ITMAL - Journal 1

Jeppe Stærk - 201271201

Forår 2019

Litteratur

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

[2] A. Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017. ISBN 1491962291, 9781491962299.

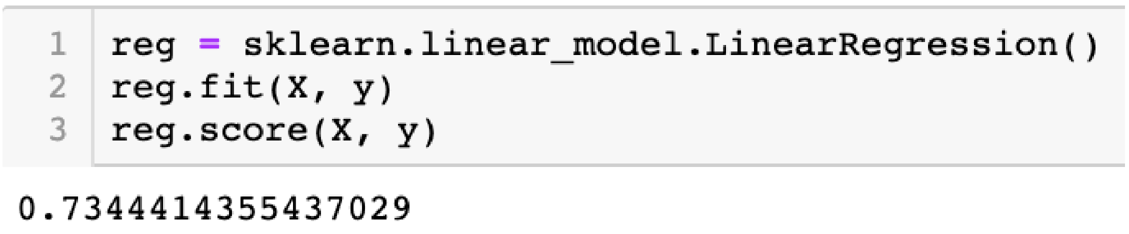
Indholdsfortegnelse

L01 intro.ipynb 1

1.1 Qa The θ parameters and the R^2 Score

1.1.1 How do you extract the θ_0 and θ_1 coefficients in his life-satisfaction figure form the linear regression model?

Det løse vha hhv $coef\_$ og $intercept\_$ functionerne fra $sklearn.linear\_model.LinearRegression$



```
reg = sklearn.linear_model.LinearRegression()
reg.fit(X, y)
reg.score(X, y)
```

```
0.7344414355437029
```

1.1.2 What are the minimum and maximum values for R^2?

1 er maximun for R^2 hvilket vil medføre at datasættet vil fitte perfekt. R^2 kan være 0 eller under, og dermed vise at datasættet fitter dårligt.

1.1.3 Is it best to have a low R^2 score (a measure of error/loss via a cost-function) or a high R^2 score (a measure of fitness/goodness)?

Vi vil altid ønske en R^2 værdi så tæt på 1 som muligt, da vi i såfald vil score bedst.

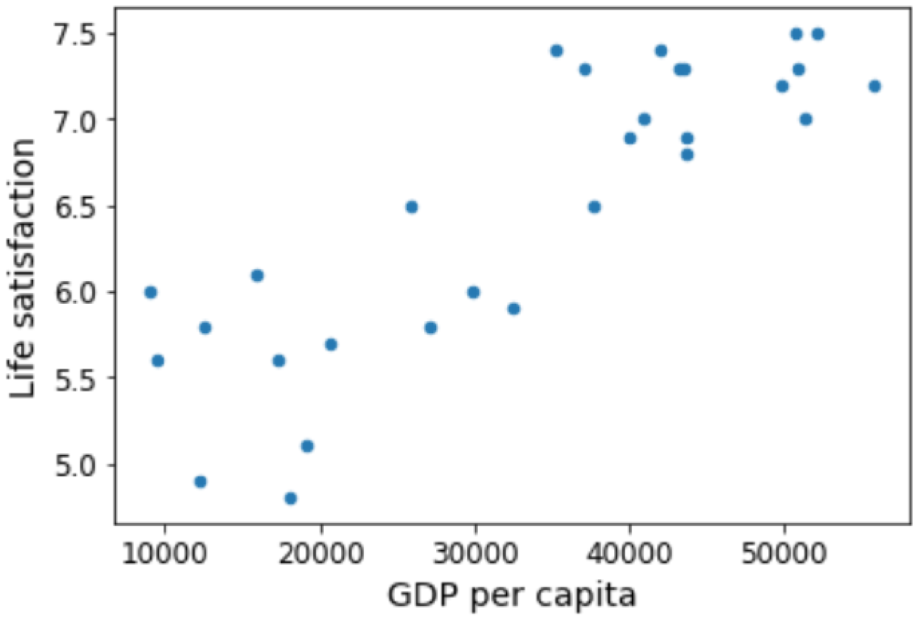
1.2 Qb Using k-Nearest Neighbors

1.2.1 What do the k-nearest neighbours estimate for Cyprus, compared to the linear regression (it should $yield = 5.77$)?

Her sætter vi $k\_neighbors = 3$

```
# TODO: add code here...
Xsample = np.c_[sample_data["GDP per capita"]]
ysample = np.c_[sample_data["Life satisfaction"]]
X_Cyprus = [[22587]]

reg = sklearn.linear_model.LinearRegression()
reg.fit(X, y)
y_pred_reg = reg.predict(X_Cyprus)
print('LinearRegression: ')
print(y_pred_reg)

from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=3)
neigh.fit(X, y)
y_pred_neigh = neigh.predict(X_Cyprus)
print('KNeighborsRegressor: ')
print(y_pred_neigh)
```

```
X.shape= (29, 1)
y.shape= (29, 1)
```



```
LinearRegression:
[[5.96242338]]
KNeighborsRegressor:
[[5.76666667]]
```

1.2.2 What *score-method* does the k-nearest model use, and is it comparable to the linear regression model?

Score fungen meget identisk med linear regression, her benytter man dog de 3 tæsttese naboer til udregningen modsat hele datasættet i linear regression.

1.2.3 Seek out the documentation in Scikit-learn, if the scoring methods are not equal, can they be compared to each other at all then?

Da det er samme datasæt, så vil man altid kunne samligne netop på score parameteret hvilken model der performer bedst.

1.3 Qc Tuning Parameter for k-Nearest Neighbors and A Sanity Check

1.3.1 Does $k\_neighbor = 1$ not look beautiful regarding the score (should yield $score = 1$)? ...or does it?

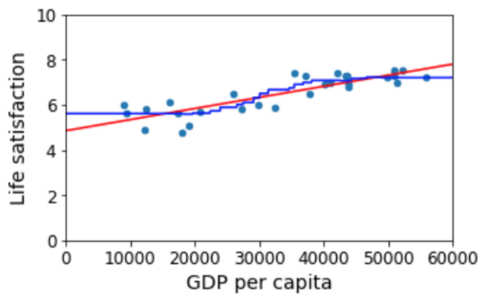
Da det er det samme data vi har brugt til både test og træning, så vil vi nå et urelistisk godt resultat.

1.3.2 Plot the two models in a 'Life Satisfaction-vs-GDP capita' 2D plot by creating an array in the range 0 to 60000 (USD) and then predict the corresponding y value. Reuse the plots stubs below, and explain why the k-nearest neighbour with $k\_neighbor = 1$ has such a good score.

Med $k\_neighbor = 1$ får man en “overfitted"estimator, som er tunet til det specifikke datasæt man har trænet den med. Det giver et meget pænt resultat hvis man validerer på samme datasæt som man har trænet med, men i praksis vil den classifier ikke nødvendigvis være god på andre datasæt.

```python
sample_data.plot(kind='scatter', x="GDP per capita", y='Life satisfaction', figsize=(5,3))
plt.axis([0, 60000, 0, 10])

# create an test matrix M, with the same dimensionality as X, and in the range [0;60000]
# and a step size of your choice
m=np.linspace(0, 60000, 1000)
M=np.empty([m.shape[0],1])
M[:,0]=m

# TODO from this test M data, predict the y values via the lin.reg. and k-nearest models
y_pred_lin = sklearn.linear_model.LinearRegression().fit(X, y).predict(M)
y_pred_kn  = KNeighborsRegressor(n_neighbors=10).fit(X, y).predict(M)

# TODO use plt.plot to plot x-y into the sample_data plot...
plt.plot(M, y_pred_lin, "r")
plt.plot(M, y_pred_kn, "b")
```

```
[<matplotlib.lines.Line2D at 0x10eb9d438>]
```



1.3.3 Does a $score = 1$ with $k\_neighbor = 1$ also mean that this would be the prefered estimator for the job?

Ligeledes her, har vi en udfording i det at vi tester og træner med samme datasæt, og derved opnår et urealistisk godt fit. Så at $score = 1$ med $k\_neighbor = 1$ betyder ikke at det vil være den foretrukne estimator til andre datasæt.

L02 math.ipynb 2

2.1 Qa Given the following $x^{(i)}$'s, construct and print the X matrix in python.

```python
# Qa

import numpy as np

y = np.array([1,2,3,4]) # NOTE:  you'll need this later

# TODO..create and print the full matrix
x_1 = np.array([1,2,3])
x_2 = np.array([4,2,1])
x_3 = np.array([3,8,5])
x_4 = np.array([-9,-1,0])
X = np.matrix([x_1.T,x_2.T,x_3.T,x_4.T])
print(X)
```

```
[[ 1  2  3]
 [ 4  2  1]
 [ 3  8  5]
 [-9 -1  0]]
```

2.2 Qb Implement the $L1$ and $L2$ norms for vectors in python

```python
# TODO: solve Qb...implement the L1, L2 and L2Dot functions...
from math import *
def L1(x):
    l1_sum = 0
    for n in x:
        l1_sum += abs(n)
    return l1_sum

def L2(y):
    l2_sum = 0
    for n in y:
        l2_sum += abs(n)**2
    return l2_sum**0.5

def L2Dot(x):
    return np.sqrt(x.dot(x.T))

# TEST vectors: here I test your implementation...calling your L1() and L2() functions
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("tx-ty=",tx-ty,", d1-expected_d1=",d1-expected_d1,", d2-expected_d1=",d2-expected_d2)

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

# comment-in once your L2Dot fun is ready...
d2dot=L2Dot(tx-ty)
print(d2dot)
print("d2dot-expected_d2=",d2dot-expected_d2)
assert fabs(d2dot-expected_d2)<eps, "L2Ddot dist seem to be wrong"
```

```
tx-ty= [-2  3 -1 -2] , d1-expected_d1= 0.0 , d2-expected_d1= 0.0
4.242640687119285
d2dot-expected_d2= 0.0
```

