Aprendizado de Máquina e Reconhecimento de Padrões 2021.2

Evaluation Metrics for Classification

Prof. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

CC BY NC SA

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas

Measuring Performance

When designing **a new model/solution** for a given problem, we desire to know about its **quality**: how **good** is your model?

The common way to do that is to calculate some **evaluation metric** that **quantitatively** expresses the **model's quality.**

Measuring Performance

When designing **a new model/solution** for a given problem, we desire to know about its **quality**: how **good** is your model?

The common way to do that is to calculate some **evaluation metric** that **quantitatively** expresses the **model's quality.**

Performance

- How **good** is our model?
- How does it compare to others?

Measuring Performance

When designing **a new model/solution** for a given problem, we desire to know about its **quality**: how **good** is your model?

The common way to do that is to calculate some **evaluation metric** that **quantitatively** expresses the **model's quality.**

Performance

- How **good** is our model?
- How does it compare to others?

A good evaluation metric should:

- **Quantity** the **quality** of the model (output)
- Be directly comparable
- Do not be biased

Basic Definition

Positive

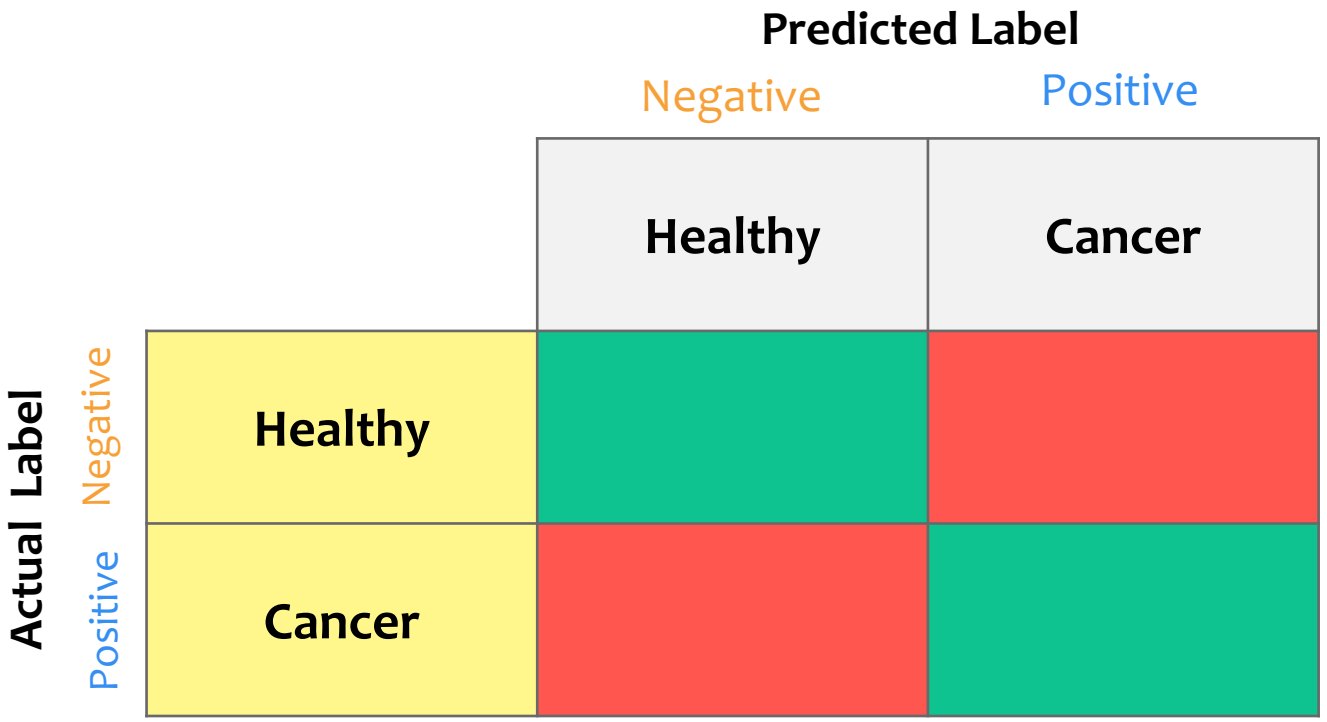
- The class/label of interest

Negative

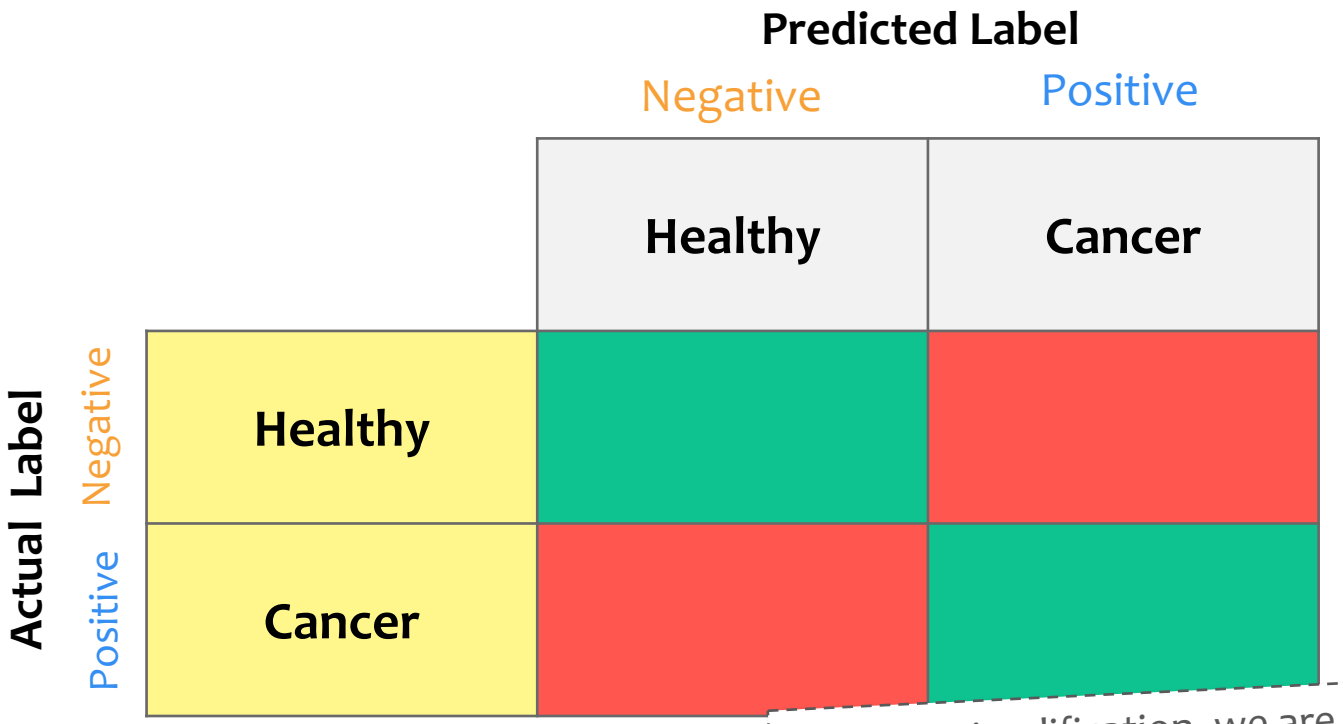
- The other classes/labels.

Confusion Matrix

Confusion Matrix

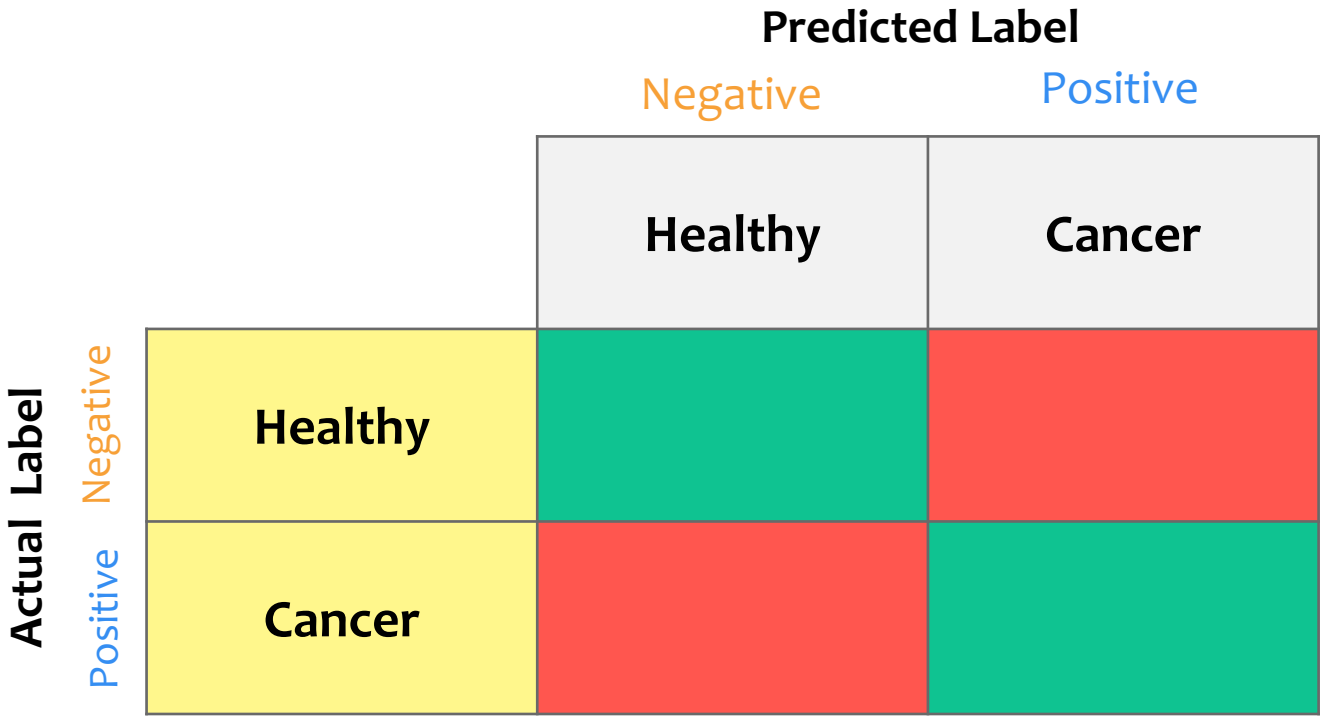


Confusion Matrix

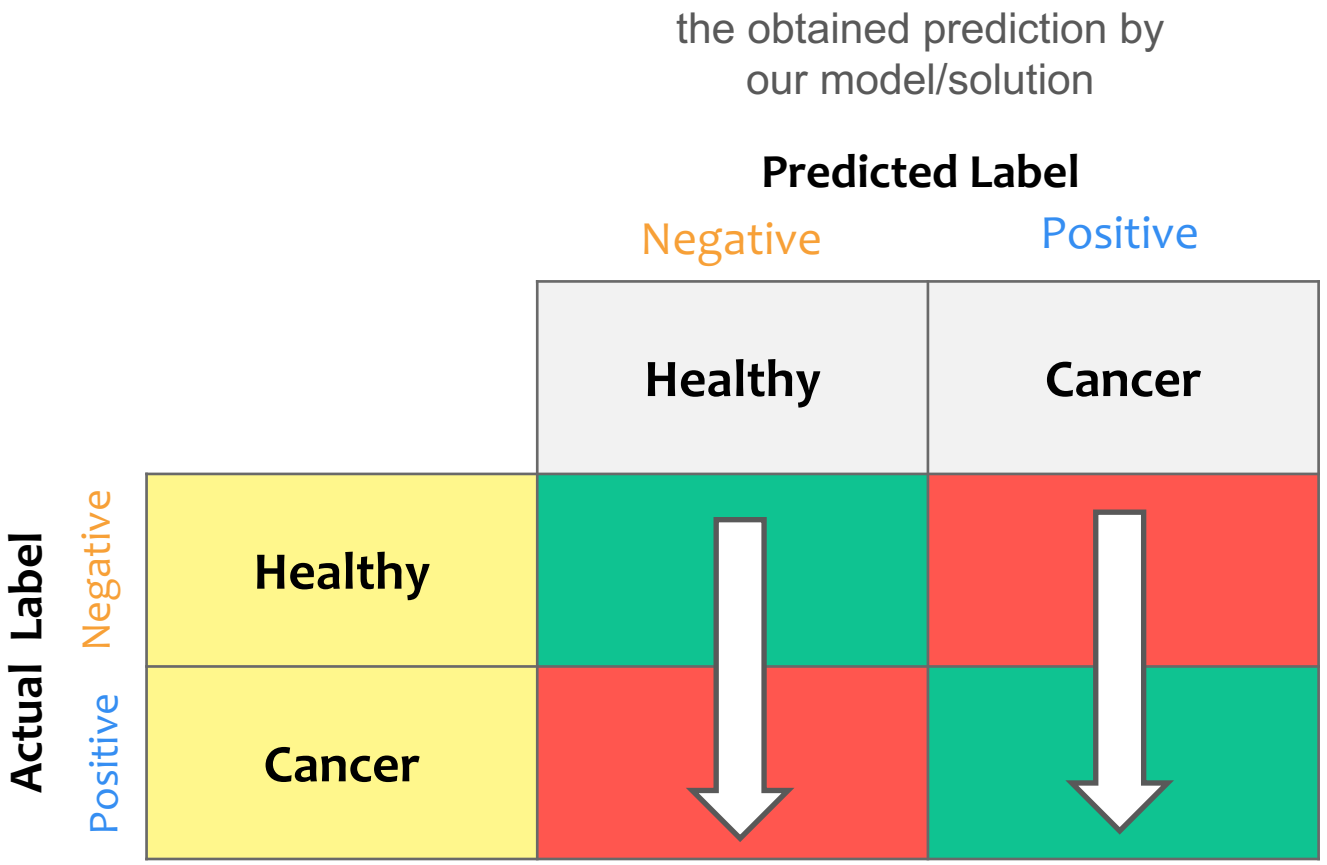


For simplification, we are considering **'healthy'** as a person that **does not have cancer**.
But note that this person may have other diseases, so that **(s)he is not necessarily healthy**.

Confusion Matrix

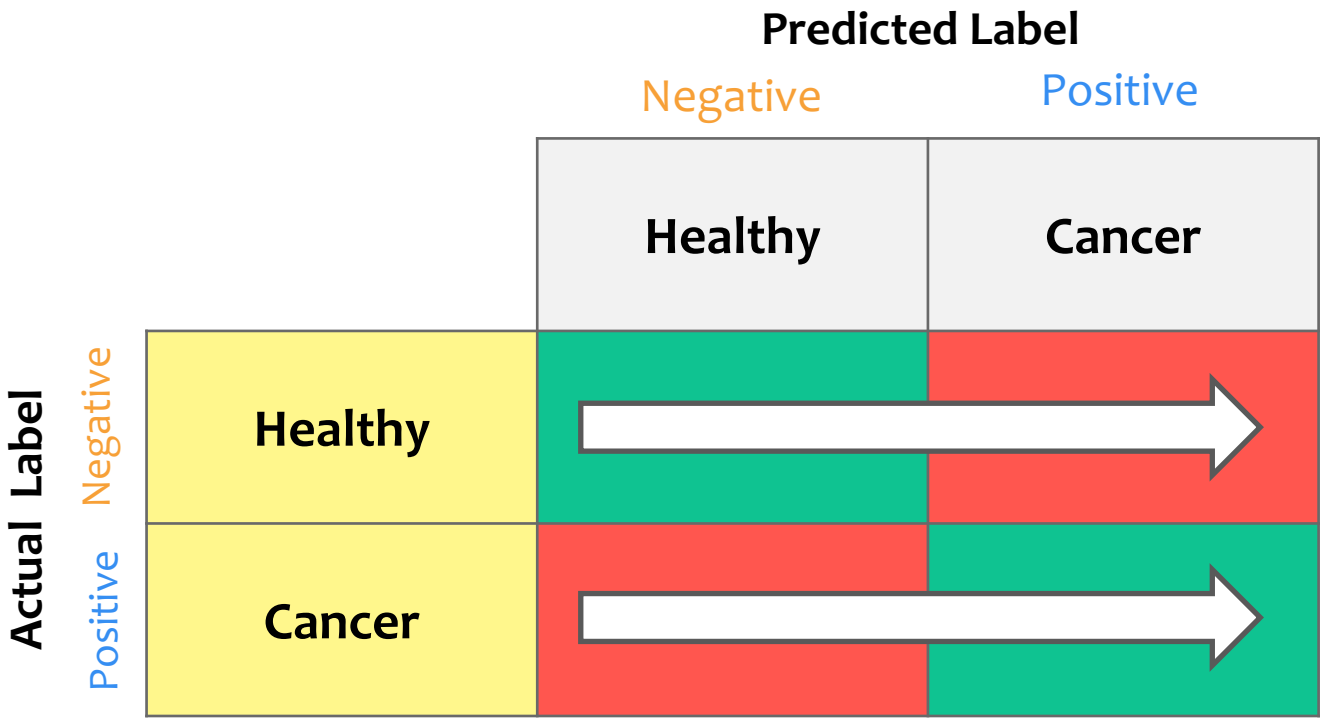


Confusion Matrix

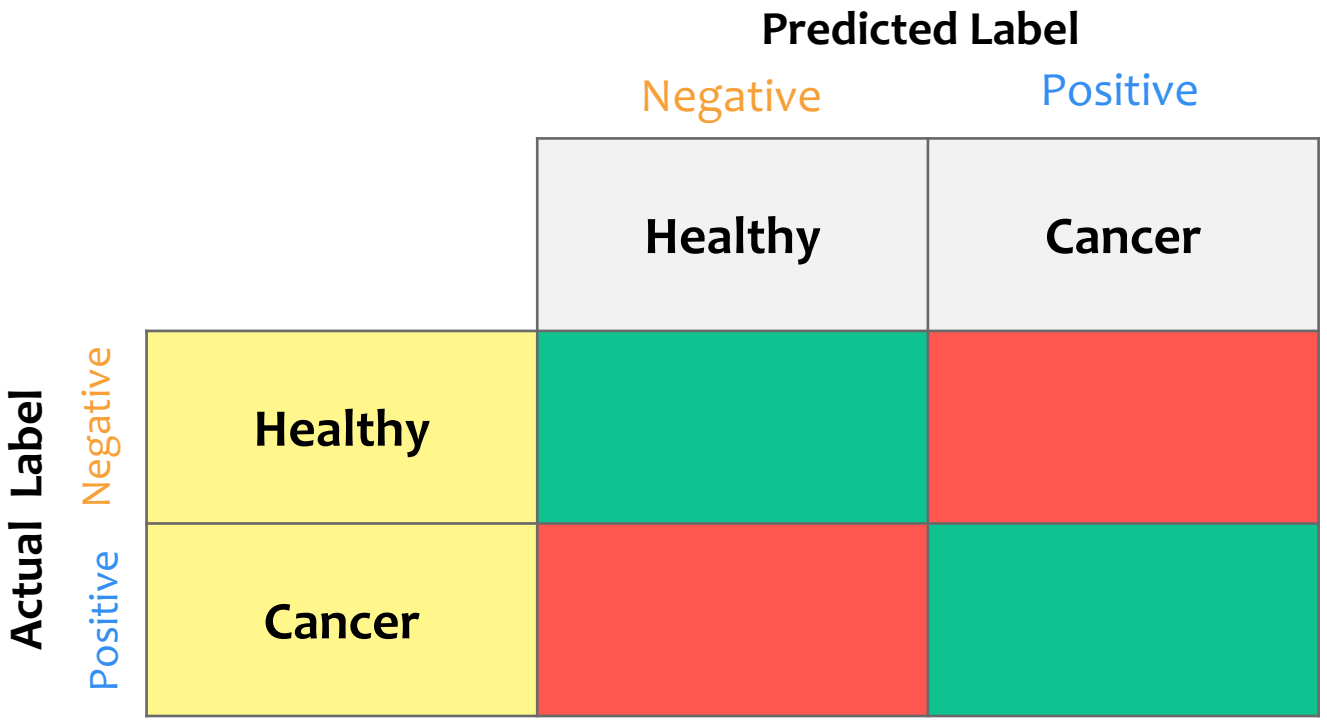


Confusion Matrix

known ground-truth

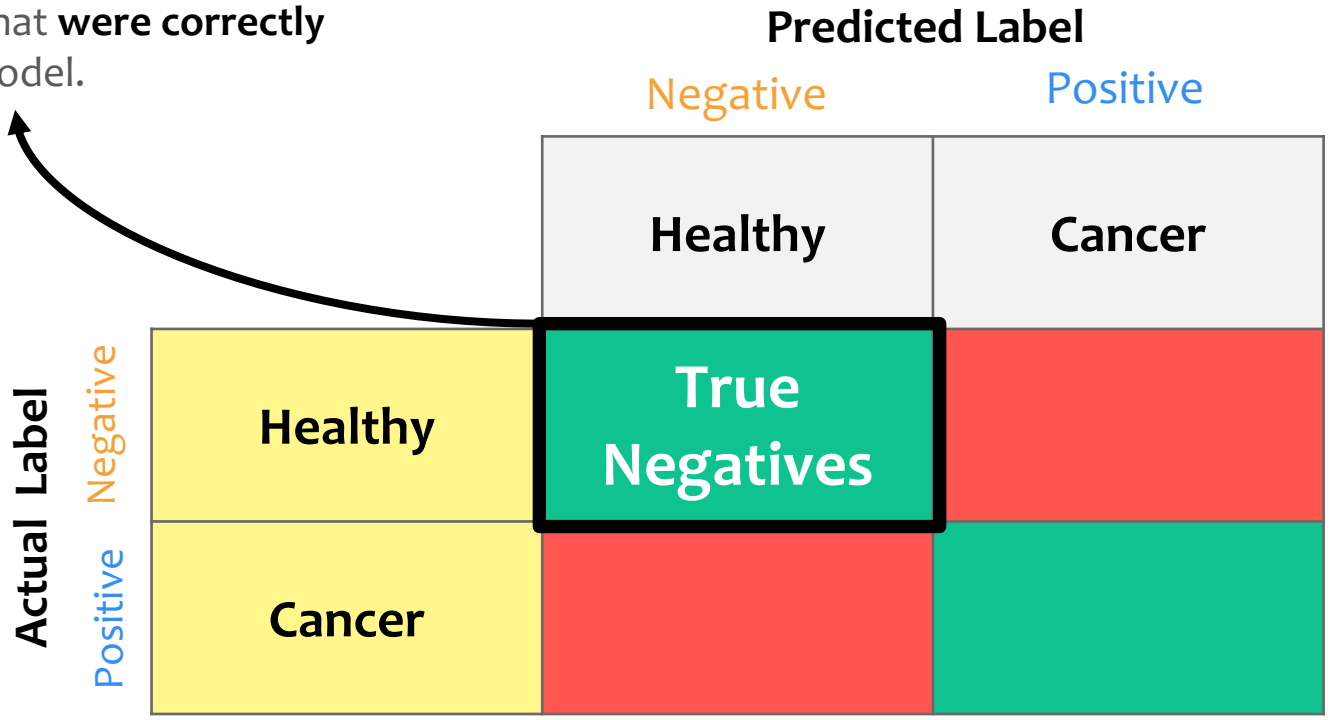


Confusion Matrix

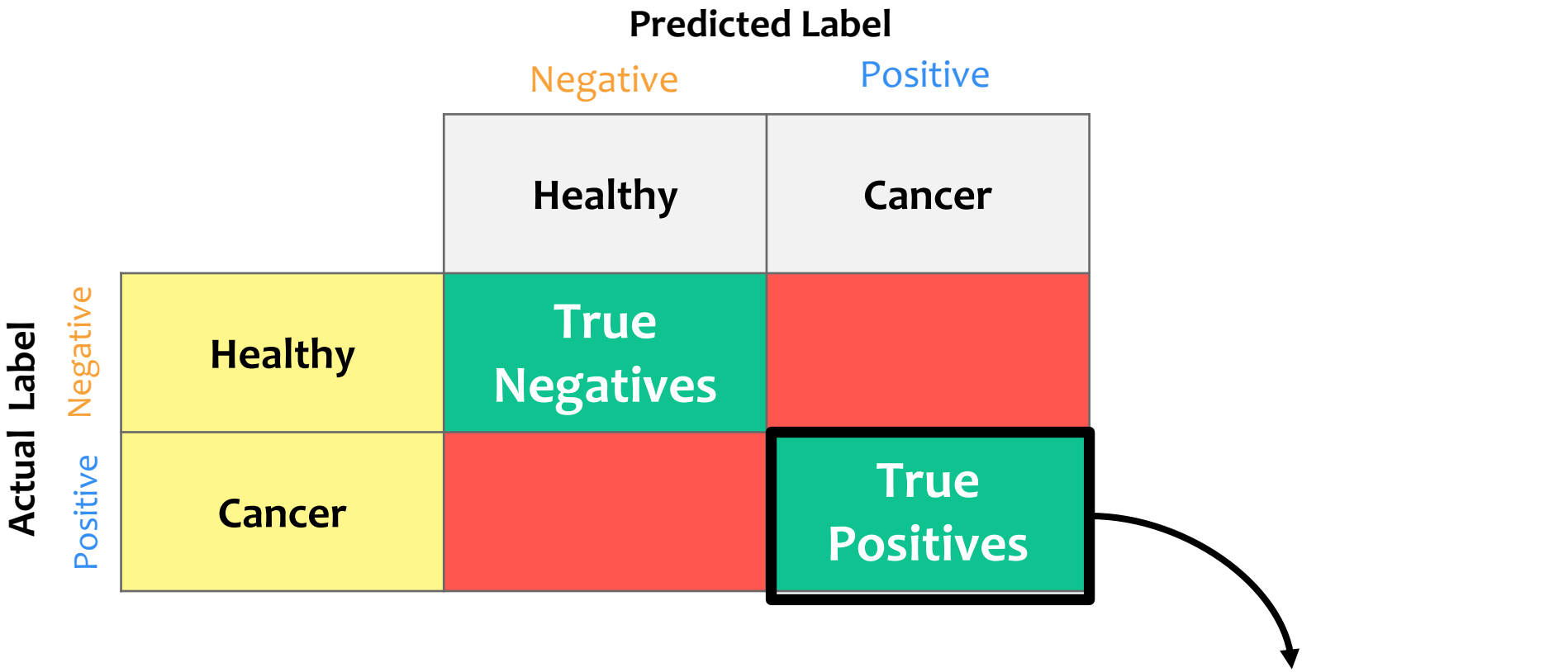


Confusion Matrix

The **healthy patients** (those that **do not have cancer**) that **were correctly classified** by the model.

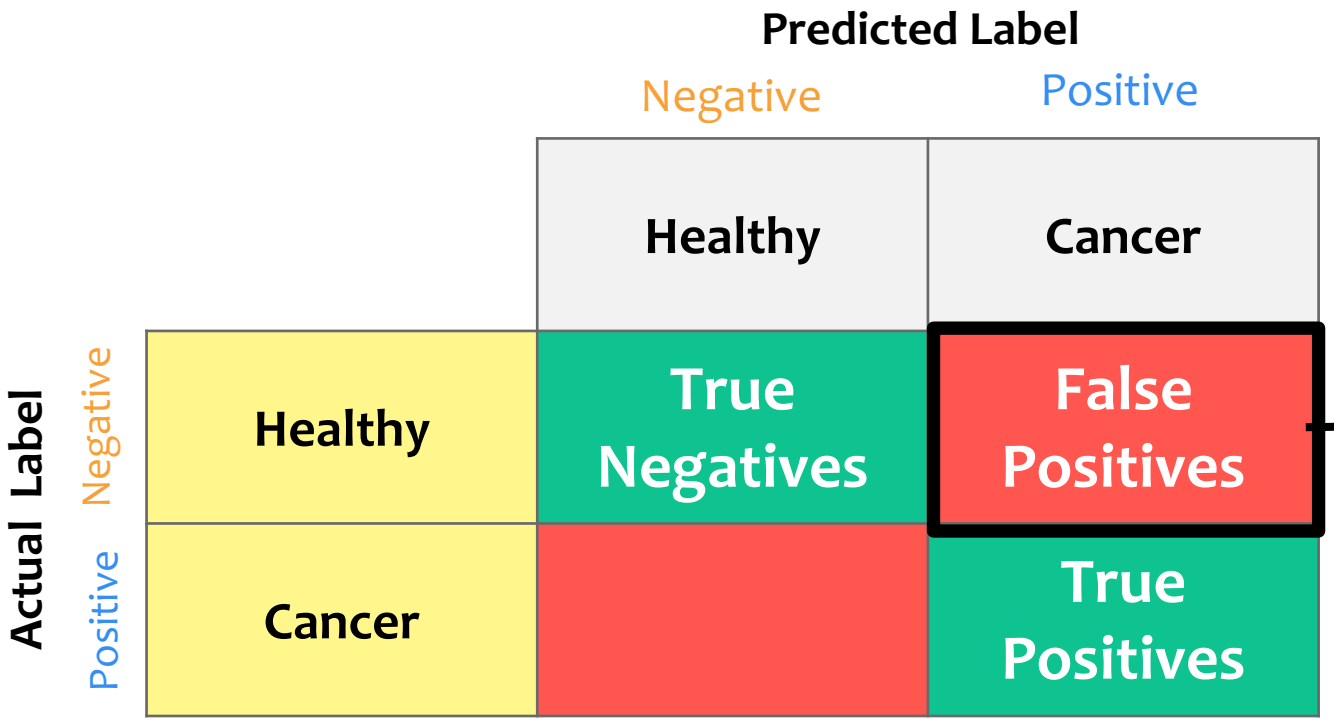


Confusion Matrix



The patients that **had cancer** that **were correctly classified** by the model.

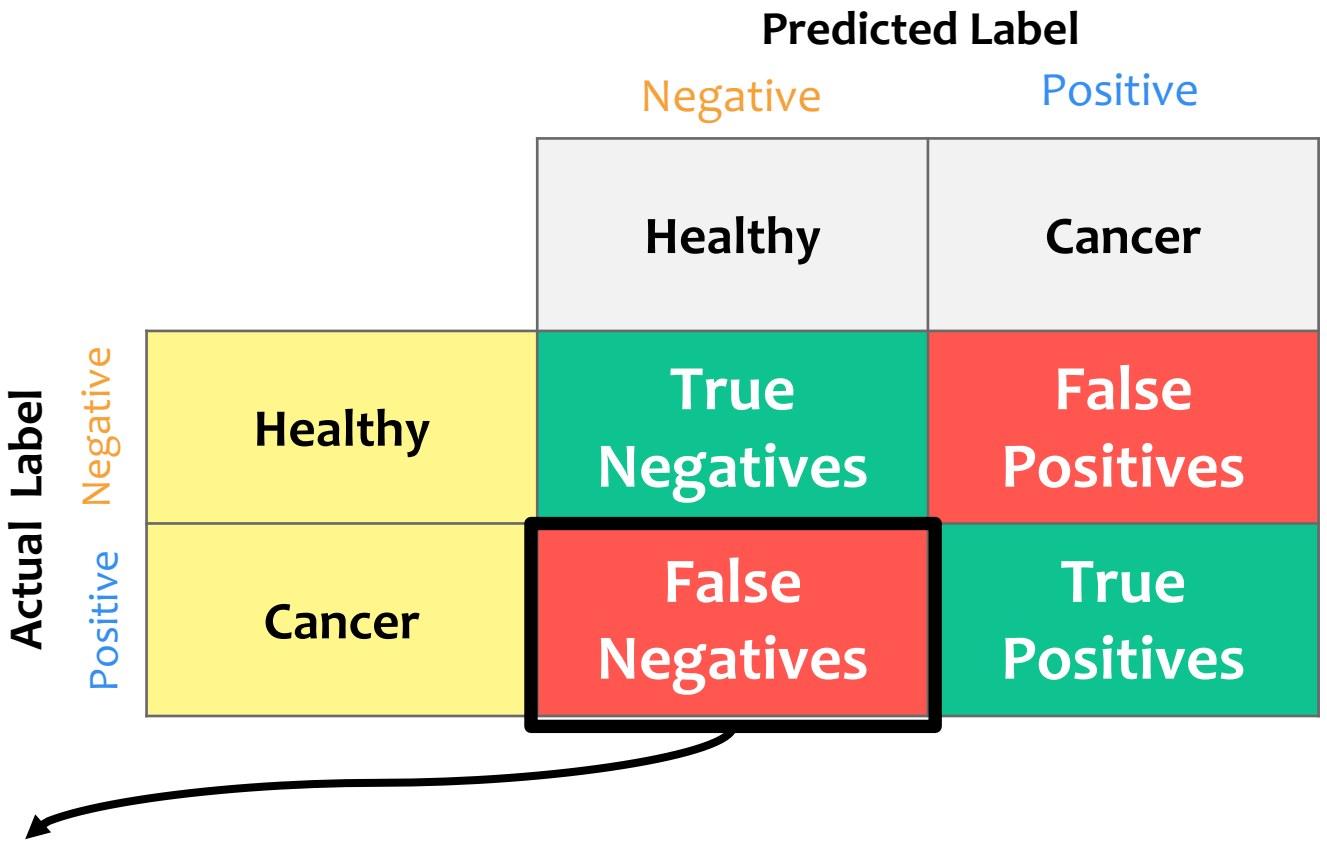
Confusion Matrix



Also called **Type I error** or **false alarm**

The healthy patients (those that **do not have cancer**) but the model **incorrectly** said **they have**.

Confusion Matrix



The patients that **have cancer** but the model **incorrectly** said they **did not**.

→ Also called **Type II error** or **miss**

Type I Error
(False Positive)

Type II Error
(False Negative)

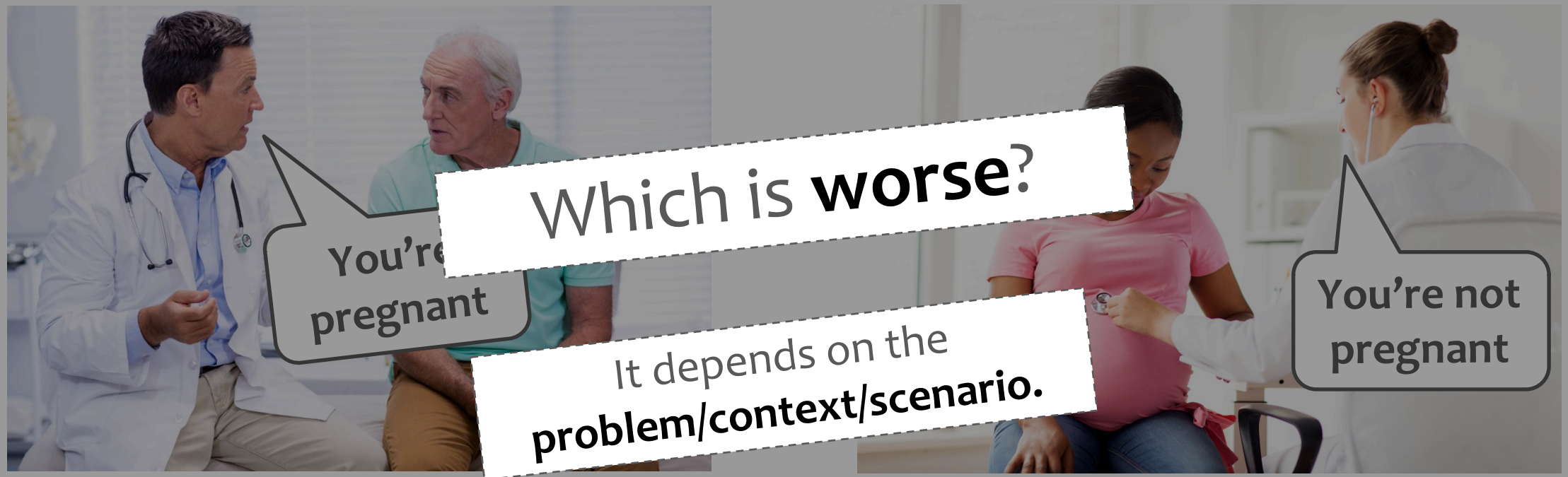
Type I Error
(False Positive)

Type II Error
(False Negative)

Which is **worse**?

Type I Error
(False Positive)

Type II Error
(False Negative)



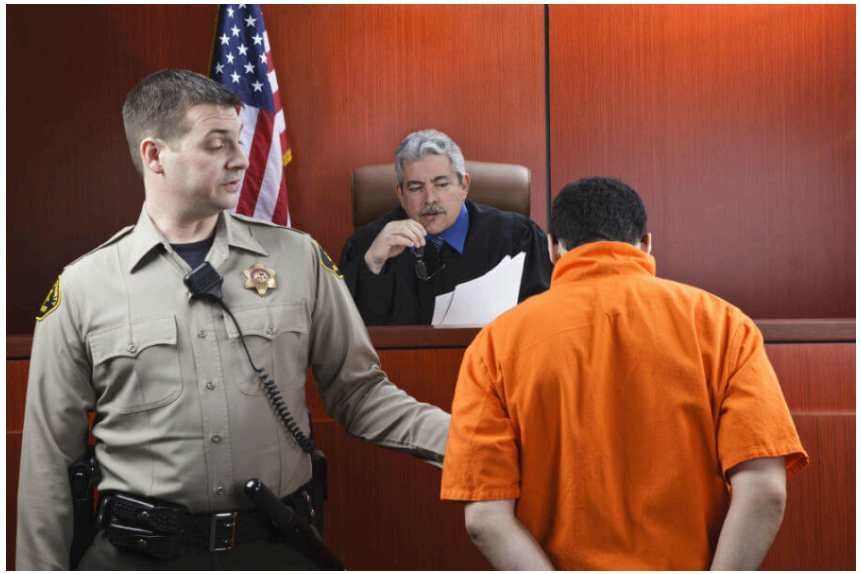
Which is **worse**?

