

# SWMAL gruppe 21- O[1]

Navn	StudieNummer	AUID
Mohamed Hashem Abodu	2019100895	Au656408

26/09/2023

<b>L01/Intro.ipynb</b>	<b>2</b>
Qa) The Parameter and the R2 score	2
Qb) using K-nearest Neighbours	2
Qc) Tuning Parameter for K-nearest	3
Qd) Trying out a Neural Network	3
<b>L01/Modules_and_classes.ipynb</b>	<b>3</b>
Qa) Load and test the libitmal module	3
Qb) Create my own Module with some function and test it.	3
Qc) How to recompile a module?	4
Qe) Extend the class with public and private functions	4
Qf) Extend the class with Constructor	4
Qg) Extend the class with a to-string function	4
<b>L02/cost_function.ipynb</b>	<b>4</b>
Qa) Given the following $x(t)$ 's construct and print the X matrix in python	4
Qb) Implement the L1 and L2 norms for vectors in python	4
Qc) Construct the root mean square error (rmse) function	5
Qd) Similar Construct the Mean Absolute Error (MAE) & evaluate it	5
Qe) robust code	5
Qf) Conclusion	5
<b>L02/dummy_Classifier.ipynb</b>	<b>5</b>
Qa) Load and display the MNIST data	5
Qb) Add a stochastic Gradient Decent SGD classifier	5
Qc) Implement a dummy binary classifier	6
Qd) Conclusion	6
<b>L02/performance_metrics.ipynb</b>	<b>6</b>
Qa) implement the Accuracy function and test it on the MNIST data	6
Qb) Implement precision, Recall and F1- score and test it on the MNIST data for both SGD and dummy classifier models	6
Qc) the confusion matrix	6
Qd) A confusion matrix Heat-map	6
Qe) Conclusion	6

# L01/Intro.ipynb

## Qa) The Parameter and the R2 score:

To extract the coefficients  $\theta_0$  and  $\theta_1$  from a linear regression model, we need to use the `coef_` and `intercept_` attributes of the model from Sklearn.

- Defining R2 score in broad terms:
  - In broad terms, R2 score is a statistical measure that represents the goodness of fit of a regression model. It tells us how well the model fits the observed data.
  - The maximum value for R2 score is 1, which indicates a perfect fit. It can range from infinity negative to 1. Where 0 indicates the model is no better than a simple mean. And the negative score indicates that the model is arbitrary worse [[Sklearn](#)]
- Which R2 score is best to have? What describes the R2 score best ?

We get R2 equals 0 when the model does not predict any variability in the model and it does not learn any relationship between the dependent and independent variables, however where we get R2 equals 1 is when the model perfectly fits the data and there is no difference between the predicted value and the actual value.

[[Geeksforgeeks](#)]

With that being said, a higher R2 score is better, where R2 is a measure of the goodness or fitness for a linear regression or a model.

```
[ ] # TODO: add your code here..
# to extract the coefficients theta0 and theta1 from a linear regression model
# we can use coef_ and intercept_ attributes of the model

from sklearn.linear_model import LinearRegression
model = LinearRegression()

model.fit(X,y)

theta0 = model.intercept_
theta1 = model.coef_[0]

r2_score = model.score(X,y)

print(theta0)
print(theta1)
print("\n")
print(f"R2 score: {r2_score: .3f}")

[4.8530528]
[4.91154459e-05]

R2 score: 0.734
```

Figure 1. Extracting the coefficients  $\theta_0$  and  $\theta_1$ .

## Qb) using K-nearest Neighbours

- Lets estimate the k-nearest neighbours for Cyprus, and then compare it to linear regression:
  - as shown in the code output in the screenshot below, the k-nearest neighbours estimate is 5.77 while using linear regression the result was 5.94.
- Can we compare the score-method for both k-nearest and linear regression models ? Do they have the same score-method?
  - According to the documentations for both [[k-neighbours](#)] and [[linear regression](#)], the score method is the same. which means they can be compared to assess their performance. Also using k-neighbours regression model has a better score of 0.85 while linear regression model has 0.734

```
# TODO: add your code here..
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors=3)

# Train the model
knn.fit(X, y)
X_new = [[22587]] # Cyprus' GDP per capita
y_pred = knn.predict(X_new)
print(f"prededion for Cyprus: {y_pred}")
r2_score = knn.score(X,y)
print(f"r2 score: {r2_score}")

X.shape= (29, 1)
y.shape= (29, 1)
prededion for Cyprus: [[5.76666667]]
r2 score: 0.8525732853499179
```

Figure 2. Using K-nearest model

## Qc) Tuning Parameter for K-nearest

- Why does the parameter k\_neighbor = 1 give k-nearest a good score ?
  - At k=1, the KNN tends to closely follow the training data and thus shows a high training score. [Medium](#)
  - The model is designed to make predictions based on the closest single data point in the training set.
- Is it preferred to have an estimator with score= 1 for the job?
  - No, the Knn model with k=1 may be impressive with the score=1, but it's highly prone to overfitting. It may perform poorly on data they haven't seen before

