Machine Learning - Journal 1

Opgaver fra lektion 1 & 2

Afleveret: 23/02/2022

Afleveret af: Gruppe 21

Deltagere i afleveringen

Studienummer	Navn	Studieretning
201900058	Rasmus Holm Lund	Elektronik
201906917	Lukas Kezic	Elektronik
201907898	Jakob Peter Aarestrup	Elektronik
201906146	Frederik Thomsen	Elektronik

Kontaktperson

Studienummer	Navn	E-mail	Studieretning
201900058	Rasmus Holm Lund	201900058@post.au.dk	Elektronik

Indholdsfortegnelse

L01 - intro

A: Parametre for lineær regression og R2 score

Der skal i denne opgave findes koefficienterne θ_0 og θ_1 ud fra den lineære regression. Hertil kan der bruges intercept() og coef() funktionen til at printe de to koefficienter, der henholdsvis har værdierne $\theta_0 = 4{,}85$ og $\theta_1 = 4{,}91$.

Ligeledes skal der også findes R^2 scoren på 0,734, hvor der bliver gjort brug af score() funktionen. R^2 scoren er en måde at evaluere sin regression baseret machine learning model. Den virker ved at måle mængden af variation i forudsigelsen ift. datasættet. Den maksimale værdi for R^2 er 1 og den minimale værdi er 0, hvor jo tættere værdien kommer på 1 desto bedre. Dvs. at R^2 scoren er en funktion, der måler fitness/goodness frem for loss/cost, da den måler hvor godt modellen fungerer. Dokumentationen for dette kan ses her[1].

Koden og outputtet nævnt i denne opgave kan ses i Kode snippet 1.

```
print(model.score(X,y))
print(model.intercept_[0])
print(model.coef_[0][0])
>0.734441435543703
 4.853052800266435
 4.911544589158485e-05
```

Kode snippet 1: Kode og output for opgave L01 – 1.A

B: Brug af k-nearest Neighbors

Nu skal den lineære regressions model ændres til en k-nearest neighbor model, hvor k=3. Med den nye model skal ”fit” og ”predict” køres igen. Koden og output kan ses i Kode snippet 2.

Det kan ses at med den nye model fås et estimat på 5,77, hvor med den lineære regressions model fås 5,96. Ligeledes kan det også ses at scoren er lidt anderledes end med den tidligere model. De bruger dog den samme score-metode hvor R^2 findes. Da de bruger samme scoringsmetoden, kan effektiviteten af de to modeller naturligvis også sammehttps://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.scorenlignes. Dokumentationen, som kan ses på Figur 1 kan også findes her[2].

[1] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.score
[2] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor.score

score(*X*, *y*, *sample_weight=None*) [source]

Return the coefficient of determination of the prediction.

The coefficient of determination R^2 is defined as $(1 - \frac{u}{v})$, where u is the residual sum of squares `((y_true - y_pred)** 2).sum()` and v is the total sum of squares `((y_true - y_true.mean()) ** 2).sum()`. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of `y`, disregarding the input features, would get a R^2 score of 0.0.

Figur 1: knn.score dokumentation fra scikit-learn.org

```
import sklearn.neighbors

# Valg af k-nearest neighbor model
knn = sklearn.neighbors.KNeighborsRegressor(3)

# Køre fit igen
knn.fit(X, y)

# Køre predict for Cyprus igen
X_new = [[22587]]  # Cyprus' GDP per capita
y_pred = knn.predict(X_new)
print(y_pred
print(knn.score(X,y))
>X.shape= (29, 1)
 y.shape= (29, 1)
 [[5.76666667]]
 0.8525732853499179
```

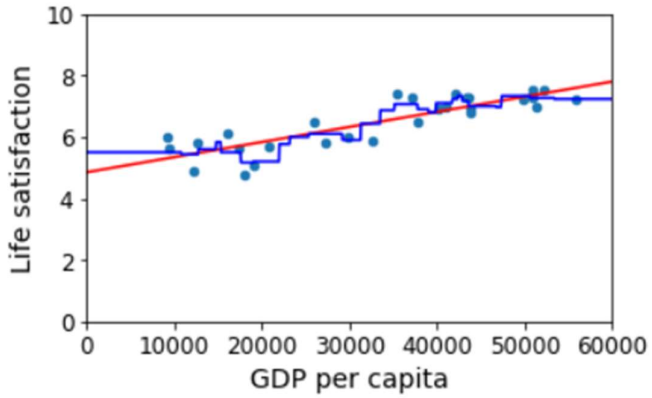
Kode snippet 2: Kode og output for opgave L01 – 1.B

C: Fintuning af parametre for knn og sammenligning

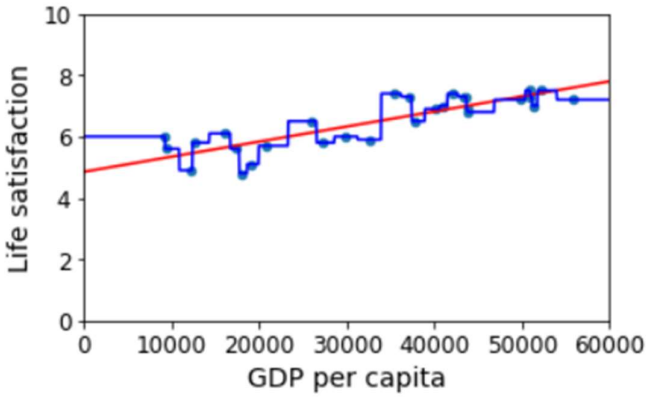
Hvis vi plotter vores prediction for knn modellen med andre k-værdier kan vi sammenligne præcisionen i plotsene. Som udgangspunkt virker det godt med en score på 1 når k sættes til 1. Grunden til at k=1 får en så god score er at den bare går til næste tætteste nabo for hvert punkt og derved får en præcision på 100%. Problemet her er at der ikke rammes medianen og man kan derfor komme langt ud ved yderværdier. Ligeledes hvis k får en for høj værdi får man bare en lige streg. Derfor er knn værdien på 3 et bedre bud. Koden og outputtet kan ses i Kode snippet 3. Ligeledes er der lavet 2 plots, der sammenligner knn-modellen med den lineære regressions model med to forskellige k-værdier.

```
print("knn score med k-neighbors = 1 ->",knn2.score(X,y))
print("knn score med k-neighbors = 3 ->",knn.score(X,y))
>knn score med k-neighbors = 1 -> 1.0
 knn score med k-neighbors = 3 -> 0.8525732853499179
```

