Dataset: Heart

Classification & NN Classification

Markus Kinn

I dette dokumentet kommer jeg til å diskutere valg jeg har tatt og sammenligne resultater fra ulike algoritmer. Jeg har ett dokument for hvert datasett.

For å forbedre resultatene på modellene har jeg prøvd å implementere så mye so mulig fra *Machine Learning Performance Improvement Cheat Sheet.*

**Generelt om datasettet:**

Datasettet går ut på å predikere om en person har høy sannsynlighet for hjerte feil eller ikke, basert på disse attributtene: Age, Exng, Ca, Cp, Trtbps, Chol, Fbs, Rest\_ecg og Thalach. Forklaring av hva hvert begrep betyr finnes i EDA notebook.

Datasettet har ingen feil. For mer grundig gjennomgang av datasettet, se EDA i Oblig 2 mappen.

Jeg prøvde å one-hot encode alle kategoriske kolonner, men dette ga dårligere resultat for alle algoritmene.

**XGBoost:**

For Heart datasettet trengte algoritmen rundt regnet ingen databehandling.

**Før tuning:**

**Text

Description automatically generated**

**Valg av tuning strategi:**

Opprinnelig testet jeg ut en svært sekvensiell tuning strategi hvor jeg kjørte griden på kun en og en variabel. Denne strategien brukte betydelig kortere tid enn den litt mindre sekvensielle strategien jeg brukte, men presterte som regel litt dårligere.

Jeg testet også ut Halving grid search, men denne virker som fungerer kun bedre på store datasett, noe jeg ikke har.

Til slutt testet jeg ut Random Search. I random serach kan jeg dytte inn ekstremt mange parametere og verdier, samtidig som resultatet blir ofte bra og på kort tid. Det er derfor kun denne strategien jeg velger å bruke i de fleste notebookene. Random search blir også tunet mtp f1-score.

For dette datasettet og denne algoritmen presterte min sekvensielle metode best og har derfor kun med bilder fra den i dette dokumentet.

**Etter tuning:**

**Text

Description automatically generated**

**Diskusjon av resultater:**

Før tuning var modellen ekstremt over-fitted; den presterte altså mye bedre på trenings data enn hva den gjorde på test data, men resultatene var fremdeles generelt bra.

Etter tuning klarte jeg å få ned over-fittingen, ved at trenings resultatet gikk ned, men test resultatet gikk opp. Cross validation scoren forbedret litt, med en oppgang på rundt 0.5%. f1 scoren gikk også opp. Jeg kan dermed trygt konkludere med at modellen var bedre etter tuning enn før.

**Random Forest:**

For dette datasettet fikk jeg den nøyaktig samme databehandlingen som ved XGBoost.

**Før tuning:**

**Text

Description automatically generated**

**Valg av tuning strategi:**

Her testet jeg ut to strategier: Grid search, men noe sekvensielt og Random search. Begge strategiene presterte nesten helt likt. For dette datasettet og denne algoritmen klarte Random Search å prestere best, jeg tar derfor kun med bilder fra denne strategien.

**Etter tuning:**

**Text

Description automatically generated**

**Diskusjon av resultater:**

Baseline modellen er svært over-fitted, men presterer generelt bedre enn baseline XGBoost modellen.

Etter tuning så presterte modellen bedre på test data enn trenings data. Dette er normalt sett et tegn på at noe ikke er som det bør være. Jeg har prøvd å fikse det, men uten hell. Cross validation scoren synker, men får en bra økning i alle andre scorer.

**Gaussian Naive Bayes:**

For denne algoritmen kunne det vært en ide å gjøre om alle kontinuerlige kolonner til kategoriske, men jeg ser at baseline modellen prestert rundt regnet likt som alle andre baseline modeller og tror derfor at det er lite å hente fra å gjøre dette. Hadde jeg hatt bedre tid ville jeg prøvd på dette.

**Før tuning:**

**Text

Description automatically generated**

**Valg av tuning strategi:**

Her brukte jeg kun et vanlig grid search da algoritmen ikke har mange parametere å tune

**Etter tuning:**

**Text

Description automatically generated**

**Diskusjon av resultater:**

Baseline modellen presterer nokså bra i forhold til de andre algoritmene, men her er test score betraktelig høyere enn trenings score. Dette er ikke ideelt og er ofte en indikator på at noe ikke er som det skal. Etter litt research klarer jeg ikke å finn ut av hvorfor dette fenomenet oppstår.

I tuning modellen klarer jeg ikke å øke noen av scorene. Dette kan komme som et resultat av dårlig tuning av meg, eller at dette er rett og slett så bra modellen klarer med den dataen jeg gir den.

