Artificial Neural Network // Oblig 3

Skrevet av Stian Larsen, med hjelp i koden fra Andreas Mathisen.

Data preprosessering

Jeg gikk for «diabetes» skjemaet som jeg tidligere hadde kjennskap til fra «classification» oppgaven i Oblig 2. Ettersom jeg hadde håndtert det tidligere syntes jeg det var litt enklere å komme i gang med denne.

Alle andre obligene har jeg skrevet så lange og detaljerte, så jeg renger med du som lærer/sensor vil forstå at jeg kan data prosessering og klargjøring av datasettet nå.

Men kan forklare hva dette datasettet handler om, hva jeg har gjort, og hvordan oppgaven har løst seg.

Datasettet jeg jobber med tar for seg at man skal «predicte» om en person har diabetes eller ikke. Til sammenligning fra Oblig 2, fant jeg ut av at en SVM / SVC modell hadde en accuracy i prediction på ca: 0.844 ➔ 84.4% accuracy!

Kan vi slå den denne gangen?

Datasettet er bygd opp av kolonner / features:

i) Pregnancies
ii) Glucose
iii) BloodPressure
iv) SkinThickness
v) Insulin
vi) BMI
vii) DiabetesPedigreeFunction
viii) Age
ix) Outcome (target)

Hvor ix) = Target

Basert på infoen som ligger tilgjengelig her, vil vi kunne gjette på om en person har diabetes eller ikke. Accuracy scoren til modellen vil da ha en viss % den har gjettet riktig av tilfellene.

Jeg syntes denne oppgaven var litt vanskeligere enn vanlig. Jeg fikk ikke vært i klassen dessverre når stoffet ble gjennomgått, og har fått lite ut av videoer på nettet føler jeg. Lærer bedre i klasserommet, det har jeg alltid gjort. Så derfor har jeg sittet med andreas Mathisen som har hjulpet meg i gang og forklart meg nærmere. Så stor takk til han for hjelpen han gav meg, til å forstå nærmere, i alle fall.

Data analysering

Gjorde ikke stort annet enn å normalisere dataene, sørge for at det ikke var noen feil verdier i noen rader eller kolonner.

Resultater fra egen modell

Noe både jeg og et par andre stussa over var at tallene utgjorde forskjellige utslag på scoren vår. Feks ved bruk av MEAN normalization, ble accuracy hakket bedre enn med min-max. Jeg følte min-max var mer konstant og stabil, så valgte å forholde meg til denne.

Jeg fikk hjelp av en klassekamerat fra linje 99 i kodearket mitt. Jeg stod rett og slett fast og fikk hjelp til å komme meg videre.

Resultatene er som følger:

```
Final evaluation:
[0.6141543388366699, 0.6510416865348816]
```

Jeg vil si det er en dårlig score sånn egentlig, spesielt når jeg vet at SVC klarte 0.84. Men nå er jeg kanskje ikke dyktig nok til å skjønne hvordan den kan tunes til det bedre.

Har merket igjennom obligene at de er krevende for hodet, er mange små ting man skal ta høyde for, og veldig små marginer kan skape store utslag, men helt klart interessant og spennende.

Resultater fra multi layer perceptiton classifier

Jeg benyttet meg av MLPClassifier sitt bibliotek for å initialisere modellen. Jeg brukte samme måte som tidligere, lagde en liste modeller [] som jeg appendet inn modellene mine, som jeg så itererte over i en FOR løkke:

```
"""          Create Multi Layer Perception          """
max_iter = len(x_test)
modeller = []

# Default model
multi_layer_model_1 = MLPClassifier()
modeller.append(multi_layer_model_1)
```

Tuning

For å tune den brukte jeg også samme løsning som jeg har gått for tidligere!

Jeg brukte GridSearch for å finne beste parametere for meg, slik:

```
# Tuning
# == The tuning ==
activation = ["identity", "logistic", "tanh", "relu"]
solver = ["lbfgs", "sgd", "adam"]
batch_size = list(range(1, 5, 1))
learning_rate = ["constant", "invscaling", "adaptive"]

gridSearch_Multi_Layer = {
            "activation": activation,
            "solver": solver,
            "batch_size": batch_size,
            "learning_rate": learning_rate,

}
mlpc_tuner = RandomizedSearchCV(
            estimator=multi_layer_model_1,
            param_distributions=gridSearch_Multi_Layer,
            error_score="raise")
mlpc_tuner.fit(x_train, y_train)


# Console log the generated parameter settings option
print(f"\nparameter generated: {mlpc_tuner.best_params_}")
```

Output i konsollen fikk jeg da:

```
parameter generated: {'solver': 'sgd', 'learning_rate': 'constant', 'batch_size': 3, 'activation': 'identity'}
```

Deretter kunne jeg bruke denne infoen til å lage modellen automatisk:

```
# Creating the best model
multi_layer_tuner = MLPClassifier(
    activation=mlpc_tuner.best_params_['activation'],
    solver=mlpc_tuner.best_params_['solver'],
    batch_size=mlpc_tuner.best_params_['batch_size'],
    learning_rate=mlpc_tuner.best_params_['learning_rate'],
)
# Append model to modeller
modeller.append(multi_layer_tuner)
```

Resultat

Resultatet her ble som følger:

```
Model used: MLPClassifier()
Training score: 1.0
Test score: 1.0
Avvik mellom Train & Test set: 0.0
Vi er 0.0 fra 1.0
med et resultat på: 100.0%


Model used: MLPClassifier(activation='identity', batch_size=3, solver='sgd')
Training score: 1.0
Test score: 1.0
Avvik mellom Train & Test set: 0.0
Vi er 0.0 fra 1.0
med et resultat på: 100.0%
```

Med oversikt:

```
Model                                                           Percent
MLPClassifier()                                                 100.0
MLPClassifier(activation='identity', batch_size=3, solver='sgd')  100.0
```

Modellen til Multi Layer Perception virker særdeles bra, om det ikke er noe feil jeg har i koden min, men ikke noe skal tilsi det. Veldig fornøyd med siste resultatet.