Kode snippet 3: Kode og output for opgave L01 - 1.C



Figur 2: Plot med knn=3 og lineær regression



Figur 3: Plot med knn=1 og lineær regression

D: Afprøvelse af et Neuralt netværk

Ved brug af fit-predict interface kan vi bruge en ny model i vores allerede eksisterende kode. Her vil der blive brugt Scikit-learns for at kunne bruge MLPRegressor. Dog er det dataset vi bruger til at træne vores MLP ikke godt skaleret. Derfor skal der ændres på nogle parametre for at få et ordentligt output. Uden brug af preprocessering vil skalering af vores input data, gøre outputtet ubrugeligt.

MLP-regression modellen trænes og der predictes en værdi for Cyprus, samt en score for træningssættet. Dette kan ses i Kode snippet 4, hvor det også kan konkluderes, at MLP-regressionsscoren godt kan sammenlignes med knn-scoren, men også med den lineære.

Dette er fordi scoren er baseret på R^2 ligesom de to øvrige scorings-metoder. Hvilket også står klart hvis man kigger i dokumentationen for MLPRegressor[3].

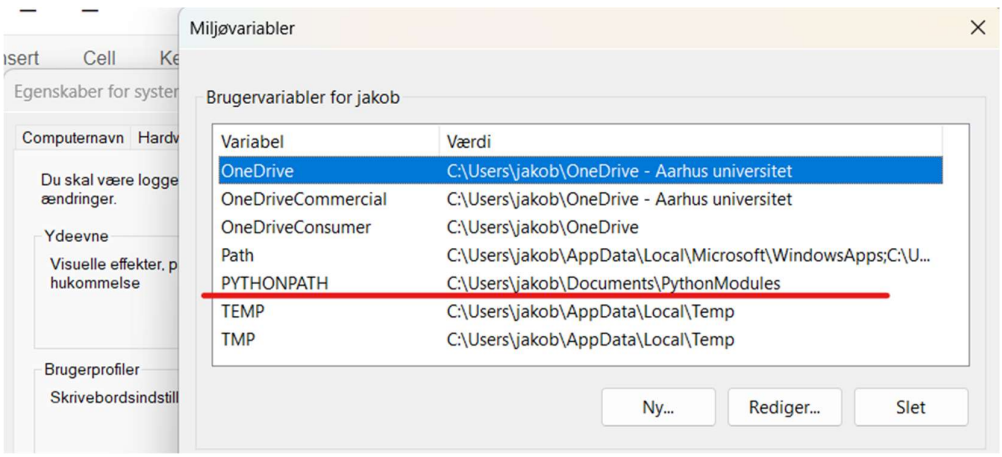
```
print("\nmlp score = ", mlp.score(X,y))
print("mlp Cyprus = ", mlp.predict(X_new))

print("\nKNN score = ", knn.score(X,y))
print("KNN Cyprus = ", knn.predict(X_new))

print("\nlinear score = ", model.score(X,y))
print("linear Cyprus = ", model.predict(X_new))
>mlp score =  -6.235423490637903
 mlp Cyprus =  [3.60467776]

 KNN score =  0.8525732853499179
 KNN Cyprus =  [[5.76666667]]

 linear score =  0.734441435543703
 linear Cyprus =  [[5.96242338]]
```

Kode snippet 4: Kode og output for opgave L01 1.D

L01 – Modules and classes

A: Indlæs og test libitmal modulet

I denne opgave skal libitmal modules afprøves. Først og fremmest skal der sættes en python path inde i miljøvariabler i windows. Dette kan ses på Figur 4.



Figur 4: Miljøvariabler indstillinger, hvor pythonpath tiljøjes

Herefter kan pathen tilføjes i koden og det kan ses på Kode snippet 5 at fungere efter hensigten.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor.score

```
import sys,os
sys.path.append('C:\\Users\\jakob\\Documents\\PythonModules')



from libitmal import utils as itmalutils
itmalutils.TestAll()
>TestPrintMatrix...(no regression testing)
 X=[[   1.    2.]
    [   3. -100.]
    [   1.   -1.]]
 X=[[ 1.  2.]
    ...
    [ 1. -1.]]
 X=[[   1.
        2.    ]
    [   3.0001
     -100.    ]
    [   1.
       -1.    ]]
 X=[[   1.    2.]
    [   3. -100.]
    [   1.   -1.]]
 OK
 TEST: OK
 ALL OK
```

Kode snippet 5: Kode og output for opgave L01 - 2.A

B: Oprettelse af eget modul og test af eget modul

I denne opgave er der lavet et modul med en simpel funktion skrevet ind i. Dette modul er døbt mymodule.py og har fået en simpel funktion ind i som kan ses på Kode snippet 6.

```
def greeting(name):
  print("Hello, " + name)
```

Kode snippet 6: mymodule.py kode

Modulet skal placeres i mappen hvor python pathen blev sat i den tidligere opgave. Dvs. den i dette tilfælde ligger i `C:\\Users\\jakob\\Documents\\PythonModules`.

