1 Introduktion

Martin har lige startet sin første ejendomsmægler-virksomhed. Han har ikke noget erfaring, og det kan tage mange år at få en god fornemmelse for hvad husene skal prissættes til. Heldigvis har han fået adgang til en database over huse som der tidligere er solgt. I databasen er der oplysninger så som husets størrelse, postnummer, hvor langt der er til den nærmeste skole m.m., og hvilken pris huset blev solgt for. Men han gider ikke sidde og studere alle de tidligere cases. I stedet har han hørt at man kan lave en machine learning algoritme, der kan sige hvad husene skal koste. Han har aldrig lavet machine learning før, men hvor svært kan det være? Og på rigtig direktør manér, har han besluttet at uddelegere opgaven, til sin nye billigt ansatte studentermedhjælper, dig.

Vi prøver at bygge en funktion der kan tage mange parametre; placering, størrelse, husnummer m.m. (du er sikkert kun vant til en), og spytter en værdi ud; salgsprisen - det kan være svært (umuligt). Hvordan kan vi få computeren til at skrive sådan en funktion?

Hvis du for eksempel skulle gætte hvor meget en liter mælk koster i supermarkedet, hvad ville du så gerne vide? Det kunne være relevant at spørge om mælken var økologisk, om den var produceret i Danmark eller udlandet, eller hvilket supermarked der solgte mælken. Du ved, at de er relevante, fordi du har set priser mange gange før. Det samme gør sig gældende ved machine learning. Vi giver modellen en masse eksempler hvor den får "svaret" at vide, og kan bruge det til at vurdere hvilke faktorer der er vigtige.

I science bruges machine learning ofte til 2 forskellige formål. Den ene, **Classification**, hvor den skal vurdere om noget er enten eller. I det her tilfælde kunne det være at den ud fra hvad den ved om husene, skulle vurdere om der er tale om et hus eller en lejlighed, og på den måde gruppere dataen. Den anden, **Regression**, bruges når vi gerne vil gætte noget specifikt som kan være mange værdier, som for eksempel prisen på hvert enkelt hus. 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.

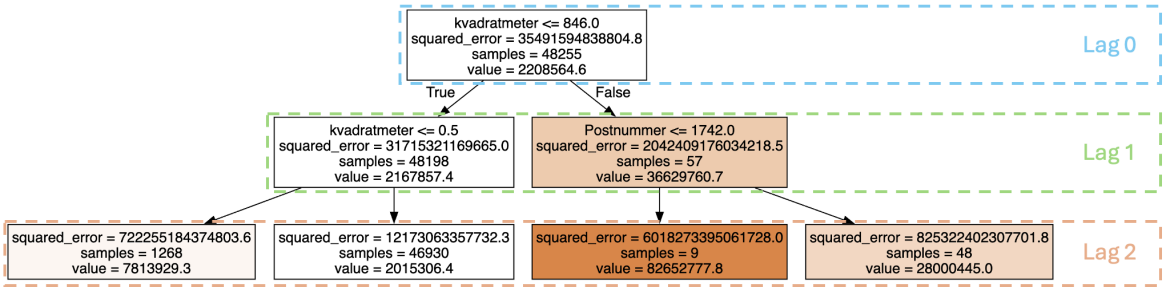

Figure 1: Enter Caption

Hvis algoritmen ikke får lov til at stille nogen spørgsmål, kan den kun gætte på gennemsnitsprisen for alle huse i datasættet som i dette tilfælde er ca. 2,2 mio. kr. 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 spørger træet: "Er huset større end 846 km^2?". Så gætter det på gennemsnitsprisen inden for hver gruppe. Det er stadig groft, men allerede bedre end kun ét gennemsnit.

Med to lag kan træet stille et nyt spørgsmål i hver gruppe. Store huse bliver fx opdelt efter postnummer, mens små huse deles yderligere efter størrelse (for at skelne mellem rigtige huse og tomme grunde). Nu har træet fire mulige gennemsnitspriser at gætte ud fra, og præcisionen forbedres.

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 huse

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 måde at lave en algoritme på, som ofte er den man hører om, er Neurale Netværk (NN). Navnet kommer fra at opbygningen af det, minder om den måde vores neuroner i hjernen snakker sammen på. På samme måde som et decision tree er der forskellige lag og vi kan styre hvor mange lag der er, men nu er det ikke kun sandt eller falsk, i stedet fungerer noderne som knapper der kan fintunes.

