Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU[1]

[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

Classification Error

One of the common metrics to measure the performance of a classifier is the **classification error**. Considering array of true class labels:

$$y\_true = [0, 0, 1, 0, 1, 1, 0, 0, 1, 1]$$

The prediction by our model:

$$y\_pred = [1, 0, 1, 0, 1, 0, 0, 0, 1, 1]$$

Classification Error

Dễ dàng nhận thấy, model phân loại sai 2 ví dụ

$$y\_true = [0, 0, 1, 0, 1, 1, 0, 0, 1, 1]$$

$$y\_pred = [1, 0, 1, 0, 1, 0, 0, 0, 1, 1]$$

Vậy nên, $\boxed{classification\ error = 2/10 = 0.2}$

Classification accuracy

Classification accuracy: Độ chính xác phân loại

The classification error and the classification accuracy are actually closely related. In fact, the accuracy is just one minus the error.

$$Accuracy = 1 - Error$$

$$y\_true = [0, 0, 1, 0, 1, 1, 0, 0, 1, 1]$$

$$y\_pred = [1, 0, 1, 0, 1, 0, 0, 0, 1, 1]$$

Màu xanh (blue) là số lượng ví dụ dự đoán đúng,

$$Accuracy = 8/10 = 0.8$$

Beware of Class Imbalance

There's *one issue of classification error and the classification accuracy* that **we have to be aware of and that is class imbalance**, which means that one class label is overrepresented in a data set. Let's take a look at another example:

$$y\_true = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1]$$

$$y\_pred = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Quan sát được rằng lớp 0 chiếm đến 8/10 các ví dụ cần dự đoán.

Since we have an imbalance problem, it's heavily skewed towards the majority class. So here the baseline would be 80% and the goal of the classification algorithm is of course being better than 80%.

Beyond 2 Classes

How can we evaluate the performance of models trained on more than two classes?

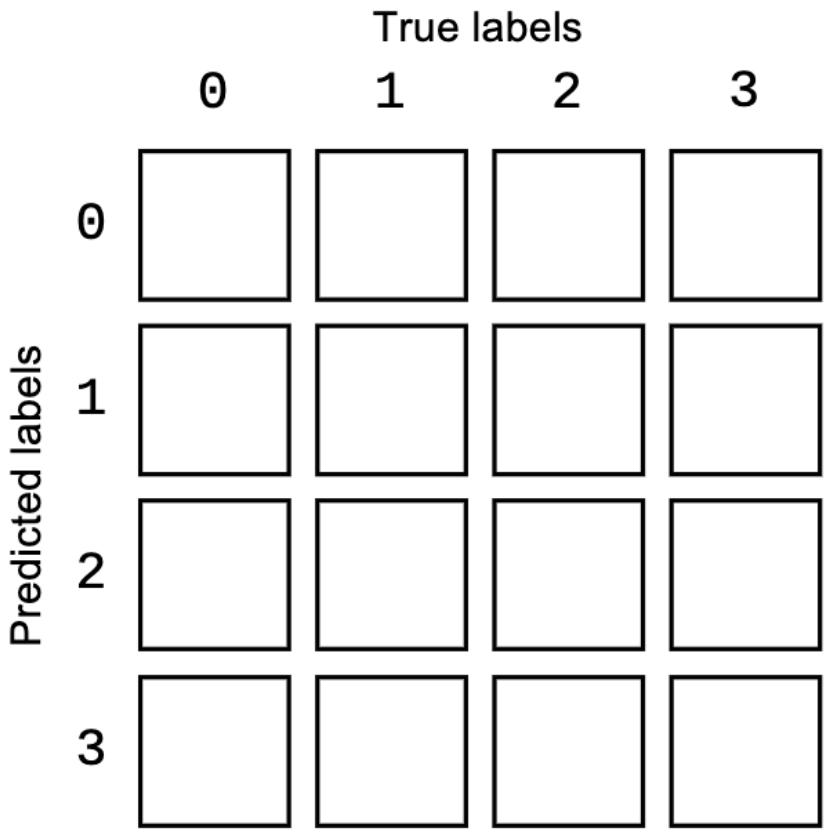
$$y\_true = [0, 2, 1, 3, 1, 3, 0, 0, 2, 1]$$

$$y\_pred = [1, 2, 1, 3, 1, 1, 0, 0, 2, 1]$$

we can just use the accuracy to count how many times the model makes a correct prediction.

