Evaluation methods

MAC 2014-2020 Cooperación Territorial

Interreg
Fondo Europeo de Desarrollo Regional
EUROPEAN UNION

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

T M E D C
Tecnología Médica para el Desarrollo Sostenible

1

Deep Learning workflow

1.1 Example of PyTorch workflow

What we are going to cover:

In this bootcamp, we will explore a standard PyTorch workflow, which can be adapted as necessary while following the main outline of steps. Steps 1-3 will be improved through experimentation until we get the desired results in step 4.

1. Getting and preparing data

- Collect the (labeled) data
- Split them into a training /testing set
- Apply transforms

2. Building or picking a pretrained model

2 options:

- Train a model from scratch
- Transfer learning using a pretrained model

3. Training

- Pick a loss function and optimizer
- Build a training loop
- Fit the model to the training set

4. Evaluation

Calculate performance metrics on the testing set such as accuracy, recall and precision to evaluate the model

5. Deployment

Embeb the model wherever you need (desktop apps, enterprise systems…) and use it to make predictions

1.1 Example of PyTorch workflow

In this module we will explore the 4th phase

4. Evaluation

When evaluating PyTorch models, a wide range of useful libraries is available for use.
Let's check out a few examples:

- **Scikit-learn:** A versatile machine learning library that supports PyTorch models, offering various evaluation metrics and cross-validation tools.
- **TorchMetrics:** A specialized library for PyTorch models, providing deep learning-specific evaluation metrics tailored to different tasks.
- **PyTorch-Ignite**: A high-level library simplifying PyTorch model training and evaluation, with flexible APIs and built-in evaluation utilities.

1.2 Evaluation workflow

The steps involved in the evaluation process typically include:

4. Evaluation

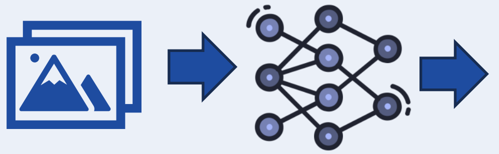
1. Data Preparation: Prepare the test dataset, ensuring it represents the real-world data the model will encounter in practical applications.

```
test_dataset
test_dataloader
```

2. Model Loading: Load the trained model that needs to be evaluated.

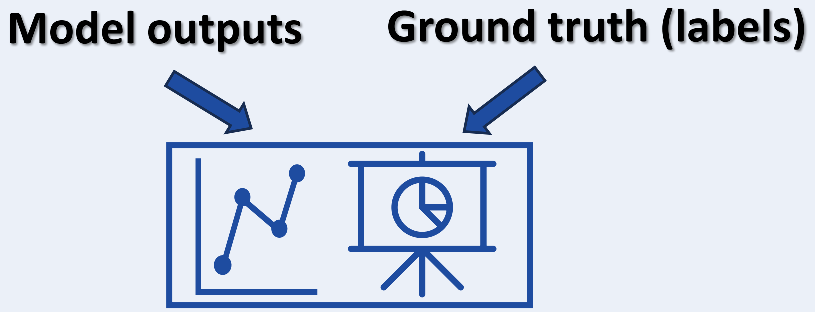
```
model = torch.load(PATH)
```

3. Forward Pass: Pass the evaluation dataset through the model to obtain predictions or outputs.



```
outputs = model(inputs)
```

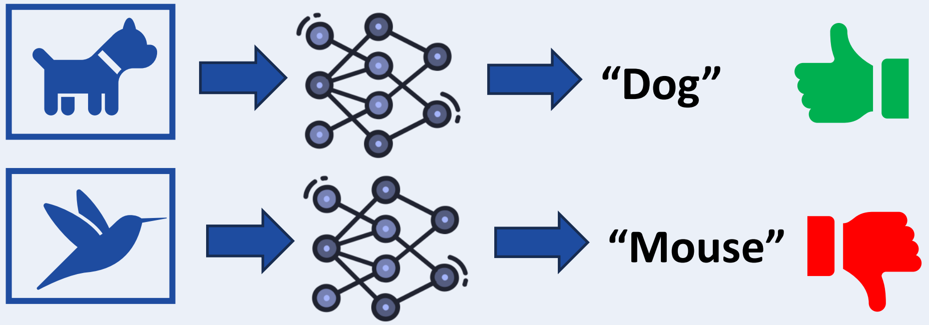
4. Metric Calculation: Calculate relevant evaluation metrics based on the model's predictions and the ground truth labels in the evaluation dataset.