It depends on the **problem/context/scenario.**

A scenario where…

A scenario where…

Type I Error is worse
(False Positive)

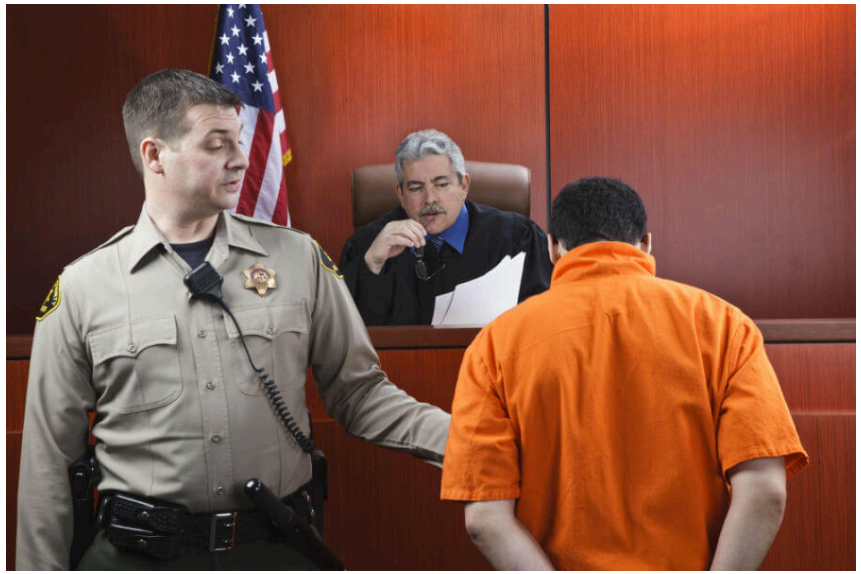


An **innocent (negative)** on trial found **guilty (positive)**.

The judicial system seeks **to minimize Type I errors** ***presumption of innocence***

A scenario where…

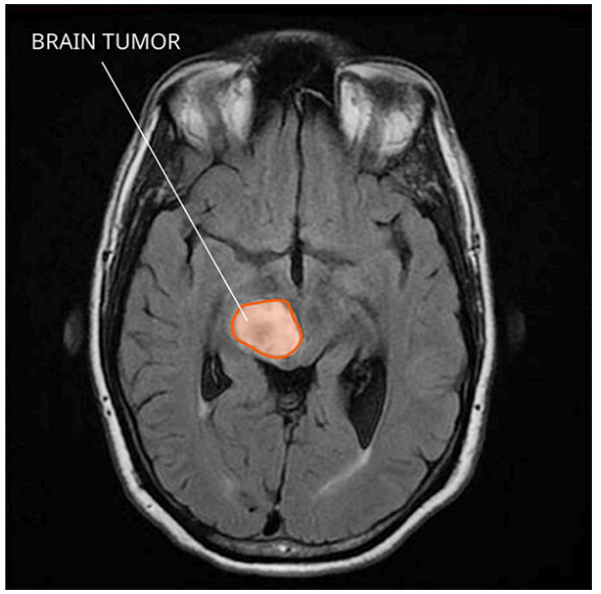
Type I Error is worse
(False Positive)



An **innocent (negative)** on trial found **guilty (positive)**.

The judicial system seeks **to minimize Type I errors**
presumption of innocence

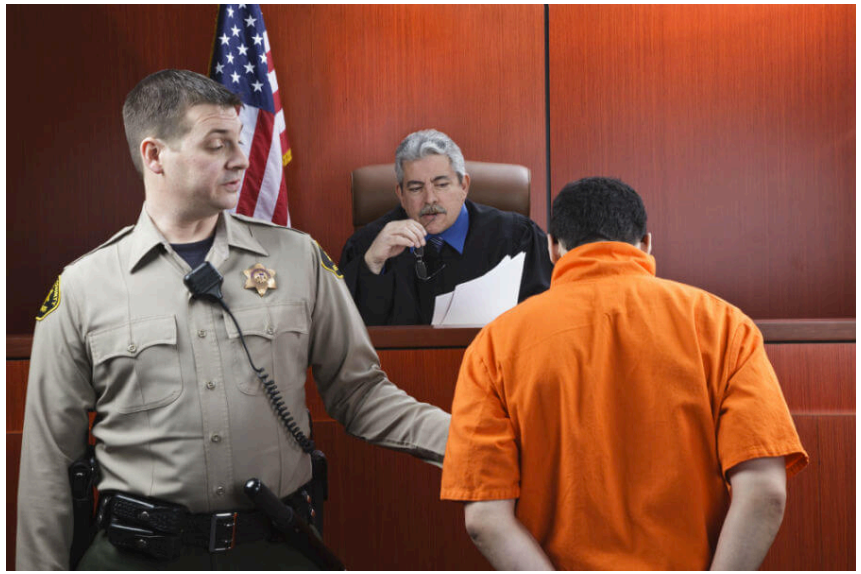
Type II Error is worse
(False Negative)



A medical screening incorrectly states that a **cancer patient (positive)** is **healthy (negative)**.

A scenario where...

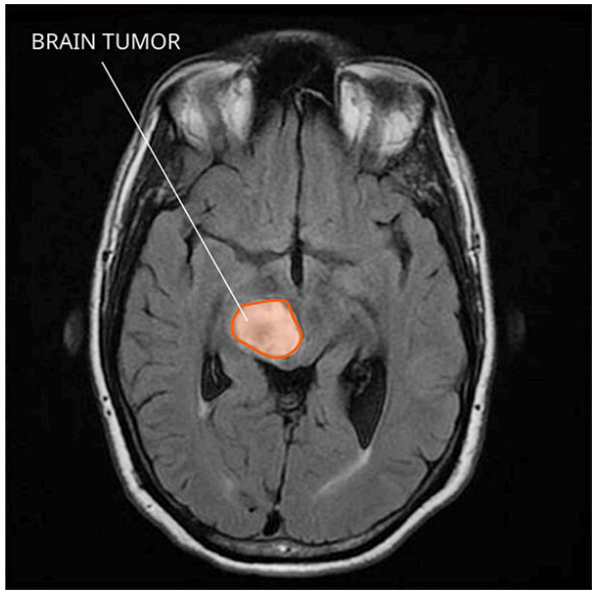
Type I Error is worse
(False Positive)



An **innocent (negative)** on trial found **guilty (positive)**.

The judicial system seeks **to minimize Type I errors**
presumption of innocence

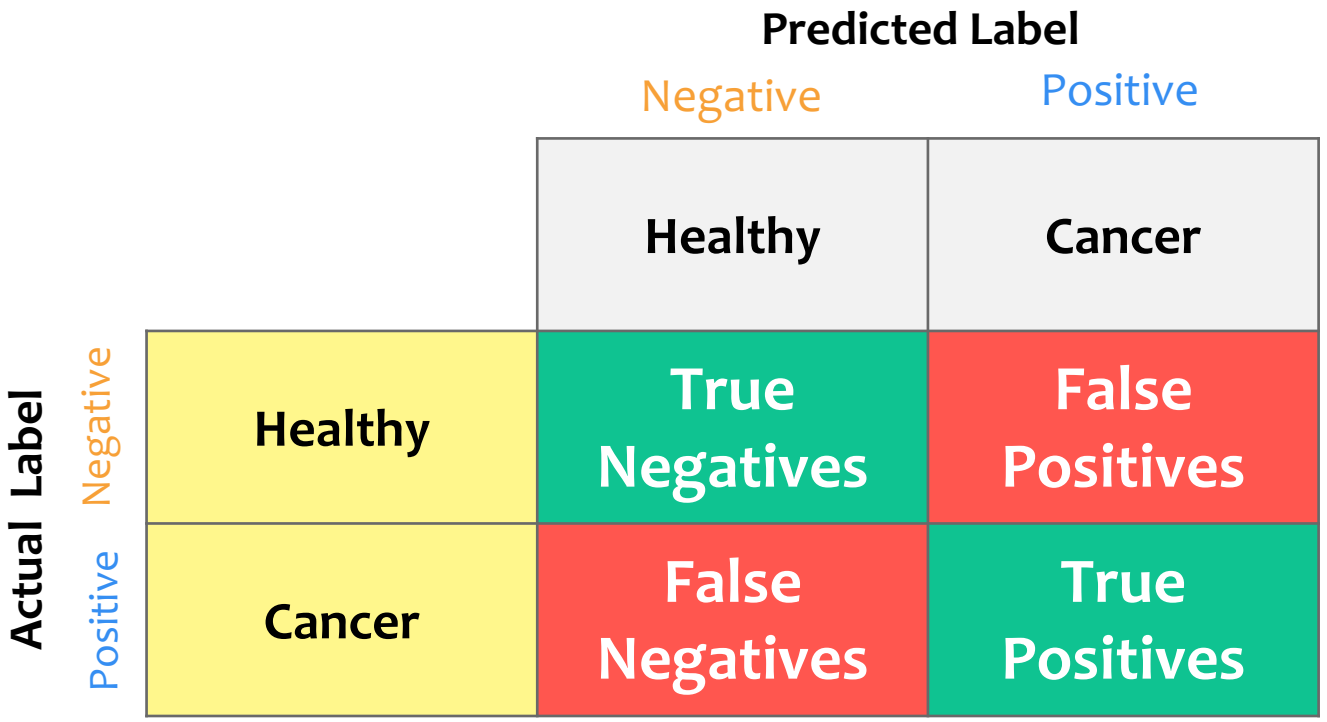
Type II Error is worse
(False Negative)



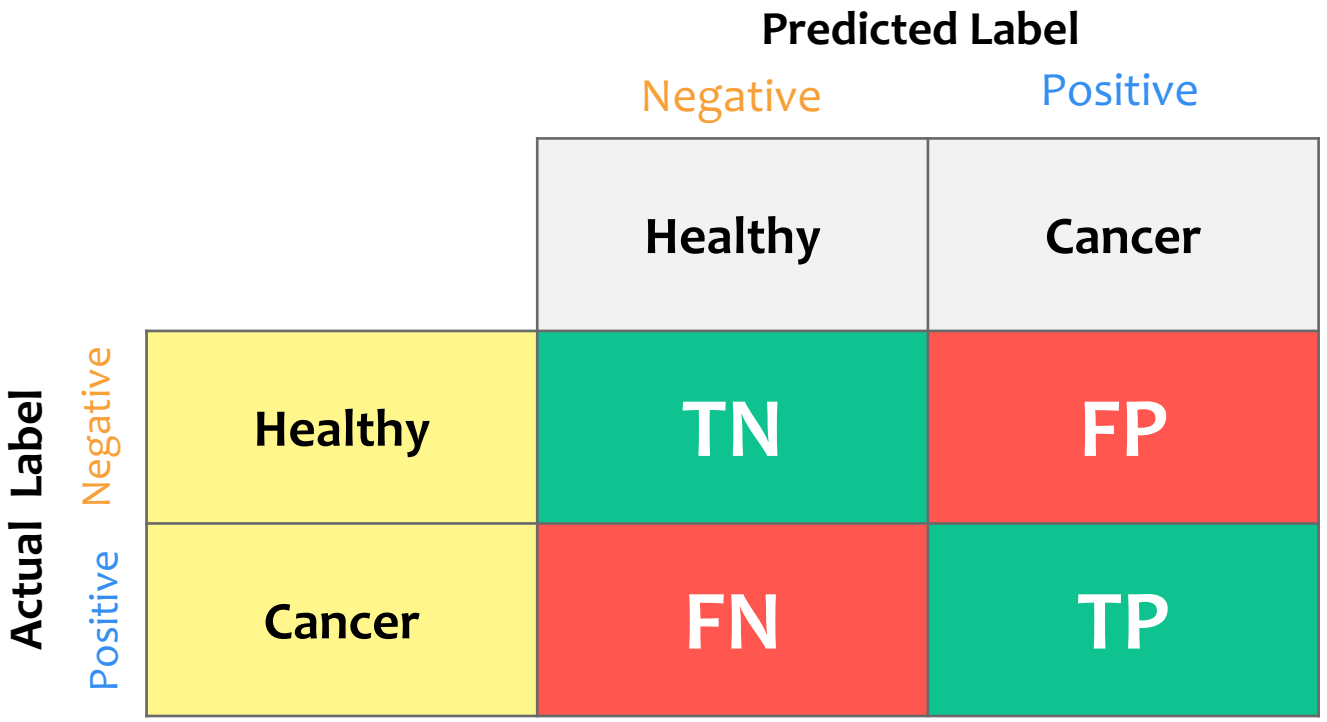
A medical screening incorrectly states that a **cancer patient (positive)** is **healthy (negative)**.

Type II error seems to be **worse** for most problems.

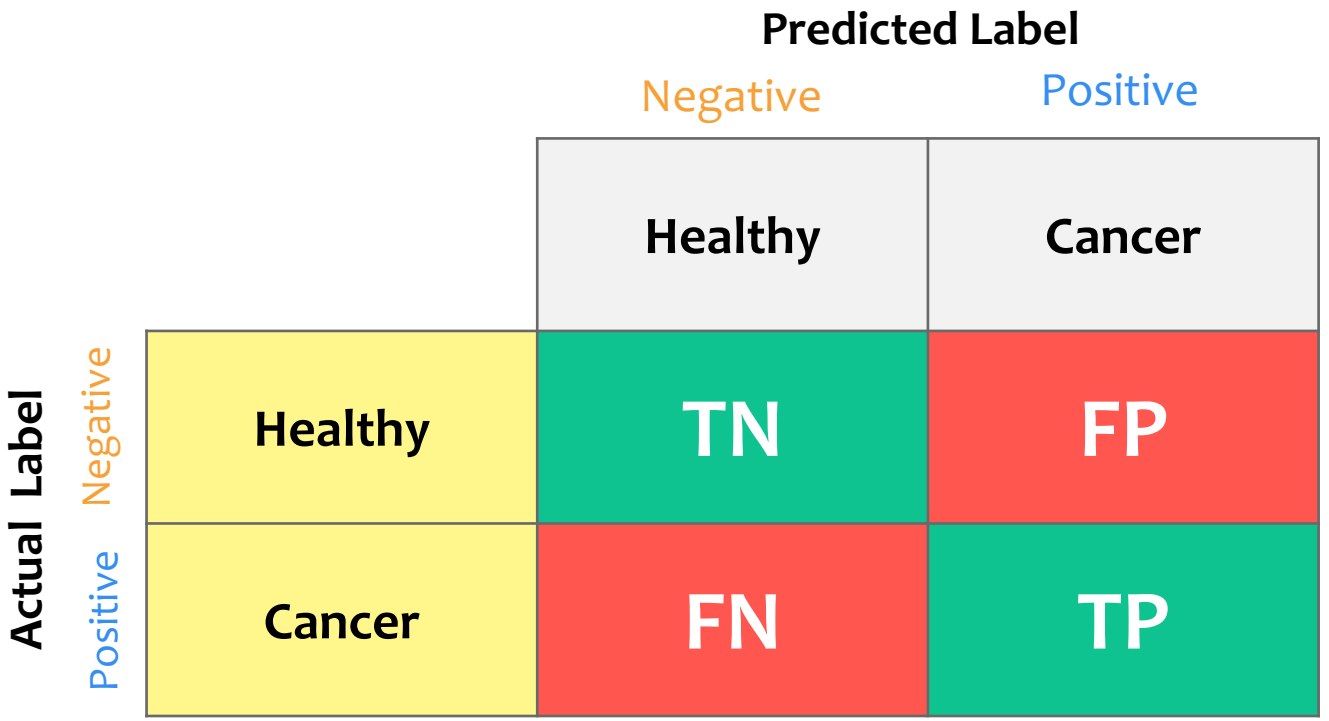
Confusion Matrix



Confusion Matrix



Confusion Matrix

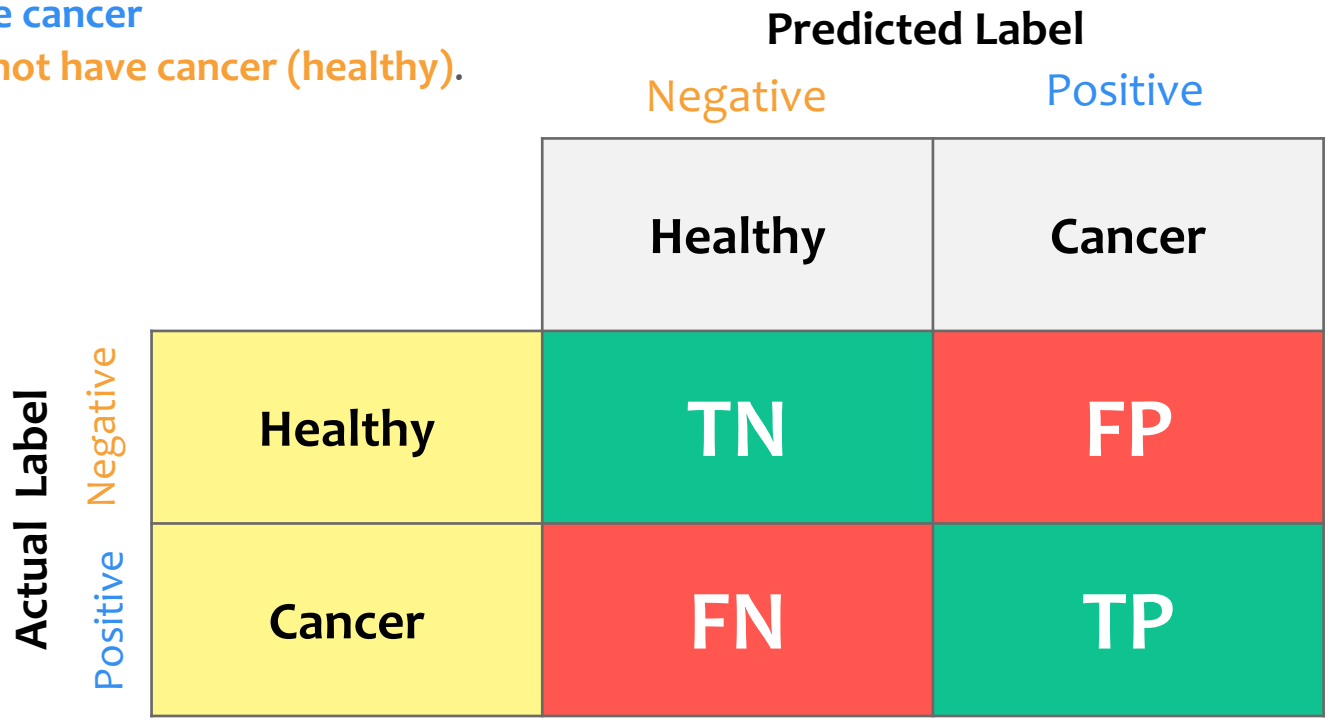


You may see this matrix with columns and rows in a different order.

Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.

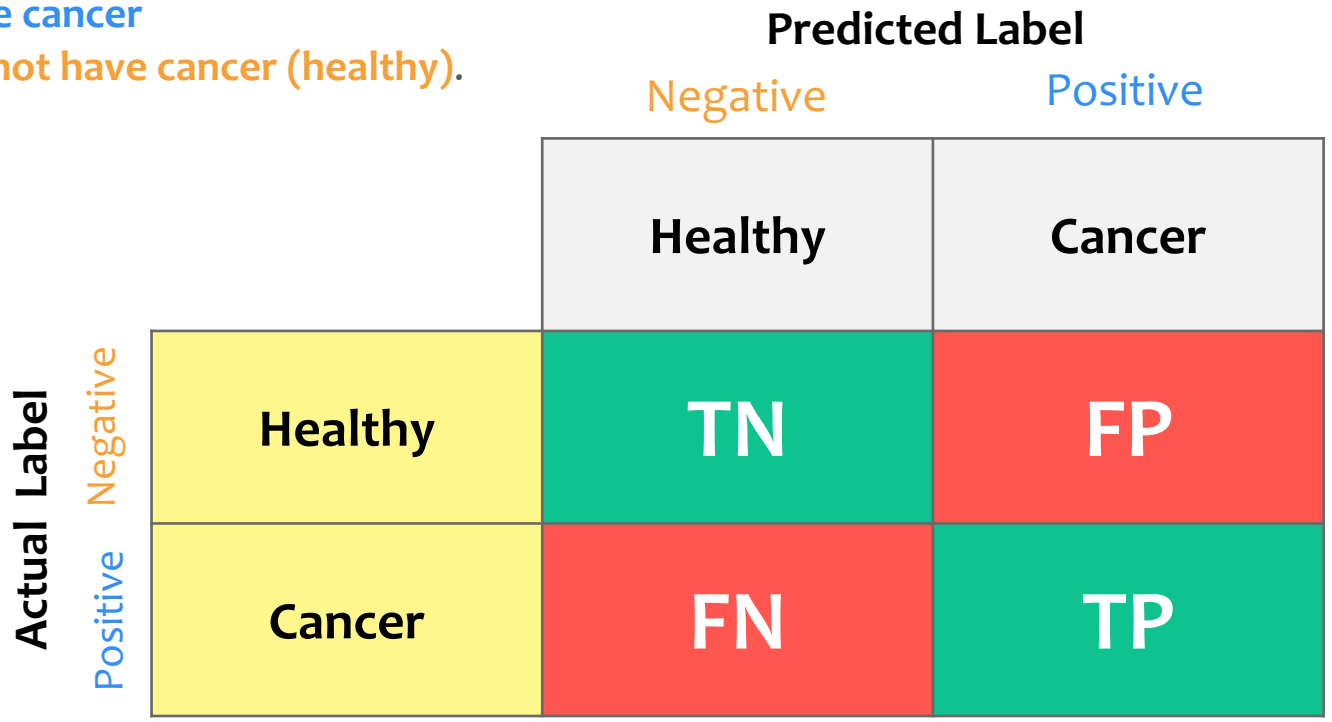


Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.

From **200 people that do not have cancer**:



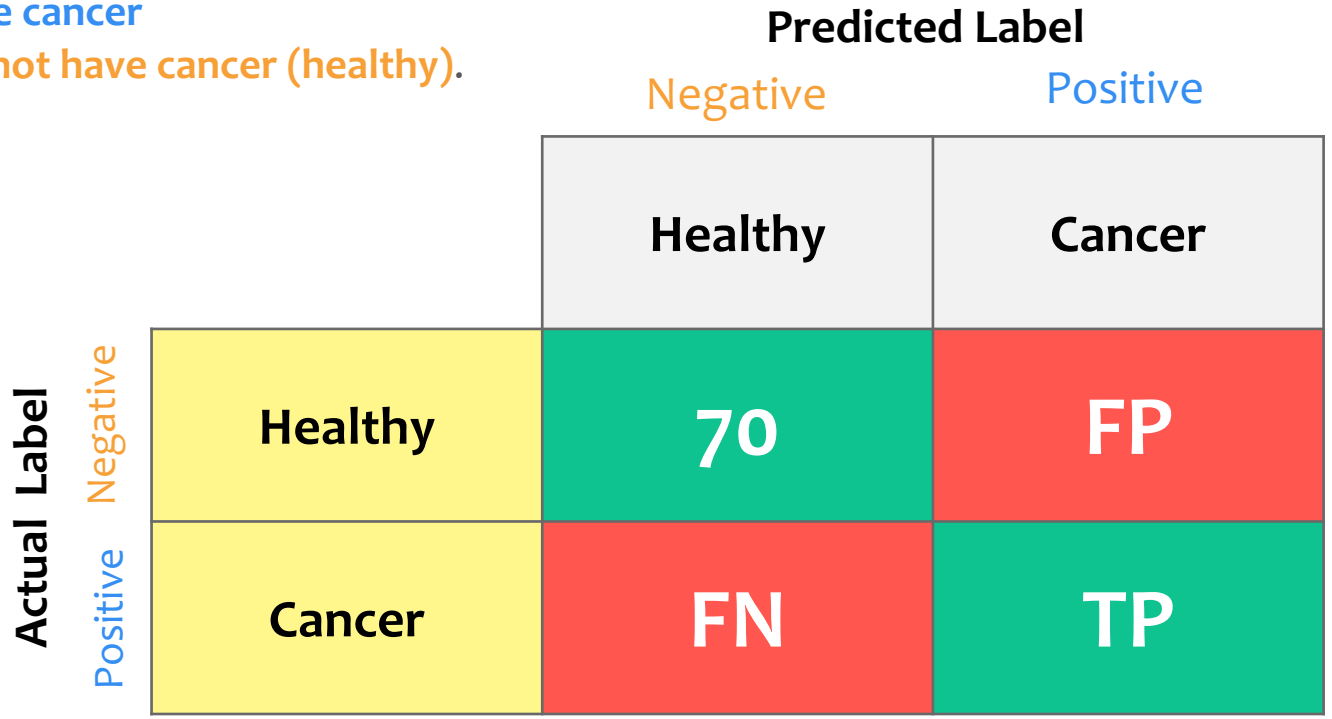
Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.

From **200 people that do not have cancer**:

- **70** were **correctly classified**



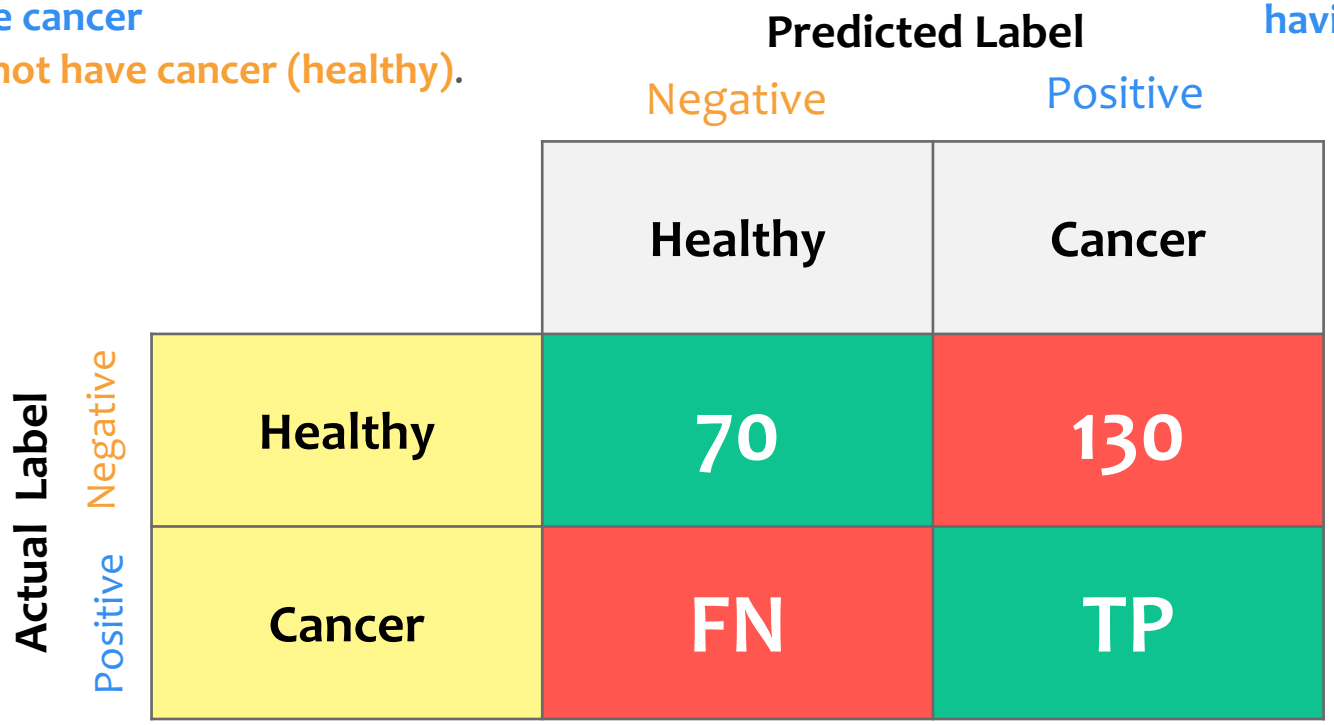
Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.

From **200 people that do not have cancer**:

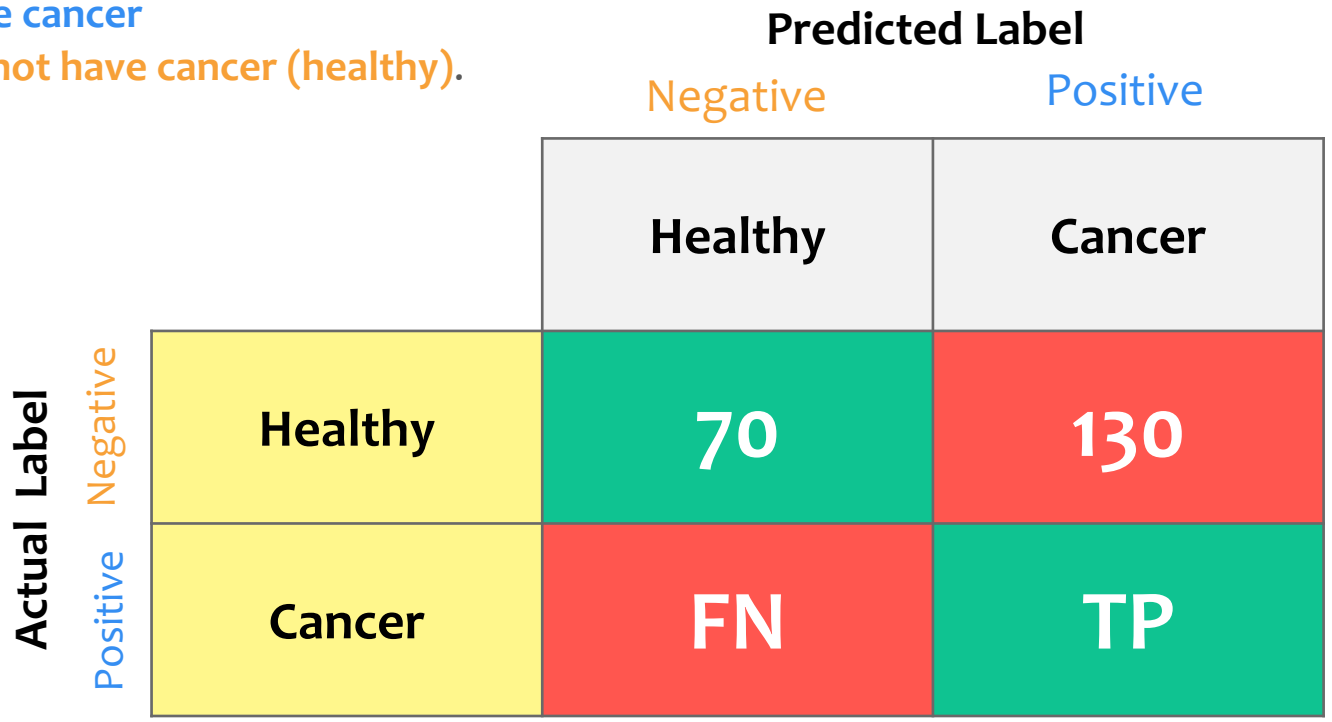
- **70** were **correctly classified**
- **130** were **incorrectly classified** were as **having cancer**.



Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.

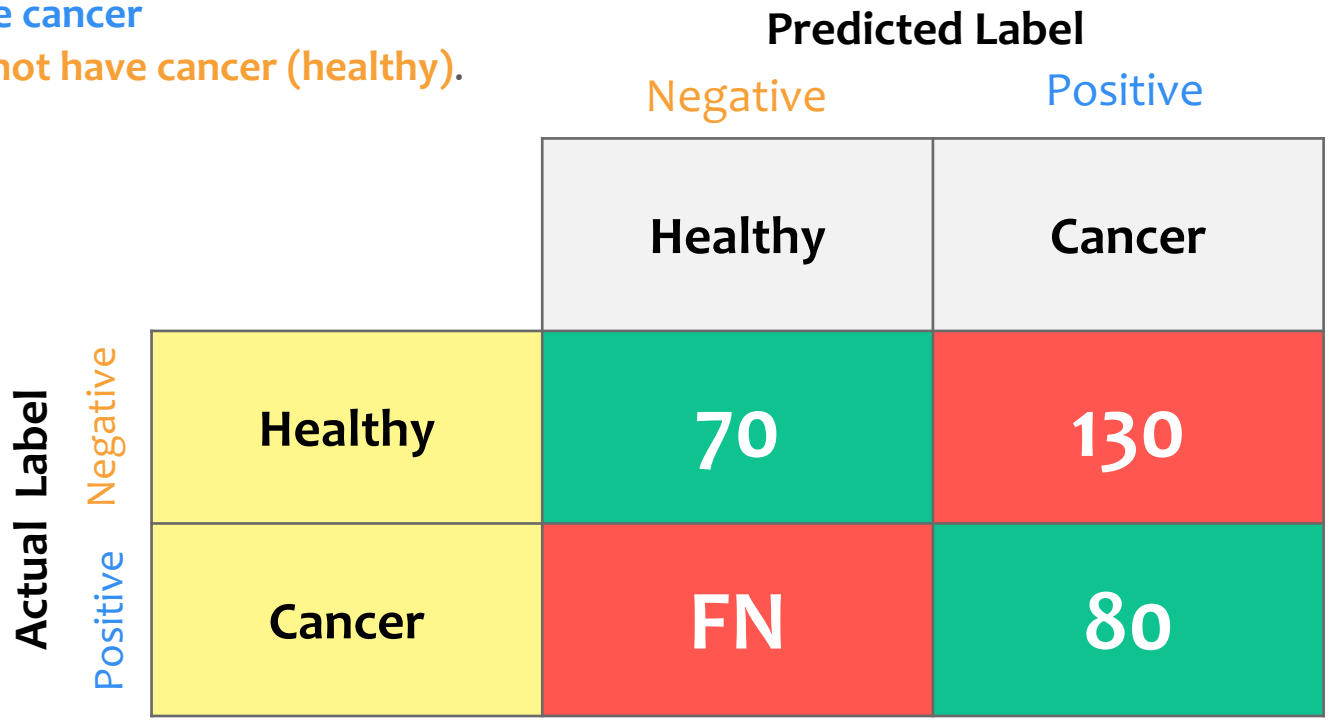


From **100 people that have cancer**:

Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.



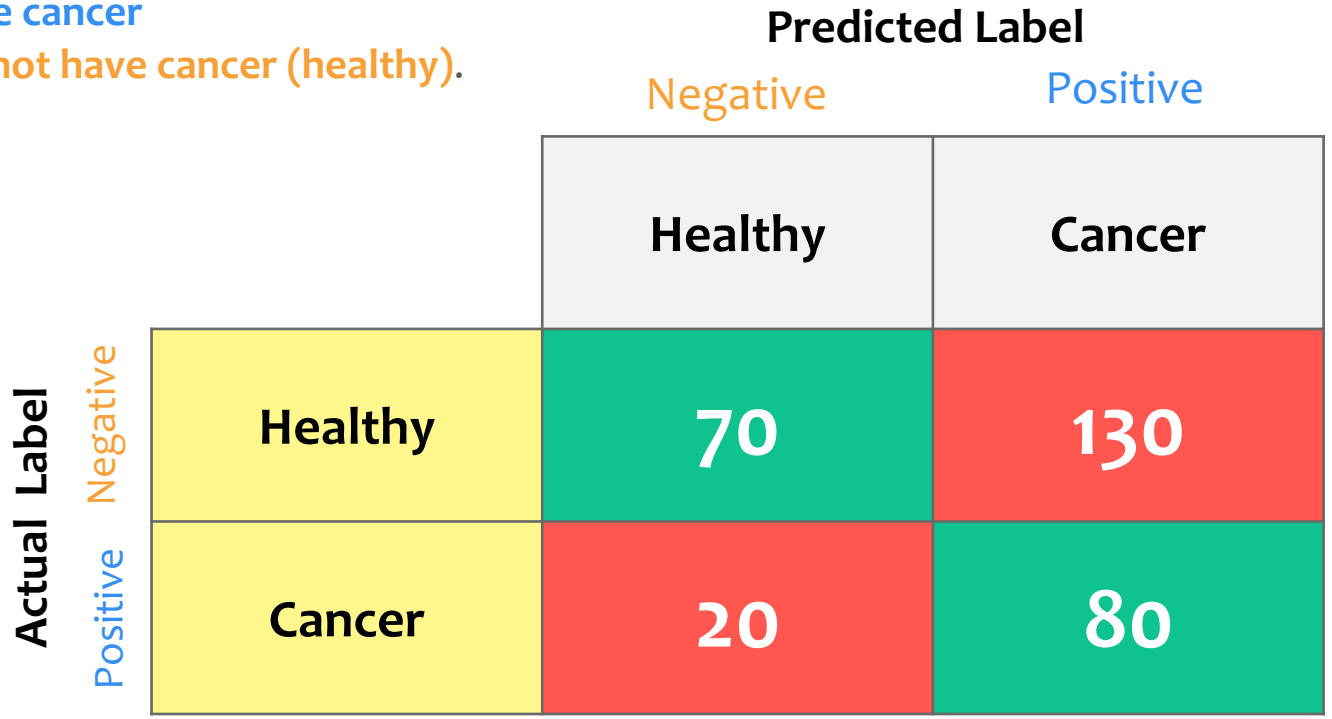
From **100 people that have cancer**:

- **80** were **correctly classified**

Example

A given **model's predictions** for a Validation set with:

- **100** people that **have cancer**
- **200** people that **do not have cancer (healthy)**.