Confusion Matrix

The confusion matrix is a matrix that shows the predicted labels and the true labels. So each cell counts how many times the predicted and the true label match.



Confusion Matrix

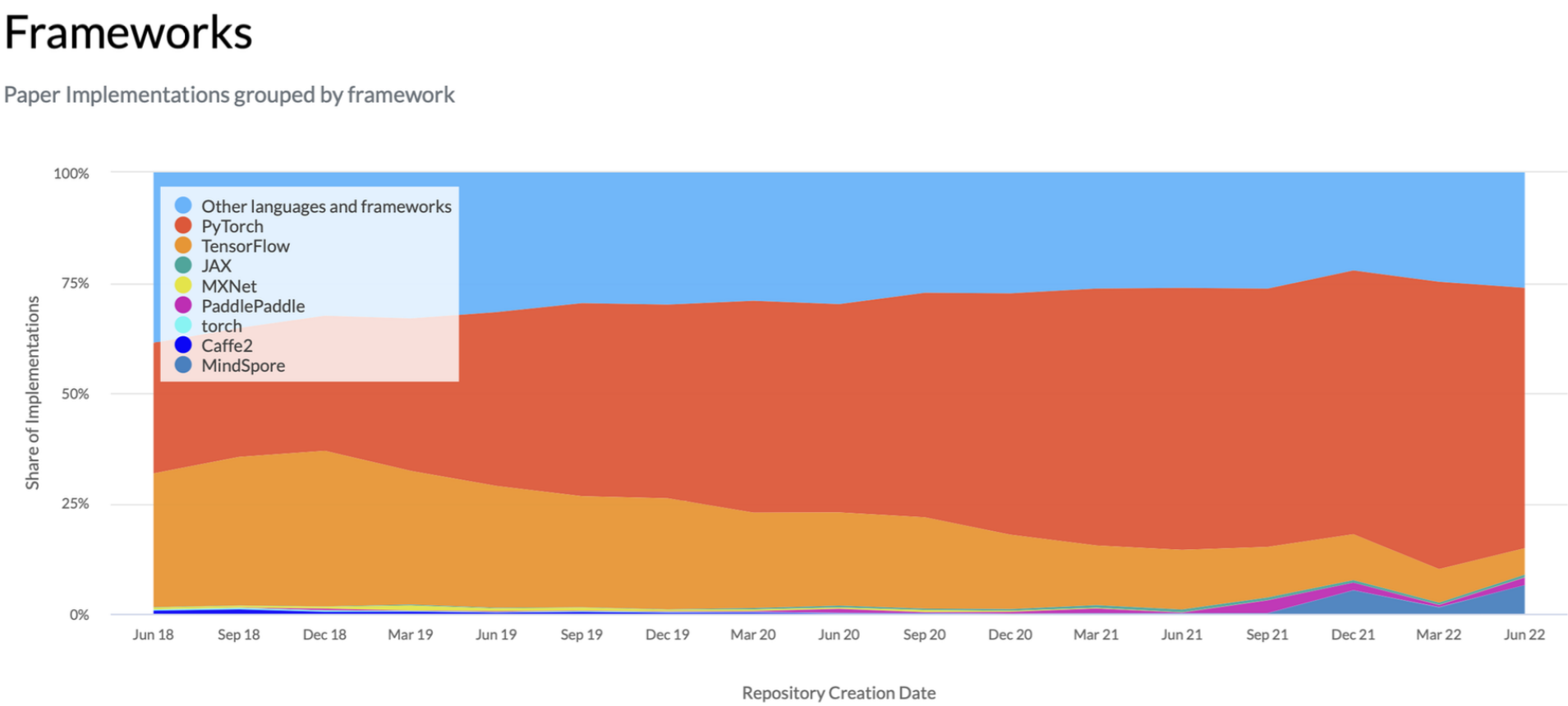
$$y\_true = [0, 2, 1, 3, 1, 3, 0, 0, 2, 1]$$
$$y\_pred = [1, 2, 1, 3, 1, 1, 0, 0, 2, 1]$$

