Introduction to Machine Learning

Ass. Prof. Sara El-Metwally
Computer Science Department,
Faculty of Computers and Information,
Mansoura University, Egypt.

sarah_almetwally4@mans.edu.eg

Agenda

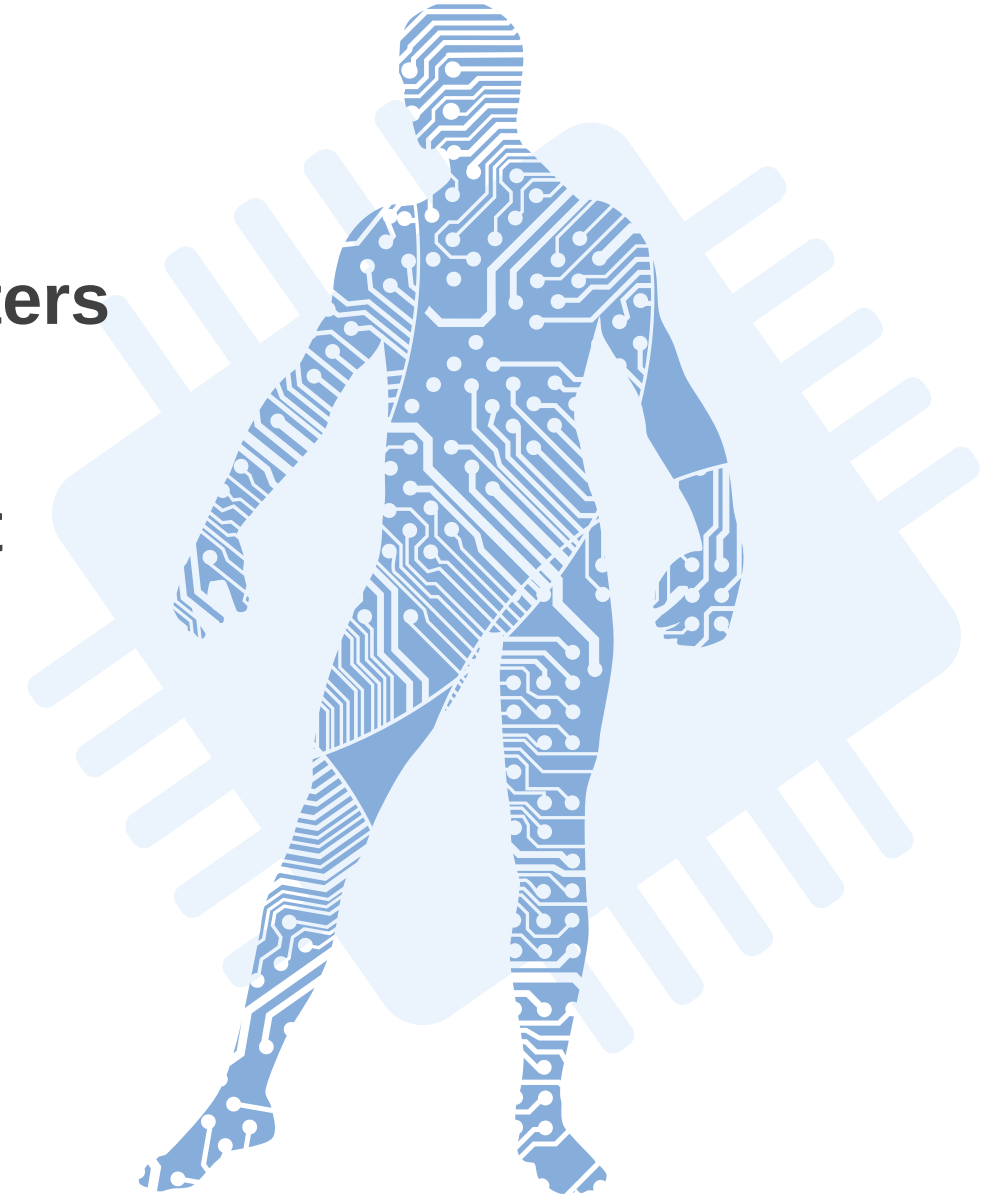
01 Parameters vs. Hyper parameters

02 Stochastic gradient descent

03 Mini-batch stochastic gradient descent

04 Example & Code

05 Confusion Matrix



Parameters vs. Hyperparameters

- Parameters are values that the model calculates during training.
  - weights
  - Biases

- Hyperparameters are variables/values that control different aspects of training:
  - Learning rate
  - Batch size
  - Epochs

Learning rate

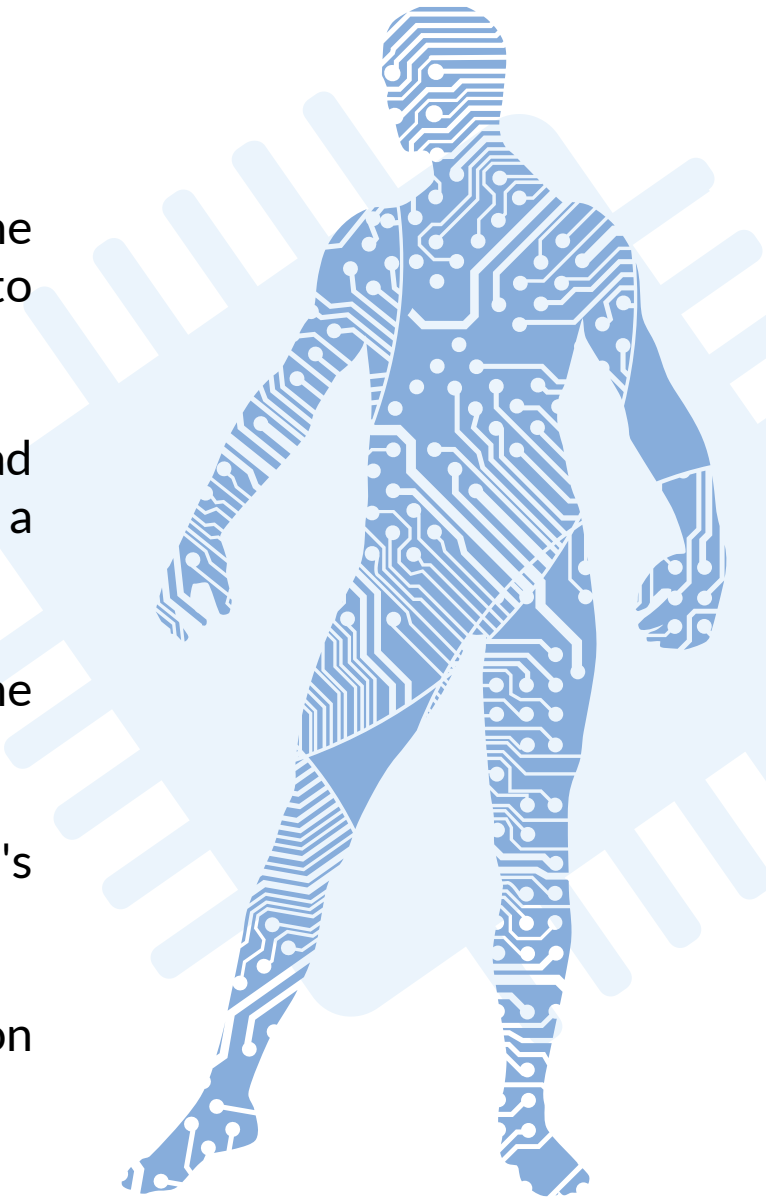
❑ Learning rate is a floating point number you set that influences how quickly the model converges. If the learning rate is too low, the model can take a long time to converge.

❑ However, if the learning rate is too high, the model never converges, but instead bounces around the weights and bias that minimize the loss. The goal is to pick a learning rate that's not too high nor too low so that the model converges quickly.

❑ The learning rate determines the magnitude of the changes to make to the weights and bias during each step of the gradient descent process.

❑ The **model multiplies the gradient by the learning rate** to determine the model's parameters (weight and bias values) for the next iteration.

❑ In the third step of gradient descent, the "small amount" to move in the direction of negative slope refers to the learning rate.