1.2 Evaluation workflow

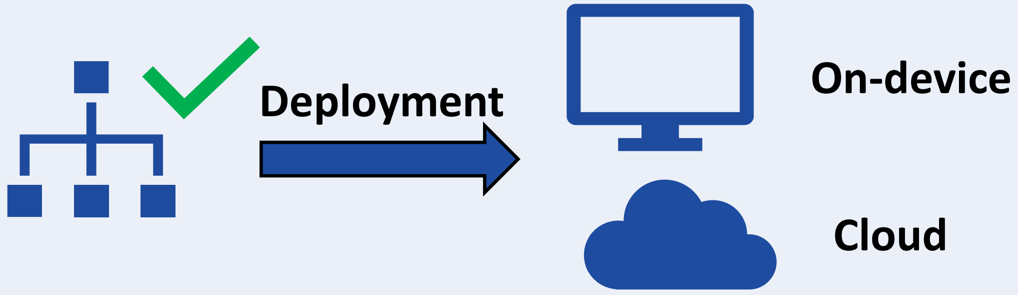
5. Performance Analysis: Analyze the evaluation metrics to understand how well the model performs on the evaluation dataset. This analysis helps identify areas that may require improvement.

6. Visualization (optional): In some cases, visualization techniques may be used to gain insights into the model's behavior and performance visually.



7. Iterative Process: Evaluation often involves an iterative process, where adjustments may be made to the model or its hyperparameters based on the evaluation results. This helps to fine-tune the model for better performance.

8. Final Assessment: After satisfactory evaluation results are achieved, the model is considered ready for deployment in real-world scenarios.



2
Evaluation metrics

2.1 Classification evaluation methods: Confusion matrix

A **confusion matrix** is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It provides a straightforward and compact breakdown of the model's predictions for each class. That way, we can pinpoint particular types of misclassifications and gain insights into the model's areas of strength and weakness

	Predicted Negative (0)	**Predicted Positive (1)**
Actually Negative (0)	**True Negative (TN)**	**False Positive (FP)**
Actually Positive (1)	**False Negative (FN)**	**True Positive (TP)**

Confusion matrix for a binary classification problem

True positive = model predicts 1 when truth is 1

True negative = model predicts 0 when truth is 0

False positive = model predicts 1 when truth is 0

False negative = model predicts 0 when truth is 1

```
Code:
from sklearn.metrics import confusion_matrix
```

2.1 Classification evaluation methods

Metric name	Definition	Metric formula
Accuracy	It is the ratio of correctly classified instances to the total number of instances in the dataset.	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	It is the ratio of correctly predicted positive instances to the total number of instances predicted as positive.	$\frac{TP}{TP+FP}$
Recall	It is the ratio of correctly predicted positive instances to the total number of actual positive instances.	$\frac{TP}{TP+FN}$
Specificity	It is the ratio of correctly predicted negative instances to the total number of actual negative instances.	$\frac{TN}{TN+FP}$
F1-Score	The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when we have imbalanced class data.	$2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$