Modulet kan herefter importeres til Jupyter og køres, som kan ses på Kode snippet 7.

```
import mymodule


mymodule.greeting("Jakob")
>Hello, Jakob
```

Kode snippet 7: mymodule køres i Jupyter

C: Hvordan reloader man et modul?

Når der er lavet ændringer i et modul, skal det først reloades. Den bedste måde at gøre det på er at bruge ImportLib eller AutoReload. Der er her brugt Autoreload som kan ses på Kode snippet 8.

```
%reload_ext autoreload
%autoreload 2
```

Kode snippet 8: Brug af autoreload

E: Udvid din klasse med nogle public funktion, private funktioner og variable members

Vi skal have udvidet vores klasse med en offentlig og en privat funktion.

```
class MyClass:
    myvar = "blah" # public class attribute
    __myvar = "blah" # private class attribute
    def myfun(self):
        print("This is a message inside the class.")
#public
    def public(self,name,age):
        self.name=name #public instance attribute
        self.age=age #public instance attribute
#private:
    def __private(self,name,age):
        self.__name=name #private instance attribute
        self.__age=age #private instance attribute
```

Kode snippet 9: MyClass kode

I Kode snippet 9 har vi udvidet klassen med offentlige og private, attributter, funktioner og metoder. Alt man skriver i en Python class er offentlig, medmindre man tilføjer __ før funktionen eller variablen. I modsætning til C++ definerer man ikke public og private, man definerer kun private via 2 lowercase bindestreg.

Ligger man mærke til funktionerne i koden har de alle self i sig. Dette skyldes self, fungere som vores @ i C++. Med andre ord så fungerer self som en adresse, så koden ved hvad den peger på. Så hvis man ikke tilføjer self, og prøver at køre en funktion som myfun(), så vil der komme en fejl om at denne funktion kræver 0 argumenter, men blev givet 1.

F: Udvid den eksisterende klasse med en konstruktor

Nu skal vi så have udvidet vores klasse med en konstruktor. Dette er egentlig meget simpelt, vi laver en helt ny klasse for at simplificere forklaringen, som kan ses på Kode snippet 10.

```
class MyConstruct:
    def __init__(self):
        print("This is a constructor")
        #body of the constructor
```

Kode snippet 10: MyConstruct klasse

En konstruktor i Python er defineret som (__init__), her er det vigtigt at man har de to nedsatte bindestreg på hver side. Det konstruktor gør er at den kører automatisk hver gang klassen kaldes. Ligeledes findes der en destruktor, som fjerner data fra hukommelsen. Denne er dog ikke nødvendig i Python, eftersom Python har en garbage collector, som håndterer hukommelsen automatisk.

G: Udvid den eksisterende klasse med en to-string funktion

I denne opgave vil der blive tilføjet en tostring() til vores klasse.

```
class MyToString:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def __str__(self):
        return f'Person name is {self.name} and age is {self.age}'
test2 = MyToString('Fred',23)
print(test2.__str__())
>Person name is Fred and age is 23
```

Kode snippet 11: MyToString klasse

I Kode snippet 11, har vi tilføjet en ToString funktion. Den er defineret som __str__, igen er de 2 nedsatte bindestrege på hver side vigtige. Det funktionen omdanner alt den får ind om til string. Dvs. at hvis man indsætter en integer på 20, så vil den lave det om til en string der siger 20.

L02 – Cost function

A: Givet følgende $x^{(i)}$'s, konstruere og print X matricen i python

I denne opgave har vi fået givet fire vektorer som står på række-vektor form. Disse skal laves om til matrix-form.

Først bruges array-funktionen fra biblioteket numpy til at lave vektorerne om til en matrix. Derefter omskrives matrixen til kolonne-form. Dette gøres med transpose-funktionen. Koden samt resultatet er vist på Kode snippet 12.

```
import numpy as np
X = np.array([[1,2,3],[4,2,1],[3,8,5],[-9,-1,0]])
X_transpose = X.transpose()
print(X_transpose)
>[[ 1  4  3 -9]
 [ 2  2  8 -1]
 [ 3  1  5  0]]
```

Kode snippet 12: Omskrivning til matrix-form fra række-vektor form

B: Implementer $\mathcal{L}_1$ og $\mathcal{L}_2$ nominelle for vektorer i python

I denne opgave skal formlerne $\mathcal{L}_1$ og $\mathcal{L}_2$ (vist nedenfor på Ligning 1 og Ligning 2) implementeres i Python. Disse formler skal herefter bruges på to data-sæt, hvorefter der til sidst skal tjekkes om resultatet er korrekt. Formlerne skal implementeres uden brug af indbyggede Python funktioner.

$$\mathcal{L}_1: \left|\left|x\right|\right|_2 = \left(\sum_{i=1}^{n} |x_i|^2\right)^{0.5}$$

Ligning 1

$$\mathcal{L}_2: \left|\left|x\right|\right|_1 = \sum_{i=1} |x_i|$$

Ligning 2

Nedenfor på Kode snippet 13 er formlerne L1 og L2 blevet implementeret i Python som hver deres funktion. Funktionerne tager et datasæt som parameter og udfører beregningerne, hvorefter datasættet returneres.

I funktionen L1 lægges den absolutte værdi af alle elementer i datasættet sammen. Dette gøres ved at kvadrere hvert element, hvorefter der tages kvadratroden af elementet. Dette kaldes den absolutte sum.