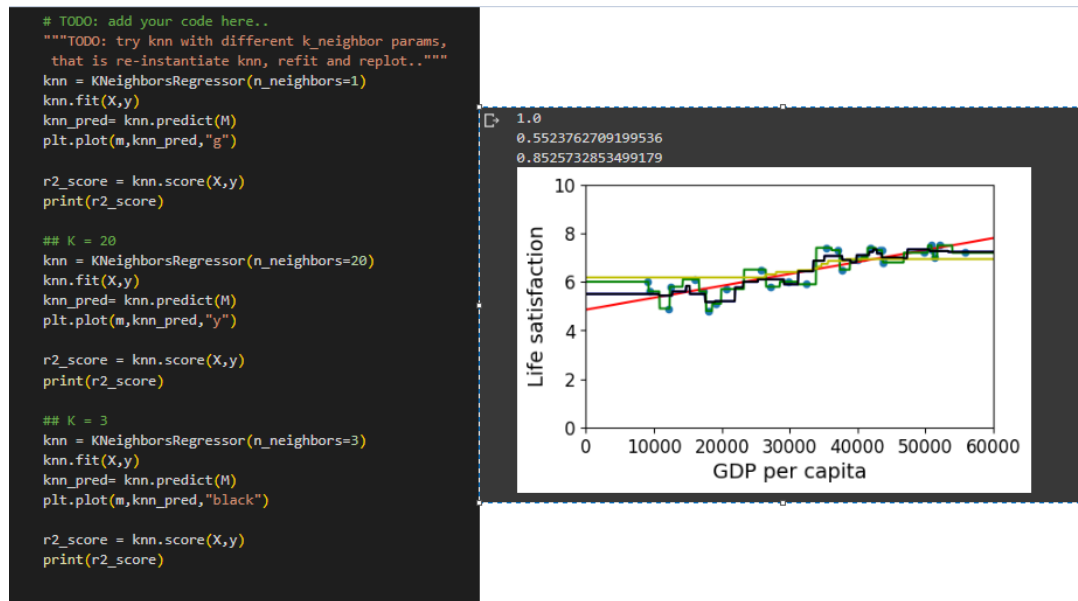


Figure 3. Tuning the code with different K, and the results- output on the right

## Qd) Trying out a Neural Network

- In the next part of the exercise I am going to predict the value for cyprus and the score value for the training set, just as in linear and Knn models.
  - The prediction value for Cyprus' last run was 2.648, which means the model predicts that Cyprus would have a life satisfaction score of ca 2.65 based on the GDP per capita from the data set. The negative R2 score shows that the mlpRegressor isn't a good fit for our data,
- Can we compare the score function between MLPRegressor and KNN?
  - Yes, but because the MLP model isn't a good fit for our dataset, the difference is big between Knn, linear and MLP score(-6.74)

```

from sklearn.neural_network import MLPRegressor

# Setup MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=(10,), solver='adam', activation='relu', tol=1E-5, max_iter=1000)
mlp.fit(X, y.ravel())

# lets make a MLP regressor prediction and redo the plots
y_pred_mlp = mlp.predict(M)

plt.plot(m, y_pred_lin, "r")
plt.plot(m, y_pred_knn, "b")
plt.plot(m, y_pred_mlp, "k")

# TODO: add your code here..
mlp.fit(X,y.ravel())

cyprus = [[18064.288]]
cyprus_pred = mlp.predict(cyprus)
print(f"MLP prediction :{cyprus_pred[0]}")
mlp_score = mlp.score(X,y.ravel())
print(f"MLP score :{mlp_score}")

#assert False, "TODO: predict value for Cyprus and fetch the score() from the fitting."

```

MLP prediction :2.6489694495609744  
MLP score :-6.742555835300249

Figure 4. Training the MLPRegressor model and the output.

## L01/Modules\_and\_classes.ipynb

In this exercise we are going to learn about Python basics in terms of libraries, modules and classes.

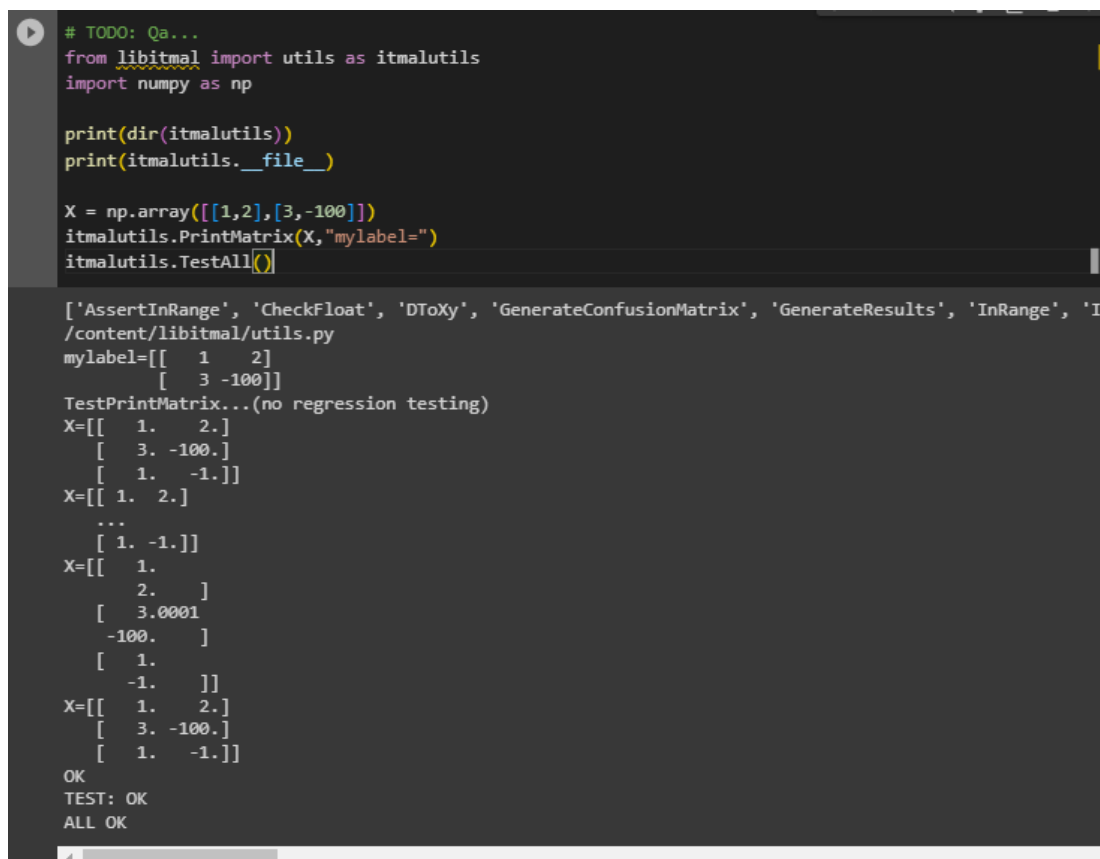
### Qa) how to load and test a module ?