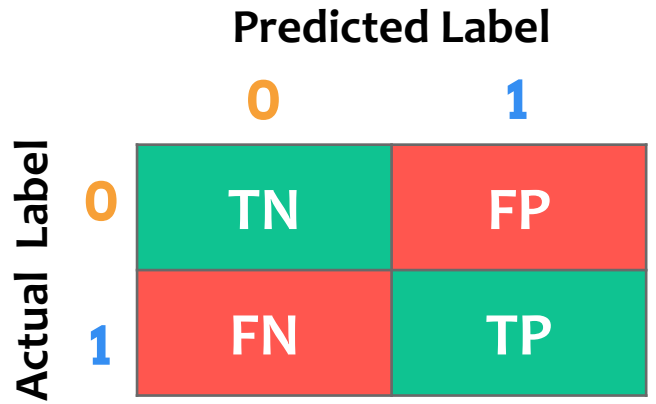
From **100 people that have cancer**:

- **80** were **correctly classified**
- **20** were **incorrectly classified** as **healthy**.

Confusion Matrix

sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

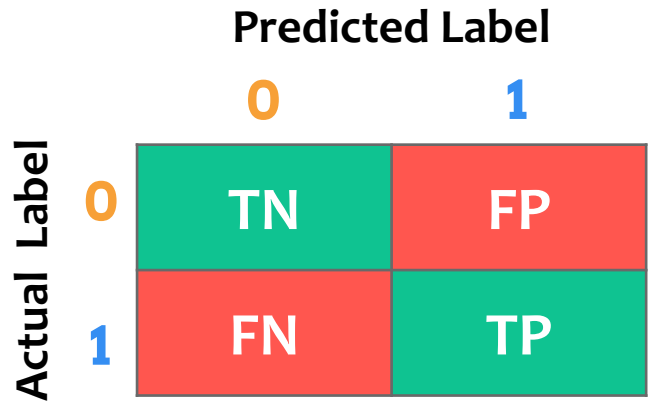


Confusion Matrix

sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

For **binary** problems, it returns a **2x2 matrix** (numpy array)

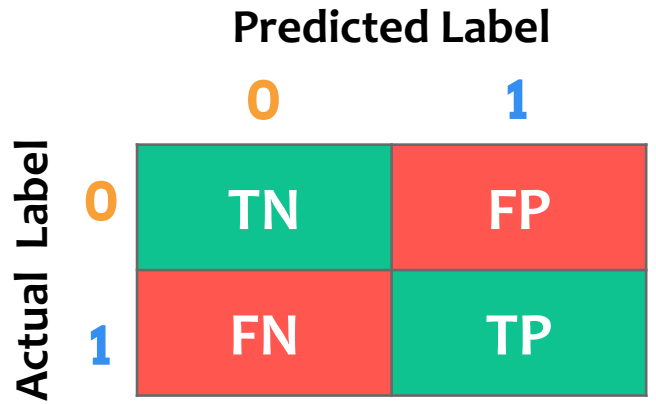


Confusion Matrix

sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

It transforms the **2x2 matrix** into a **1D numpy** array by flattening it.



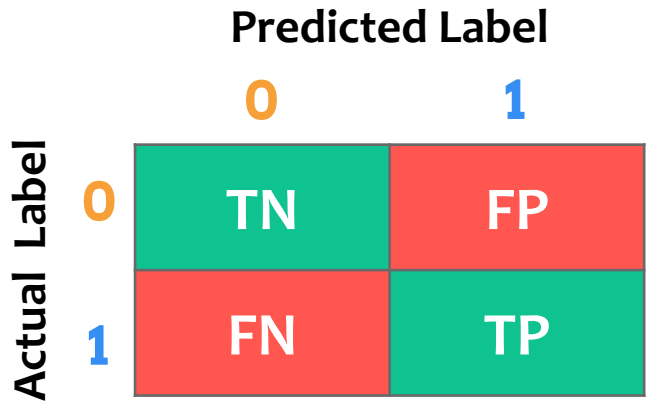
Confusion Matrix

sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

⚠ The classes can be represented with any value (string or int), not necessarily 0 and 1.

⚠ If the problem is **binary**, the **negative class** will be that **with the lowest value**.

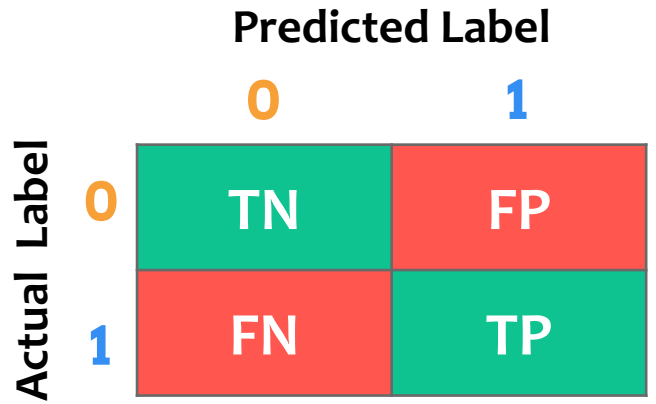


Confusion Matrix

sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

⚠ In case of **strings**, the function will consider as **negative** the one that comes **alphabetically first**. So, in our problem, **Cancer** would be considered the **negative** class.



Confusion Matrix

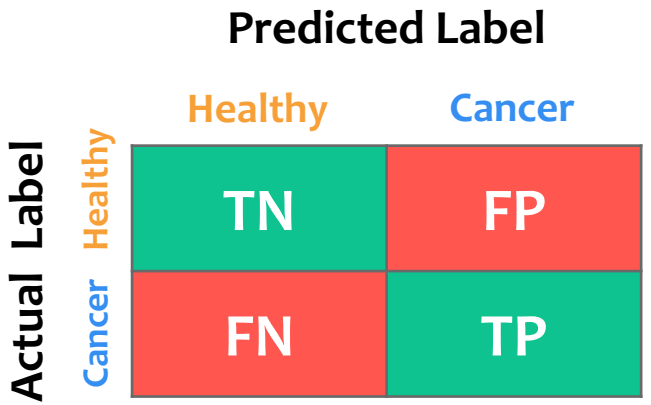
sklearn.metrics.confusion_matrix

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred,
                                  labels=['Healthy', 'Cancer']).ravel()
```

Negative Positive

In case of **strings**, the function will consider as **negative** the one that comes **alphabetically first**. So, in our problem, **Cancer** would be considered the **negative** class.

We can **define** which class will be the **negative** and **positive** classes by passing the order in **`*labels*`**.



Comparing Confusion Matrices

Confusion matrix for **classifier A**
(e.g., KNN)

Predicted Label

Actual Label (Positive / Negative)	Negative	Positive
	Healthy	**Cancer**
Healthy	70	130
Cancer	20	80

Confusion matrix for **classifier A**
(e.g., KNN)

Predicted Label

Actual Label	Negative	Positive
	Healthy	**Cancer**
Positive **Healthy**	70	130
Negative **Cancer**	20	80

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Predicted Label

Actual Label	Negative	Positive
	Healthy	**Cancer**
Positive **Healthy**	80	120
Negative **Cancer**	50	50

Confusion matrix for **classifier A**
(e.g., KNN)

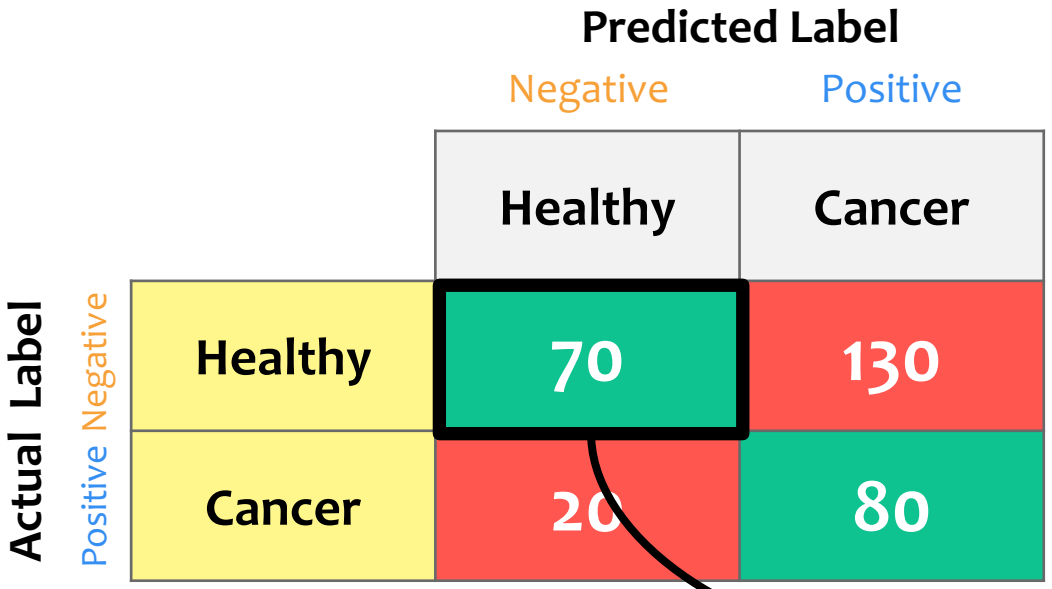
Actual Label	Predicted Label: Negative	Predicted Label: Positive
Healthy	70	130
Cancer	20	80

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

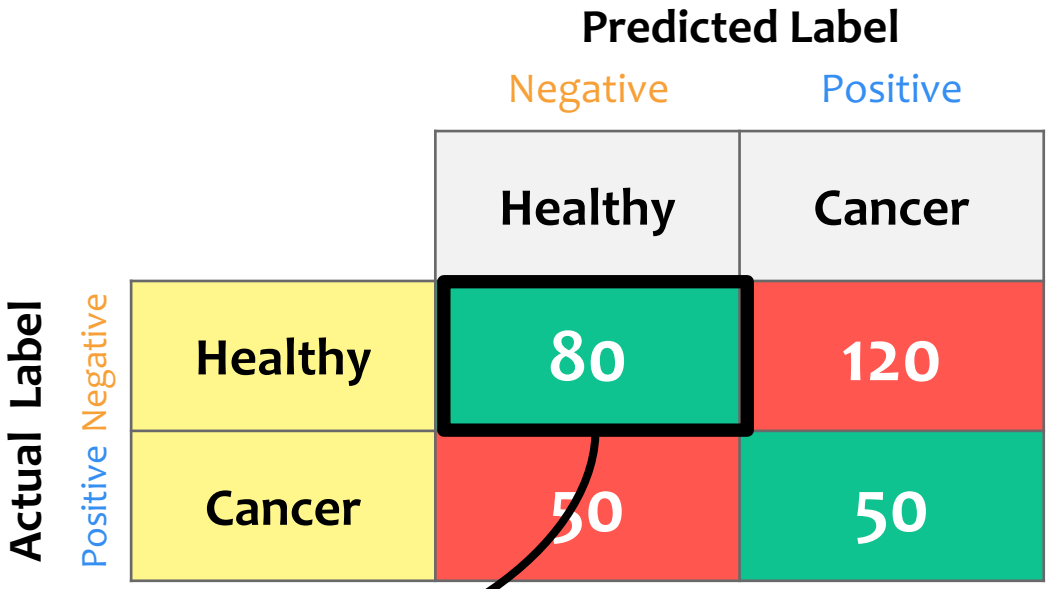
Actual Label	Predicted Label: Negative	Predicted Label: Positive
Healthy	80	120
Cancer	50	50

KNN was *better* than **Logistic Regression** at classifying **patients with cancer** (**80** vs **50**)…

Confusion matrix for **classifier A**
(e.g., KNN)

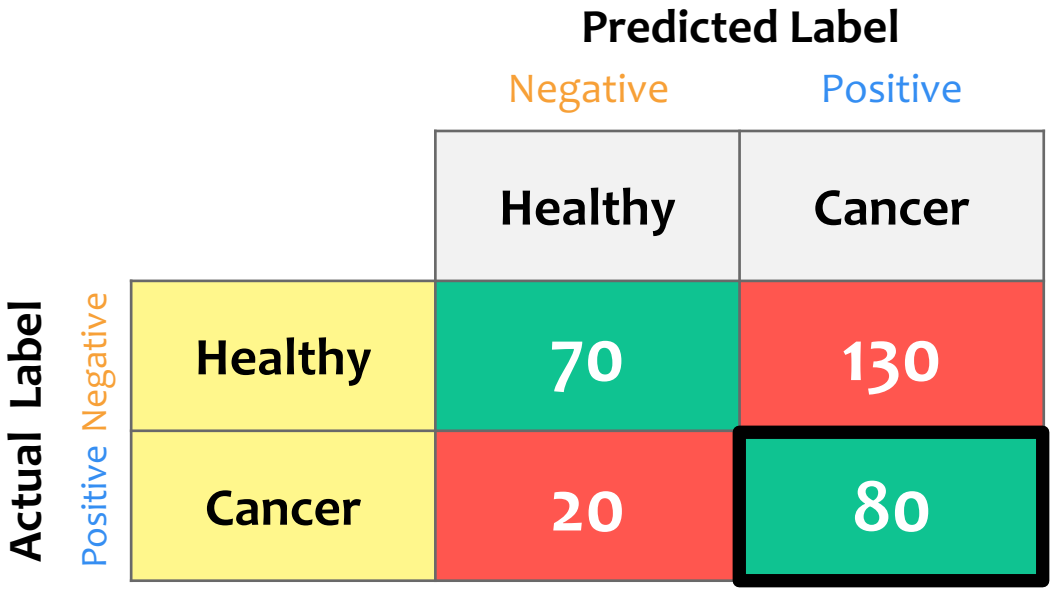


Confusion matrix for **classifier B**
(e.g., Logistic Regression)

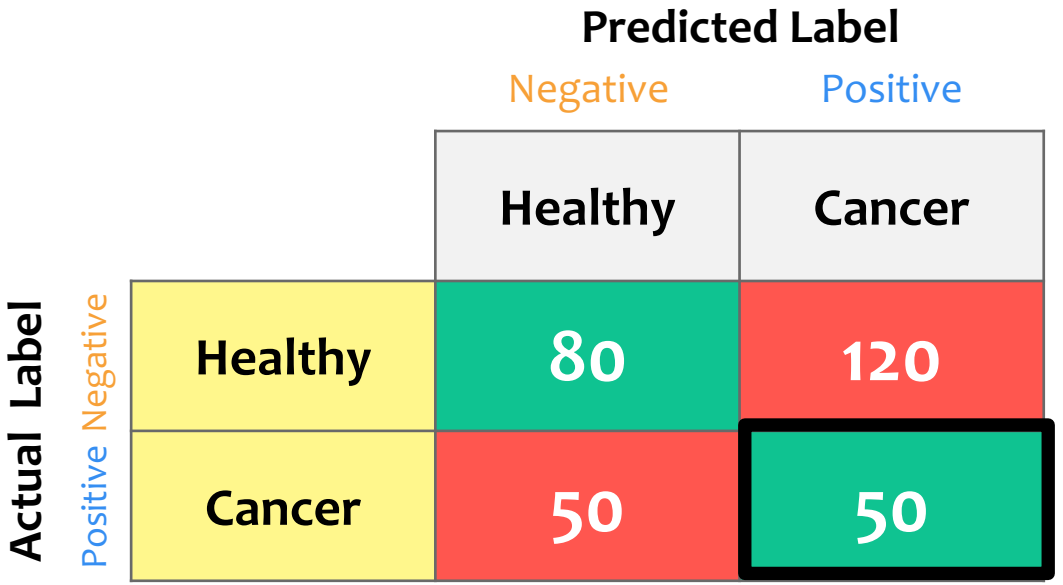


… but it is *worse* at classifying **healthy patients** (**70** vs **80**).

Confusion matrix for **classifier A**
(e.g., KNN)

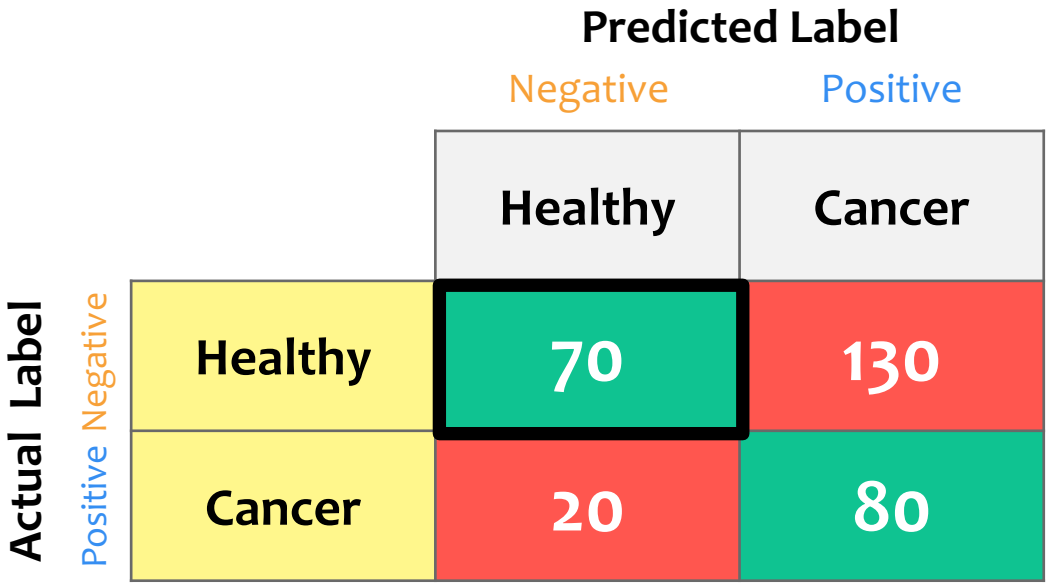


Confusion matrix for **classifier B**
(e.g., Logistic Regression)

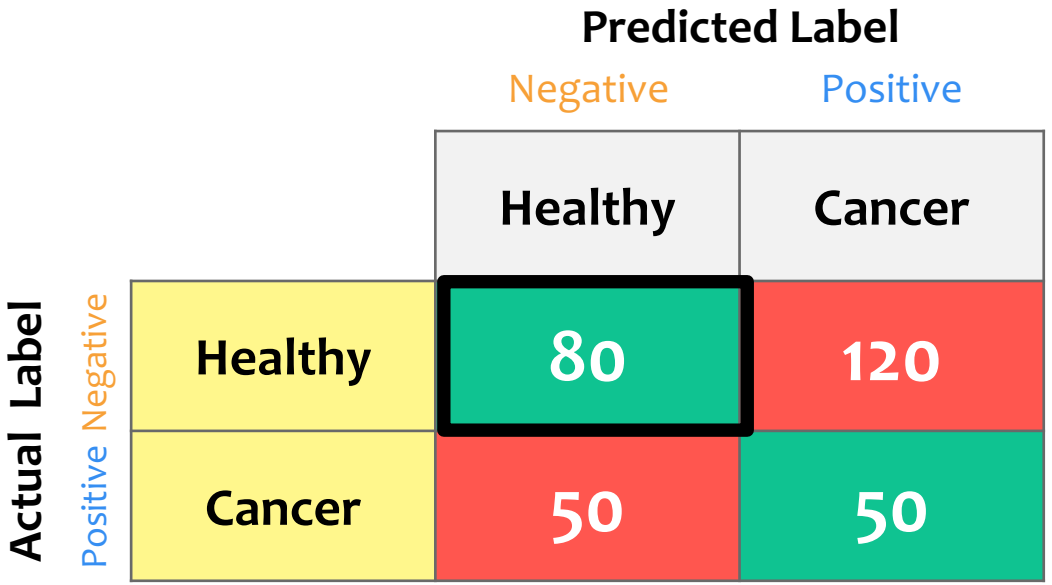


So, if it is **more important** to **correctly classify positives**, we should choose **KNN**

Confusion matrix for **classifier A**
(e.g., KNN)



Confusion matrix for **classifier B**
(e.g., Logistic Regression)



Otherwise, if it is **more important** to **correctly classify negatives**, we should choose **Logistic Regression**

Confusion matrix for **classifier A**
(e.g., KNN)

Predicted Label

Actual Label (Positive / Negative)	Negative: **Healthy**	Positive: **Cancer**
Healthy	70	130
Cancer	20	80

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Predicted Label

Actual Label (Positive / Negative)	Negative: **Healthy**	Positive: **Cancer**
Healthy	80	120
Cancer	50	50

⚠ Comparing **confusion matrices** can be **hard** especially when they are **very similar,** so *making it difficult* to know which model is better,

Confusion matrix for **classifier A**
(e.g., KNN)

Predicted Label

Actual Label (Positive / Negative)	Healthy (Negative)	Cancer (Positive)
Healthy	70	130
Cancer	20	80

Comparing **confusion matrices** can be **hard** especially when they are **very similar,** so *making it difficult* to know which model is better,

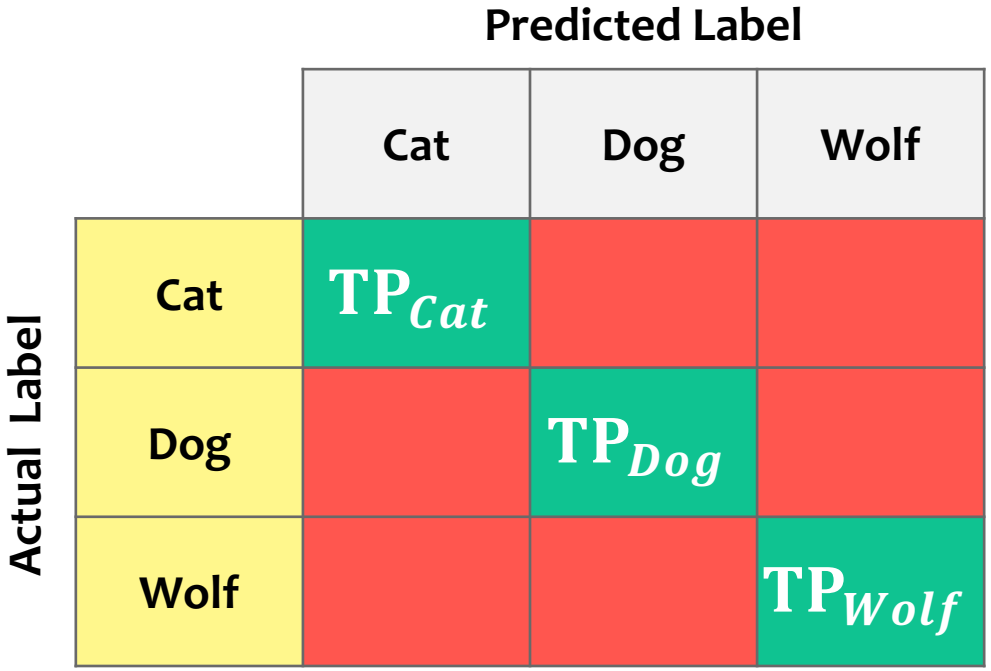
Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Predicted Label

Actual Label (Positive / Negative)	Healthy (Negative)	Cancer (Positive)
Healthy	80	120
Cancer	50	50

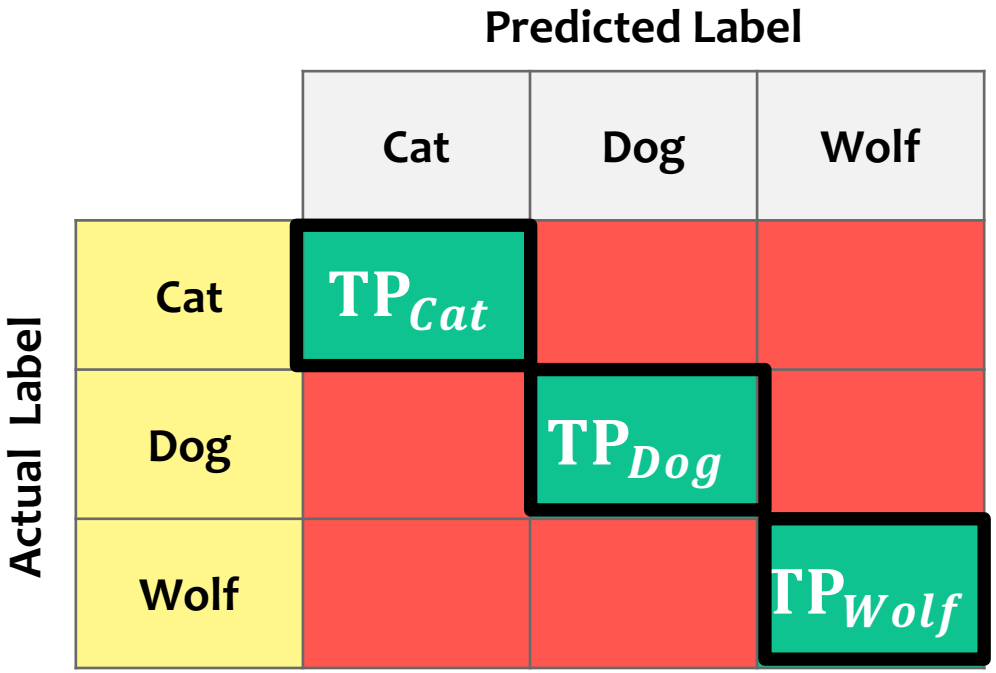
There are more sophisticated **evaluation metrics**, such as **Sensitivity**, **Specificity**, **Precision**, **Recall**, …, that can help us make a better decision by **summarizing** some points of the **confusion matrix**.

Confusion Matrix for Multiple Classes



Confusion Matrix for Multiple Classes

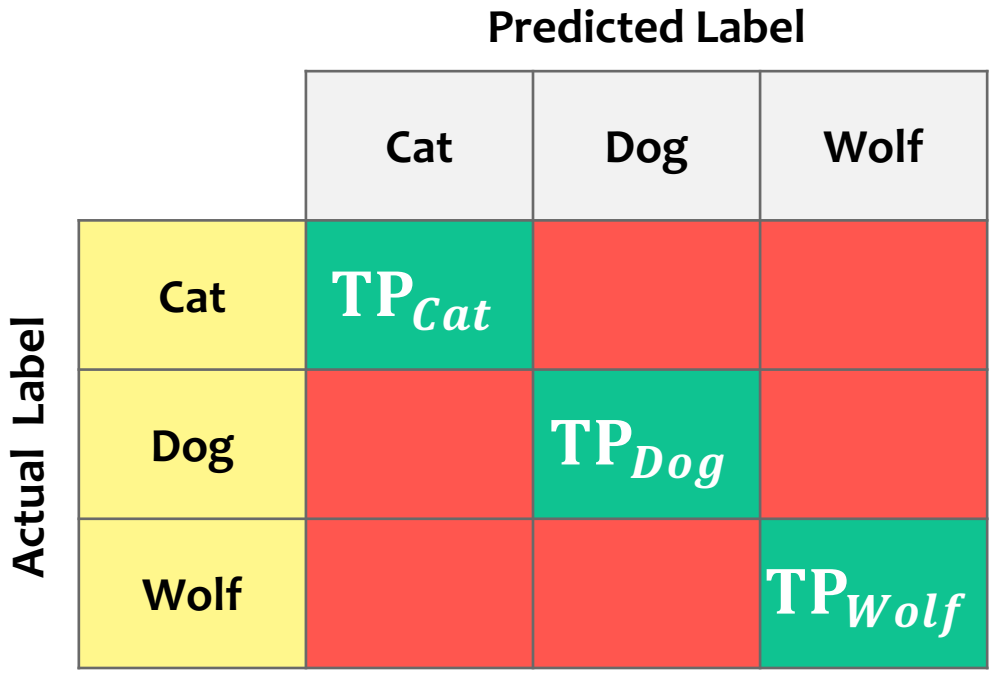
The **diagonal** keeps showing where the model/solution **hit**, *i.e.*, **correctly** **classified the classes (true positives).**



Confusion Matrix for Multiple Classes

The **diagonal** keeps showing where the model/solution **hit**, *i.e.*, **correctly** **classified the classes (true positives).**

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_pred, labels=['Cat', 'Dog', 'Wolf'])
```



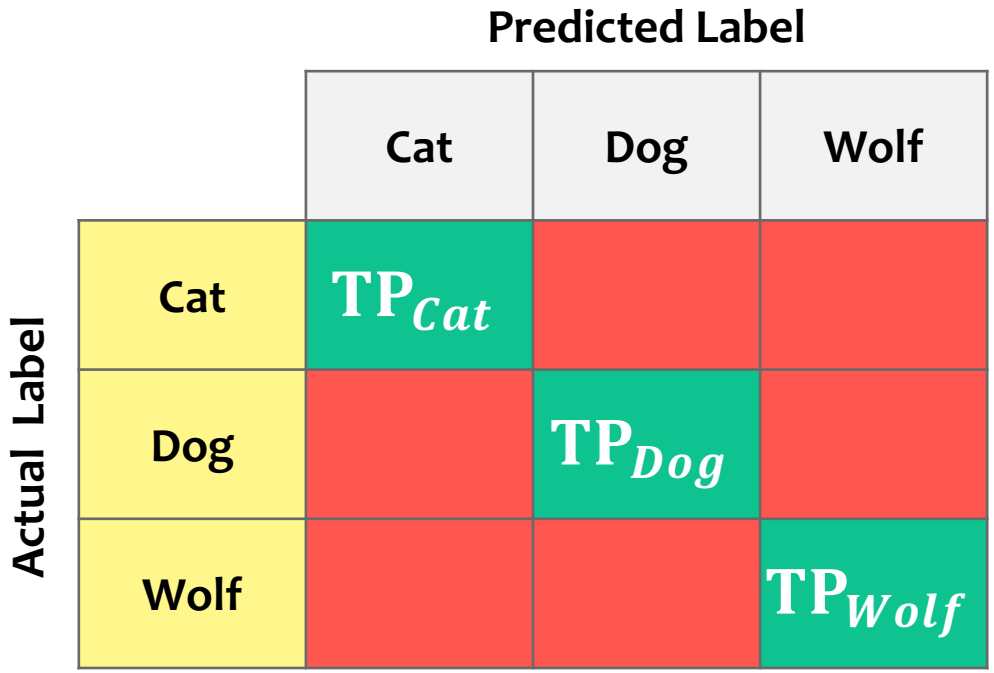
Confusion Matrix for Multiple Classes

The **diagonal** keeps showing where the model/solution **hit**, *i.e.*, **correctly** **classified the classes (true positives).**

