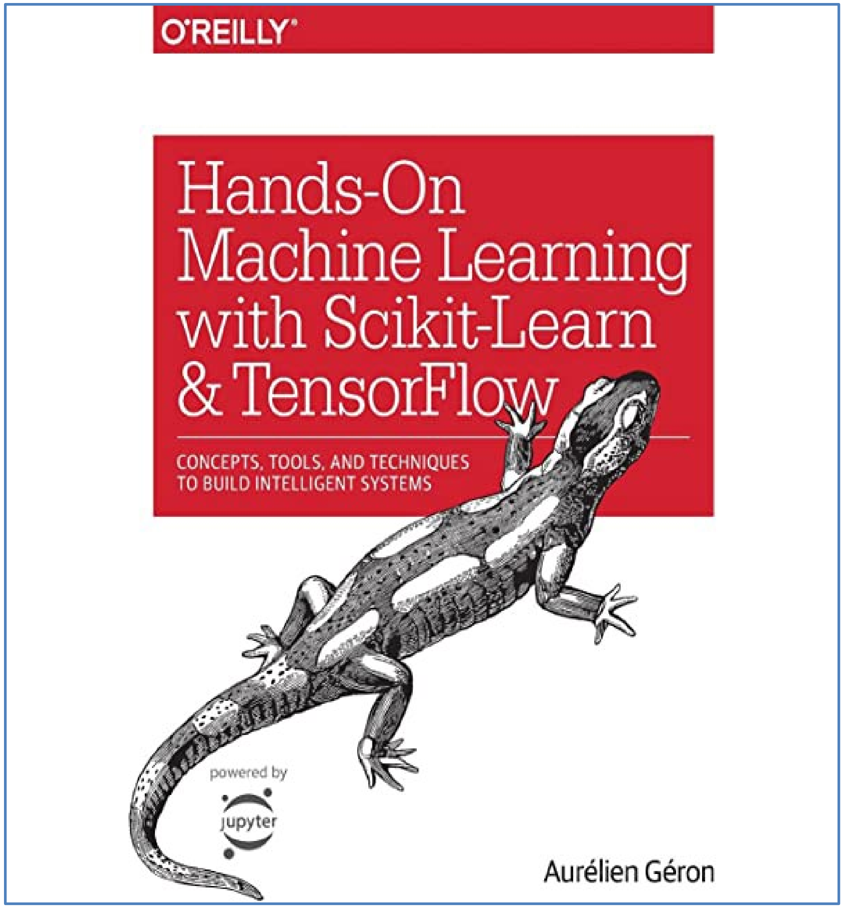
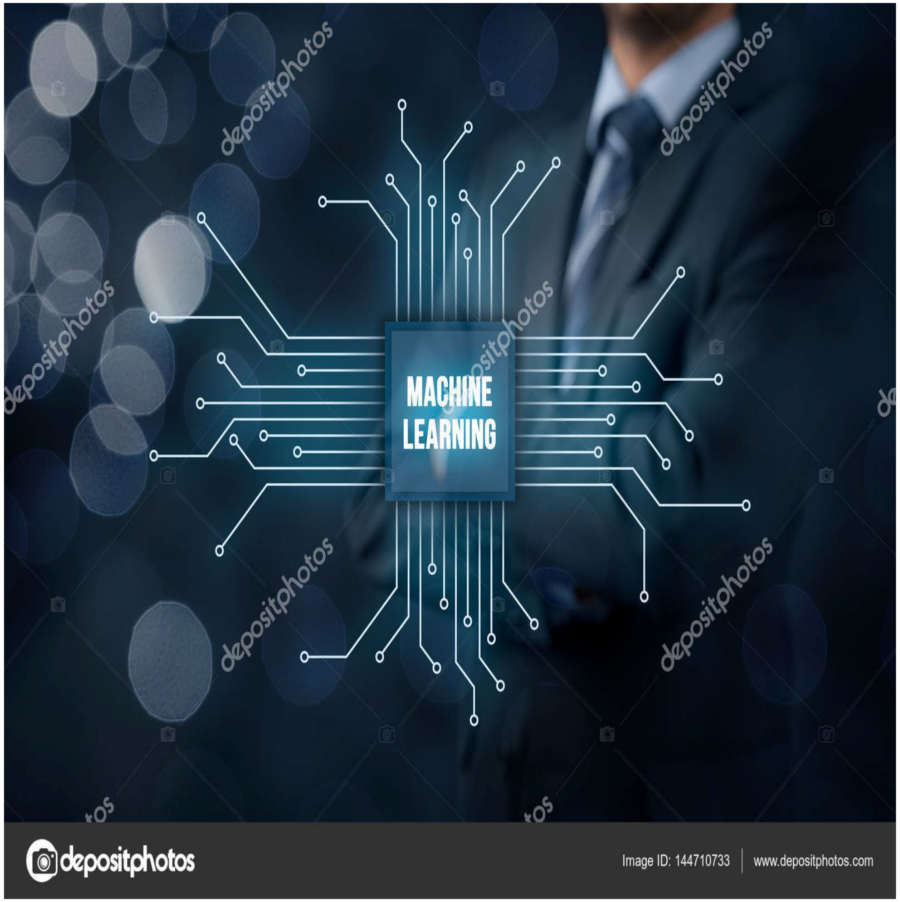
# Machine Learning