1. Tensor library
2. Automatic differentiation engine

What is PyTorch?

3. Deep learning library

PyTorch is free and open-source software, which was written in Lua.



PyTorch is widely used

Tensor library

Tensors for Data

1. Mathematically: a generalization of vectors, matrices, etc. . .
2. Computationally: a data container for storing multi-dimensional arrays

Ví dụ về tensor:

Scalar(rank-0 tensor)

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In pure Python

```
1 a = 1.
2 print(a)
```

[1] ✓ 0.0s Python

```
1.0
```

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In PyTorch

```
1 import torch
2 a = torch.tensor(1.)
3 print(a)
4 print(a.shape)
```

[2] ✓ 0.0s Python

```
tensor(1.)
torch.Size([])
```

Tensor library

Vector(rank-1 tensor)

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In pure Python

```
1 a = [1., 2., 3.]
2 print(a)
3
4
```

[4] ✓ 0.0s Python

```
[1.0, 2.0, 3.0]
```

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In PyTorch

```
1 import torch
2 a = torch.tensor([1., 2., 3.])
3 print(a)
4 print(a.shape)
```

[3] ✓ 0.0s Python

```
tensor([1., 2., 3.])
torch.Size([3])
```

We can use the **shape attribute** to check the dimensionality or the rank of a tensor

Tensor library

Matrix(rank-2 tensor)

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In pure Python

```
1 a = [[1., 2., 3.],
2      [2., 3., 4.]]
3
4 print(a)
```

[10] ✓ 0.0s Python

```
[[1.0, 2.0, 3.0], [2.0, 3.0, 4.0]]
```

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In PyTorch

```
1 import torch
2 a = torch.tensor([[1., 2., 3.],
3                   [2., 3., 4.]])
4 print(a.shape)
```

[4] ✓ 0.0s Python

```
torch.Size([2, 3])
```

Tensor library

Matrix(rank-2 tensor)

Where have we seen a structure like this before?

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In pure Python

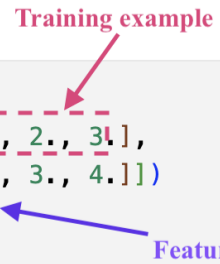
```
1 a = [[1., 2., 3.],
2      [2., 3., 4.]]
3
4 print(a)
```

[10] ✓ 0.0s Python

```
[[1.0, 2.0, 3.0], [2.0, 3.0, 4.0]]
```

+ Code + Markdown | Run All Restart ··· flask_appenv (Python 3.11.1)

In PyTorch




```
1 import torch
2 a = torch.tensor([[1., 2., 3.],
3                  [2., 3., 4.]])
4 print(a.shape)
```

[4] ✓ 0.0s Python

```
torch.Size([2, 3])
```

Actually, we can think of these rows of the matrix as our training examples. And the columns would represent our features in the data set.