I funktionen L2 lægges alle kvadrerede elementer i datasættet sammen. Herefter tages der kvadratroden af det dette. Dette kaldes den eksplicitte sum.

```
def L1(a):
    i = 0
    k = 0
    while (i<len(a)):
        k = k + ((a[i]**2)**0.5)
        i+=1
    return(k)

def L2(a):
    i = 0
    k = 0
    while (i<len(a)):
        k = k + (a[i]**2)
        i+=1
    k = k**0.5
    return(k)
```

Kode snippet 13: Formlerne L1 og L2 implementeret i Python

Funktionerne testes med brug af datasættene tx og ty. Her kaldes funktionerne L1 og L2, hvor parameteren (tx-ty) indsættes. Koden samt resultatet fra testen er vist på Kode snippet 14.

```
import math

tx=np.array([1, 2, 3, -1])
ty=np.array([3,-1, 4,  1])

expected_d1=8.0
expected_d2=4.242640687119285

d1=L1(tx-ty)
d2=L2(tx-ty)

print(d1)
print(d2)

print(f"tx-ty={tx-ty}, d1-expected_d1={d1-expected_d1}, d2-expected_d2={d2-
expected_d2}")

eps=1E-9 # remember to import math for fabs
assert math.fabs(d1-expected_d1)<eps, "L1 dist seems to be wrong"
assert math.fabs(d2-expected_d2)<eps, "L2 dist seems to be wrong"

print("OK(part-1)")
>8.0
 4.242640687119285
 tx-ty=[-2  3 -1 -2], d1-expected_d1=0.0, d2-expected_d2=0.0
 OK(part-1)
```

Kode snippet 14: Test af L1 og L2 samt resultat

Som man kan se, så returnener funktionerne L1 og L2 de forventede resultater, som er vist på Kode snippet 14. Derudover kan man se, at funktionerne assert ikke udprinter error message og det er fordi, at forskellen mellem den forventede værdi og den reelle værdi er 0. Dermed kan det konkluderes, at det rigtige resultat er opnået.

Udover at lave L1 og L2 i Python, skal der også laves en funktion kaldet L2dot, der tager den eksplicitte sum ligesom i L2. Måden hvorpå den fungerer er, at den tager dot-produktet af det transponerede datasæt. Herefter tages kvadratroden af det hele. Her bruges dot funktionen fra bibliotek math. Koden samt resultatet er vist nedenfor på Kode snippet 15.

```
def L2Dot(a):
    k = (np.dot(a.T,a)**0.5)
    return(k)
d2dot=L2Dot(tx-ty)
print(d2dot)

print("d2dot-expected_d2=",d2dot-expected_d2)
assert math.fabs(d2dot-expected_d2)<eps, "L2Ddot dist seem to be wrong"
print("OK(part-2)")
>4.242640687119285
 d2dot-expected_d2= 0.0
 OK(part-2)
```

Kode snippet 15: Test af L2Dot funktion samt resultat

Som man kan se, så returnerer L2Dot funktionen det rigtige resultat, som er vist på Kode snippet 15. Derudover kan man se, at funktionerne assert ikke udprinter error message og det er fordi, at forskellen mellem den forventede værdi og den reelle værdi er 0. Dermed kan det konkluderes, at det rigtige resultat er opnået.

C: Konstruere RMSE funktionen

I denne opgave skal RMSE (Root Mean Squared Error) formlen implementeres i Python. RMSE er en værdi for hvor meget dataen afviger fra en RMS-model af dataen. Derudover skal MSE (Mean Squared Error) formlen også implementeres i Python, som er det samme som RMSE, bare uden at der bliver taget kvadratroden af det. Formlen for RMSE og MSE er vist på Ligning 3 og Ligning 4.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Z}_i - Z_i\right)^2 = \frac{1}{n}SS$$

Ligning 3

$$RMSE = \sqrt{MSE}$$

Ligning 4

Nedenfor i Kode snippet 16 kan det ses hvordan RMSE formlen er blevet implementeret i Python. Som man kan se, så beregnes først MSE. Her gøres der brug af L2 funktionen fra opgaven tidligere. Derudover divideres der med længden af arrayet vha. len funktionen. Til sidst kvadreres det hele.

Når funktionen RMSE kaldes, så kaldes funktionen h, der tjekker om det array, der er modtaget, er todimensionelt. Hvis arrayet ikke er todimensionelt, så kommer der en error message. Derudover tjekkes der om X har nul data i 0/1 aksen. Hvis dette er tilfældet,

kommer der en error message. Koden for funktionen h samt RMSE kan ses nedenfor i Kode snippet 16.

```
def RMSE(n,y):
    MSE = (1/len(y))*(L2(n-y))**2
    RMSE = MSE**0.5
    return RMSE

# Dummy h function:
def h(X):
    if X.ndim!=2:
        raise ValueError("excpeted X to be of ndim=2, got ndim=",X.ndim)
    if X.shape[0]==0 or X.shape[1]==0:
        raise ValueError("X got zero data along the 0/1 axis, cannot
continue")
    return X[:,0]

# Calls your RMSE() function:
r=RMSE(h(X),y)
```

Kode snippet 16: Funktion for RMSE formel samt error handler funktion h

Herefter laves der en test kode, der tjekker om resultatet er som det forventede. Denne kode samt resultatet er vist på Kode snippet 17.

```
# TEST vector:
eps=1E-9
expected=6.57647321898295
print(f"RMSE={r}, diff={r-expected}")
assert math.fabs(r-expected)<eps, "your RMSE dist seems to be wrong"

print("OK")
>RMSE=6.576473218982953, diff=2.6645352591003757e-15
OK
```

Kode snippet 17: Test kode samt resultat

Som man kan se, så opnås det rigtige resultat med en meget lille difference.