**Introduced by**

**Dr. Ebtsam Adel**

# Machine Learning

Chapter 3

# Classification

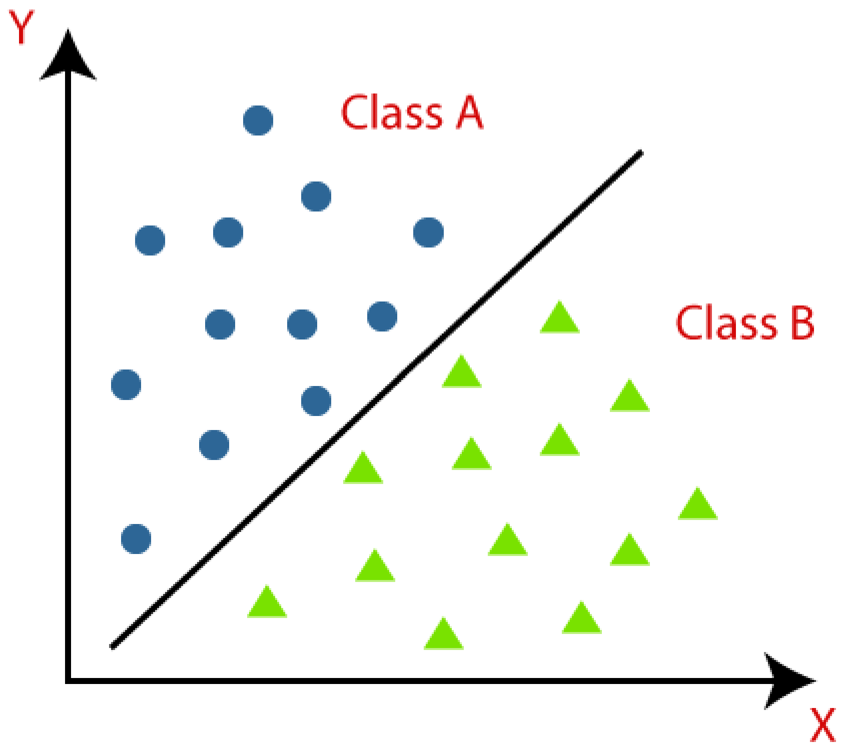
# Classification

**What is the Classification Algorithm?**

- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.

- In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog,** etc. Classes can be called as targets/labels or categories.

# Classification




- Unlike regression, the output variable of Classification is a **category**, not a value, such as "Green or Blue", "fruit or animal", etc.

# Classification

The most commonly used classifiers are as follows:

- ✓ Logistic regression
- ✓ Naive Bayes classifier
- ✓ Support vector machines
- ✓ k-nearest neighbor
- ✓ Decision trees
- ✓ Random forests
- ✓ Neural networks
- ✓ ………………

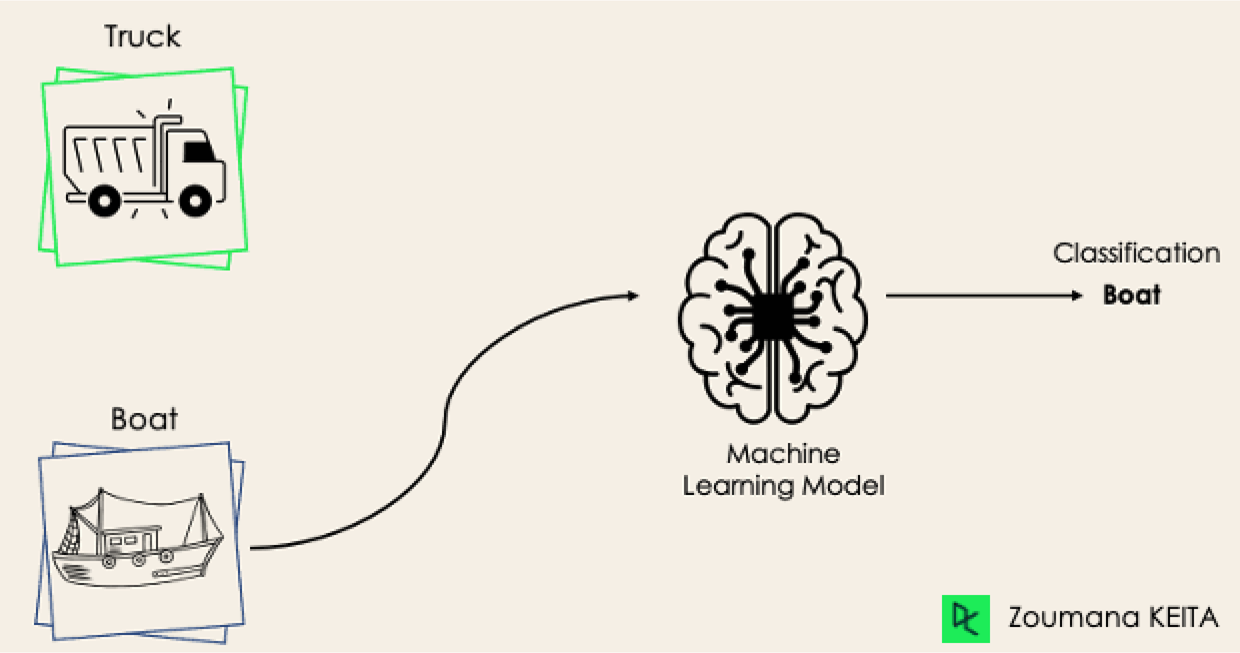
# Classification

**Different Types of Classification Tasks in Machine Learning**

There are many main classification tasks in Machine learning.