Learning rate

- small amount = 0.01= learning rate

$$New\ weight = old\ weight - (small\ amount * weight\ slope)$$

$$New\ bias = old\ bias - (small\ amount * bias\ slope)$$

$$New\ weight = 0 - (0.01) * (-119.7)$$

$$New\ bias = 0 - (0.01) * (-34.3)$$

$$New\ weight = 1.2$$

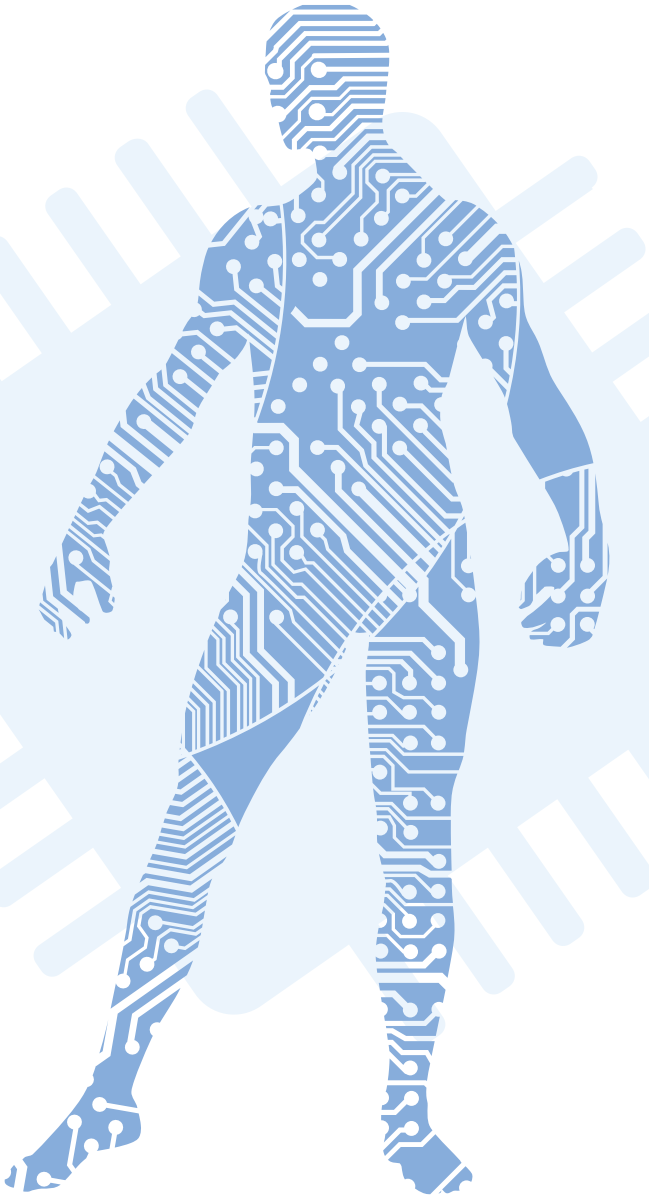
$$New\ bias = 0.34$$

Batch size

❑ Batch size is a hyperparameter that refers to the number of examples the model processes before updating its weights and bias. You might think that the model should calculate the loss for every example in the dataset before updating the weights and bias. However, when a dataset contains hundreds of thousands or even millions of examples, using the full batch isn't practical.

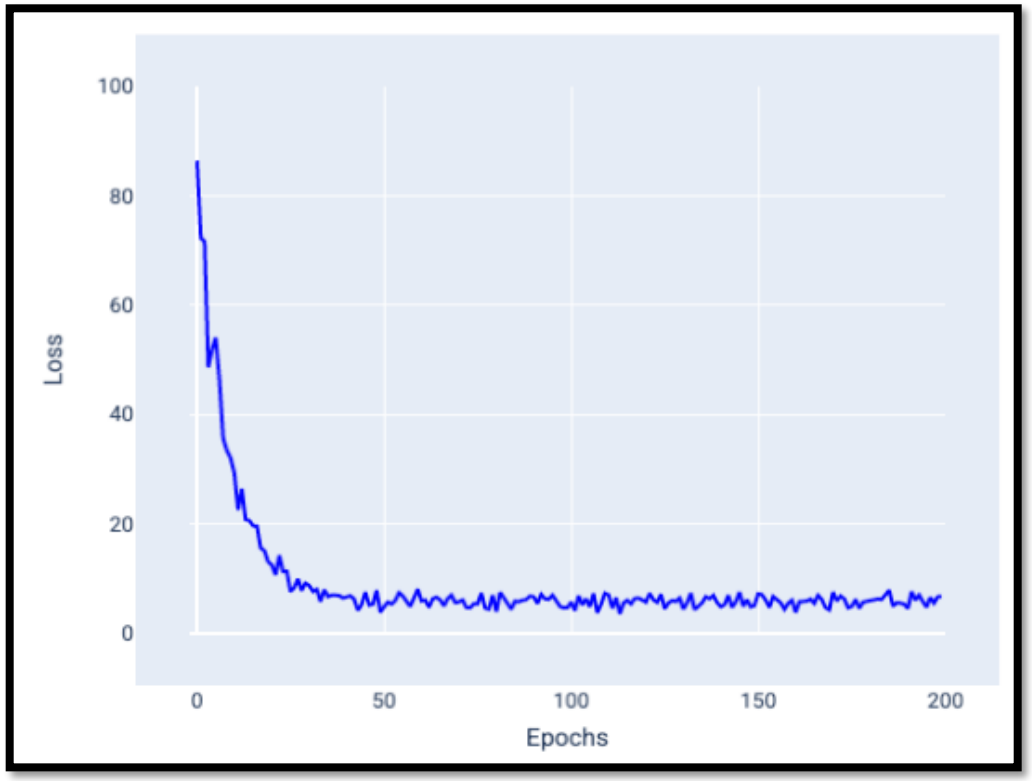
❑Two common techniques to get the right gradient (Weight Slope) on average without needing to look at every example in the dataset before updating the weights and bias are:

- ❑ Stochastic gradient descent
- ❑ Mini-batch stochastic gradient descent.

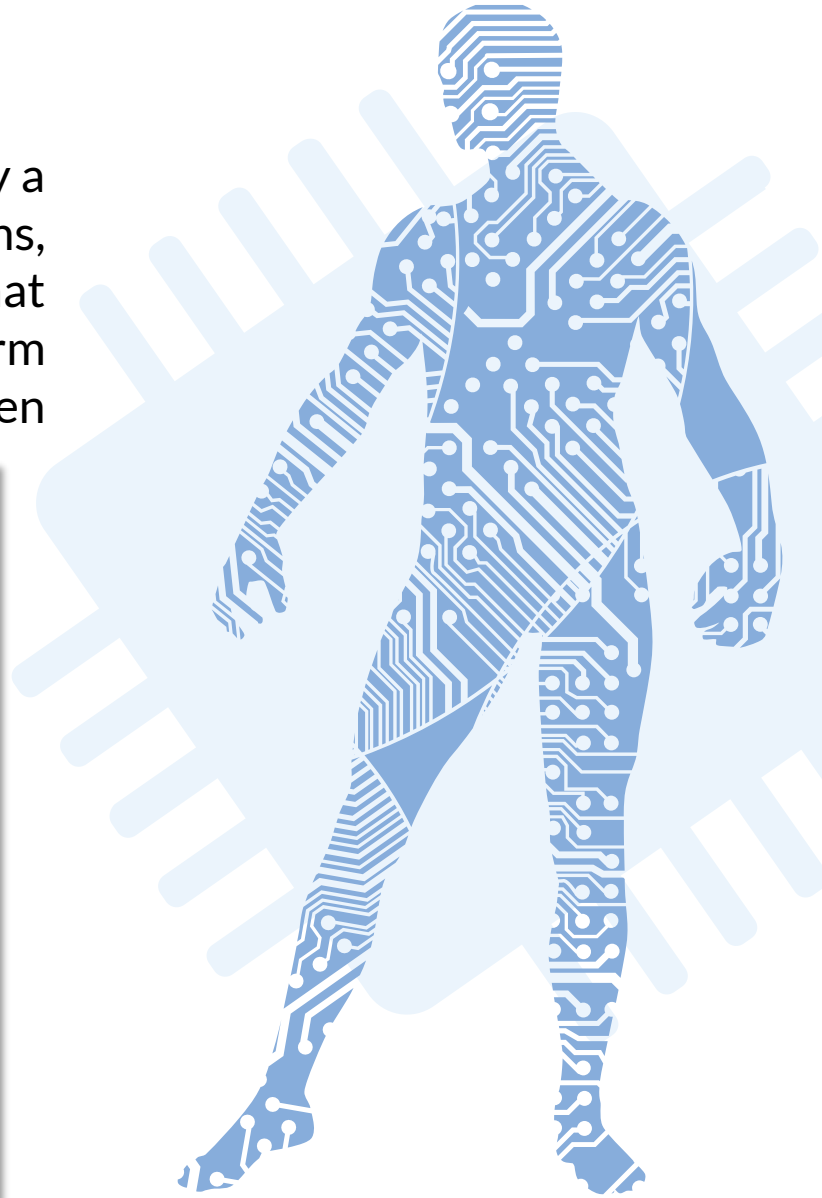


Batch size

❑ **Stochastic gradient descent (SGD):** Stochastic gradient descent uses only a single example (a batch size of one) per iteration. Given enough iterations, SGD works but is very noisy. "Noise" refers to variations during training that cause the loss to increase rather than decrease during an iteration. The term "stochastic" indicates that the one example comprising each batch is chosen at random.



Model trained with stochastic gradient descent (SGD) showing noise in the loss curve.