2.3 Qc Construct the Root Mean Square Error (RMSE) function (Equation 2-1 [HOLM]).

```python
# TODO: solve Qc...implement your RMSE function here
from sklearn.metrics import mean_squared_error
def RMSE(X,y):
    mse = mean_squared_error(X, y)
    return np.sqrt(mse)

# TEST vector:
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]

eps=1E-9
r=RMSE(h(X),y)
expected=6.57647321898295
print("RMSE=",r,", diff=",r-expected)
assert r-expected<eps, "your RMSE dist seems to be wrong"
```

```
RMSE= 6.576473218982953 , diff= 2.6645352591003757e-15
```

2.4 Qd Similar construct the Mean Absolute Error (MAE) function (Equation 2-2 [HOLM]) and evaluate it.

```python
# TODO: solve Qd
from sklearn.metrics import mean_absolute_error
def MAE(X,y):
    return mean_absolute_error(X, y)

# TEST vector:
r=MAE(h(X), y)
expected=3.75
print("MAE=",r,", diff=",r-expected)
assert r-expected<eps, "MAE dist seems to be wrong"
```

```
MAE= 3.75 , diff= 0.0
```

2.5 Qe Robust Code

Den sidste linje kode i Qb, Qc og Qd er en `assert` som laver et sanity-check af resultatet, og smider en fejl hvis resultatet ser forkert ud.

L02 statistics.ipynb 3

3.1 Qa Create a mean and variance function for some input data

```python
# TODO: Qa...
import numpy as np

def MeanAndVariance(x, biasedvar=True):
    assert(x.ndim==1),"cannot do a MeanAndVariance on a non 1D object, ndim="+str(x.ndim)
    (n,)=x.shape
    assert(n>1)
    m=0.0
    v=0.0

    for k in range(0,n):
        m += x[k]
    m /= n

    for k in range(0,n):
        t = x[k]-m
        v += t*t

    f = n-1
    if biasedvar:
        f = n

    v = v/f
    assert(v>=0)

    return m,v


# TEST vectors: mean and variance calc
y = np.array([1,2,3,4])
m, v = MeanAndVariance(y)

expected_m = 2.5
expected_v_biased = 1.25 # factor 1/n
expected_v_unbiased = 1.6666666666666667 # factor 1/(n-1)

print("m=",m,", diff=", m-expected_m)
print("v=",v,", diff=", v-expected_v_biased)
print(v,np.mean(y), np.var(y)) # np.var is biased(n)
```

```
m= 2.5 , diff= 0.0
v= 1.25 , diff= 0.0
1.25 2.5 1.25
```

3.2 Qb Create a function that generates the auto-covariance matrix

```python
# TODO: Qb...

def covariance(x, y):
    xbar, ybar = x.mean(), y.mean()
    return np.sum((x - xbar)*(y - ybar))/(len(x)-1)

def covariance_matrix(x):
    n = len(x)
    m = len(x)
    covariancematix = np.zeros((n, m))
    for i in range(0, n):
        for p in range(0, m):
            covariancematix[i, p] = covariance(x[i], x[p])
    return covariancematix

x = np.array([[4, 2, 1], [2,1,3], [1,6,3]])

print(covariance_matrix(x))
```

```
[[ 2.33333333 -0.5        -2.16666667]
 [-0.5         1.         -1.5       ]
 [-2.16666667 -1.5         6.33333333]]
```

L03 Modules and Classes 4

4.1 Qa Load and test the libitmal module

Med `os.path.abspath()` er det muligt at få fat i stien på den fil der eksekveres, i dette tilfælde Jupyter notebook. Ud fra den absolutte sti på filen, kan man tilføje mappen ét niveau højere til Python syspath, dermed bliver det muligt at importere libitmal uden at lave ændringer til environment variabler.

```python
import sys,os
dir_path = os.path.abspath('')
sys.path.append(dir_path+'/..')
print(dir_path)

from libitmal import utils as itmalutils
#print(dir(itmalutils))
#print(itmalutils.__file__)

itmalutils.TestAll()
```

```
D:\repo\itmal\L03
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
```

4.2 Qb Create your own module, with some functions, and test it

Der oprettes et modul, “l03", under mappen “jeppelib".

```
import os

from libjeppe import l03

l03.hello_world()
```

```
Hello world
```

4.3 Qc How do you ‘recompile’ a module?

For at re-kompilere et modul der er blevet ændret kan man bruge

`importlib.reload(<modul_name>)`. I eksemplet her, er modulet ændret til at printe “Hello world - Foobar", uden at Python har været genstartet imellem eksekvering af koden fra Qb og nedenstående kode.

```
import os
import importlib

from libjeppe import l03
importlib.reload(l03)

l03.hello_world()
```

```
Hello world - FooBar
```

4.4 Qd Write a Howto on Python Modules a Packages

4.4.1 Filstruktur

Opret en mappe der indeholder en tom fil, med navnet `__init__.py`. Hver .py fil i mappen er et modul som kan indeholde funktioner, klasser osv. Import et modul med `from <dirname> import <modulefilename>`. Modul stien skal være i PYTHONPATH for at kunne importere den. Man har mulighed for at specificere et alias til modulet, f.eks. `from <dirname> import <module> as <custom_name>`.

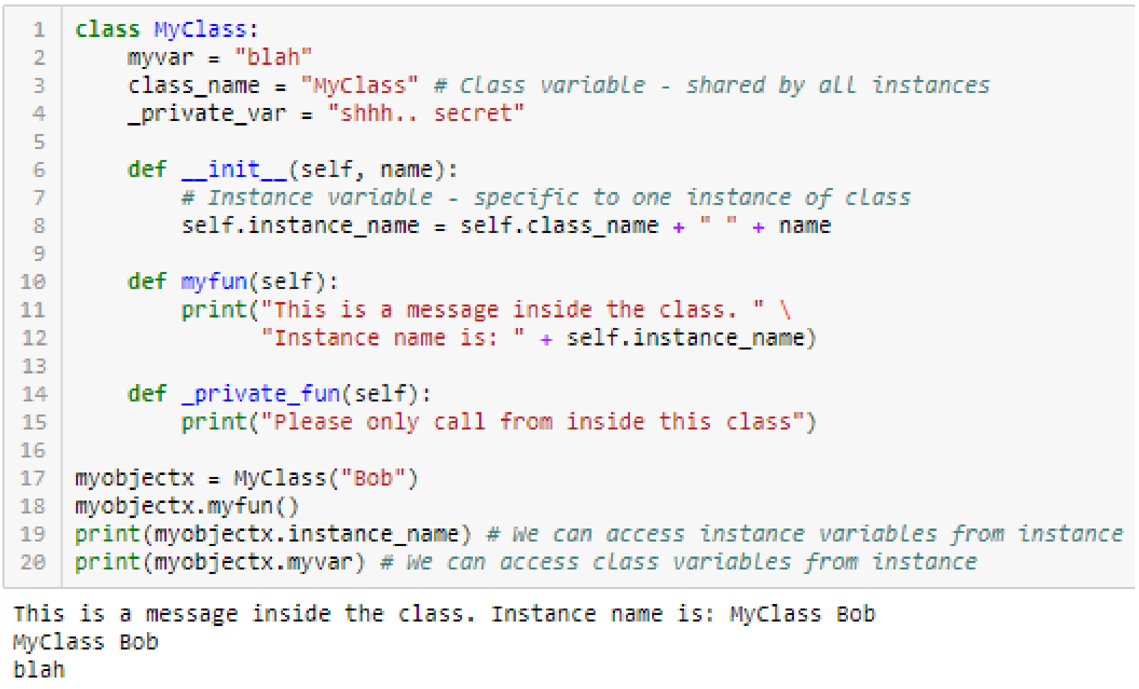
4.4.2 Reload af Moduler

For at reload et modul, kan man bruge `importlib.reload(<modulename>)`, denne metode har dog nogle ulemper. En gammel definition af en funktion/variable/klasse bliver erstattet af en ny definition, men hvis det opdaterede module udelader at definere noget som var defineret før, så lever den gamle definition videre. Det kan man omgå ved at genstarte Python eller at bruge autoreload extension til IPython.