- **Binary Classification**

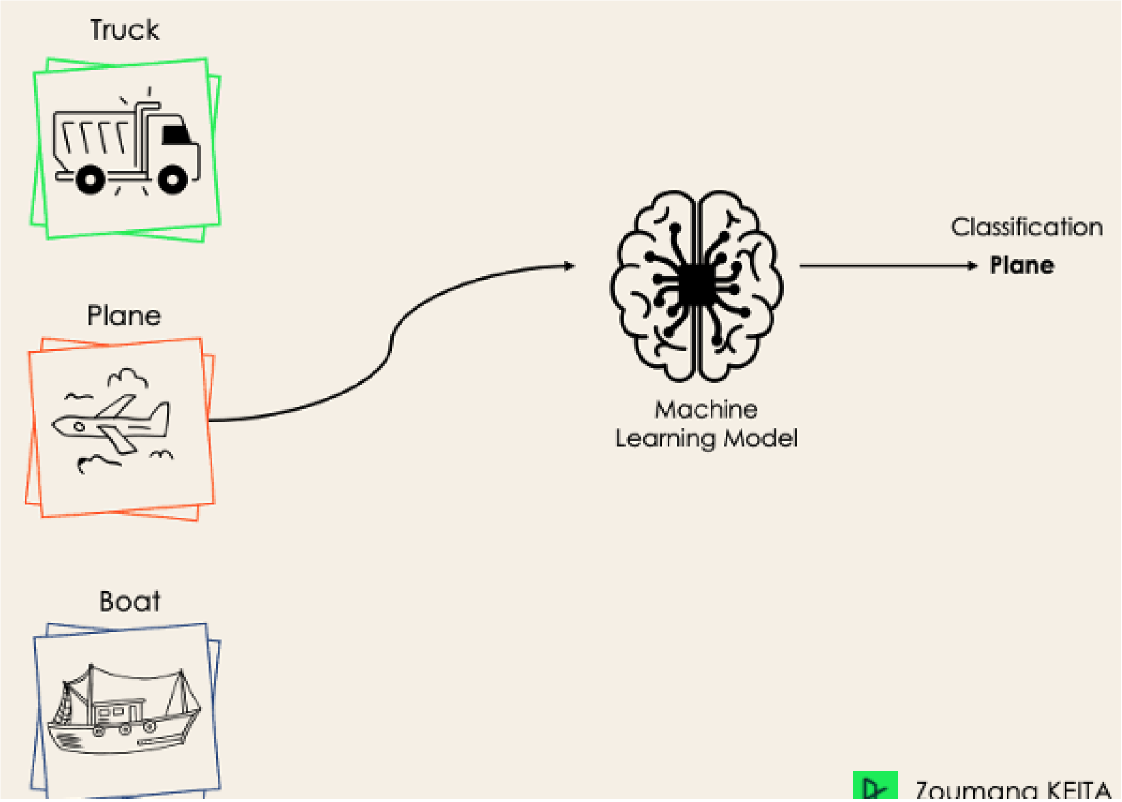
In a binary classification task, the goal is to classify the input data into two exclusive categories.



# Classification

- **Multi-Class Classification**

The multi-class classification, on the other hand, has at least two exclusive class labels, where the goal is to predict to which class a given input example belongs to.



# Classification

**MNIST**

- the MNIST dataset, which is a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau.
- Each image is labeled with the digit it represents.
- This set has been studied so much that it is often called the "Hello World" of Machine Learning.
- There are 70,000 images, and each image has **784** features. This is because each image is 28×28 pixels, and each feature simply represents one pixel's intensity, from 0 (white) to 255 (black).

# Classification

- The MNIST dataset is actually already split into a training set (the first 60,000 images) and a test set (the last 10,000 images):

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

# Classification

```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1)
>>> mnist.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details',
           'categories', 'url'])
```

- A **DESCR** key describing the dataset.
- A **data** key containing an array with one row per instance and one column per feature.
- A **target** key containing an array with the labels.

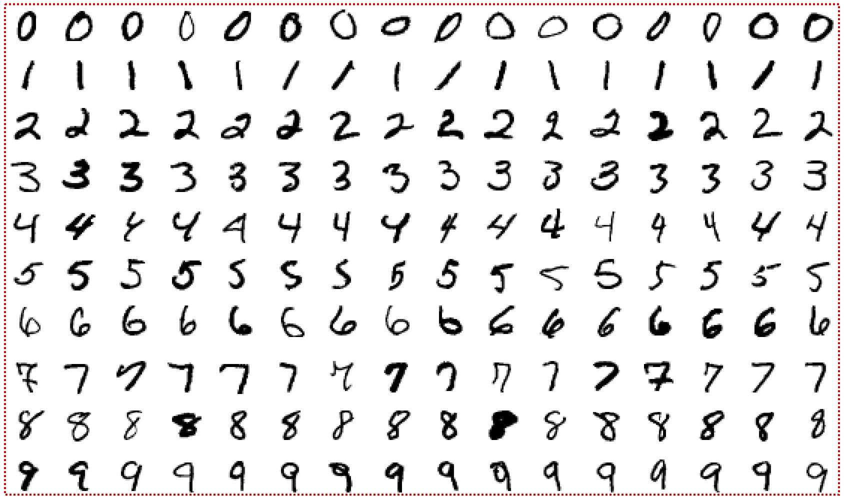
# Classification

the **complexity** of the classification task.

# Classification

- The training set is already **shuffled** for us, which is good as this guarantees that all cross-validation folds will be **similar** Moreover, some learning algorithms are sensitive to the order of the training instances, and they perform poorly if they get many similar instances in a row.

- **Shuffling** the dataset ensures that this won't happen.



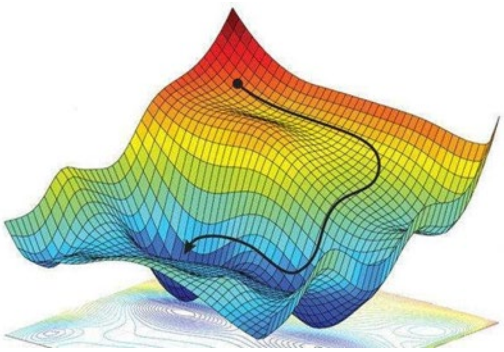
# Training a Binary Classifier

- This "**5-detector**" will be an example of a binary classifier, capable of Separating between just **two classes, 5 and not 5**.
- Let's create the target vectors for this classification task:

```
y_train_5 = (y_train == 5) #        True for all 5s, False for all other digits.
y_test_5 = (y_test == 5)
```

- good place to start is with a Stochastic Gradient Descent (**SGD**) classifier, using Scikit-Learn's **SGDClassifier** class.

- This classifier has the advantage of being capable of handling very large datasets efficiently. This is in part because SGD deals with training instances independently, one at a time.
- Suitable for online learning.