- Since I'm working with Google Colab, all I had to do was to upload the folder with the module and then append the path to sys.path to make it accessible like this:

```
from google.colab import files
uploaded = files.upload()

import sys
sys.path.append("/content/libitmal")
```

And then importing the model as described in the exercise:



```
# TODO: Qa...
from libitmal import utils as itmalutils
import numpy as np

print(dir(itmalutils))
print(itmalutils.__file__)

X = np.array([[1,2],[3,-100]])
itmalutils.PrintMatrix(X,"mylabel=")
itmalutils.TestAll()
```

['AssertInRange', 'CheckFloat', 'DToXy', 'GenerateConfusionMatrix', 'GenerateResults', 'InRange', 'I  
/content/libitmal/utils.py  
mylabel=[[ 1 2]  
[ 3 -100]]  
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

Figure 5. Importing the model and then testing its functions.

Qb) Here I'm going to create my own module and test it with some functions.

Now i have created a new module called Mathlib.py that contains some mathematical methods:

```
import math

def plus(a,b):
    return a+b

def multiply(a,b):
    return a*b

def squareroot(a):
    return math.sqrt(a)
```

Then i will import the module in the same way as the last exercise:

```
[ ] # TODO: Qb...
import importlib
import MyMatlib
# Reload the module after changing it.
importlib.reload(MyMatlib)

Multiply = MyMatlib.multiply(5,5)

Plus = MyMatlib.plus(5,5)

print(Multiply)

print(Plus)

print(MyMatlib.squareroot(23))

25
10
4.795831523312719
```

Figure 6. Importing the costume made module and testing it.

Qc) Can we recompile a module after changing it?

- When reloading the module code, in google colab, we can use the following reload() method from Importlib. Sometimes it is necessary to restart the runtime in order for the module to be reloaded. See Figure 6.

## Qe) How to implement public and private function in the class

- How to implement private functions and member variables in python?  
They are typically indicated by prefix in their names with double score “\_\_” although we can still access the private members and functions using mangling, even though it is not recommended.

Here is the extended class :

```
class MyClass:

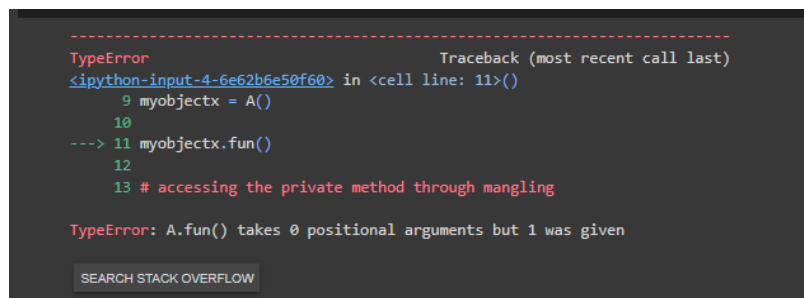
    def myfun(self):
        self.myvar = "blah" # NOTE: a per class-instance variable.
        print("calling the public function")
#private function is marked with __ in the name
    def __myfun(self):
        print("Calling the private function")

myobjectx = MyClass()

#accessing the public function
myobjectx.myfun()

# accessing the private method through mangling
myobjectx._MyClass__myfun()
```

- Explain “self” in python:
  - Self represents the instance of the class. By using self we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. [Geeksforgeeks](https://www.geeksforgeeks.org/python-self-keyword/)
- Let's say we messed up by forgetting the **self** in the parameter list, what can happen?
  - Without the self in the parameter list of a method inside a class, I encountered a type error as shown in the screenshot below. The error is “TypeError: A.fun() takes 0 positional arguments but 1 was given”



```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-4-6e62b6e50f60> in <cell line: 11>()
      9 myobjectx = A()
     10
--> 11 myobjectx.fun()
     12
     13 # accessing the private method through mangling

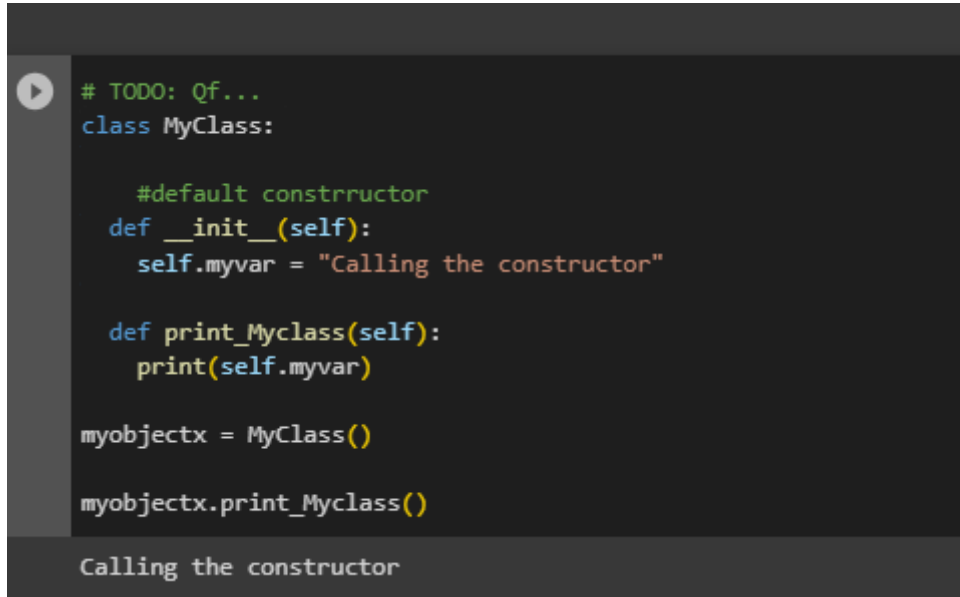
TypeError: A.fun() takes 0 positional arguments but 1 was given

SEARCH STACK OVERFLOW
```

Figure 7. TypeError message when forgetting the self in the parameter list.