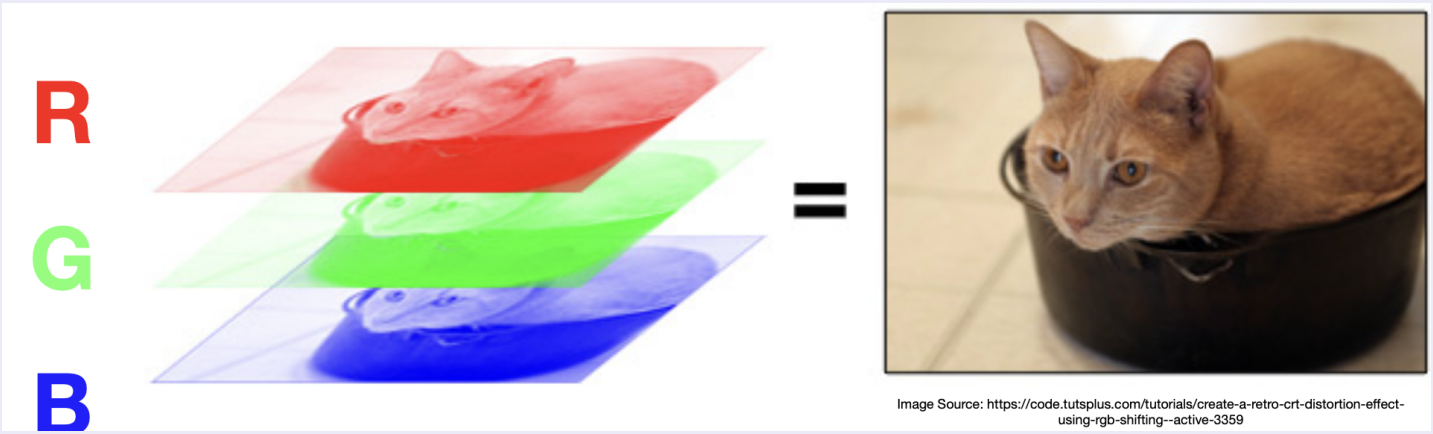
Tensor library

Image

We might think of an image as a matrix, where the rows and the columns represent the pixels in the data set. Color image as a stack of matrices.

RGB Image

RGB image stands for three color channels, a red channel, a green channel, and a blue channel. So each color channel represents one matrix.



Tensor library

3D tensor(rank-3 tensor)

A stack of multiple matrices.

```
1 import torch
2 a = torch.tensor([[[1., 2., 3.],
3                   [2., 3., 4.]],
4                  [[1., 2., 3.],
5                   [2., 3., 4.]]])
6 print(a.shape)
```

[7] ✓ 0.0s Python

```
torch.Size([2, 2, 3])
```

Tensor library

4D tensor(rank-4 tensor)

A stack of multiple color images.

```
[12]
1  import torch
2  a = torch.tensor([[[1., 2., 3.],
3                    [2., 3., 4.]],
4                   [[1., 2., 3.],
5                    [2., 3., 4.]],
6                   [[1., 2., 3.],
7                    [2., 3., 4.]]])
8  b = torch.stack((a,a,a))
9  print(b.shape)
```

✓ 0.0s Python

```
torch.Size([3, 3, 2, 3])
```

Tensors and Array Libraries

How tensor libraries differ from array libraries?

TorchTensor is almost identical to NumPy.array

$$torch.tensor \approx numpy.array$$

Tuy nhiên, Tensor library hỗ trợ tính toán:

- GPU support
- automatic differentiation support

Tensors and Array Libraries

So sánh Python List và Tensor/Array

So sánh ưu điểm (+), nhược điểm (-):

List	Tensor/Array
+ Can store heterogeneous types (mix str, float, etc.)	- All elements have to have the same type (e.g., int, float)
+ Elements can be easily added or removed	- Can't add or remove elements
- Numerical computations are slow	+ Numerical computations are fast.