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_pred, labels=['Cat', 'Dog', 'Wolf'])
```

Just to guarantee the **expected order** for the classes (rows and columns).

Otherwise, the **order** will be the **alphabetical one**, which is the same in this case.

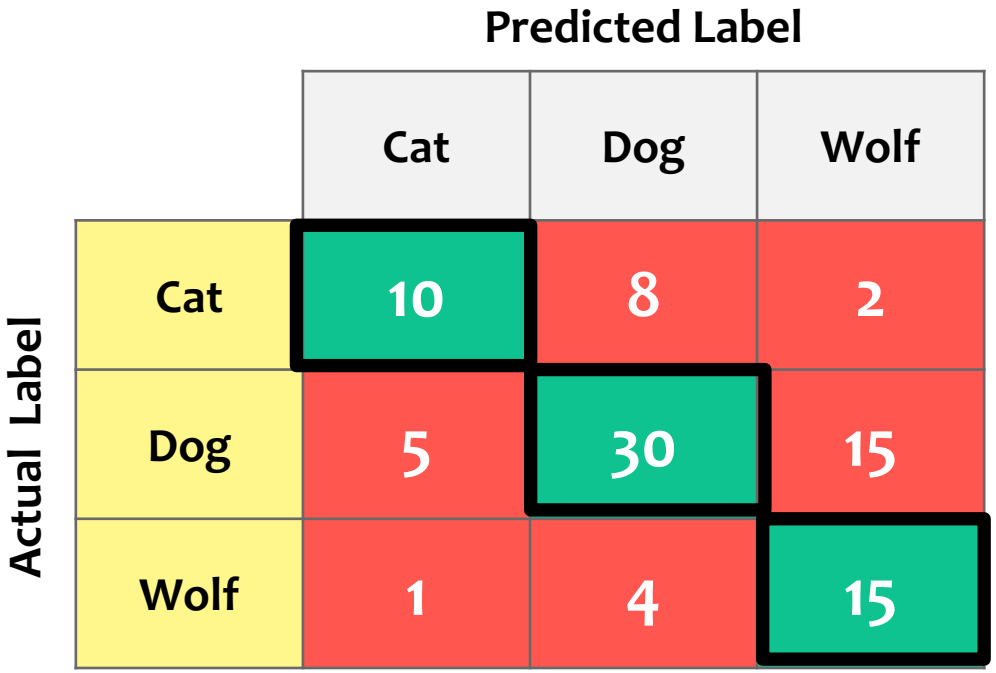


Confusion Matrix for Multiple Classes

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

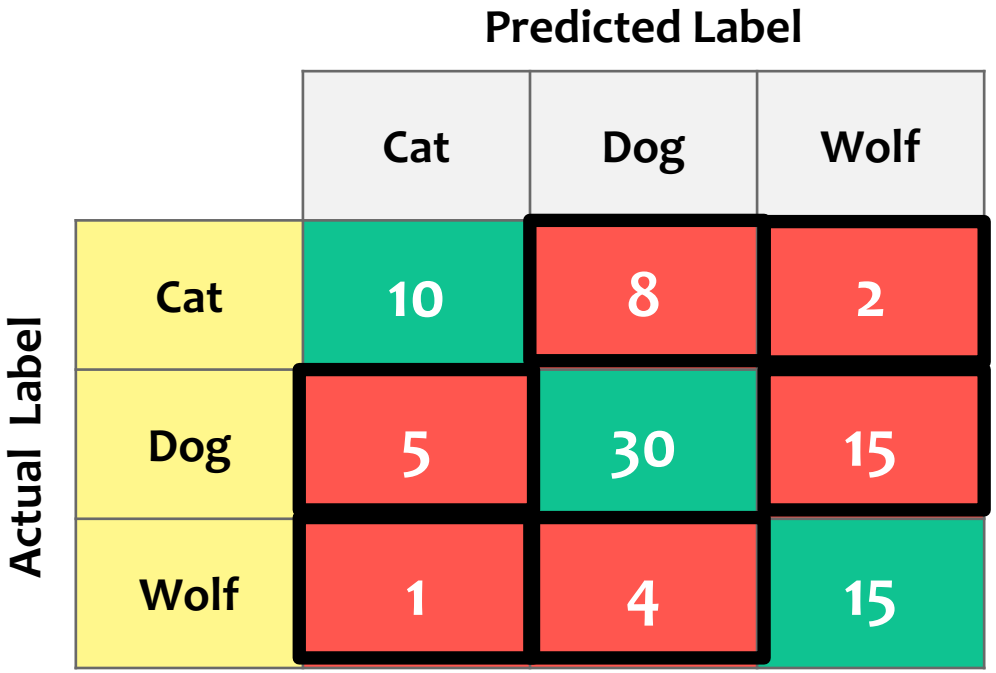
Confusion Matrix for Multiple Classes

The **diagonal** keeps showing where the model/solution **hit**, *i.e.*, **correctly** **classified the classes (true positives).**



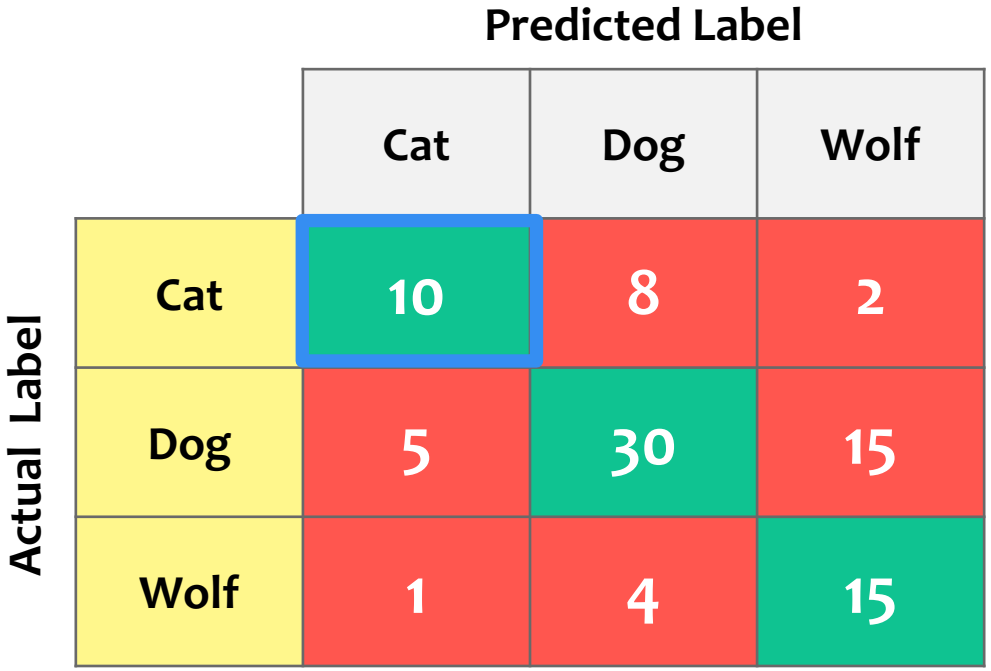
Confusion Matrix for Multiple Classes

The remaining cells shows where the model/solution **missed**, *i.e.*, **incorrectly classified the classes.**



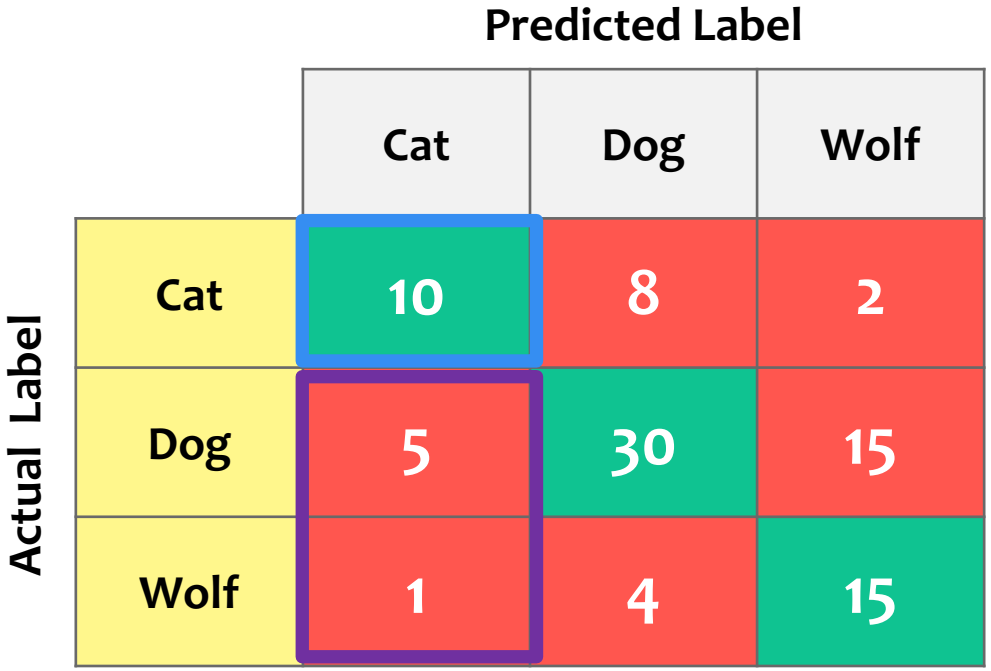
Confusion Matrix for Multiple Classes

$$TP_{Cat}$$



Confusion Matrix for Multiple Classes

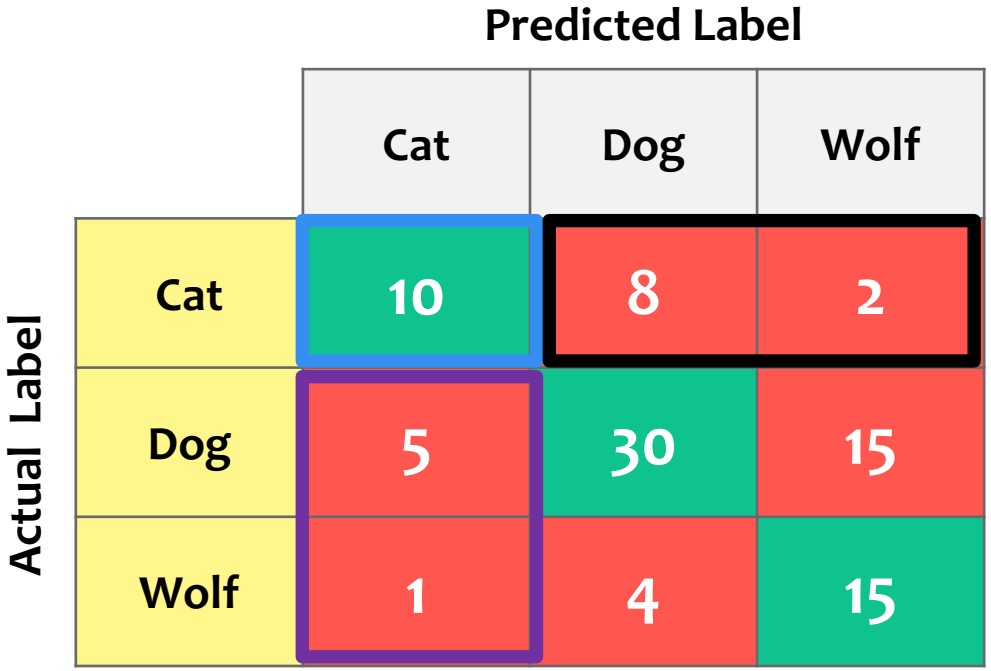
TP_{Cat} FP_{Cat}



Confusion Matrix for Multiple Classes

TP_{Cat} FP_{Cat}

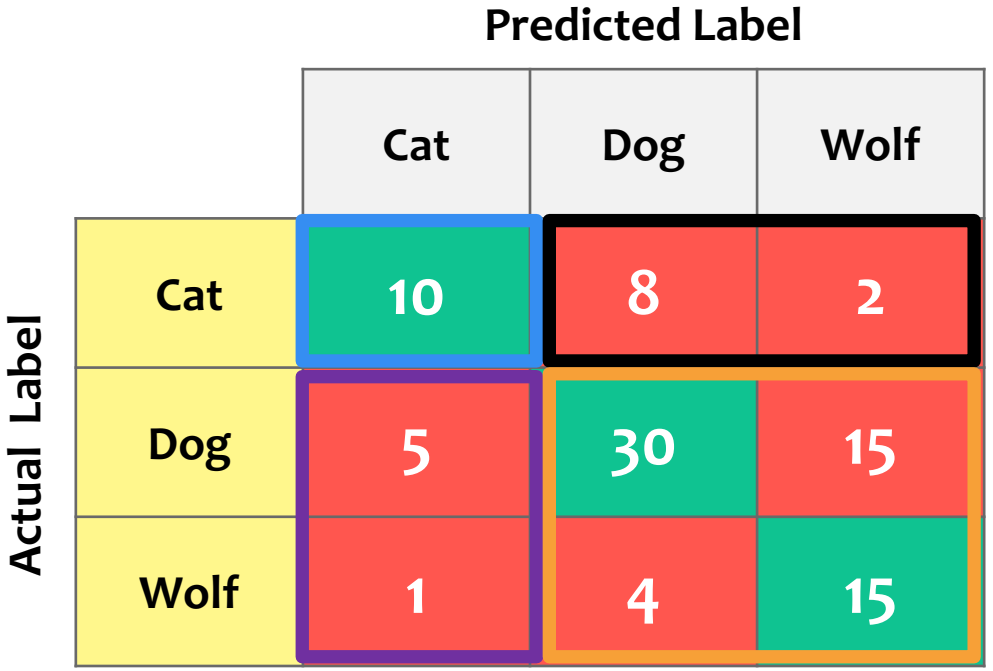
FN_{Cat}



Confusion Matrix for Multiple Classes

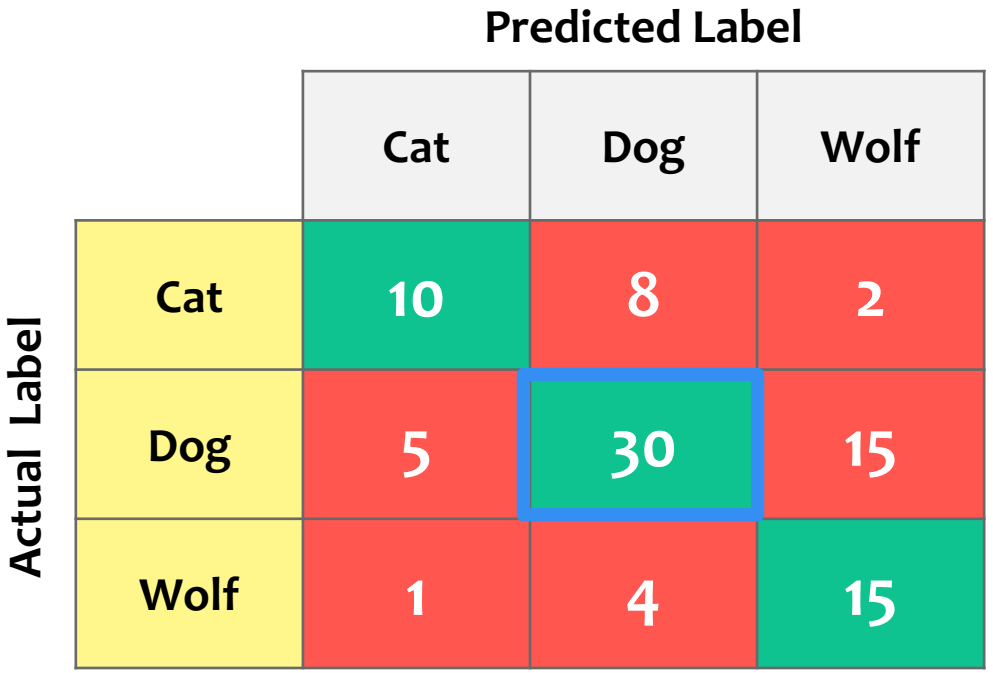
TP_{Cat} FP_{Cat}

FN_{Cat} TN_{Cat}



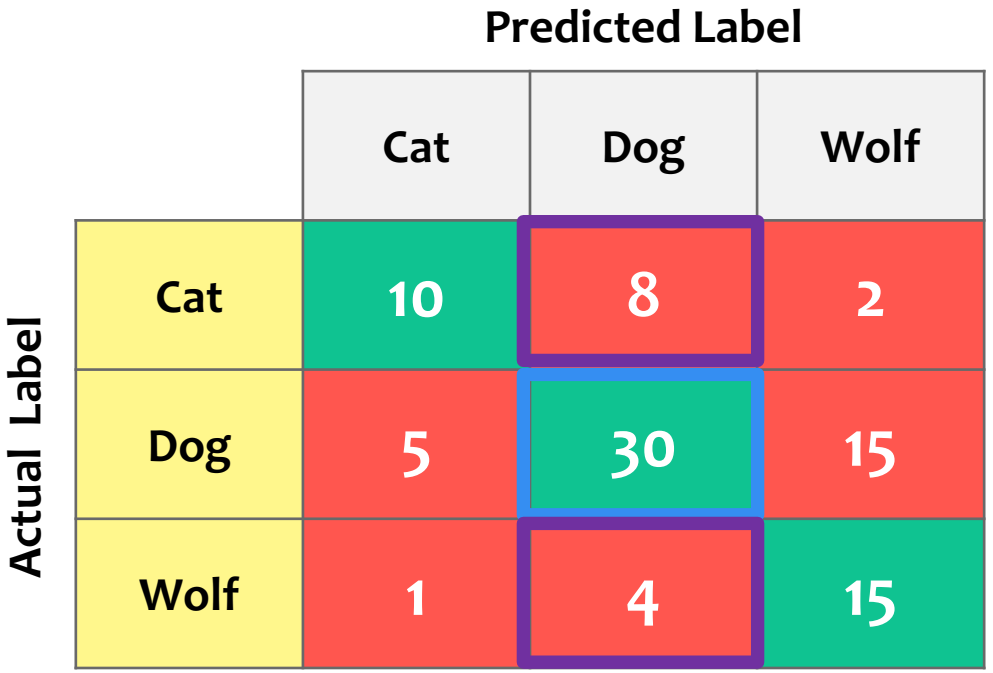
Confusion Matrix for Multiple Classes

$$TP_{Dog}$$



Confusion Matrix for Multiple Classes

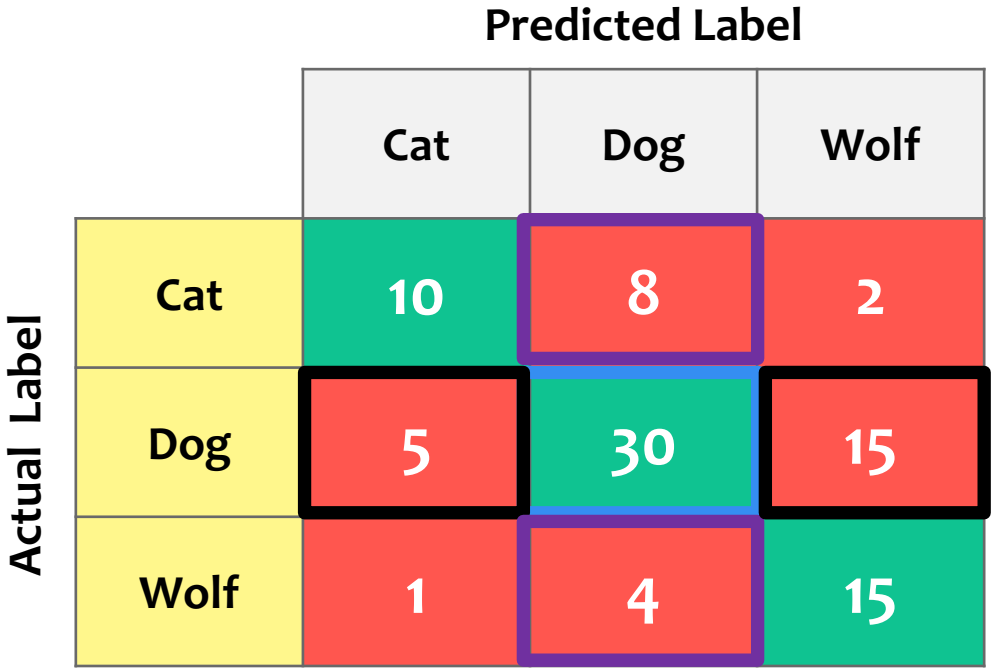
TP_{Dog} FP_{Dog}



Confusion Matrix for Multiple Classes

TP_{Dog} FP_{Dog}

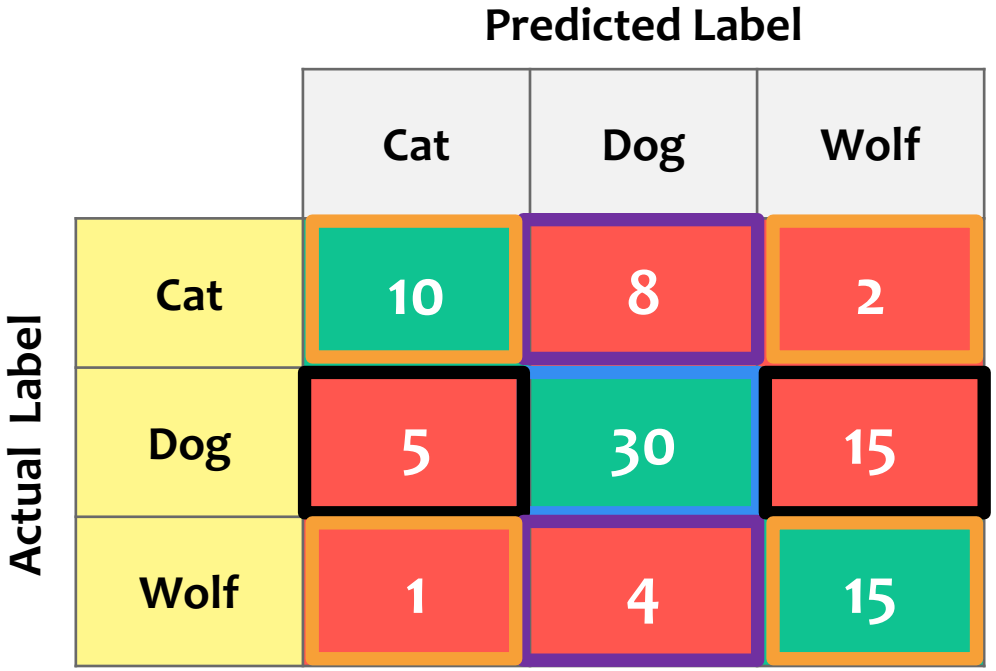
FN_{Dog}



Confusion Matrix for Multiple Classes

TP_{Dog} FP_{Dog}

FN_{Dog} TN_{Dog}



Confusion Matrix for Multiple Classes

Exercise: Compute…

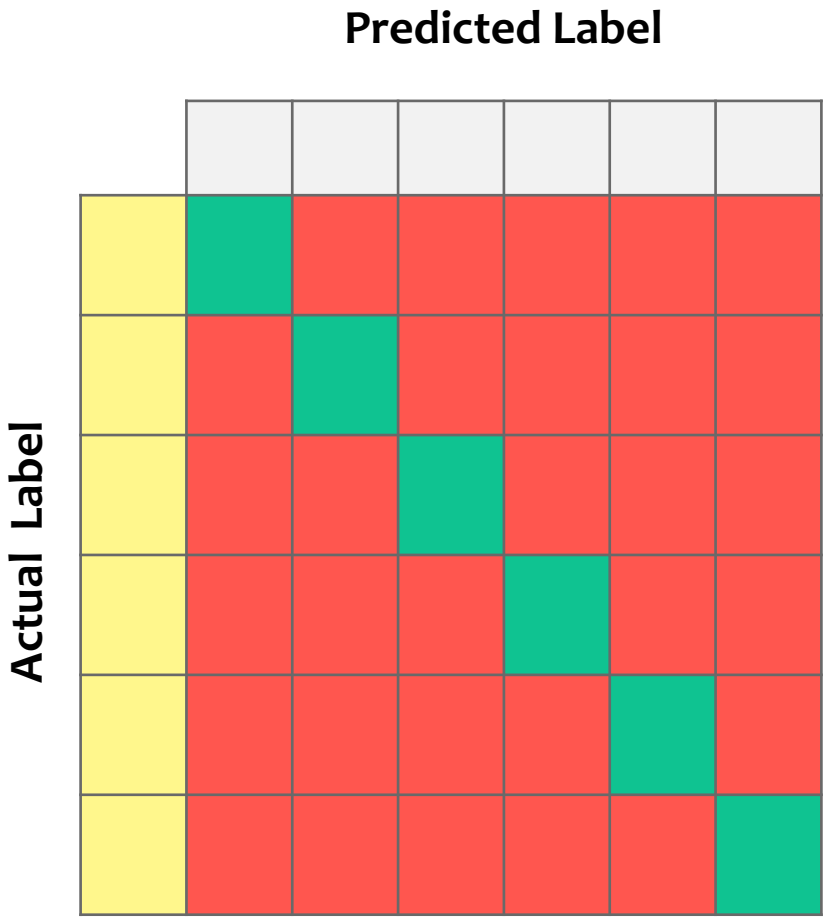
TP_{Wolf} FP_{Wolf}

FN_{Wolf} TN_{Wolf}

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

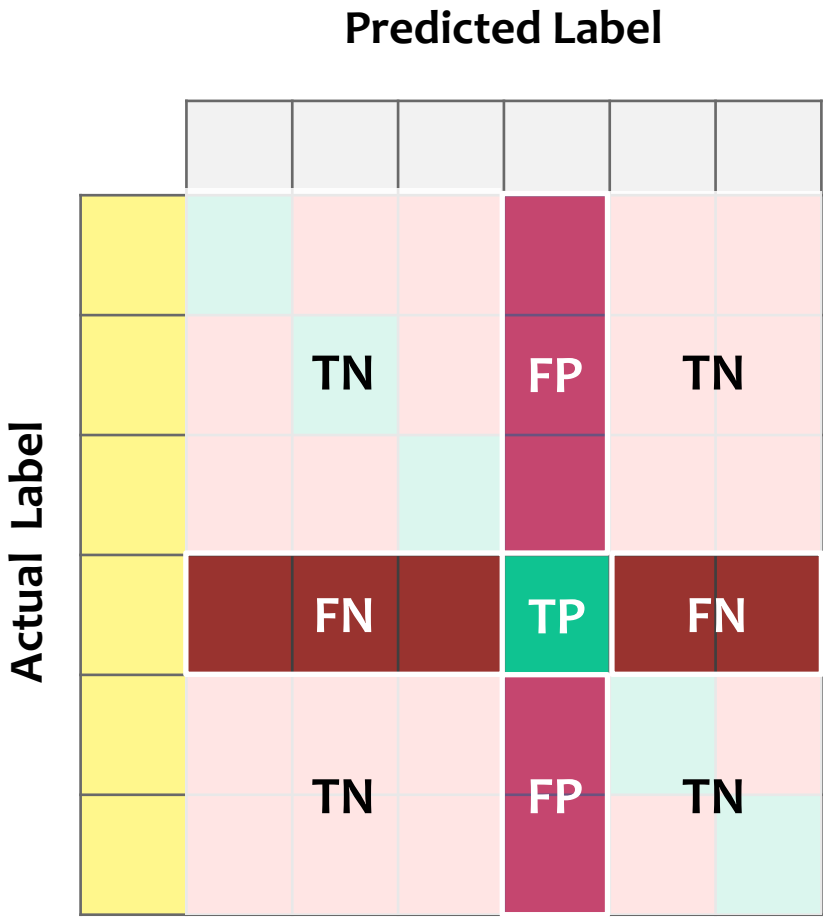
Confusion Matrix for Multiple Classes

If we had **c classes**, we would have a **confusion matrix** with **c rows and c columns**.



Confusion Matrix for Multiple Classes

If we had **c classes**, we would have a **confusion matrix** with **c rows and c columns**..



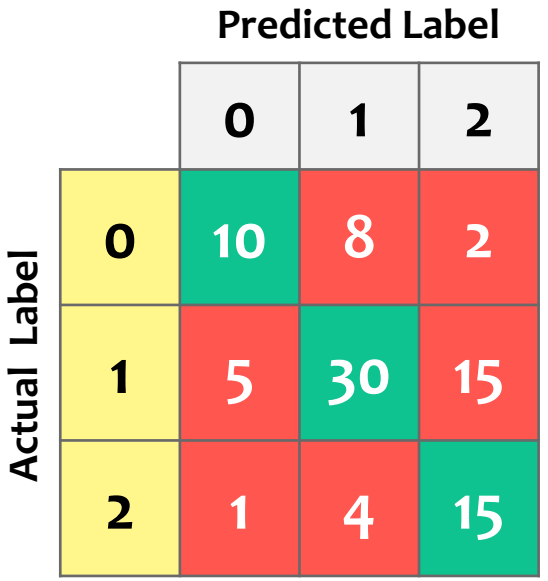
For a given **class**

Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

Confusion Matrix for Multiple Classes

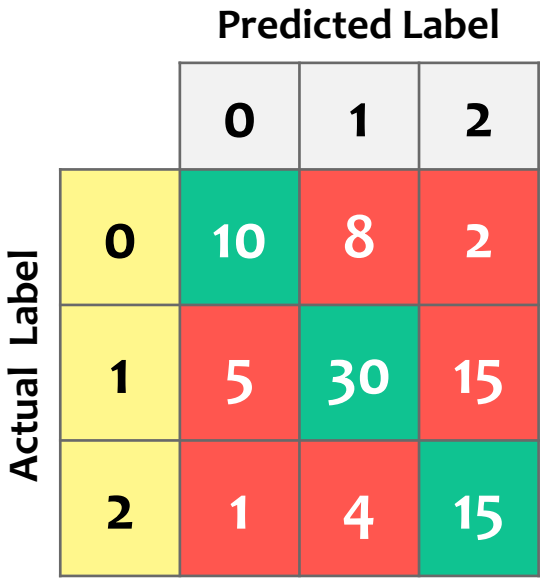
```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```



Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

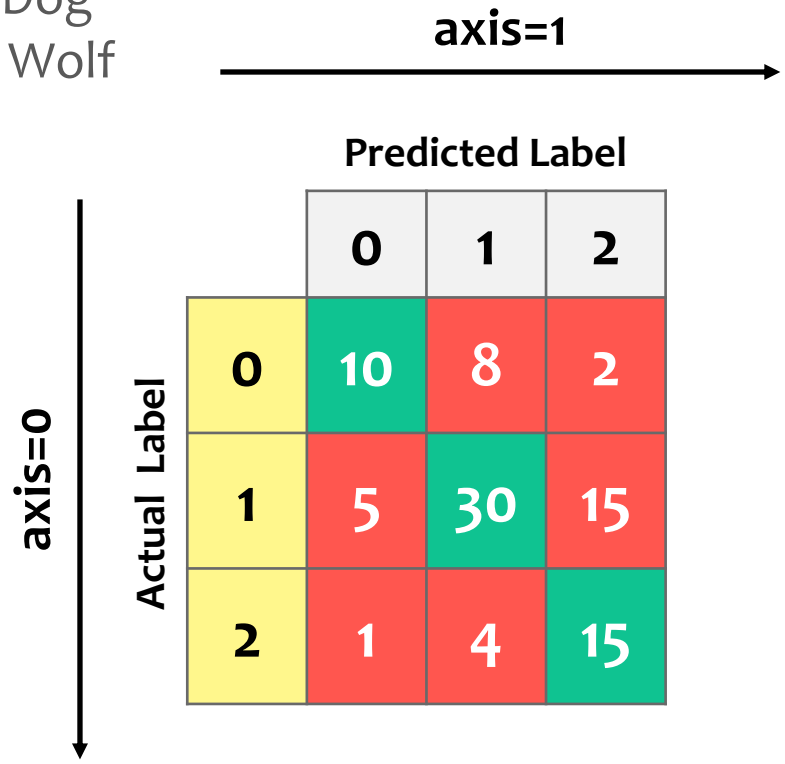
0: Cat
1: Dog
2: Wolf



Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf



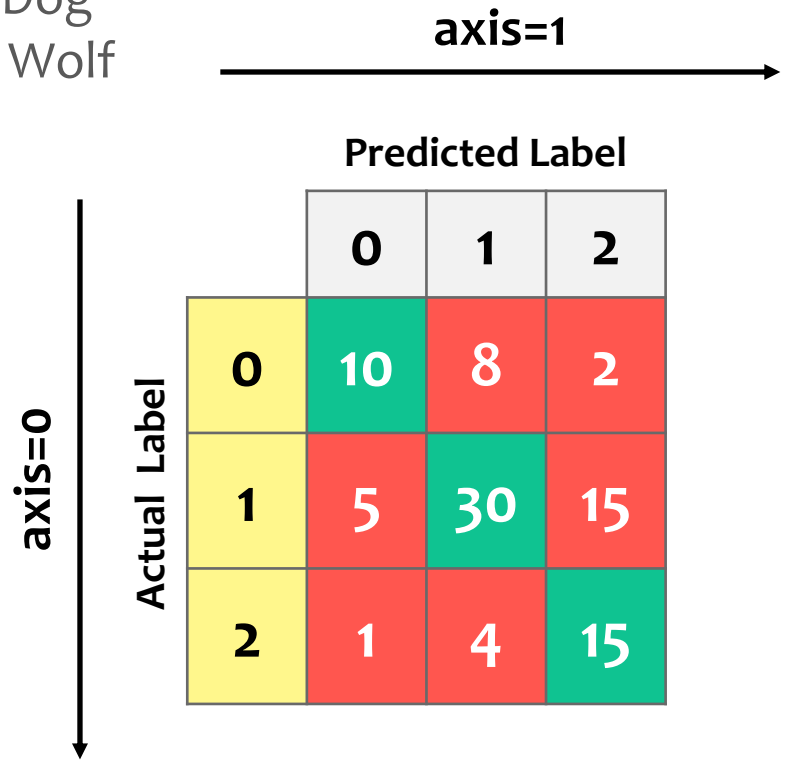
Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

$TP =$

0	1	2

0: Cat
1: Dog
2: Wolf



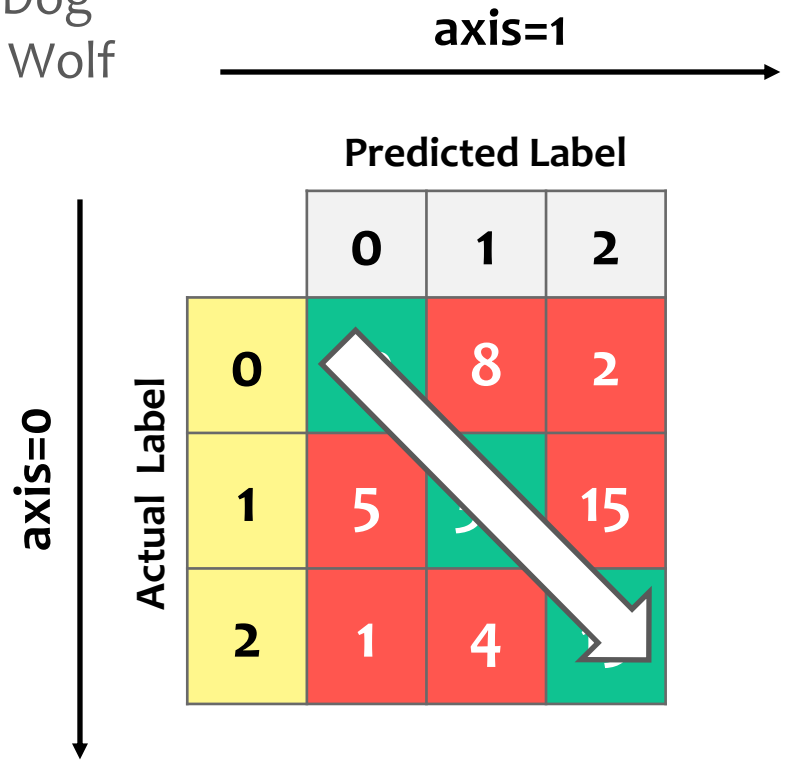
Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

$TP =$

0	1	2
10	30	15

0: Cat
1: Dog
2: Wolf

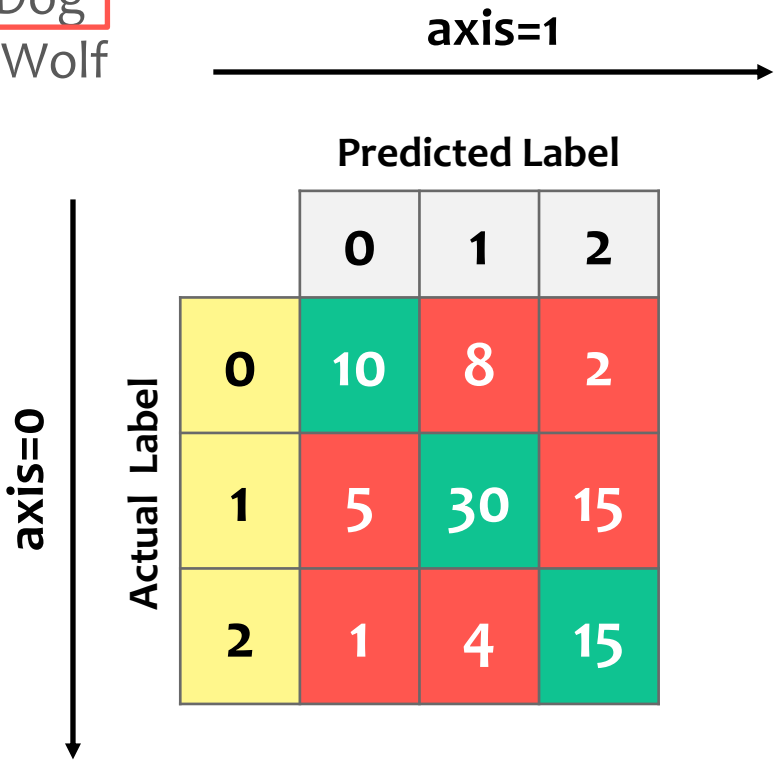


Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

$$TP = \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 10 & 30 & 15 \\ \hline \end{array}$$

0: Cat
1: Dog
2: Wolf

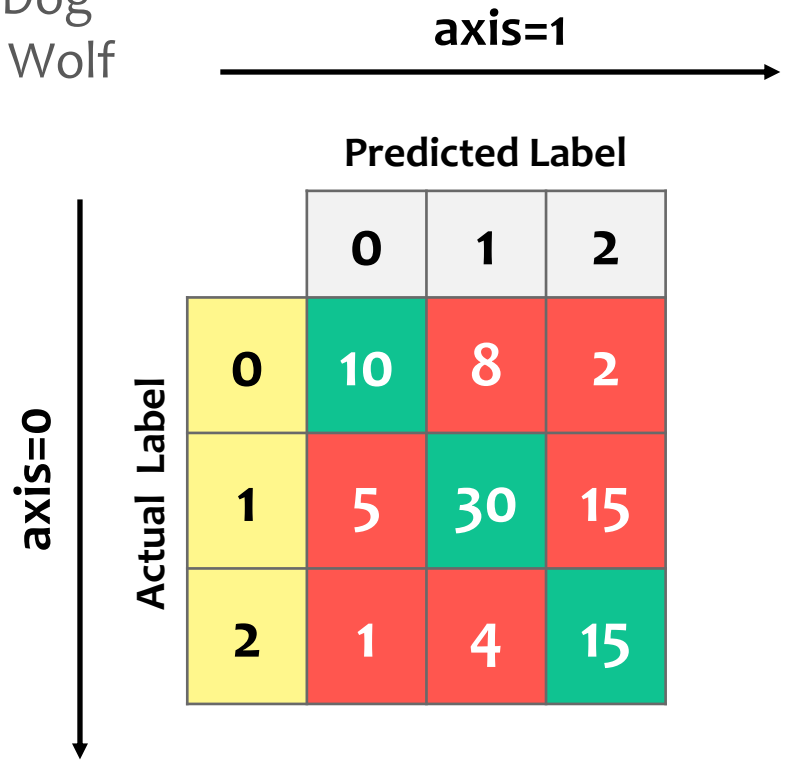


$$TP[1] = TP_{Dog}$$

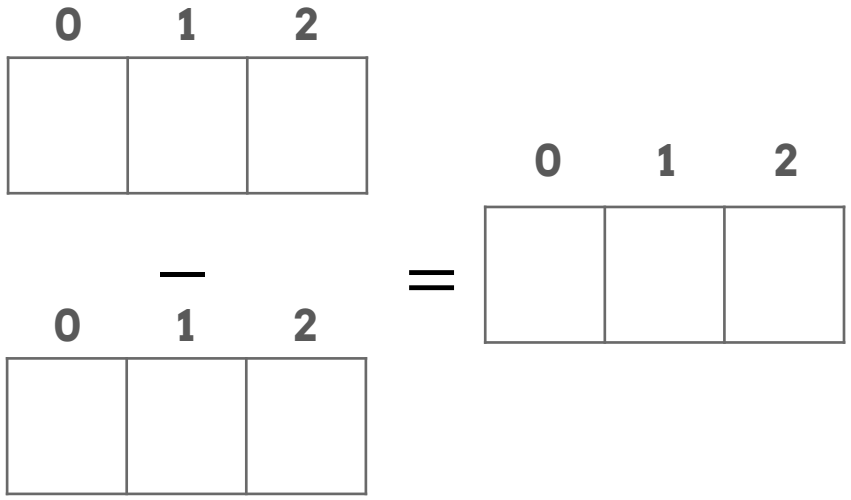
Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf



$$FP =$$

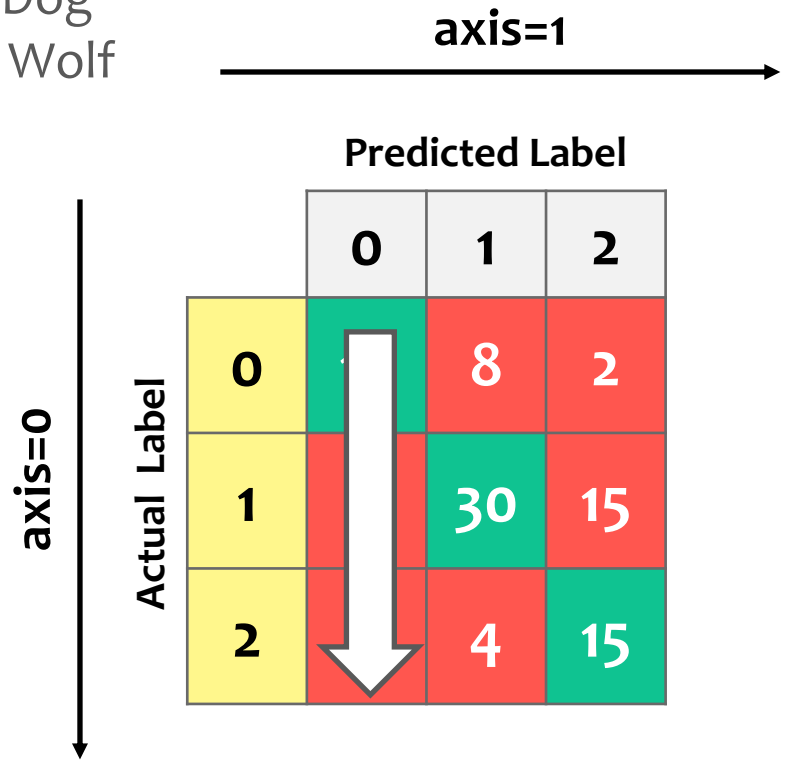


$$=$$

Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

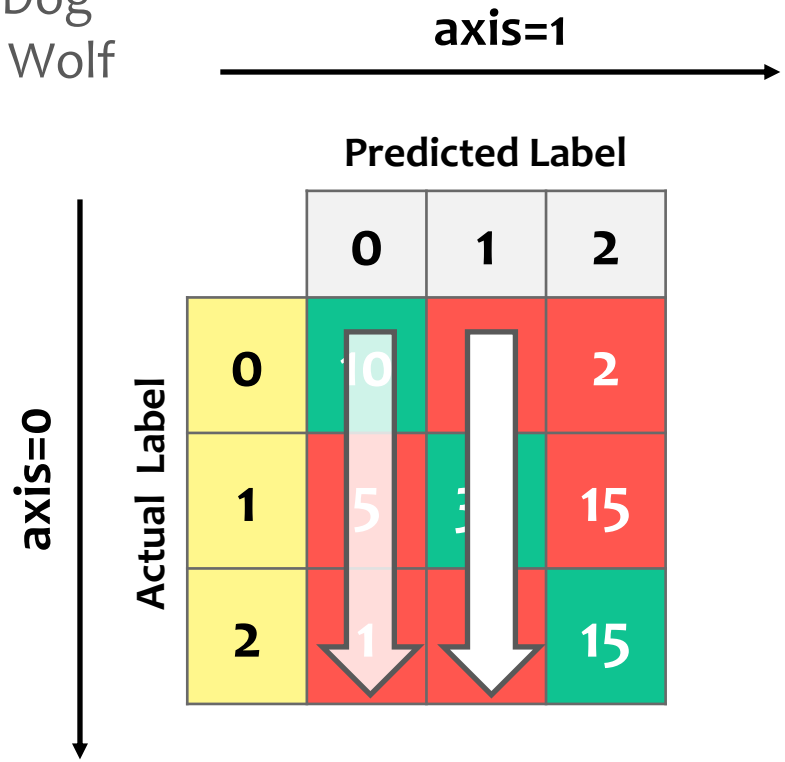


$$FP = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 16 & & \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