# Performance Measures

# Machine Learning - Performance Metrics

**Performance metrics** in machine learning are used to **evaluate** the performance of a machine learning model.

- Performance metrics are important because they help us understand how well our model is performing and whether it is meeting our requirements.

- There are many performance metrics that can be used in machine learning, depending on the type of problem being solved and the specific requirements of the problem.

- Some common performance metrics include: Accuracy , Recall, F1 Score,…

# Machine Learning - Performance Metrics

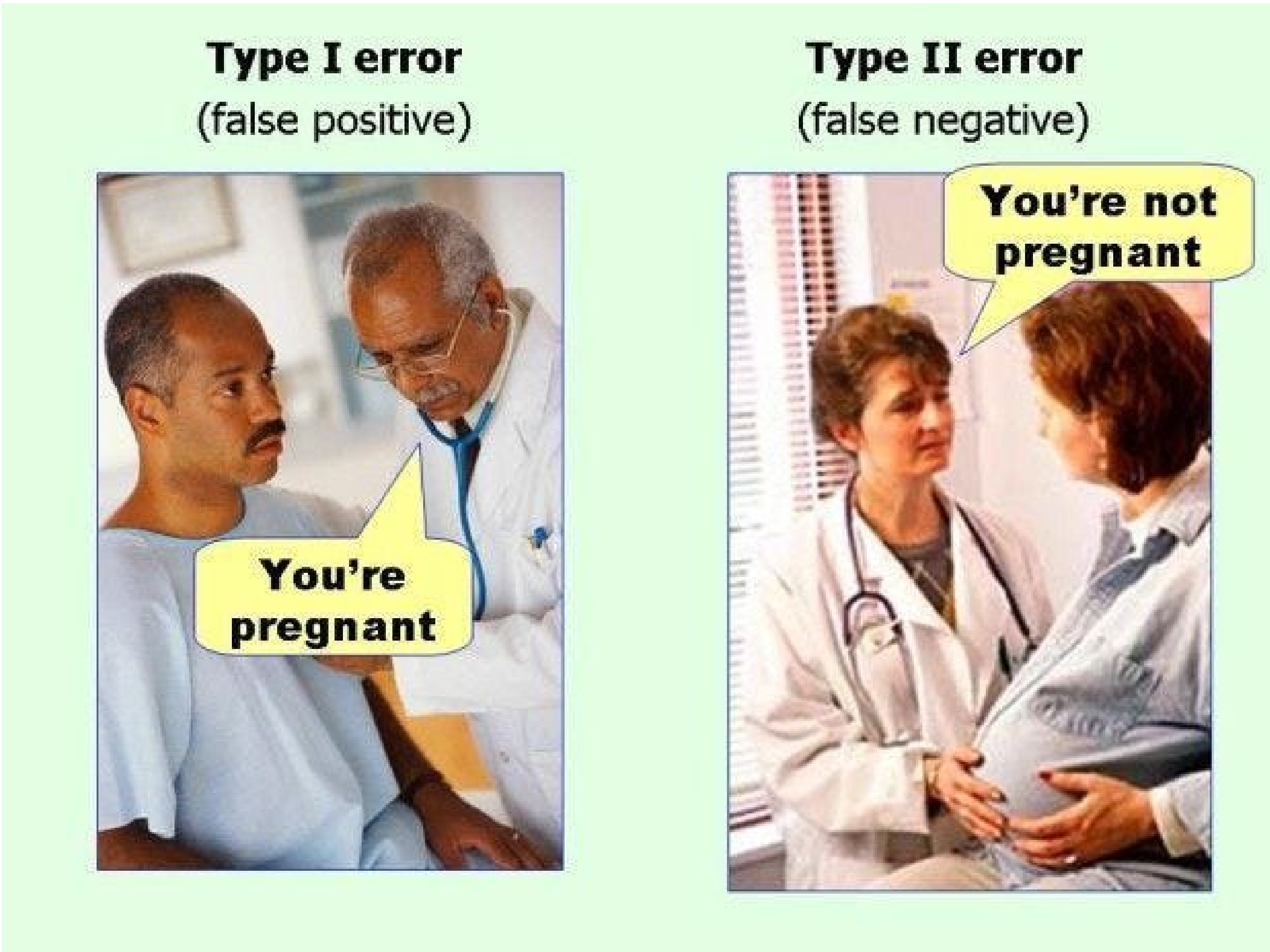
**Confusion Matrix** – A confusion matrix is a table that is used to evaluate the performance of a classification model. It shows the number of true positives, true negatives, false positives, and false negatives for each class in the dataset.

FN: Ham mail in spam folder.
FP: spam email in inbox.

## Confusion Matrix

| | Actually Positive (1) | Actually Negative (0) |
|---|---|---|
| Predicted Positive (1) | True Positives (TPs) | False Positives (FPs) |
| Predicted Negative (0) | False Negatives (FNs) | True Negatives (TNs) |

# Machine Learning - Performance Metrics



# Machine Learning - Performance Metrics

- True Positive(**TP**): the prediction outcome is true, and it is true in reality, also.
- True Negative(**TN**): the prediction outcome is false, and it is false in reality, also.
- False Positive(**FP**): prediction outcomes are true, but they are false in actuality.
- False Negative(**FN**): predictions are false, and they are true in actuality.

| | Actual **YES** | Actual **NO** |
|---|---|---|
| Predicted **YES** | True Positive | False Positive |
| Predicted **NO** | False Negative | True Negative |

# Machine Learning - Performance Metrics

- **Accuracy** – Accuracy is one of the most basic performance metrics and measures the proportion of correctly classified instances in the dataset.

- It is calculated as the number of correctly classified instances divided by the total number of instances in the dataset.