D: Konstruere også MAE funktionen og evaluere den

Her skal der kigges på om MAE funktionen duer som forventet ift. at få værdien 3.75.

```
def MAE(x,y):
    n = len(x)
    mae = 0
    mae = (1/n)*(L1(y-x))
    return mae
# Calls your MAE function:
r=MAE(h(X), y)

# TEST vector:
expected=3.75
print(f"MAE={r}, diff={r-expected}")
assert math.fabs(r-expected)<eps, "MAE dist seems to be wrong"
>MAE=3.75
 diff=0.0
```

Kode snippet 18: MAE funktion

Det kan ses at MAE duer som den skal, da den værdi som er forventet er den samme som vi får. Dvs. at funktionen duer som den skal. Dette er også fordi der er ingen difference mellem forventet resultat og resultat.

E: Robust kode

Efter MAE og RMSE er blevet testet skal der laves error handling. For at få en mere robust kode. Denne kode skal selvfølgelig også testet hvilket det bliver gjort i koden nedenfor.

```
def MAE(x,y):
    if x.ndim!=1:
        raise ValueError("expected X to be of ndim=1, got ndim=",x.ndim)
    if X.shape[0]==0 or X.shape[1]==0:
        raise ValueError("X got zero data along the 0/1 axis, cannot
continue")
    if y.ndim!=1:
        raise ValueError("expected y to be of ndim=1, got ndim=",y.ndim)
    n = len(x)
    mae = 0
    mae = (1/n)*(L1(y-x))
    return mae
r=MAE(h(X),y)

def RMSE(x,y):
    if x.ndim!=1:
        raise ValueError("expected X to be of ndim=1, got ndim=",x.ndim)
    if X.shape[0]==0 or X.shape[1]==0:
        raise ValueError("X got zero data along the 0/1 axis, cannot
continue")
    if y.ndim!=1:
        raise ValueError("expected y to be of ndim=1, got ndim=",y.ndim)
    n = len(x)
    MSE = (L2(x-y)**2)
    RMSE = ((1/n)*MSE)**0.5
    return(RMSE)

r = RMSE(h(X),y)
```

Kode snippet 19: MAE og RMSE definition

Det kan ses på Figur 5, at error handlingen fungerer fint idet der bliver printet noget. Vi kan forcere en fejl ved at sætte stort X. (r = RMSE(X,y)) og se at ValueError printer det den skal.

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7432/202144818.py in <module>
     27     return(RMSE)
     28
---> 29 r = RMSE(X,y)

~\AppData\Local\Temp/ipykernel_7432/202144818.py in RMSE(x, y)
     17 def RMSE(x,y):
     18     if x.ndim!=1:
---> 19         raise ValueError("exspected X to be of ndim=1, got ndim=",x.ndim)
     20     if X.shape[0]==0 or X.shape[1]==0:
     21         raise ValueError("X got zero data along the 0/1 axis, cannot continue")

ValueError: ('exspected X to be of ndim=1, got ndim=', 2) .
```

Figur 5: Error handling

F: Konklusion

Ud fra denne øvelse kan vi konkludere at vi har fået mere kendskab til matematikken bag vores modeller og hvordan vi selv kan bruge den. Specifikt har vi fået et bedre kendskab til hvordan man finder norm af en vektor og bruger denne norm til at udregne andre ting, som RMSE og MAE. Vi har lært præcist hvordan RMSE og MAE er opbygget og fungerer.

L02 – Dummy classifier

A: Indlæs og vis MNIST dataen

Der skal i denne opgave loades X- og y værdier fra et MNIST data-sæt. Herefter skal en enkelt værdi fra dette data-sæt loades, hvorefter der skal der laves et NMIST subimage, som er et plot af den loadede værdi.

Først laves funktionen MNIST_GetDataSet(), m står for at loade X- og y-værdierne fra MNIST data-sættet. For at gøre dette bruges fetch_openml, som tager datasættet og gemmer dets værdier. Herefter sættes X-værdierne som ”data” og y-værdierne som ”target”, hvorefter de returneres. Koden er vist på Kode snippet 20.

```
from sklearn.datasets import fetch_openml
def MNIST_GetDataSet():
    # Load data from https://www.openml.org/d/554
    mnist = fetch_openml('mnist_784',cache=True, as_frame=False)
    X, y = mnist["data"], mnist["target"]
    return X, y
```

Kode snippet 20: Funktionen MNIST_GetDataSet(), som loader MNIST-dataen

Funktionen MNIST_PlotDigit(data) står for at plotte en enkelt værdi af de data, der er loadet fra MNIST data-sættet. Først importeres pyplot fra matplotlib biblioteket, som bruges til at plotte den loadede værdi. Herefter sættes opløsningen af plottet til 28p, med rescale funktionen. Til sidst plottes værdien med imshow funktionen fra pyplot. Koden er vist på Kode snippet 21.

```
%matplotlib inline
def MNIST_PlotDigit(data):
    import matplotlib
    import matplotlib.pyplot as plt
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = matplotlib.cm.binary, interpolation="nearest")
    plt.axis("off")
```

Kode snippet 21: Funktionen MNIST_PlotDigit(Data), som plotter den loadede data

For at teste funktionerne er en test-kode blevet lavet. Denne er vist på Kode snippet 22. Først gemmes de returnerede værdier fra GetDataSet i værdierne X,y. Herefter printes den første værdi i datasættet X ved at den sættes ind som parameter i funktionen MNIST_PlotDigit.