(indsæt illustration)

Tænk tilbage på den funktion vi gerne ville skrive. Hvis vi havde 3 parametre, for eksempel postnummer, kvadratmeter og husnummer, og så ville vi gerne skrive en funktion der kunne give os prisen. Den kunne for eksempel være:

$$a \cdot \text{postnummer} + b \cdot \text{kvadratmeter} + c \cdot \text{husnummer} = \text{salgspris}. \tag{1}$$

Hvor a, b og c er forskellige vægte vi tildeler de forskellige parametre. For at det giver mening, normaliserer vi først alle parametre til at ligge mellem 0 og 1. Et specifikt hus kunne fx have postnummer 2000, 60 kvadratmeter og nummer 12. Normaliseret kunne det være 0.2, 0.6 og 0.12 (afhængigt af hele datasættets minimum og maksimum). Vi kunne forestille os, at postnummer og kvadratmeter har høj vægt, mens husnummer ikke betyder så meget. Så kunne en formel se sådan ud:

$$0.9 \cdot 0.2 + 0.7 \cdot 0.6 + 0.1 \cdot 0.12 = 0.612 \tag{2}$$

altså ville gættet være 6.12 mio. Men hvad hvis vi gav algoritmen lov til at lave de her ligninger mange gange efter hinanden? Og den næste ligning var afhængig af resultatet af den forrige ligning? Så ville vi pludselig have mange vægte og mange "knapper" at finjustere, og på den måde kunne modellen lære at komme med meget præcise gæt. Det er grundidéen i et neuralt netværk: mange lag af vægte, der justeres, indtil forudsigelsen passer godt.

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

En algoritme skal bruge noget til at måle efter 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 prøver den forskellige vægte i de forskellige noder, og finder ud af hvilke kombinationer der giver den mindste fejl.

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 den 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 den noget ny data, som den ikke har trænet på, og bede den om at forudsige, og så sammenligne med de rigtige værdier. Typisk tager man en del af den data der skal bruges til træning med svar og gemmer, så algoritmen ikke har set den data før, og vi kender svarene.

For regression kan vi visualisere det ved at plotte forskellen på forudsagte og sande værdi, eller plotte forudsagte vs. sande værdi. I det første tilfælde vil vi gerne have at forskellen er så tæt på 0 som muligt. I den anden tilfælde vil vi gerne have punkterne følger en ret linje y=x, så forudsagte og sande værdi er den samme.

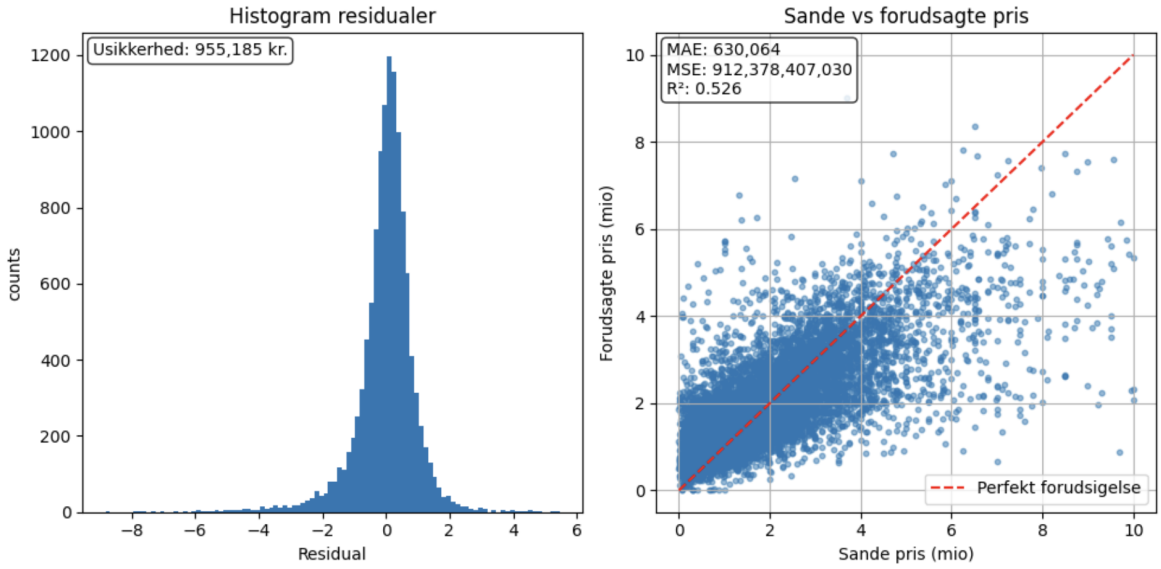

Figure 2: Enter Caption

5 Appendix

Her kommer et appendix til lærerne hvor nogle af funktionerne of dele af koden er forklaret i dybden.

vis struktur af større træ - uden tekst