## Qf) Extend the class with Constructor

- How to define and declare a constructor in python?
  - There are 2 types of constructors in python, default constructor and parameterized constructor. The default constructor as shown in the code below is a simple constructor which does not accept any arguments other than a reference to the instance being constructed.



```
# TODO: Qf...
class MyClass:

    #default constructor
    def __init__(self):
        self.myvar = "Calling the constructor"

    def print_Myclass(self):
        print(self.myvar)

myobjectx = MyClass()

myobjectx.print_Myclass()
```

Calling the constructor

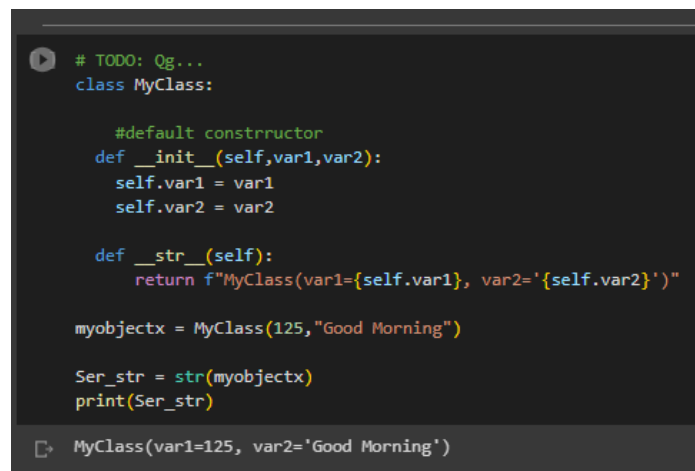
Figure 8. The extended class with a constructor

- How to define and declare a destructor in python ?
  - There isn't much need for destructors in python as much as in C++, because of Python garbage collector that handles memory management automatically. [Geeksforgeek](https://www.geeksforgeeks.org/python-destructor/)
  - Though `__del__` method is commonly used in python, it is called when all references to the object have been deleted.

## Qg) Extend the class with a to-string function

Figure 9. The extended class with a to-string function.

In the figure here, `__str__` method returns a string representation of the object, including its attribute values. This way we serialise the class to a string in a way similar to C++ ostream operator overloading.



```
# TODO: Qg...
class MyClass:

    #default constructor
    def __init__(self,var1,var2):
        self.var1 = var1
        self.var2 = var2

    def __str__(self):
        return f"MyClass(var1={self.var1}, var2='{self.var2}')"

myobjectx = MyClass(125,"Good Morning")

Ser_str = str(myobjectx)
print(Ser_str)
```

MyClass(var1=125, var2='Good Morning')

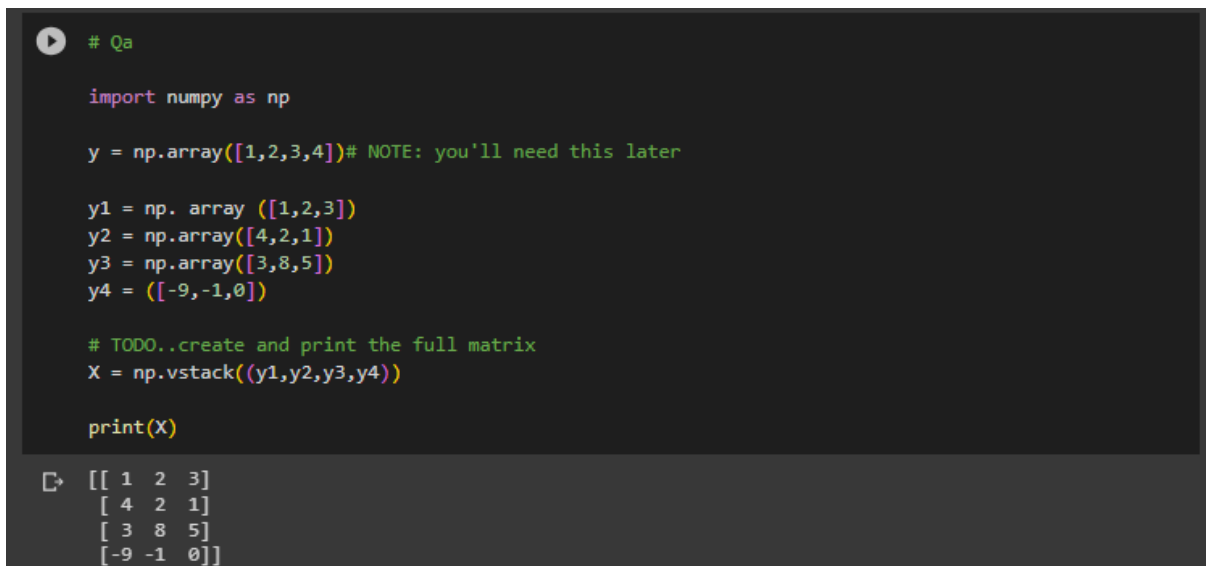


## L02/cost\_function.ipynb

In this exercise we are introduced to the cost function.

Qa) Given the following  $x(t)$ 's construct and print the X matrix in python

In the following figure is the code to print the X matrix in python using `np.vstack`



```
# Qa

import numpy as np

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

y1 = np. array ([1,2,3])
y2 = np.array([4,2,1])
y3 = np.array([3,8,5])
y4 = ([-9,-1,0])

# TODO..create and print the full matrix
X = np.vstack((y1,y2,y3,y4))

print(X)
```

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

Figure 10. Printing Matrix X.