```
X,y = MNIST_GetDataSet()
MNIST_PlotDigit(X[0])
```

Kode snippet 22: Test kode

Nedenfor på Figur 6 kan man se resultatet af test-koden. Man kan se, at, at den første værdi i datasættet X bliver plottet, og at den plottede værdi er et 5-tal.

Figur 6: Resultat af testkode

B: Tilføj en Stochastic Gradient Decent (SGD) classifier

I denne opgave skal der laves et træningssæt af MNIST-dataen, hvorefter en Stochastic Gradient Decent (SGD) classifier skal bruges til at fit og predict dataen.

Først opdeles X- og y-værdierne fra MNIST datasættet i X- og y train og test sæt. Train-dataen er den data der vil blive 'trænet', og test-dataen er den data, der vil blive sammenlignet med til sidst.

Efter dette laves to variabler y_train_5 og y_test_5, der indeholder alt klasse-5 dataen fra henholdsvis y-train og y-test dataen.

Dette er vist på Kode snippet 23.

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], [60000:]
y_train_5 = (y_train == '5')
y_test_5  = (y_test == '5')
```

Kode snippet 23: Opdeling af MNIST-data i train og test data samt definering af variabler med kun klasse-5 data

Herefter vil SGD classifieren nu blive brugt. Først udføres en fit på X_train dataen og y_train_5 dataen. Dette vil sige, at X_train dataen vil blive fittet, så den rammer så tæt på y_train_5 variablen som muligt. Efter dataen er blevet fittet i X_train, kan der laves en predict som vil sammenligne X_train med det første element i X_test. Koden kan ses på Kode snippet 24.

```
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
predictedResult = sgd_clf.predict([X_test[0]])
```

Kode snippet 24: Initialisering af SGDClassifier samt brug af fit og predict funktionerne til dataen

Ved at printe værdien af predict'en fås resultatet "True". Dette betyder at det første element i X_test stemmer overens med X_train, hvilket giver mening, da X_train er blevet fittet til kun at indeholde klasse 5 data og det første element i X_test er klasse 5 data. Testkoden og resultatet kan ses på Kode snippet 25.

```
print(predictedResult)
>[ True]
```

Kode snippet 25: Print af predict funktion samt output

C: Implementere en dummy binær klassificere

I denne opgave vil der blive lavet en Scikit-learn kompatible estimator, som er implementeret vha. en python klasse.

Først defineres klassen DummyClassifier(BaseEstimator). Klassen har classifieren BaseEstimator som parameter, som definerer den classifier, der vil blive brugt. Klassen indeholder derudover en fit() og predict() funktion, som vil blive brugt på test-dataen. Klassen er vist på Kode snippet 26.

```
from sklearn.base import BaseEstimator
from sklearn.metrics import accuracy_score

class DummyClassifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

Kode snippet 26: Dummyclassifier klasse

DummyClassifier klassen vil nu blive brugt. Derfor oprettes et objekt. Herefter kaldes fit funktionen, hvor X_train dataen fittes til y_train_5, som kun indeholder klasse 5 data. Efter

dette laves der en predict på X_test dataen, dvs. at X_train dataen sammenlignes med X_test dataen. Til sidst bruges funktionen accuracy_score fra biblioteket sklearn-metrics, som outputter en procent-værdi på hvor meget y_test_5 værdierne stemmer overens med de forventede værdier fra predict funtkionen. Koden samt outputtet er vist på Kode snippet 27.

```
DummyClassifierObj = DummyClassifier()

DummyClassifierObj.fit(X_train,y_train_5)

predictedResult = DummyClassifierObj.predict(X_test)

accuracy = accuracy_score(y_test_5,predictedResult) #91.08% nøjagtighed

print(accuracy)
>0.9108
```

Kode snippet 27: Dummyclassfier accuracy

Som man kan se på Kode snippet 27, så giver funktionen accuracy_score en score på 91,08%. Dvs. at 91,08% af den forventede data fra predict funktionen stemmer overens med y_test_5 dataen.

D: Konklusion

Det kan konkluderes, at det har været muligt at loade MNIST data, samt plotte et af tallene med funktionen pyplot. Derudover det været muligt at opdele dataen i train- samt test data, hvorefter classifieren Stochastic Gradient Decent (SGD) er blevet brugt til at train dataen, og predict brugt til at sammenligne test-dataen med train-dataen. At dette virker, er bekræftet i opgave B, da det kan ses, at train-dataen kan trænes til at blive til klasse 5 data. I opgave C har det været muligt at integrere en classifier i en klasse indeholdende en fit og predict funktion. Derudover har det været muligt at beregne nøjagtigheden, dvs. hvor meget i procent det forventede resultat stemmer overens med det reelle. Her blev en nøjagtighed på 91,08% opnået.

L02 – Performance metrics

A: Implementer nøjagtighedsfunktionen og test den med MNIST data

I denne opgave skal der laves en funktion der udregner nøjagtigheden af data. Der vil i denne opgave blive målt på MNIST dataen. Det første vi gør, er at lave vores egen Confusion matrix, eftersom vi skal bruge nogle værdier til at udregne nøjagtigheden.

```
def MyConfusion_matrix(y_true,y_pred):
    TP = FP = TN = FN = 0
    for i in range(len(y_pred)):
        if y_true[i]==y_pred[i]==1:
            TP += 1
        elif y_true[i]==1 and y_true[i]!=y_pred[i]:
            FN += 1
        elif y_true[i]==y_pred[i]==0:
            TN += 1
        else:
            FP += 1
    return(TP, FP, TN, FN)
```

Kode snippet 28: MyConfusion matrix

Dette giver os vores True Positive, False positive, True negative og False negative. Som vi bruger igennem resten af denne opgave. Derefter kan vi så implementere vores nøjagtigheds funktion.

```
def MyAccuracy(y_true, y_pred):
    TP,FP,TN,FN = MyConfusion_matrix(y_true,y_pred)
    accuracy = (TP+TN)/len(y_true)
    return accuracy
```

Kode snippet 29: MyAccuracy

Nu har vi så en funktion, der udregner nøjagtigheden, så lad os prøve at teste den imod sklearn.metrics.accuracy_score(). Før vi kan teste dem, skal vi dog have noget data, så der genbruges vores MNIST-data loader fra tidligere øvelser, dette vises ikke igen. Vi starter med at definere en test funktion, som vi bruger til at teste to forskellige modeller, henholdsvis DummyClassifier og SGDClassifier. Disse 2 modeller er blevet lavet i tidligere opgaver og vil derfor ikke blive beskrevet yderligere.