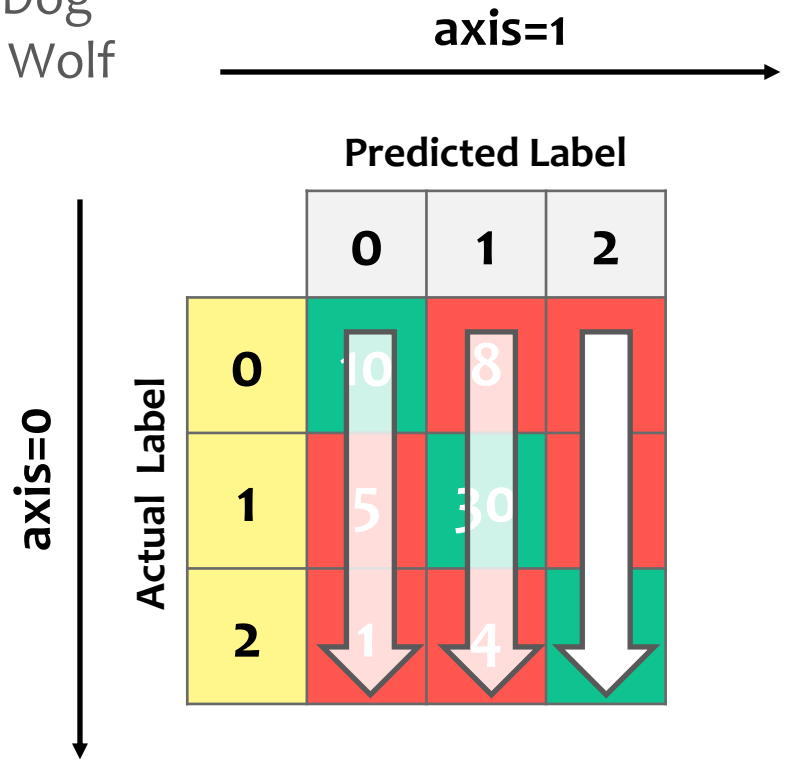


$$FP = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 16 & 42 & \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

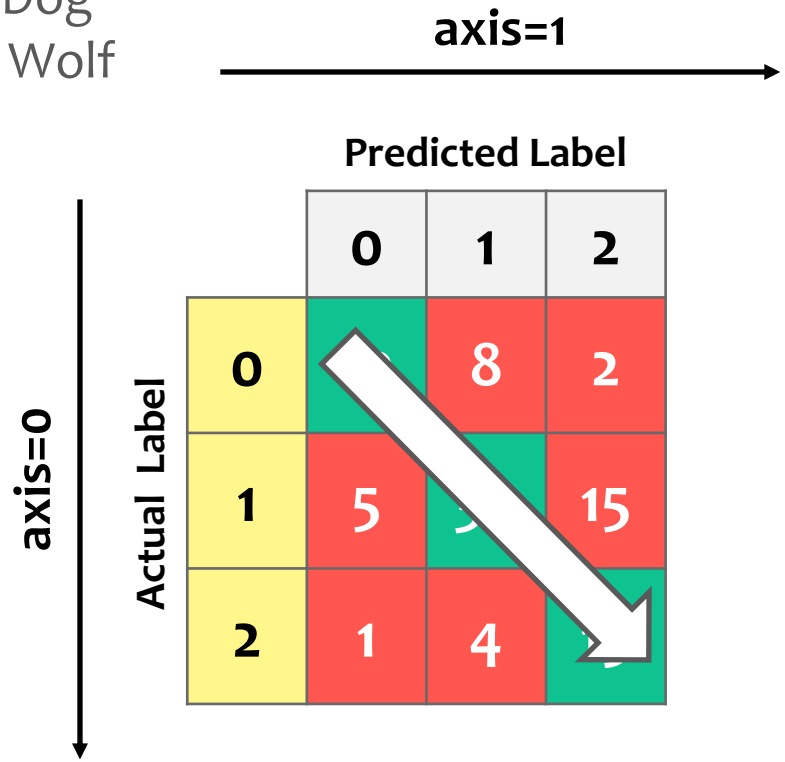


$$FP = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 16 & 42 & 32 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

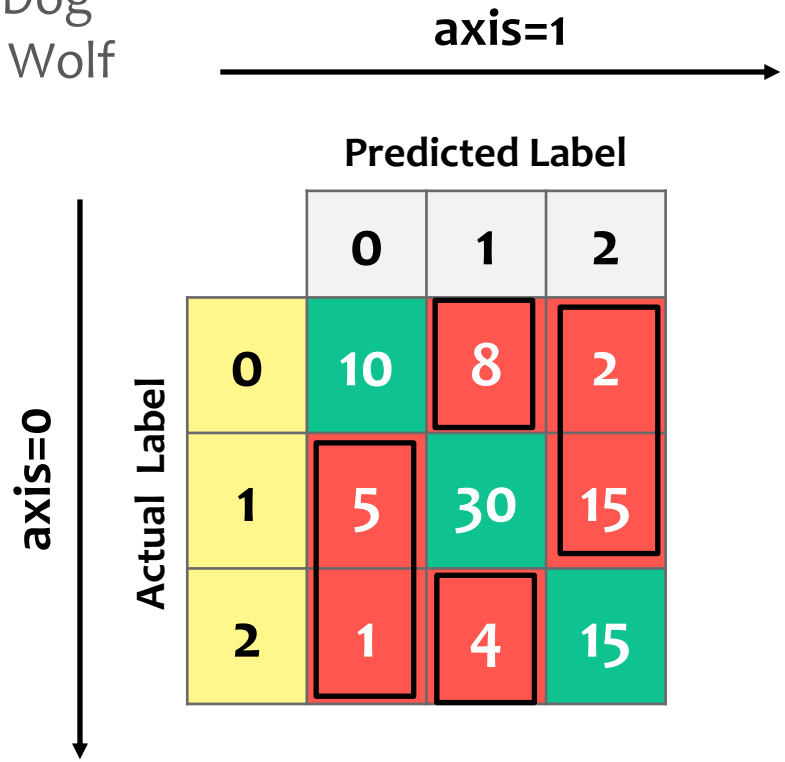


$$FP = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 16 & 42 & 32 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 30 & 15 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

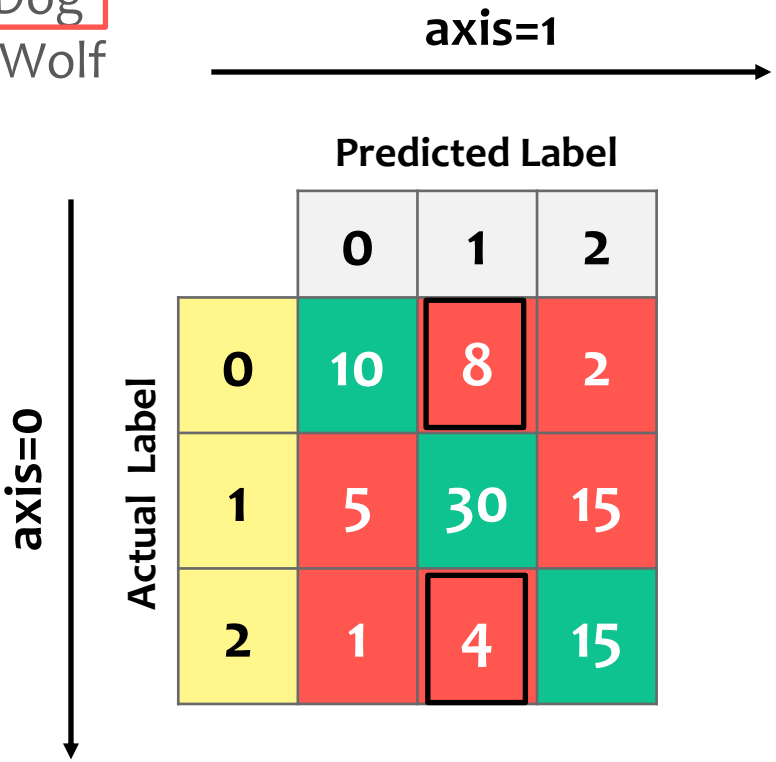


$$FP = \begin{array}{ccc} 0 & 1 & 2 \\ \hline 16 & 42 & 32 \end{array} - \begin{array}{ccc} 0 & 1 & 2 \\ \hline 10 & 30 & 15 \end{array} = \begin{array}{ccc} 0 & 1 & 2 \\ \hline 6 & 12 & 17 \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf



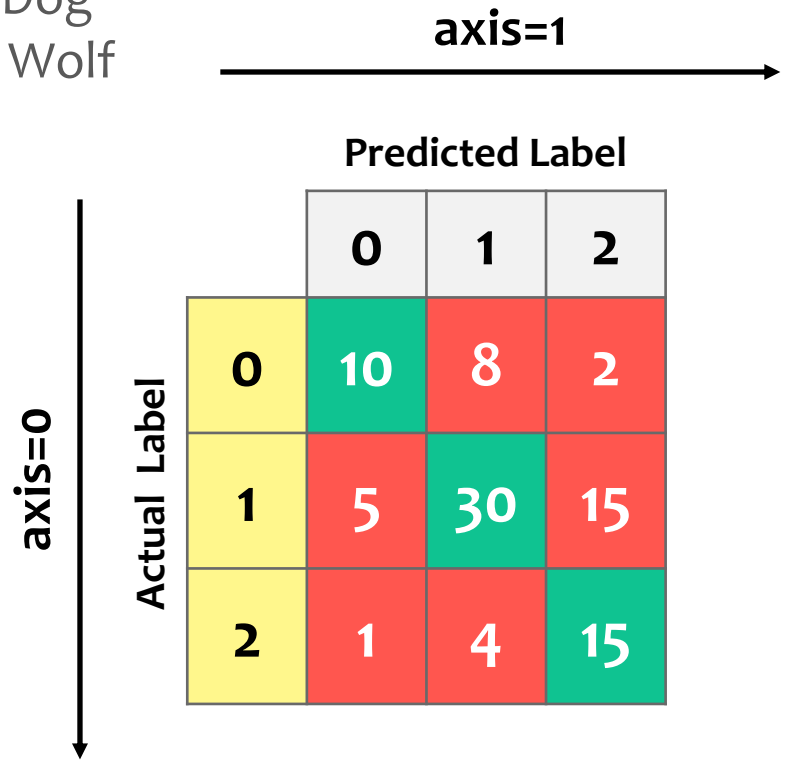
$$FP = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 16 & 42 & 32 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 30 & 15 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 6 & 12 & 17 \\ \hline \end{array}$$

$$FP[1] = FP_{Dog}$$

Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

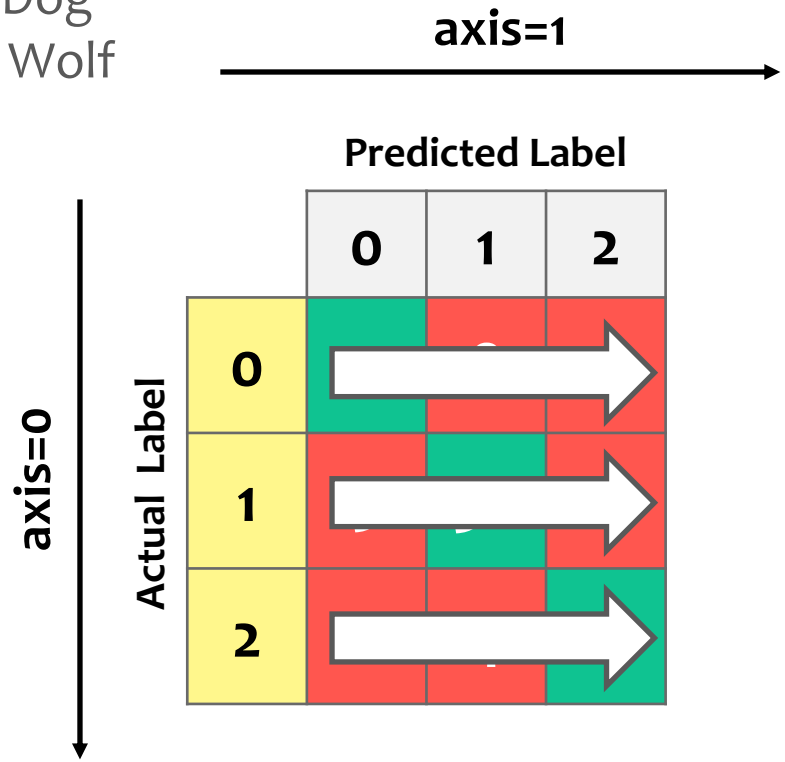


$$FN = \begin{array}{c} 0\ \ 1\ \ 2 \\ \square\square\square \\ - \\ 0\ \ 1\ \ 2 \\ \square\square\square \end{array} = \begin{array}{c} 0\ \ 1\ \ 2 \\ \square\square\square \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

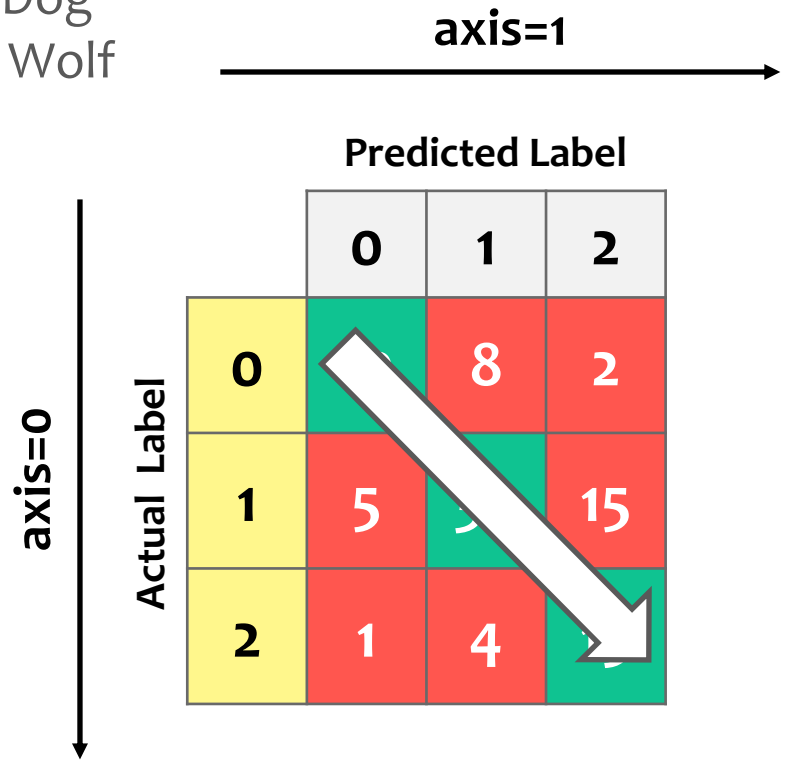


$$FN = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 20 & 50 & 20 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

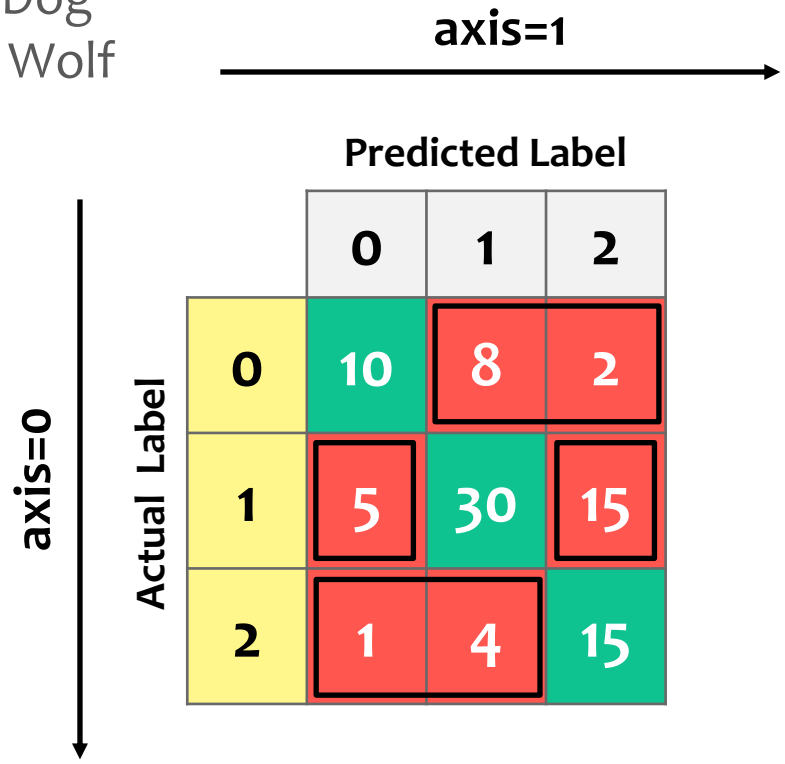


$$FN = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 20 & 50 & 20 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 30 & 15 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

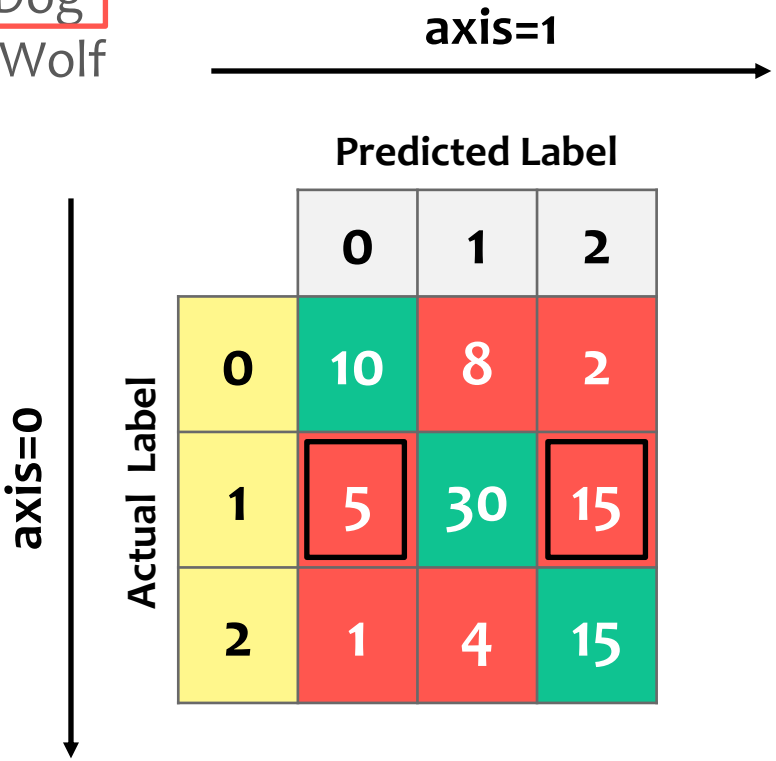


$$FN = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 20 & 50 & 20 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 30 & 15 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 20 & 5 \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf



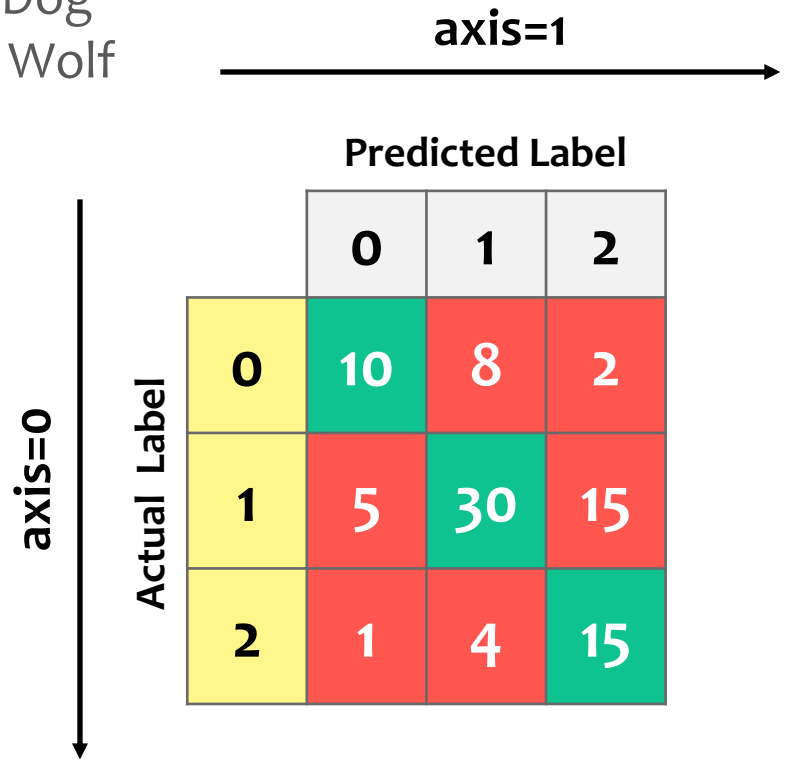
$$FN = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 20 & 50 & 20 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 30 & 15 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 10 & 20 & 5 \\ \hline \end{array}$$

$$FN[1] = FN_{Dog}$$

Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf



$$TN =$$

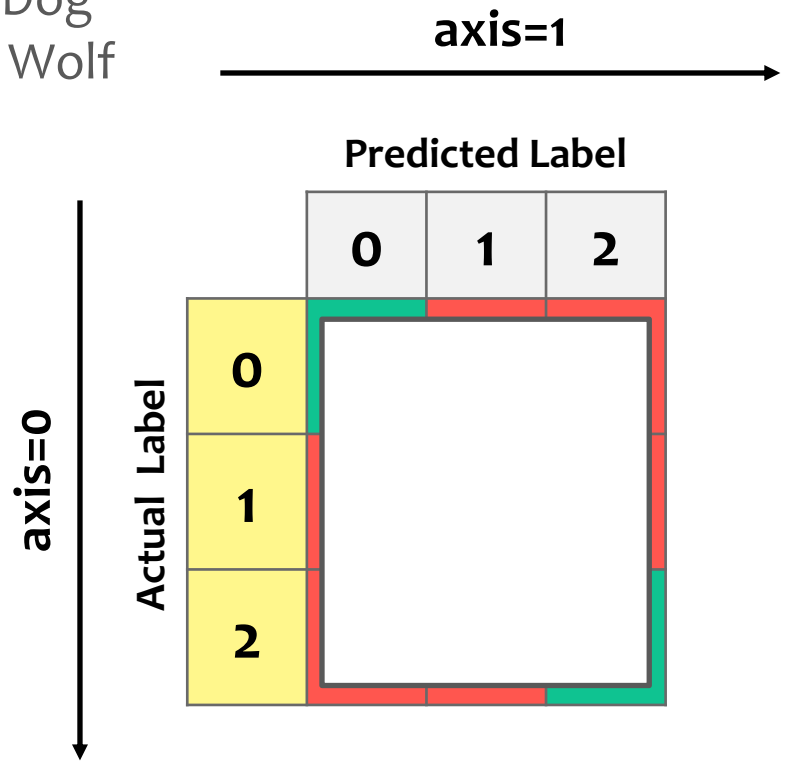
Confusion Matrix for Multiple Classes

```
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

90

$$TN =$$

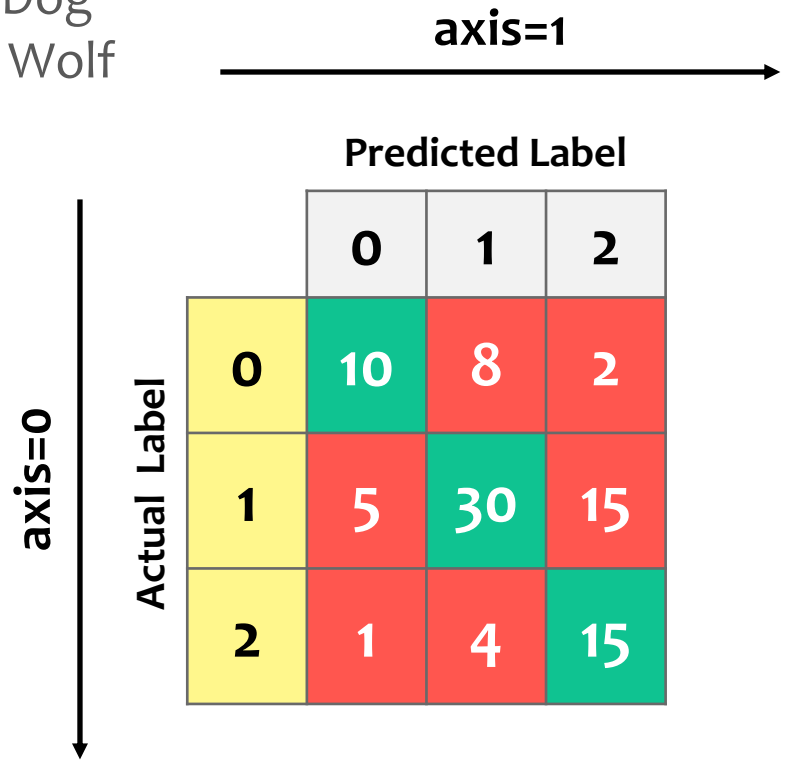
0: Cat
1: Dog
2: Wolf



Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

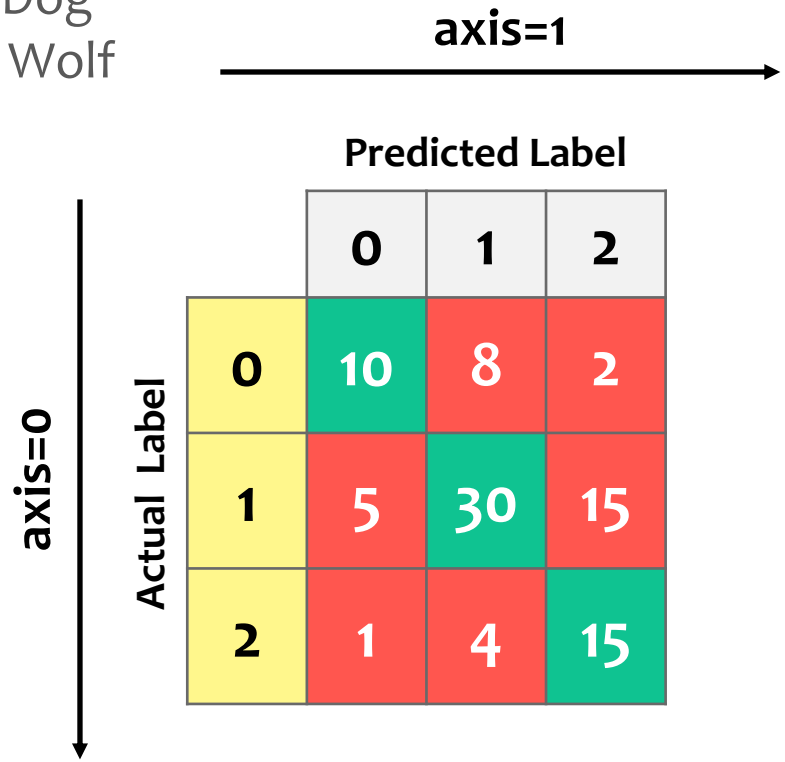


$$TN = \frac{90}{\begin{bmatrix} & 0 & 1 & 2 \\ TP & 10 & 30 & 15 \\ + & & & \\ FP & 6 & 12 & 17 \\ + & & & \\ FN & 10 & 20 & 5 \end{bmatrix}}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

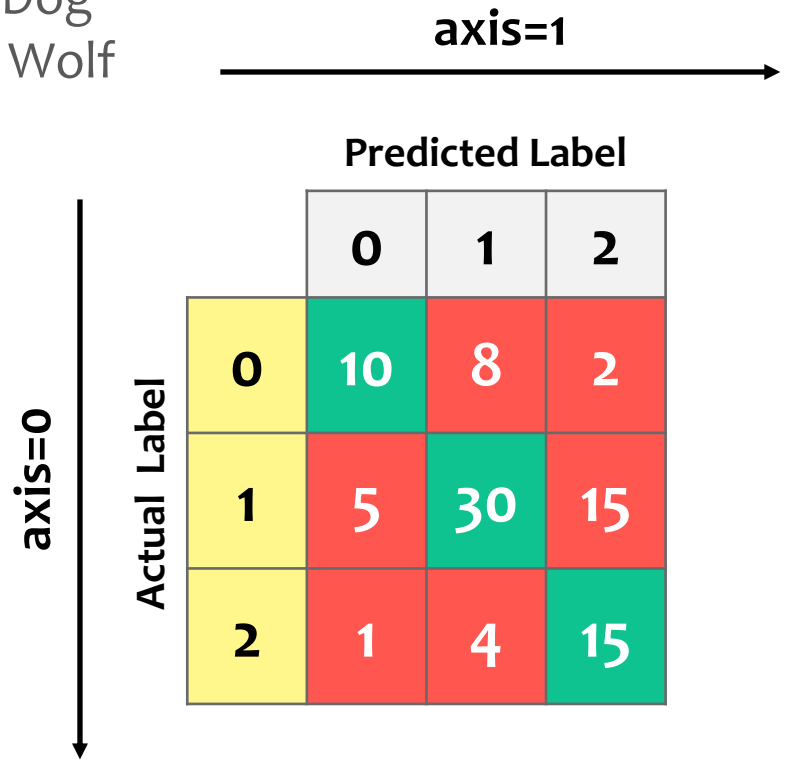


$$TN = \underbrace{\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 26 & 62 & 37 \\ \hline \end{array}}_{90} = \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline & & \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf



$$TN = \begin{array}{c} \overline{90} \\ \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 26 & 62 & 37 \\ \hline \end{array} \end{array} = \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 64 & 28 & 53 \\ \hline \end{array}$$

Confusion Matrix for Multiple Classes

```python
from sklearn.metrics import confusion_matrix
import numpy as np

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)
```

0: Cat
1: Dog
2: Wolf

axis=1

Predicted Label

axis=0

Actual Label

	0	1	2
0	10	8	2
1	5	30	15
2	1	4	15

90

$TN =$

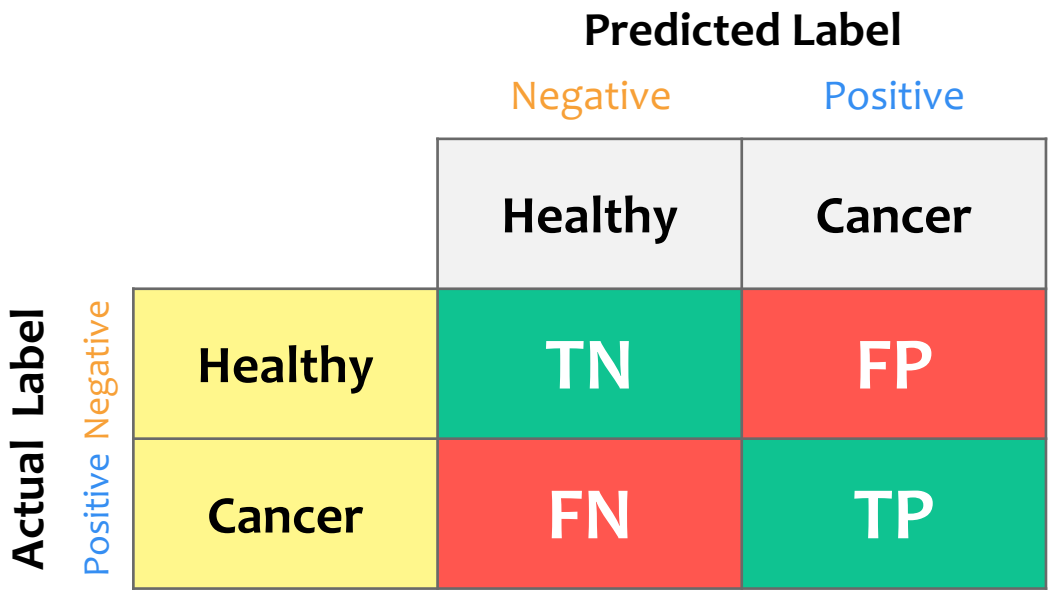
0	1	2
26	62	37

$=$

0	1	2
64	28	53

$$TN[1] = TN_{Dog}$$

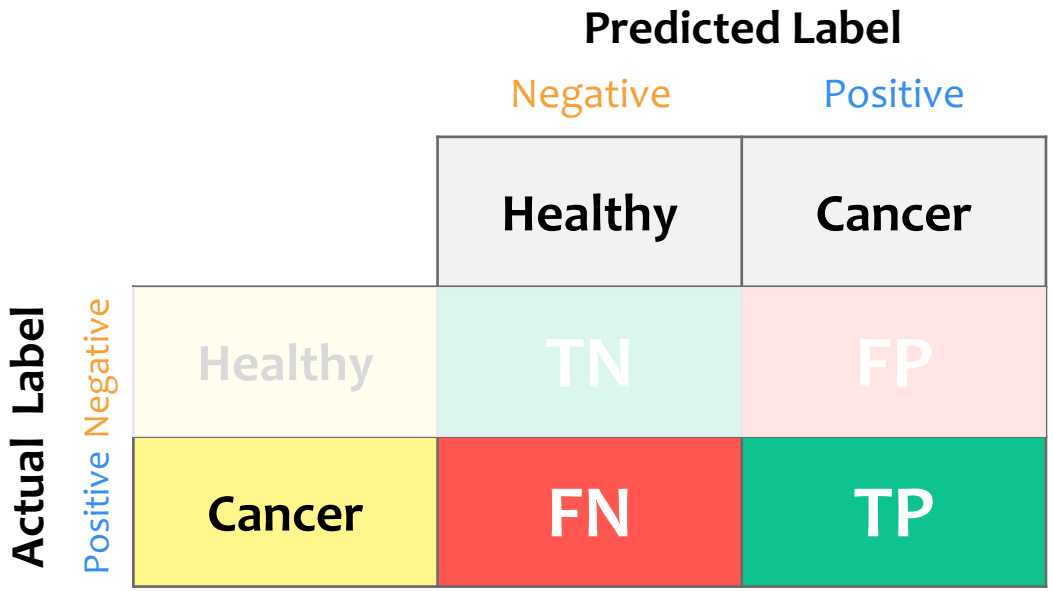
Sensitivity vs Specificity


Predicted Label
Negative
Positive
Healthy
Cancer
Actual Label
Positive Negative
Healthy
TN
FP
Cancer
FN
TP

Sensitivity (Recall or True Positive Rate)

What is the **proportion** of **positives** that were **correctly classified** as **positives**?
What is the **proportion** of **patients with cancer** that were **correctly identified**?
How sensitive is the classifier **to correctly identify patients with cancer**?

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

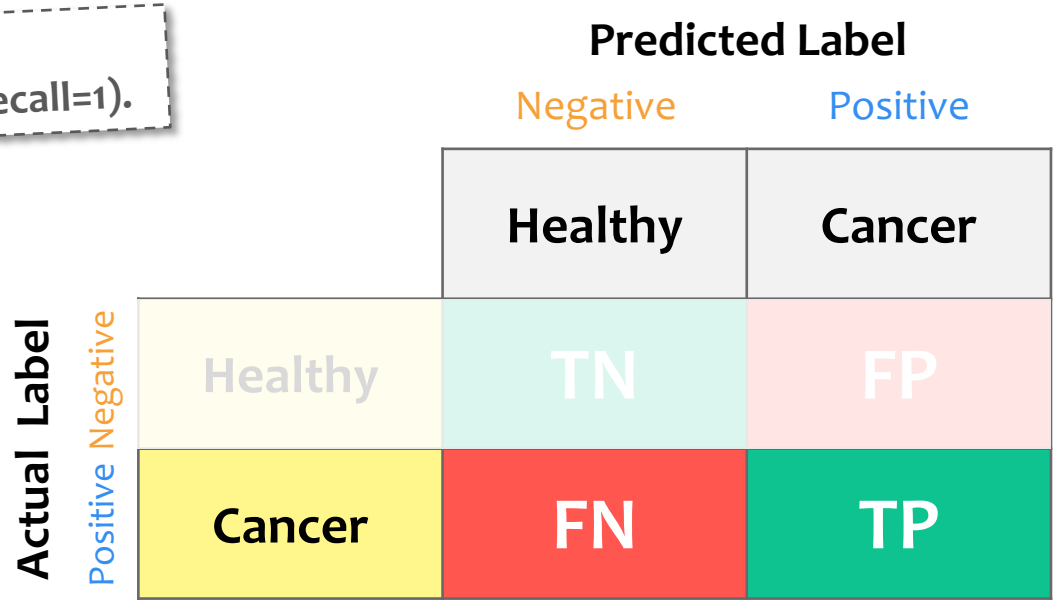


Sensitivity (Recall or True Positive Rate)

What is the **proportion** of **positives** that were **correctly classified** as **positives**?
What is the **proportion** of **patients with cancer** that were **correctly identified**?
How sensitive is the classifier **to correctly identify patients with cancer**?

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

⚠ It can **cheat** by guessing **everything** as **positive** (**recall=1**).



Sensitivity (Recall or True Positive Rate)

What is the **proportion** of **positives** that were **correctly classified** as **positives**?
What is the **proportion** of **patients with cancer** that were **correctly identified**?
How sensitive is the classifier **to correctly identify patients with cancer**?

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

sklearn.metrics.recall_score

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred)
```

By default, the **positive class** must have **value/label +1**.

