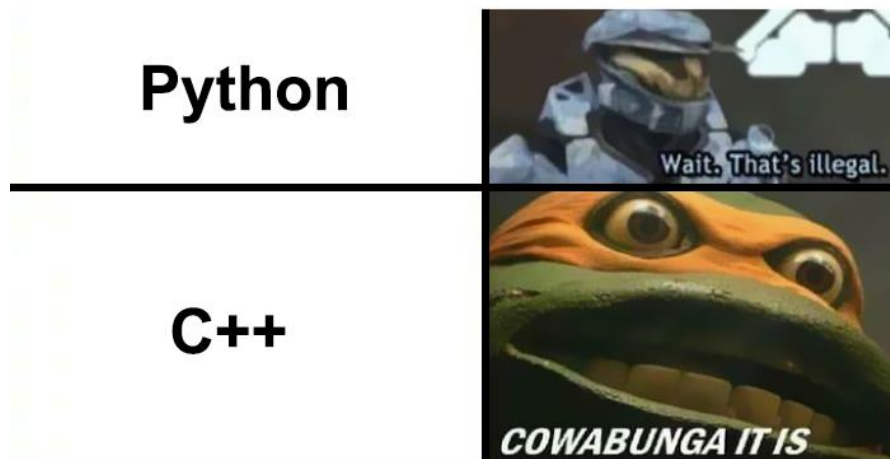


O1

## SWMAL-01 GROUP 13

When I code some stupid stuff that  
will obviously break my computer:



### *Group members*

Christoffer Regnar Mølck - 202009347  
Martin Teit Bruun Andersen - 202109592  
Morten Kierkegaard Erichsen - 201807571

5. marts 2024

# 1 INTRO L01

## 1.0.1 Qa) the $\Phi$ and the $R^2$ score

Code:

Previous to this code, a linear regression model called: "model" was made with sklearn, trained with the X and y data, and then run. This gave the possibility to extract the "score", "coef" and intercept.

```
1  
2 model.score(X,y), model.coef_, model.intercept_
```

Which outputs:

```
1 R^2: 0.734  
2 Coefficient: 4.91e-5  
3 Intercept: 4.85
```

The code grabs the information from the model, information being "score" ( $R^2$ ) given the X and y data, coefficient being the slope in a linear line  $F = Ax + b$  <- A here. Intercept being b.

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

$$R^2 = [-\infty:1]$$

Is it best to have a low  $R^2$  score or a high  $R^2$  score? Better to have a high score.

This means, is  $R^2$  a loss/cost function or a function that measures of fitness/goodness? It is a function that returns a coefficient of determination. Meaning it is a fitness score, the better the fit, the closer to 1  $R^2$  goes.

The  $R^2$  value describes how well the data is fitted, 1 being best (albeit unrealistic) The higher the better  $R^2$ , measuring goodness of fit Coefficient parameter is the slope, and intercept is, well, the intercept with the y-axis

## 1.0.2 Qb) Using k-Nearest neighbors

```

1
2 # Select and train a model
3 # TODO: add your code here..
4 import sklearn.neighbors
5
6 #assert False, "TODO: add you instatiation and training of the knn model here.."
7 k_model = sklearn.neighbors.KNeighborsRegressor(3)
8
9 k_model.fit(X,y)
10
11 pred = [[22587]]
12 k_y_pred = k_model.predict(pred)
13
14 print(f"\nKnn value = {k_y_pred}")
15 k_model.score(X,y)

```

Which outputs:

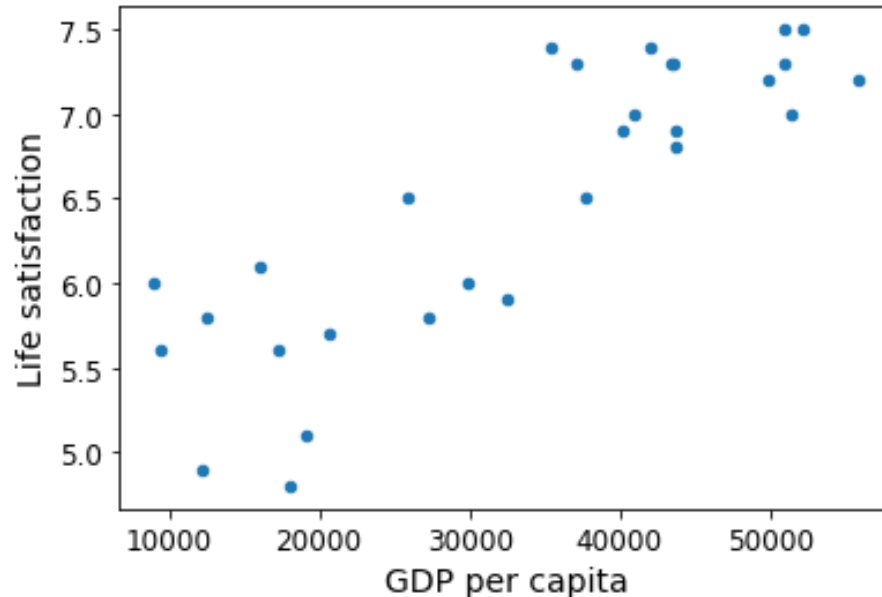


Figure 1.1: Life satisfaction as a function of GDP pr capita

And:

```

1 Knn value: 5.766
2 k score: 0.85

```

## Group 13

What do the k-nearest neighbors (KNN) estimate for Cyprus, compared to the linear regression (it should yield=5.77)? It returns the mean of the n-neighbors (here  $n = 3$ ), and it estimates the mean life satisfaction given a gdp of 22587usd.

What score-method does the KNN model use, and is it comparable to the linear regression model? K model:  $R^2$  Linear model:  $R^2$  Can be compared, since they use the same scoring method

Seek out the documentation in Scikit-learn, if the scoring methods are not equal, can they be compared to each other at all then? They cannot be directly compared, if the scoring methods are different from each other, but, gaining an understanding of how they score, should they differ, gives a foundation for comparison.

### 1.0.3 Qc) Tuning Parameter for k-Nearest Neighbors and A Sanity Check

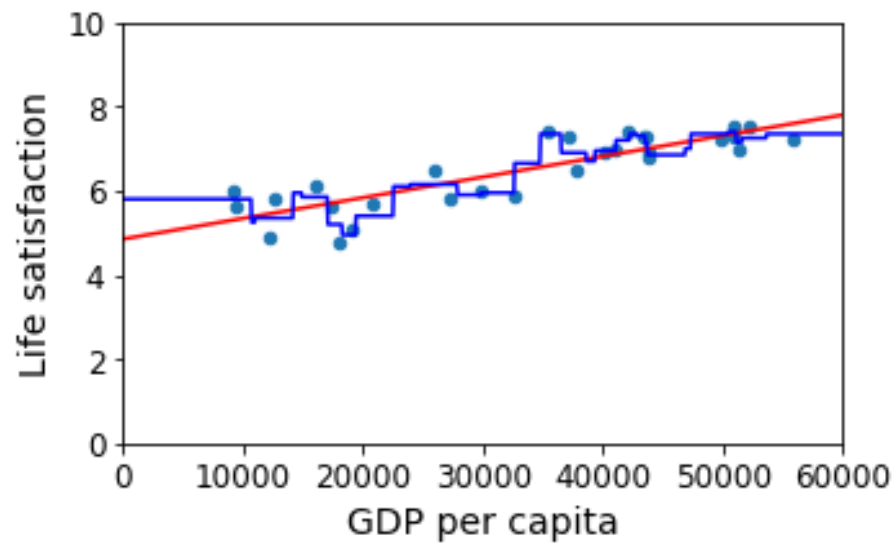
Code:

```

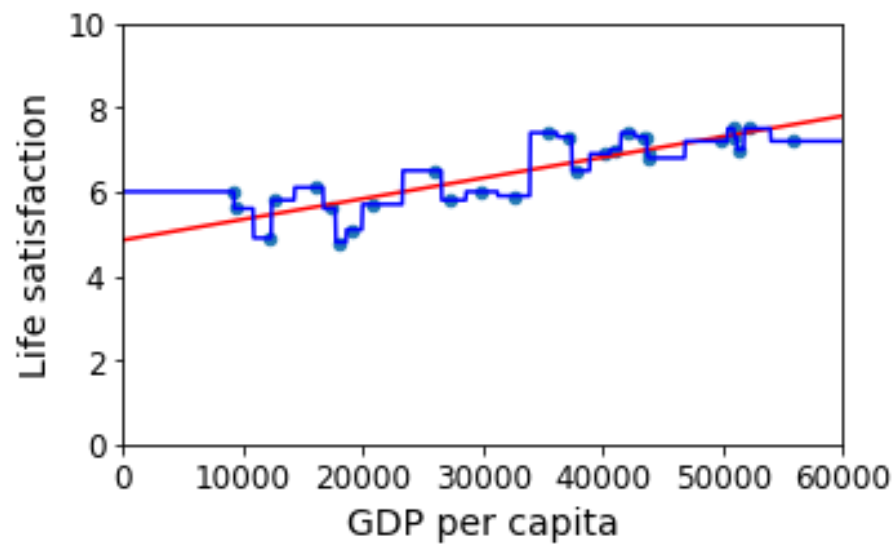
1  # TODO: add your code here..
2  # assert False, "TODO: try knn with different k_neighbor params, that is
    ↳ re-instantiate knn, refit and replot.."
3  k_model = sklearn.neighbors.KNeighborsRegressor(2)
4
5  k_model.fit(X,y)
6
7  pred = [[22587]]
8  k_y_pred = k_model.predict(pred)
9
10
11 # from this test M data, predict the y values via the lin.reg. and k-nearest
    ↳ models
12 y_pred_lin = model.predict(M)
13 y_pred_knn2 = k_model.predict(M) # ASSUMING the variable name 'knn' of your
    ↳ KNeighborsRegressor
14
15 # use plt.plot to plot x-y into the sample_data plot..
16 plt.plot(m, y_pred_lin, "r")
17 plt.plot(m, y_pred_knn, "b")
18
19 k_model.score(X,y)

```

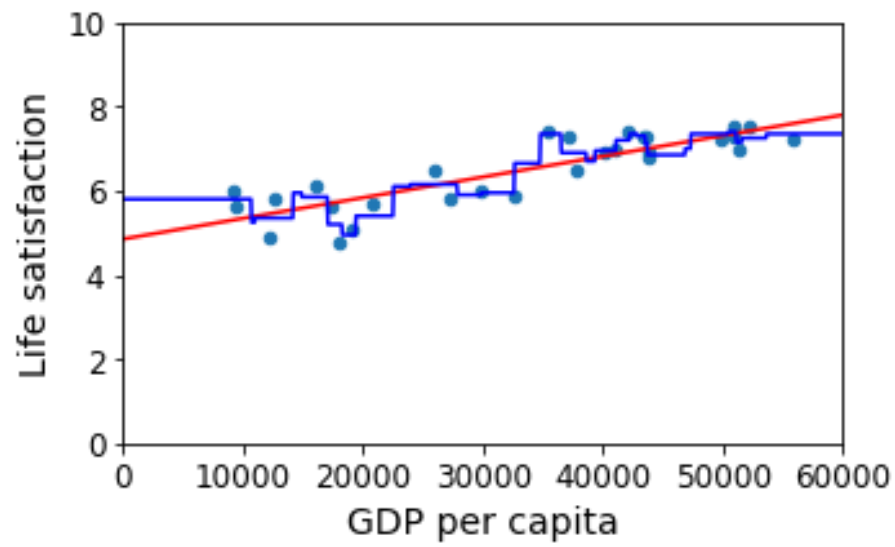
Which outputs for kneighbors: 1,2,3 and 15:



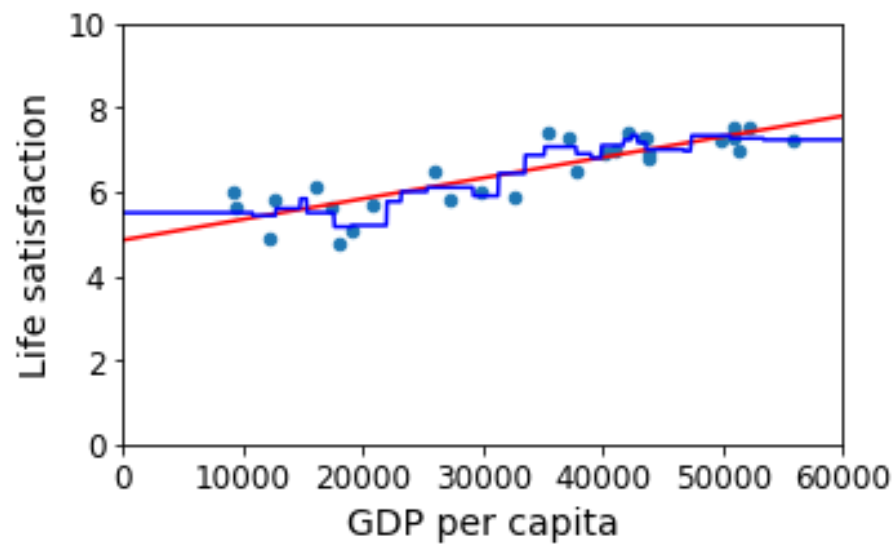
Figur 1.2: k neighbors = 1



Figur 1.3: k neighbors = 2



Figur 1.4: k neighbors = 3



Figur 1.5: k neighbors = 15

```
1 k model score: 1 (for k neighbors = 1)
2 k model score: 0.909 (for k neighbors = 2)
3 k model score: 0.852 (for k neighbors = 3)
4 k model score: 0.715 (for k neighbors = 15)
```

In conclusion, the performance of `sklearn.neighbors.KNeighborsRegressor(n)` varies significantly depending on the choice of parameter `n`. For `n=1`, the model tends to overfit the data, resulting in high variance and poor generalization to unseen data. When `n=2` or `n=3`, the model's performance improves as it begins to capture more general trends in the data, but there might still be some overfitting. However, with `n=15`, the model achieves a good balance between bias and variance, leading to better generalization and more reliable predictions. Therefore, the selection of `n` should be carefully considered based on the specific dataset and the desired trade-off between bias and variance.

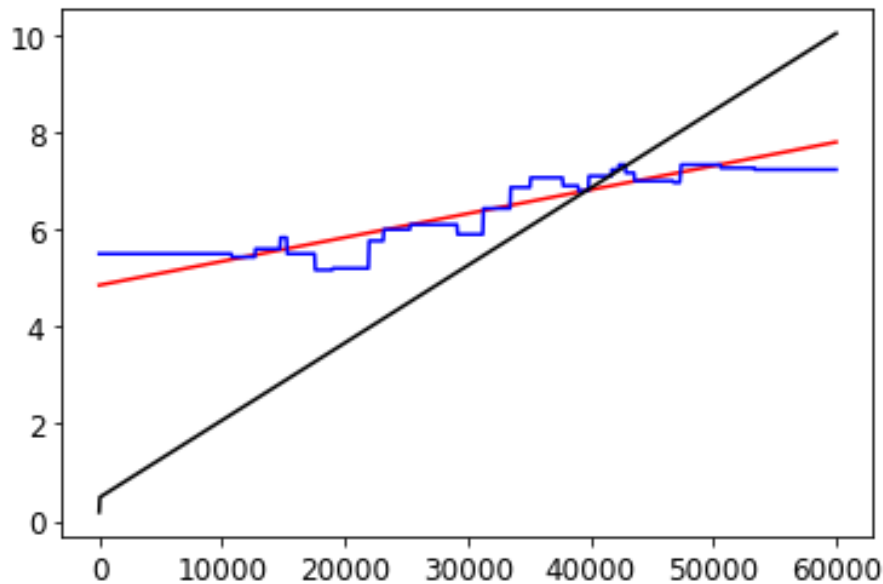


### 1.0.4 Qd) Trying out a neural network

Code:

```
1  from sklearn.neural_network import MLPRegressor
2
3  # Setup MLPRegressor
4  mlp = MLPRegressor( hidden_layer_sizes=(10,), solver='adam', activation='relu',
5                      ↪ tol=1E-5, max_iter=100000, verbose=True)
6  mlp.fit(X, y.ravel())
7
8  # lets make a MLP regressor prediction and redo the plots
9  y_pred_mlp = mlp.predict(M)
10
11 plt.plot(m, y_pred_lin, "r")
12 plt.plot(m, y_pred_knn, "b")
13 plt.plot(m, y_pred_mlp, "k")
14
15 # TODO: add your code here..
16 #assert False, "TODO: predict value for Cyprus and fetch the score() from the
17 ↪ fitting."
18 pred = [[22875]]
19 y_pred = mlp.predict(pred)
20
21 y_pred, mlp.score(X, y)
```

Output:



Figur 1.6: MLPRegressor comparison with linear and KNN-scores

Throughout 718 iterations the loss was reduced from  $J = 290434295$  to  $J = 1.590$ , resulting in a predicted life satisfaction score for Cyprus, and score of:

1	Life satisfaction prediction: 4.1
2	MLP score: -3.67

With a GDP predicting value of 22875

Can the ‘MLPRegressor’ score function be compared with the linear and KNN-scores?  
Yes, since it also returns an  $R^2$  coefficient of determination.

### 1.0.5 [OPTIONAL] Qe) Neural network with pre-scaling

Code:

```

1      # TODO: add your code here..
2      #assert False, "TODO: try prescale data for the MPL...any better?"
3
4      from sklearn import preprocessing
5      import numpy as np
6
7      # X transformation
8      min_max_scaler = preprocessing.MinMaxScaler([0,1])
9      X_new = min_max_scaler.fit_transform(X)
10
11     mlp = MLPRegressor( hidden_layer_sizes=(10,), solver='adam', activation='relu',
12         ↪ tol=1E-5, max_iter=100000, verbose=True)
13     mlp.fit(X_new, y.ravel())
14
15     #
16     # create an test matrix M, with the same dimensionality as X, and in the range
17     ↪ [0;1]
18     # and a step size of your choice
19     m=np.linspace(0, 1, 1000)
20     M=np.empty([m.shape[0],1])
21     M[:,0]=m
22
23     # lets make a MLP regressor prediction and redo the plots
24     y_pred_mlp = mlp.predict(M)
25
26     plt.plot(m, y_pred_lin, "r")
27     plt.plot(m, y_pred_knn, "b")
28     plt.plot(m, y_pred_mlp, "k")
29
30     # TODO: add your code here..
31     #assert False, "TODO: predict value for Cyprus and fetch the score() from the
32     ↪ fitting."
33     pred = [[0.22587]]
34     y_pred = mlp.predict(pred)
35
36     y_pred, mlp.score(X_new, y)
37
38     # With the scaled input, it is much better

```

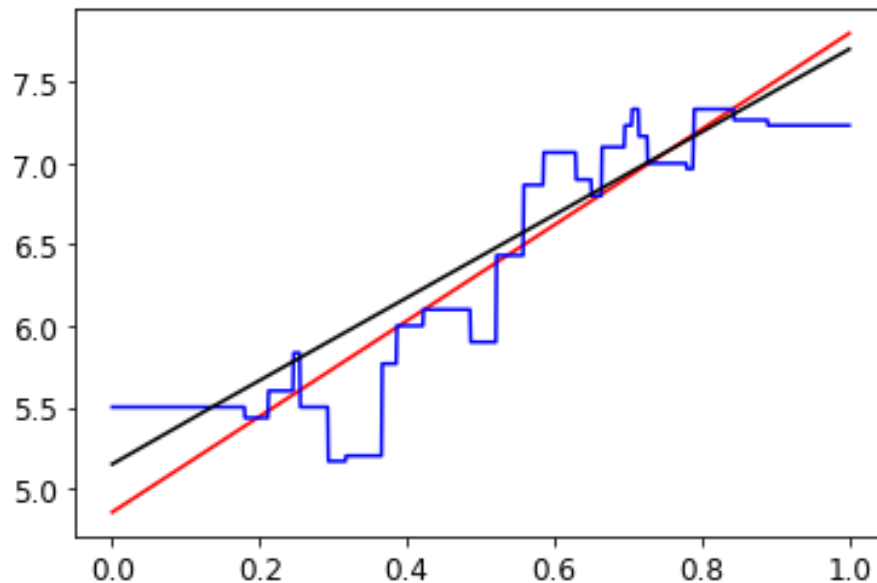


Figure 1.7: MLPRegressor with [0:1] scaling

After 2792 iterations the loss was reduced from  $J = 16.53$  to  $J = 0.00973$  (much better than the nonscaled version). Since the neurons in the neural net expects/better handles values between [0:1], meaning it doesnt saturate as often/easily as the unscaled version.

This gives a more accurate prediction and score, of:

Output:

1	Life satisfaction prediction: 5.73
2	MLP score: 0.725

## 1.0.6 Intro L01 conclusion

ChatGPT conclusion:

"LinearRegression() is good for simple, straight relationships between features and targets. KNeighborsRegressor() is like a nosy neighbor, it's good for picking up on patterns but can get confused easily with too much noise. MLPRegressor() is like a multi-tool, it can handle complex relationships but might need a lot of tinkering to work just right."

The conclusion that ChatGPT wrote makes a lot of sense, especially MLPregressor with unscaled training input. After scaling, the predictions were much better. The same with KNN, it can easily underfit/overfit, meaning that some tinkering with the kneighbors value

is necessary. Linear regression is linear regression - good for data that has a somewhat apparent linear relationship.

## 2 MODULE AND CLASSES

### 2.1 Qa Load and test the 'libitmal' module

The group uses a mixture of IDE's. Here the choosen IDE's are Visual Studio Code and JetBrains DataSpell. This meanings that the need to execute the "declare -x PYTHON-PATH..."is not needed. However, when trying to access a module that is not placed within the opened folder of the IDE's explorer, it is needed to access it through a PATH call, as presented in the code snippet below.

For Question A the provided code snippet was inserted into the Jupyter code-cell and executed cording to 'libitmal's directory PATH. Here the group have decided to create a shared Git-Depository for the course SWMAL-01 called MLKursus.

```
1 import sys,os
2 sys.path.append(R'..\UndervisningsGit\GITMAL')
3 from libitmal import utils as itmalutils
4 itmalutils.TestAll()
5 print(dir(itmalutils))
6 print(itmalutils.__file__)
```

The print statement output was the following:

```
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
['AssertInRange', 'CheckFloat', 'DToXy', 'GenerateConfusionMatrix', 'GenerateResults',
 → 'InRange', 'Iterable', 'ListToMatrix', 'ListToVector', 'PrintMatrix', 'ResetRandom',
 → 'ShowResult', 'TEST', 'TestAll', 'TestCheckFloat', 'TestPrintMatrix', 'TestVarName',
 → 'VarName', 'XyToD', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
 → '__name__', '__package__', '__spec__', '__warningregistry__', 'ctxlib', 'inf',
 → 'inspect', 'isFloat', 'isList', 'isNumpyArray', 'nan', 'np', 'random', 're', 'sklearn']
c:\Users\45201\OneDrive\Uni\6_Semester\ML\MLKursus\Martin\..\UndervisningsGit\GITMAL\libitmal\utils.py
```

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

For Question B a module was created containing two functions. The module was created within a sub-folder for easier access. Here the sub-folder got called O1 and module O2. The module contains a Python file called 'Test\_Module'.

'Test\_Module' contains the following code:

```
1 def test():
2     print("Hello this is a test, and you passed")
3
4 def AOR(Lenght, Width):
5     return Lenght * Width
6
7 def HappyDog():
8     print("""
9     /  \__
10    ( @\__
11    /  0
12   / (____/
13  /_____/ U
14     """)
```

Here the executing code for the inclusion of the module looks like the following:

```
1 import sys,os
2 sys.path.append(R'C:\Users\45201\OneDrive\Uni\6_Semester\ML\MLkursus\Martin\O1')
3 from O2_Qb import Test_Module as QB
4
5 print(dir(QB))
6 print(QB.__file__)
7
8 print()
9
10 # Testing function AOR in module
11 ## Function should return area of a square or rectangle
12 A = 12
13 B = 8
14 print(QB.AOR(A,B))
15
16 print()
17
18 # Tesing function HappyDog
19 ## Function should return a good boi
```

```
20 print(QB.HappyDog())
```

Running the code-cell from the Jupyter Notebook returns the following output:

```
['AOR', 'HappyDog', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
  ↳ '__name__', '__package__', '__spec__', 'test']
C:\Users\45201\OneDrive\Uni\6_Semester\ML\MLKursus\Martin\01\02_Qb\Test_Module.py

96

  / \__
 ( @\___
 / 0
 / (____/
 /_____/ U

None
```

Since the output matches the wanted function setup in the 'Test\_Module' the inclusion of our module is working.

## 2.3 Qc How do you 'recompile' a module?

There is a built-in reload function in the "importlib-package. Calling the `importlib.reload(#ModuleName)` will recall the module and recall the latest version.

```
1 ## For our created modules it could be called like so:
2 import importlib
3
4 importlib.reload(QB)
```

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

### 2.4.1 How are private function and member variables represented in python classes?

In python functions can be made "private" by prefixing them with a underscore. Forexample in a class a single underscore in front of the function name indicates that the function is



intended for internal use only and should not be accessed outside of the class. Another use of underscores could be a double underscore in front and after the function name. This serves as a signal for developers that the variable is not part of the public interface but should rather be treated as a private function within the class.

### 2.4.2 What is the meaning of 'self' in python classes?

The self instance refers to the class it self. This is usually done through an "\_\_init\_\_" function that takes self as first parameter with other parameters referring to the class. This makes it possible to modify the input object's properties and create scenarios that are unique to the input.

### 2.4.3 What happens to a function inside a class if you forget 'self' in the parameter list?

In the copy-paste of the questions code snippet will initiate it 'self'. Here the variable myobjectx will have myfun() functions print statement printed in the output "box" when runned:

```
1 class MyClass:
2
3     def myfun(self):
4         self.myvar = "blah" # NOTE: a per class-instance variable.
5         print(f"This is a message inside the class, myvar={self.myvar}.")
6
7 myobjectx = MyClass()
8
9 myobjectx.myfun()
```

With output:

```
This is a message inside the class, myvar=blah.
```

When the 'self' instance is not included within a class function like so:

```
1 class MyClass:
2
3     def myfun():
4         self.myvar = "blah" # NOTE: a per class-instance variable.
5         print(f"This is a message inside the class, myvar={self.myvar}.")
6
```

```

7 myobjectx = MyClass()
8
9 myobjectx.myfun()

```

The output "box" will return a TypeError:

```

-----
TypeError Traceback (most recent call last)
Cell In[41], line 46
      42 print(f"This is a message inside the class, myvar={self.myvar}.")
      44 myobjectx = MyClass()
--> 46 myobjectx.myfun()

TypeError: MyClass_1.myfun() takes 0 positional arguments but 1 was given

```

The outcome makes sense, since variable 'myvar' has properties that require the 'self' instance.

## 2.5 Qf Extend the class with a Constructor

### 2.5.1 Figure a way to declare/define a constructor (CTOR) in a python class

A constructor is (can be) created to initialize the objects of a class when created. This can be done the following way:

```

1 class university:
2     def __init__(self, student, major):
3         self.student = student
4         self.major = major
5
6     def reg(self):
7         print(f"{self.student} is currently on his third redelivery. He has such
            ↳ high hopes that third time is a charm as they say. He really does
            ↳ love {self.major}.")
8
9 x = university("Alex", "SWMAL")
10 x.reg()
11
12 # Can also be called like so:
13 # university('Alex', 'Math').reg()

```

## 2.6. QG EXTEND THE CLASS WITH A TO-STRING FUNCTION Group 13

With output:

```
Alex is currently on his third redelivery. He has such high hopes that third time is a charm as  
    ↪ they say. He really does love SWMAL.
```

In the above code snippet the constructor is created within the 'init' function so the rest of the functions can inherit and be used. However, when calling the class, the call needs to include the variables as shown above.

### 2.5.2 Is there a class destructor in python (DTOR)?

Python does not contain a class destructor as so. If for example a file-handler was made, and within the class a function was made that closes the file, then a deconstructor would have been made.

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

We are asked to serialize a class, to make it printable and have some kind of meaning behind the return value. In python we can use the `__str__` built in function to define a return string for a class when called with print, like so:

```
1 class myclass:
2     value1 = None
3
4     def __init__(self, val1):
5         self.value1 = val1
6
7     def __str__(self):
8         return f"I am a class of type {self.__class__.__name__}, i have the
9             ↪ variable 1 of value: {self.value1}"
10
11 myclass1 = myclass(10)
12 myclass2 = myclass(20)
13
14 print(myclass1)
15 print(myclass2)
```

this outputs:

```
I am a class of type myclass, i have the variable 1 of value: 10
```

## 2.6. QG EXTEND THE CLASS WITH A TO-STRING FUNCTION **Group 13**

```
I am a class of type myclass, i have the varaible 1 of value: 20
```

Note that the `self.__class__.__name__` is kinda dirty and ugly but functional.

## 3 COST FUNCTION

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

```
1 # TODO..create and print the full matrix
2
3 X = np.array([[1,2,3],[4,2,1],[3,8,5],[-9,-1,0]])
4
5 print(f"Matrix x:\n {X}")
```

Output:

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

### 3.0.2 Qb Implement the $L^1$ and $L^2$ norms for vectors in python.

The np.dot operator has now been used instead of the np.linalg.norm, as specified in Qb. The result is the same.

In the below code, low-level definition of the functions has been made, including assertion errors, should different requirements not be met.

Code:

```
1      # TODO: solve Qb...implement the L1, L2 and L2Dot functions...
2 def L1(vec):
3     if len(vec) == 1:
4         raise AssertionError("Must be a vector, not a scalar!")
5
6     v0 = 0
7     for v in vec:
8         if v < 0:
9             v = -v
10        v0 += v
11
12    # return sum(abs(vec))
13    return v0
14
```

```

15 def L2(vec):
16     if vec.ndim != 1:
17         raise AssertionError("Vector must be 1-dimensional!")
18     v0 = 0
19     for v in vec:
20         v0 += (v**2)
21
22     v0 = v0**0.5
23     # return (sum(vec**2))**0.5
24     return v0
25
26 def L2Dot(vec):
27     if not isinstance(vec, np.ndarray):
28         raise ValueError("Wrong input type, must be numpy.ndarray")
29     #return np.linalg.norm
30     return np.dot(vec.T,vec)**0.5

```

Output:

```

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

```

### 3.0.3 Qc Construct the Root Mean Square Error (RMSE) function (Equation 2-1 [HOML]).

In the below code, low-level definition of the functions has been made, including assertion errors, should different requirements not be met. The equation does not look 1:1 as in HOML, which is because it uses L2, that already sums the values.

Code:

```

1
2 def RMSE(X,y):
3     if not X.shape == y.shape:
4         raise ValueError(f"The dimensions of inputs are incorrect.\nX has shape:
5             ↪ {X.shape}, while y has shape: {y.shape}")
6
7     m = len(X)
8
9     dvar = (1/m)**0.5 * L2(X-y)

```

```

10 RMSE = dvar
11 return RMSE

```

Output:

```

1 RMSE=6.576473218982953, diff=2.6645352591003757e-15
2 OK

```

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

Same description as Qc, with eq 2-2 and L1 instead.

```

1 def MAE(X,y):
2     for i in range(len(X)):
3         if isinstance(X[i],np.complex128):
4             raise ValueError("Complex number spotted! \nOnly real numbers
5                                     ↪ please!")
6
7     if X.shape != y.shape:
8         raise ValueError("X and y must have the same shape!")
9
10    m = len(X)
11    print(type(X))
12
13    MAE = 1/m * L1(X-y)
14
15    return MAE

```

Output:

```

1 MAE=3.75, diff=0.0
2 OK

```

### 3.0.5 Qe Robust Code

This has been implemented in all functions, after having made them without the error-checking

### 3.0.6 Qf Conclusion

Given the design matrix  $X$  at the beginning, we learned that it was important to fully understand the nomenclature behind how it is designed. Meaning that a row vector is "disguised" as a column vector in the design matrix. We had trouble in the beginning, thinking that we actually had to transpose the  $x(1)$ ,  $x(2)$ ... and ended with a  $3 \times 4$  matrix, instead of the correct  $4 \times 3$  matrix.

The first exercise was good at testing how to write simple functions in a low-level way, better understanding the coding aspect.

The MSE/RMSE function gave us a good understanding of what it means, when we want to reduce the loss function, since relating it to linear algebra in terms of relating it to minimizing a distance makes great sense. I can't say that we feel as if we grasp the concept fully, but it makes more and more sense.

Comparing the MAE and RMSE it seems that the RMSE is less robust than MAE, since its value is more influenced by squares and roots.

The errorhandling exercise was fine, since it gave a better understanding of the functions, making sure that the inputs correspond to what we want the function to do. For example, we got to the exercise with the incorrect  $3 \times 4$  design matrix, which the function used just fine, but gave an incorrect result. But when raising the check for equal dimensionality between  $h(X)$  and  $y$ , it became clear that the design matrix was incorrect.

All in all, good exercises, though the errorhandling was a bit tedious (but good to learn and use in the future nonetheless).



## 4 DUMMY CLASSIFIER

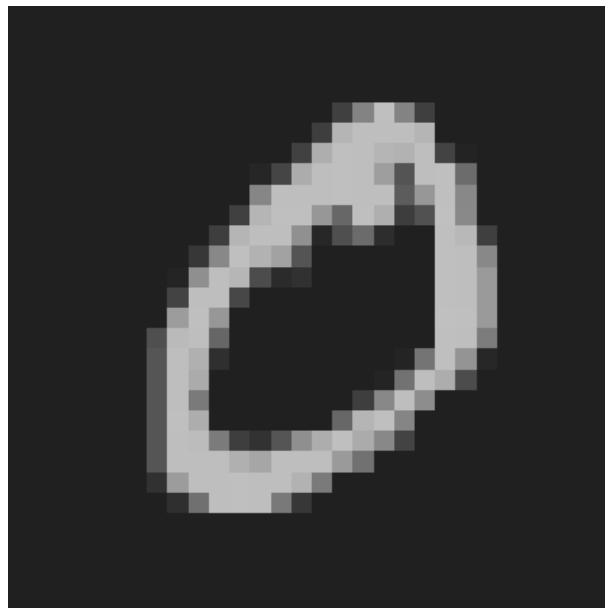
### 4.1 Qa Load and display the MNIST data

we are asked to build a function that gets the data, and then to plot a single digit of the data:

```
1 def MNIST_GetDataSet():
2     Data, Target = fetch_openml(name="mnist_784",return_X_y=True,
3     ↪ as_frame=False)
4     Data = Data / 255 #Converts to binary
5     return Data,Target
```

then we can create an image from the binary data using the provided MNIST\_PlotDigit function called with the data[i] at some index i.

given  $i = 1$  we get an image of a zero:



### 4.2 Qb Add a Stochastic Gradient Decent [SGD] Classifier

we are asked to use the SGD classifier from sklearn our data, and then display a correct number using the plotDigit function.

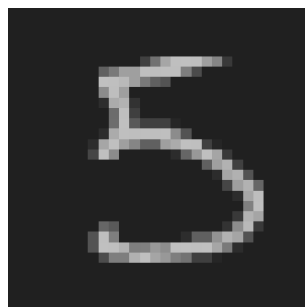
```
1 X,y = MNIST_GetDataSet()
```

```

2
3 #y = y.astype(np.uint8)
4
5 """
6 Below our training and test data is trained on the first 10000 entries and
7   ↪ tested on the last 10000.
8 """
9
10 X_train, X_Test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
11
12 y_train_5 = (y_train== '5')
13 y_test_5 = (y_test== '5')
14
15 sgd_clf = SGDClassifier(random_state=42)
16 sgd_clf.fit(X_train,y_train_5)
17 Arr = sgd_clf.predict(X_Test)
18
19 # Defines a function to read through a boolean array and indicate which
20   ↪ location returns a "True" value.
21
22 def True_seeker(arr):
23     transitions = []
24     for i, val in enumerate(arr):
25         if val:
26             transitions.append(i)
27     return transitions

```

here our truth seeker function simply prints the indexes of the predictions that the classifier got correct, just to find a good index to display. As 15 was one of the true numbers we can then display this number.



## 4.3 Qc Implement a dummy binary classifier

we are asked to build a dummy clacifier that simply returns zeros, in our case, that would mean that the classifies everything as not 5.

```

1 class DummyClassifier(BaseEstimator):
2     def fit(self, X, y=None):
3         pass
4     def predict(self,X):
5         return np.zeros((len(X), 1), dtype=bool)
6
7 # Instantiates the DummyClassifier
8 Dummy_clf = DummyClassifier()
9
10 pred = Dummy_clf.predict(X_Test)
11
12 accuracy = accuracy_score(Arr,pred)
13
14 print("Accuracy of DummyClassifier:", accuracy)

```

this prints:

```
Accuracy of DummyClassifier: 0.9259
```

this is worse then the one from HOLM as expected, but its still quite high as most numbers in our dataset isn't 5. Or in other words, 92% of our dataset is not 5.

## 4.4 Qd Conclusion

The first assignment was a useful introduction to the openml.org. This exercise showed us that there exist with-in reach training data, should we try and train our machine learning knowledge in our spare time. In the second exercise we used the SGDClassifier to predict either or not the number/picture in our test sample were a five or not. The exercise gave us an hands on experience with setting up and visualising the output of a binary SGDClassifier. In the finale exercise we used the accuracy\_score to see what the prediction-score would be if tested on our boolean prediction from the SGDClassifier. Here the prediction was that 92.59% of boolean array was zeroes! In correlation to HOML "Measurring Accuracy Using Cross-Validation"the procentage seemed reasonable due to the way each accuracy is calculated differently.

## 5 L02 PERFORMANCE METRICS

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

The purpose of this assignment was to create a custom accuracy function and run it on both a "real" classifier and a dummy classifier that only returns zeros.

```
1 def MyAccuracy(y_true, y_pred):
2     if len(y_true) == 0: #Safeguard against division by zero
3         # return an error message if the length of y_true is 0
4         print("Error: y_true is empty")
5         return 0
6     TrueValues = 0
7     for y, x, in zip(y_true, y_pred): #zip is just syntactic sugar for pairing
8         # the two arrays together
9         if y == x:
10            TrueValues += 1
11 return TrueValues / len(y_true) #This is the accuracy formula as TrueValues
12 # contain both TP and TN
```

we then run the function on our test data vs our predictions made by our two classifiers from earlier.

```
1 print("MyAccuracy: ", MyAccuracy(y_test_5, clf_Predict))
2 print("Sklearn accuracy: ", accuracy_score(y_test_5, clf_Predict))
3
4 print("MyAccuracy: ", MyAccuracy(y_test_5, Dummy_pred))
5 print("Sklearn accuracy: ", accuracy_score(y_test_5, Dummy_pred)) # Cirka 9%
6 # numbers are 5's
```

which outputs:

```
MyAccuracy: 0.9769
Sklearn accuracy: 0.9769
MyAccuracy: 0.9108
Sklearn accuracy: 0.9108
```

As we can see, the accuracy is the same for both my custom function and the sklearn function. Meaning that my custom function works as intended.

## 5.2 Qb Implement Precision, Recall and $F_1$ -score and test it on the MNIST data for both the SGD and Dummy classifier models

We have been asked to implement the different types of "scoring". In our case this will be Recall, sensitivity, Precision and F1 score.

```
1 def MyPrecision(y_true, y_pred):
2     TP = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp == True) #let's
3     # use list comprehension to make the code more readable
4     FP = sum(1 for yt, yp in zip(y_true, y_pred) if yt == False and yp == True)
5
6     if TP + FP == 0:
7         return 0 #Safeguard against division by zero
8
9     return TP / (TP + FP) #Precision formula
10
11
12 def MyRecall(y_true, y_pred):
13
14     TP = sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp == True)
15     FN = sum(1 for yt, yp in zip(y_true, y_pred) if yt == True and yp == False)
16
17     if TP + FN == 0:
18         return 0 #Safeguard against division by zero
19
20     return TP / (TP + FN) #Recall formula
21
22
23 def MyF1Score(y_true, y_pred):
24
25     precision = MyPrecision(y_true, y_pred)
26     recall = MyRecall(y_true, y_pred)
27
28     if precision + recall == 0:
29         return 0 #Safeguard against division by zero
30
31     return 2/((1/recall) + (1/precision)) #F1 score formula
```

we also build in safety for when the length is zero by simply returning zero. We could also interrupt the code at this point or use assert.

we then test the different scorings using the same datasets and classifiers as before:

5.2. QB IMPLEMENT PRECISION, RECALL AND  $F_1$ -SCORE AND TEST IT ON THE MNIST DATA FOR BOTH THE SGD AND DUMMY CLASSIFIER MODELS

Group 13

```
1 #Test the functions with the SGD classifier
2
3 print("MyPrecision: ", MyPrecision(y_test_5, clf_Predict))
4 print("Sklearn Precision: ", precision_score(y_test_5, clf_Predict))
5
6 print("MyRecall: ", MyRecall(y_test_5, clf_Predict))
7 print("Sklearn Recall: ", recall_score(y_test_5, clf_Predict))
8
9 print("MyF1Score: ", MyF1Score(y_test_5, clf_Predict))
10 print("Sklearn F1Score: ", f1_score(y_test_5, clf_Predict))
11
12 #Test the functions with the Dummy classifier
13 print("\n\n")
14
15 print("MyPrecision: ", MyPrecision(y_test_5, Dummy_pred))
16 print("Sklearn Precision: ", precision_score(y_test_5, Dummy_pred))
17
18 print("MyRecall: ", MyRecall(y_test_5, Dummy_pred))
19 print("Sklearn Recall: ", recall_score(y_test_5, Dummy_pred))
20
21 print("MyF1Score: ", MyF1Score(y_test_5, Dummy_pred))
22 print("Sklearn F1Score: ", f1_score(y_test_5, Dummy_pred))
```

this prints:

```
MyPrecision: 0.9460188933873145
Sklearn Precision: 0.9460188933873145
MyRecall: 0.7858744394618834
Sklearn Recall: 0.7858744394618834
MyF1Score: 0.8585425597060624
Sklearn F1Score: 0.8585425597060624

MyPrecision: 0
Sklearn Precision: 0.0
MyRecall: 0.0
Sklearn Recall: 0.0
MyF1Score: 0
Sklearn F1Score: 0.0
```

As we can see, the precision, recall and F1 score is the same for both my custom function and the sklearn function. Meaning that my custom function works as intended. Precision, recall and F1 score are all 0 for the dummy classifier, as it always predicts 0.

## 5.3 Qc The Confusion Matrix

Here we are asked to construct a confusion matrix using sklearn's own functions using our data. We were also asked what would happen if we input the data in the wrong order.

its quite simply done like so:

```
1 from sklearn.metrics import confusion_matrix
2
3 M_dummy = confusion_matrix(y_test_5, Dummy_pred)
4 M_SGD = confusion_matrix(y_test_5, clf_Predict)
5
6 print("M_dummy: ", M_dummy)
7 print("M_SGD: ", M_SGD)
```

this prints:

```
M_dummy: [[9108 0]
          [ 892 0]]
M_SGD: [[9068 40]
        [ 191 701]]
```

The confusion matrix is organized as follows:

[[TN FP]

[FN TP]]

If you mess up the parameters, you will get a transposed version of the confusion matrix. This is because the first parameter is the predicted values and the second parameter is the actual values. If you switch the two parameters, you will get the predicted values as the rows and the actual values as the columns.

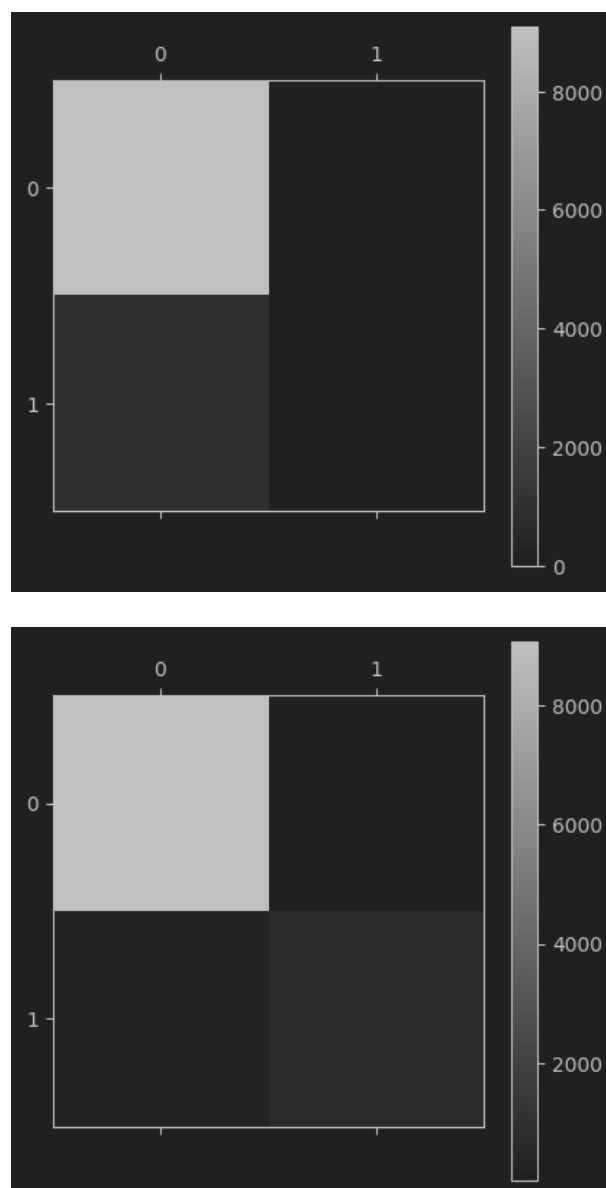
## 5.4 Qd A Confusion Matrix Heat-map

We were asked to create a heatmap of our confusion matrix. We can use matplotlib lib for this.

```
1 import matplotlib.pyplot as plt
2
3 plt.matshow(M_dummy, cmap=plt.cm.gray_r) # gray_r is a gray colormap, r stands
    ↪ for reverse, meaning that 0 is white and 1 is black
```

```
4 plt.colorbar()
5 plt.show()
6
7 plt.matshow(M_SGD, cmap=plt.cm.gray_r)
8 plt.colorbar()
9 plt.show()
```

we get the following plots:



as we can see, we more values that are TN then TP as expected since we are only predicting 5s



## 5.5 Qe Conclusion

Its really important to fully understand why we use certain metrics and how they are calculated. This is important because it allows us to understand the performance of our models (kinda) and how they can be improved. The overall learning outcome of the exercises is that we now have a better understanding of how to calculate and use the accuracy, precision, recall and F1 score metrics. We also have a better understanding of how to use the confusion matrix and how to interpret it. This is important because it allows us to understand the performance of our models and how they can be improved. One note is that all these metrics perform very poorly on more complex datasets, as they are not robust against outliers or rapid change in data.