$\}$ All these measures are in the range [0, 1], with 1 being the ideal value

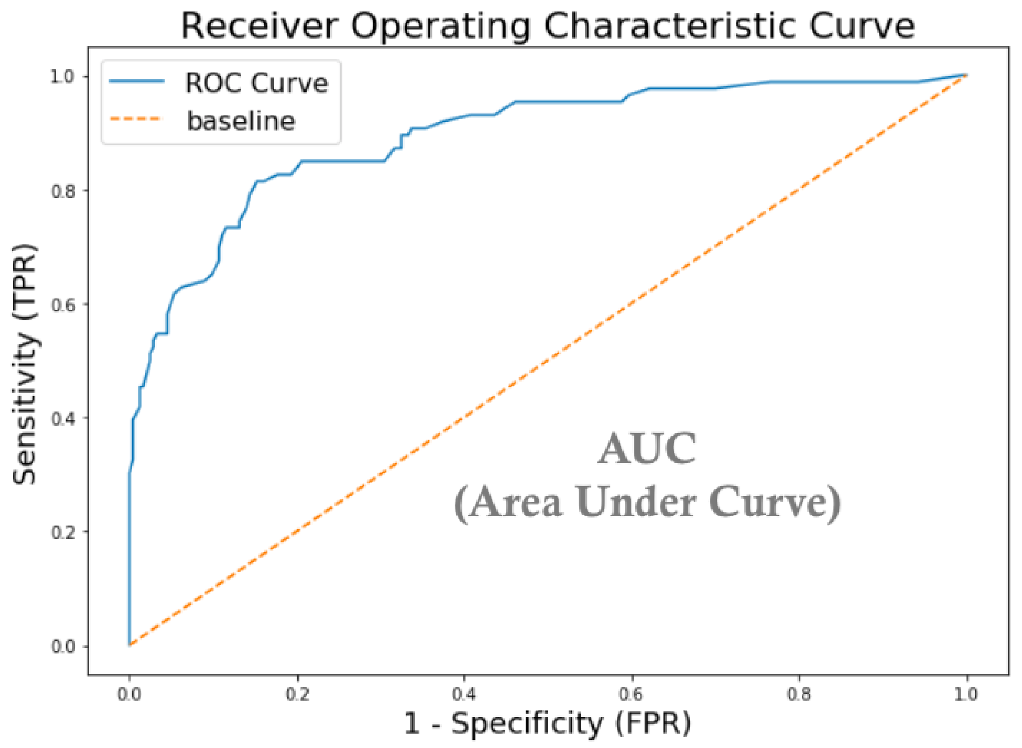
```
Code:
from sklearn.metrics import accuracy_score, precision_score . . .
```

2.1 Classification evaluation methods: ROC curve and AUC

The **ROC** (Receiver Operating Characteristic) curve is a graphical representation that illustrates the performance of a binary classification model across different discrimination thresholds. It plots the true positive rate (sensitivity) on the y-axis against the false positive rate (1 - specificity) on the x-axis.

The **AUC**, short for Area Under the Curve, is a scalar value that represents the overall performance of a binary classifier as a single number. It quantifies the area under the ROC curve. The AUC ranges between 0 and 1, where 1 is equivalent to perfect classification.

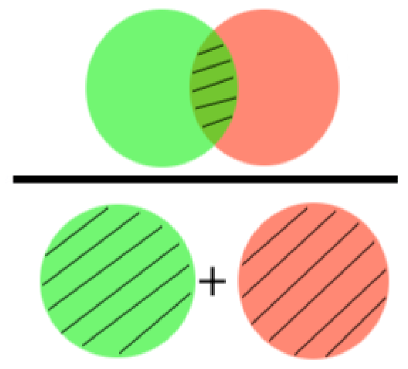
```
Code:
from sklearn.metrics import roc_curve, auc
```