Sensitivity (Recall or True Positive Rate)

What is the **proportion** of **positives** that were **correctly classified** as **positives**?
What is the **proportion** of **patients with cancer** that were **correctly identified**?
How sensitive is the classifier **to correctly identify patients with cancer**?

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

sklearn.metrics.recall_score

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred)
```

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, pos_label='Cancer')
```

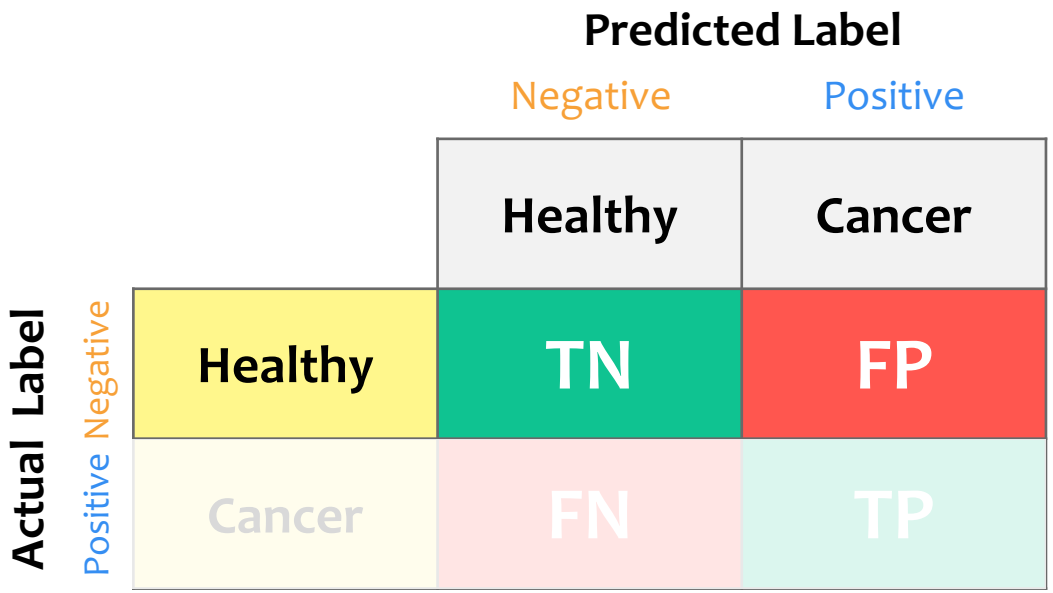
By default, the **positive class** must have **value/label +1**.

We can **indicate** which is the **value/label** of the **positive class.**

Specificity (True Negative Rate)

What is the **proportion** of **negatives** that were **correctly classified** as **negatives**?
What is the **proportion** of **healthy patients** that were **correctly identified**?

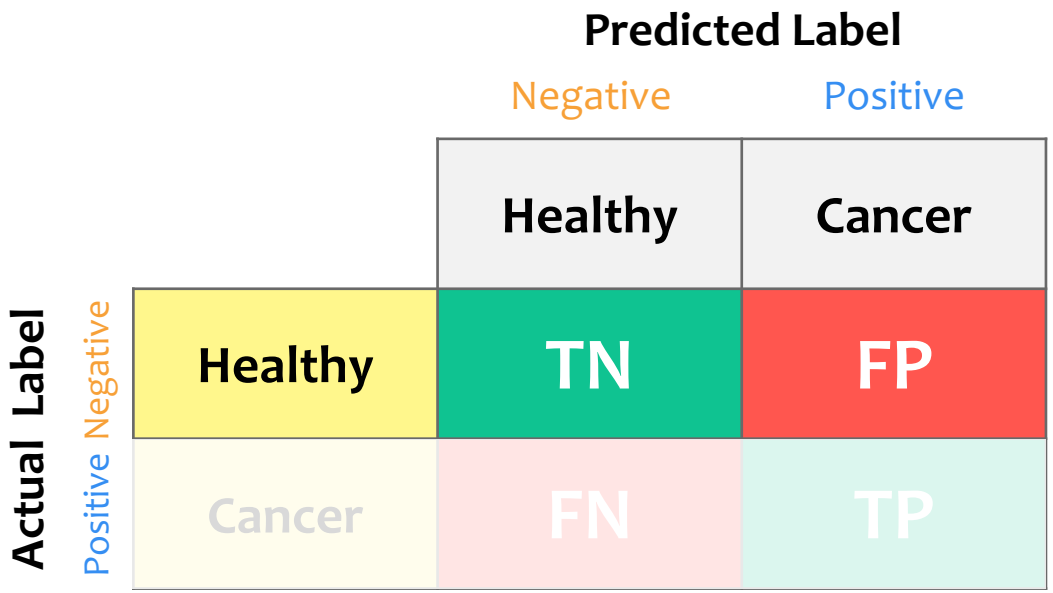
$$Specificity = \frac{TN}{TN + FP}$$



Specificity (True Negative Rate)

What is the **proportion** of **negatives** that were **correctly classified** as **negatives**?
What is the **proportion** of **healthy patients** that were **correctly identified**?

$$Specificity = \frac{TN}{TN + FP}$$



⚠ It can **cheat** by guessing **everything** as **negative** (**specificity=1**).

Specificity (True Negative Rate)

What is the **proportion** of **negatives** that were **correctly classified** as **negatives**?
What is the **proportion** of **healthy patients** that were **correctly identified**?

$$Specificity = \frac{TN}{TN + FP}$$

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

specificity = tn / (tn + fp)
```

```
from sklearn.metrics import confusion_matrix

tn, fp, fn, tp = confusion_matrix(y_true, y_pred,
                                  labels=['Healthy', 'Cancer']).ravel()

specificity = tn / (tn + fp)
```

Specificity (True Negative Rate)

What is the **proportion** of **negatives** that were **correctly classified** as **negatives**?
What is the **proportion** of **healthy patients** that were **correctly identified**?

$$Specificity = \frac{TN}{TN + FP}$$

For the **binary case**, we could say that ***specificity*** is the **'sensitivity/recall'** of the **negative class**.

If we replace N with P, we would have:

$$\frac{TP}{TP + FN}$$

which is exactly the formula for **'sensitivity/recall'** .

Specificity (True Negative Rate)

What is the **proportion** of **negatives** that were **correctly classified** as **negatives**?
What is the **proportion** of **healthy patients** that were **correctly identified**?

$$Specificity = \frac{TN}{TN + FP}$$

sklearn.metrics.recall_score

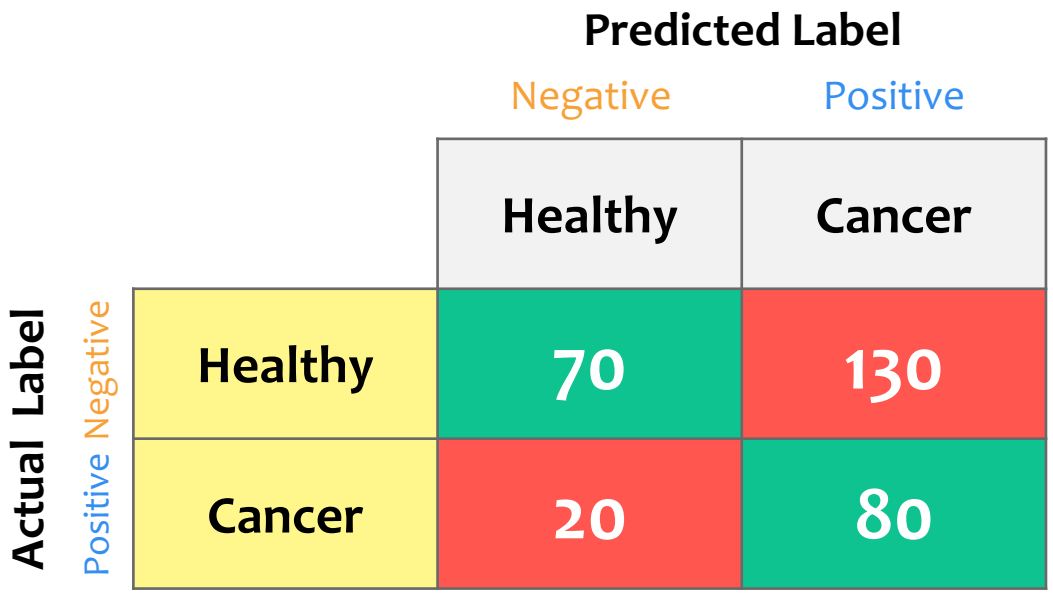
```
from sklearn.metrics import recall_score

# for two classes, we can compute sensitivity by assuming
# the negative class as the positive class in `recall_score`.
recall_score(y_true_bin, y_pred_bin_knn, pos_label='Healthy')
```

For the **binary case**, we could say that ***specificity*** is the **'sensitivity/recall'** of the **negative class**.

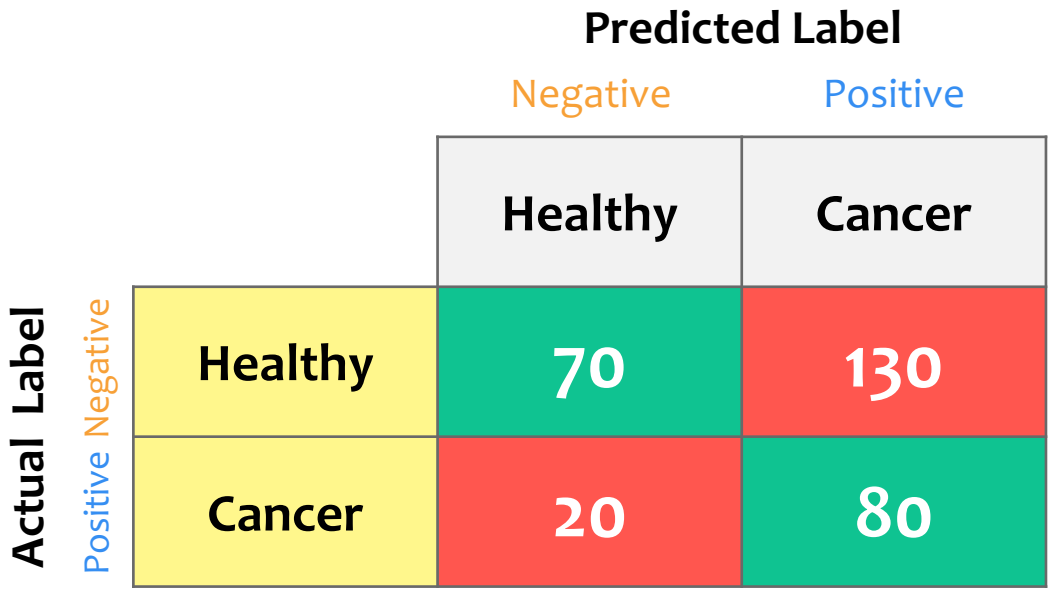
If we replace N with P, we would have:

$$\frac{TP}{TP + FN}$$

which is exactly the formula for **'sensitivity/recall'** .

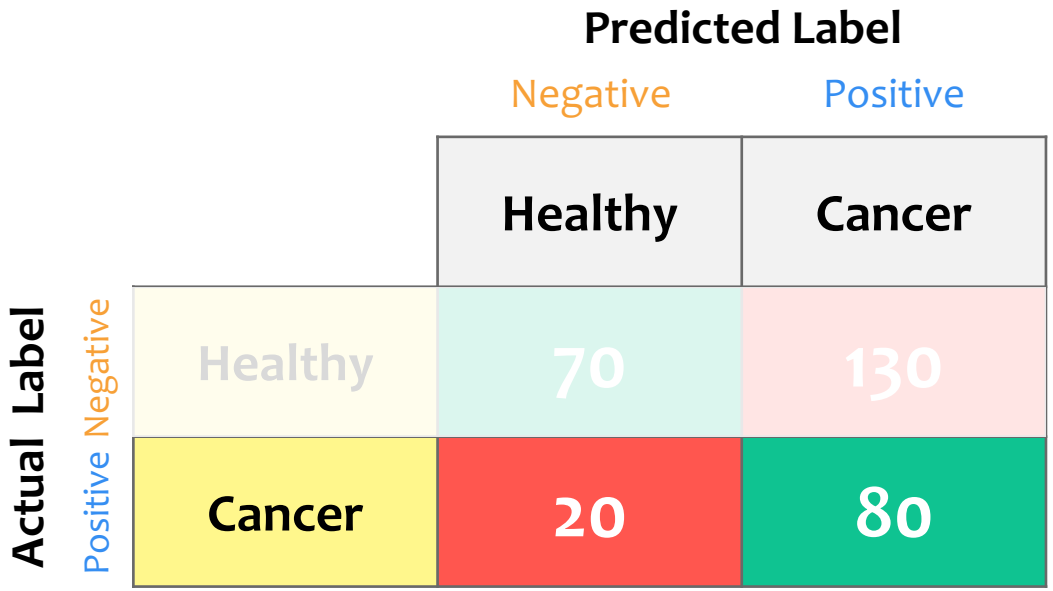

Predicted Label
Negative
Positive
Healthy
Cancer
Actual Label
Positive Negative
Healthy
70
130
Cancer
20
80

$$Sensitivy = \frac{TP}{TP + FN}$$



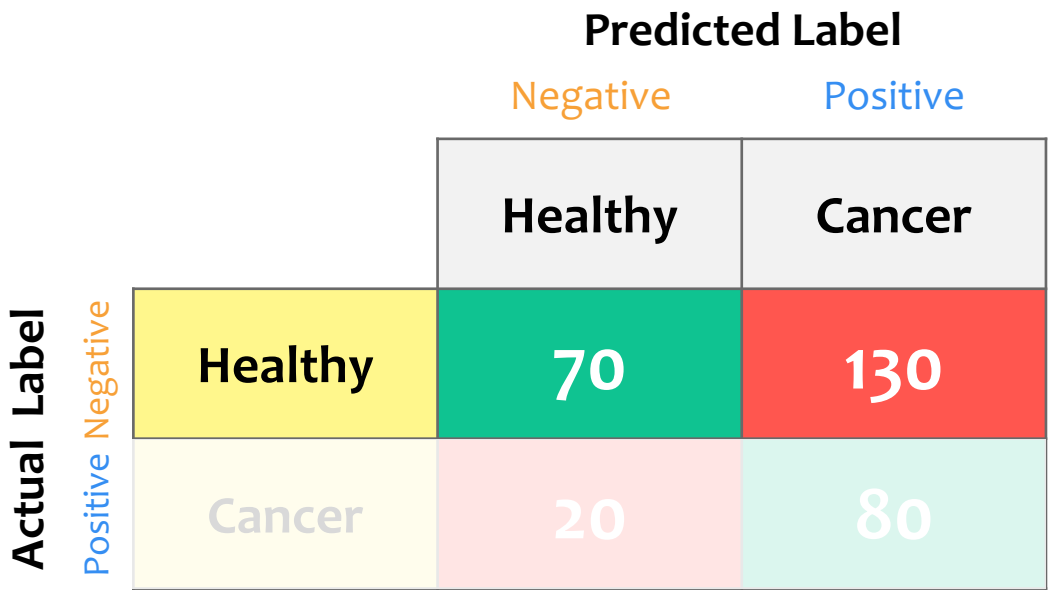
$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

80% of the **people with cancer** were **correctly identified**.



$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

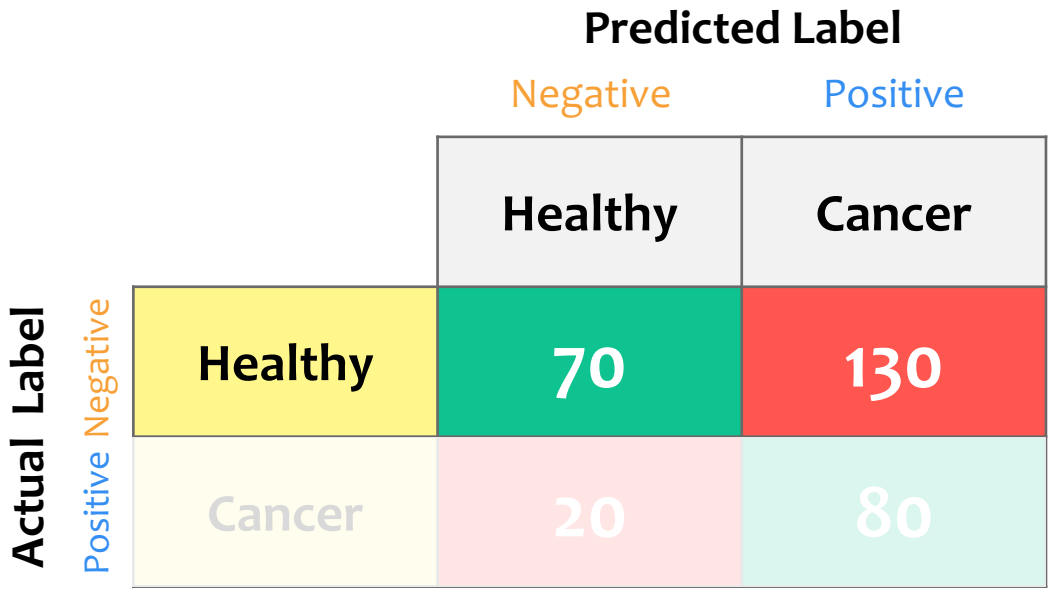
$$Specificity = \frac{TN}{TN + FP}$$



$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

$$Specificity = \frac{70}{70 + 130} = 0.35$$

35% of the **healthy people** were **correctly identified.**



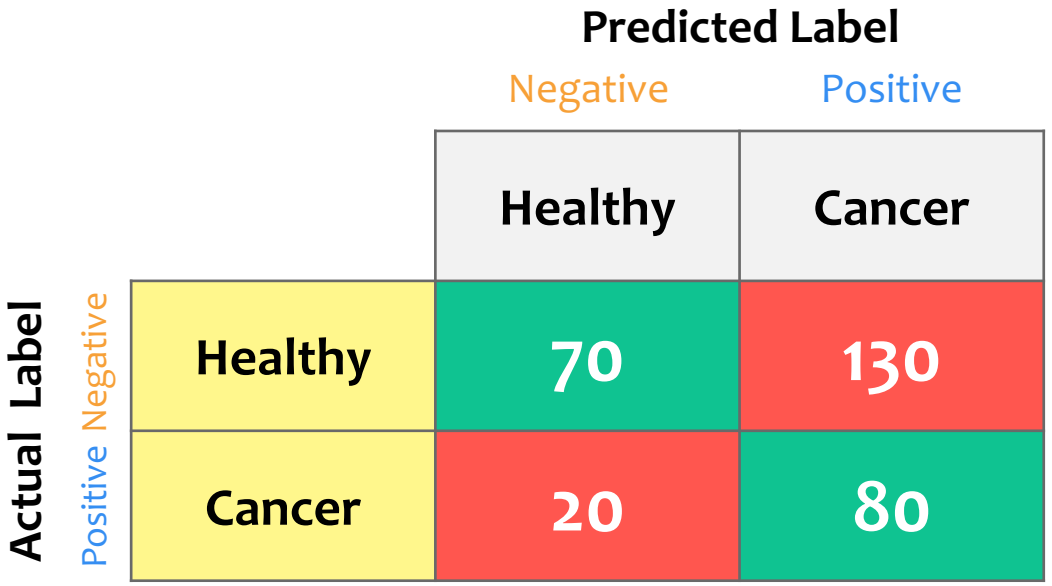
Confusion matrix for **classifier A**
(e.g., KNN)

Actual Label	Predicted Label: Negative (Healthy)	Predicted Label: Positive (Cancer)
Negative (Healthy)	70	130
Positive (Cancer)	20	80

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Actual Label	Predicted Label: Negative (Healthy)	Predicted Label: Positive (Cancer)
Negative (Healthy)	80	120
Positive (Cancer)	50	50

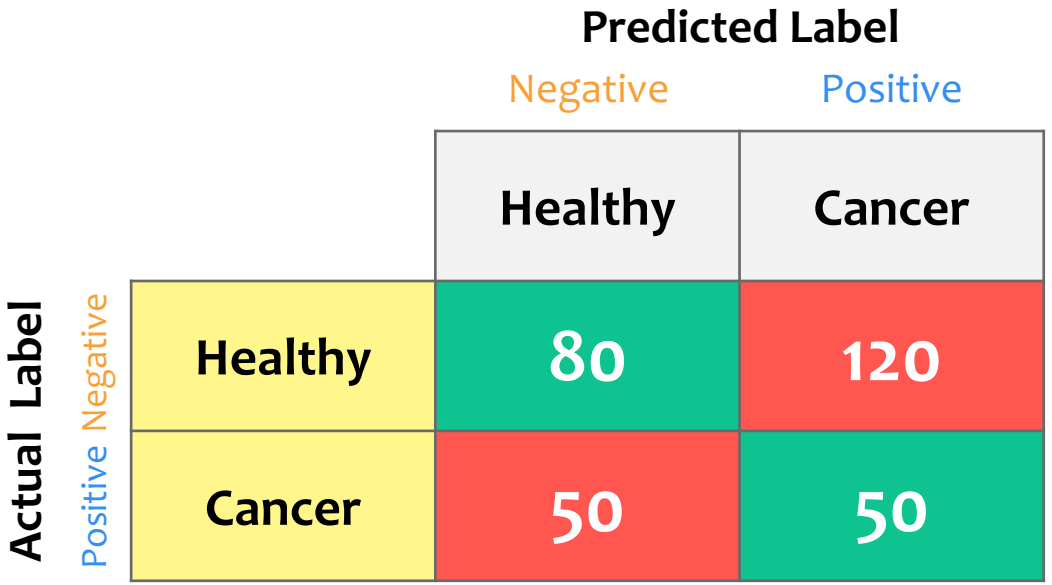
Confusion matrix for **classifier A**
(e.g., KNN)



$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

$$Specificity = \frac{70}{70 + 130} = 0.35$$

Confusion matrix for **classifier B**
(e.g., Logistic Regression)



$$Sensitivy = \frac{50}{50 + 50} = 0.5$$

$$Specificity = \frac{80}{80 + 120} = 0.4$$

Confusion matrix for **classifier A**
(e.g., KNN)

Actual Label (Positive / Negative)	Predicted Label: Healthy (Negative)	Predicted Label: Cancer (Positive)
Healthy	70	130
Cancer	20	80

$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Actual Label (Positive / Negative)	Predicted Label: Healthy (Negative)	Predicted Label: Cancer (Positive)
Healthy	80	120
Cancer	50	50

$$Sensitivy = \frac{50}{50 + 50} = 0.5$$

Sensitivity tells us that **KNN** is better at **correctly classifying positives (people with cancer)**

$$Specificity = \frac{80}{80 + 120} = 0.4$$

Confusion matrix for **classifier A**
(e.g., KNN)

Actual Label	Predicted Label: Healthy (Negative)	Predicted Label: Cancer (Positive)
Healthy (Negative)	70	130
Cancer (Positive)	20	80

$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

$$Specificity = \frac{70}{70 + 130} = 0.35$$

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Actual Label	Predicted Label: Healthy (Negative)	Predicted Label: Cancer (Positive)
Healthy (Negative)	80	120
Cancer (Positive)	50	50

Specificity tells us that **Logistic Regression** is better at **correctly classifying negatives (healthy people)**

$$Specificity = \frac{80}{80 + 120} = 0.4$$

Confusion matrix for **classifier A**
(e.g., KNN)

Actual Label (Positive Negative)	Predicted Label: Healthy (Negative)	Predicted Label: Cancer (Positive)
Healthy	70	130
Cancer	20	80

$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

$$Specificity = \frac{70}{70 + 130} = 0.35$$

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

Actual Label (Positive Negative)	Predicted Label: Healthy (Negative)	Predicted Label: Cancer (Positive)
Healthy	50	50
Cancer	120	80

We would choose the **KNN model** if **correctly identifying patients with cancer (positives) is more important** than **correctly identifying healthy people (negatives).**

$$Sensitivy = \frac{50}{50 + 50} = 0.5$$

$$Specificity = \frac{80}{80 + 120} = 0.4$$

Confusion matrix for classifier A
(e.g., KNN)

Predicted Label

Negative Positive

Healthy Cancer

Actual Label

Otherwise, we would choose the **Logistic model** if **correctly identifying healthy people (negatives) is more important** than **correctly identifying patients with cancer (positives).**

$$Sensitivy = \frac{80}{80 + 20} = 0.8$$

$$Specificity = \frac{70}{70 + 130} = 0.35$$

Confusion matrix for **classifier B**
(e.g., Logistic Regression)

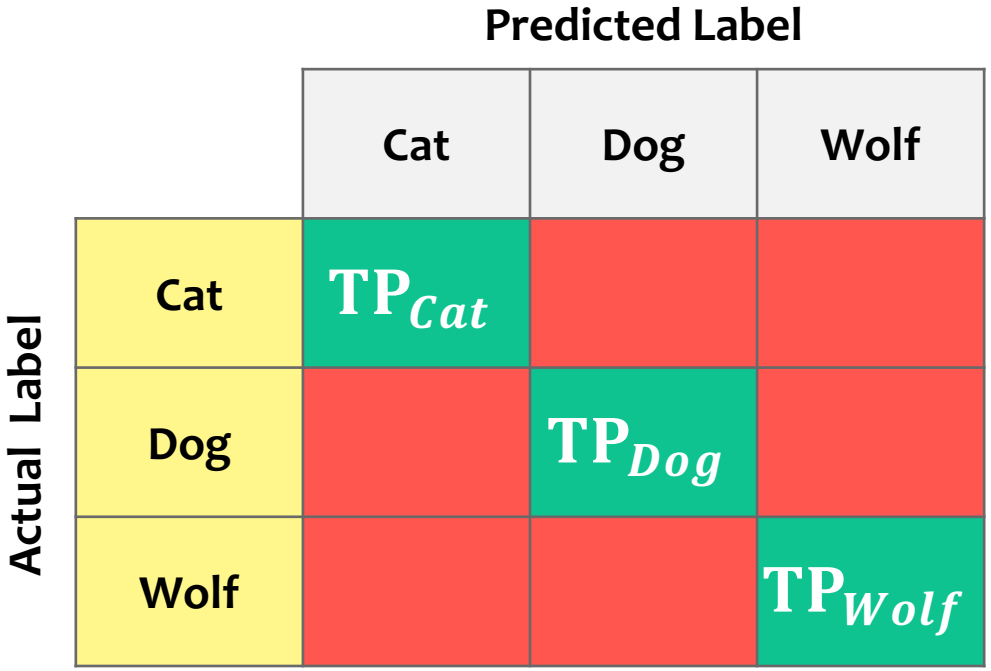
Predicted Label

Actual Label	Negative	Positive
	Healthy	**Cancer**
Healthy (Negative)	80	120
Cancer (Positive)	50	50

$$Sensitivy = \frac{50}{50 + 50} = 0.5$$

$$Specificity = \frac{80}{80 + 120} = 0.4$$

Sensitivity for Multiple Classes



Sensitivity for Multiple Classes

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

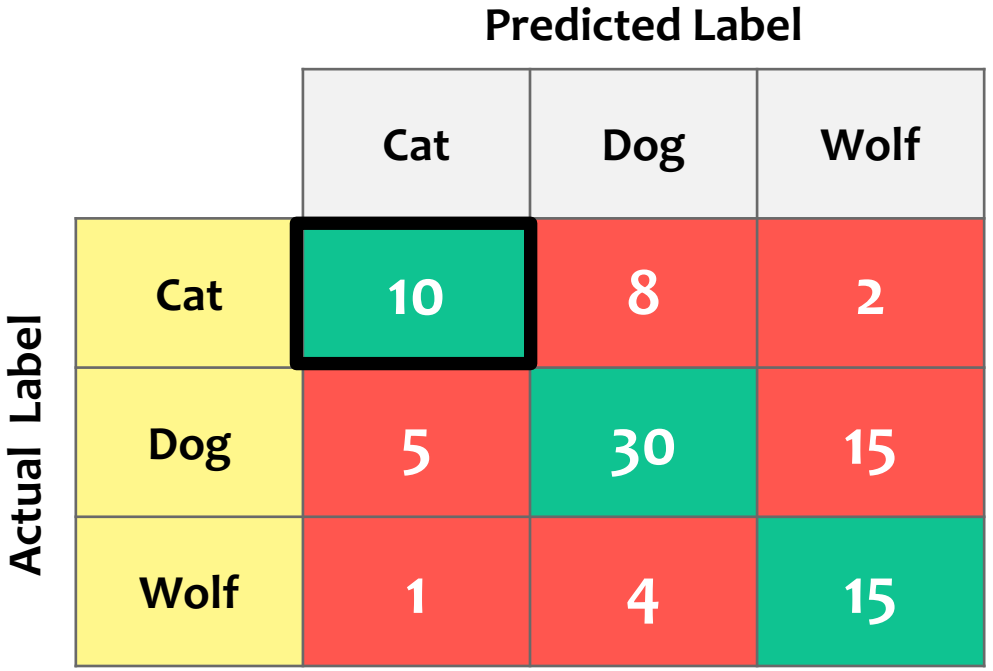
Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} =$$

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

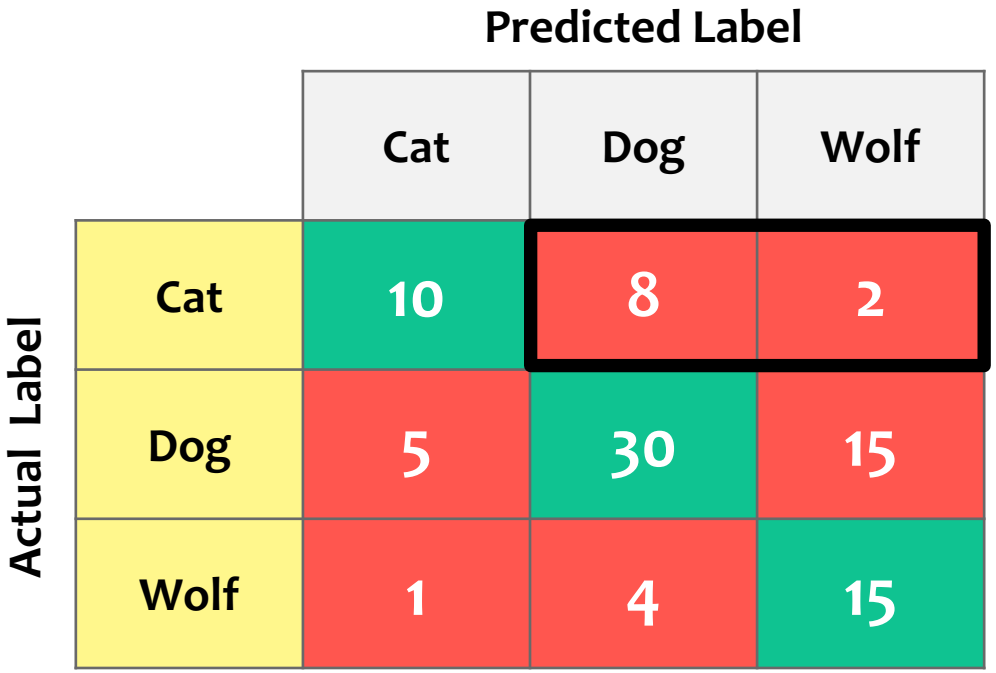
Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 +}$$



Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 + 10}$$



Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 + 10} = 0.5$$

$Sensitivy_{Cat}$ tells us that **50% of cats** were **correctly identified**.

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 + 10} = 0.5$$

$Sensitivy_{Cat}$ tells us that **50% of cats** were **correctly identified**.

```
from sklearn.metrics import confusion_matrix
recall_score(y_true, y_pred, average=None, labels=['Cat', 'Dog', 'Wolf'])
```

```
array([0.5 , 0.6 , 0.75])
```

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 + 10} = 0.5$$

$Sensitivy_{Cat}$ tells us that **50% of cats** were **correctly identified**.

```
from sklearn.metrics import confusion_matrix
recall_score(y_true, y_pred, average=None, labels=['Cat', 'Dog', 'Wolf'])
```

```
array([0.5 , 0.6 , 0.75])
```

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 + 10} = 0.5$$

$Sensitivy_{Cat}$ tells us that **50% of cats** were **correctly identified**.

```
from sklearn.metrics import confusion_matrix
recall_score(y_true, y_pred, average=None, labels=['Cat', 'Dog', 'Wolf'])
```

```
array([0.5 , 0.6 , 0.75])
```

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Sensitivity for Multiple Classes

$$Sensitivy_{Cat} = Recall_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FN_{Cat}} = \frac{10}{10 + 10} = 0.5$$

$Sensitivy_{Cat}$ tells us that **50% of cats** were **correctly identified**.

```
from sklearn.metrics import confusion_matrix
recall
array
```

We can return a **single "global"** sensitivity by **averaging** the metrics according to some strategy.

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Single Sensitivity for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Sensitivity_{macro} = Recall_{macro} = \frac{\sum_{j=0}^{c-1} Sensitivity_j}{c}$$

Single Sensitivity for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Sensitivity_{macro} = Recall_{macro} = \frac{\sum_{j=0}^{c-1} Sensitivity_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

Single Sensitivity for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Sensitivity_{macro} = Recall_{macro} = \frac{\sum_{j=0}^{c-1} Sensitivity_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, average='macro',
             labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6166666666666667
```

Single Sensitivity for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Sensitivity_{macro} = Recall_{macro} = \frac{\sum_{j=0}^{c-1} Sensitivity_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, average='macro',
             labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6166666666666667
```

micro

Calculate **metrics globally** by counting the total true positives, false negatives and false positives.

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$Sensitivity_{micro} = Recall_{micro} = \frac{\sum_{j=0}^{c-1} TP_j}{m}$$

Single Sensitivity for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Sensitivity_{macro} = Recall_{macro} = \frac{\sum_{j=0}^{c-1} Sensitivity_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, average='macro',
             labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6166666666666667
```

micro

Calculate **metrics globally** by counting the total true positives, false negatives and false positives.