Top 10 tensor functions and methods in PyTorch

1. Creating Tensors

```
1  import torch
2  m = torch.tensor([[1., 2., 3.],
3                    [4., 5., 6.]])
4  print(m)
5  # -> tensor([[1., 2., 3.],
6  #            [4., 5., 6.]])
```

2. Checking the Shape

Using the dot **shape attribute**, we can check the number of elements in the tensor.

```
1  import torch
2  m = torch.tensor([[1., 2., 3.],
3                    [4., 5., 6.]])
4  print(m.shape)
5  # -> torch.Size([2, 3])
```

3. Checking the Rank / Number of Dimensions

```
1  import torch
2  m = torch.tensor([[1., 2., 3.],
3                    [4., 5., 6.]])
4  print(m.ndim)
5  # -> 2
```

4. Checking the Data Type

We can check the **type of data stored in tensor by using *.dtype* atribute.**

```
import torch
m = torch.tensor([[1., 2., 3.],
                  [4., 5., 6.]])
print(m.dtype)
# -> torch.float32
# So the 32 here refers to 32 bit precision.
```

Ví dụ các phần tử có kiểu dữ liệu số nguyên trong Pytorch.

```
import torch
m = torch.tensor([[1, 2, 3],
                  [4, 5, 6]])
print(m.dtype)
# -> torch.int64
```

5. Creating a Tensor From NumPy Arrays

Chúng ta có thể **chuyển đổi trực tiếp mảng trong NumPy thành Tensor tại cùng một địa chỉ bộ nhớ bằng phương thức *torch.from_numpy()***.

```
1 import numpy as np
2 import torch
3 np_ary = np.array([1., 2., 3.])
4 m2 = torch.from_numpy(np_ary)
5 print(m2) # -> tensor([1., 2., 3.], dtype=torch.float64)
```

Hoặc chúng ta có thể gọi phương thức ***.tensor()*** để chuyển đổi. **Tuy nhiên nó sẽ tạo ra một bản sao trong bộ nhớ**.

```
1 import numpy as np
2 import torch
3 np_ary = np.array([1., 2., 3.])
4 m2 = torch.tensor(np_ary)
5 print(m2) # -> tensor([1., 2., 3.], dtype=torch.float64)
```

Chú ý: độ chính xác mặc định của numpy là 64bit.

6. Changing Data Types

Chúng ta có thể **chuyển đổi kiểu dữ liệu dễ dàng bằng phương thức *.to(kiểu dữ liệu mới)***

```
1 import numpy as np
2 import torch
3 np_ary = np.array([1., 2., 3.])
4 m2 = torch.from_numpy(np_ary)
5 m2 = m2.to(tensor.float32)
6 print(m2.dtype) # -> torch.float32
```

7. Checking the Device Type

Tensors also have a ***.device* attribute** that show us where on our computer the tensor is located.

```
import numpy as np
import torch
np_ary = np.array([1., 2., 3.])
m2 = torch.from_numpy(np_ary)
print(m2.device) # -> device(type='cpu')
#return CPU, which means that the tensor is on the CPU's memory.
```

Lưu ý: trong DL chúng ta sẽ cần chuyển Tensor về xử lý trên bộ nhớ của GPU.

8. Changing the Tensor Shape

Chúng ta có thể **thay đổi shape của Tensor** (ví dụ như flip the rows and columns) bằng cách **sử dụng phương thức *.view()***

```
1  import torch
2  m = torch.tensor([[1., 2., 3.],
3                    [4., 5., 6.]])
4  print(m.view(3,2))
5  # -> tensor([[1., 2.],
6  #            [3., 4.],
7  #            [5., 6.]])
```

Lưu ý: .view() sẽ không thay đổi trực tiếp tensor ban đầu mà tạo ra vùng nhớ copy.