- **Precision** :It is calculated as the number of true positive instances divided by the sum of true positive and false positive instances.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

**Accuracy = (TP+TN)/(TP+TN+FP+FN)**

$$\boldsymbol{Precision = \frac{TP}{(TP+FP)}}$$

# Machine Learning - Performance Metrics

- **Recall "**sensitivity**"**: It is calculated as the number of true positive instances divided by the sum of true positive and false negative instances.

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

# Machine Learning - Performance Metrics

- **F1 Score** – F1 score is the harmonic mean of precision and recall. It is a balanced measure that takes into account both precision and recall.
- It is calculated as 2 * (precision × recall) / (precision + recall).

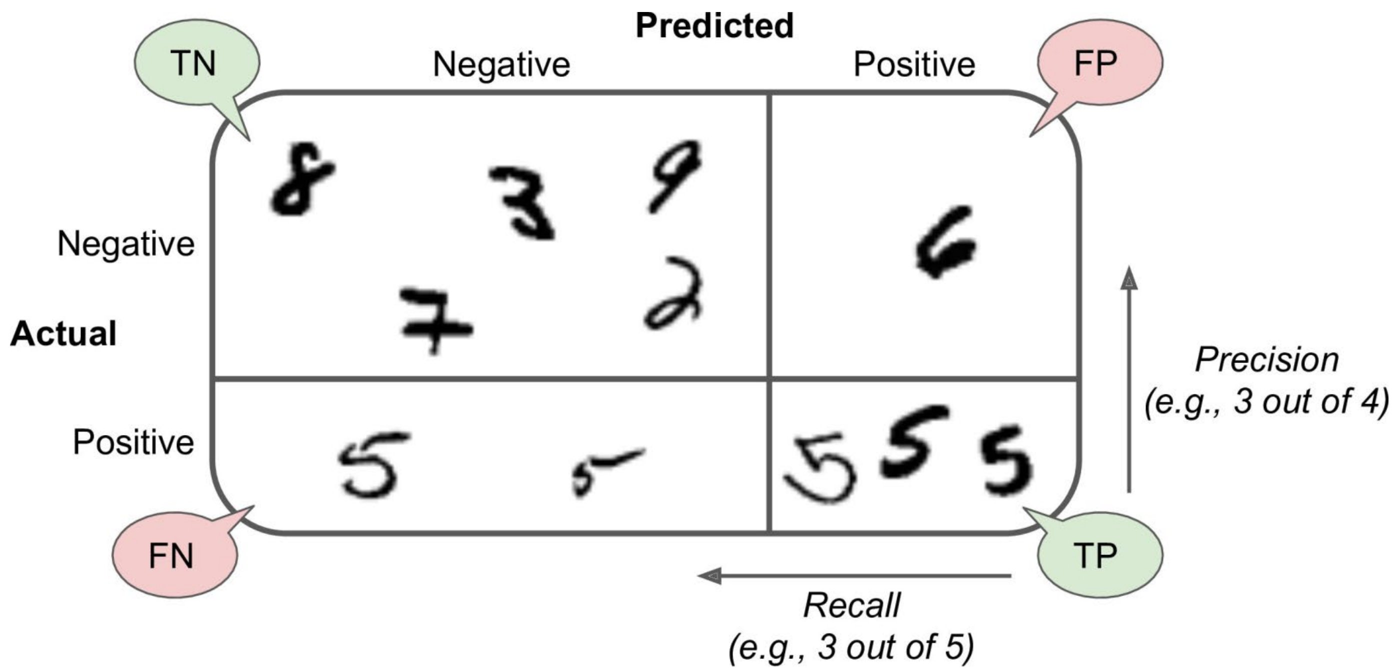
- the classifier will only get a high F1 score if both recall and precision are high.

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

To compute the $F_1$ score, simply call the `f1_score()` function:

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

# Machine Learning - Performance Metrics




*Equation 3-1. Precision*

$$\text{precision} = \frac{TP}{TP + FP}$$

*Equation 3-2. Recall*

$$\text{recall} = \frac{TP}{TP + FN}$$

# Machine Learning - Performance Metrics

Truth

**Prediction** (based on model)

| | Positive | Negative |
|---|---|---|
| Positive | True Positive (TP) | False Positive (FP) |
| Negative | False Negative (FN) | True Negative (TN) |