Qb)Implement the L1 and L2 norms for vectors in python

**L1:**

```
def L1(vector):
    sum = 0.0
    for v in vector:
        sum += v if v >= 0 else -v
    dist = sum
    return dist
```

**L2:**

```
def L2(vector):
    sum = 0.0
    for v in vector:
        sum += v **2
    dist = (sum)** 0.5
```

```
return dist
```

## L2Dot:

```
def L2Dot(x,y):  
    diff = x - y  
    dot_product = np. dot ( diff .T, diff )  
    dot_product = math.sqrt ( dot_product )  
    return dot_product
```

```
# comment-in once your L2Dot fun is ready...  
d2dot=L2Dot(tx,ty)  
print("d2dot-expected_d2=",d2dot-expected_d2)  
assert math.fabs(d2dot-expected_d2)<eps, "L2Ddot dist seem to be wrong"  
print("OK(part-2)")
```

```
tx-ty=[-2  3 -1 -2], d1-expected_d1=0.0, d2-expected_d2=0.0  
OK(part-1)  
d2dot-expected_d2= 0.0  
OK(part-2)
```

Figure 11. The result for L1, L2 implementation

## Qc) Construct the root mean square error (rmse) function

“Root mean square error” measures the average difference between values predicted by a model and the actual values. [SAP](#)

```
def RMSE(x,y):  
    if (len(x) != len(y)):  
        raise Exception("Length dont match")  
    S_R = L2Dot(x,y)**2  
    rmse_value = ( S_R /len(y)) **0.5  
    return rmse_value
```

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

Figure 12. Output for RMSE test

## Qd) Similar Construct the Mean Absolute Error (MAE) & evaluate it

Mean Absolute Error refers to the magnitude of the difference between the prediction of an observation and the true value of the observations. [C3.ai](#)

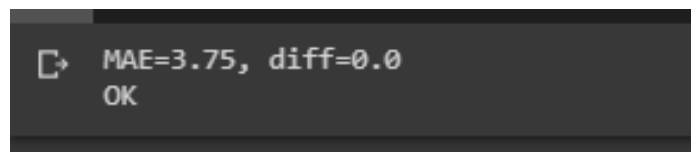
In broad terms, MAE tells us how well a model's predictions align with actual observations.

```
def MAE(x,y):
    if (len(x)!= len(y)):
        raise Exception("lengths dont match")
    mae_val = L1(x-y)/len(y)
    return mae_val

# 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"

print("OK")
```

A terminal window showing the output of the MAE test. The first line displays 'MAE=3.75, diff=0.0' and the second line displays 'OK'.

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

Figure 13. Output for MAE test

## Qe) robust code

To write a robust code, if statements, exceptions and asserts are used. I have already achieved that by having the following lines in my code specially RMSE and MAE.

```
assert math.fabs(r-expected)<eps, "your RMSE dist seems to be wrong"
assert math.fabs(r-expected)<eps, "MAE dist seems to be wrong"
#MAE and RMSE conditions
if (len(x)!= len(y)):
    raise Exception("lengths dont match")
```

## Qf) Conclusion

In the last exercises for L02/Cost\_function, the target is to establish a strong foundation in ML concepts. Norms and Error metrics are essential for understanding and evaluating models. As well as implementing norms from scratch, and constructing error metrics as **RMSE** and **MAE** to asset our model. The combination of math and coding skills prepare us to work effectively in machine learning, where both are essential for developing accurate models and robust softwares..

## L02/dummy\_Classifier.ipynb

In this exercise I will implement a dummy binary-classifier with fit-predict interface.

### Qa)Load and display the MNIST data

```
from sklearn.datasets import fetch_openml
%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")

def MNIST_GetDataSet():
    mnist = fetch_openml('mnist_784',version=1,
                        return_X_y=True, as_frame=False)
    return mnist

X,y = MNIST_GetDataSet()

MNIST_PlotDigit(X[0])
```

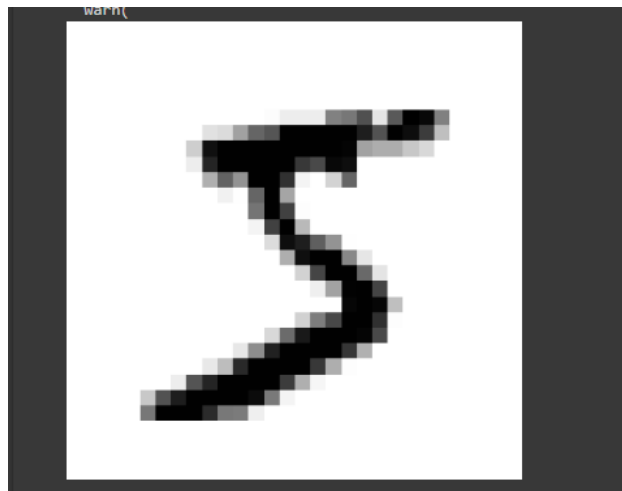


Figure 14. Plot of the first digit from the dataset

## Qb)Add a stochastic Gradient Decent SGD classifier

I added an SDG classifier as mentioned in the exercise, which can be used to estimate which number of the dataset reflects. Here the snippet code:

```
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
import matplotlib

X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000],
y[60000:]
y_train_5 = (y_train == '5') # True for all 5s, False for all other digits.
y_test_5 = (y_test == '5')
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)

print(f"X.shape={X.shape}") # print X.shape= (70000, 28, 28)
print (f" Predict={ sgd_clf . predict ( X_test )}")

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(f"X.shape={X.shape}") # X.shape= (70000, 784)

digit = X_test[15]
digit_image =digit.reshape(28,28)
plt.imshow((digit_image), cmap = matplotlib.cm.binary,
interpolation="nearest")
plt.axis("off")
plt.show()
```

The code above loads the MNIST dataset and prepares it for detecting the digit '5',and then trains the sdg classifier on the data. And we can see the output in the next figure.

