlige, eller hvad der gør forskellen mellem to typer af data, uden at have nogen forhåndsviden. Det er her, Machine Learning virkelig bliver spændende: når den lærer os noget nyt om verden, som vi ikke selv havde set.