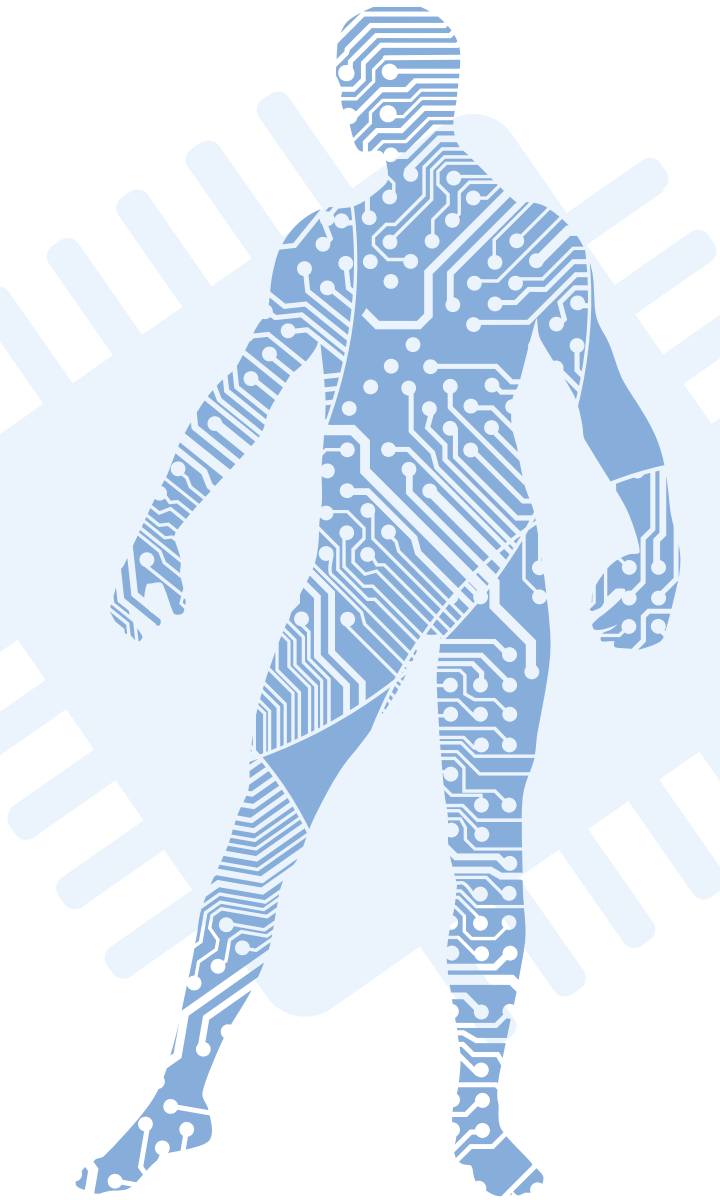


Batch size

❑ **Mini-batch stochastic gradient descent (mini-batch SGD)**: Mini-batch stochastic gradient descent is a compromise between full-batch and SGD. For number N of data points, the batch size can be any number greater than 1 and less than N.

❑ The model chooses the examples included in each batch at random, averages their gradients, and then updates the weights and bias once per iteration.

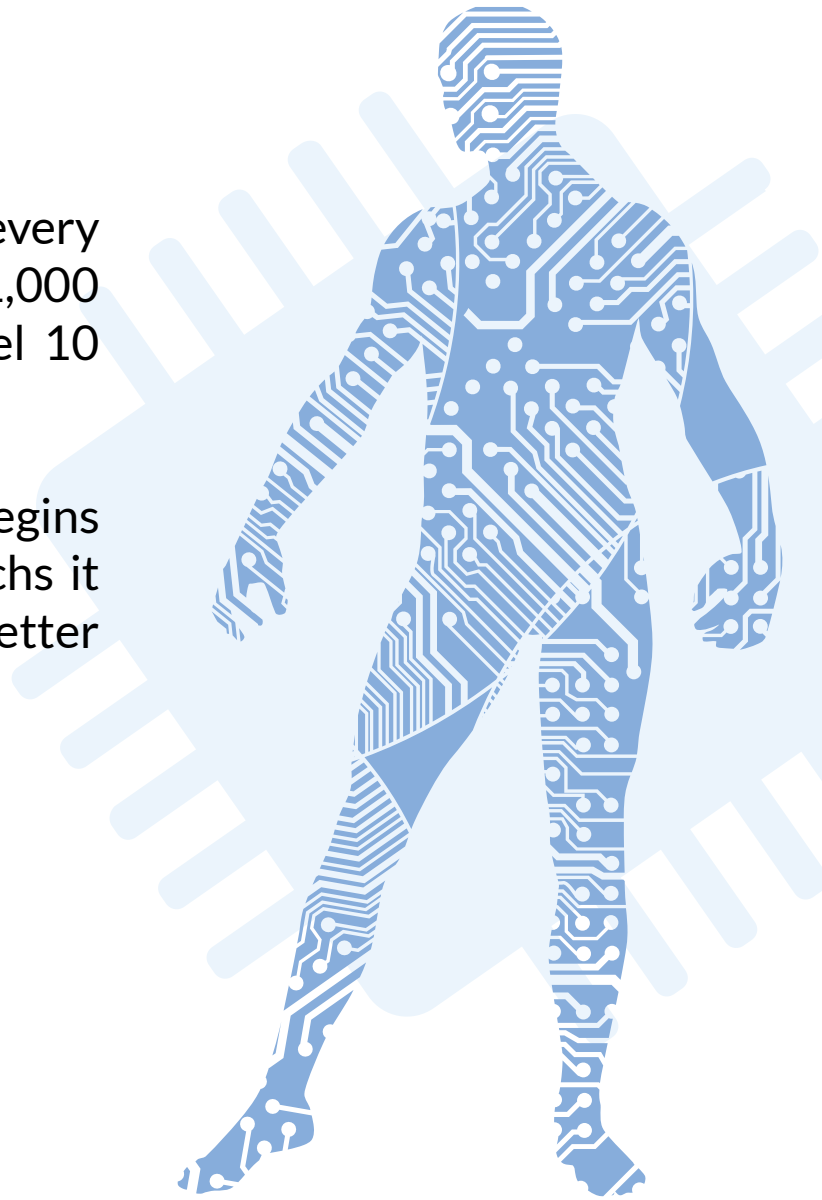
❑ Determining the number of examples for each batch depends on the dataset and the available compute resources. In general, small batch sizes behaves like SGD, and larger batch sizes behaves like full-batch gradient descent



Epochs

❑ During training, an epoch means that the model has processed every example in the training set once. For example, given a training set with 1,000 examples and a mini-batch size of 100 examples, it will take the model 10 iterations to complete one epoch.

❑ The number of epochs is a hyperparameter you set before the model begins training. In many cases, you'll need to experiment with how many epochs it takes for the model to converge. In general, more epochs produces a better model, but also takes more time to train.



Epochs/Batch size

Full batch: an entire dataset

	Feature	Label
0		
.		
.		
.		
.		
500		
.		
.		
.		
.		
1000		

Single batch: a mini batch

	Feature	Label
1		
.		
.		
.		
100		

One epoch: a full batch composed of ten mini batches

	Feature	Label
1		
.		
.		
.		
100		

	Feature	Label
101		
.		
.		
.		
200		

	Feature	Label
201		
.		
.		
.		
300		

	Feature	Label
301		
.		
.		
.		
400		

	Feature	Label
401		
.		
.		
.		
500		

	Feature	Label
501		
.		
.		
.		
600		

	Feature	Label
601		
.		
.		
.		
700		

	Feature	Label
701		
.		
.		
.		
800		

	Feature	Label
801		
.		
.		
.		
900		

	Feature	Label
901		
.		
.		
.		
1000		

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

6 samples

Machine learning model

$$price = w1 * area + w2 * bedrooms + bias$$

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Initialize w1, w2 and bias to be 1

Machine learning model

$price = 1 * area + 1 * bedrooms + 1$

$\widehat{price} = 2604$

$price = 550000$

$error1 = (price - \widehat{price})^2$

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 1 * area + 1 * bedrooms + 1$$

$\widehat{price} = 3005$

$price = 565000$

$$error2 = (price - \widehat{price})^2$$

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 1 * area + 1 * bedrooms + 1$$

$$\widehat{price} = 4107$$

$$price = 810000$$

$$error6 = (price - \widehat{price})^2$$

End of first epoch

Example

$$Total\ Error = error1 + error2 + \ ... + error6$$

$$Mean\ Squred\ Error(a.k.a.MSE) = \frac{Total\ Error}{6}$$

$$w1 = w1 - learning\ rate * {}^{\partial(\mathrm{MSE})}/_{\partial w1} \qquad w1 = 1 - (-50) = 51$$

$$w2 = w2 - learning\ rate * {}^{\partial(\mathrm{MSE})}/_{\partial w2} \qquad w2 = 1 - (-8) = 9$$

$$b = b - learning\ rate * {}^{\partial(\mathrm{MSE})}/_{\partial b} \qquad bias = 1 - (-20000) = 20001$$

Example

Beginning of the second epoch

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 51 * area + 9 * bedrooms + 20001$$

$$\widehat{price} = 152610$$

$$price = 550000$$

$$error1 = (price - \widehat{price})^2$$

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

Machine learning model

$$price = 51 * area + 9 * bedrooms + 20001$$

$$\widehat{price} = 229155$$

$$price = 550000$$

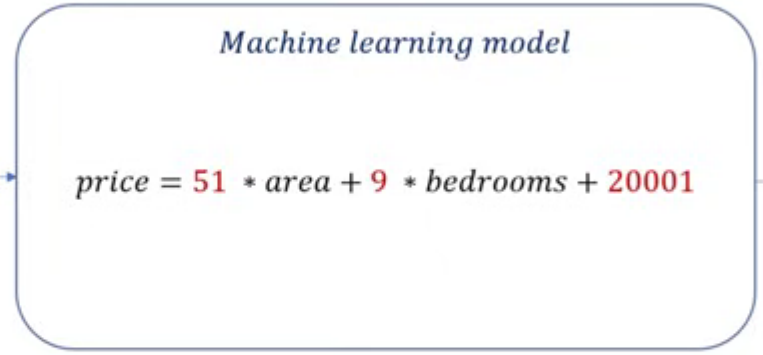
$$error6 = (price - \widehat{price})^2$$

End of second epoch

Continue until the total number of epochs finished or the MSE converges to a lower value and hence the model is said to be trained.