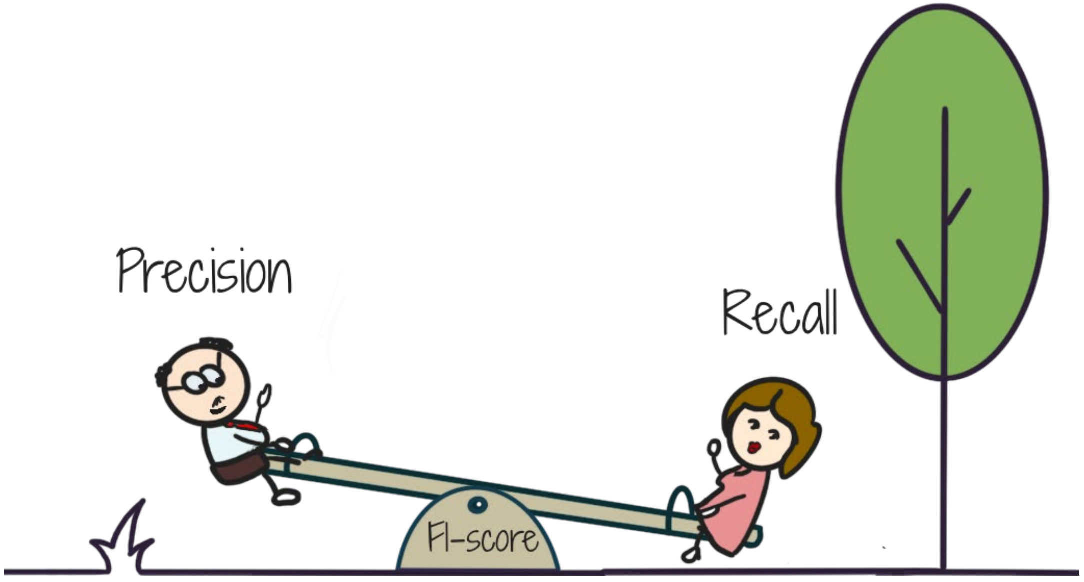
**Precision = TP / (TP+FP)**

**Sensitivity/ Recall = TP / (TP+FN)**

**Specificity = TN / (FP+TN)**

**Accuracy = (TP+TN)/(TP+TN+FP+FN)**

# Precision/Recall Tradeoff

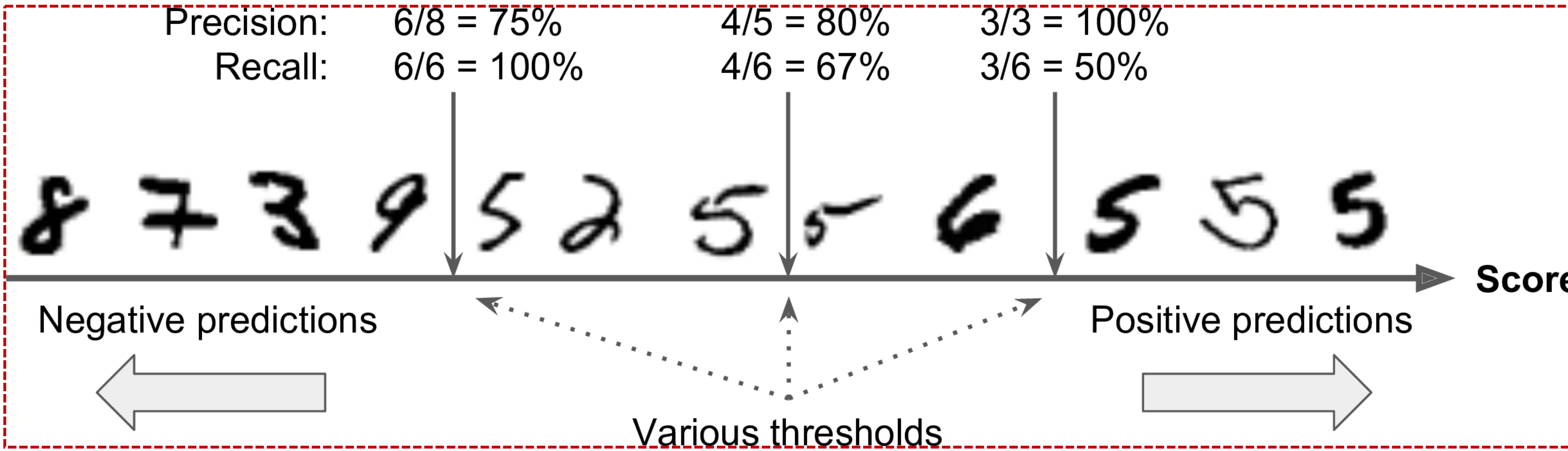


# Machine Learning - Precision/Recall Tradeoff

- Unfortunately, you can't have it both ways: increasing precision reduces recall, and vice versa. This is called the **precision/recall tradeoff**.

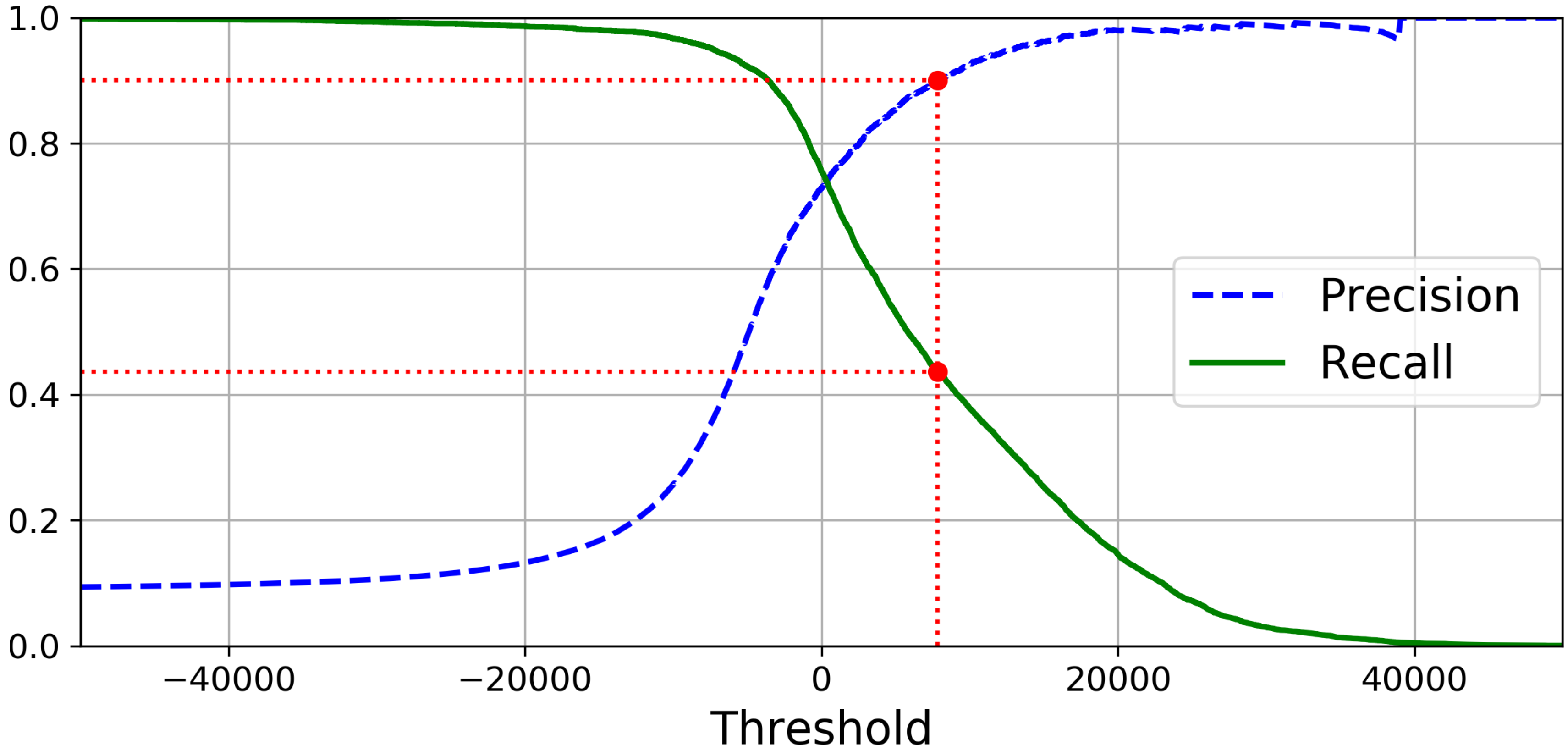
- To understand this **tradeoff**, let's look at how the SGDClassifier makes its classification decisions. For each instance, it computes a score based on **a decision function**, and if that score is greater than a **threshold**, it assigns the instance to the positive class, or else it assigns it to the negative class.

