Dataset: Titanic

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 kom til å overleve Titanic ulykken, basert på disse attributtene: PassengerId, Pclass, Name, Age, SibSp, Parch, Ticket, Fare, Cabin og Embarked.

Datasettet har lite feil, noe som gjør at man ikke trenger å gjøre mye data pre-Processing. Det har derimot noen manglende verdier og noen kolonner som ikke trengs. For mer grundig gjennomgang av datasettet, se EDA i Oblig 1 mappen.

**XGBoost:**

XGBoost er en algoritme som håndterer de aller fleste tabell datasett. For Titanic datasettet trengte algoritmen rundt regnet ingen databehandling. Algoritmen håndterer null-verdier selv, men jeg valgte selv å håndtere disse, da jeg fikk hakke bedre resultat på min måte.

Jeg valgte å fortsette med Binning på alder kolonnen da dette ga hakke bedre resultat enn uten.

XGBoost takler heller ikke string input og jeg måtte derfor one-hot encode kolonnen Embarked.

Fra EDA så vi at både Fare og Embarked har dårlig distribusjon og jeg vurderte derfor å fjerne disse. Etter å ha testet ut å lage baseline modeller med ulike kombinasjoner å ha med og fjerne disse kolonnene fant jeg ut dette:

* Ha med begge:

Text

Description automatically generated

* Fjerne Embarked:

Text

Description automatically generated

* Fjerne Fare og Embarked:

Text

Description automatically generated

Fra dette kan vi se at det å beholde begge fører til mye over-fitting og generelt ok resultat. Fjerne Embarked, men beholde Fare fører til noe redusert over-fitting, men generelt bedre resultat. Fjerne begge fører til lite over-fitting, men generelt dårligere resultat. Her er det altså Fare som fører til så stor over-fitting.

Jeg valgte å gå videre med datasettet som har fjernet Embarked, men beholdt Fare da dette førte til de beste resultatene.

**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 falt valget på 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.

**Etter tuning (Random search):**

**Text

Description automatically generated**

**Diskusjon av resultater:**

Jeg klarte å øke alle scorene ganske mye, samt redusere over-fittingen drastisk. F1-scoren gikk minimalt ned, men sier meg fremdeles fornøyd med tuning resultatet basert på alle de andre scorene.

**Random Forest:**

For dette datasettet måtte jeg også fjerne Fare kolonnen for å kunne få de beste resultatene på baseline modellen.

**Før tuning:**

**Text

Description automatically generated**

**Valg av tuning strategi:**

Her prøvde jeg både min egne litt mindre sekvensielle metode og Random search, men klarte ikke å få bedre resultat med Random search enn med min egen metode. Jeg har derfor valgt å la begge strategiene være i notebooken, men viser kun resultatet av den beste metoden i dokumentet her.

**Etter tuning:**

**Text

Description automatically generated**

**Diskusjon av resultater:**

Relativt OK baseline modeler, men noe over-fitting. Meget fornøyd med resultatet etter tuning da jeg klarte å fjerne over-fittingen helt, samt øke alle andre scorer.

**Gaussian Naive Bayes:**

For denne algoritmen valgte jeg å gjøre noen endringer i datasettet. Primært gikk disse endringene ut på å fikse opp i kontinuerlige verdier og fjerne høyt korrelerte kolonner. I dette datasettet var det to kontinuerlige kolonner: Fare og Age. Det var også to høyt korrelerte kolonner: Plcass og Fare. Dette endte opp med at jeg lagde AgeGroups og fjernet Fare.

GaussianNB takler heller ikke string input og jeg måtte derfor one-hot encode kolonnen Embarked

**Før tuning:**

**Graphical user interface, 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:**

Denne algoritme slet ikke med over-fitting på baseline modellen eller den ferdig tunet modellen. Resultatet var for så vidt dårligere enn de to forrige algoritmene.

Jeg klarte å forbedre alle resultatene etter tuning, men det ble noe over-fitting etter tuning.

**Support Vector Machine:**

For denne algoritmen valgte jeg å fjerne Fare og AgeGroup, da disse to forverret resultatet med opptil 2%. Jeg måtte også skalere dataen for denne algoritmen. For dette brukte jeg RobustScaler() funksjonen. Grunnen til at dette er viktig for SVM er at den bruker distansen mellom observasjoner til å predikere (veldig kort fortalt).

**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:**

Tilnærmet ingen over-fitting på noen av modellene og bra resultater på baseline modellen. Her slet jeg med å øke resultatet etter tuning og fikk kun en liten økning i cross validation scoren, men en liten redusering i de andre scorene.

**Neural Network Architecture 1:**

For begge neurale nettverkene valgte jeg å fjerne kolonnen Fare grunnet høy korrelasjon med Pclass. Her måtte jeg også one-hot encode alle kategoriske variabler og deretter gjøre alle features om til tensors.

**Før tuning:**

**Text

Description automatically generated**

**Etter tuning:**

**?????**

**Diskusjon av resultater:**

Modellen presterte veldig bra og var presterte nummeret inntil helt likt som XGBoost modellen nevnt ovenfor. Fremdeles litt over-fitting, men ikke en problematisk mengde.

**Neural Network Architecture 2:**

Brukte det samme datasettet som i forrige neurale nettverk

**Før tuning:**

**Text

Description automatically generated**

**Etter tuning:**

**?????**

**Diskusjon av resultater:**

Presterte også ganske bra, men mitt forsøk på å introdusere et drop-out layer for å redusere over-fitting endte med totalt sett dårligere resultat.

**Sammenligning av alle resultater:**

Alle modellene gjorde det ganske bra, men de to som skiller seg ut er XGBoost etter tuning og det første neurale nettverket. Disse to modellene presterte ganske likt, men XGBoost tok seieren over alle algoritmene. XGBoost presterte bedre enn alle de andre algoritmene på alle scorene jeg tok med. Dette er en god indikator på at dette faktisk er den beste modellen.

Etter å ha sett på andres modeller, så ser det ut som det er rundt den scoren jeg fikk, som andre også som regel får.