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000



$\widehat{price} = 229155$

$price = 550000$

$error6 = (price - \widehat{price})^2$

End of second epoch

This is called full-batch gradient descent

Example

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 million samples

1) To find cumulative error for first round (epoch) now we need to do a forward pass for 10 million samples
2) We have 2 features (area and bedroom). This requires finding 20 million derivatives

What if I have 200 features ?

Example

1. *Randomly pick single data training sample*

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 *million samples*

Machine learning model

$price = 1 * area + 1 * bedrooms + 1$

$\widehat{price} = 3204$

$price = 610000$

$error = (price - \widehat{price})^2$

Example

2. Adjust weights

$$w1 = w1 - learning\ rate * {}^{\partial(\text{error})}/_{\partial w1} \qquad w1 = 1 - (-13) = 14$$

$$w2 = w2 - learning\ rate * {}^{\partial(\text{error})}/_{\partial w2} \qquad w2 = 1 - (-3) = 4$$

$$b = b - learning\ rate * {}^{\partial(\text{error})}/_{\partial b} \qquad bias = 1 - (-1500) = 1501$$

Example

3. Again randomly pick a training sample

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 million samples

Machine learning model

$price = 14 * area + 4 * bedrooms + 1501$

$\widehat{price} = 3204$

$price = 37913$

$error = (price - \widehat{price})^2$

Example

4. Again adjust weights

$$w1 = w1 - learning\ rate * {}^{\partial(\text{error})}/_{\partial w1} \qquad w1 = 14 - (-100) = 114$$

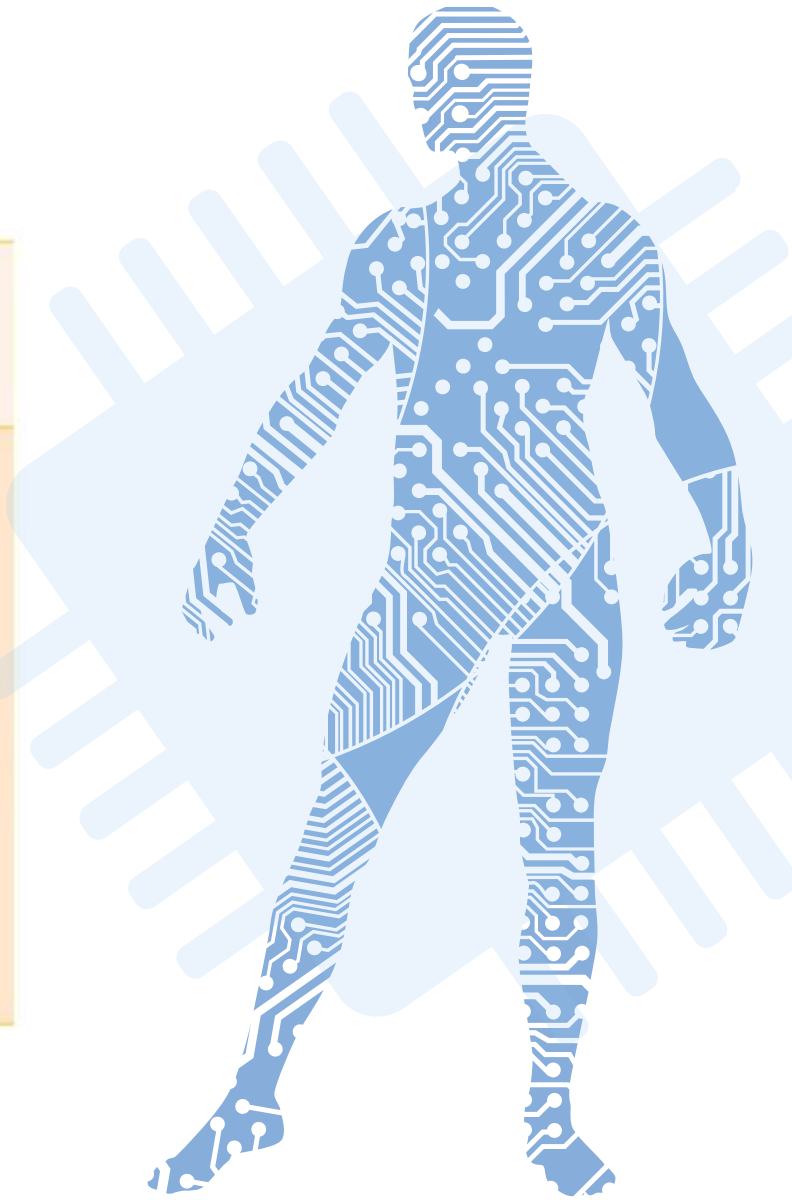
$$w2 = w2 - learning\ rate * {}^{\partial(\text{error})}/_{\partial w2} \qquad w2 = 4 - (-12) = 16$$

$$b = b - learning\ rate * {}^{\partial(\text{error})}/_{\partial b} \qquad bias = 1501 - (-2001) = 3502$$

Adjust weight after every training sample forward path
This is called Stochastic Gradient Descent

Example

Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini batch gradient descent
Use **all** training samples for one forward pass and then adjust weights	Use **one** (randomly picked) sample for a forward pass and then adjust weights	Use **a batch of** (randomly picked) samples for a forward pass and then adjust weights



Example

```
def batch_gradient_descent(X, y_true, epochs, learning_rate = 0.01):

    number_of_features = X.shape[1]
    # numpy array with 1 row and columns equal to number of features. In
    # our case number_of_features = 2 (area, bedroom)
    w = np.ones(shape=(number_of_features))
    b = 0
    total_samples = X.shape[0] # number of rows in X

    cost_list = []
    epoch_list = []

    for i in range(epochs):
        y_predicted = np.dot(w, X.T) + b

        w_grad = -(2/total_samples)*(X.T.dot(y_true-y_predicted))
        b_grad = -(2/total_samples)*np.sum(y_true-y_predicted)

        w = w - learning_rate * w_grad
        b = b - learning_rate * b_grad

        cost = np.mean(np.square(y_true-y_predicted)) # MSE (Mean Squared Error)

        if i%10==0:
            cost_list.append(cost)
            epoch_list.append(i)

    return w, b, cost, cost_list, epoch_list

w, b, cost, cost_list, epoch_list = batch_gradient_descent(scaled_X,scaled_y.reshape(scaled_y.shape[0]
```

Example

```
def stochastic_gradient_descent(X, y_true, epochs, learning_rate = 0.01):

    number_of_features = X.shape[1]
    # numpy array with 1 row and columns equal to number of features. In
    # our case number_of_features = 3 (area, bedroom and age)
    w = np.ones(shape=(number_of_features))
    b = 0
    total_samples = X.shape[0]

    cost_list = []
    epoch_list = []

    for i in range(epochs):
        random_index = random.randint(0,total_samples-1) # random index from total samples
        sample_x = X[random_index]
        sample_y = y_true[random_index]

        y_predicted = np.dot(w, sample_x.T) + b

        w_grad = -(2/total_samples)*(sample_x.T.dot(sample_y-y_predicted))
        b_grad = -(2/total_samples)*(sample_y-y_predicted)

        w = w - learning_rate * w_grad
        b = b - learning_rate * b_grad

        cost = np.square(sample_y-y_predicted)

        if i%100==0: # at every 100th iteration record the cost and epoch value
            cost_list.append(cost)
            epoch_list.append(i)

return w, b, cost, cost_list, epoch_list
```

Example

```
def mini_batch_gradient_descent(X, y_true, epochs = 100, batch_size = 5, learning_rate = 0.01):

    number_of_features = X.shape[1]
```

```
w = np.ones(shape=(number_of_features))
b = 0
total_samples = X.shape[0] # number of rows in X

if batch_size > total_samples: # In this case mini batch becomes same as batch gradient descent
    batch_size = total_samples

cost_list = []
epoch_list = []

num_batches = int(total_samples/batch_size)
```

Example

```
    for i in range(epochs):
        random_indices = np.random.permutation(total_samples)
        X_tmp = X[random_indices]
        y_tmp = y_true[random_indices]

        for j in range(0,total_samples,batch_size):
            Xj = X_tmp[j:j+batch_size]
            yj = y_tmp[j:j+batch_size]
            y_predicted = np.dot(w, Xj.T) + b

            w_grad = -(2/len(Xj))*(Xj.T.dot(yj-y_predicted))
            b_grad = -(2/len(Xj))*np.sum(yj-y_predicted)

            w = w - learning_rate * w_grad
            b = b - learning_rate * b_grad

            cost = np.mean(np.square(yj-y_predicted)) # MSE (Mean Squared Error)

        if i%10==0:
            cost_list.append(cost)
            epoch_list.append(i)

    return w, b, cost, cost_list, epoch_list

w, b, cost, cost_list, epoch_list = mini_batch_gradient_descent(
    scaled_X,
    scaled_y.reshape(scaled_y.shape[0],),
    epochs = 120,
    batch_size = 5
)
```

Notes

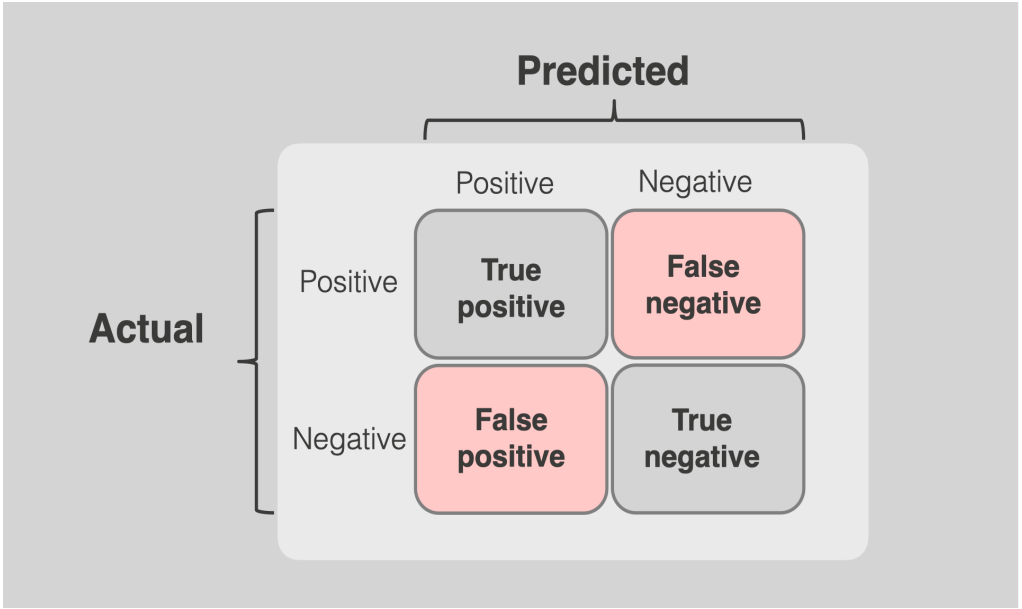
❑ The following table describes how batch size and epochs relate to the number of times a model updates its parameters.

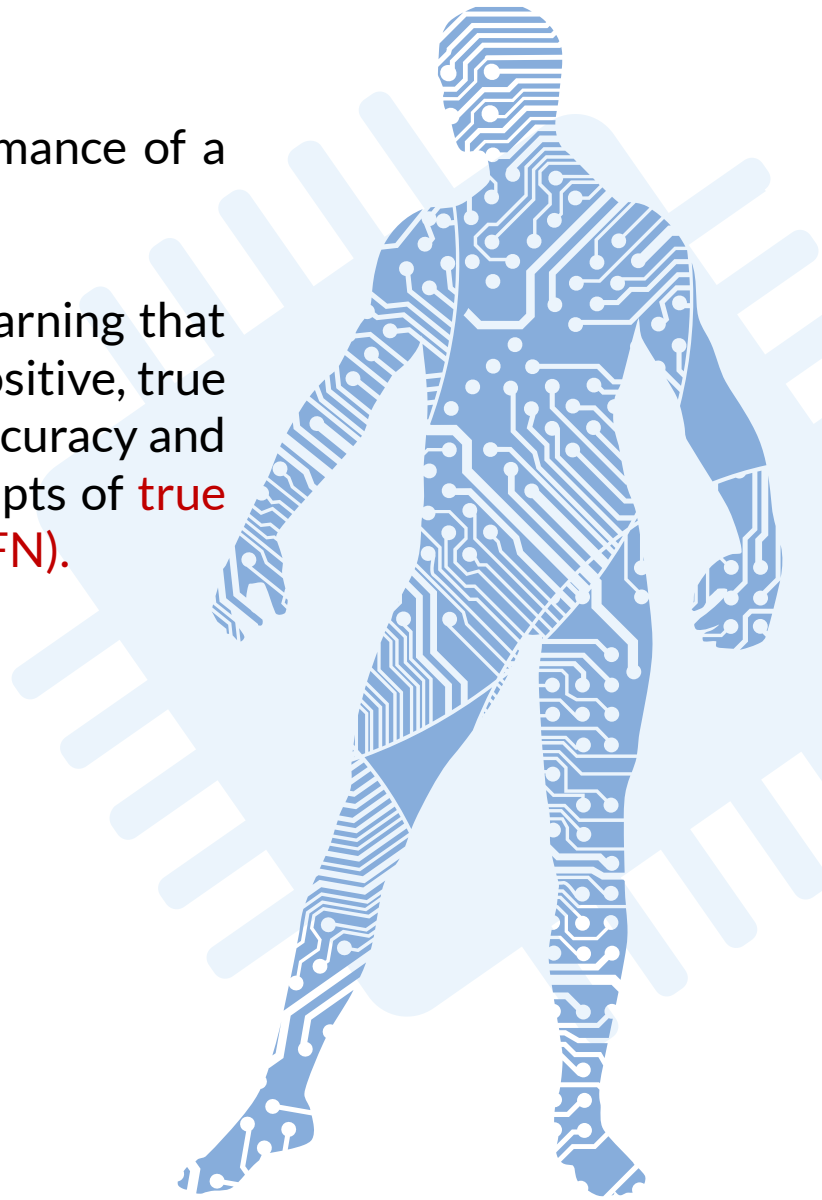
Batch type	When weights and bias updates occur
Full batch	After the model looks at all the examples in the dataset. For instance, if a dataset contains 1,000 examples and the model trains for 20 epochs, the model updates the weights and bias 20 times, once per epoch.
Stochastic gradient descent	After the model looks at a single example from the dataset. For instance, if a dataset contains 1,000 examples and trains for 20 epochs, the model updates the weights and bias 20,000 times.
Mini-batch stochastic gradient descent	After the model looks at the examples in each batch. For instance, if a dataset contains 1,000 examples, and the batch size is 100, and the model trains for 20 epochs, the model updates the weights and bias 200 times.

Decoding the confusion matrix

❑ A confusion matrix or error matrix is used for summarizing the performance of a classification algorithm.

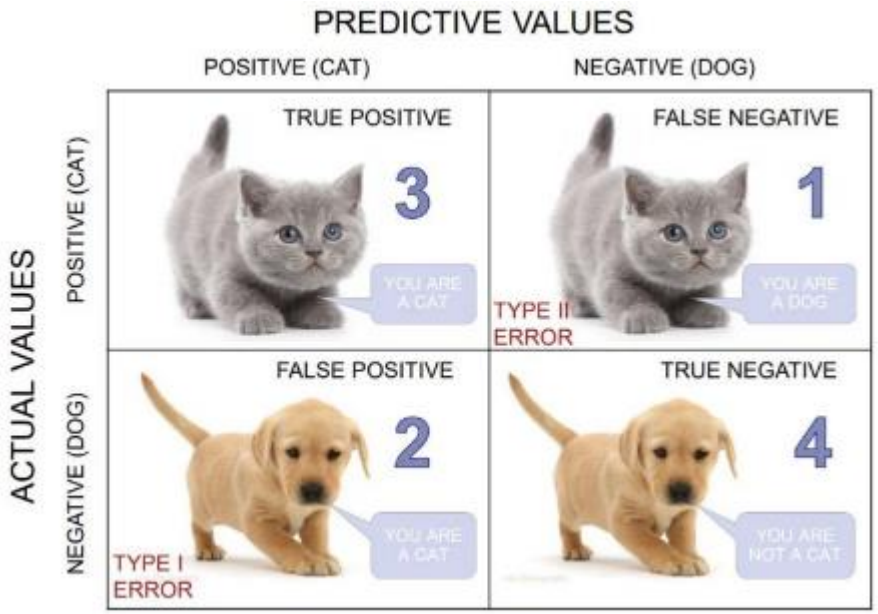
❑ A confusion matrix is a performance evaluation tool used in machine learning that summarizes the performance of a classification model by tabulating true positive, true negative, false positive, and false negative predictions. It helps assess the accuracy and effectiveness of the model's predictions. The matrix is based on the concepts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).





Notes

(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

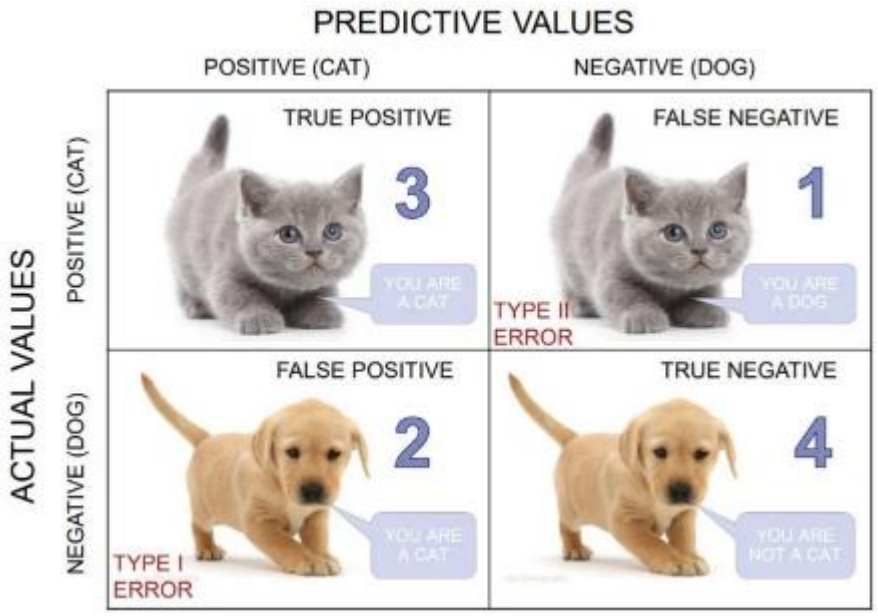
Predicated Values

Actual Values

	1	0
1	y_true=1 y_pred=1	y_true=1 y_pred=0
0	y_true=0 y_pred=1	y_true=0 y_pred=0

Notes

(First term: If the model answers the prediction question correctly, we said True, else False)
(Second term: describes the predicted value)



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

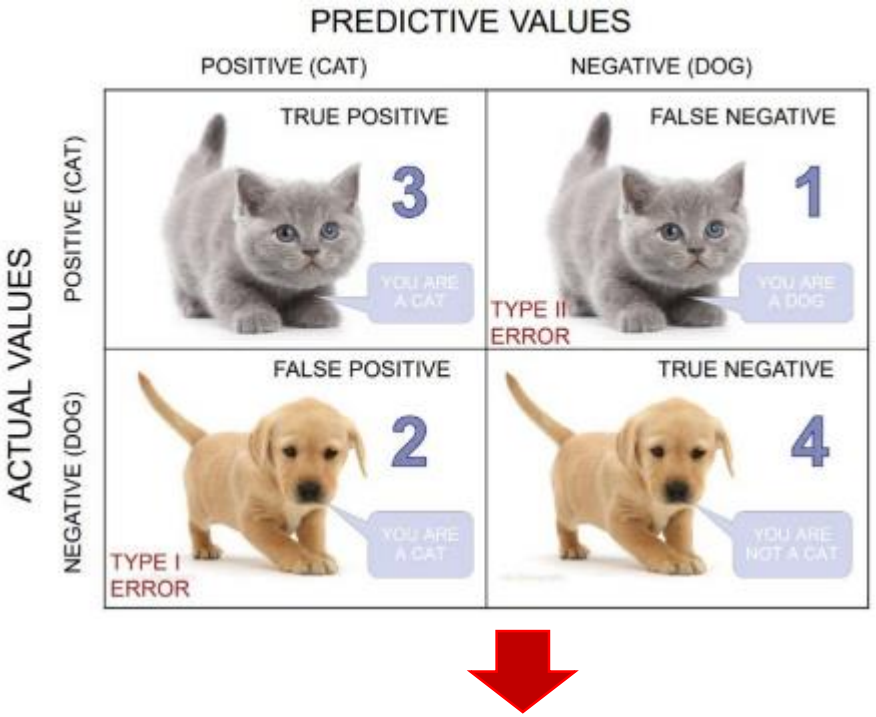
Predicated Values

Actual Values

	1	0
1	y_true=1 y_pred=1	y_true=1 y_pred=0
0	y_true=0 y_pred=1	y_true=0 y_pred=0

Notes

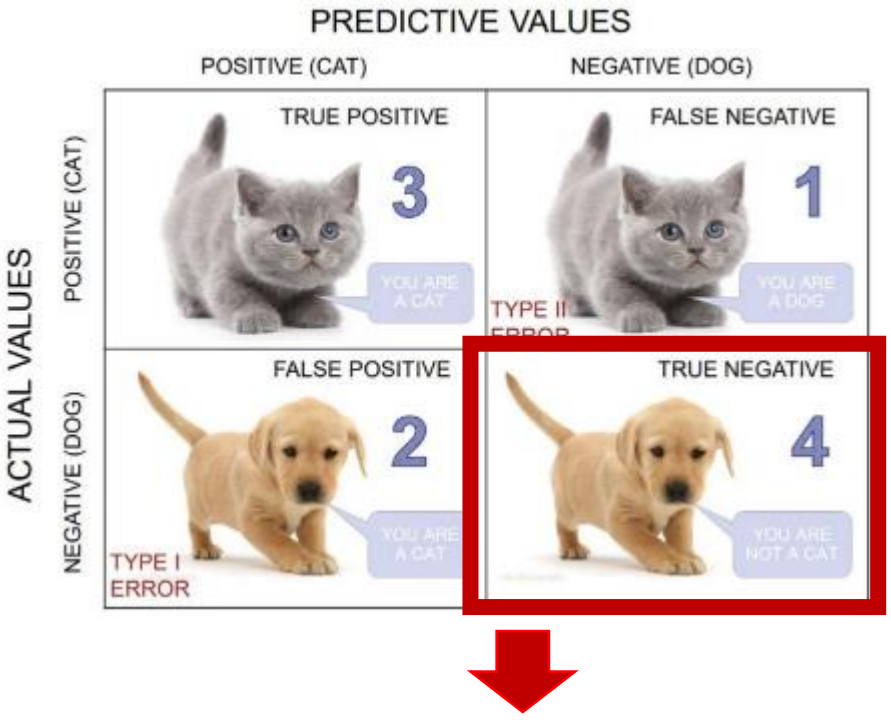
(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)



Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

Notes

(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)

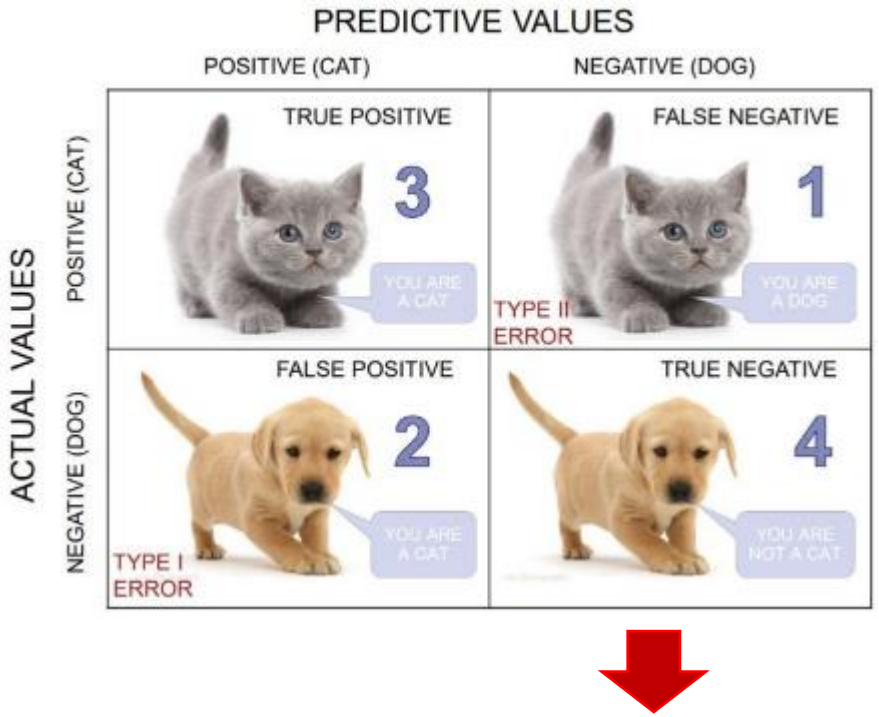


Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

TN

Notes

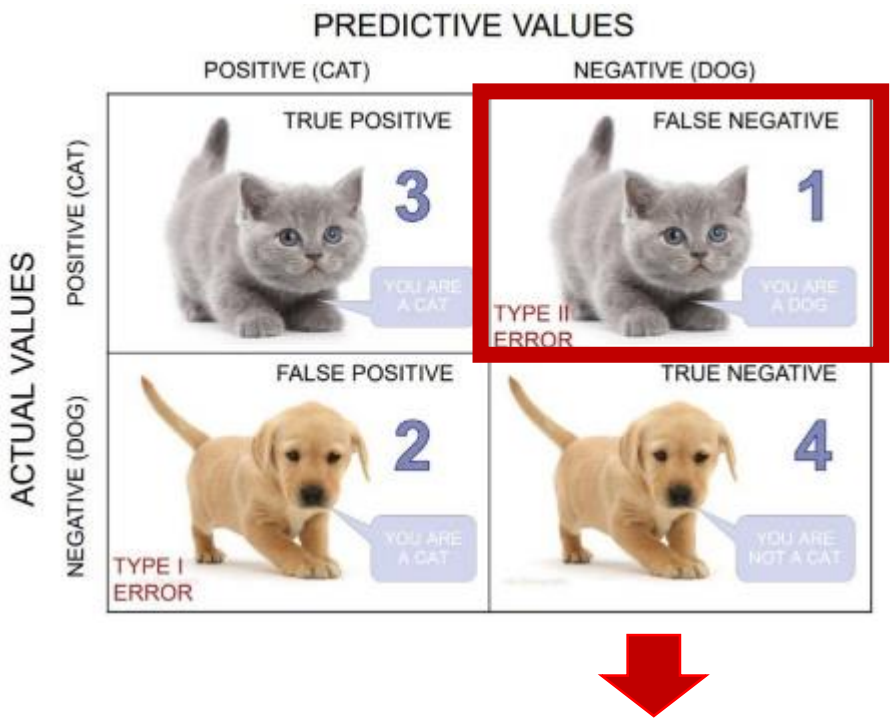
(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)



Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

Notes

(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)

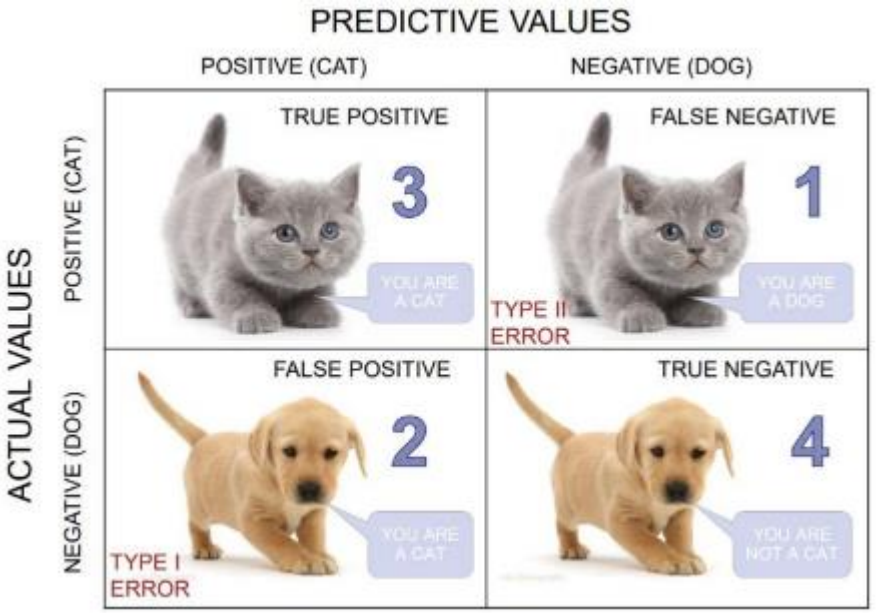


Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

FN

Notes

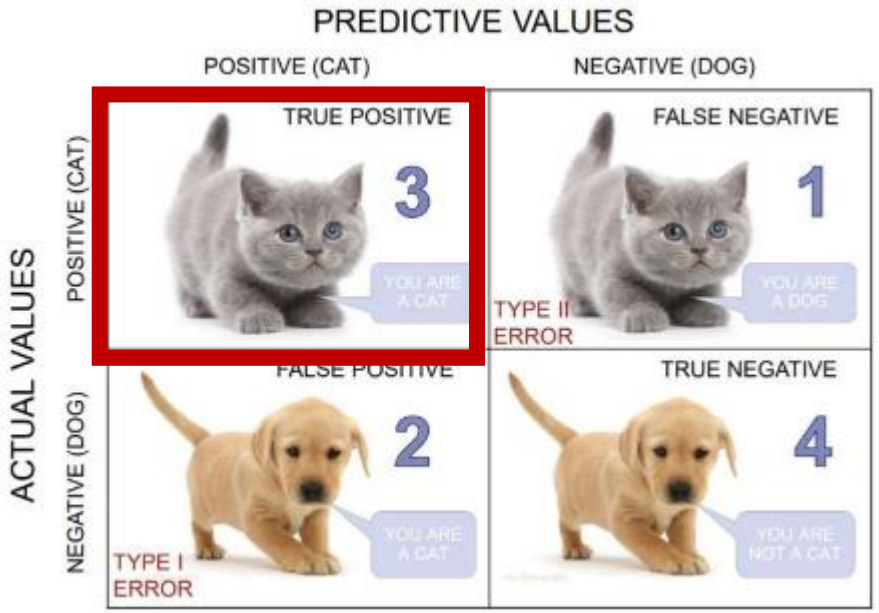
(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)



Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

Notes

(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)

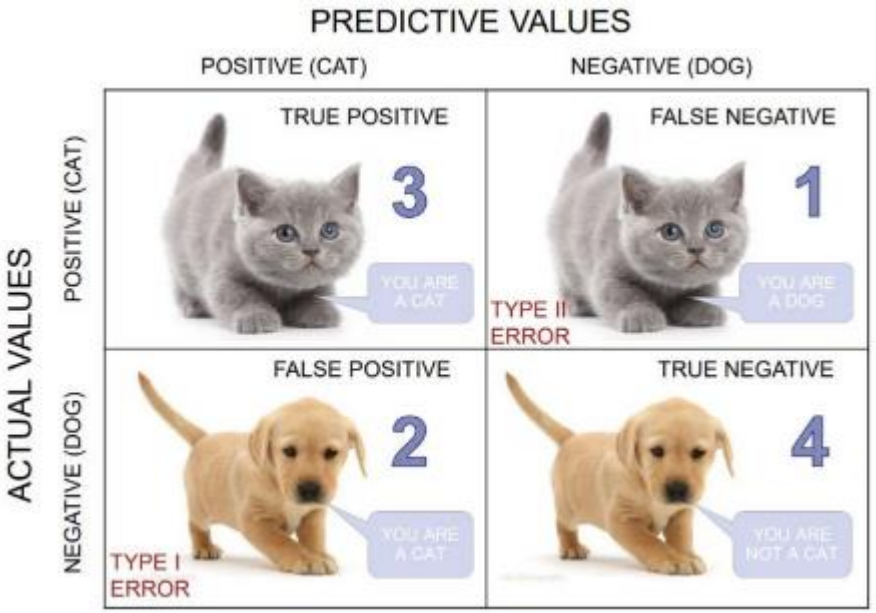


Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

TP

Notes

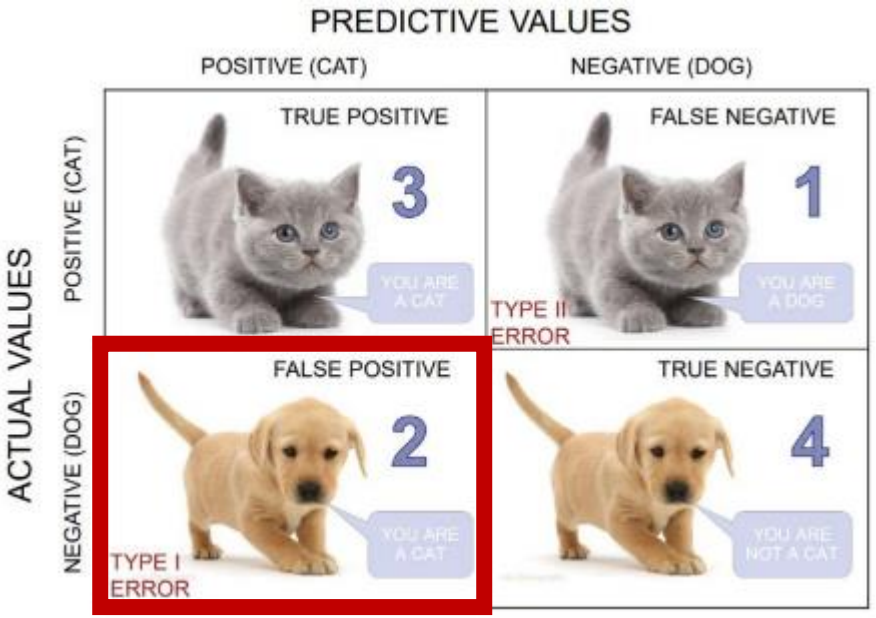
(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)



Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

Notes

(Remember, we describe predicted values as Positive/Negative and actual values as True/False.)



Actual values = [‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘cat’, ‘dog’]
Predicted values = [‘dog’, ‘dog’, ‘dog’, ‘cat’, ‘dog’, ‘dog’, ‘cat’, ‘cat’, ‘cat’, ‘cat’]

FP

Accuracy & Recall

❑ Accuracy is the proportion of all classifications that were correct, whether positive or negative. It is mathematically defined as:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

❑ The true positive rate (TPR), or the proportion of all actual positives that were classified correctly as positives, is also known as recall.

Recall is mathematically defined as:

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

Accuracy & Recall

❑ **Accuracy** is the proportion of all classifications that were correct, whether positive or negative. It is mathematically defined as:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

❑ **The true positive rate (TPR),** or the proportion of all actual positives that were classified correctly as positives, is also known as **recall** (aka. **Sensitivity**).

Recall is mathematically defined as:

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

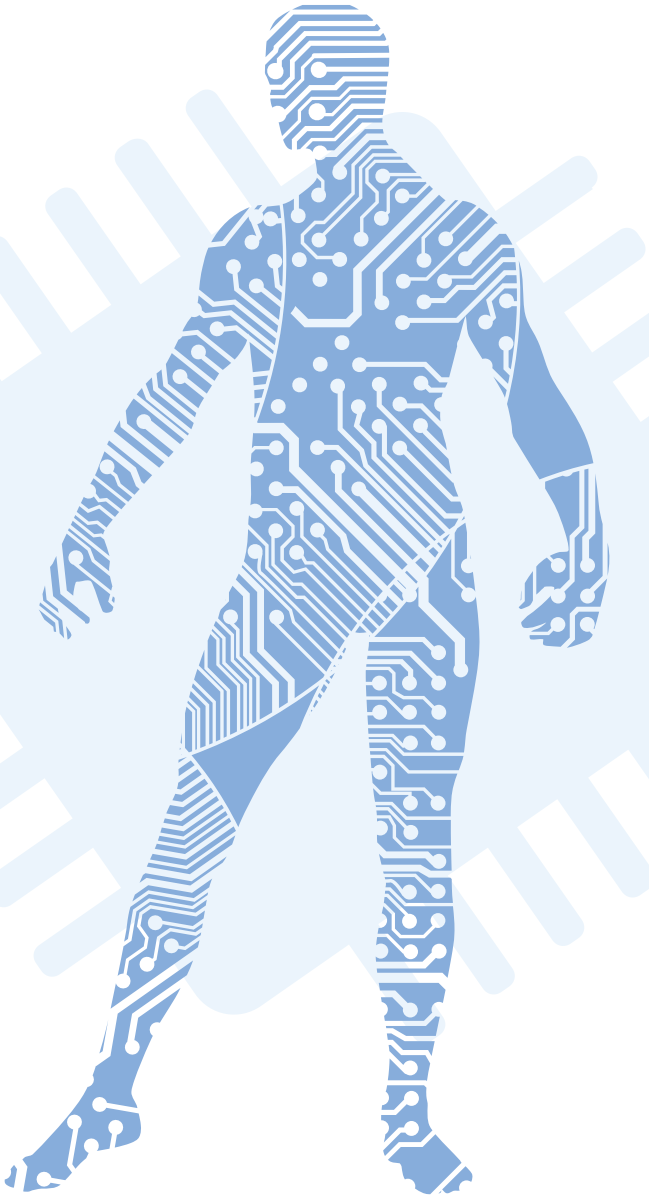
FPR

- In an **imbalanced dataset** where the number of actual positives is very low, **recall is a more meaningful metric than accuracy** because it measures the ability of the model to correctly identify all positive instances.

- For applications like disease prediction, correctly identifying the positive cases is crucial. **A false negative typically has more serious consequences than a false positive.**

- **The false positive rate (FPR)** is the proportion of all actual negatives that were classified incorrectly as positives, also known as the probability of false alarm. It is mathematically defined as:

$$\text{FPR} = \frac{\text{incorrectly classified actual negatives}}{\text{all actual negatives}} = \frac{FP}{FP + TN}$$



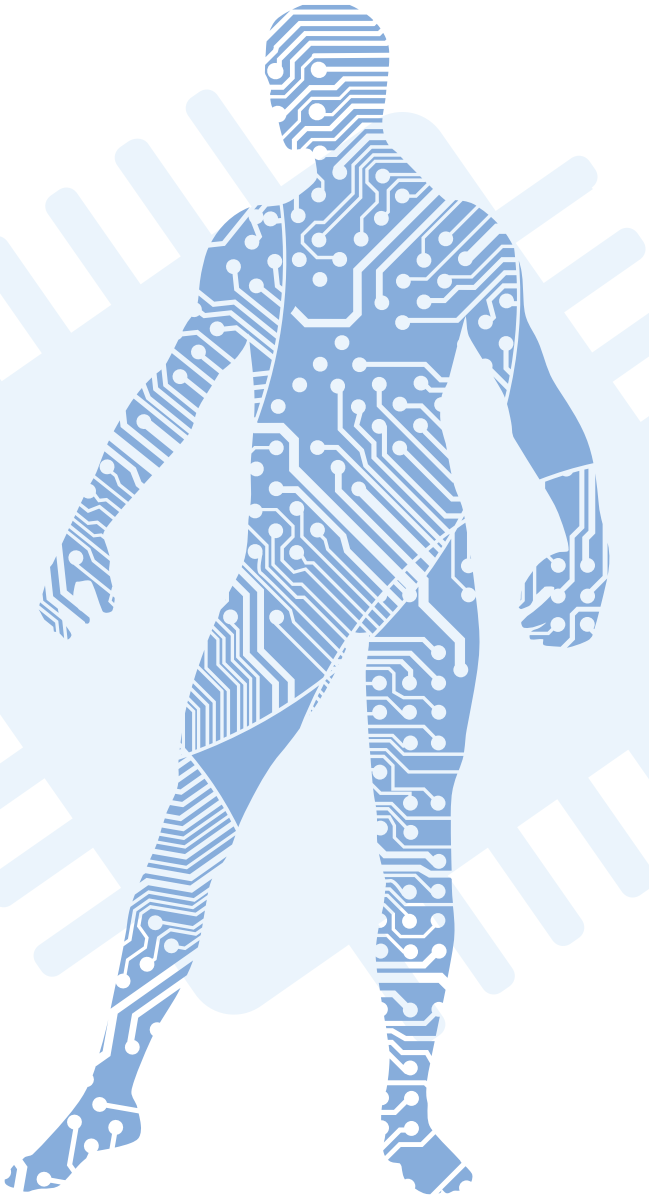
FPR

- **Precision** is the proportion of all the model's positive classifications that are actually positive. It is mathematically defined as:

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

- **F-score or F1-score**: It is difficult to compare two models with different Precision and Recall. So to make them comparable, we use F-Score. It is the Harmonic Mean of Precision and Recall.

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

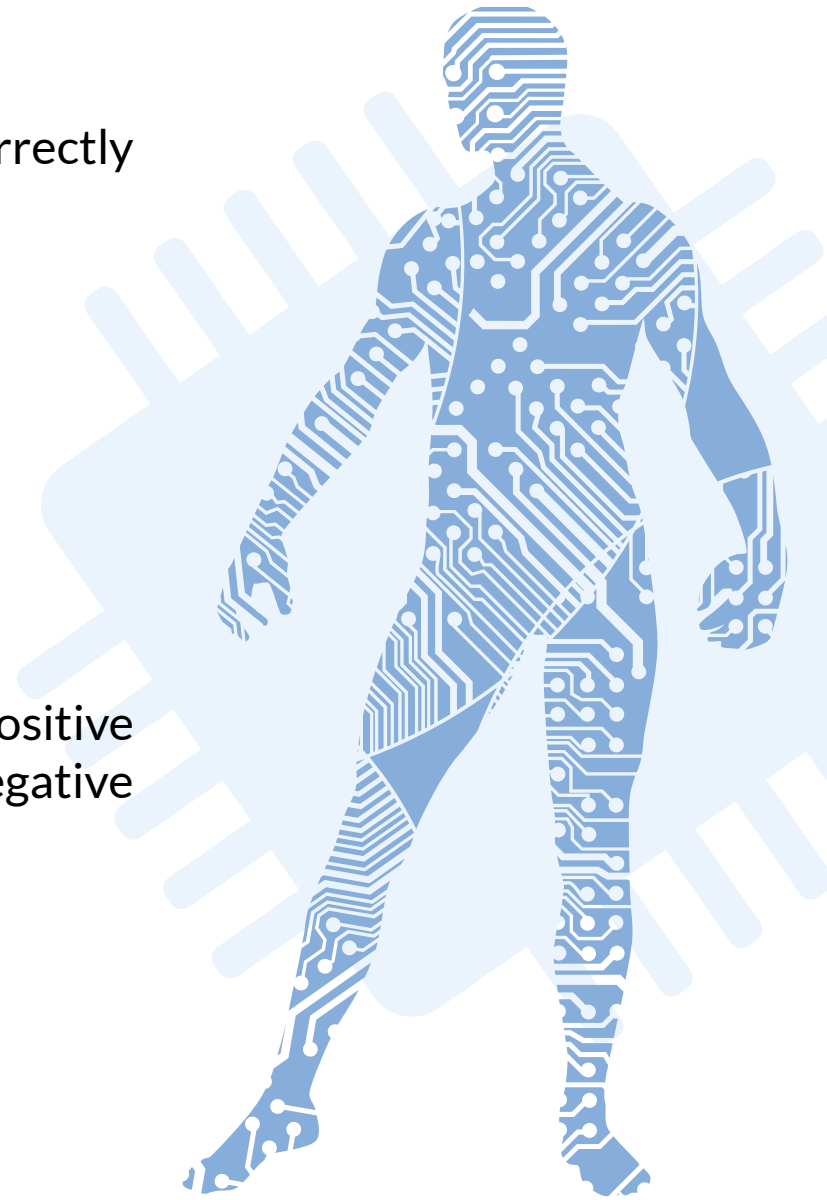


Specificity

- **Specificity** determines the proportion of actual negatives that are correctly identified.

$$\text{Specificity} = \frac{\text{True negative}}{\text{True negative + false positive}}$$

- **Sensitivity** measures the model's ability to correctly identify actual positive cases whereas specificity focuses on correctly identifying actual negative cases.




Thank You