2.2 Segmentation evaluation methods

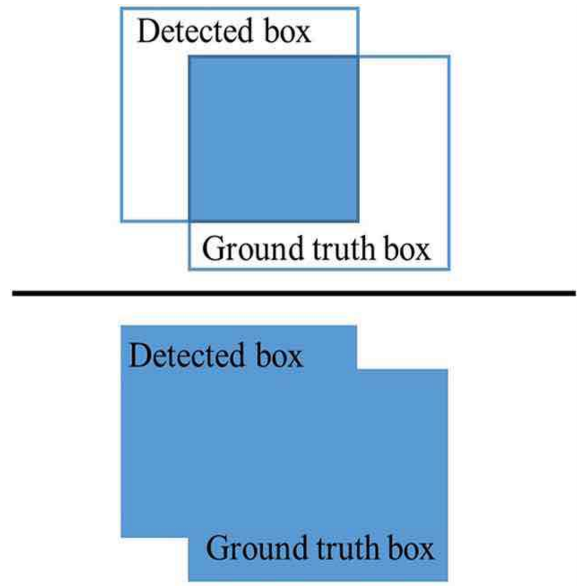
The **DICE** coefficient is used to quantify the overlap between two masks (A and B). It is defined as twice the area of intersection between A and B divided by the sum of the areas of A and B, where A corresponds to the model prediction, and B corresponds to the Ground Truth. The DICE coefficient ranges from 0 to 1, with 0 indicating no overlap between the sets and 1 indicating perfect overlap.

$$DICE = \frac{2 * |A \cap B|}{|A| + |B|} = 2 \text{ x}$$



The **IoU** (Intersection over Union) coefficient is measured by dividing the area of intersection between two sets (predicted and ground truth masks) by the area of their union. It ranges from 0 (no overlap) to 1 (perfect overlap). It is commonly used to evaluate segmentation model performance. Higher IoU scores indicate better alignment between model predictions and the actual ground truth.

$$IoU = \frac{A \cap B}{A \cup B} =$$



2

Testing loop

2.1 Build a Pytorch testing loop

Let’s image that we have a multiclass classification problem. Now, we ‘ll construct a testing loop to generate model predictions for our testing subset and then compare those predictions with the corresponding labels.

```python
model.eval()

y_true = []
y_pred = []

with torch.no_grad():
    for test_data in test_dataloader:
        test_images, test_labels = (test_data[0].to(DEVICE),
                                    test_data[1].to(DEVICE))
        output = model(test_images)
        pred_label = torch.softmax(output, dim = 1).argmax(dim = 1)

        y_true.append(test_labels.cpu())
        y_pred.append(pred_label.cpu())

y_true = torch.cat(y_true).numpy()
y_pred = torch.cat(y_pred).numpy()
```

Set the model in **evaluation mode**, disabling gradient computation and dropout layers.

Avoid updating model weights during inference and **loop through the test data** using the test_dataloader.

Move test images and labels to the **specified device** (e.g., GPU).

Perform a **forward pass** through the model to get the output. Then, apply softmax to the output and find the index of the maximum value to **get the predicted label** for each test image.

Append the true labels and predicted labels to the **respective lists.**

Concatenate the true and predicted labels lists and convert them to NumPy arrays.

2.2 Compute metrics

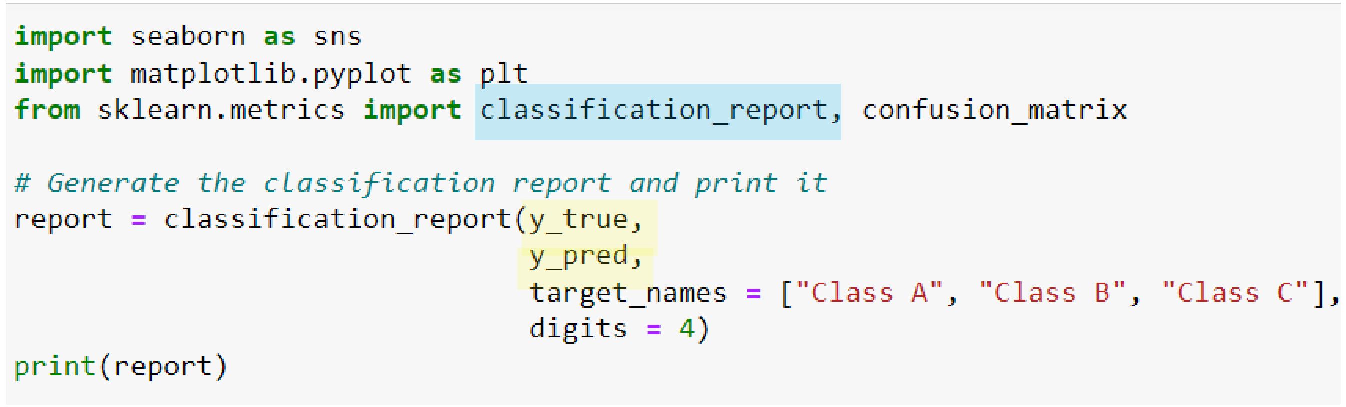
Let's consider a classification task with three classes:

- **Class A:** Label 0
- **Class B:** Label 1
- **Class C:** Label 2

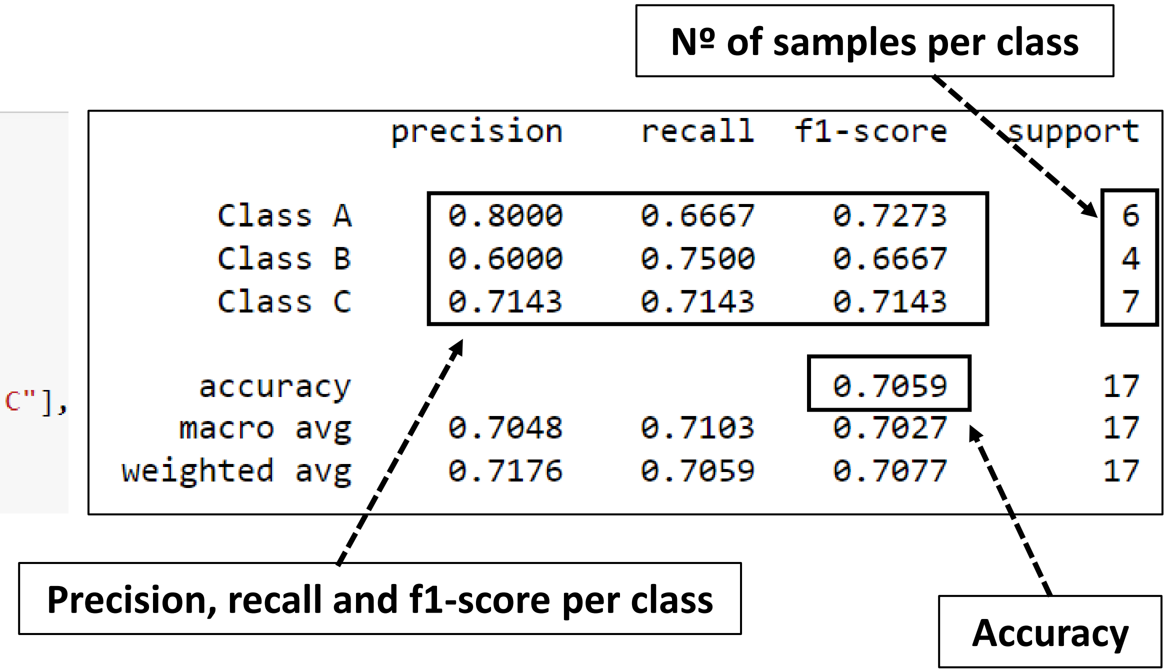
In our test dataset, let's suppose the true and predicted labels (obtained using the code from the previous slide) are:

```
y_true = [2,0,1,0,2,0,0,0,1,1,2,2,2,2,2,0,1]
y_pred = [2,1,1,0,2,2,0,0,1,2,1,2,0,2,2,0,1]
```

