L04 performance_metricstest

February 13, 2026

1 L04 Performance Metrics

Qa Implement the Accuracy function and test it on the MNIST data. Implement a general accuracy function MyAccuracy(y_true, y_pred). The accuracy is calculated using the formular:

$a = \frac{TP+TN}{TP+TN+FP+FN}$

Where

TP is the True Positive ie. the number of cases that is both prediced positive and actually is positive

TN is the True Negative ie. the number of cases that is prediced negative and actually is negative

FP is the False positive ie. the number of cases that is prediced positive but actually is negative

FN is the False negative ie. the number of cases that is prediced negative but actually is positive

In the case that the denominator is zero, 0.0 is returned.

The accuracy of the '5/not-5' ground-truth data, is compared with the acutal value from sklearn.metrics.accuracy_score(), these are the same.

[ ]:
```
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
from sklearn.datasets import fetch_openml
import matplotlib
import matplotlib.pyplot as plt

def MyAccuracy(y_true, y_pred):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    # Calculate components
    TP = np.sum((y_pred == True)  & (y_true == True))
    FN = np.sum((y_pred == False) & (y_true == True))
    FP = np.sum((y_pred == True)  & (y_true == False))
    TN = np.sum((y_pred == False) & (y_true == False))

    # Handle the zero-division case mentioned in the exercise
```

```
    denominator = (TP + FN + FP + TN)
    if denominator == 0:
        return 0.0

    acc = (TP + TN) / denominator
    return acc

# TEST FUNCTION: example of a comperator, using Scikit-learn accuracy_score
def TestAccuracy(y_true, y_pred):
   a0=MyAccuracy(y_true, y_pred)
   a1=accuracy_score(y_true, y_pred)

   print(f"\nmy a          ={a0}")
   print(f"scikit-learn a={a1}")

def MNIST_GetDataSet():
    X, y = fetch_openml('mnist_784', return_X_y=True, as_frame=False,␣
 ↪cache=True)
    # return_X_y=True: Returns (data, target) instead of a dictionary
    # as_frame=False: Returns data as Numpy arrays
    # cache=True: Saves data locally to avoid re-downloading

    # Convert labels from str. to int.
    y = y.astype(np.int8)

    return X, y

X, y = MNIST_GetDataSet()
print(f"X.shape={X.shape}")
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}")

train_index=50000
X_train, X_test, y_train, y_test = X[:train_index], X[train_index:], y[:
 ↪train_index], y[train_index:]
y_train_5 = (y_train == 5) # Use integer 5
y_test_5  = (y_test == 5)

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)

# Get predictions for the test set
y_pred = sgd_clf.predict(X_test)
```

```
TestAccuracy(y_test_5, y_pred)
```

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

my a           =0.95505
scikit-learn a=0.95505
```

Qb Implement Precision, Recall and F_1-score and test it on the MNIST data for both the SGD and Dummy classifier models To implement the MyPrecision, MyRecall and MyF1Score functions, use the following formulars:

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

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

F_1-score $F_1 = \frac{2pr}{p+r}$

These are calulated for the SGD and the Dummy classifiers of the '5/not-5' ground-truth data and compared to the functions found in Scikit-learn, resulting in similar results.

In the case that the denominator is zero, 0.0 is returned.

[6]:
```
# Qb
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

def MyPrecision(y_true, y_pred):
    # Ensure inputs are numpy arrays
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    # Calculate components
    TP = np.sum((y_pred == True)  & (y_true == True))
    FN = np.sum((y_pred == False) & (y_true == True))
    FP = np.sum((y_pred == True)  & (y_true == False))
    TN = np.sum((y_pred == False) & (y_true == False))

    # Handle the zero-division case
    denominator = (TP + FP)
    if denominator == 0:
        return 0.0
    p = TP / denominator
    return p

def MyRecall(y_true, y_pred):
    # Ensure inputs are numpy arrays
    y_true = np.array(y_true)
```

```
    y_pred = np.array(y_pred)

    # Calculate components
    TP = np.sum((y_pred == True)  & (y_true == True))
    FN = np.sum((y_pred == False) & (y_true == True))
    FP = np.sum((y_pred == True)  & (y_true == False))
    TN = np.sum((y_pred == False) & (y_true == False))

    # Handle the zero-division case
    denominator = (TP + FN)
    if denominator == 0:
        return 0.0
    r = TP / (TP + FN)
    return r

def MyF1Score(y_true, y_pred):
    p=MyPrecision(y_true, y_pred)
    r=MyRecall(y_true, y_pred)

    # Handle the zero-division case
    denominator = (p+r)
    if denominator == 0:
        return 0.0
    F1=2*p*r / denominator
    return F1

# TEST FUNCTION: example of a comperator, using Scikit-learn accuracy_score
def TestPrecision(y_true, y_pred):
   p0=MyPrecision(y_true, y_pred)
   p1=precision_score(y_true, y_pred)

   print(f"\nmy p          ={p0}")
   print(f"scikit-learn p={p1}")

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

   print(f"\nmy r          ={r0}")
   print(f"scikit-learn r={r1}")

def TestF1Score(y_true, y_pred):
   F1_0=MyF1Score(y_true, y_pred)
   F1_1=f1_score(y_true, y_pred)

   print(f"\nmy F1          ={F1_0}")
   print(f"scikit-learn F1={F1_1}")
```

```
TestPrecision(y_test_5, y_pred)
TestRecall(y_test_5, y_pred)
TestF1Score(y_test_5, y_pred)
```

```
my p          =0.7123479887745556
scikit-learn p=0.7123479887745556

my r          =0.8428334255672385
scikit-learn r=0.8428334255672385

my F1          =0.7721166032953105
scikit-learn F1=0.7721166032953105
```

Qc The Confusion Matrix Generating the confusion matrix for both the Dummy and the SGD classifier using the scklearn.metrics.confusion_matrix function.

The Scikit-learn confusion matrix organized like this:

scklearnconf=[[TN FP] [FN TP]]

If the parameter calling is wrong (see under) the confusion matrix would be transposed.

Wrong: confusion_matrix(y_test_5_pred, y_test5)

Correct: confusion_matrix(y_test_5, y_test_5_pred)

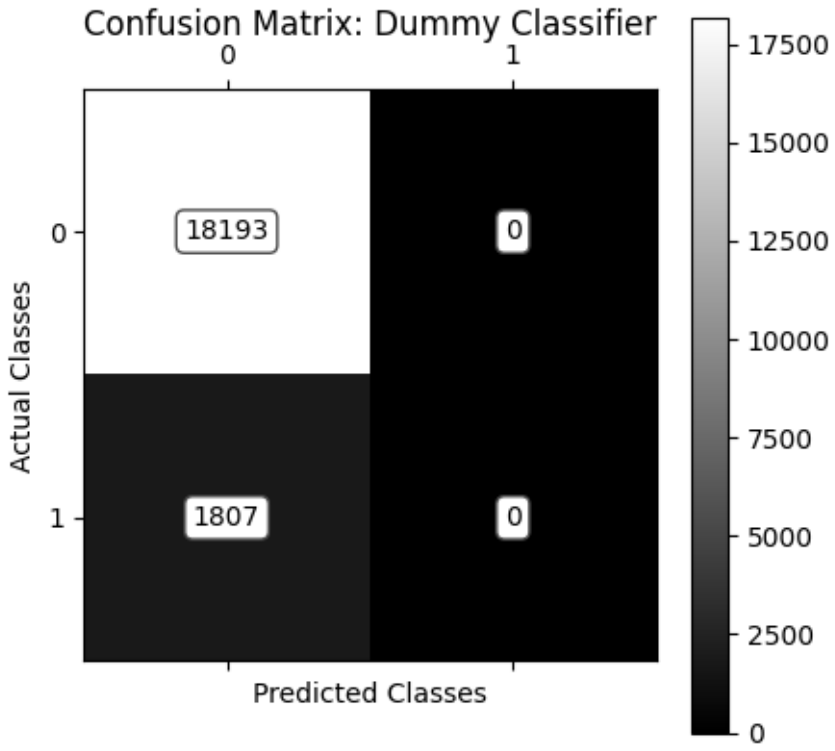
[7]:
```
# Qc
from sklearn.metrics import confusion_matrix
from sklearn.base import BaseEstimator
from sklearn.linear_model import SGDClassifier

class DummyClassifier(BaseEstimator):
    def fit(self, X, y=None):
        return self # Standard practice to return self

    def predict(self, X):
        # Change (len(X), 1) to just (len(X),) to make it 1D
        return np.zeros(len(X), dtype=bool)

# Instantiate and predict for dnmmy
never_5_clf = DummyClassifier()
y_never_5_pred = never_5_clf.predict(X_test)

# Generate Matrix for dummy
conf_matrix = confusion_matrix(y_test_5, y_never_5_pred)
print("Confusion Matrix for Dummy Classifier:")
print(conf_matrix)
```

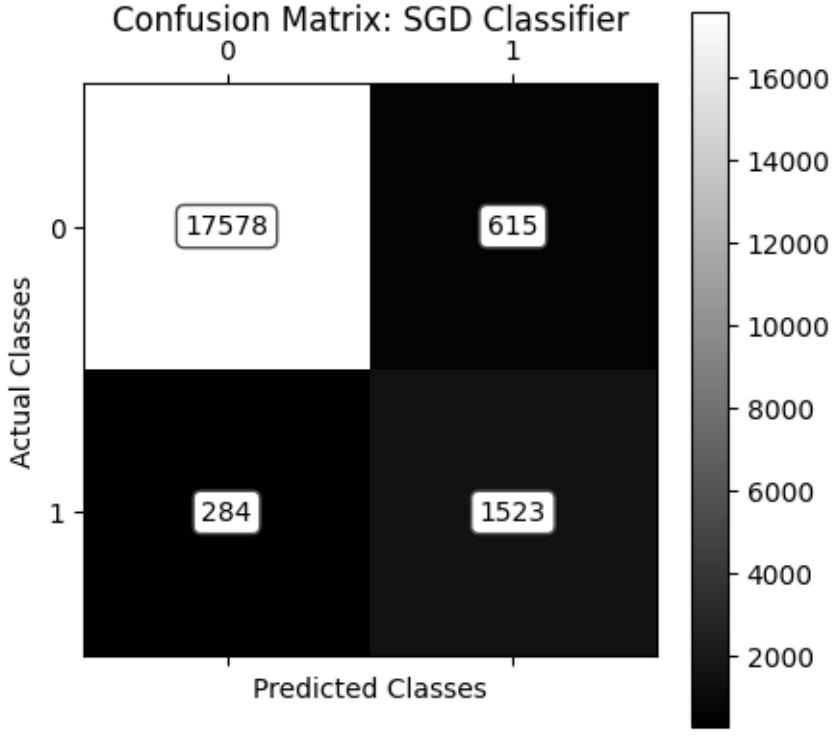
```
# Instantiate, fit and predict the model
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
y_sgd_pred = sgd_clf.predict(X_test)

# Generate Confusion Matrix for SGD
M_SGD = confusion_matrix(y_test_5, y_sgd_pred)
print("Confusion Matrix for SGD Classifier:")
print(M_SGD)
```

```
Confusion Matrix for Dummy Classifier:
[[18193     0]
 [ 1807     0]]
Confusion Matrix for SGD Classifier:
[[17578   615]
 [  284  1523]]
```

Qd A Confusion Matrix Heat-map Generating a *heat map* image for the confusion matrices, M_dummy and M_SGD respectively, by plotting the confusion matrix and overlaying the respective number of instants.

[8]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

M_dummy = confusion_matrix(y_test_5, y_never_5_pred)
M_SGD = confusion_matrix(y_test_5, y_sgd_pred)

def plot_confusion_matrix(matrix, title):
    # matshow displays an array as a matrix
    plt.matshow(matrix, cmap=plt.cm.gray)
    plt.title(title, pad=20)
    plt.colorbar()
    plt.ylabel('Actual Classes')
    plt.xlabel('Predicted Classes')

    # Overlaying number of instants
    for (i, j), z in np.ndenumerate(matrix):
        plt.text(j, i, '{:d}'.format(z), ha='center', va='center',
                 bbox=dict(boxstyle='round', facecolor='white', edgecolor='0.
 ↪3'))

    plt.show()

# Plotting
plot_confusion_matrix(M_dummy, "Confusion Matrix: Dummy Classifier")
plot_confusion_matrix(M_SGD, "Confusion Matrix: SGD Classifier")
```


Confusion Matrix: Dummy Classifier
0
1
0
1
18193
0
1807
0
Actual Classes
Predicted Classes
17500
15000
12500
10000
7500
5000
2500
0


Confusion Matrix: SGD Classifier
0
1
0
1
17578
615
284
1523
Actual Classes
Predicted Classes
16000
14000
12000
10000
8000
6000
4000
2000

1.0.1 Qe Conclusion

`L2MatrixMult` The essence of these exercises lies in dismantling the illusion that Accuracy is a sufficient metric for evaluating machine learning models. By implementing a "Dummy Classifier" that achieves ~90% accuracy while failing to identify a single actual digit '5', we exposed the Accuracy Paradox. This demonstrated why accuracy is dangerously misleading in imbalanced datasets.

Through the manual implementation of MyAccuracy and the generation of Confusion Matrices, focus shifted from a single percentage to the four outcomes: TP, TN, FP, and FN. The learning outcome was the realization that a model's value depends on the balance between Precision (avoiding false alarms) and Recall (avoiding missed detections). Visualizing these via Heat-maps further reinforced how to perform "Error Analysis" to see exactly where a model fails. Ultimately, we learned that selecting the right performance metric is very important and often several matrics could be used.