8. Changing the Tensor Shape

When we use the *.view()*, there's also this magic placeholder called *minus one*. So if we drop in a minus one, this **dimension will be determined automatically**.

```
import torch
m = torch.tensor([[1., 2., 3.],
                  [4., 5., 6.]])
print(m.view(-1,2))
print(m.view(3,-1))
# -> tensor([[1., 2.],
#            [3., 4.],
#            [5., 6.]])
```

Chuyển về rank-1 tensor:

```
print(m.view(-1))
# -> tensor([1., 2., 3., 4., 5., 6.])
```

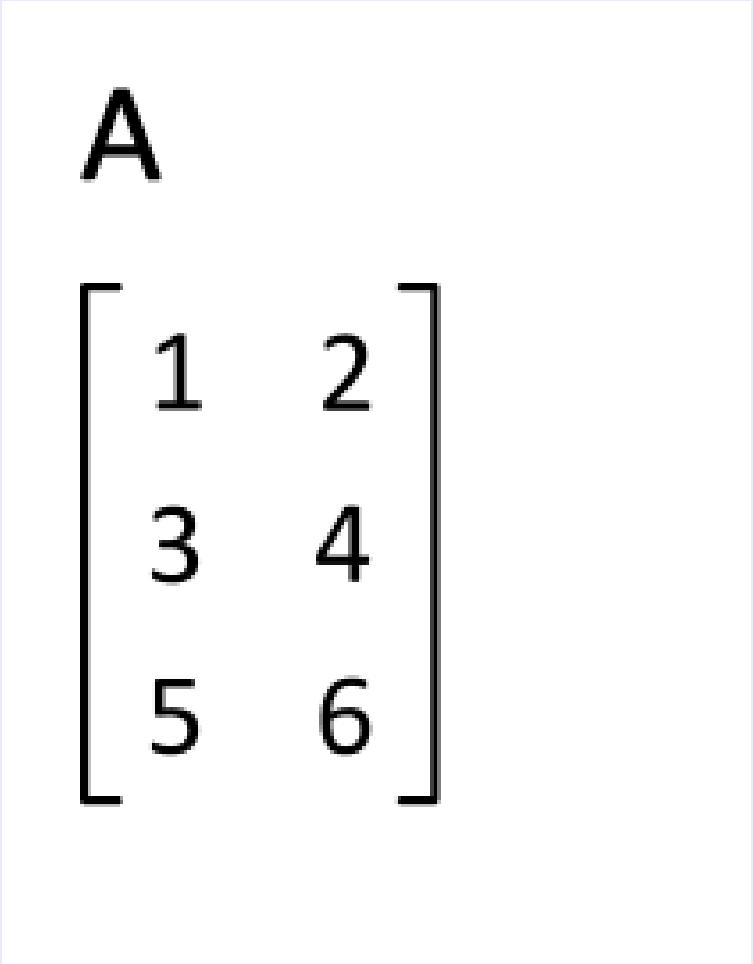
9. Transposing a Tensor

Chuyển vị của một Tensor thường xuyên được sử dụng trong DL để thực hiện các phép nhân Tensor. Ý nghĩa tương tự như chuyển vị của một ma trận.

```
1  import torch
2  m = torch.tensor([[1., 2., 3.],
3                    [4., 5., 6.]])
4  print(m.T)
5  # -> tensor([[1., 4.],
6  #            [2., 5.],
7  #            [3., 6.]])
```

*Lưu ý: **.T** sẽ không thay đổi trực tiếp tensor ban đầu mà tạo ra vùng nhớ copy.*

9. Transposing a Tensor



10. Multiplying Matrices

Nhân ma trận/Tensor tương tự như trong đại số tuyến tính là công việc thực hiện rất nhiều trong DL nhằm giúp cho chương trình nhanh, gọn và hiệu quả hơn. Để nhân ta sử dụng phương thức *torch.matmul()* trong PyTorch:

```
1  import torch
2  m = torch.tensor([[1., 2., 3.],
3                    [4., 5., 6.]])
4  m_T = m.T
5  print(torch.matmul(m,m_T))
6  # - > tensor([[14., 32.],
7  #             [32., 77.]])
```

Tài liệu tham khảo

Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.

Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions and Answers on Machine Learning and AI (2024). ISBN-13: 978-1-7185-0377-9 (ebook).

LightningAI
LightningAI: PyTorch Lightning (2024) .