# Machine Learning - Precision/Recall Tradeoff




✓ Raising threshold increases precision, but decrease recall.


# Machine Learning - Precision/Recall Tradeoff




- If someone says "let's reach 99% precision," you should ask, "at what recall?"
- *Determine threshold from the curve*.

# The ROC Curve

## Receiver Operating Characteristic

# Machine Learning - ROC Score

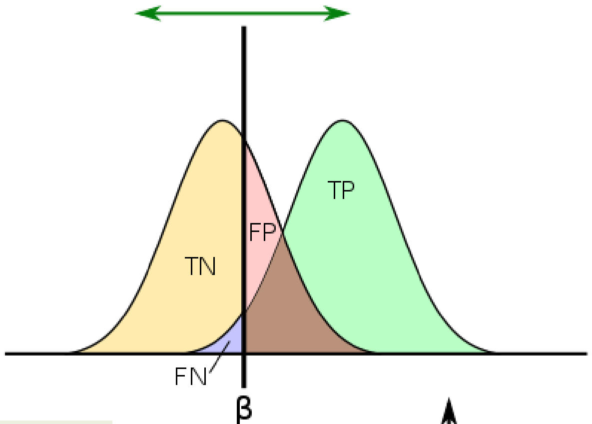
- The **receiver operating characteristic** (**ROC**) curve is another common tool used with binary classifiers.

- It is very similar to the precision/recall curve, but instead of plotting precision versus recall, the ROC curve plots the true positive rate (another name for recall) against the false positive rate.

- The FPR is the ratio of negative instances that are incorrectly classified as positive. It is **equal to one minus** the true negative rate, which is the ratio of negative instances that are correctly classified as negative.

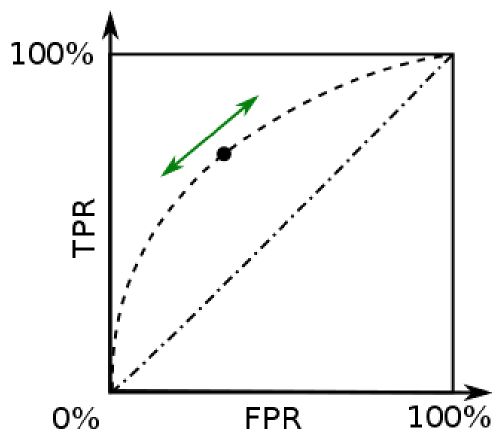
- The TNR is also called specificity. Hence the ROC curve plots sensitivity (recall) versus 1 – specificity