```
def TestAccuracy(y_true, y_pred):
    a0=MyAccuracy(y_true, y_pred)
    a1=accuracy_score(y_true, y_pred)
    print(f"\nmy accuracy = {a0}")
    print(f"scikit-learn accuracy = {a1}")
    eps=1E-9
    if fabs(a0-a1)<eps:
        print(F"Accuracy correct, difference = {a0-a1}")
    else:
        print(F"Accuracy not correct, difference = {a0-a1}")
print("DummyClass =")
TestAccuracy(y_test_5, y_pred)
print("\nSGDClass =")
TestAccuracy(y_test_5, sgd_predict)
```

```
>DummyClass =

 my accuracy = 0.9108
 scikit-learn accuracy = 0.9108
 Accuracy correct, difference = 0.0

 SGDClass =

 my accuracy = 0.9492
 scikit-learn accuracy = 0.9492
 Accuracy correct, difference = 0.0
```

Kode snippet 30: Test af my accuracy

Vi får altså et resultat der ens med sklearn.metrics.accuracy_score(), dermed er vores funktion godkendt. Der er dog den chance at ens nævner i udregningen af nøjagtigheden kan blive 0. For at løse dette problem kan man tilføje en if-statement.

```
def MyAccuracy(y_true, y_pred):
    TP,FP,TN,FN = MyConfusion_matrix(y_true,y_pred)
    if len(y_true) == 0:
        return 0
    accuracy = (TP+TN)/len(y_true)
    return accuracy
```

Kode snippet 31: Tilføjelse af if-statement

Det if-statement gør, at den tjekker om længden af y_true er 0, hvis den er dette, sender den 0 retur. Dermed sikre vi os at funktionen ikke prøver at dividere med 0.

B: Implementer Precision, Recall og F_1-scorer og test det på MNIST-data for både SGB og Dummy klassificerer modellerne

I denne opgave skal man selv lave funktioner for Precision, Recall og F_1-score. I den tidligere opgave lavede vi en funktion der fandt vores TN, FN, FP, TP. Denne funktion genbruger vi i denne opgave. Vi starter med Precision og Recall funktionerne.

```
def MyPrecision(y_true, y_pred):
    TP,FP,TN,FN = MyConfusion_matrix(y_true,y_pred)
    if TP+FP == 0:
        return 0
    precision = TP/(TP+FP)
    return precision

from sklearn.metrics import recall_score
def MyRecall(y_true, y_pred):
    TP,FP,TN,FN = MyConfusion_matrix(y_true,y_pred)
    if TP+FN == 0:
        return 0
    recall = TP/(TP+FN)
    return recall
```

Kode snippet 32: MyPrecision og MyRecall definition

Vi har her vores Precision og Recall funktion. Det vigtige her er hvordan de udregnes, samt de to if statements, som gør at vi ikke dividere med 0.

Formel for precision:

$$p = \frac{TP}{TP + FP}$$

Ligning 5

Formel for Recall:

$$r = \frac{TP}{TP + FN} = \frac{TP}{N_P}$$

Ligning 6

Til sidst laver vi funktionen for F_1-score. Formlen F-score er: $F_\beta = (1 + \beta^2) \cdot \frac{p \cdot r}{\beta^2 \cdot p + r}$. Denne formel er mere indviklet, men vi ved at vi skal F_1-score, så vi kan gøre regnestykket mere overskueligt. Vores β bliver kaldt for harmonic og i en F_1-score er denne normalt $\beta = 1$. Så vores nye regnestykke bliver:

$$F_1 = \frac{2 \cdot p \cdot r}{p + r} = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

Ligning 7

Her er $p = precision$ og $r = Recall$. Nu hvor vi ved dette, kan vi lave funktionen.

```
p def MyF1Score(y_true, y_pred):
    p = MyPrecision(y_true, y_pred)
    r = MyRecall(y_true, y_pred)
    if (p == 0 and r == 0):
        return 0
    f1 = 2/((1/r)+(1/p))
    return f1
```

Kode snippet 33: MyF1Score definition

Læg mærke til at vi har implementeret et if-statement, som returnerer 0, hvis r eller p er 0. Dette gør så vi igen ikke dividere med 0. Til sidst sammenligner vi vores funktioner med sklearns funktioner. Vi bruger igen de to modeller SGDClassifier og DummyClassifier.

```
MyPrecision(y_test_5, y_pred)

precision_score(y_test_5, y_pred,zero_division=0)

MyRecall(y_test_5, y_pred)
recall_score(y_test_5, y_pred,zero_division=0)
```

Kode snippet 34: Sammenligning af de to funktioner

Vi kalder altså vores funktion, efterfulgt af sklearns relateret funktion, dette gentages for alle funktionerne og begge modeller. Resultatet af dette printes ud, så det kan sammenlignes. Selve printe delen af koden er ikke medtaget eftersom det er ren formaterings design.