Using these results, we can easily compute the confusion matrix and the metrics we discussed earlier with just a few lines of code.



```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

# Generate the classification report and print it
report = classification_report(y_true,
                               y_pred,
                               target_names = ["Class A", "Class B", "Class C"],
                               digits = 4)
print(report)
```



```
              precision    recall  f1-score   support

     Class A     0.8000    0.6667    0.7273         6
     Class B     0.6000    0.7500    0.6667         4
     Class C     0.7143    0.7143    0.7143         7

    accuracy                         0.7059        17
   macro avg     0.7048    0.7103    0.7027        17
weighted avg     0.7176    0.7059    0.7077        17
```

2.2 Compute metrics

```
              precision    recall  f1-score   support

     Class A     0.8000    0.6667    0.7273         6
     Class B     0.6000    0.7500    0.6667         4
     Class C     0.7143    0.7143    0.7143         7

    accuracy                         0.7059        17
   macro avg     0.7048    0.7103    0.7027        17
weighted avg     0.7176    0.7059    0.7077        17
```

Metrics per class

Overall metrics calculated using the total number of samples

$$\frac{0.8 + 0.6 + 0.7143}{3}$$

It equally averages class-specific metrics, regardless of class size or dataset imbalance.

$$\frac{0.8 \cdot 6 + 0.6 \cdot 4 + 0.7143 \cdot 7}{17}$$

The contribution of each class to the average is weighted by the number of samples in that class.

```
y_true = [2,0,1,0,2,0,0,0,1,1,2,2,2,2,2,0,1]
y_pred = [2,1,1,0,2,2,0,0,1,2,1,2,0,2,2,0,1]
```

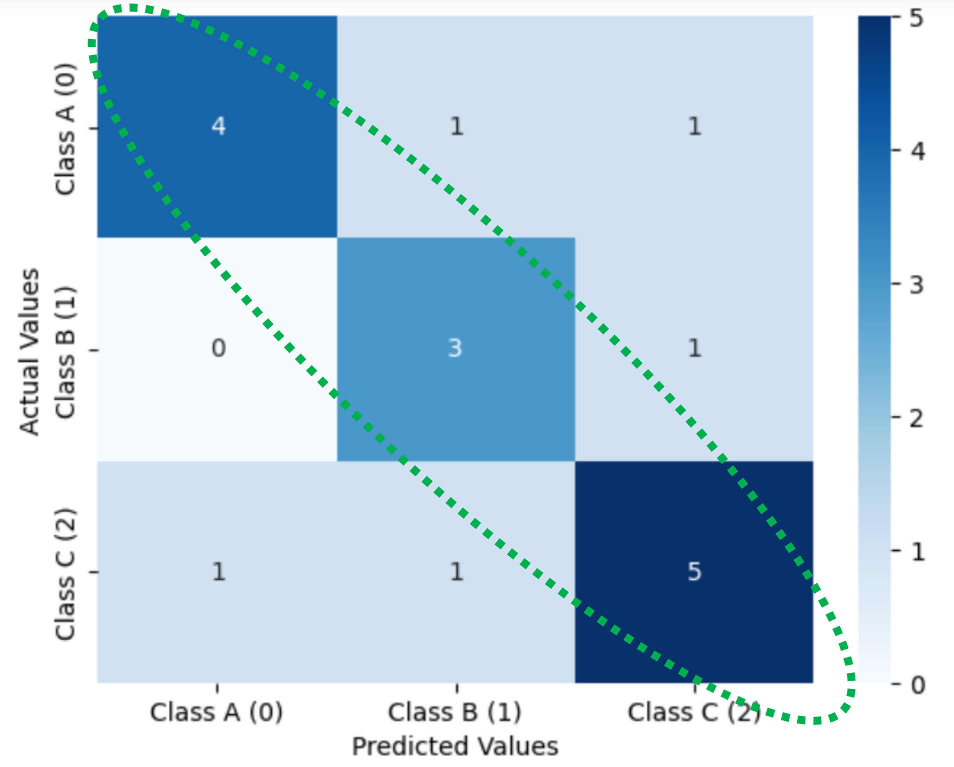
Class A: Label 0
Class B: Label 1
Class C: Label 2

2.2 Compute metrics: Confusion matrix

To visualize the confusion matrix, you can use the code:

```python
# Generate the confusion matrix
cmat = confusion_matrix(y_true, y_pred)

# Create a heatmap for the confusion matrix visualization
ax = sns.heatmap(cmat, annot = True, cmap = 'Blues')

# Set labels for x-axis and y-axis
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')

# Set the labels
ax.xaxis.set_ticklabels(['Class A (0)', 'Class B (1)', 'Class C (2)'])
ax.yaxis.set_ticklabels(['Class A (0)', 'Class B (1)', 'Class C (2)'])

# Display the visualization of the Confusion Matrix
plt.show()
```

Correct predictions

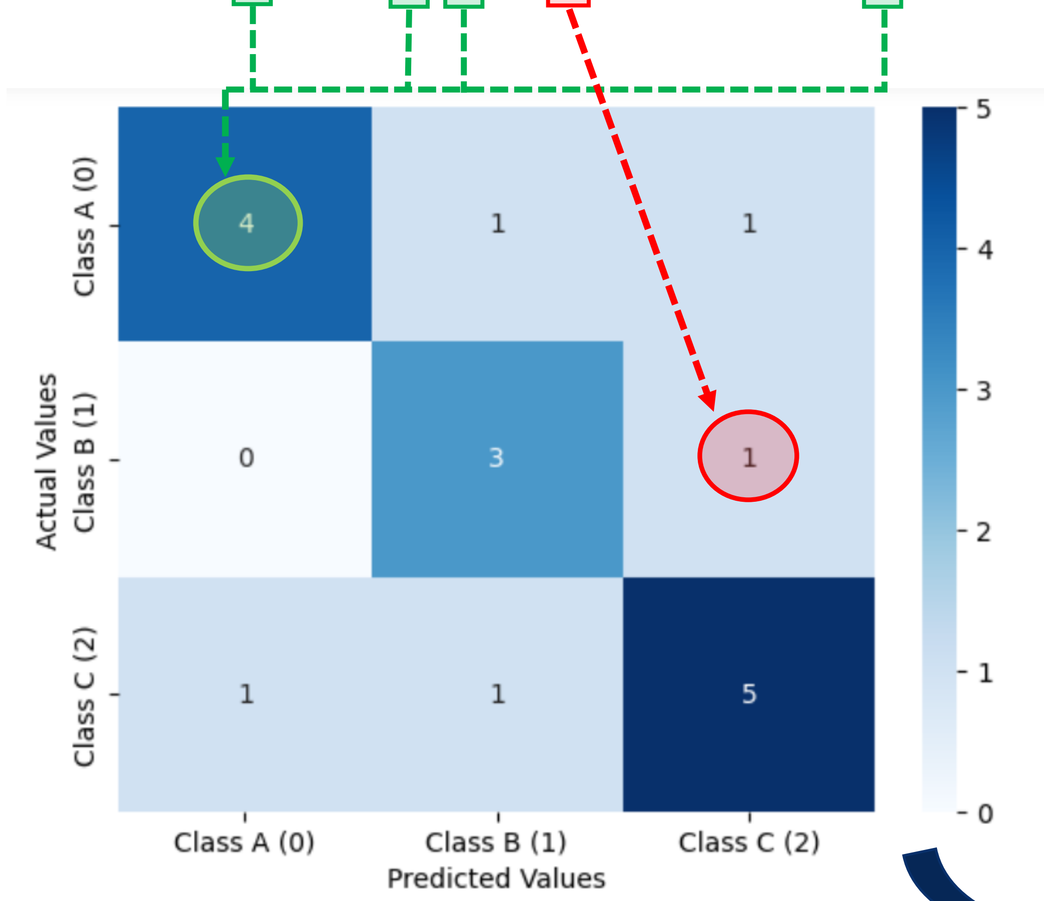


After computing the metrics, check and analyze the results to understand how well the model performs on the evaluation dataset.