4.5 Qe Extend the class with some public and private functions and member variables

I dette eksempel ses en Python klasse med:

1. klasse variabler
2. instans variabler
3. constructor
4. offentlig metode (myfun)
5. privat metode (_private_fun)



```
class MyClass:
    myvar = "blah"
    class_name = "MyClass" # Class variable - shared by all instances
    _private_var = "shhh.. secret"

    def __init__(self, name):
        # Instance variable - specific to one instance of class
        self.instance_name = self.class_name + " " + name

    def myfun(self):
        print("This is a message inside the class. " \
              "Instance name is: " + self.instance_name)

    def _private_fun(self):
        print("Please only call from inside this class")

myobjectx = MyClass("Bob")
myobjectx.myfun()
print(myobjectx.instance_name) # We can access instance variables from instance
print(myobjectx.myvar) # We can access class variables from instance
```

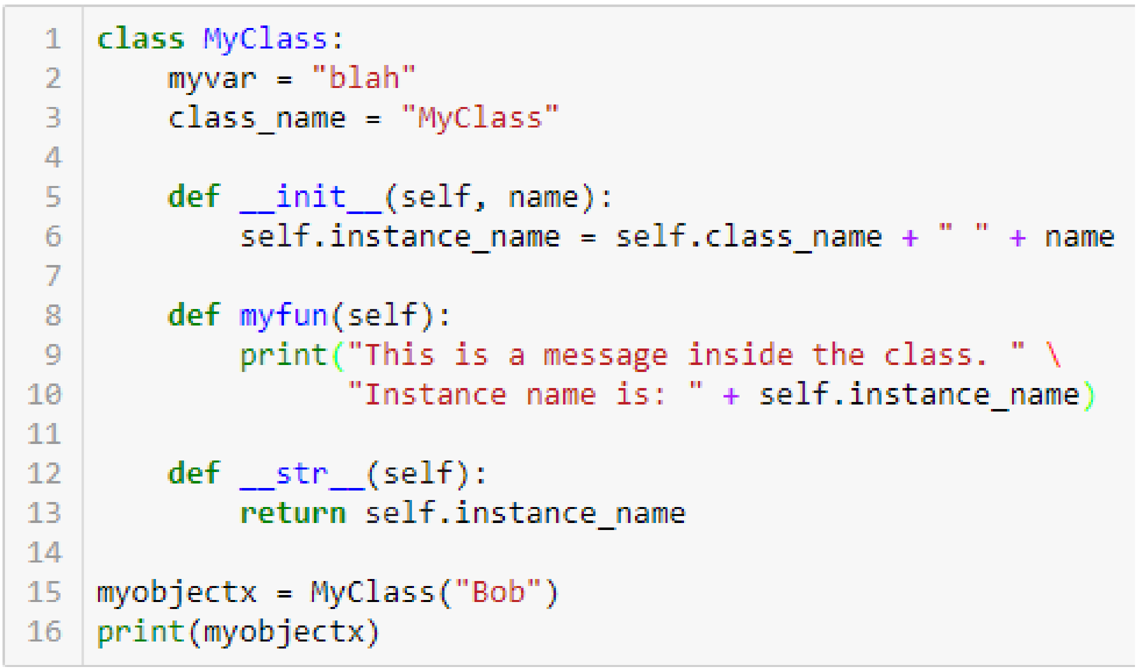
```
This is a message inside the class. Instance name is: MyClass Bob
MyClass Bob
blah
```

4.6 Qf Extend the class with a Constructor

Er allerede gjort i Qc, constructor defineres: `def __init__(self)`.

4.7 Qg Extend the class with a to-string function

I Python kan man override to-string funktionen på en klasse ved at definere en `__str__(self)` metode.



```python
class MyClass:
    myvar = "blah"
    class_name = "MyClass"

    def __init__(self, name):
        self.instance_name = self.class_name + " " + name

    def myfun(self):
        print("This is a message inside the class. " \
              "Instance name is: " + self.instance_name)

    def __str__(self):
        return self.instance_name

myobjectx = MyClass("Bob")
print(myobjectx)
```

```
MyClass Bob
```

4.8 Qh Write a Howto on Python Classes

En interessant detalje omkring klasser i Python, er at man ikke har “private"variabler/metoder i en klasse, i hvert fald ikke som man kender det fra andre sprog. I Python bruger man den konvention at interne/private metoder er præfikset med et underscore, f.eks. `def _do_stuff(self)`. Det er så op til brugeren af klassen/API at kende til denne konvention og undlade at bruge interne dele af API - eller i hvert fald vide at man gør det på egen risiko.

Vil man gøre det endnu mere tydeligt at en intern metode ikke skal bruges udenfor klassen, kan man præfiks metodenavnet med dobbelt underscore, `__do_stuff()`. Det får til at obfuskere metodenavnet for kode udenfor klassen.

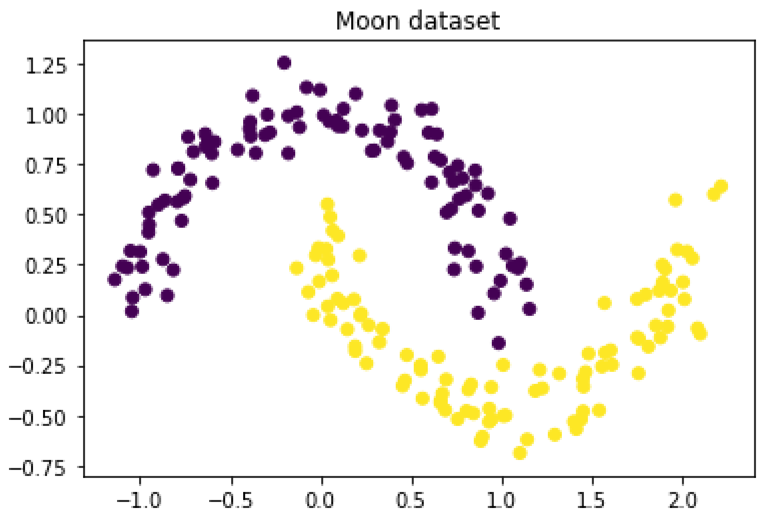
L03 Datasets 5

5.1 Qa Data load function (Moon)

Her genereres et sæt "Moon"data med `sklearn` biblioteket, `noise` er valgt til 0.1.

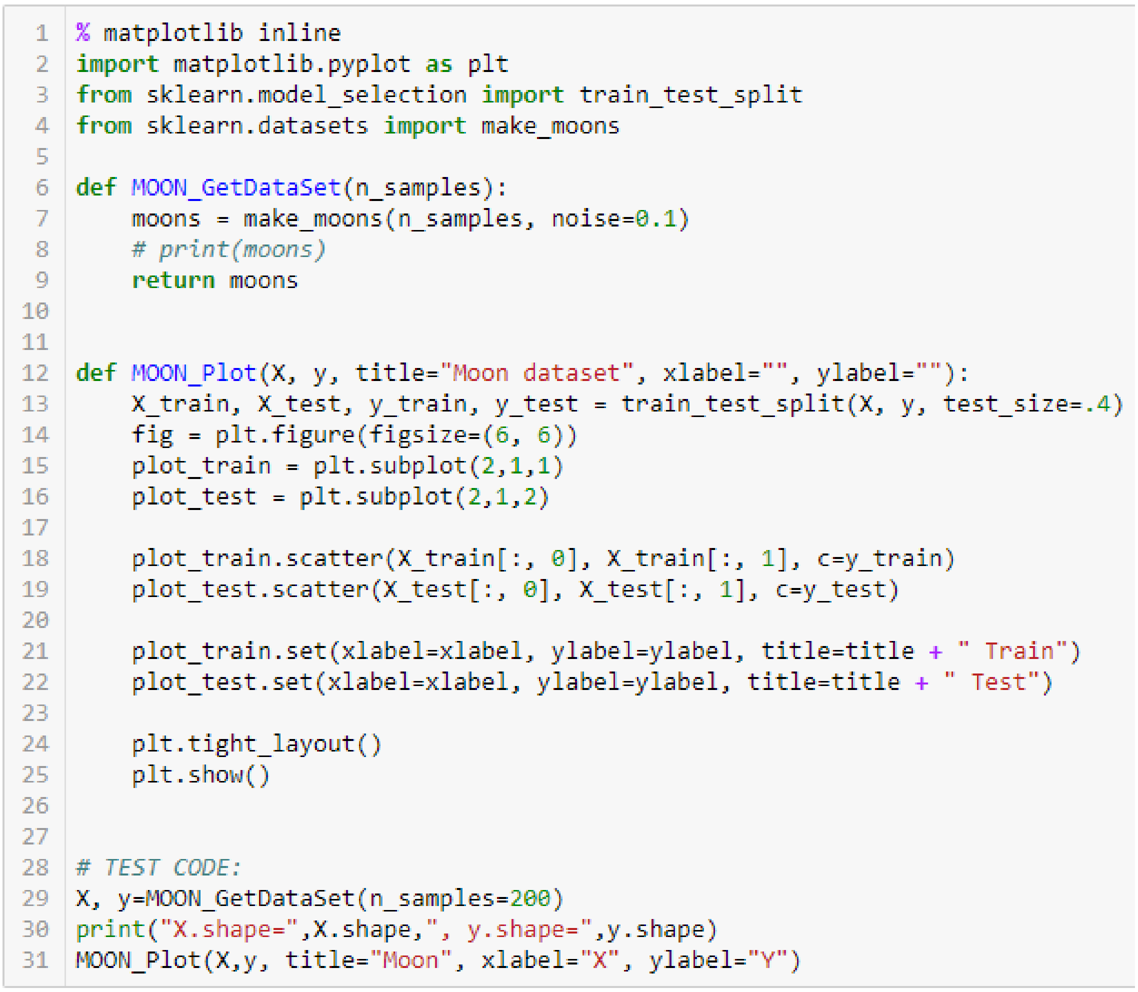
```python
% matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_moons

def MOON_GetDataSet(n_samples):
    moons = make_moons(n_samples, noise=0.1)
    return moons


def MOON_Plot(X, y):
    fig, ax = plt.subplots()
    ax.scatter(X[:, 0], X[:, 1], c=y)
    ax.set(title='Moon dataset')
    plt.show()


# TEST CODE:
X, y=MOON_GetDataSet(n_samples=200)
print("X.shape=",X.shape,", y.shape=",y.shape)
MOON_Plot(X,y)
```

```
X.shape= (200, 2) , y.shape= (200,)
```



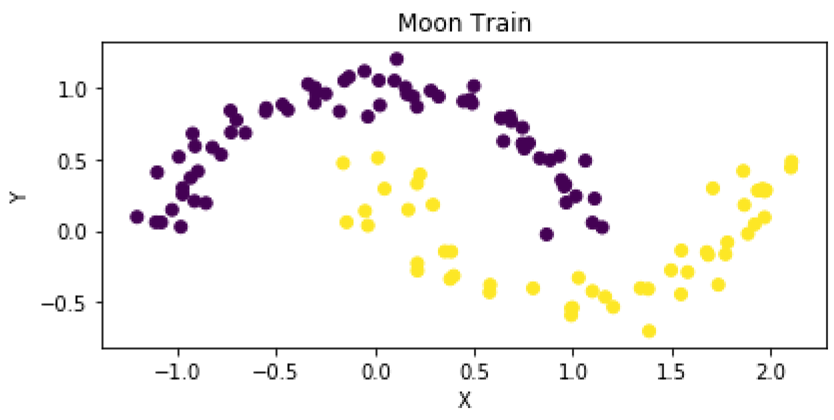
5.2 Qb Try it with a train-test split function

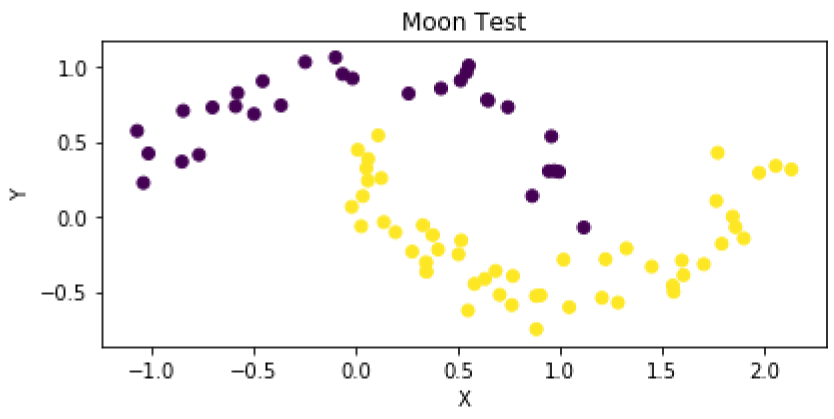
Med `train_test_split` fra `sklearn` biblioteket, er det meget let at splitte et datasæt op i to sæt. Dermed kan man træne sin algoritme på det ene sæt data, og bruge det andet til validering. Her vælges det at 40% af den samlede mængde data skal bruges til test, dermed bliver 60% brugt til træning.



```python
% matplotlib inline
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

def MOON_GetDataSet(n_samples):
    moons = make_moons(n_samples, noise=0.1)
    # print(moons)
    return moons


def MOON_Plot(X, y, title="Moon dataset", xlabel="", ylabel=""):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)
    fig = plt.figure(figsize=(6, 6))
    plot_train = plt.subplot(2,1,1)
    plot_test = plt.subplot(2,1,2)

    plot_train.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
    plot_test.scatter(X_test[:, 0], X_test[:, 1], c=y_test)

    plot_train.set(xlabel=xlabel, ylabel=ylabel, title=title + " Train")
    plot_test.set(xlabel=xlabel, ylabel=ylabel, title=title + " Test")

    plt.tight_layout()
    plt.show()


# TEST CODE:
X, y=MOON_GetDataSet(n_samples=200)
print("X.shape=",X.shape,", y.shape=",y.shape)
MOON_Plot(X,y, title="Moon", xlabel="X", ylabel="Y")
```

```
X.shape= (200, 2) , y.shape= (200,)
```



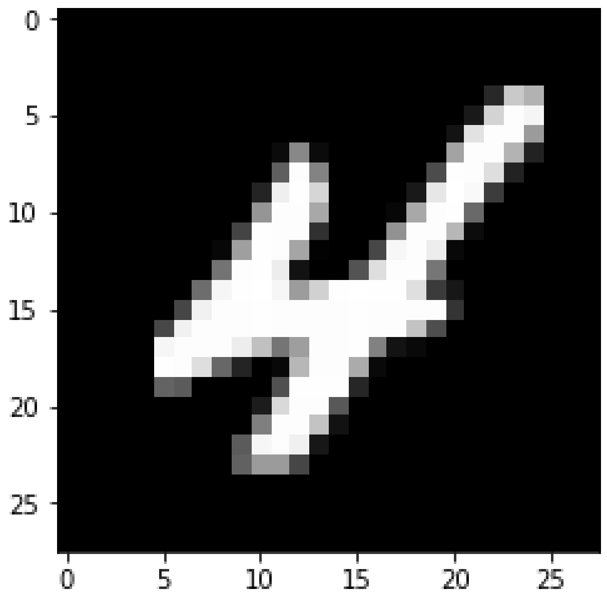


5.3 Qc Data load function (MNIST)

MNIST datasættet indeholder 70.000 eksempler på håndskrevne tal. I eksemplet her bruges `sklearn` til at hente datasættet, og `matplotlib` bruges til at illustrere hvordan et af data punkterne ser ud.

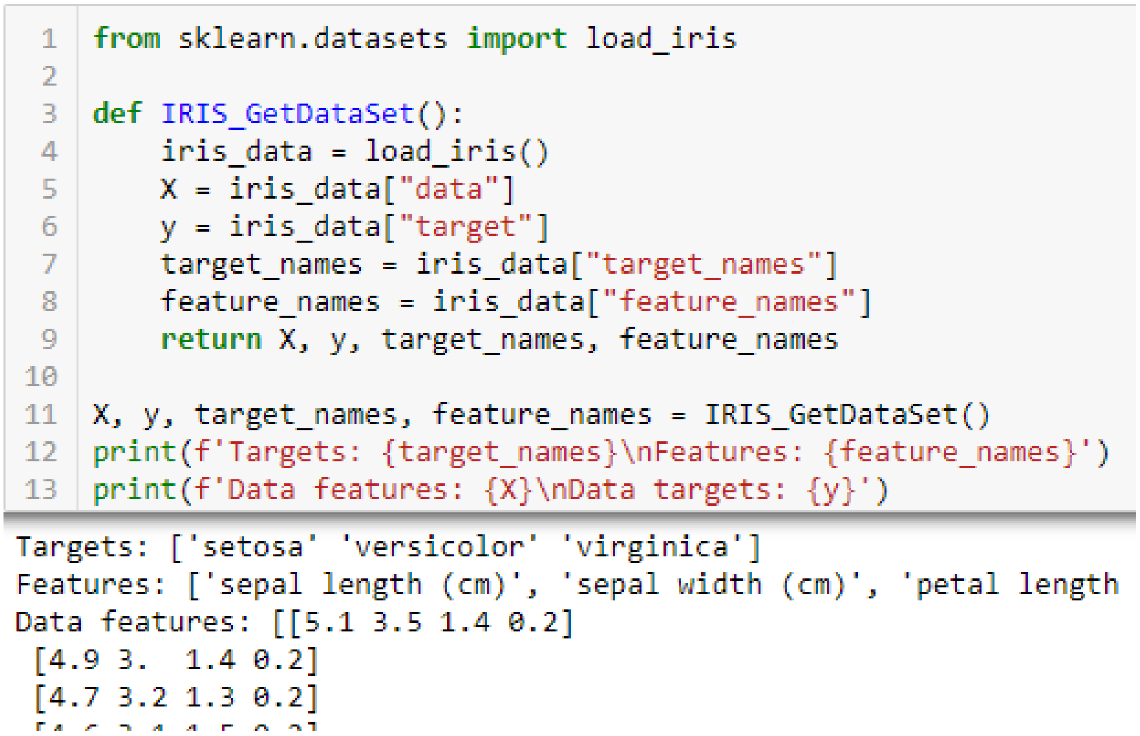
```python
% matplotlib inline
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
import numpy as np


def MNIST_PlotDigit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap='gray')
    plt.show()

def MNIST_GetDataSet():
    mnist_data = fetch_mldata('MNIST original')
    X = mnist_data["data"]
    y = mnist_data["target"]
    return X, y

# TEST CODE:
X, y = MNIST_GetDataSet()
print("X.shape=",X.shape, ", y.shape=",y.shape)

MNIST_PlotDigit(X_train[57])
```

```
X.shape= (70000, 784) , y.shape= (70000,)
```



5.4 Qd Data load function (IRIS)

IRIS datasættet indeholder informationer om 150 iris blomster. For hver blomst er der fire egenskaber: både længde og bredde på hhv. blad og bægerblad. Blomsterne tilhører én af tre forskellige arter af iris blomst: setosa, versicolor og virginica. Funktionen herunder henter IRIS datasættet og deler det op i et array med de fire egenskaber, et array med de tre arter, et array med alle data punkter for egenskaber (150x4) og et array med data punkter for art (150x1).



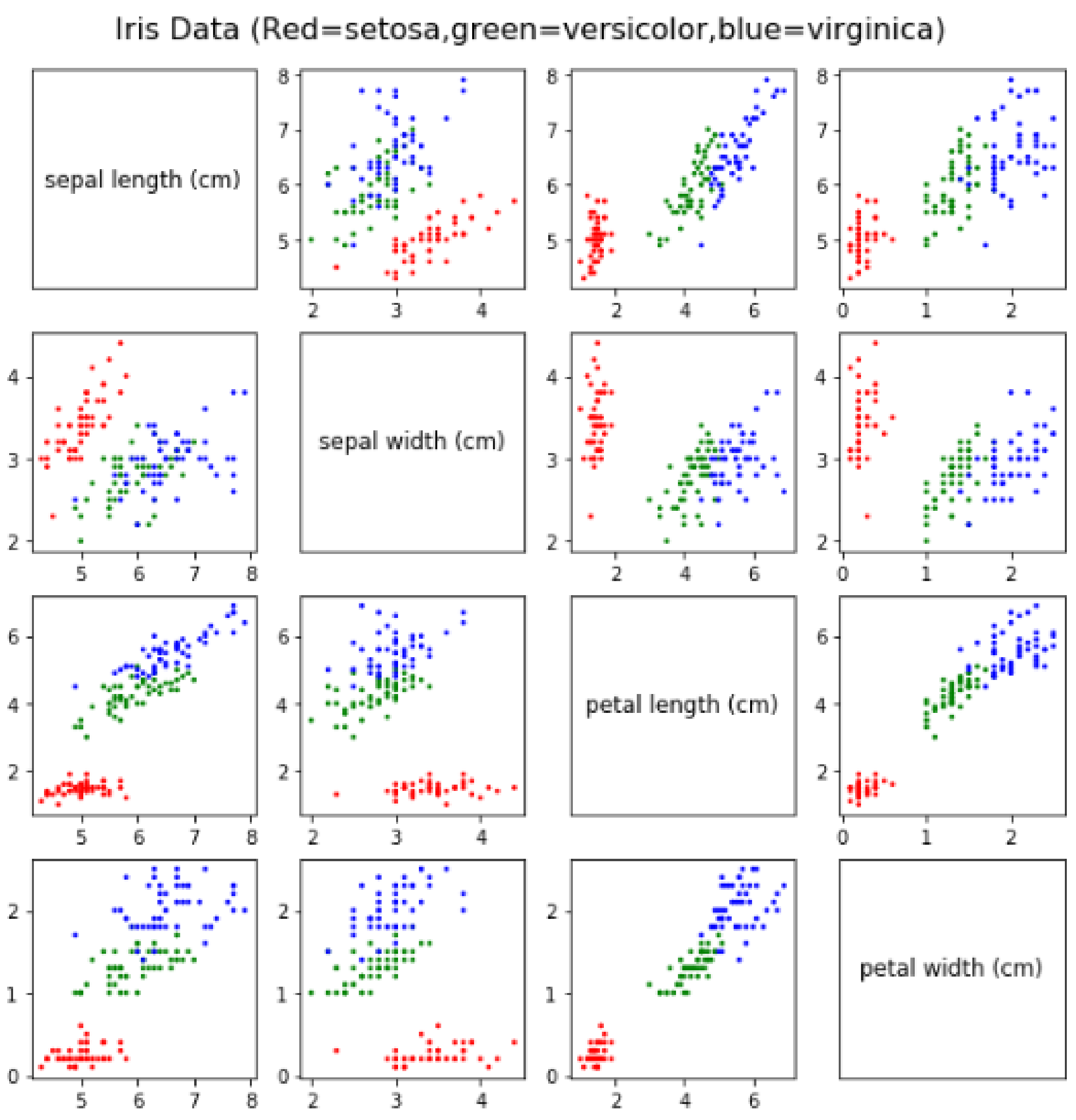
```python
from sklearn.datasets import load_iris

def IRIS_GetDataSet():
    iris_data = load_iris()
    X = iris_data["data"]
    y = iris_data["target"]
    target_names = iris_data["target_names"]
    feature_names = iris_data["feature_names"]
    return X, y, target_names, feature_names

X, y, target_names, feature_names = IRIS_GetDataSet()
print(f'Targets: {target_names}\nFeatures: {feature_names}')
print(f'Data features: {X}\nData targets: {y}')
```

```
Targets: ['setosa' 'versicolor' 'virginica']
Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length
Data features: [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

5.5 Qe Examine the data via scatter plots

Ved at plotte de forskellige egenskaber mod hinanden, kan man se hvordan arterne generelt er forskellige i størrelse.

```python
% matplotlib inline
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris


def IRIS_PlotFeatures(X, y, targets, features):
    colors = [None] * len(X)
    for i in range(len(X)):
        if y[i] == 0:
            colors[i] = 'r'
        elif y[i] == 1:
            colors[i] = 'g'
        else:
            colors[i] = 'b'
    fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                             frameon=True)
    title="Iris Data (Red=setosa,green=versicolor,blue=virginica)"
    fig.suptitle(title, fontsize=16, y=0.92)

    for i in range(4):
        for j in range(4):
            ax = axes[i,j]
            if i == j:
                ax.text(0.5, 0.5, features[i], fontsize=12,
                        transform=ax.transAxes,
                        horizontalalignment='center',
                        verticalalignment='center')
                ax.get_yaxis().set_visible(False)
                ax.get_xaxis().set_visible(False)
            else:
                ax.scatter(X[:,j], X[:,i], c=colors, s=3)

    plt.show()


def IRIS_GetDataSet():
    iris_data = load_iris()
    X = iris_data["data"]
    y = iris_data["target"]
    target_names = iris_data["target_names"]
    feature_names = iris_data["feature_names"]
    return X, y, target_names, feature_names


X, y, targets, features = IRIS_GetDataSet()
IRIS_PlotFeatures(X, y, targets, features)
```



5.6 Qf Add your function to the libitmal python module

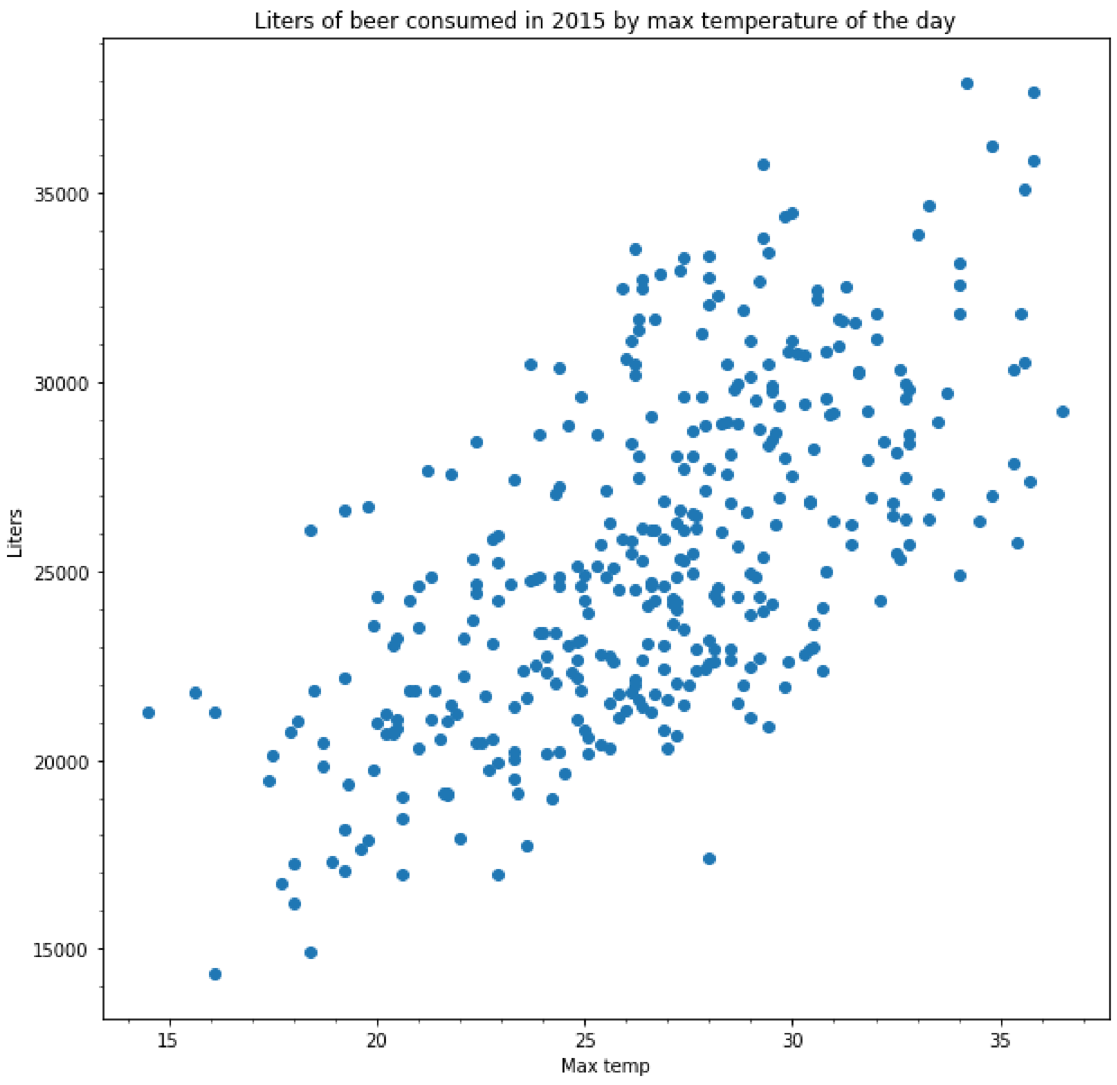
De forskellige funktioner der er lavet i L03 tilføjes et `dataloaders` modul under `libitmal`, så de er lette at genbruge senere. Resultat af at køre koden i Jupyter er ikke medtaget, da det er samme figurer som allerede vist i de tidligere sektioner.

```python
import sys,os
sys.path.append(os.path.abspath('')+'/..')
from sklearn.model_selection import train_test_split
import importlib
from libitmal import dataloaders
importlib.reload(dataloaders)

X, y=dataloaders.MOON_GetDataSet(n_samples=200)
dataloaders.MOON_Plot(X,y)

dataloaders.MNIST_PlotDigit(X_train[57])

X, y, targets, features = dataloaders.IRIS_GetDataSet()
dataloaders.IRIS_PlotFeatures(X, y, targets, features)
```

5.7 Qg Download a data set and do some data exploration of it

Denne opgave gik ud på selv at vælge et datasæt og udføre en analyse af det. Det foreslåede datasæt vedr. øl indtag i Sao Paulo blev valgt. Analysen fokuserer på sammenhæng mellem max temperatur på en given dag, og mængden af øl der blev indtaget den dag.

```python
from numpy import genfromtxt
import matplotlib.pyplot as plt
import numpy
import matplotlib

def transform_commas(line):
    retval = ''
    inQuote = False
    for char in line:
        if char == '"':
            inQuote = not inQuote
        if inQuote == False and char == ',':
            retval += ';'
        else:
            retval += str(char)
    return retval

def to_float(x):
    if x:
        return float(x.strip('"').replace(',','.'))
    else:
        return None

def to_int(x):
    if x:
        return int(x.strip('"').replace('.',''))
    else:
        return None

def get_beer_consumption_data():
    f = open("../datasets/beer_consumption/Consumo_cerveja.csv", "r")
    replaced = (transform_commas(line) for line in f)
    rawData = numpy.loadtxt(replaced,delimiter=';', skiprows=0, dtype=str)
    # features = rawData[0]
    data = rawData[1:]
    max_temp_data = [to_float(x) for x in data[:,3]]
    liters_data = [to_int(x) for x in data[:,6]]
    return max_temp_data, liters_data


def plot_max_temp_vs_liters_consumed(x, y):
    fig, ax = plt.subplots(figsize=(10,10))
    ax.scatter(x, y)
    ax.set(xlabel='Max temp', ylabel='Liters',
           title='Liters of beer consumed in 2015 by max temperature of the day')
    plt.gca().yaxis.set_major_locator(matplotlib.ticker.MultipleLocator(5000))
    plt.gca().yaxis.set_minor_locator(matplotlib.ticker.MultipleLocator(1000))
    plt.gca().xaxis.set_major_locator(matplotlib.ticker.MultipleLocator(5))
    plt.gca().xaxis.set_minor_locator(matplotlib.ticker.MultipleLocator(1))
    plt.show()


x,y = get_beer_consumption_data()
plot_max_temp_vs_liters_consumed(x,y)
```


Liters of beer consumed in 2015 by max temperature of the day
Liters
35000
30000
25000
20000
15000
15
20
25
30
35
Max temp

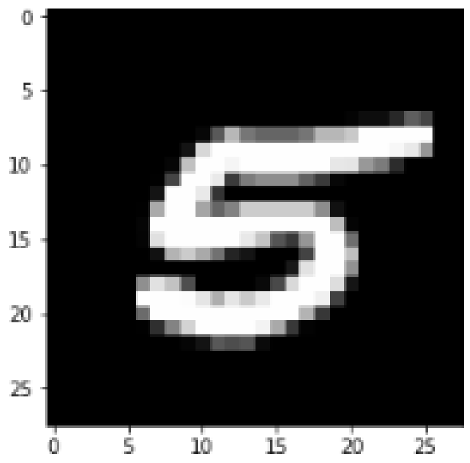
L03 Dummy Classifier 6

6.1 Qa Add a Stochastic Gradient Decent [SGD] Classifier

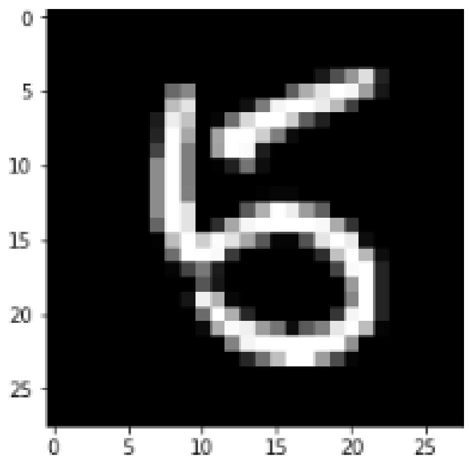
Der laves en SGD classifier som trænes på data fra MNIST sættet. I resultatet ses der eksempel på både “True-Positive"og “False-Negative", hvor SGD Classifier først rammer korrekt og derefter fejlagtigt klassificerer et 5-tal som værende ikke-5.

```python
import sys,os
sys.path.append(os.path.abspath('')+'/..')

%matplotlib inline
from libitmal import dataloaders
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import SGDClassifier
import numpy as np
import sklearn


X, y = dataloaders.MNIST_GetDataSet()
print("X.shape=",X.shape) # print X.shape= (70000, 28, 28)
if X.ndim==3:
    print("reshaping X..")
    assert y.ndim==1
    X = X.reshape((X.shape[0],X.shape[1]*X.shape[2]))
assert X.ndim==2
print("X.shape=",X.shape) # X.shape= (70000, 784)

target=5.0
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)
y_train_5 = (y_train == target)
y_test_5  = (y_test == target)

sgd_clf = SGDClassifier(max_iter=5, tol=None, random_state=42)
sgd_clf.fit(X_train, y_train_5)

result_limit = 5
print('Trying classification, be patient...')
for i in range(0, len(y_test)):
    classification = sgd_clf.predict(X_test[i].reshape(1,-1))
    if y_test[i] == target and classification:
        print('Classification successful:')
        dataloaders.MNIST_PlotDigit(X_test[i])
        result_limit = result_limit - 1
    elif (y_test[i] != target and classification) \
        or (y_test[i] == target and not classification):
        print('Classification failed. Actual value = ' \
              f'{y_test[i]}, classifier prediction = {classification}:')
        dataloaders.MNIST_PlotDigit(X_test[i])
        result_limit = result_limit - 1
    if result_limit <= 0:
        break

if result_limit == 5:
    print('Made it through the data set without encountering any training ' \
          'items of the desired target, try re-running')
else:
    print('Classification trial complete')
```

```
X.shape= (70000, 784)
X.shape= (70000, 784)
Trying classification, be patient...
Classification successful:
```

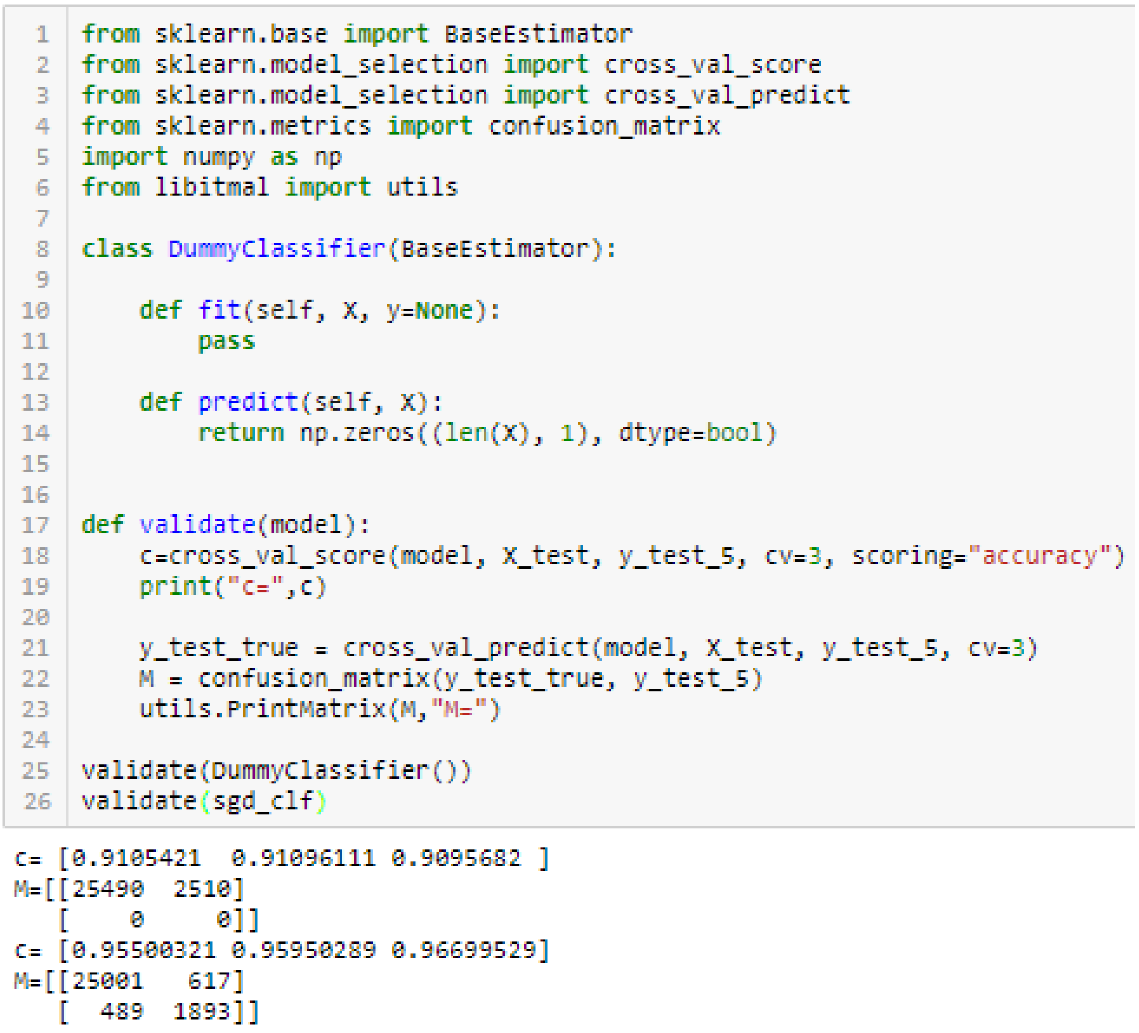


```
Classification failed. Actual value = 5.0, classifier prediction = [False]:
```



6.2 Qb Implement a dummy binary classifier

En Python klasse er vist herunder som overholder det forventede interface til en classifier, da den har både `fit(X, y)` og `predict(X)` metoder. Det eneste denne classifier gør, er dog at klassificere alt input fra MNIST sættet som værende forskellige fra 5.

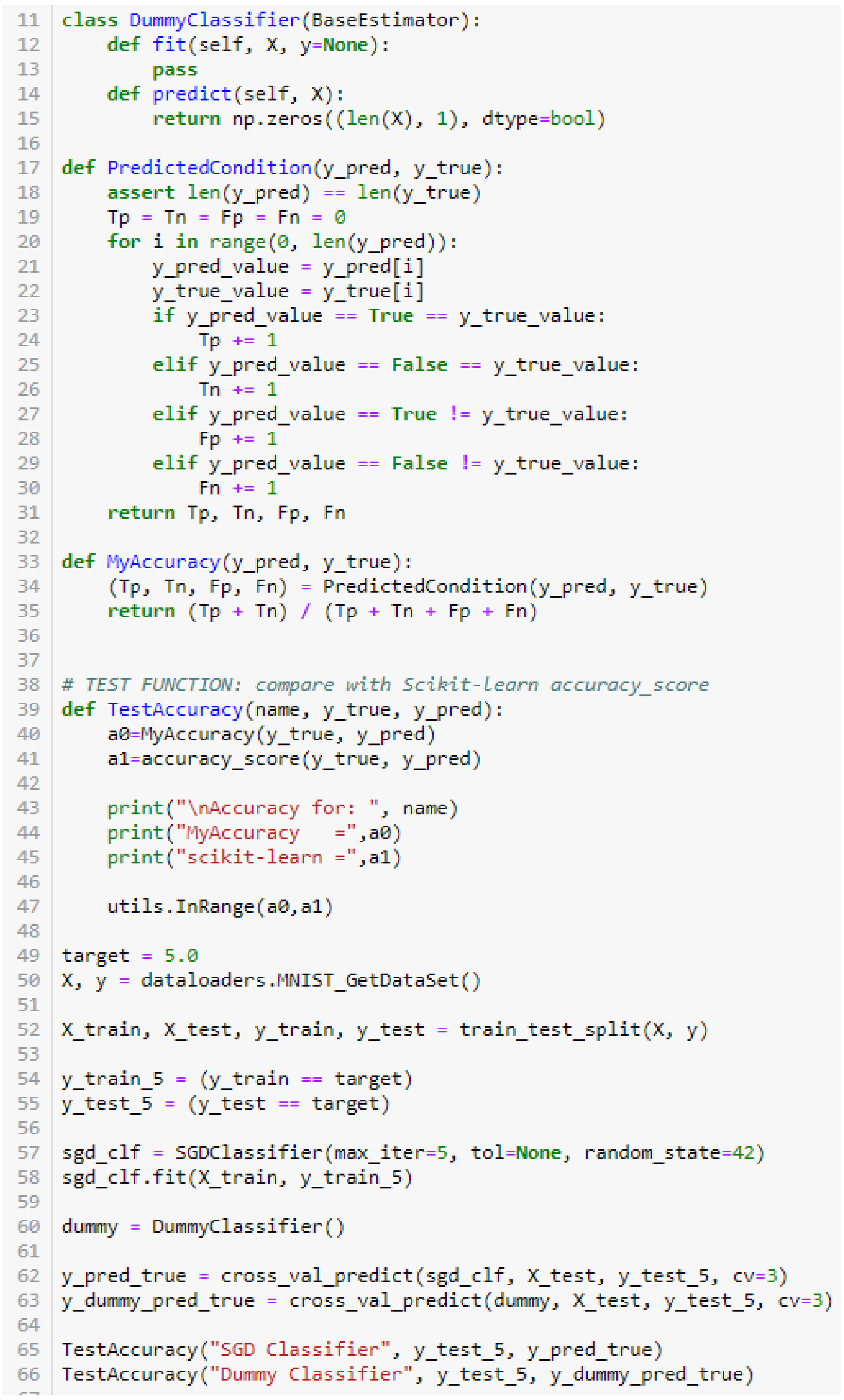


```python
from sklearn.base import BaseEstimator
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
import numpy as np
from libitmal import utils

class DummyClassifier(BaseEstimator):

    def fit(self, X, y=None):
        pass

    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)


def validate(model):
    c=cross_val_score(model, X_test, y_test_5, cv=3, scoring="accuracy")
    print("c=",c)

    y_test_true = cross_val_predict(model, X_test, y_test_5, cv=3)
    M = confusion_matrix(y_test_true, y_test_5)
    utils.PrintMatrix(M,"M=")

validate(DummyClassifier())
validate(sgd_clf)
```

```
c= [0.9105421  0.91096111 0.9095682 ]
M=[[25490  2510]
   [    0     0]]
c= [0.95500321 0.95950289 0.96699529]
M=[[25001   617]
   [  489  1893]]
```

L03 Metrics 7

7.1 Qa Implement the Accuracy function and test it on the MNIST data.



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

def PredictedCondition(y_pred, y_true):
    assert len(y_pred) == len(y_true)
    Tp = Tn = Fp = Fn = 0
    for i in range(0, len(y_pred)):
        y_pred_value = y_pred[i]
        y_true_value = y_true[i]
        if y_pred_value == True == y_true_value:
            Tp += 1
        elif y_pred_value == False == y_true_value:
            Tn += 1
        elif y_pred_value == True != y_true_value:
            Fp += 1
        elif y_pred_value == False != y_true_value:
            Fn += 1
    return Tp, Tn, Fp, Fn

def MyAccuracy(y_pred, y_true):
    (Tp, Tn, Fp, Fn) = PredictedCondition(y_pred, y_true)
    return (Tp + Tn) / (Tp + Tn + Fp + Fn)


# TEST FUNCTION: compare with Scikit-learn accuracy_score
def TestAccuracy(name, y_true, y_pred):
    a0=MyAccuracy(y_true, y_pred)
    a1=accuracy_score(y_true, y_pred)

    print("\nAccuracy for: ", name)
    print("MyAccuracy    =",a0)
    print("scikit-learn =",a1)

    utils.InRange(a0,a1)

target = 5.0
X, y = dataloaders.MNIST_GetDataSet()

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_train_5 = (y_train == target)
y_test_5 = (y_test == target)

sgd_clf = SGDClassifier(max_iter=5, tol=None, random_state=42)
sgd_clf.fit(X_train, y_train_5)

dummy = DummyClassifier()

y_pred_true = cross_val_predict(sgd_clf, X_test, y_test_5, cv=3)
y_dummy_pred_true = cross_val_predict(dummy, X_test, y_test_5, cv=3)

TestAccuracy("SGD Classifier", y_test_5, y_pred_true)
TestAccuracy("Dummy Classifier", y_test_5, y_dummy_pred_true)
```

Output:

```
Accuracy for:  SGD Classifier
MyAccuracy   = 0.9633142857142857
scikit-learn = 0.9633142857142857

Accuracy for:  Dummy Classifier
MyAccuracy   = 0.9140571428571429
scikit-learn = 0.9140571428571429
```

7.2 Qb Implement Precision, Recall and F_1-score and test it on the MNIST data.

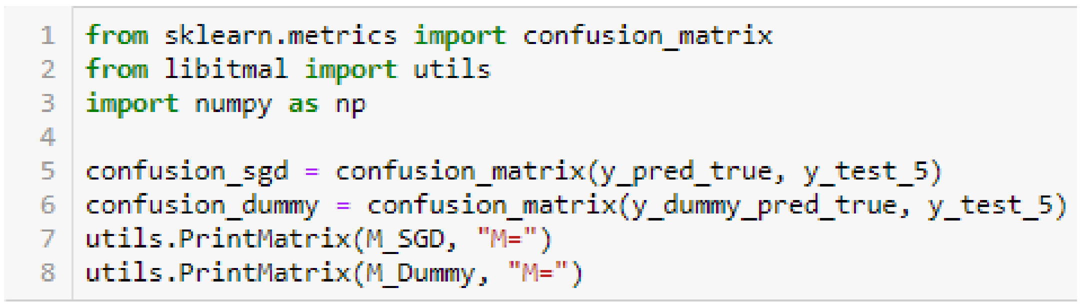
```python
def MyPrecision(y_true, y_pred):
    (Tp, Tn, Fp, Fn) = PredictedCondition(y_true, y_pred)
    if Tp == 0 and Fp == 0:
        return 0
    return Tp / (Tp + Fp)

def MyRecall(y_true, y_pred):
    (Tp, Tn, Fp, Fn) = PredictedCondition(y_true, y_pred)
    if Tp == 0 and Fn == 0:
        return 0
    return Tp / (Tp + Fn)

def MyF1Score(y_true, y_pred):
    p = MyPrecision(y_pred, y_true)
    r = MyRecall(y_pred, y_true)
    if p == 0:
        return 0
    return (2* p * r)/(p + r)

def test(own, scikit, metric, y_true, y_pred):
    my_val=own(y_pred, y_true)
    scikit_val=scikit(y_true, y_pred)
    print(f"My {metric}".ljust(20) + f": {my_val}")
    print(f"Scikit {metric}".ljust(20) + f": {scikit_val}")
    utils.InRange(my_val,scikit_val)

def test_predictor(name, y_true, y_pred):
    print("Test results for predictor: " + name)
    test(MyPrecision, precision_score, "Precision", y_true, y_pred)
    test(MyRecall, recall_score, "Recall", y_true, y_pred)
    test(MyF1Score, f1_score, "F1 Score", y_true, y_pred)
    print("")

test_predictor("SGD", y_pred_true, y_test_5)
test_predictor("Dummy", y_dummy_pred_true, y_test_5)
```

Output:

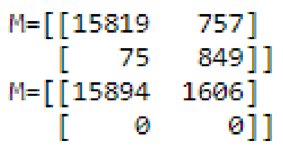
```
Test results for predictor: SGD
My Precision        : 0.6868351063829787
```

```
Scikit Precision     : 0.6868351063829787
My Recall            : 0.8579734219269103
Scikit Recall        : 0.8579734219269103
My F1 Score          : 0.7629246676514033
Scikit F1 Score      : 0.7629246676514033

Test results for predictor: Dummy
My Precision         : 0.0
Scikit Precision     : 0.0
My Recall            : 0
Scikit Recall        : 0.0
My F1 Score          : 0
Scikit F1 Score      : 0.0
```

7.3 Qc The Confusion Matrix



```python
from sklearn.metrics import confusion_matrix
from libitmal import utils
import numpy as np

confusion_sgd = confusion_matrix(y_pred_true, y_test_5)
confusion_dummy = confusion_matrix(y_dummy_pred_true, y_test_5)
utils.PrintMatrix(M_SGD, "M=")
utils.PrintMatrix(M_Dummy, "M=")
```

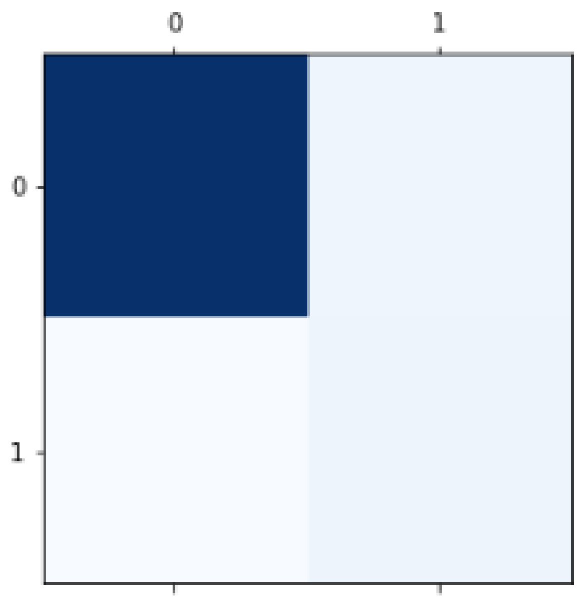


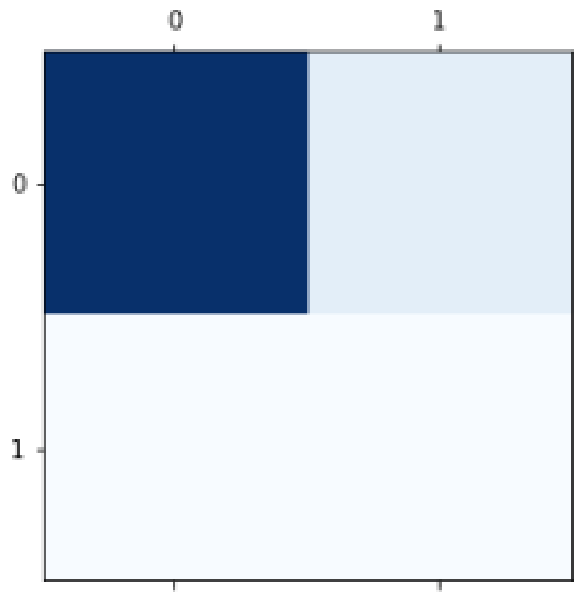
```
M=[[15819   757]
   [   75   849]]
M=[[15894  1606]
   [    0     0]]
```

7.4 Qd A Confusion Matrix Heat-map

```python
%matplotlib inline
import matplotlib.pyplot as plt

plt.matshow(confusion_sgd, cmap=plt.cm.Blues)
plt.matshow(confusion_dummy, cmap=plt.cm.Blues)
```

```
<matplotlib.image.AxesImage at 0x1a4c25cef0>
```

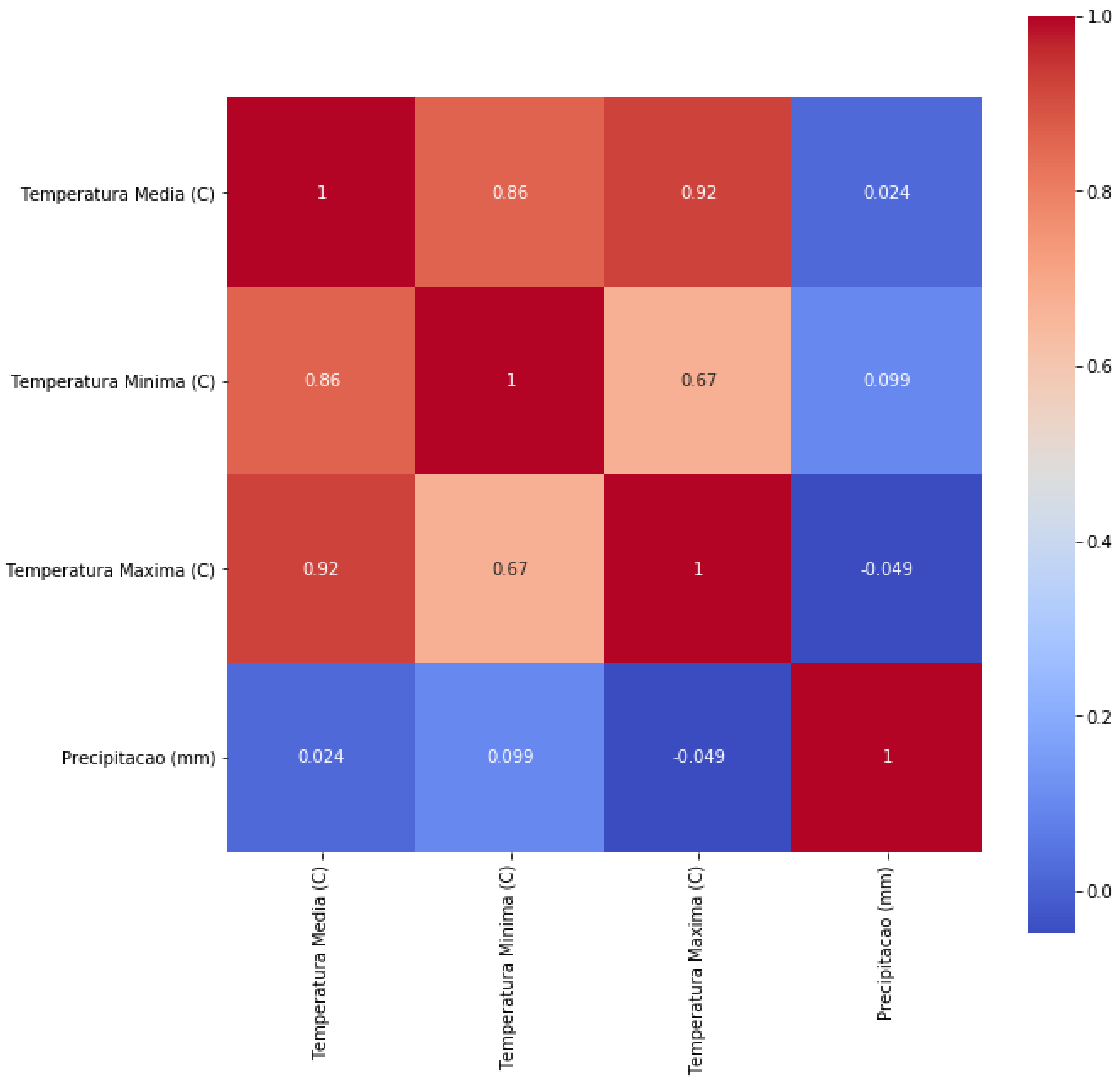




7.5 Qe Run a classifier on your data

```python
# inspired from https://www.kaggle.com/rcfreitas/python-ml-breast-cancer-diagnostic-data-set
data = pd.read_csv("../datasets/beer_consumption/Consumo_cerveja.csv", delimiter=',').dropna()
data = data.drop(data.columns[[-1, 0]], axis=1)
data = data.replace({',':'.'},regex=True).apply(pd.to_numeric,1)

features_mean= list(data.columns[0:4])

plt.figure(figsize=(10,10))
sns.heatmap(data[features_mean].corr(), annot=True, square=True, cmap='coolwarm')
plt.show()

X = data.loc[:,features_mean]
y = data.loc[:, 'Final de Semana']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

accuracy_all = []
cvs_all = []

def evaluate_classifier(model, name):
    start = time.time()

    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    scores = cross_val_score(model, X, y, cv=5)

    end = time.time()

    accuracy_all.append(accuracy_score(prediction, y_test))
    cvs_all.append(np.mean(scores))

    print(name + " Classifier Accuracy: {0:.2%}".format(accuracy_score(prediction, y_test)))
    print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(np.mean(scores), np.std(scores)*2))
    print("Execution time: {0:.5} seconds \n".format(end-start))


evaluate_classifier(SGDClassifier(random_state=45), "SGD")
evaluate_classifier(SVC(random_state=45), "SVC")
evaluate_classifier(NuSVC(random_state=45), "NuSVC")
evaluate_classifier(LinearSVC(random_state=45), "LinearSVC")


df = pd.DataFrame({'accuracy_all':accuracy_all}, index=['SGD', 'SVC', 'NuSVC', 'LinearSVC'])
print(df)
```

Output:



```
SGD Classifier Accuracy: 35.62%
Cross validation score: 47.42% (+/- 40.39%)
Execution time: 0.044998 seconds

SVC Classifier Accuracy: 60.27%
Cross validation score: 70.69% (+/- 3.64%)
Execution time: 0.09088 seconds

NuSVC Classifier Accuracy: 58.90%
Cross validation score: 65.75% (+/- 3.21%)
Execution time: 0.071895 seconds

LinearSVC Classifier Accuracy: 61.64%
Cross validation score: 46.41% (+/- 41.38%)
Execution time: 0.203 seconds

           accuracy_all
```

```
SGD          0.356164
SVC          0.602740
NuSVC        0.589041
LinearSVC    0.616438
```