**Support Vector Machine:**

For denne algoritmen måtte jeg skalere dataen. For dette brukte jeg RobustScaler() funksjonen. Resultatet blir betydelig bedre etter. Jeg tok også å one-hot encodet alle kategoriske kolonner

**Før tuning:**

**Text

Description automatically generated**

**Valg av tuning strategi:**

Her brukte jeg kun et vanlig grid search da algoritmen ikke har mange parametere å tune

**Etter tuning:**

**Graphical user interface, text

Description automatically generated**

**Diskusjon av resultater:**

Baseline modellen presterte ekstremt bra, men hadde noe over-fitting.

Etter tuning ble det 0.5% mindre over-fitting, men fikk generelt dårligere prestasjon på alle scorer unntatt cross validation score.

**Neural Network Architecture 1:**

For begge neurale nettverkene valgte jeg å normalisere alle verdiene, da flere kolonner hadde kontinuerlige verdier, deretter one-hot encode alle kategoriske variabler og deretter gjøre alle features om til tensors. Jeg kjørte modellene mange ganger for å finne de beste verdiene.

**Før tuning:**

**Text

Description automatically generated**

**Etter tuning:**

Jeg møtte på mange problemer med Automatisk hyperparameter tuningen når det gjaldt NN og endte derfor opp med å gjøre dette manuelt. Jeg dokumenterer derfor prosessen her:

Først bestemte jeg meg for epcohs som ble 200, av den grunn at dette er en fin balanse mellom tid og gevinst. Optimizer funksjonen ble Adam da jeg som regel har hatt best resultat med denne fra tidligere erfaring. Loss funksjonen ble binary crossentropy da dette er et klassifiserings problem med kun to klasser. Jeg gjorde dette for å kutte ned på mengden variabler jeg måtte teste og testet derfor variabler som kun hørte til selve NN-arkitekturen. Jeg vet at dette er langt fra optimalt, men fra mye testing på workshop oppgavene var det ofte disse verdiene som gikk igjen.

Etter det fant jeg ut hvilke activation functions som egnet seg for datasettet. Jeg visste fra før at Sigmoid ofte egner seg som activation function i output-laget for klassifiserings problemer. Jeg måtte da teste ut activation functions på alle de skulte lagene. For å spare tid brukte jeg den samme aktiverings funksjonen på alle skulte lag. Softmax, sigmoid, tanh og relu var de aktiverings funksjonene jeg valgte å teste. Av disse var relu best.

Etter activation function gikk jeg over til batch size. Her fant jeg ut at den beste verdien er 6. Etter at jeg senket den ned til 6 ble meste parten av over-fittingen borte.

Neste gikk jeg over til å teste antall neuroner i hvert lag. Her hadde det vært ekstremt nyttig med en automatisk versjon av dette, da det finnes enormt mange ulike kombinasjoner man kan velge mellom. Her valgte jeg å teste relativt tilfeldig. Kjapt skjønte jeg at en form for pyramide struktur er den beste, hvor neuronene øker mot midten og synker mot slutten.

Den beste modellen ble seende sånn ut:

Text

Description automatically generated

**Diskusjon av resultater:**

Baseline modellen presterte bra, men hadde noe over-fitting. Dette kan komme av både arkitekturen, mengden neuroner, samt batch size. Etter at jeg var ferdig med den manuelle tuningen klarte jeg å eliminere over-fittingen, samt øke test scoren. Tuning modellen er generelt veldig bra.

**Neural Network Architecture 2:**

Brukte det samme datasettet som i forrige neurale nettverk. Jeg kjørte modellene mange ganger for å finne de beste verdiene.

I denne arkitekturen valgte jeg å introdusere drop-out lag, samt redusere mengden «vanlige» skjulte lag fra arkitektur nummer 1.

**Før tuning:**

**Text

Description automatically generated**

**Etter tuning:**

Jeg fulgte den samme tuning prosessen som i den første arkitekturen.

Text

Description automatically generated

**Diskusjon av resultater:**

Begge modellene med denne arkitekturen fikk bedre resultater enn forrige arkitektur. Dette kan komme som et resultat av en annen kombinasjon av skjulte lag, men også grunnet drop-out lag sammen med høyere batch size.

Baseline modellen presterte nesten identisk til tuning modellen, men dette er grunnet små endringer i hyperparameterne. Tuning modellen presterte aller best, med en mikroskopisk nedgang i training score, men en god økning i test score.

**Sammenligning av alle resultater:**

Alle modellene gjorde det ganske bra, men for dette datasettet tok SVM en soleklar seier, både før og etter tuning. Jeg tror grunnen til dette er at SVM sies å prestere bedre enn andre algoritmer på små datasett, noe dette er. Dette kan også komme som ett resultat av at jeg ikke gjort en god nok jobb med hyperparameter optimaliseringen hos de andre algoritmene. Siden SVM hadde kun to interessante parametere å tune var det mye lettere å tune denne veldig bra, i motsetning til de andre modellene som hadde mange flere interessante parametere.