2.2 Compute metrics: Confusion matrix

Let's take a closer look at how the model's successful predictions and errors are mirrored in the confusion matrix.

```
y_true = [2,0,1,0,2,0,0,0,1,1,2,2,2,2,2,0,1]
y_pred = [2,1,1,0,2,2,0,0,1,2,1,2,0,2,2,0,1]
```



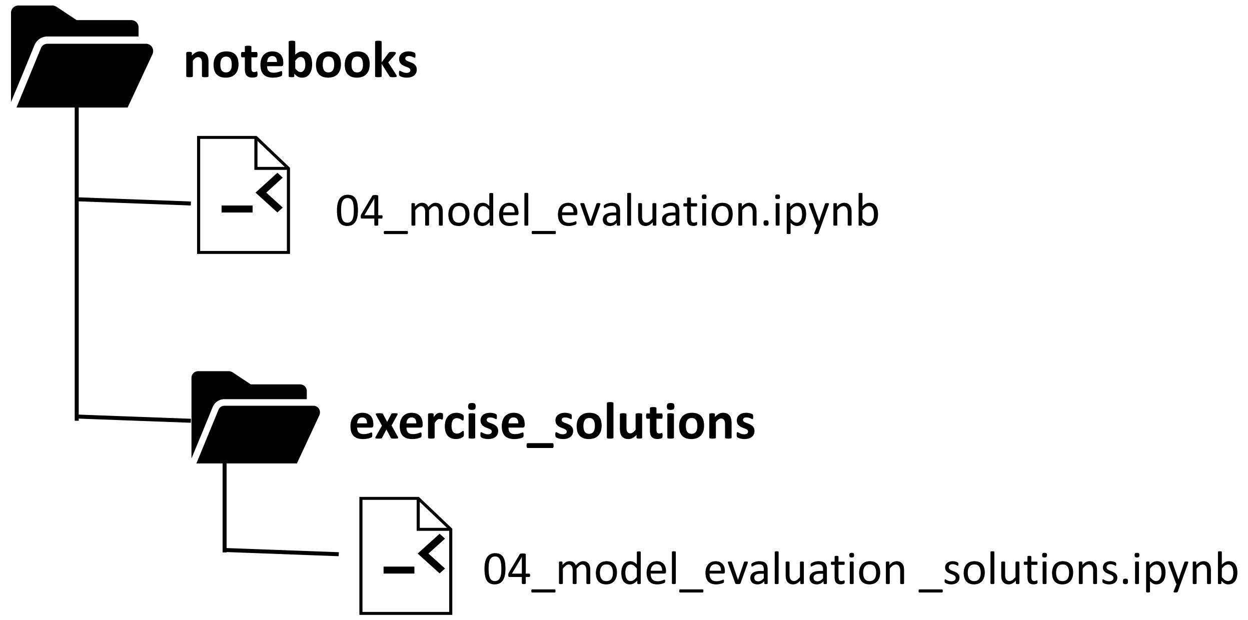
The confusion matrix for binary classification is a 2x2 matrix, as we showed a few slides before. For **multiclass classification**, we have an **NxN matrix,** where N represents the number of classes (3 in this case).

The correct predictions are represented by the diagonal elements. Each element (i,i) corresponds to the true positives for class i, indicating the number of instances correctly classified within that specific class.

2.3 Code examples

Let’s code!

In the upcoming notebook, our focus will be on evaluating the models trained in Section 03 using our testing dataset. We aim to gain a comprehensive understanding of their performance and generalization capabilities. By analyzing the evaluation results, we can draw valuable insights and identify areas for potential improvement. In addition, we will propose an exercise to help you learn how to obtain and interpret the metrics for a binary classification problem.


notebooks
04_model_evaluation.ipynb
exercise_solutions
04_model_evaluation _solutions.ipynb