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$Sensitivity_{micro} = Recall_{micro} = \frac{\sum_{j=0}^{c-1} TP_j}{m}$$

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, average='micro',
             labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6111111111111112
```

Single Sensitivity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1,**
m the number samples, and m_j is the number of true samples of class j.

$$Sensitivity_{weighted} = Recall_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Sensitivity_j}{m}$$

Single Sensitivity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1,**
m the number samples, and m_j is the number of true samples of class j.

$$Sensitivity_{weighted} = Recall_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Sensitivity_j}{m}$$

weight (w_j) for class j

Single Sensitivity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

$$Sensitivity_{weighted} = Recall_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Sensitivity_j}{m}$$

weight (w_j) for class j

It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

Single Sensitivity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, average='weighted',
             labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6111111111111112
```

$$Sensitivity_{weighted} = Recall_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Sensitivity_j}{m}$$

weight (w_j) for class j

It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

Single Sensitivity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred, average='weighted',
             labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6111111111111112
```

$$Sensitivity_{weighted} = Recall_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Sensitivity_j}{m}$$

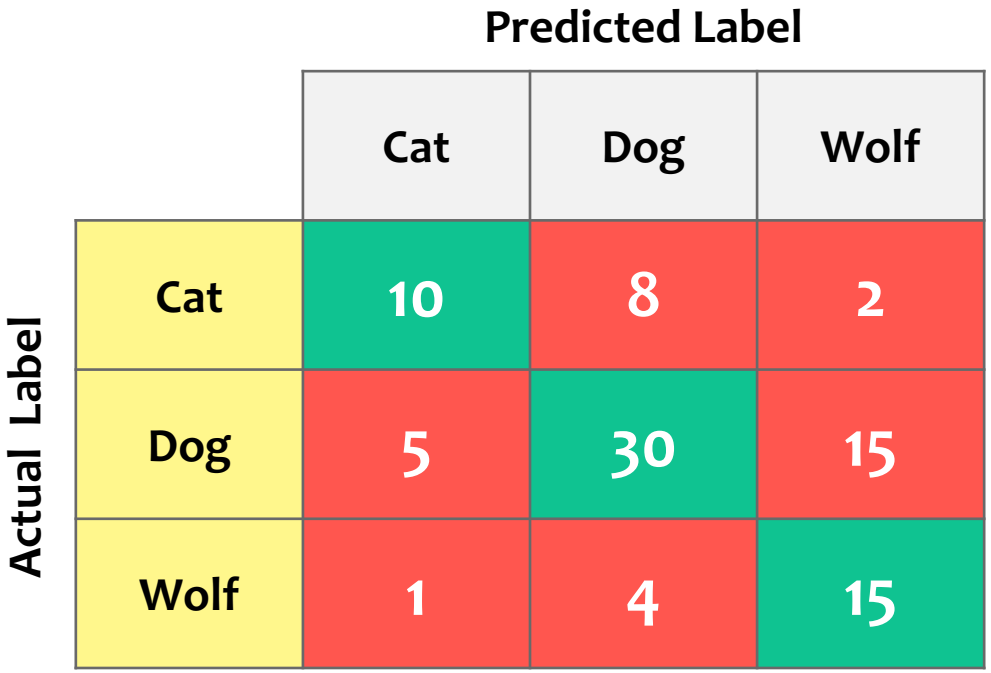
weight (w_j) for class j

⚠ It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

💡 $Sensitivy_{weighted}$ is equal to **accuracy**.

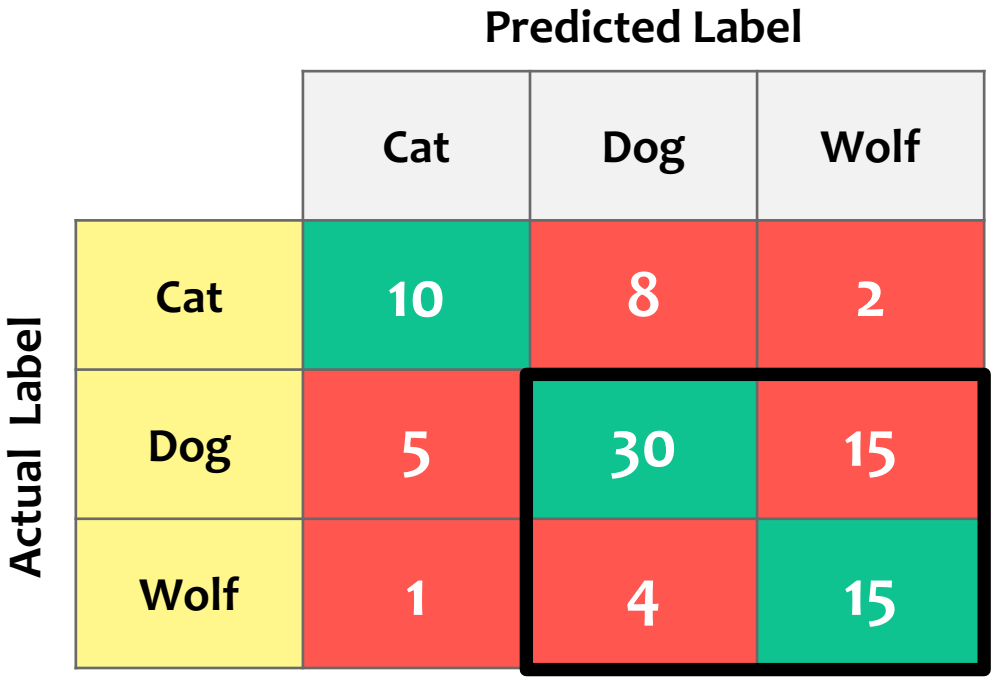
Specificity for Multiple Classes

$$Specificity_{Cat} = \frac{TN_{Cat}}{TN_{Cat} + FP_{Cat}}$$



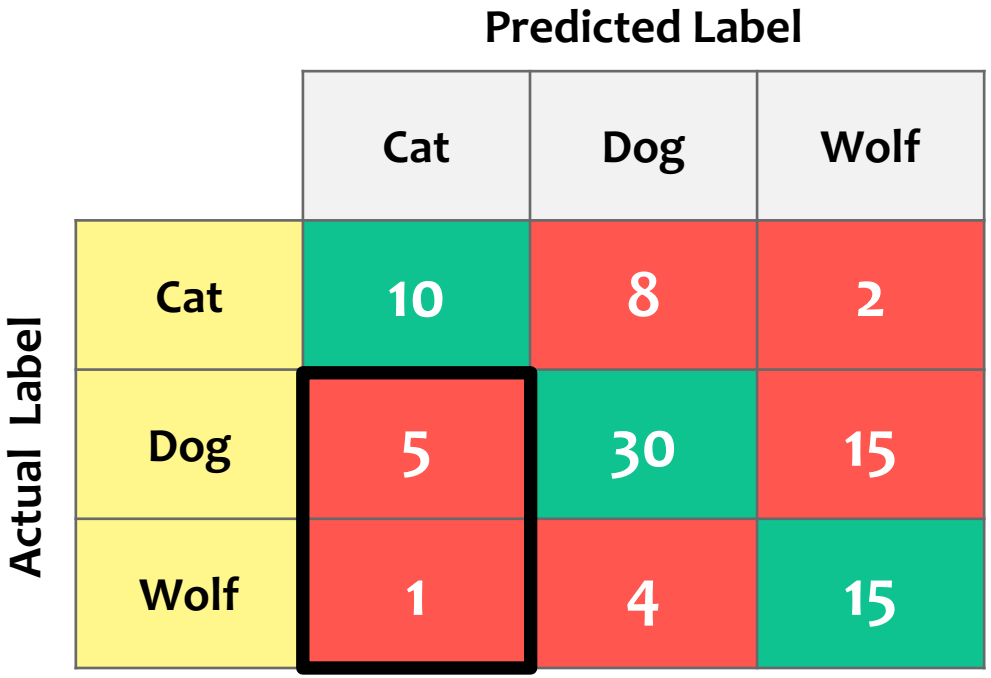
Specificity for Multiple Classes

$$Specificity_{Cat} = \frac{TN_{Cat}}{TN_{Cat} + FP_{Cat}} = \frac{64}{64 +}$$



Specificity for Multiple Classes

$$Specificity_{Cat} = \frac{TN_{Cat}}{TN_{Cat} + FP_{Cat}} = \frac{64}{64 + 6}$$



Specificity for Multiple Classes

$$Specificity_{Cat} = \frac{TN_{Cat}}{TN_{Cat} + FP_{Cat}} = \frac{64}{64 + 6} = 0.9142$$

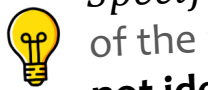
$Specificity_{Cat}$ tells us that **91.42%** of the pets that are **not cats** were **not identified as cats**.

Predicted Label

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Specificity for Multiple Classes

$$Specificity_{Cat} = \frac{TN_{Cat}}{TN_{Cat} + FP_{Cat}} = \frac{64}{64 + 6} = 0.9142$$



$Specificity_{Cat}$ tells us that **91.42%** of the pets that are **not cats** were **not identified as cats.**

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

specifity = TN / (TN + FP)
```

```
[0.91428571 0.7         0.75714286]
```

Actual Label

Wolf | 1 | 4 | 15

Specificity for Multiple Classes

$$Specificity_{Cat} = \frac{TN_{Cat}}{TN_{Cat} + FP_{Cat}} = \frac{64}{64 + 6} = 0.9142$$


```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

specifity = TN / (TN + FP)
```

```
[0.91428571 0.7         0.75714286]
```

Actual Label

Single Specificity for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Specificity_{macro} = \frac{\sum_{j=0}^{c-1} Specificity_j}{c}$$

⚠ Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```python
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_true, y_pred,
                      labels=['Cat', 'Dog', 'Wolf'])

TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

specifity = TN / (TN + FP)
specifity_macro = specifity.mean()
```

```
[0.91428571 0.7        0.75714286]
0.7904761904761904
```

Single Specificity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

$$Specificity_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Specificity_j}{m}$$

weight (w_j) for class j

Single Specificity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

$$Specificity_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Specificity_j}{m}$$

weight (w_j) for class j

Single Specificity for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

$$Specificity_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Specificity_j}{m}$$

weight (w_j) for class j

> It **take into account class proportion (imbalance)** for computation.
> Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

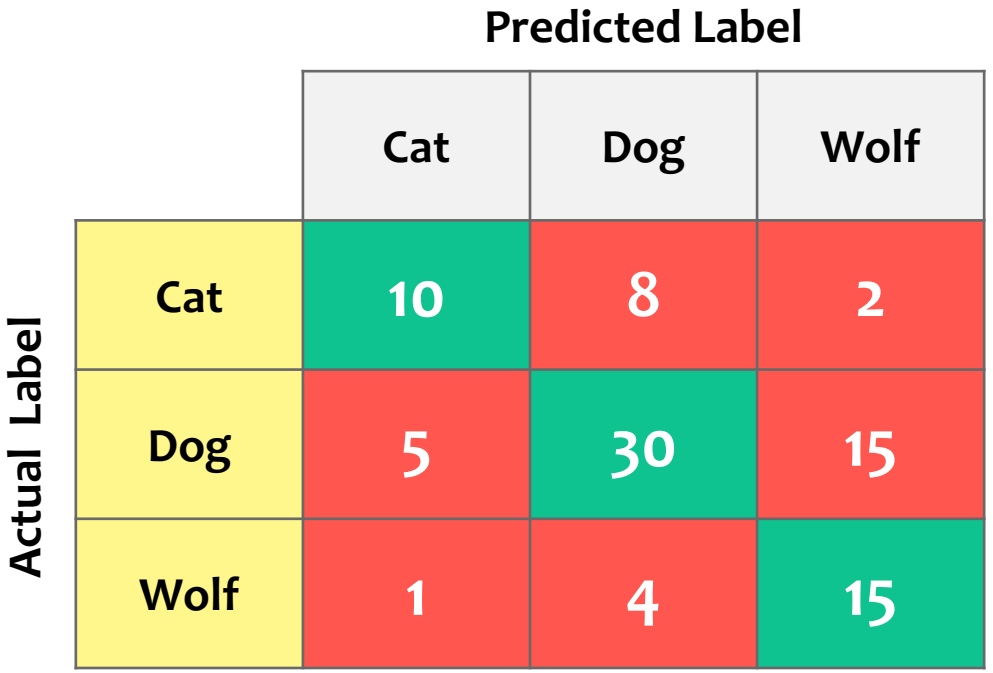
```
...

specificity = TN / (TN + FP)

# weighting
class_sizes = cm.sum(axis=1)
number_of_samples = cm.sum()
weights = class_sizes / number_of_samples

specificity_weighted = np.sum(specificity * weights)
```

```
[0.91428571 0.7        0.75714286]

[20 50 20]
90
[0.22222222 0.55555556 0.22222222]

0.7603174603174603
```

Exercise: Compute…

$Sensitivity_{Dog}$ $Specificity_{Dog}$

$Sensitivity_{Wolf}$ $Specificity_{Wolf}$



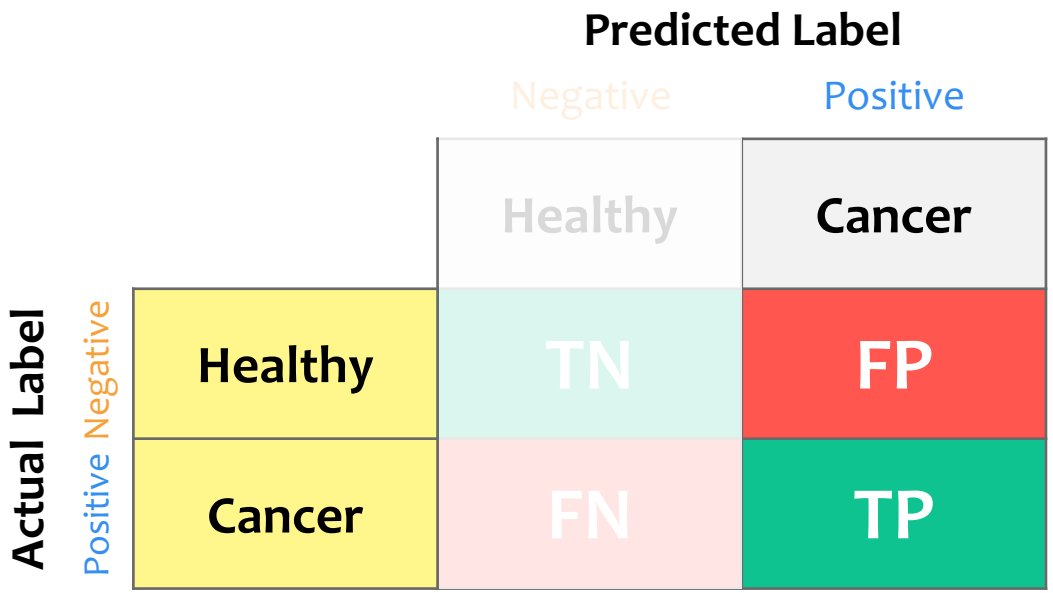
More Metrics

Precision

What is the **proportion** of true positives among all instances **classified** (correctly and incorrectly) as positives?
How precise is the positive classification?

"From all patients classified as **cancer**, how many (**proportion**) actually had **cancer**?"

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



Precision

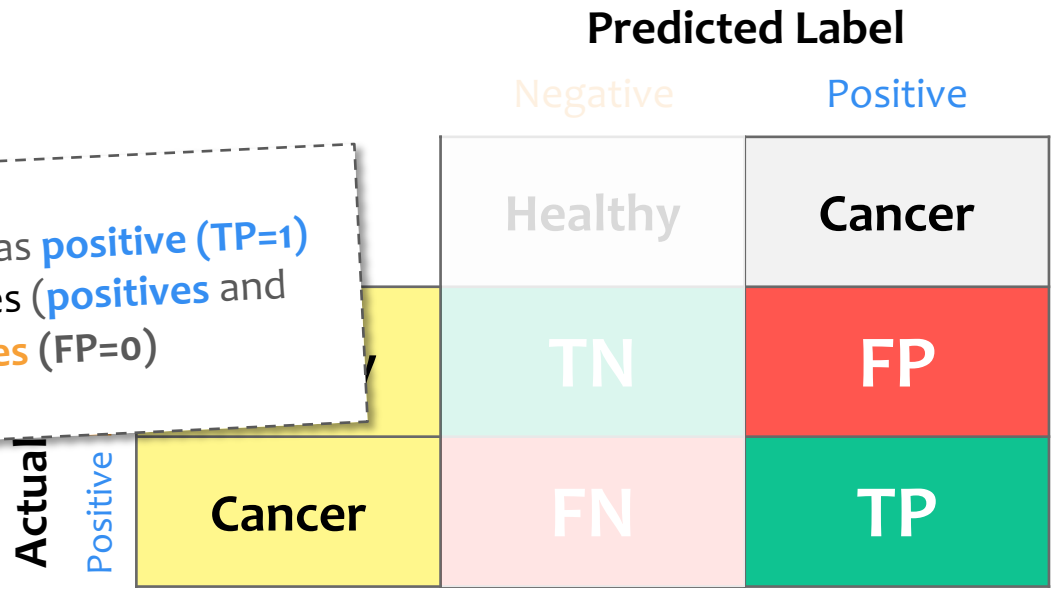
What is the **proportion** of **true positives** among all instances **classified** (**correctly** and **incorrectly**) as **positives**?
How precise is the **positive** classification?

"From all patients classified as **cancer**, how many (**proportion**) actually had **cancer**?"

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

Negatives are left out of the metric.

We can **cheat** by:
- Guess 1 **correct** guess as **positive (TP=1)**
- Guess all other samples (**positives** and **negatives**) as **negatives (FP=0)**
- $\boldsymbol{Precision = 1}$



Precision

What is the **proportion** of true positives among all instances **classified** (correctly and incorrectly) as positives?
How precise is the positive classification?

"From all patients classified as cancer, how many (**proportion**) actually had cancer?"

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

`sklearn.metrics.precision_score`

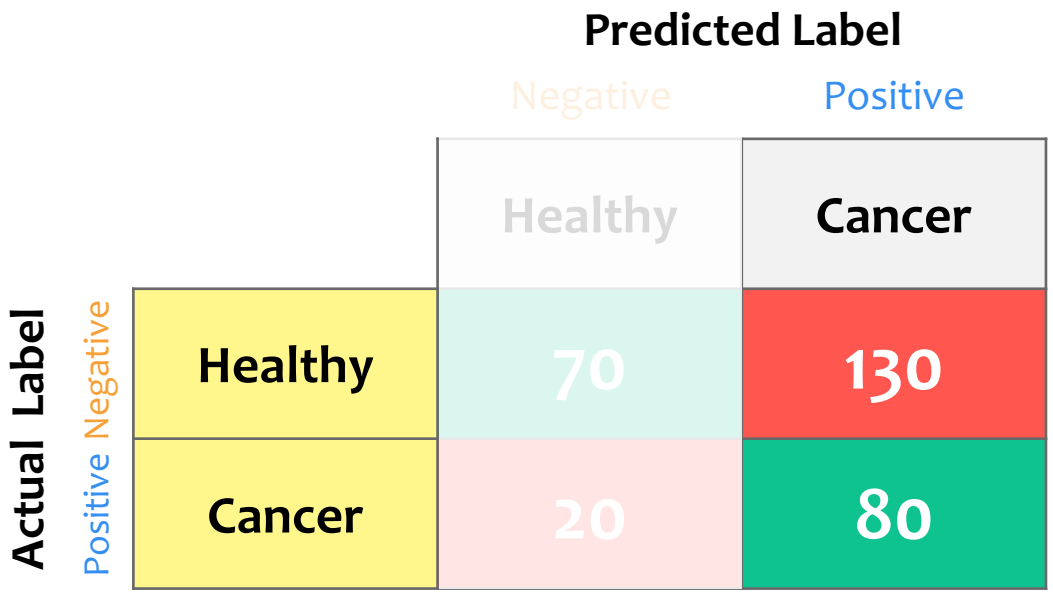
```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred)
```

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred, pos_label='Cancer')
```

By default, the positive class must have **value/label +1**.

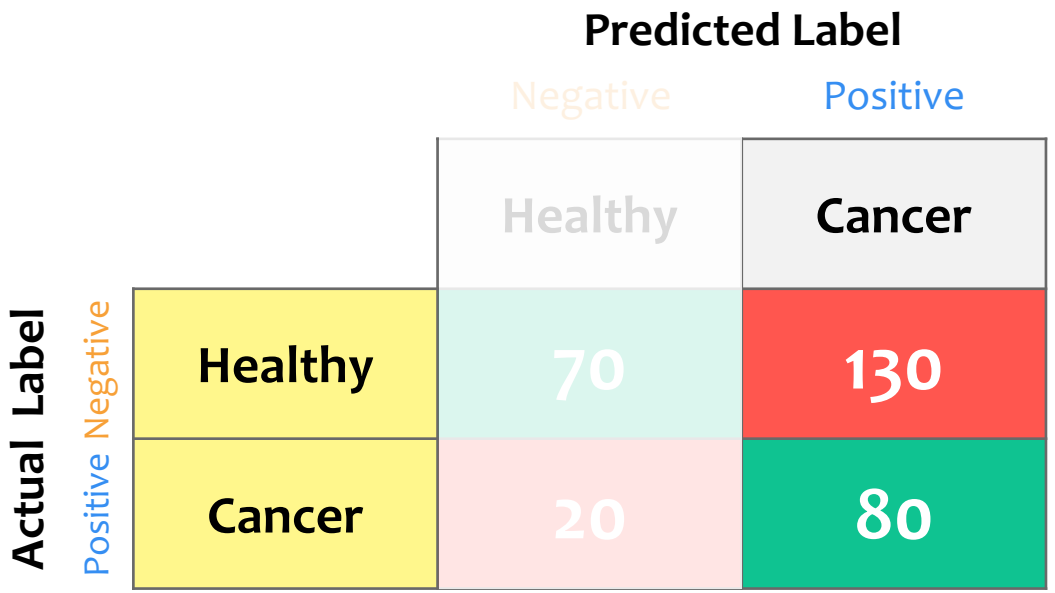
We can **indicate** which is the **value/label** of the positive class.

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



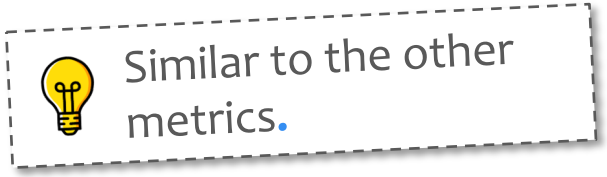
$$Precision = \frac{TP}{TP + FP} = \frac{80}{80 + 130} = 0.38$$

Only **38%** of all people classified **as having cancer** actually had **cancer.**



Precision for Multiple Classes

$$Precision_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FP_{Cat}} =$$



Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Precision for Multiple Classes

$$Precision_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FP_{Cat}} = \frac{10}{10 + 6} = 0.625$$

50% of pets classified as **Cat** are in fact **cats**.

Similar to the other metrics.

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Precision for Multiple Classes

$$Precision_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FP_{Cat}} = \frac{10}{10 + 6} = 0.625$$

50% of pets classified as **Cat** are in fact **cats**.

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred, average=None, labels=['Cat', 'Dog', 'Wolf'])

array([0.625     , 0.71428571, 0.46875   ])
```

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Precision for Multiple Classes

$$Precision_{Cat} = \frac{TP_{Cat}}{TP_{Cat} + FP_{Cat}} = \frac{10}{10 + 6} = 0.625$$

50% of pets classified as **Cat** are in fact **cats**.

```
from sklearn.metrics import precision_score
precision_s                                    olf'])

array([0.625
```

We can return a **single "global"** precision by following the same strategies we have seen before!

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Single Precision for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Precision_{macro} = \frac{\sum_{j=0}^{c-1} Precision_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred, average='macro',
                labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6026785714285715
```

Single Precision for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$Precision_{macro} = \frac{\sum_{j=0}^{c-1} Precision_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred, average='macro',
                labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6026785714285715
```

micro

Calculate **metrics globally** by counting the total true positives, false negatives and false positives.

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$Precision_{micro} = \frac{\sum_{j=0}^{c-1} TP_j}{m}$$

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred, average='micro',
                labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6111111111111112
```

Single Precision for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

$$Precision_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * Precision_j}{m}$$

weight (w_j) for class j

```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred, average='weighted',
                labels=['Cat', 'Dog', 'Wolf'])
```

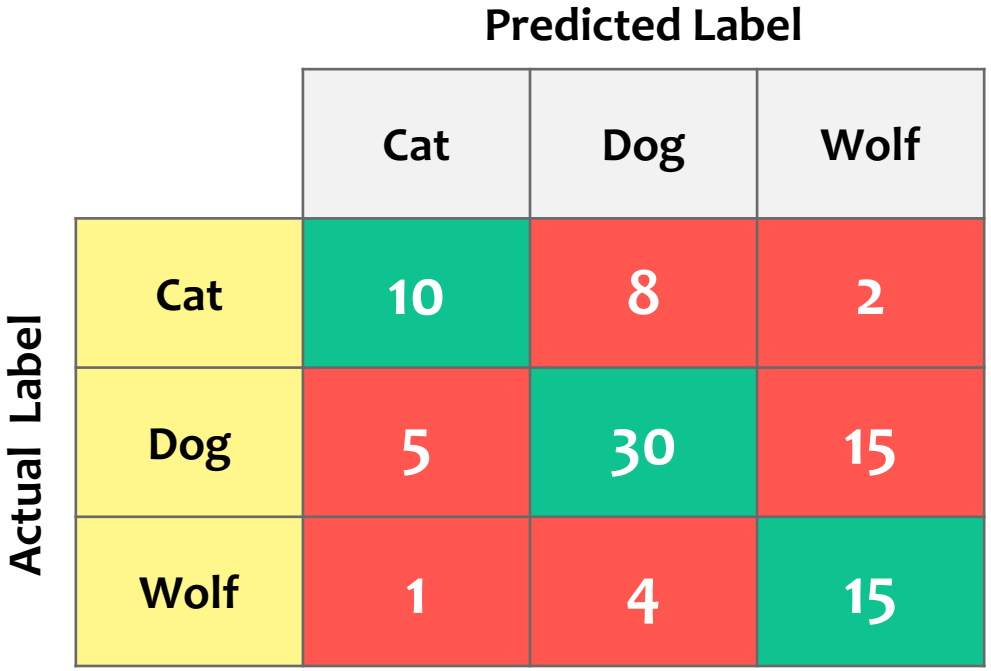
```
0.6398809523809524
```

It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

Exercise: Compute…

$Precision_{Dog}$

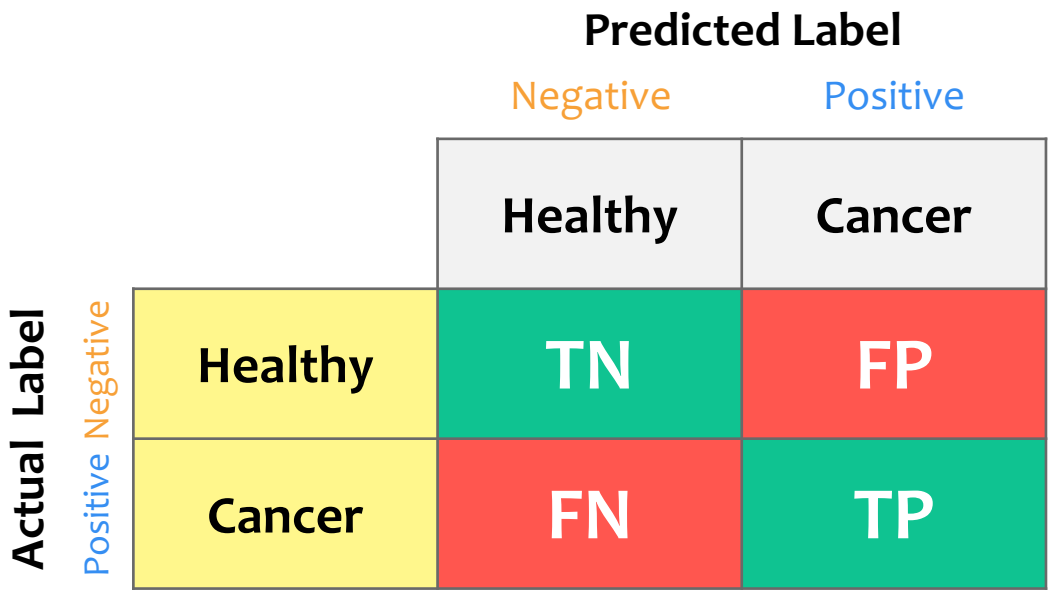
$Precision_{Wolf}$



Accuracy

What was the (overall) **classification hit rate**?

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



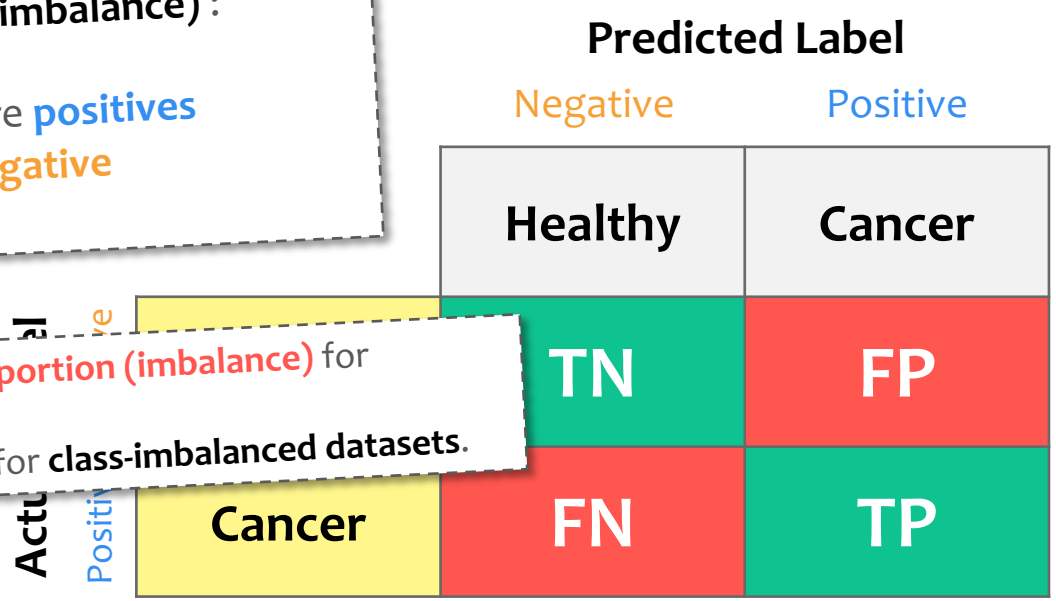
Accuracy

What was the (overall) **classification hit rate**?

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

Issue with **biased dataset (imbalance)** :
Easy to **cheat** the score:
- Consider that only **1%** are **positives**
- Guess everything as **negative**
- Achieve **99% accuracy**

It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.



Accuracy

What was the (overall) **classification hit rate**?

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

The same idea is applied to **extend it** to **multiple classes**.

Issue with **biased dataset (imbalance)** :
Easy to **cheat** the score:
- Consider that only **1%** are **positives**
- Guess everything as **negative**
- Achieve **99% accuracy**

It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

		Predicted Label: Negative	Predicted Label: Positive
		Healthy	**Cancer**
Actual: Negative	**Healthy**	**TN**	**FP**
Actual: Positive	**Cancer**	**FN**	**TP**

Accuracy

What was the (overall) **classification hit rate**?

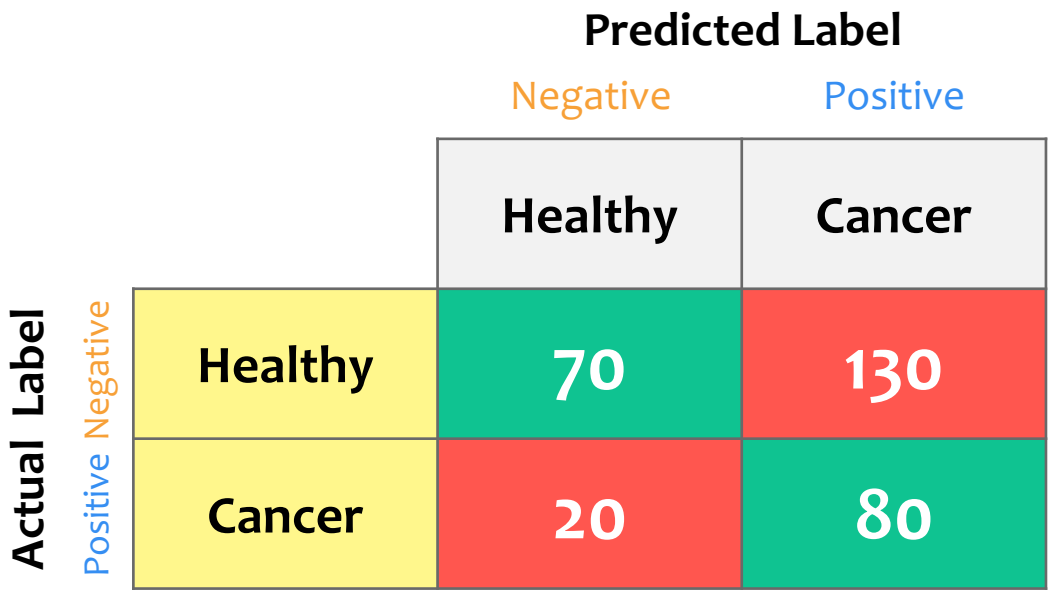
$$Accuracy = \frac{TP + TN}{TN + FN + FP + TP}$$