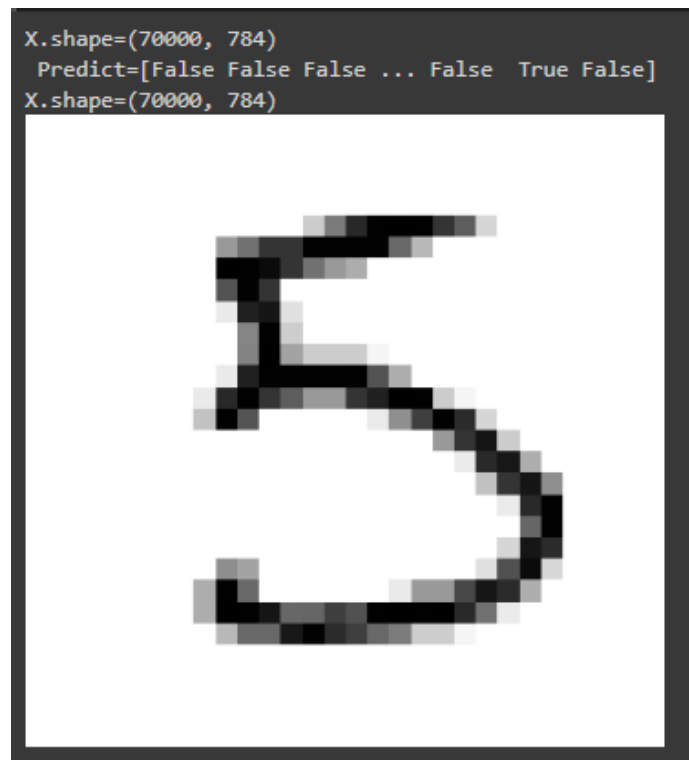


Figure 15. Output for the model prediction and plt.imshow

## Qc) Implement a dummy binary classifier

Following the instructions in HOML i have implemented the following classifier, to predict when it is not-5, by returning np.zeros() array of zeros with a **dtype** of **bool**. This means that this custom classifier predicts the negative class for all samples.

```
from sklearn.model_selection import cross_val_score
# TODO: add your code here..
import numpy as np
from sklearn.base import BaseEstimator

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

sgd_cvs= cross_val_score(sgd_clf,X_train,y_train_5,cv=3, scoring
="accuracy")
print (f"SGD Cross value Scor: {sgd_cvs}")

myclass = myClassifier()

myclass.fit(X_train,y_train_5)
myclass_cvs =
cross_val_score(myclass,X_train,y_train_5,cv=3,scoring="accuracy")

print(f"myclass Cross Value Score {myclass_cvs}")
```

As we can see in the figure below, the accuracy scores for both SGD classifier and my custom classifier are both above 90%, which is great, but still not optimal. The SGD scores are higher because it's more sophisticated than my custom classifier, plus that my classifier predicts the negative class which is 90% of the cases in the MNIST dataset.

"This demonstrates why accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets". [HOML]

```
SGD Cross value Scor: [0.95035 0.96035 0.9604 ]
myclass Cross Value Score [0.91125 0.90855 0.90915]
```

Figure 16. Cross value score output for both SDG and custom classifier

## Qd) Conclusion

In the last exercise we dived into the fundamentals of ML concepts. I started with loading and displaying the MNIST dataset and visualising a sample digit. Then I was introduced to the use of the SGD classifier to estimate a dataset whether it is true or false, based on our input. Also we have implemented a dummy classifier that is compared to a well trained classifier function SGD, and still, performed adequately. I learned that accuracy can be misleading for imbalance datasets, and the need for suitable evaluation metrics.

## L02/performance\_metrics.ipynb

In this exercise we dive into evaluating machine learning models using various metrics to assess their effectiveness and suitability for different tasks.

## Qa) implement the Accuracy function and test it on the MNIST data

**MyAccuracy** Implementation:

```
# TODO: Qa...
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_openml
from sklearn.linear_model import SGDClassifier
from sklearn.dummy import DummyClassifier
import sklearn.metrics as metric

def MyAccuracy(y_true, y_pred):
    # TODO: you impl here
    TP=0
    TN=0
    FP=0
    FN=0
    for x in range (len(y_true)):
        if y_true[x] == 1 and y_pred[x] == 1:
            TP += 1
        elif y_true[x]== 0 and y_pred[x] == 0 :
```

```

        TN += 1
    elif y_true[x]== 0 and y_pred[x] == 1 :
        FP += 1
    elif y_true[x]== 1 and y_pred[x] == 0 :
        FN += 1

MyAccuracyHelper = (TP + TN)/(TP+TN+FP+FN)
print(f"my accuracy helper:{MyAccuracyHelper}")

return MyAccuracyHelper

```

Testing the **Accuracy** from each model:

I evaluate the accuracy of both SGD classifier and a Dummy classifier of the MNIST dataset, then I compare the custom accuracy scores with scikit-learn '**accuracy\_score**'.

```

# Testing accuracy
X,y = fetch_openml('mnist_784',version=1, return_X_y=True, as_frame=False)
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000],
y[60000:]
y_train_5 = (y_train == '5') # True for all 5s, False for all other digits.
y_test_5 = (y_test == '5')

sgd_clf = SGDClassifier(random_state =42)
sgd_clf.fit(X_train,y_train_5)
pred = sgd_clf.predict(X_test)
print ("SGD classifier test : ")
TestAccuracy(y_test_5,pred)

acyScore_SGD = MyAccuracy(y_test_5,pred)
sklAcyScore_SGD = metric.accuracy_score(y_test_5,pred)
assert(sklAcyScore_SGD == acyScore_SGD)
print("SGD Classifier test passed")

print("\n")

dc = DummyClassifier()
dc.fit(X_train,y_train_5)
dc_pred = dc.predict(X_test)
print("Dummy Classifier test: ")

TestAccuracy(y_test_5,dc_pred)
acyScore_dc = MyAccuracy(y_test_5,dc_pred)
sklScore_dc = metric.accuracy_score(y_test_5,dc_pred)
assert (sklScore_dc == acyScore_dc)
print("Dummy classifier test passed")

```



```

/usr/local/lib/python3.10/dist-pack
warn(
SGD classifier test :
my accuracy helper:0.9492

my a          =0.9492
scikit-learn a=0.9492
my accuracy helper:0.9492
SGD Classifier test passed

Dummy Classifier test:
my accuracy helper:0.9108

my a          =0.9108
scikit-learn a=0.9108
my accuracy helper:0.9108
Dummy classifier test passed

```

Figure 17. Output results of the test.

From the figure 17 we can see the results that show that both the SGD classifier and the dummy Classifier reached similar accuracy scores, from both Scikit-learn **accuracy\_score** and the custom accuracy function '**MyAccuracy**'.

Qb) Implement precision, Recall and F1- score and test it on the MNIST data for both SGD and dummy classifier models

**MyPrecision:** the function below calculates precision, which is the ratio of true positive prediction to the sum of true positive and false positive predictions.

```

def MyPrecision(y_true, y_pred):
    # TODO: you impl here
    TP = 0
    FP = 0
    for x in range(len(y_true)):
        if y_true[x] == 1 and y_pred[x] == 1:
            TP += 1
        elif y_true[x] == 0 and y_pred[x] == 1:
            FP += 1
    if TP + FP != 0:
        precision = TP / (TP + FP)
    else:
        return 0
    return precision

```

**MyRecall:** calculates recall, which is the ratio of TP predictions to the sum of TP and FP predictions.

```

def MyRecall(y_true, y_pred):
    # TODO: you impl here

```

```

TP = 0
FP = 0
for x in range(len(y_true)):
    if y_true[x] == 1 and y_pred[x] == 1:
        TP += 1
    elif y_true[x] == 1 and y_pred[x] == 0:
        FP += 1
if TP + FP != 0:
    recall = TP / (TP + FP)
else:
    return 0
return recall

```

**MyF1Score:** calculates the F1-score, which is the mean of precision and recall.

```

def MyF1Score(y_true, y_pred):
    # TODO: you impl here
    p = MyPrecision(y_true, y_pred)
    r = MyRecall(y_true, y_pred)
    if (p+r != 0):
        F1_score = 2*(p*r)/(p+r)
        return F1_score
    else:
        return 0

```

**Dummy CClassifier and SGD classifier Tests:**

```

# TODO: your test code here!
assert(MyPrecision(y_test_5, dc_pred) == metric.precision_score(y_test_5, dc_pred))
print("precision test passed")
assert(MyRecall(y_test_5, dc_pred) == metric.recall_score(y_test_5, dc_pred))
print("MyRecall test passed")
assert(MyF1Score(y_test_5, dc_pred) == metric.f1_score(y_test_5, dc_pred))
print("MyF1Score test passed")

#SGD classifier
assert(MyPrecision(y_test_5, pred) == metric.precision_score(y_test_5, pred))
print("SGD precision test passed")
assert(MyRecall(y_test_5, pred) == metric.recall_score(y_test_5, pred))
print("SGD MyRecall test passed")
assert(MyF1Score(y_test_5, pred) == metric.f1_score(y_test_5, pred))
print("SGD MyF1Score test passed")

```

We can see in the figure below that the custom functions are correctly implemented and produce results consistent with scikit-learn functions.

```
/usr/local/lib/python3.10/dist
_warn_prf(average, modifier
precision test passed
MyRecall test passed
MyF1Score test passed
SGD precision test passed
SGD MyRecall test passed
SGD MyF1Score test passed
```

Figure 18. Test output for SGD and Dummy custom classifier

## Qc) the confusion matrix

```
from sklearn.metrics import confusion_matrix
# TODO: Qcc
# correctly implemented
dc1_cm = confusion_matrix(y_test_5,dc_pred)
print (f"Dummy classifier: \n {dc1_cm}")
SGD1_cm = confusion_matrix(y_test_5,pred)
print (f"SGD classifier:\n {SGD1_cm}")
print ("-----")

# Wrong implementation
dc2_cm = confusion_matrix(dc_pred,y_test_5)
print (f"Dummy classifier:\n {dc2_cm}")
SGD2_cm = confusion_matrix(pred,y_test_5)
print (f"SGD classifier:\n {SGD2_cm}")
```

From this output in figure 18 we can see the confusion matrix for both SGD and the dummy classifier we made earlier. When confusion\_matrix parameters are incorrectly implemented, we can see that the function returns the T transpose of the first matrix, which swaps the roles of predicted and actual classes, so the counts are arranged differently, but the information is the same.

Figure 19. Output for the confusion matrix

```
Dummy classifier:
[[9108  0]
 [ 892  0]]
SGD classifier:
[[8707  401]
 [ 107  785]]
-----
Dummy classifier:
[[9108  892]
 [  0   0]]
SGD classifier:
[[8707  107]
 [ 401  785]]
```

## Qd) A confusion matrix Heat-map

To Generate a heatmap image for the confusion matrices, I decided to import the seaborn library and matplotlib to generate an easier heatmap.

### Dummy Classifier

```
import matplotlib.pyplot as plt

import seaborn as sns
import numpy as np

plt.figure(figsize=(8,6))
sns.heatmap(dc1_cm,annot=True, fmt='', cmap='Blues')
plt.xlabel('predicted labels')
plt.ylabel('True labels')
plt.title('confusion Matrix for Dummy classifier')
plt.show()
```

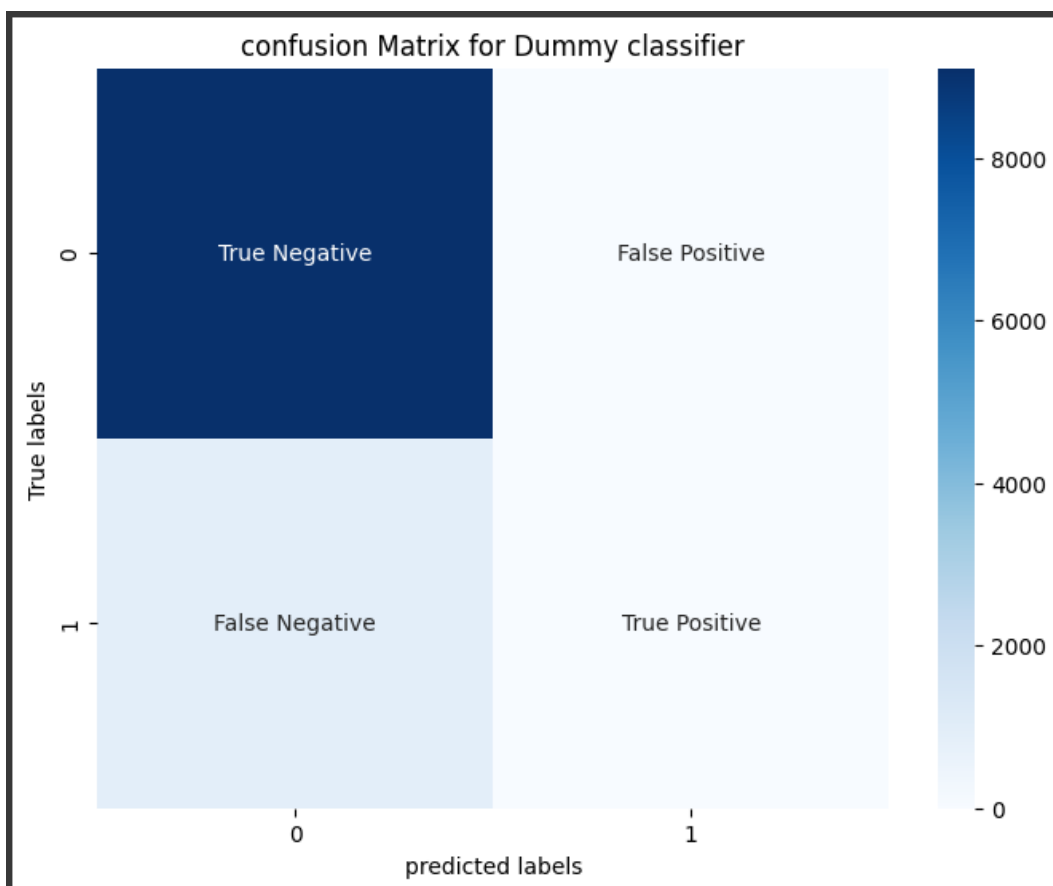


Figure 20. COnfusion matrix for the dummy classifier

### SGD Classifier:

```
plt.figure(figsize=(8,6))
sns.heatmap(SGD1_cm,annot=True, fmt='', cmap='Blues')
plt.xlabel('predicted labels')
plt.ylabel('True labels')
plt.title('confusion Matrix for SGD classifier')
plt.show()
```

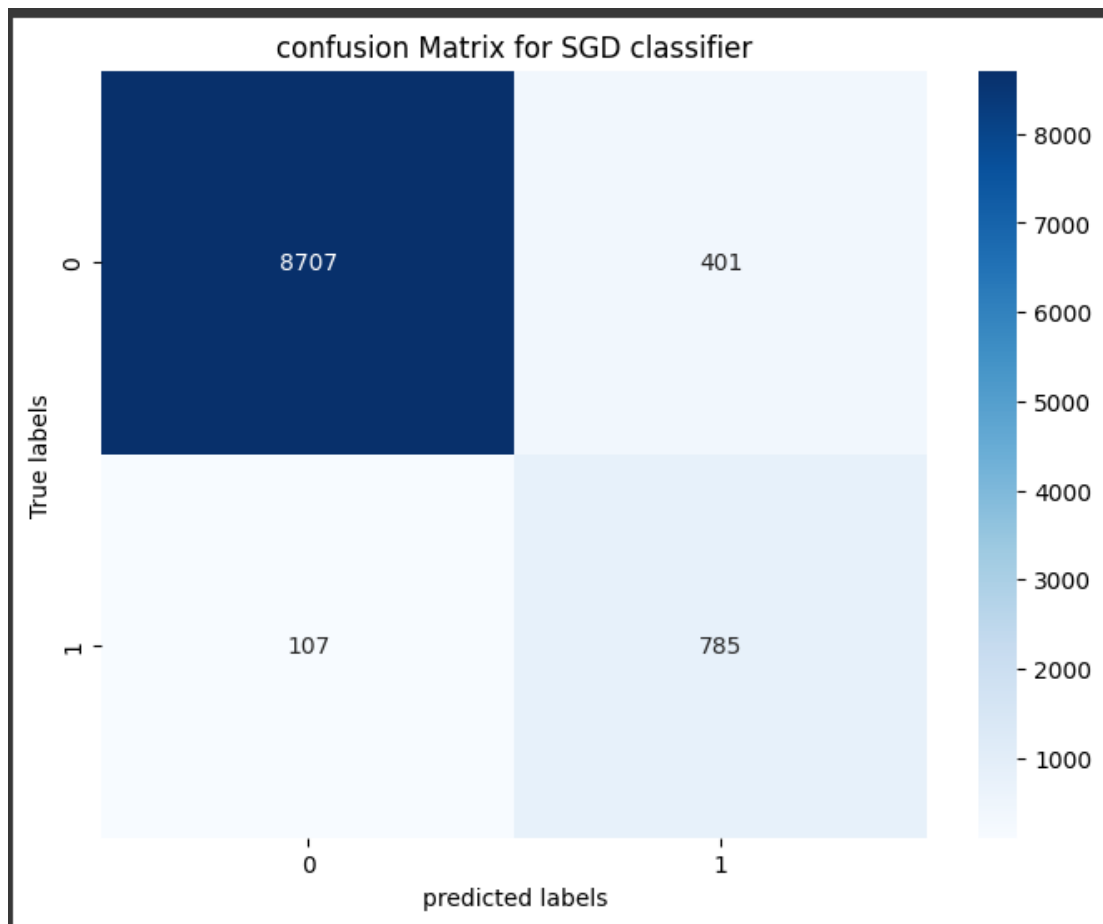


Figure 21. Confusion matrix for SGDClassifier

## Qe) Conclusion

I have learned in the last exercise about performance metrics that are commonly used in ML, such as accuracy, precision, recall, and the F1 score. I have also been introduced to the concept of confusion matrix, this matrix helps analyse classification results in detail. I also compared the classification of SGD (Stochastic Gradient Decent) classifier and custom classifier that I made in this exercise which was very helpful in understanding how different classifiers work.