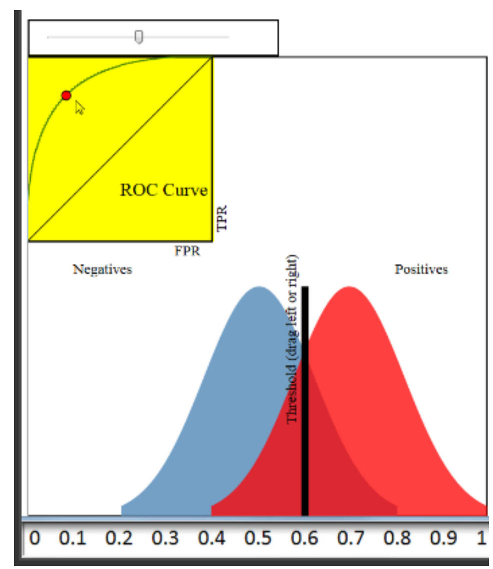
# Machine Learning - ROC Score




| | | |
|---|---|---|
| TP | FP | ROC1 |
| FN | TN | ROC2 |

histogram



# Machine Learning - ROC Score

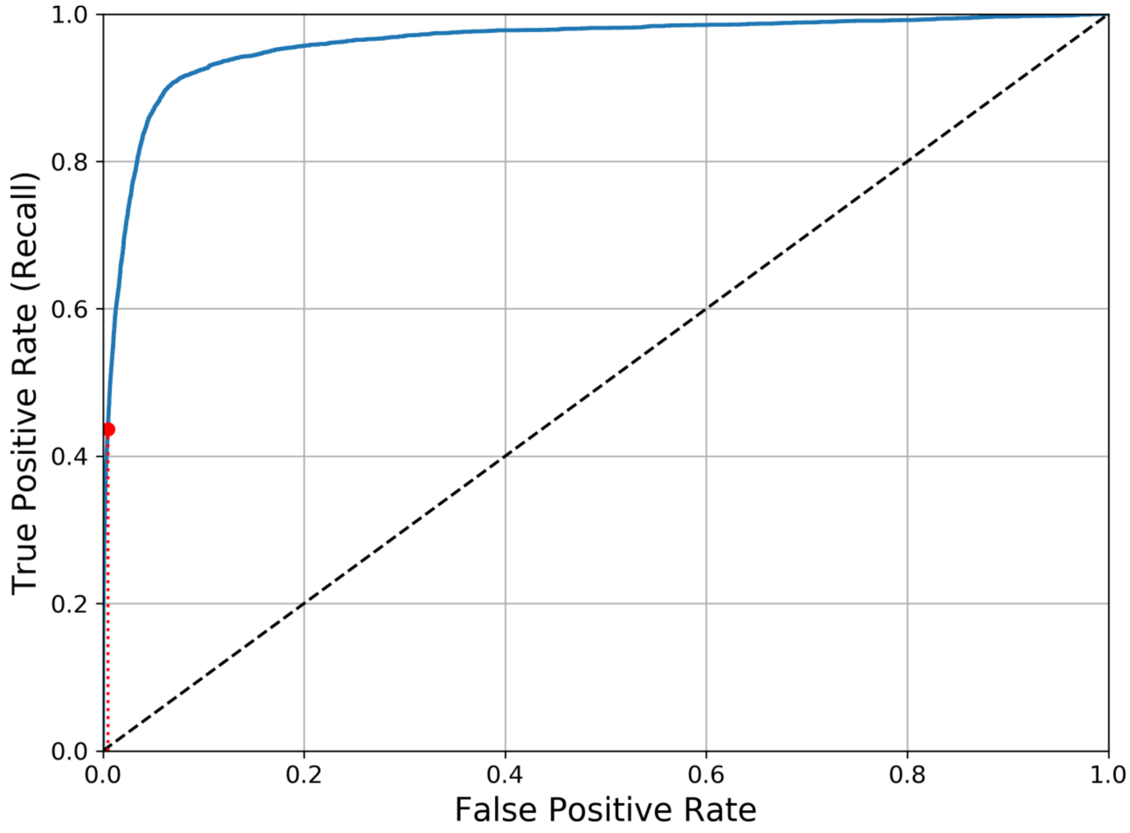
What is the ROC Curve?

- The ROC curve is a graphical representation of the performance of a binary classification model at different threshold values.

- It plots the true positive rate (**TPR**) on the y-axis and the false positive rate (**FPR**) on the x-axis.

- The **TPR** is the proportion of actual positive cases that are correctly identified by the model, while the **FPR** is the proportion of actual negative cases that are incorrectly classified as positive by the model.

# Machine Learning - ROC Score



# Machine Learning - ROC AUC Score

- One way to compare classifiers is to measure the area under the curve (**AUC**). A perfect classifier will have a **ROC AUC** equal to 1, whereas a purely random classifier will have a **ROC AUC** equal to 0.5.

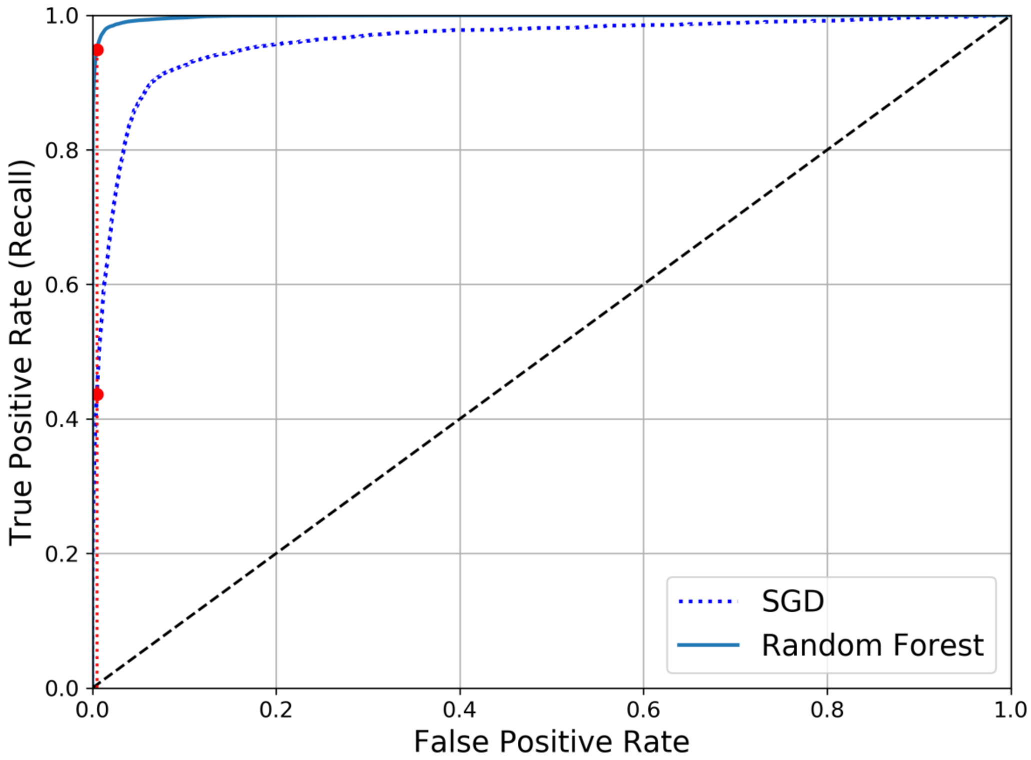
- Scikit-Learn provides a function to compute the ROC AUC:

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.9611778893101814
```

**Why is the AUC-ROC Curve Important?**

The AUC-ROC curve is an important performance metric in machine learning because it provides a comprehensive measure of a model's ability to recognize **positive and negative cases**.

# Machine Learning - ROC AUC Score




- Let’s train a RandomForestClassifier and compare its ROC curve and ROC AUC score to the SGDClassifier.
- the RandomForestClassifier’s ROC curve looks much **better** than the SGDClassifier’s.

# Practical Part

# Practical part

- Calculate all the performance measurements in Python
- Implement the AUC ROC Curve in Python


Thank You !
Any questions?