sklearn.metrics.accuracy_score

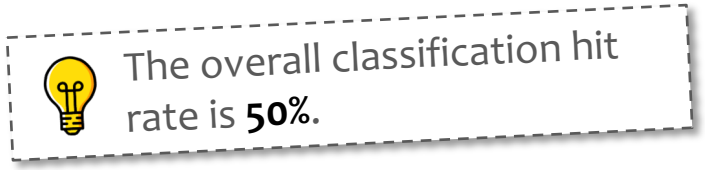
```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

We **do not** need to specify which is the **positive** (or **negative**) class, since **accuracy** takes into account **all metrics** from the **confusion matrix**.

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



$$Accuracy = \frac{70 + 80}{70 + 20 + 130 + 80} = \frac{150}{300} = 0.5$$



Actual Label (Positive / Negative)	Predicted Label: Negative — Healthy	Predicted Label: Positive — Cancer
Healthy	70	130
Cancer	20	80

Accuracy for Multiple Classes

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$Accuracy = \frac{\sum_{j=0}^{c-1} TP_j}{m} = \frac{TP_{Cat} + TP_{Dog} + TP_{Wolf}}{m}$$

Sum of **all values** from the **confusion matrix**

Predicted Label

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Accuracy for Multiple Classes

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$Accuracy = \frac{\sum_{j=0}^{c-1} TP_j}{m} = \frac{TP_{Cat} + TP_{Dog} + TP_{Wolf}}{m}$$

Sum of **all values** from the **confusion matrix**

$$Accuracy = \frac{10 + 30 + 15}{90} = 0.6111$$

Predicted Label

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Accuracy for Multiple Classes

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$Accuracy = \frac{\sum_{j=0}^{c-1} TP_j}{m} = \frac{TP_{Cat} + TP_{Dog} + TP_{Wolf}}{m}$$

Sum of **all values** from the **confusion matrix**

$$Accuracy = \frac{10 + 30 + 15}{90} = 0.6111$$

Predicted Label

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
		30	15
Wolf	1	4	15

⚠ It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

Balanced Accuracy

Adapted version of **accuracy** that deals with **class-imbalanced datasets**.
Each class **contributes/weights equally** to the final metric.

It is defined as the **average of sensitivity/recall** obtained on each class.

It is equal to $\boldsymbol{Recall_{macro}}$.

Balanced Accuracy

Adapted version of **accuracy** that deals with **class-imbalanced datasets**.
Each class **contributes/weights equally** to the final metric.

It is defined as the **average of sensitivity/recall** obtained on each class.

It is equal to $\boldsymbol{Recall_{macro}}$.

Binary case

"sensitivity/recall" of the **negative class**

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2}$$

Two classes.

Balanced Accuracy

Adapted version of **accuracy** that deals with **class-imbalanced datasets**.
Each class **contributes/weights equally** to the final metric.

It is defined as the **average of sensitivity/recall** obtained on each class.

It is equal to $Recall_{macro}$.

Binary case

"sensitivity/recall" of the **negative class**

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2}$$

Two classes.

Multiclass

For **c** classes, enumerated from **0** to **c-1**

$$BalancedAccuracy = \frac{\sum_{j=0}^{c-1} Recall_j}{c}$$

Balanced Accuracy

Adapted version of **accuracy** that deals with **class-imbalanced datasets**.
Each class **contributes/weights equally** to the final metric.

It is defined as the **average of sensitivity/recall** obtained on each class.

It is equal to $Recall_{macro}$.

Binary case

"sensitivity/recall" of the **negative class**

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2}$$

Two classes.

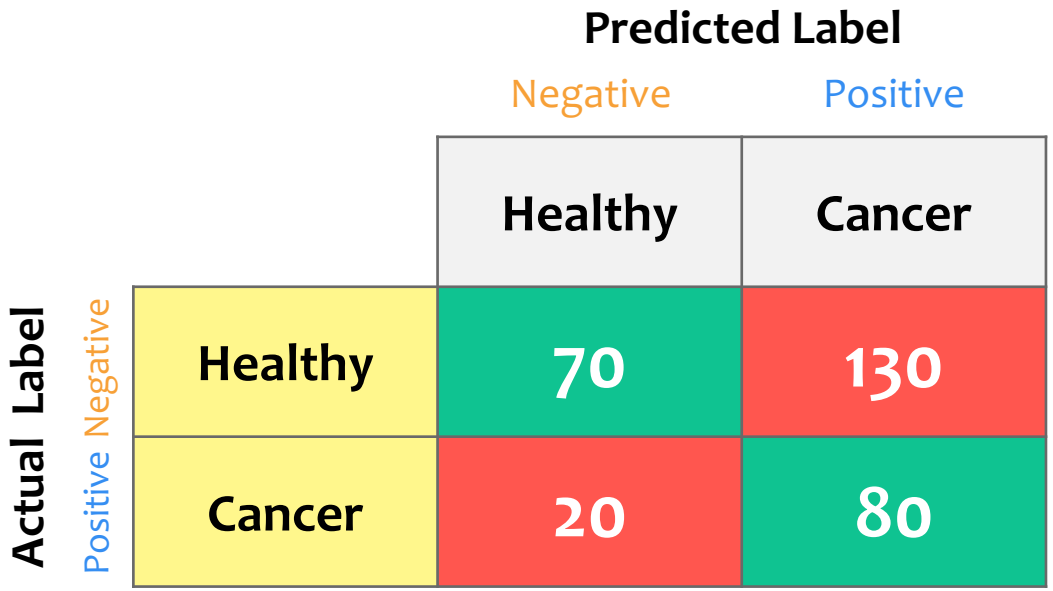
```
from sklearn.metrics import balanced_accuracy_score
balanced_accuracy_score(y_true, y_pred, adjusted=False)
```

Multiclass

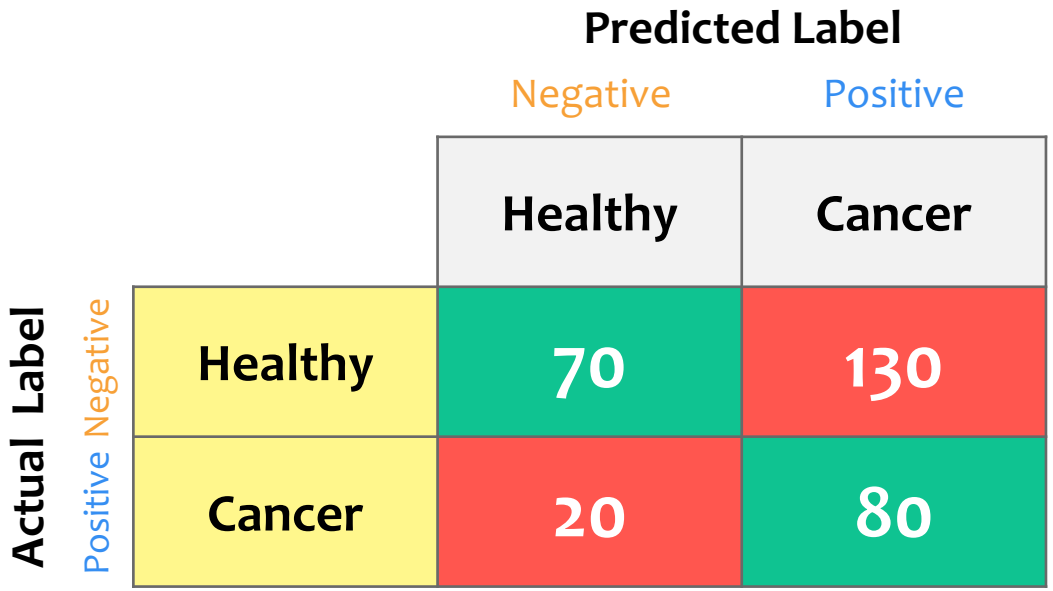
For **c** classes, enumerated from **0** to **c-1**

$$BalancedAccuracy = \frac{\sum_{j=0}^{c-1} Recall_j}{c}$$

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2}$$



$$BalancedAccuracy = \frac{0.8 + 0.35}{2} = 0.575$$



Balanced Accuracy for Multiple Classes

For **c** classes, enumerated from **0** to **c-1**

$$BalancedAccuracy = \frac{\sum_{j=0}^{c-1} Recall_j}{c} = \frac{Recall_{Cat} + Recall_{Dog} + Recall_{Wolf}}{3}$$

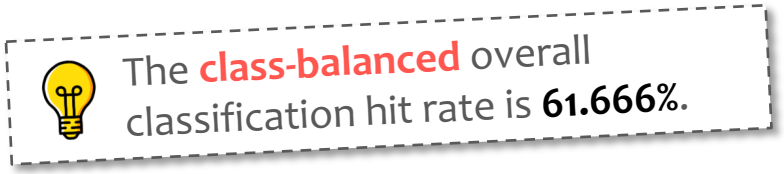
Predicted Label

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Balanced Accuracy for Multiple Classes

For **c** classes, enumerated from **0** to **c-1**

$$BalancedAccuracy = \frac{\sum_{j=0}^{c-1} Recall_j}{c} = \frac{0.5 + 0.6 + 0.75}{3} = 0.616666$$



Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Class Balanced Accuracy (CBA)

A version proposed by *Mosley et al.*, "A balanced approach to the multi-class imbalance problem", IJCV 2010.

- Section 3.2, Equations 3.1 – 3.4

For **c** classes, enumerated from **0** to **c-1**

$$CBA = \frac{\sum_{j=0}^{c-1} \min(Recall_j, Precision_j)}{c}$$

Class Balanced Accuracy (CBA)

A version proposed by *Mosley et al.*, "A balanced approach to the multi-class imbalance problem", IJCV 2010.

- Section 3.2, Equations 3.1 – 3.4

For **c** classes, enumerated from **0** to **c-1**

$$CBA = \frac{\sum_{j=0}^{c-1} \min(Recall_j, Precision_j)}{c}$$

It is a **more rigid metric** than **Balanced Accuracy**, penalizing **class imbalance** and classification errors **more strongly**.

Class Balanced Accuracy (CBA)

A version proposed by *Mosley et al.*, "A balanced approach to the multi-class imbalance problem", IJCV 2010.

- Section 3.2, Equations 3.1 – 3.4

For **c** classes, enumerated from **0** to **c-1**

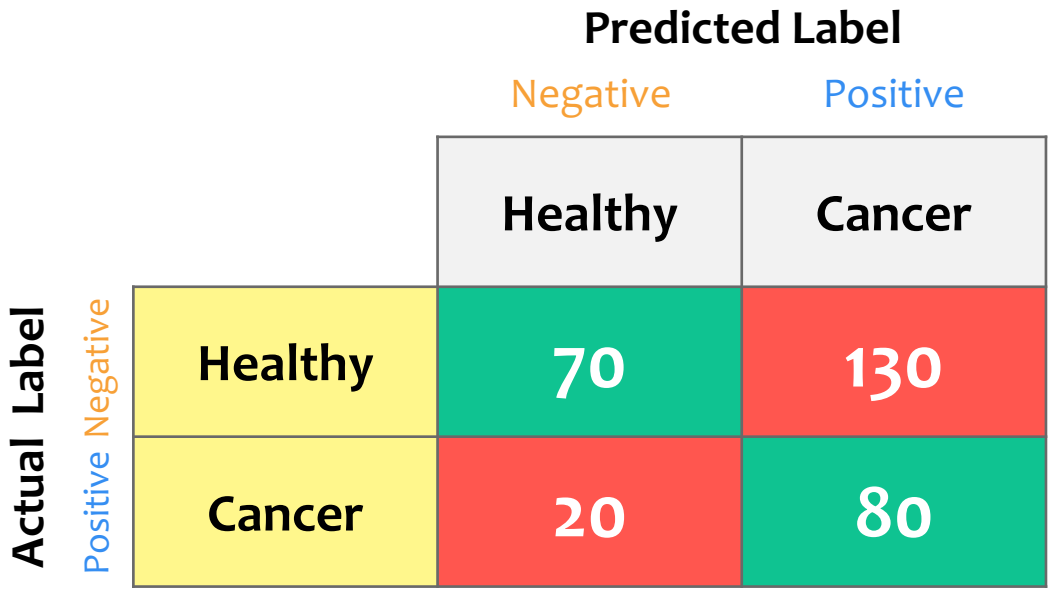
$$CBA = \frac{\sum_{j=0}^{c-1} \min(Recall_j, Precision_j)}{c}$$

It is a **more rigid metric** than **Balanced Accuracy**, penalizing **class imbalance** and classification errors **more strongly**.

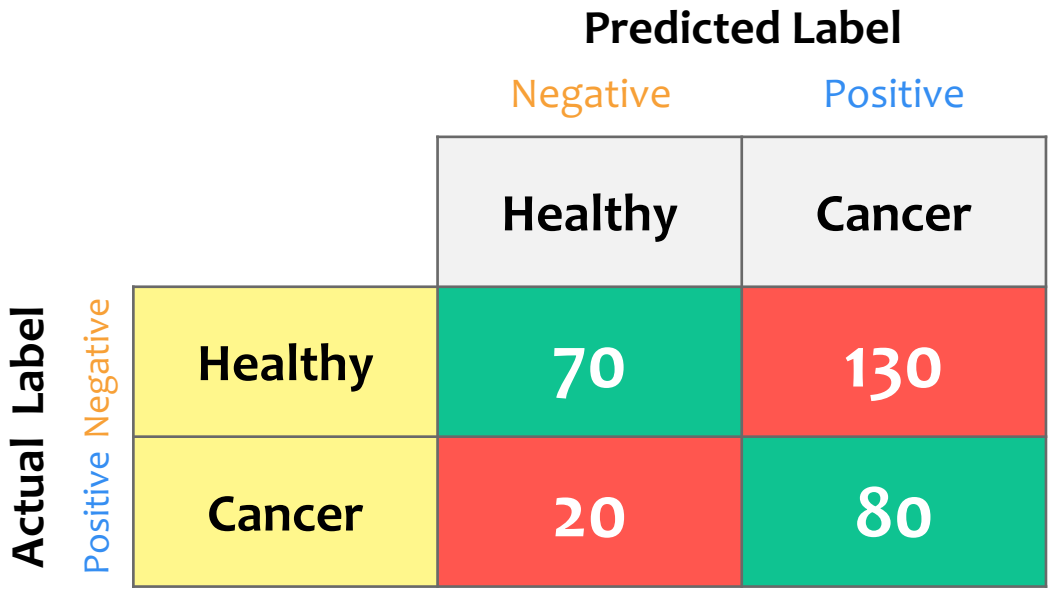
```python
def class_balanced_accuracy(y_true, y_pred):
    labels = list(set(y_true))

    recall = recall_score(y_true, y_pred, average=None, labels=labels)
    precision = precision_score(y_true, y_pred, average=None, labels=labels)

    cba = np.minimum(recall, precision).mean()
    return cba

class_balanced_accuracy(y_true, y_pred)
```

For **c** classes, enumerated from **0** to **c-1**

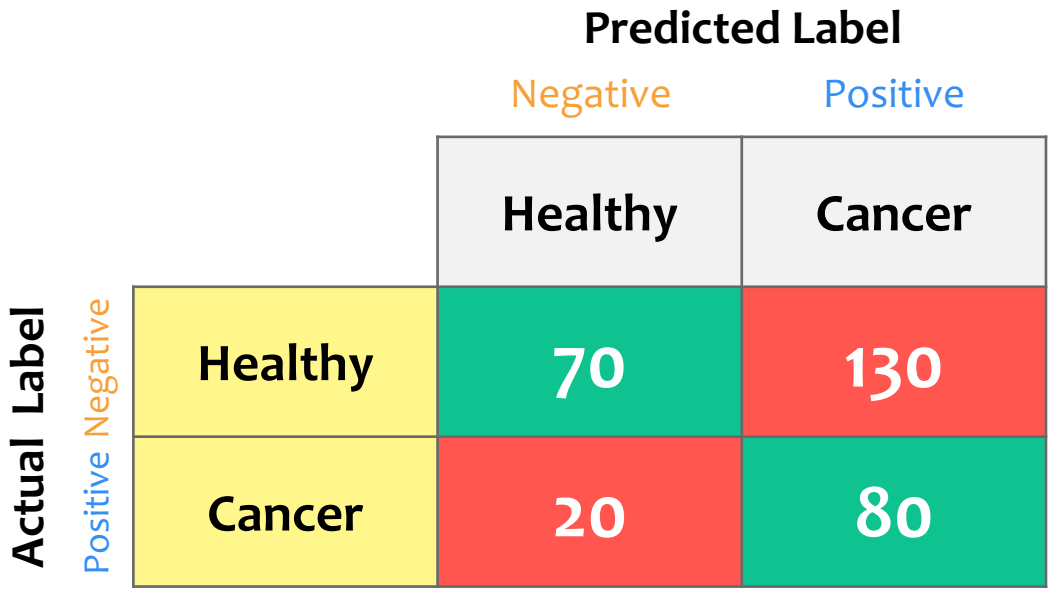
$$CBA = \frac{\sum_{j=0}^{c-1} \min(Recall_j, Precision_j)}{c}$$



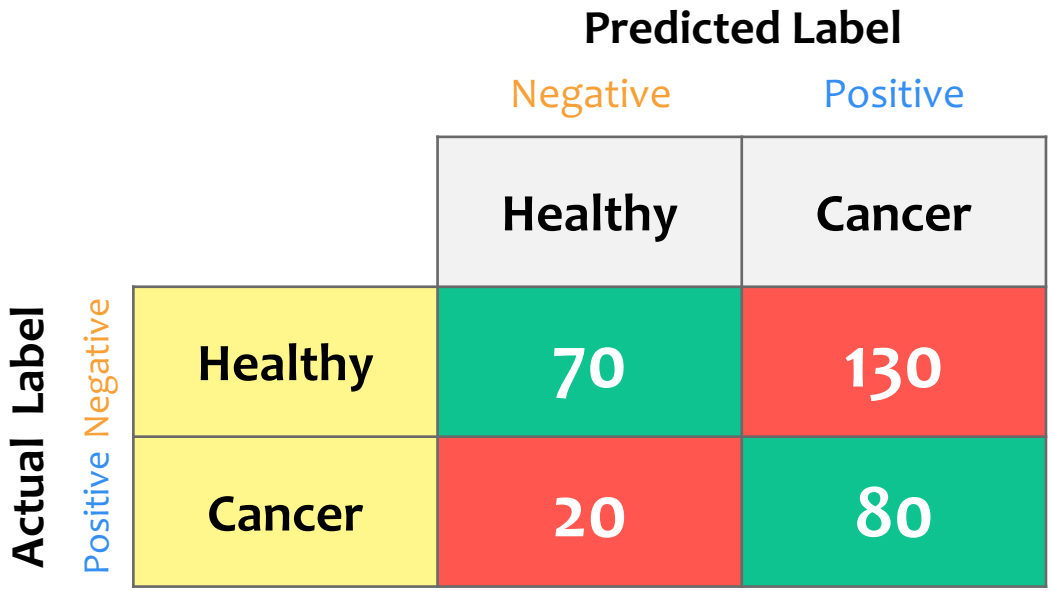
$$CBA = \frac{\min(Recall_{Healthy}, Precision_{Healthy}) + \min(Recall_{Cancer}, Precision_{Cancer})}{2}$$



$$CBA = \frac{\min(0.35, 0.7778) + \min(0.8, 0.381)}{2}$$



$$CBA = \frac{\min(0.35, 0.7778) + \min(0.8, 0.381)}{2} = \frac{0.35 + 0.381}{2} = 0.3655$$

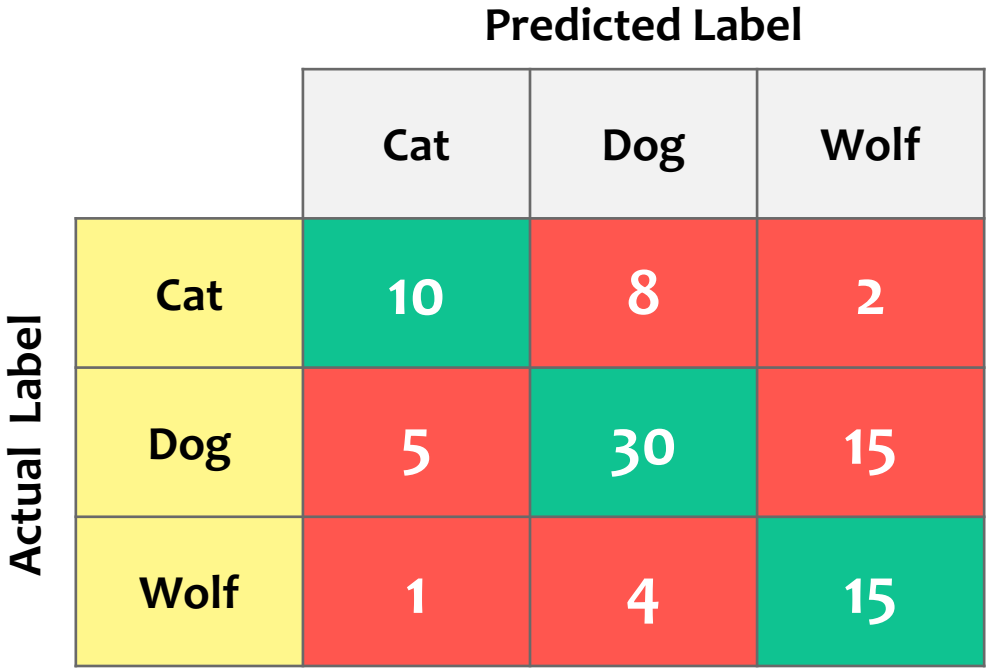


Exercise: Compute…

For **c** classes, enumerated from **0** to **c-1**

$$CBA = \frac{\sum_{j=0}^{c-1} \min(Recall_j, Precision_j)}{c}$$

$$CBA = \frac{\min(Precision_{Cat}, Precision_{Cat}) + \min(Precision_{Dog}, Precision_{Dog}) + \min(Precision_{Wolg}\ Precision_{Wolf})}{3}$$



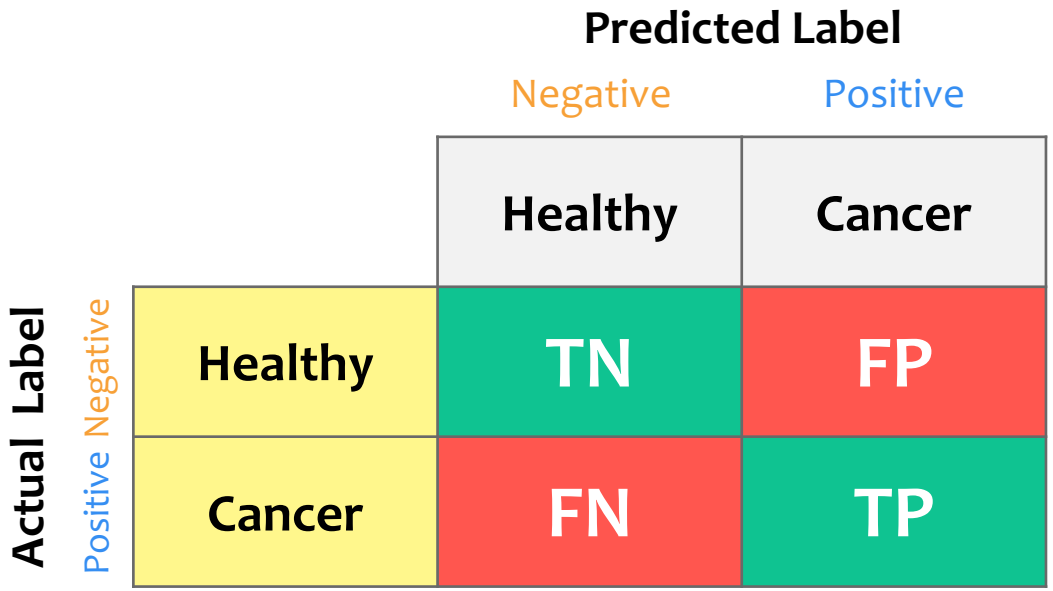
F1-Score

How **good** and **complete** are the predictions?
Middle ground between **precision** and **recall**.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

It solves the **accuracy cheating** caused by **class imbalance**.

The same idea is applied to **extend it** to **multiple classes**.



F1-Score

How **good** and **complete** are the predictions?
Middle ground between **precision** and **recall**.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

`sklearn.metrics.f1_score`

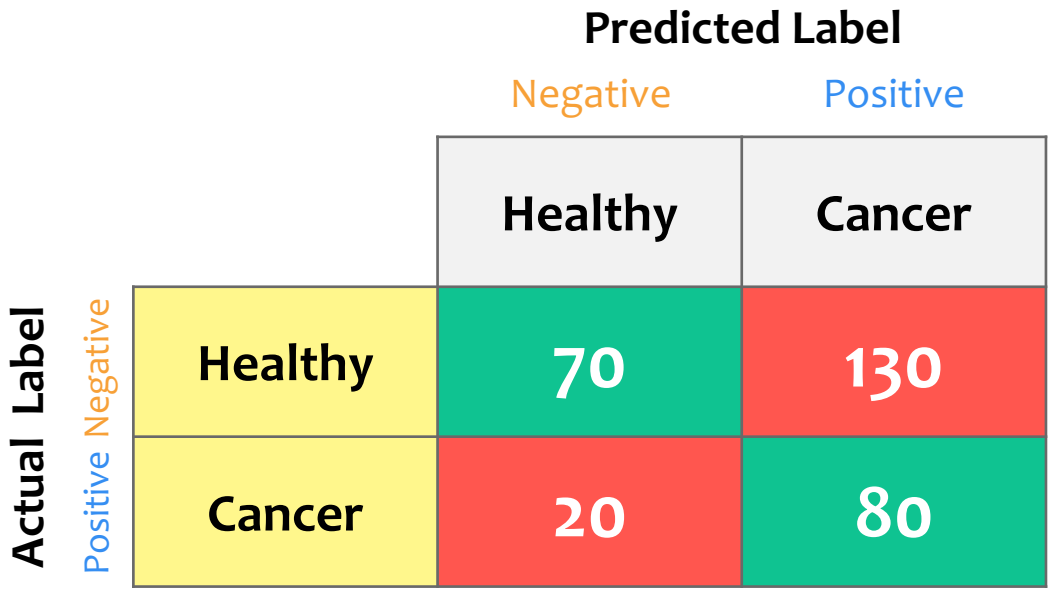
```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred)
```

By default, the **positive class** must have **value/label +1**.

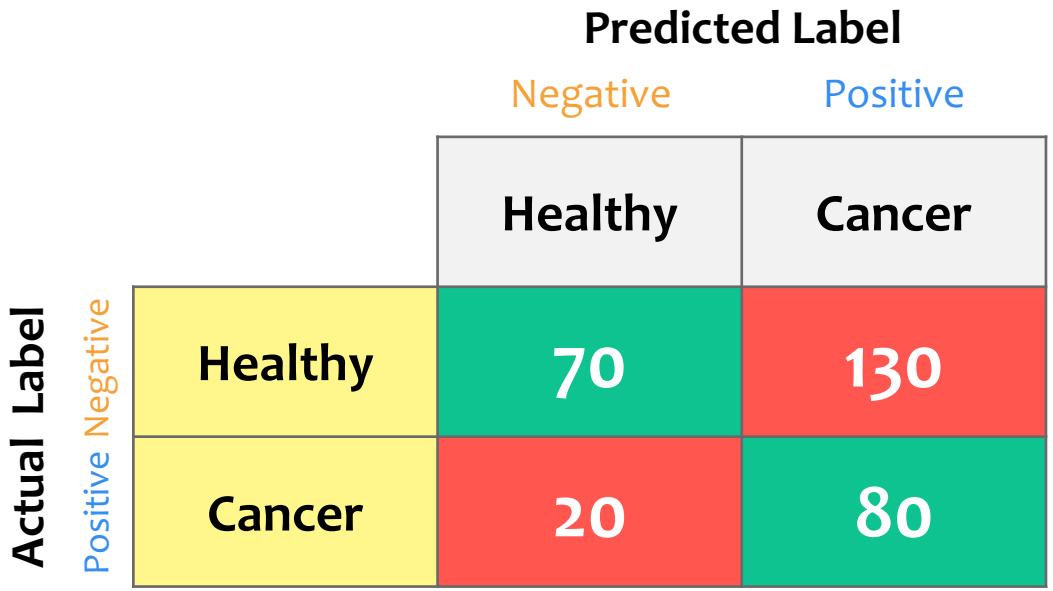
```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, pos_label='Cancer')
```

We can **indicate** which is the **value/label** of the **positive class.**

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$



$$F1 = 2 * \frac{Precision\ * Recall}{Precision + Recall} = 2 * \frac{0.381 * 0.8}{0.381 + 0.8} = 0.5161$$



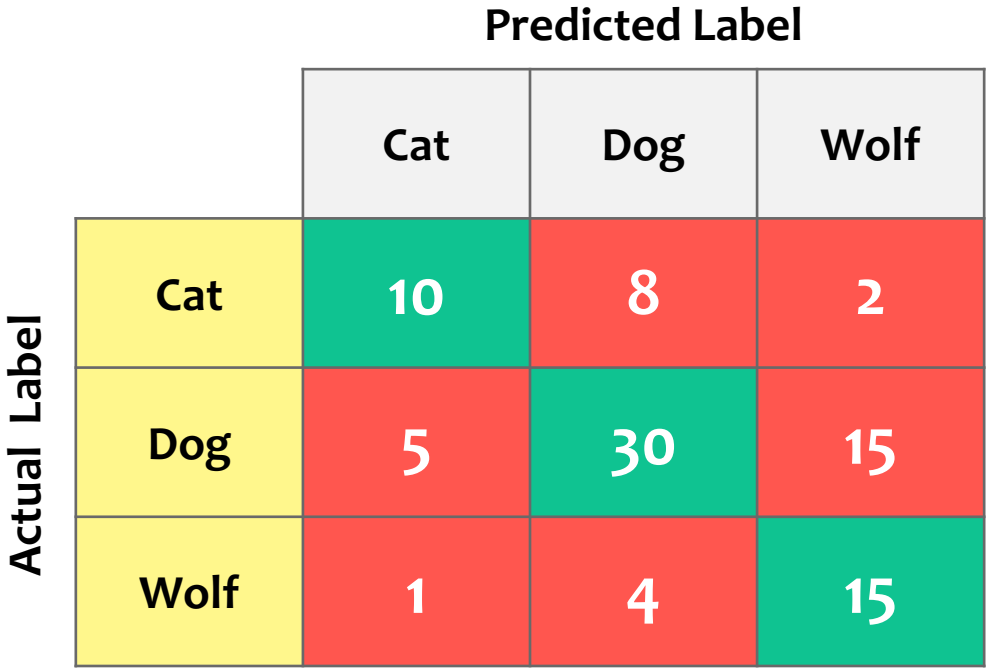
F1-Score for Multiple Classes

$$F1_{Cat} = 2 * \frac{Precision_{Cat} * Recall_{Cat}}{Precision_{Cat} + Recall_{Cat}}$$

Actual Label \ Predicted Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

F1-Score for Multiple Classes

$$F1_{Cat} = 2 * \frac{Precision_{Cat} * Recall_{Cat}}{Precision_{Cat} + Recall_{Cat}} = 2 * \frac{0.625 * 0.5}{0.625 + 0.5} = 0.555$$



F1-Score for Multiple Classes

$$F1_{Cat} = 2 * \frac{Precision_{Cat} * Recall_{Cat}}{Precision_{Cat} + Recall_{Cat}} = 2 * \frac{0.625 * 0.5}{0.625 + 0.5} = 0.555$$

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average=None, labels=['Cat', 'Dog', 'Wolf'])

array([0.55555556, 0.65217391, 0.57692308])
```

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

F1-Score for Multiple Classes

$$F1_{Cat} = 2 * \frac{Precision_{Cat} * Recall_{Cat}}{Precision_{Cat} + Recall_{Cat}} = 2 * \frac{0.625 * 0.5}{0.625 + 0.5} = 0.555$$

```
from sklearn.metrics import f1
f1_sco                                                    f']

array(                                  0.57692308])
```

We can return a **single "global" F1-Score** by following the same strategies we have seen before!

Actual Label	Cat	Dog	Wolf
Cat	10	8	2
Dog	5	30	15
Wolf	1	4	15

Single F1-Score for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$F1_{macro} = \frac{\sum_{j=0}^{c-1} F1_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average='macro',
         labels=['Cat', 'Dog', 'Wolf'])
```

```
0.5948841818407036
```

Single F1-Score for Multiple Classes

macro

Calculate **metrics** for **each label** and **average** them.

For **c** classes, enumerated from **0** to **c-1**

$$F1_{macro} = \frac{\sum_{j=0}^{c-1} F1_j}{c}$$

Each class **contributes/weights equally** to the final metric since it **does not** take **class proportion (imbalance)** into account. Thus, it is appropriated for **class-imbalanced datasets**.

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average='macro',
         labels=['Cat', 'Dog', 'Wolf'])
```

```
0.5948841818407036
```

micro

Calculate **metrics globally** by counting the total true positives, false negatives and false positives.

For **c** classes, enumerated from **0** to **c-1,** and **m** samples

$$F1_{micro} = \frac{\sum_{j=0}^{c-1} TP_j}{m}$$

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average='micro',
         labels=['Cat', 'Dog', 'Wolf'])
```

```
0.6111111111111112
```

Single F1-Score for Multiple Classes

weighted

Calculate metrics for each class, and find their **average weighted** by the **number of true samples of each class**.

Let c be the number of classes, enumerated from **0** to **c-1**,
m the number samples, and m_j is the number of true samples of class j.

$$F1_{weighted} = \frac{\sum_{j=0}^{c-1} m_j * F1_j}{m}$$

weight (w_j) for class j

```
from sklearn.metrics import f1_score
f1_score(y_true, y_pred, average='weighted',
         labels=['Cat', 'Dog', 'Wolf'])

0.6139807589082951
```

It **take into account class proportion (imbalance)** for computation.
Thus, it is **NOT appropriated** for **class-imbalanced datasets**.

Putting it all together

Binary problem

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred, labels=['Healthy', 'Cancer'], digits=4))
```

```
              precision    recall  f1-score   support

     Healthy     0.7778    0.3500    0.4828       200
      Cancer     0.3810    0.8000    0.5161       100

    accuracy                         0.5000       300
   macro avg     0.5794    0.5750    0.4994       300
weighted avg     0.6455    0.5000    0.4939       300
```

Putting it all together

Binary problem

Provides a **report** with some **classification metrics**.

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred, labels=['Healthy', 'Cancer'], digits=4))
```

```
              precision    recall  f1-score   support

     Healthy     0.7778    0.3500    0.4828       200
      Cancer     0.3810    0.8000    0.5161       100

    accuracy                         0.5000       300
   macro avg     0.5794    0.5750    0.4994       300
weighted avg     0.6455    0.5000    0.4939       300
```

Putting it all together

Binary problem

Optional: it is just to keep a **given desired order** in the report.

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred, labels=['Healthy', 'Cancer'], digits=4))
```

```
              precision    recall  f1-score   support

     Healthy     0.7778    0.3500    0.4828       200
      Cancer     0.3810    0.8000    0.5161       100

    accuracy                         0.5000       300
   macro avg     0.5794    0.5750    0.4994       300
weighted avg     0.6455    0.5000    0.4939       300
```

Putting it all together

Binary problem

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred, labels=['Healthy', 'Cancer'], digits=4))
```

```
              precision    recall  f1-score   support

     Healthy     0.7778    0.3500    0.4828       200
      Cancer     0.3810    0.8000    0.5161       100

    accuracy                         0.5000       300
   macro avg     0.5794    0.5750    0.4994       300
weighted avg     0.6455    0.5000    0.4939       300
```

Number of samples of each class.

Putting it all together

Binary problem

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred, labels=['Healthy', 'Cancer'], digits=4))
```

	precision	recall	f1-score	support
Healthy	0.7778	0.3500	0.4828	200
Cancer	0.3810	0.8000	0.5161	100
accuracy			0.5000	300
macro avg	0.5794	0.5750	0.4994	300
weighted avg	0.6455	0.5000	0.4939	300

Balanced Accuracy

Putting it all together

Multiclass problem

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred, labels=['Cat', 'Dog', 'Wolf'], digits=4))
```

```
              precision    recall  f1-score   support

         Cat     0.6250    0.5000    0.5556        20
         Dog     0.7143    0.6000    0.6522        50
        Wolf     0.4688    0.7500    0.5769        20

    accuracy                         0.6111        90
   macro avg     0.6027    0.6167    0.5949        90
weighted avg     0.6399    0.6111    0.6140        90
```

Accuracy vs F1-Score

Accuracy

F1-Score

Accuracy vs F1-Score

Accuracy

1. Used when the **True Positives** and **True Negatives** are **more important** (overall hit rate);

F1-Score

1. Used when the **False Negatives** and **False Positives** are **crucial** (classification errors);

Accuracy vs F1-Score

Accuracy

1. Used when the **True Positives** and **True Negatives** are **more important** (overall hit rate);
2. Suitable for **(roughly) class-balanced** datasets;

F1-Score

1. Used when the **False Negatives** and **False Positives** are **crucial** (classification errors);
2. Works fine for **class-imbalanced** datasets;

Accuracy vs F1-Score

Accuracy

1. Used when the **True Positives** and **True Negatives** are **more important** (overall hit rate);
2. Suitable for **(roughly) class-balanced** datasets;

F1-Score

1. Used when the **False Negatives** and **False Positives** are **crucial** (classification errors);
2. Works fine for **class-imbalanced** datasets;
3. One of the **most used** classification metric;

Accuracy vs F1-Score

Accuracy

1. Used when the **True Positives** and **True Negatives** are **more important** (overall hit rate);
2. Suitable for **(roughly) class-balanced** datasets;
3. Its **balanced version** (**Balanced Accuracy - BA**) deals with **class-imbalance** while keeping the goal (1);

F1-Score

1. Used when the **False Negatives** and **False Positives** are **crucial** (classification errors);
2. Works fine for **class-imbalanced** datasets;
3. One of the **most used** classification metric;

Accuracy vs F1-Score

Accuracy

1. Used when the **True Positives** and **True Negatives** are **more important** (overall hit rate);
2. Suitable for **(roughly) class-balanced** datasets;
3. Its **balanced version** (**Balanced Accuracy - BA**) deals with **class-imbalance** while keeping the goal (1);
4. The **balanced version** by *Mosley et al.* (**Class Balanced Accuracy - CBA**) is **more rigid** than BA when **strongly penalizing** class imbalance and classification errors;

F1-Score

1. Used when the **False Negatives** and **False Positives** are **crucial** (classification errors);
2. Works fine for **class-imbalanced** datasets;
3. One of the **most used** classification metric;

Accuracy vs F1-Score

Accuracy

1. Used when the **True Positives** and **True Negatives** are **more important** (overall hit rate);
2. Suitable for **(roughly) class-balanced** datasets;
3. Its **balanced version** (**Balanced Accuracy - BA**) deals with **class-imbalance** while keeping the goal (1);
4. The **balanced version** by *Mosley et al.* (**Class Balanced Accuracy - CBA**) is **more rigid** than BA when **strongly penalizing** class imbalance and classification errors;
5. **CBA** is more interesting for **rigorous experiments** where **True Positives** and **True negatives** are **more important** ;

F1-Score

1. Used when the **False Negatives** and **False Positives** are **crucial** (classification errors);
2. Works fine for **class-imbalanced** datasets;
3. One of the **most used** classification metric;

Putting it all together

Binary problem

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Classifier 1 (KNN)	0.8	0.35	0.38	0.5	0.575	0.365	0.516
Classifier 2 (Log. Regression)	0.5	0.4	0.29	0.433	0.45	0.347	0.37

Putting it all together

Binary problem

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Classifier 1 (KNN)	0.8	0.35	0.38	0.5	**0.575**	**0.365**	**0.516**
Classifier 2 (Log. Regression)	0.5	0.4	0.29	0.433	0.45	0.347	0.37

We should pick **KNN** since it is **consistently better** than Log. Regression for Balanced and Class Balanced Accuracy, and F1-Score

Putting it all together

Multiclass

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Cat	0.5	0.9142	0.625	--	--		0.555
Dog	0.6	0.7	0.7143	--	--		0.652
Wolf	0.75	0.757	0.4688	--	--		0.576
Macro	0.6167	0.79	0.6	0.6111	0.6167	0.522	0.594

Putting it all together

Multiclass

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Cat	0.5	0.9142	0.625	--	--		0.555
Dog	0.6	0.7	0.7143	--	--		0.652
Wolf	0.75	0.757	0.4688	--	--		0.576
Macro	0.6167	0.79	0.6	0.6111	0.6167	0.522	0.594

The classifier is **not effective** when presenting **poor evaluation scores**.

Putting it all together

Multiclass

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Cat	0.5	0.9142	0.625	--	--		0.555
Dog	0.6	0.7	0.7143	--	--		0.652
Wolf	0.75	0.757	0.4688	--	--		0.576
Macro	0.6167	0.79	0.6	0.6111	0.6167	0.522	0.594

It *considerably fails* when classifying **Cats** and **Dogs:** **low sensitivity** results, for example, in many cats not classified as cats.

Putting it all together

Multiclass

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Cat	0.5	0.9142	0.625	--	--		0.555
Dog	0.6	0.7	0.7143	--	--		0.652
Wolf	0.75	0.757	0.4688	--	--		0.576
Macro	0.6167	0.79	0.6	0.6111	0.6167	0.522	0.594

The **high specificity** for Cats indicates that the classifier is correctly classifying non-cats as non-cats.

However, this does not necessarily mean that the classifier is not making mistakes.
For example, a dog (non-cat) may have been classified as a wolf (non-cat) and vice versa.

Putting it all together

Multiclass

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Cat	0.5	0.9142	0.625	--	--		0.555
Dog	0.6	0.7	0.7143	--	--		0.652
Wolf	0.75	0.757	0.4688	--	--		0.576
Macro	0.6167	0.79	0.6	0.6111	0.6167	0.522	0.594

Although most **Wolfs** in the dataset were **correctly classified as Wolfs**…

Putting it all together

Multiclass

	Sensitivity/ Recall	Specificity	Precision	Accuracy	Balanced Accuracy	Class Balanced Accuracy	F1-Score
Cat	0.5	0.9142	0.625	--	--		0.555
Dog	0.6	0.7	0.7143	--	--		0.652
Wolf	0.75	0.757	0.4688	--	--		0.576
Macro	0.6167	0.79	0.6	0.6111	0.6167	0.522	0.594

Although most **Wolfs** in the dataset were **correctly classified as Wolfs,** the classify tends to classify **most pets as Wolfs** (see the **low precision**).

Other Metrics

- ROC curve and AUC
- Fbeta-Score
- G-Mean
- Jaccard score
- Matthews correlation coefficient
- Cohen’s kappa
- Brier score

Aprendizado de Máquina e Reconhecimento de Padrões 2021.2

Evaluation Metrics for Classification

Prof. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

CC BY NC SA

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas