# L03_metrics

## Introduction

In this exercise we will explore the confusion matrix and the terms Accuracy-, Precision-, Recall- and F1Score.
We will create our own functions of these terms and compare them with scikit-learns integrated score functions.
Furthemore, there will be generated heat maps of the confusion matrix and at last we will work with our dataset from L03_dataset.

## Confusion Matrix

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

Implement a general accuracy function `MyAccuracy` , that takes `y_pred` and `y_true` as input parameters.

Reuse your MNIST data loader and test the `MyAccuracy` function both on your dummy classifier and on the Stochastic Gradient Descent classifier (with setup parameters as in [HOLM]).

#### *Qa implementation*

In [1]:
```python
# Assignment Qa:
from libitmal import dataloaders as dl
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
import numpy as np
from libitmal import utils as itmalutils

# Function that calculates the accuracy from the confusion matrix
def MyAccuracy(y_pred, y_true):
    TP, FP, FN, TN= GetConfusionMatrix(y_pred, y_true)
    N = (TP + FP + TN + FN)
    accuracy = (TP + TN ) / N
    return accuracy

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

    print("my a          =",a0)
    print("scikit-learn a=",a1, "\n")

    itmalutils.InRange(a0,a1)
    return

# Function to Generate a Confusion Matrix
def GetConfusionMatrix(y_pred, y_true):
    cm = confusion_matrix(y_true, y_pred)
    print("Confusion Matrix:\n", cm) # only for debugging
    TP = cm[0][0]
    FP = cm[0][1]
    FN = cm[1][0]
    TN = cm[1][1]
    return (TP, FP, FN, TN)

# Get data set and train_test_split
X, y_true = dl.MNIST_GetDataSet()
print(f" X.shape={X.shape}, y_true.shape={y_true.shape}")
X_train, X_test, y_train, y_test = train_test_split(X, y_true, test_size=0.2,shuffl
e=True, random_state=42)

y_train_5 = (y_train == '5')
y_test_5 = (y_test == '5')

# DummyClassifier Accuracy
dclf = dl.DummyClassifier()
dclf.fit(X_train, y_test)
y_pred_dummy = dclf.predict(X_test)

print("\nDummyClassifier Accuracy:")
acc_dummy = TestAccuracy(y_test_5, y_pred_dummy)

# SGDClassifier Accuracy
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
y_pred_sgd = sgd_clf.predict(X_test)

print("SGDClassifier Accuracy:")
acc_sgd = TestAccuracy(y_test_5, y_pred_sgd)
```

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

DummyClassifier Accuracy:
Confusion Matrix:
 [[12727  1273]
 [    0     0]]
my a          = 0.9090714285714285
scikit-learn a= 0.9090714285714285


C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradi
ent.py:166: FutureWarning: max_iter and tol parameters have been added in SGDCla
ssifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None
. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max
_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)

SGDClassifier Accuracy:
Confusion Matrix:
 [[12630   428]
 [   97   845]]
my a          = 0.9625
scikit-learn a= 0.9625
```

### *Qa results*

The MyAccuracy function produces the same output as sklearns accuracy_score function, which is a huge success considering the two value are identical. The DummyClassifier accuracy is around 90.9% accuracy while the SGDClassifier is around 96.2% accuracy.

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

Now, implement the `MyPrecision`, `MyRecall` and `MyF1Score` functions, again taking MNIST as input, using the SGD and the Dummy classifiers and make some test vectors to compare to the functions found in Scikit-learn...

### *Qb implementation*

In [5]:
```python
# Assignment Qb
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Function to calculate Precision score
def MyPrecision(y_pred, y_true):
    TP, FP, FN, TN = GetConfusionMatrix(y_pred, y_true)
    PPV = TP/(TP+FP)
    return PPV

# Function to calculate Recall/Sensitivity score
def MyRecall(y_pred, y_true):
    # TODO: you impl here
    TP, FP, FN, TN = GetConfusionMatrix(y_pred, y_true)
    return (TP / (TP + FN))

# Function to calculate F1Score
def MyF1Score(y_pred, y_true):
    # TODO: you impl here
    TP, FP, FN, TN = GetConfusionMatrix(y_pred, y_true)
    return (2*TP)/(2*TP + FP + FN)

## Test Functions for Precision, Recall and F1Score
def TestPrecision(y_pred, y_true):
    p0=MyPrecision(y_pred, y_true)
    p1=precision_score(y_pred, y_true)

    print("my p          =",p0)
    print("scikit-learn p=",p1, "\n")

    itmalutils.InRange(p0,p1)
    return

def TestRecall(y_pred, y_true):
    r0=MyRecall(y_pred, y_true)
    r1=recall_score(y_pred, y_true)

    print("my r          =",r0)
    print("scikit-learn r=",r1, "\n")

    itmalutils.InRange(r0,r1)
    return

def TestF1Score(y_pred, y_true):
    f1_0=MyF1Score(y_pred, y_true)
    f1_1=f1_score(y_pred, y_true)

    print("my f1          =",f1_0)
    print("scikit-learn f1=",f1_1, "\n")

    itmalutils.InRange(f1_0,f1_1)
    return


## Testing Precision
print("\nDummyClassifier Precision:")
p_dummy = TestPrecision(y_test_5, y_pred_dummy)

print("\nSGDClassifier Precision:")
p_sgd = TestPrecision(y_test_5, y_pred_sgd)

print("***************************")
```

```
DummyClassifier Precision:
Confusion Matrix:
 [[12727  1273]
 [    0     0]]
my p          = 0.9090714285714285
scikit-learn p= 0.0


SGDClassifier Precision:
Confusion Matrix:
 [[12630   428]
 [   97   845]]
my p          = 0.9672231582171849
scikit-learn p= 0.8970276008492569

**************************

DummyClassifier Recall:
Confusion Matrix:
 [[12727  1273]
 [    0     0]]
my r          = 1.0
scikit-learn r= 0.0


SGDClassifier Recall:
Confusion Matrix:
 [[12630   428]
 [   97   845]]
my r          = 0.9923784081087452
scikit-learn r= 0.6637863315003928

**************************

DummyClassifier F1Score:
Confusion Matrix:
 [[12727  1273]
 [    0     0]]
my f1          = 0.9523702622815879
scikit-learn f1= 0.0


SGDClassifier F1Score:
Confusion Matrix:
 [[12630   428]
 [   97   845]]
my f1          = 0.9796393251890634
scikit-learn f1= 0.7629796839729119
```

***Qb results***

There is a difference between Precision, Recall and F1Score between our own functions using algebra and scikit-learns integrated functions. Furthermore, scikit-learn does not have the ability to produce the scores for the DummyClassifier even though the Matrix is properly constructed.

### Qc The Confusion Matrix

Revisit your solution to Qb in the `dummy_classifier.ipynb` . Did you manage to print the confusion matrix for both the Dummy and the SGD classifier?

How are the Scikit-learn confusion matrix organized, where are the TP, FP, FN and TN located in the matrix indices, and what happens if you mess up the parameters calling

```
confusion_matrix(y_train_pred, y_train_5)
```

instead of

```
confusion_matrix(y_train_5, y_train_pred)
```

***QUESTION:*** Finally, compare the real and symmetric auto-covariance matrix, $\Sigma$, with the real but non-symmetric confusion matrix, $\mathbf{M}$. What does the diagonal represent in the covar- and confusion matrix respectively, and why is the covar- symmetric, but the confusion not?

***ANSWER:*** The auto-covariance matrix measues variance between all features, which is the explanation behind why it is symmetrical. Furthermore the diagonal fo the auto-covariane matrix is the variance $\sigma^2$.
On the other hand the Confusion Matrix diagonal shows the correct amount of predictions. It is possible to identify all the correct classifications in the diagonal and the wrong classifications in the off-diagonal.

### *Qc implementation*

We managed to print the following confusion matrices from Qb in dummy_classifier.ipynb

```
M_dummy=[[54579     0]
        [ 5421      0]]

M_SDG=[[52953   1626]
      [  967   4454]]
```

Following python code shows the difference when swapping the arguments in confusion_matrix() function:

In [4]:
```
print("Printing confusion_matrix(y_test_5, y_pred_sgd):")
GetConfusionMatrix(y_test_5, y_pred_sgd)

print("\nPrinting confusion_matrix(y_pred_sgd, y_test_5):")
GetConfusionMatrix(y_pred_sgd, y_test_5)
```

```
Printing confusion_matrix(y_test_5, y_pred_sgd):
Confusion Matrix:
 [[12630   428]
 [   97   845]]

Printing confusion_matrix(y_pred_sgd, y_test_5):
Confusion Matrix:
 [[12630    97]
 [  428   845]]
```

Out[4]:
```
(12630, 97, 428, 845)
```

As we can see from the two prints. When using Scikit-learns confusion_matrix, it is important to call the arguments properly. Otherwise the output Matrix will result in values that are displaced.
In this case we can see that the FP and FN has swapped positions in the confusion matrix, which is very misleading.

### *Qc results*

Scikit-learn confusion matrix is organized as following:
tn, fp, fn, tp = confusion_matrix(...)
Source: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

It has been proved that the arguments must be placed properly when using sklearn's confusion_matrix() function, otherwise FP and FN may be displaced in the confusion matrix.

Furthermore the explanation for auto-covariance and confusion matrix were explained in the beginning of assigment Qc.

### Qd A Confusion Matrix Heat-map

Generate a *heat map* image for the confusion matrices, `M_dummy` and `M_SGD` respectively, getting inspiration from [HOML], pp96-97.

This heat map could be an important guide for you when analysing multiclass data in the future.

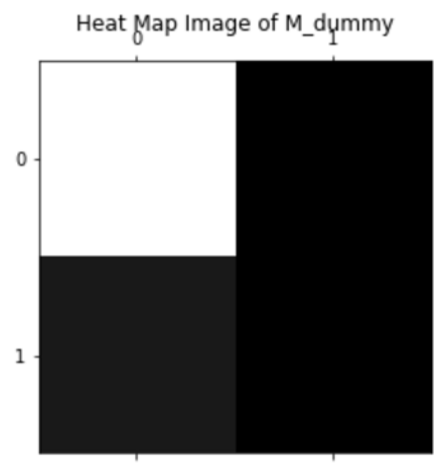
### *Qd implementation*

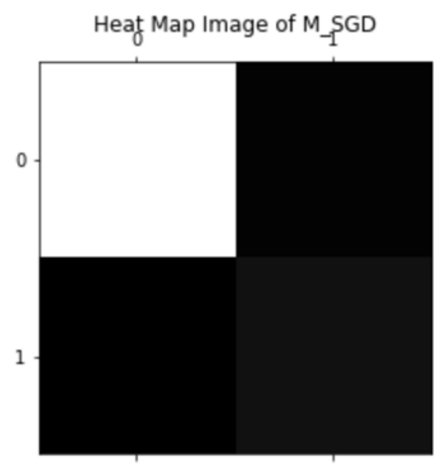
In [6]:
```python
# Assignment Qd:
import matplotlib.pyplot as plt

M_dummy=[[54579, 0],
        [5421, 0]]

M_SGD=[[52953, 1626],
      [967,4454]]
plt.matshow(M_dummy, cmap=plt.cm.gray)
plt.title("Heat Map Image of M_dummy")
plt.show()

plt.matshow(M_SGD, cmap=plt.cm.gray)
plt.title("Heat Map Image of M_SGD")
plt.show()
```




***Qd results***

The heat map image has been constructed as seen on the two plots. With the heat map, we are able to differentiate the low and high values of the confusion matrix.

### Qe Run a classifier on your data

Finally, try to run a classifier on the data-set you selected previously, perhaps starting with the SGD.

Is it possible to classify at all on your data, or do we need regression instead?

Are you able to do supervised learning, or are there no obvious `y_true` data in your set at all?

If your data is in the form, where you are able to do supervised-classification, could you produce a confusion matrix heatmap, then?

#### *Qe implementation*

In [8]:

```
# Assignment Qe
import pandas as pd
import numpy as np
from libitmal import dataloaders as dl
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Following code is from: https://www.kaggle.com/microtang/poe-path-of-exile-statis
tics-an-exploration
# Who has made a train_and_split on the dataset with the use of xgboost library
# df = dl.GetOrderedClassInLadder('SSF Harbinger HC')
# print(df)
df = pd.read_csv('poe_stats.csv', delimiter = ',')
from sklearn.model_selection import train_test_split
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')
#data process
df['is_in_top_30'] = np.zeros(len(df))
df.loc[df['rank'] <= 30, 'is_in_top_30'] = True
df.loc[df['rank'] >30,  'is_in_top_30'] = False
labelencoder_y= LabelEncoder()
df['is_in_top_30'] = labelencoder_y.fit_transform(df['is_in_top_30'])
X = df[['dead', 'online', 'level', 'class', 'challenges', 'ladder']]
y = df['is_in_top_30']
# Encoding the categorical data
labelencoder_X_1 = LabelEncoder()
X['dead'] = labelencoder_X_1.fit_transform(X['dead'])
labelencoder_X_2 = LabelEncoder()
X['online'] = labelencoder_X_2.fit_transform(X['online'])
labelencoder_X_3 = LabelEncoder()
X['class'] = labelencoder_X_3.fit_transform(X['class'])
labelencoder_X_5 = LabelEncoder()
X['ladder'] = labelencoder_X_5.fit_transform(X['ladder'])
labelencoder_X_6 = LabelEncoder()
#X['twitch'] = labelencoder_X_6.fit_transform(X['twitch'])
# # Splitting the dataset into the Training set and Validation set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_
state = 0)
dt = xgb.DMatrix(X_train.as_matrix(),label=y_train.as_matrix())
dv = xgb.DMatrix(X_test.as_matrix(),label=y_test.as_matrix())
```

In [9]:
```
params = {
    "eta": 0.2,
    "max_depth": 4,
    "objective": "binary:logistic",
    "silent": 1,
    "base_score": np.mean(y_train),
    'n_estimators': 1000,
    "eval_metric": "logloss"
}
### Ignore the following output, it's for validation since it's a slow process.
model = xgb.train(params, dt, 5000, [(dt, "train"),(dv, "valid")], verbose_eval=500
)
```

```
[0]     train-logloss:0.011386  valid-logloss:0.009517
[500]   train-logloss:0.002008  valid-logloss:0.004369
[1000]  train-logloss:0.001814  valid-logloss:0.004994
[1500]  train-logloss:0.001738  valid-logloss:0.005486
[2000]  train-logloss:0.001699  valid-logloss:0.005857
[2500]  train-logloss:0.00168   valid-logloss:0.006095
[3000]  train-logloss:0.001666  valid-logloss:0.006329
[3500]  train-logloss:0.001656  valid-logloss:0.006535
[4000]  train-logloss:0.001648  valid-logloss:0.006739
[4500]  train-logloss:0.001642  valid-logloss:0.006927
[4999]  train-logloss:0.001638  valid-logloss:0.007084
```

In [10]:
```
from sklearn.metrics import confusion_matrix
# Prediction on validation set
y_pred = model.predict(dv)

# Calculating accuracy of the validation set
TestAccuracy((y_pred>0.5),y_test)
```

```
Confusion Matrix:
 [[11935     3]
 [   10     8]]
my a           = 0.9989126798260288
scikit-learn a= 0.9989126798260288
```

In [13]:
```
# Creating DummyClassifier and calculating its accuracy
dclf = dl.DummyClassifier()
dclf.fit(X_train, y_test)
y_pred_dummy = dclf.predict(X_test)

acc_dummy = TestAccuracy(y_test, y_pred_dummy)
```

```
Confusion Matrix:
 [[11938    18]
 [    0     0]]
my a           = 0.9984944797591168
scikit-learn a= 0.9984944797591168
```

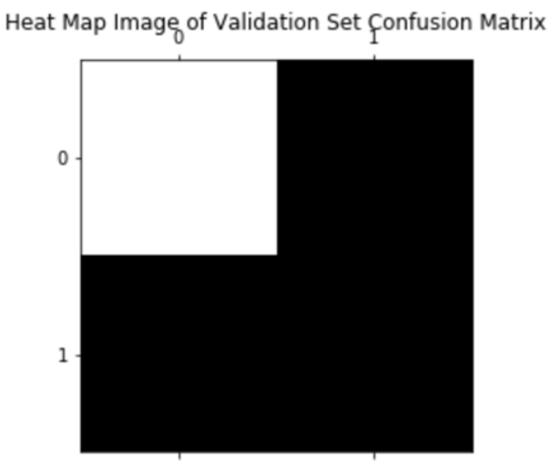
```
In [14]: # Creating SGDClassifier and calculating its accuracy
         sgd_clf = SGDClassifier(random_state=42)
         sgd_clf.fit(X_train, y_train)
         y_pred_sgd = sgd_clf.predict(X_test)

         print("SGDClassifier Accuracy:")
         acc_sgd = TestAccuracy(y_test, y_pred_sgd)
```

```
SGDClassifier Accuracy:
Confusion Matrix:
 [[11938    18]
 [    0     0]]
my a           = 0.9984944797591168
scikit-learn a= 0.9984944797591168
```

Both the DummyClassifier and SGDClassifier removes the False Negatives and True Positives while increasing the False Positives if we were to compare it with the confusion matrix given from the validation set. However their accuracy is not too far behind the validation set.

```
In [16]: # Generating heatmaps of the confusion matrix for validation set
         M_validation =[[11935, 3],
                        [10, 8]]
         plt.matshow(M_validation, cmap=plt.cm.gray)
         plt.title("Heat Map Image of Validation Set Confusion Matrix")
         plt.show()
```




As expected, the heat map image produces only the white block for the TP in our scenario.

### ***Qe results***

We ran the DummyClassifier and SGDClassifier on the data-set, furthermore we were able to classify our data but we could also have made a regression.

It is possible to do supervised learning in this dataset, as there are several parameters to be experimented with.

At last we successfully generated a confusion matrix heatmap out of the confusion matrix given from the validation set.

# Conclusion

We were able to implement our own accuracy score function which produces the identical value as sklearns accuracy_score. However there was a difference between our precision, recall and f1score compared to sklearns integrated score functions.

Furthermore we were able to generate heatmaps of the confusion matrices, which are not too useful at this point, but if we were to expand it we would be able to use it in better scenarios.

At last we tested our data-set from L03_dataset exercise and were able to include several methods we experimented with throughout this exercise, such as accuracy, confusion matrix, heatmap etc.