```
>my DummyClassifier precision =  0
 scikit-learn DummyClassifier precision =  0.0

 my DummyClassifier recall =  0.0
 scikit-learn DummyClassifier recall =  0.0

 my DummyClassifier F1 score =  0
 scikit-learn DummyClassifier F1 score =  0.0

 my SGDClassifier precision =  0.6618887015177066
 scikit-learn SGDClassifier precision =  0.6618887015177066

 my SGDClassifier recall =  0.8800448430493274
 scikit-learn SGDClassifier recall =  0.8800448430493274

 my SGDClassifier F1 score =  0.7555341674687199
 scikit-learn SGDClassifier F1 score =  0.75553416746872
```

Kode snippet 35: Output fra sammenligning

Vi kan altså se at vores funktioner giver de samme resultater som sklearns funktioner. Dermed er vores funktioner godkendt.

C: Confusion matricen

Vi får generet dummy klassen og SGD klassen ved brug af sklearn.metrics.confusion_matrix() funktionen. Der generer en matrice. Dette bliver gjort for hhv. dummy og SGD. Matricerne er organiseret med TN, FP, FN og TP, i denne rækkefølge. Fra højre mod venstre og ned. Dvs. de 2 første er i den øverste række og de to næste i den nederst række i 2x2 matricen. Denne funktion kræver to parameter, y_true, og y_pred. Så vi tester funktionen ud fra vores 2 modeller SGDClassifier og DummyClassifier.

```
M_SGD = confusion_matrix(y_test_5,sgd_predict)
M_dummy = confusion_matrix(y_test_5,y_pred)
```

```
>[[9108    0]
 [ 892    0]]
 [[8707  401]
 [ 107  785]]
```

Kode snippet 36: Confusion matrix prediction

Vi får altså 2 matricer, men hvad sker der hvis vi bytter om på y_true og y_pred?

```
confusion_matrix(y_pred,y_test_5)
> array([[9108,  892], [   0,    0]], dtype=int64)
```

Kode snippet 37: Confusion matrix transformation

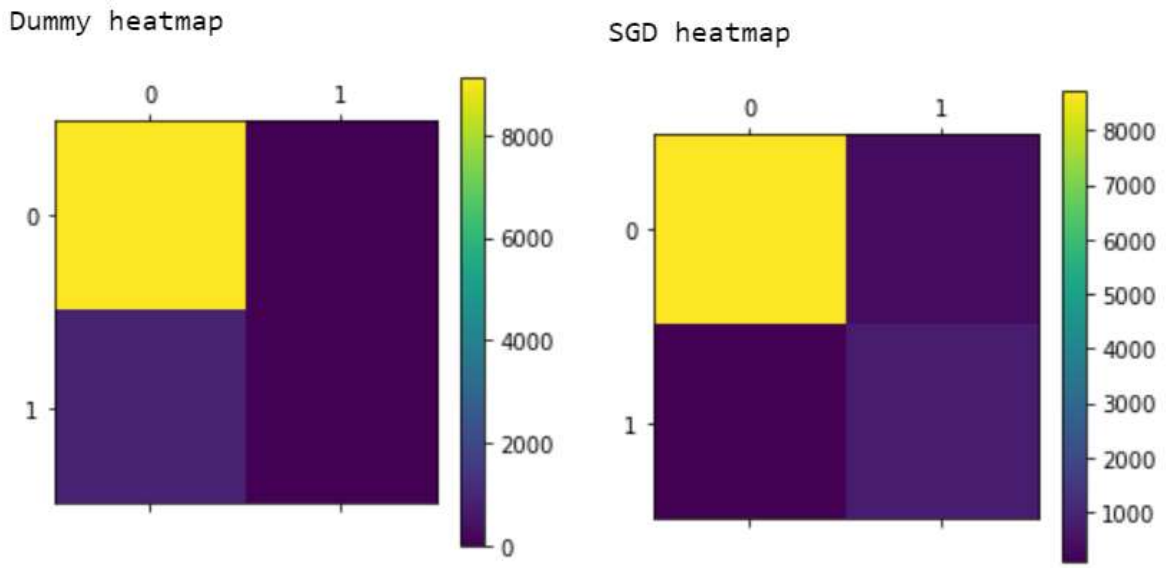
Vores matrice bliver altså transformeret. Det samme gælder så for vores TN, FP, FN og TP.

D: Confusion matrice varmekort

Der skal genereres et varmekort ud fra Confusion matricen. Dette bliver gjort vha. af denne kode.

```
import matplotlib.pyplot as plt
def plot_confusion_matrix(matrix):
    """If you prefer color and a colorbar"""
    fig = plt.figure(figsize=(4,4))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)

print("Dummy heatmap")
plot_confusion_matrix(M_dummy)

print("SGD heatmap")
plot_confusion_matrix(M_SGD)
```

Kode snippet 38: Kode for confusion matrice heatmaps



Figur 7: Dummy og SGD heatmap

Det kan ses at i Dummy klassen er data for TN gul dvs. over 8000, hvilket betyder at der er over 8000 tilfælde der ikke passer med det ønskede resultat. Dette er tilfældet for SQD også. Hvilket betyder at begge metoder har opnået primært ikke ønskede resultater. Den næste FP er nul. Så ingen falske positive data. Den næste FN er omkring de 1000 ifølge heatmappen. Hvilket tyder på at der har været en del FN. Dog ikke lige så meget som TN. Den sidste værdi er også mod nul. Hvilket vil sige at TP heller ikke er til stede. Forskel til SGD er at alle parametrene er til stede. Her er der flest TP ift. de andre værdier, hvor denne værdi er tættere på 1000. Dog er både FP og FN tættere på nul. TN er stadig over 8000 hvilket gør den gul. Dvs. at SGD-klasse er langt bedre i det den får flere TP end FN. De FN der kom, fandt SGD-klassen ud af var TP.

E: Konklusion

Ud fra denne opgave har vi lært hvordan en masse data kan blive klassificeret under fire begreber, alt efter om de er blevet vurderet positive, negative, falsk positive eller falsk negative. Samt hvordan man finder og regner på disse værdier. Generelt har vi fået en bedre forståelse for de funktioner vi bruger til at analysere vores modeller. Ved at bygge vores egne funktioner og sammenligne dem med de eksisterende, har vi formået at få en god forståelse for matematikken bagved.