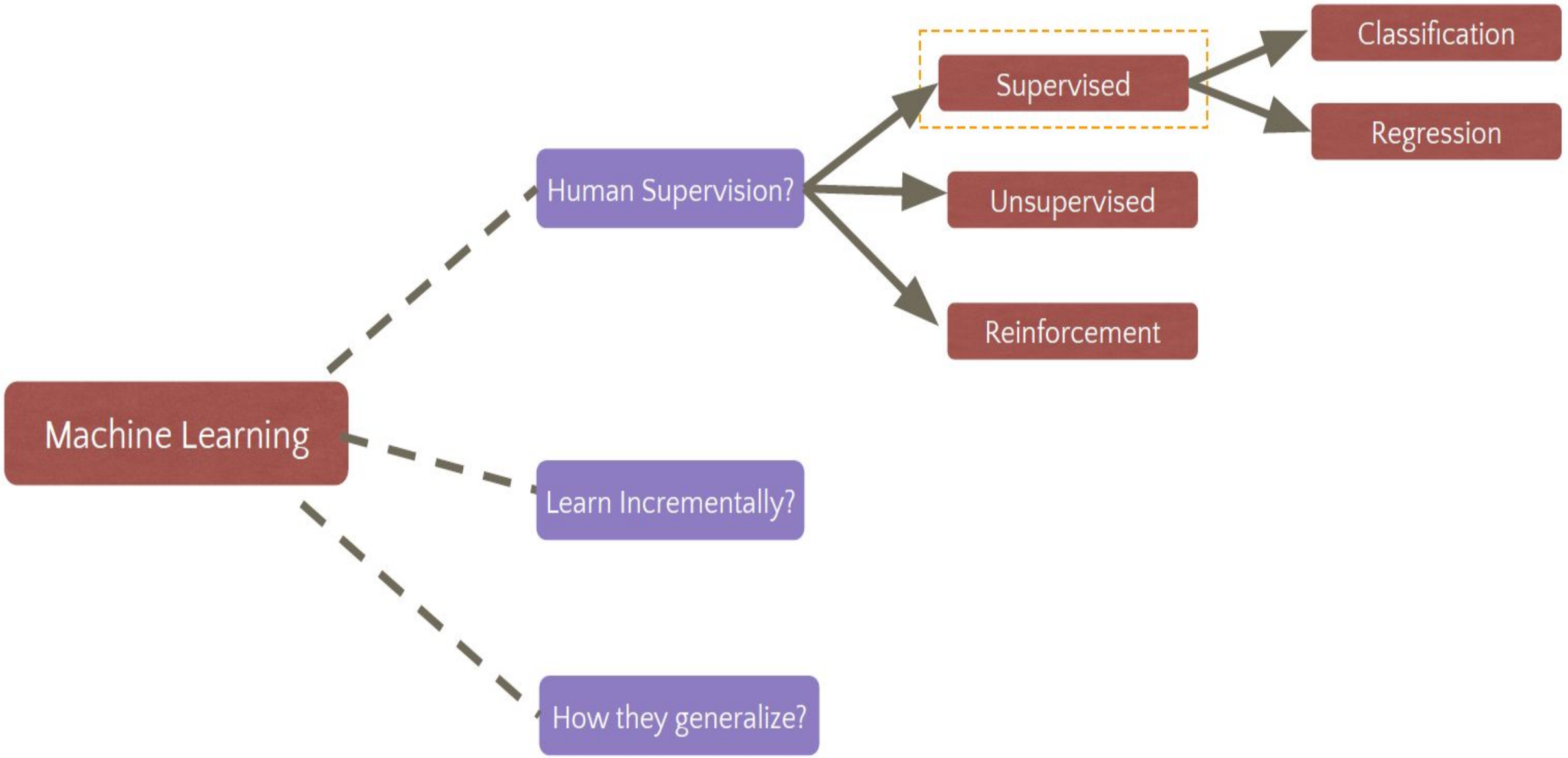
UNIT -2
CLASSIFICATION

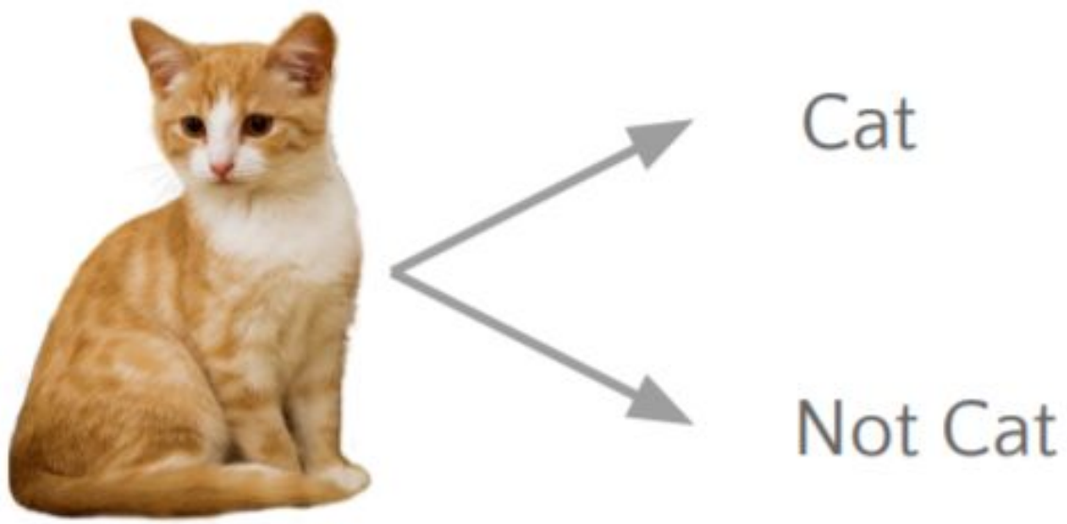
- Introduction
- Build Model with MNIST Dataset
- Training a Binary Classifier
- Performance Measures
  - Cross Validation
  - Confusion Matrix
  - Precision & Recall
  - Precision/Recall Trade-off
  - ROC Curve
- Types of Classification –
  - Multiclass Classification
- Error Analysis


Machine Learning
Human Supervision?
Supervised
Classification
Regression
Unsupervised
Reinforcement
Learn Incrementally?
How they generalize?

What is Classification ?

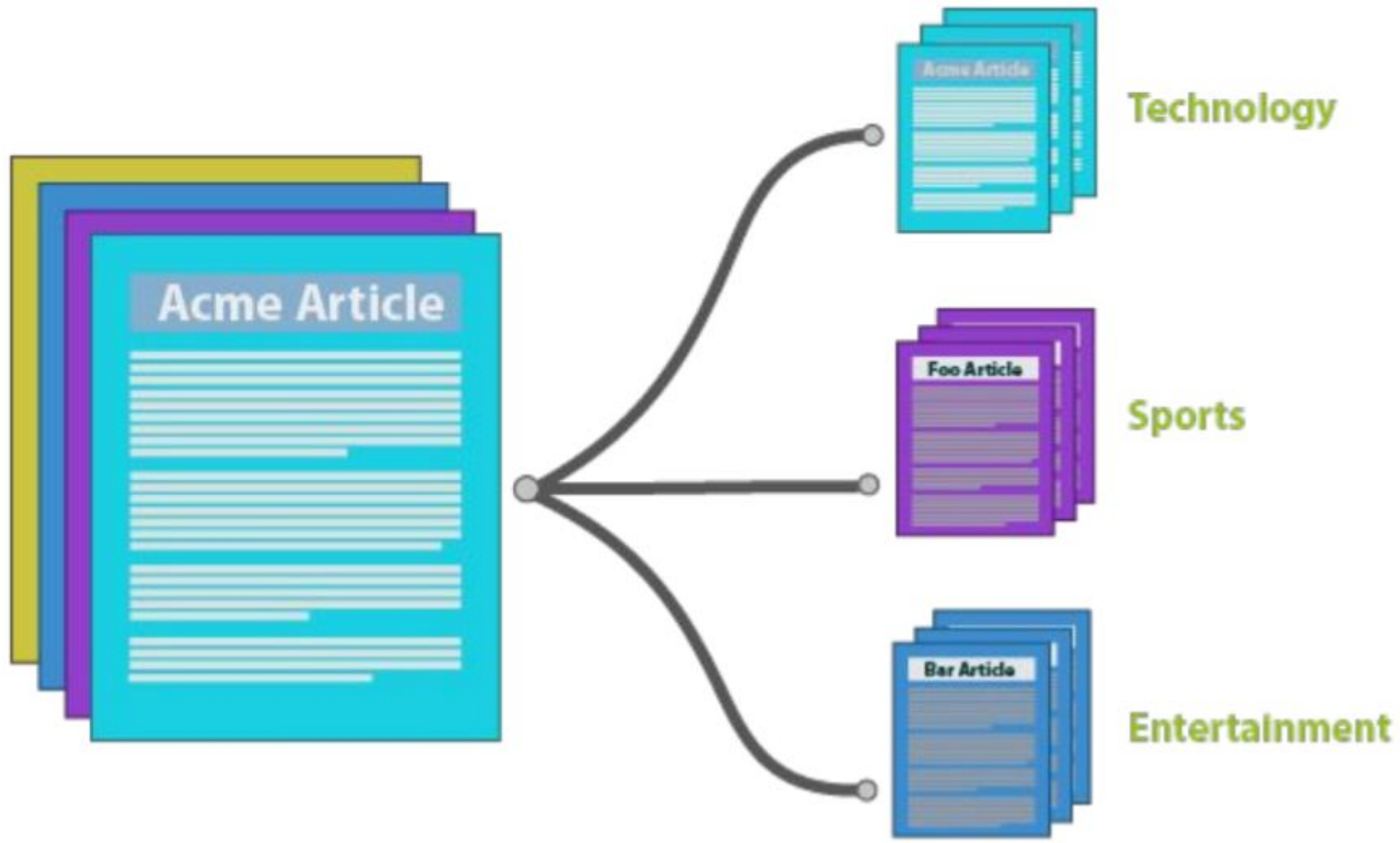
Classification is the type of supervised learning

- Identify the class to which a new observation belongs to.



Cat or Not Cat Classifier

What is Classification ?



News Article Classifier

What is Classification ?

Classification is the type of supervised learning

- Identify the class to which a new observation belongs to.

Examples of Classification

- Classifying emails as spam or not spam
- Classifying flowers of a particular species like the Iris Dataset
- Classifying a credit card transaction as fraudulent or not
- Face recognition

There are 2 types of Classification:

- Binary or binomial
- Multi-Class

So how do we train the model in classification?

Labeled Images for Building Model

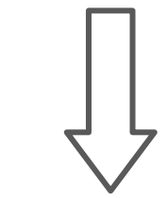
Apple

Banana

Apple

Kiwi

Grapes



Split images into training and test set

Apple

Banana

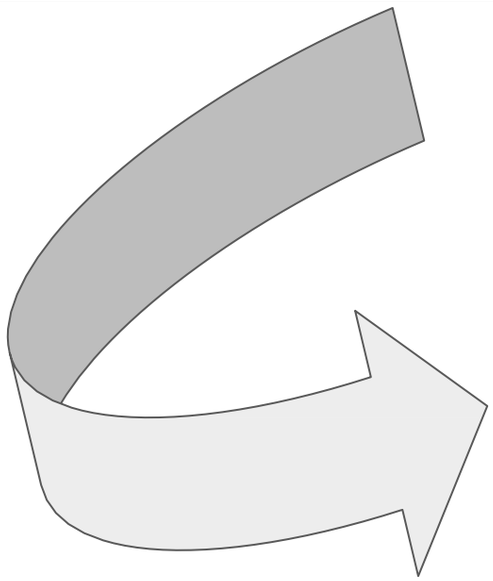
Grapes

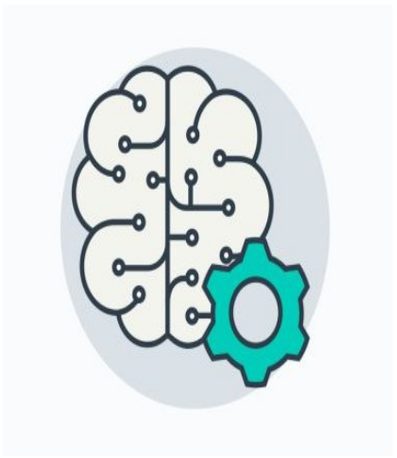
Kiwi

Apple

Training Set - 80% Data

Test Set - 20% Data





Using training set we build the model

Labeled Images for Building Model

Apple

Banana

Apple

Kiwi

Grapes

Split images into training and test set

Apple

Banana

Grapes

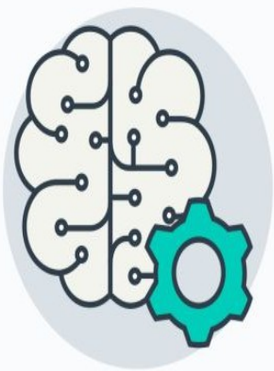
Kiwi

Training Set - 80% Data

Apple

Test Set - 20% Data

Using test set we evaluate performance of model



Predictions

Apple

Banana

Images without Label

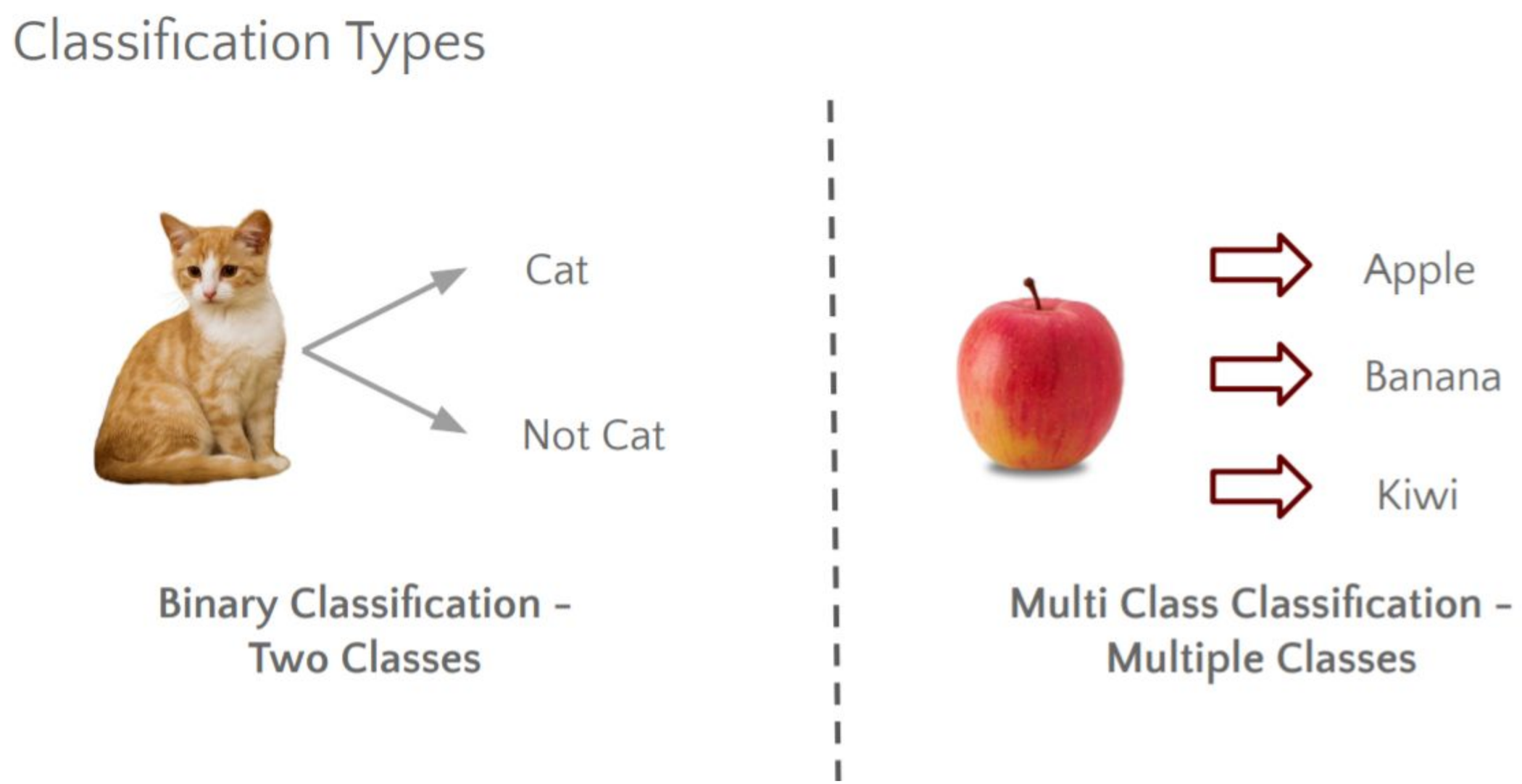
Model

Model's Predictions

Fruits Classifier

There are 2 types of Classification:

- Binary or binomial
- Multi-Class

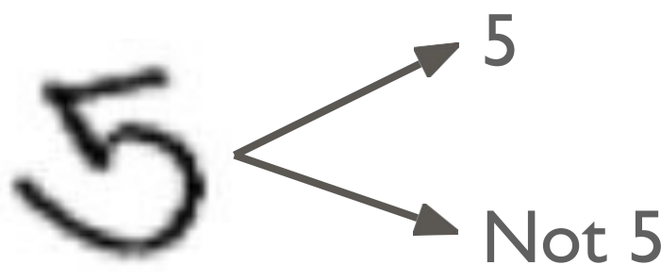


Goal - Classification

The goal of this session is to classify handwritten digits

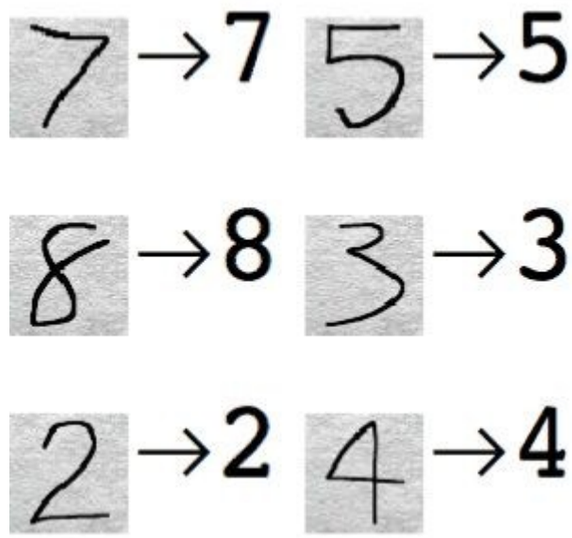
Binary and Multiclass Classification

Binary Classification



Classification is done between **2 classes**

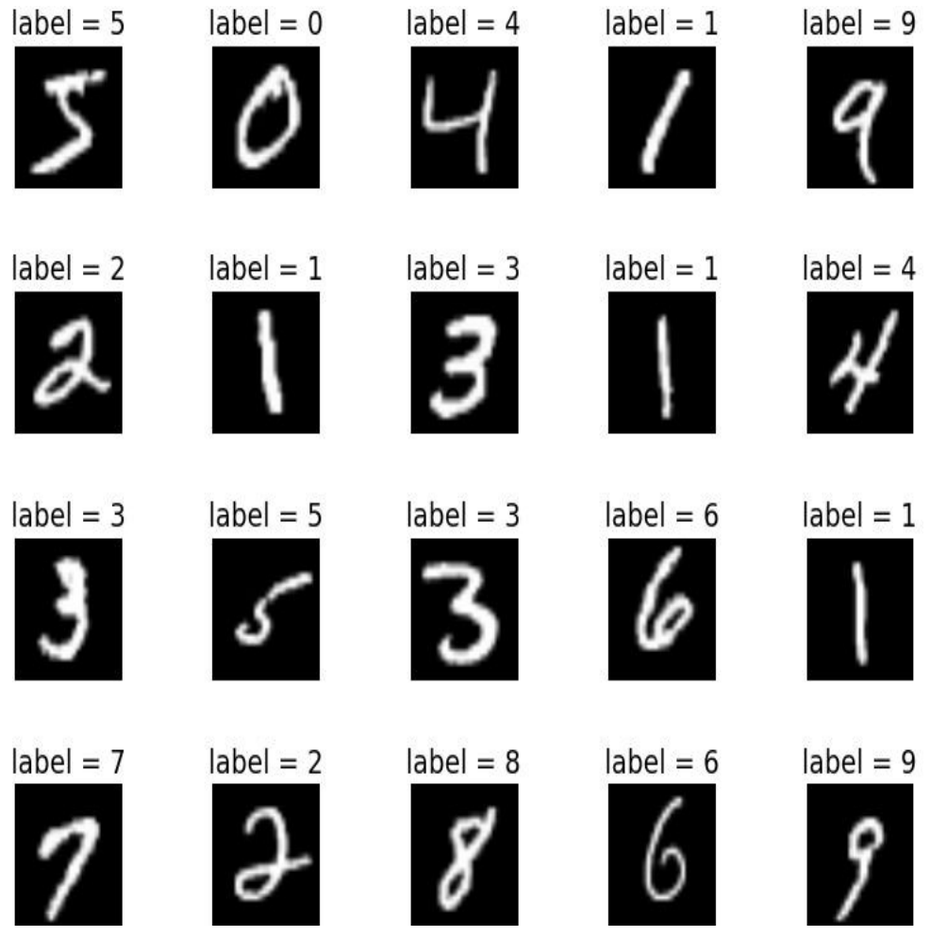
Multiclass Classification



Classification is done between **multiple classes**

Handwritten Digits Classifier - Dataset

- In this problem, we will use MNIST dataset
  - Set of 70,000 small images
  - Each image is grayscale (black & white)
  - 28 by 28 pixels

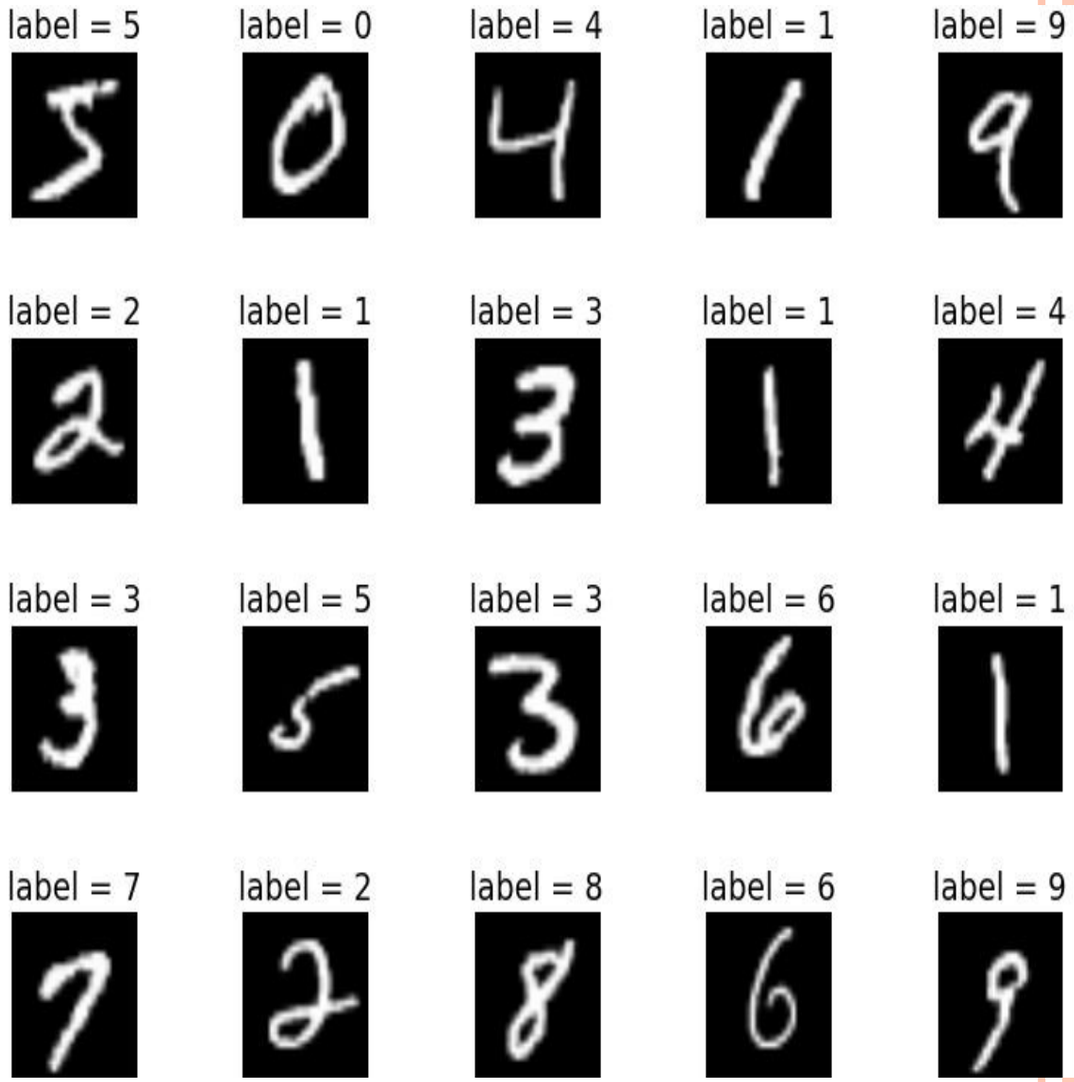


MNIST Dataset (Modified National Institute of Standards and Technology)

Handwritten Digits Classifier - Dataset

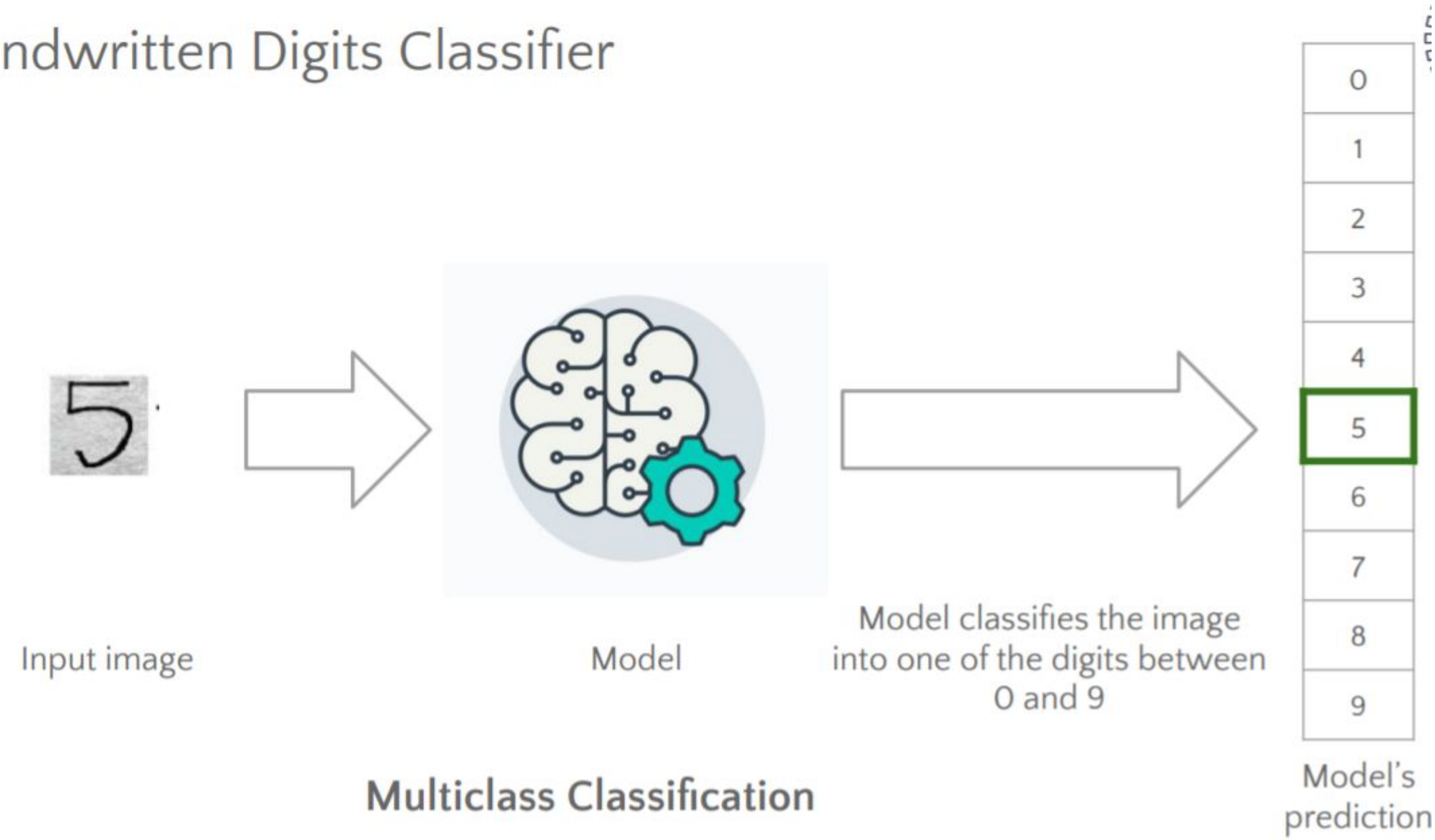
- MNIST dataset is also called
  - “Hello World” of Machine Learning
- We have to build a model

  using this labelled dataset



Every Image has Label Associated

Handwritten Digits Classifier



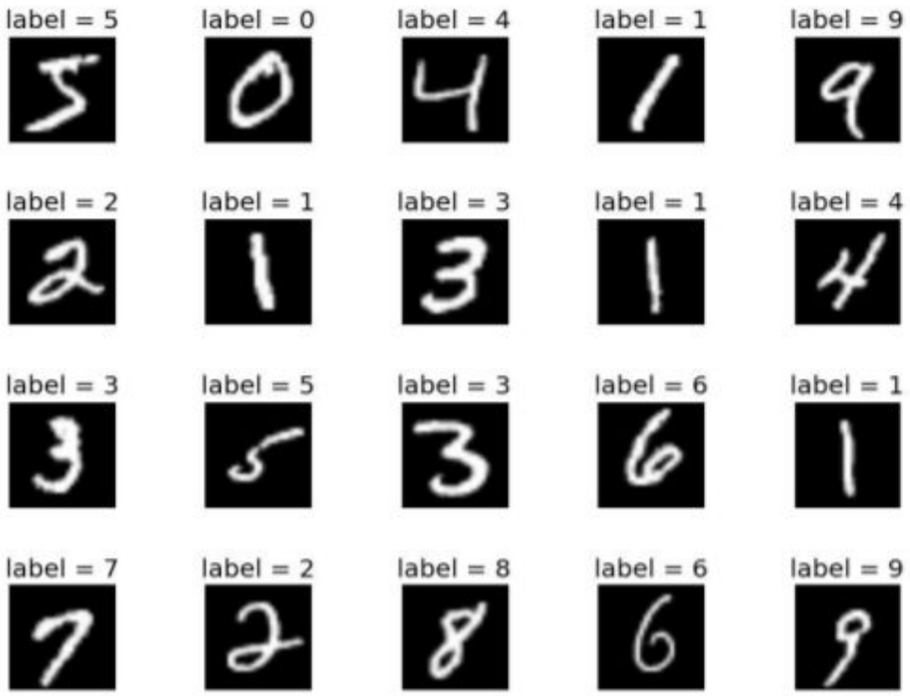
Multiclass Classification

Dataset

```
>>> 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'])
```

- Datasets loaded generally have a similar dictionary structure:
  - 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

Handwritten Digits Classifier – Training Process



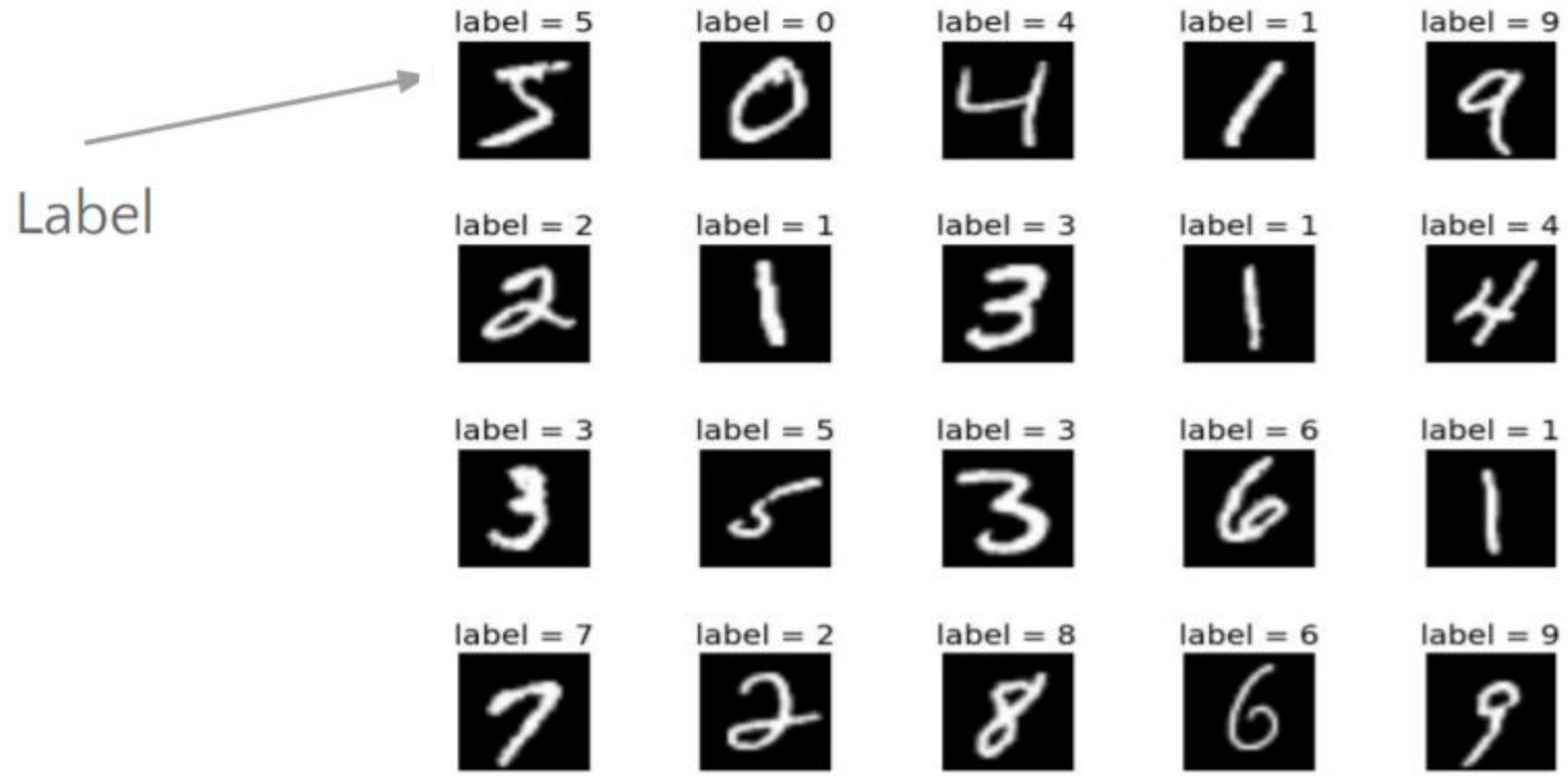
???

Tabular form – Rows and Columns

???

Instances, Features and Labels?

Handwritten Digits Classifier – Training Process



Each image is an instance

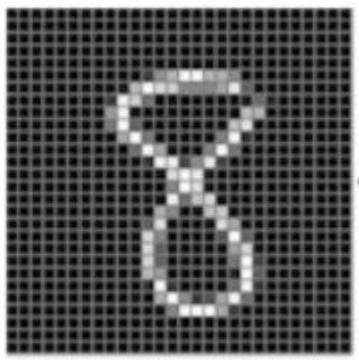
Features?

Dataset

Let's look at these arrays:

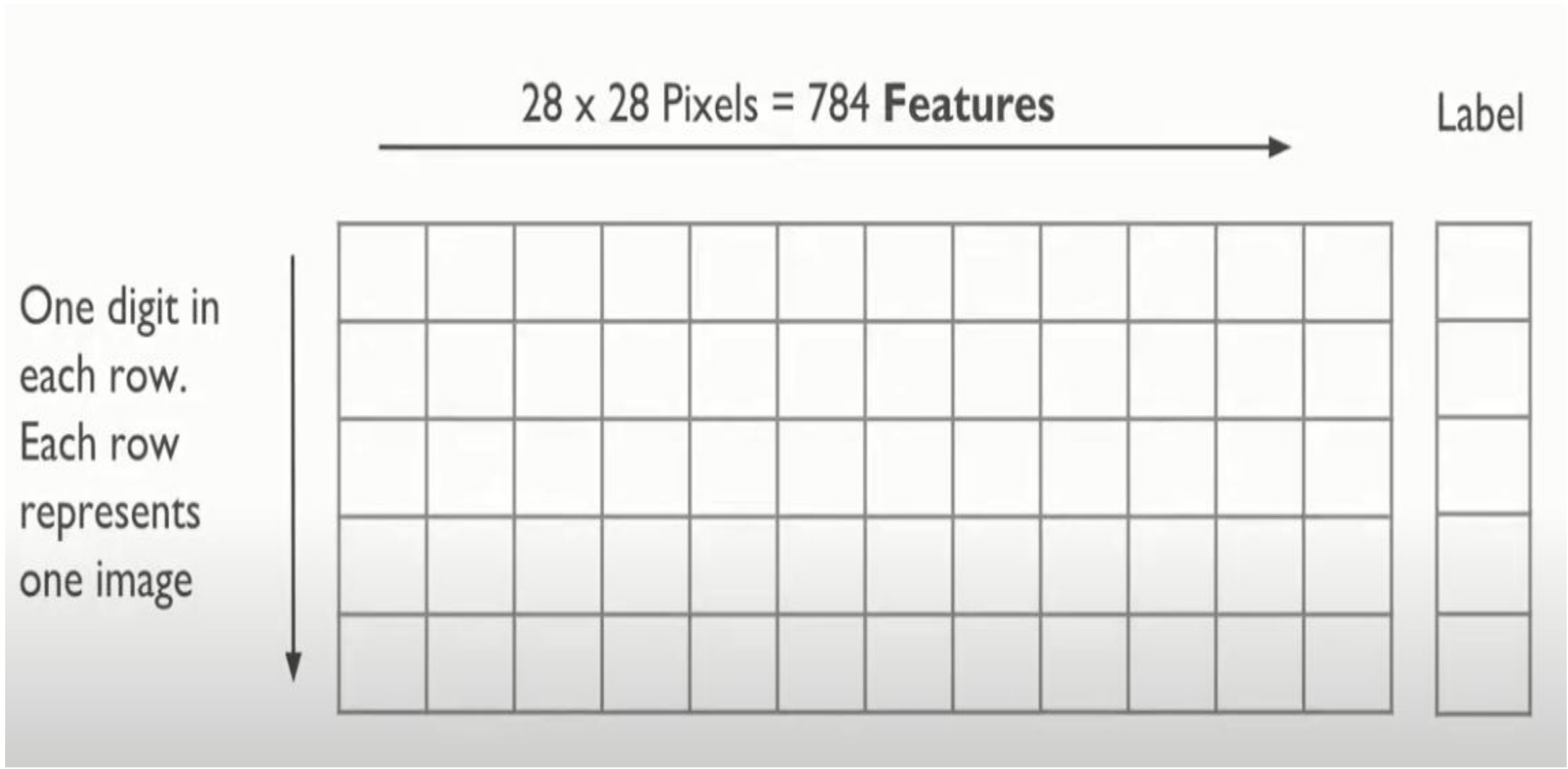
```
>>> X, y = mnist["data"], mnist["target"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

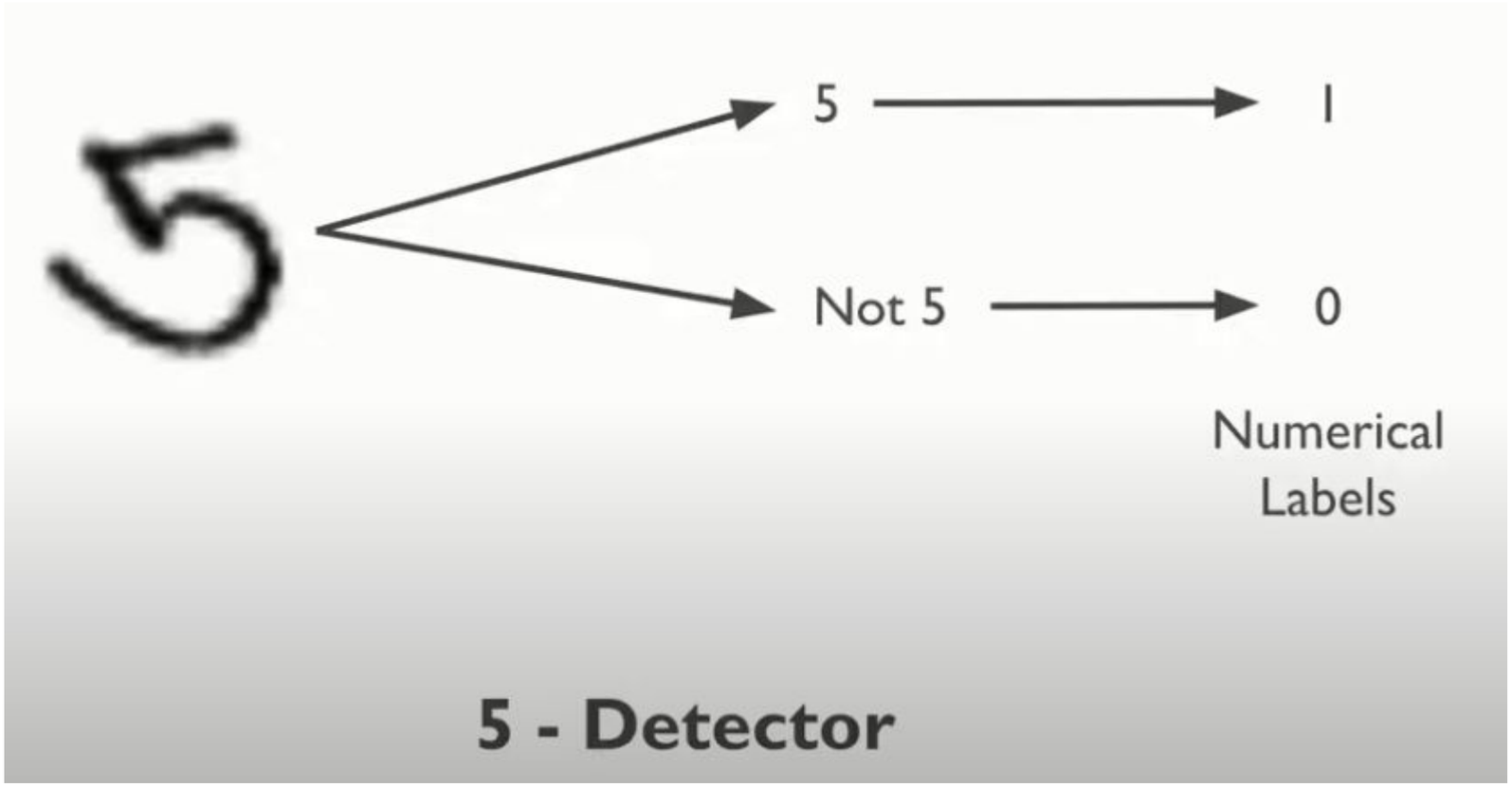
- Each image
  - 28 X 28 pixels
  - 784 features
- There are 70000 such images making the dataset dimension
  - 70000 X 784
- Each feature represents one pixel's intensity, from 0 to 255.



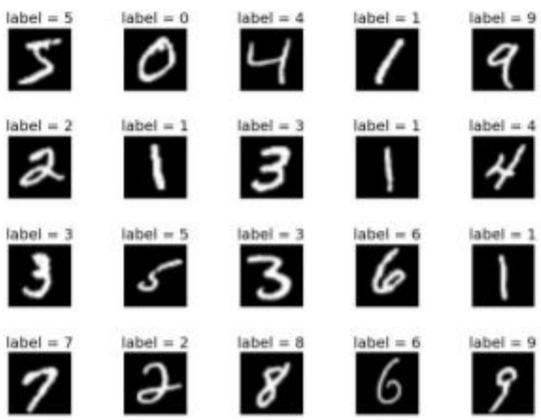
28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


28 x 28 Pixels = 784 **Features**
Label
One digit in each row.
Each row represents one image


5
5
1
Not 5
0
Numerical
Labels
5 - Detector

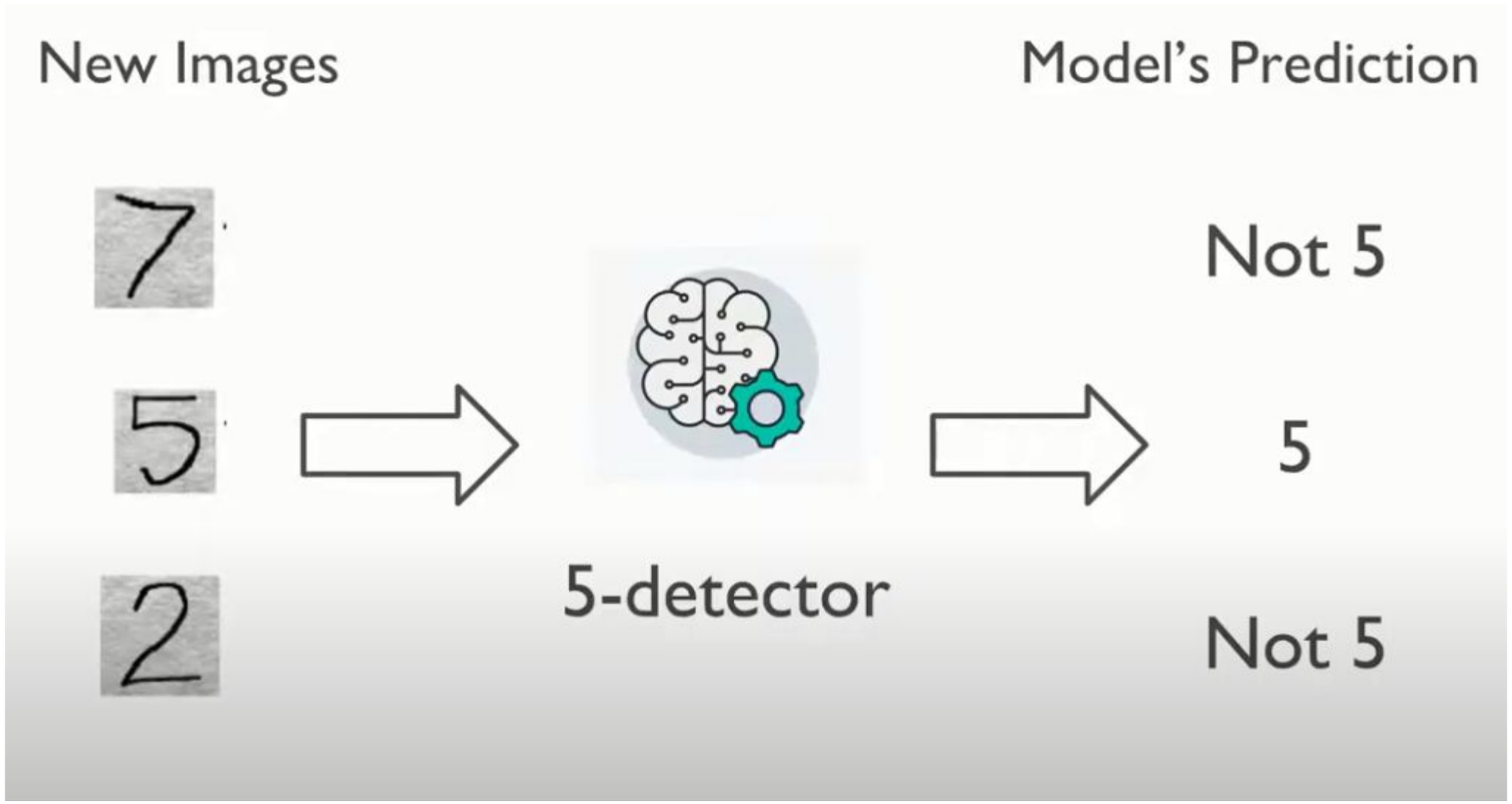
Handwritten Digits Classifier – Training Process



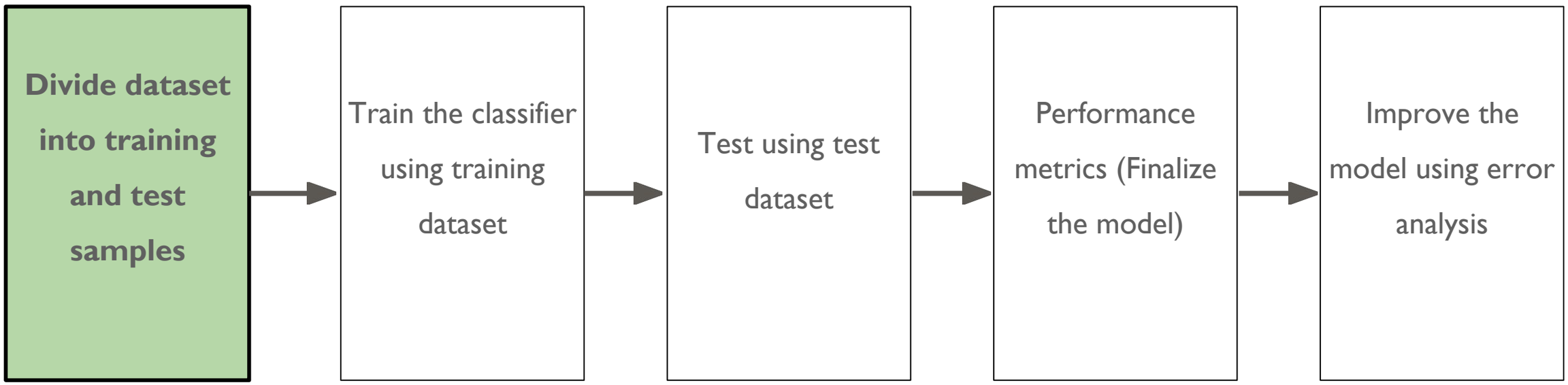
Input data in tabular form – Includes both features and labels

Algorithm

Model


New Images
Model's Prediction
Not 5
5
5-detector
Not 5

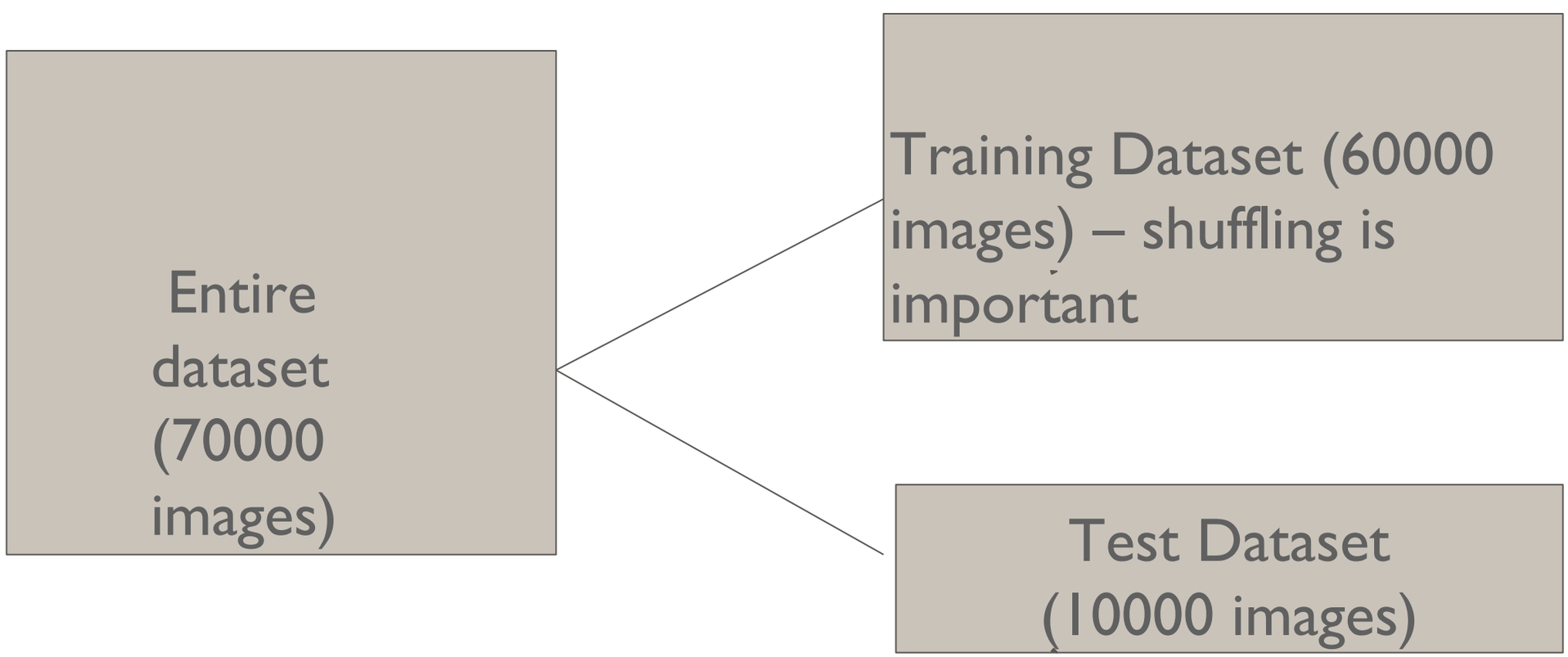
Handwritten Digits Classifier – Training Process – Steps



Training and Test dataset

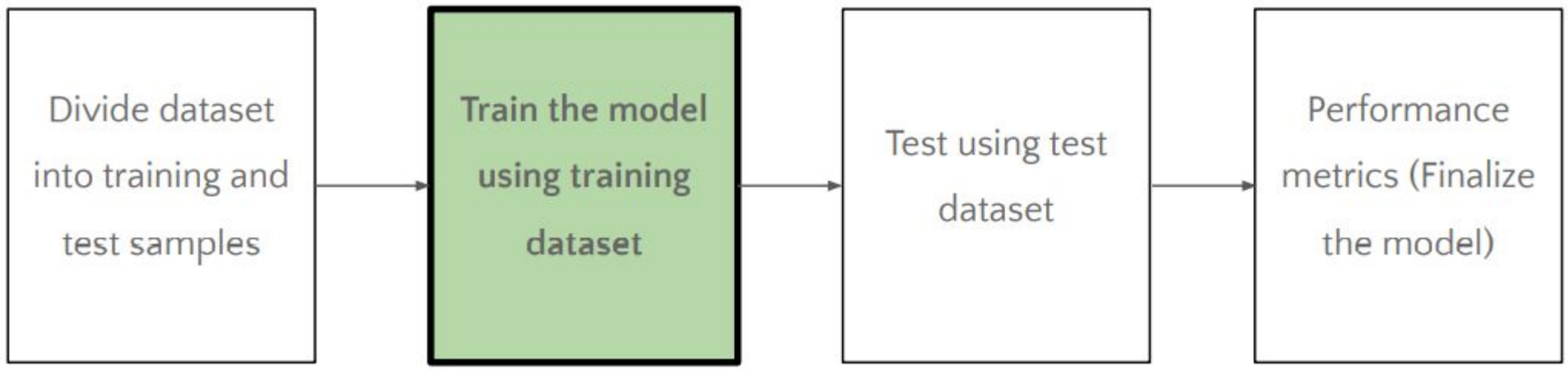
- We split the data into
  - Training set - Contains 60,000 out of 70,000 samples
  - Test set - Contains 10,000 out of 70,000 samples

- We train the model on training set and evaluate the performance of the model on test set




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

Handwritten Digits Classifier – Training Process – Steps

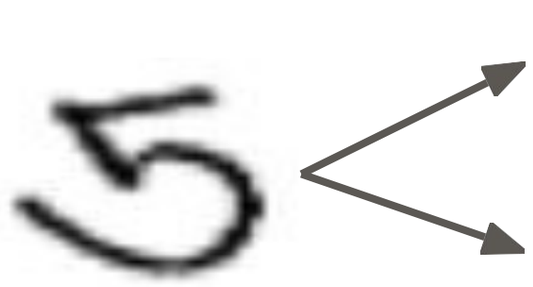


Training a Binary Classifier

- What is a Binary Classification?
  - Binary or binomial classification is the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule.

Lets try to identify one digit example, the number 5. This "5-detector" will be an example of a binary classifier.

Example:



Input:
Image

Output:
Classification

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)
```

Stochastic Gradient Descent (SGD) Classifier

Classifier used: Stochastic Gradient Descent (SGD) Classifier using Scikit-Learn's SGDClassifier class.

Training a Binary Classifier using SGD

- Stochastic Gradient Descent (SGD) Classifier
  - Capable of handling large datasets
  - Deals with training instances independently
  - Well suited for online training

Let's create an SGDClassifier and train it on the whole training set:

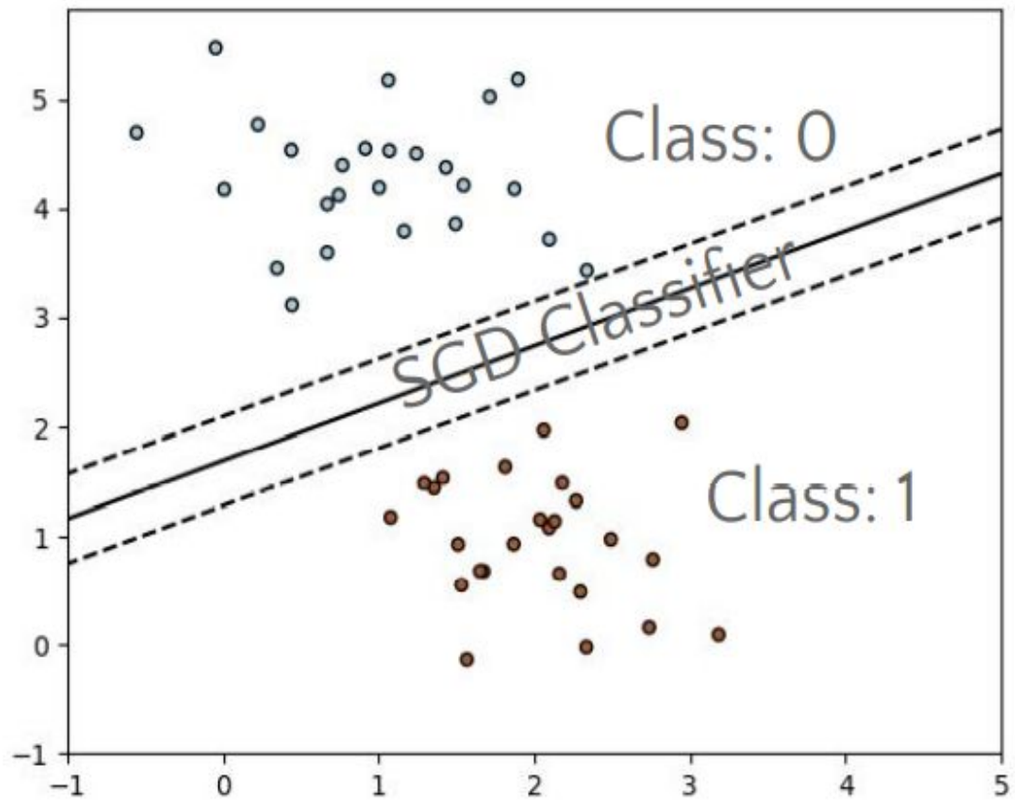
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

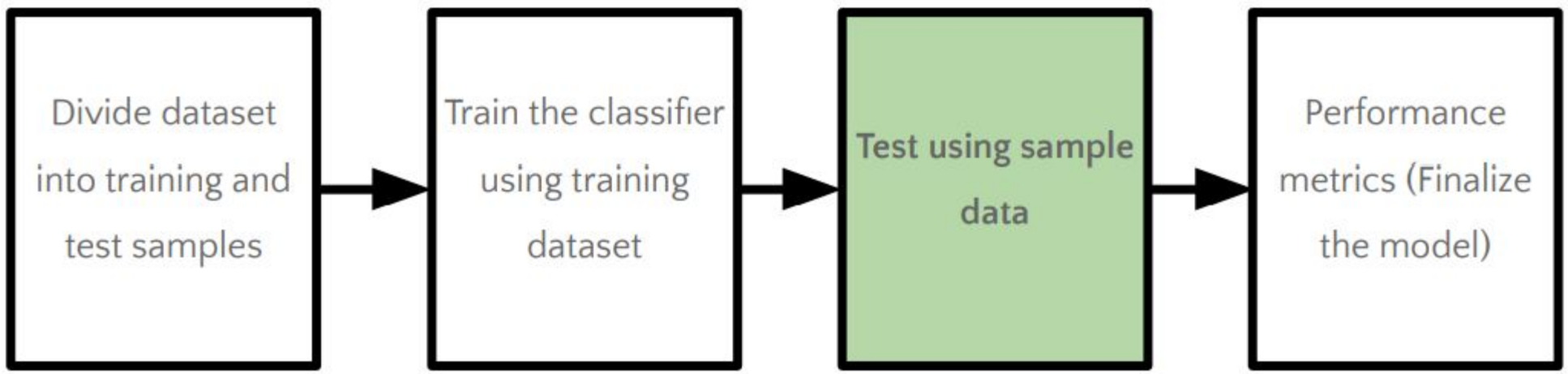
SGD - Stochastic Gradient Descent Classifier

For the two-dimensional (2 features) training dataset,

- Trying to estimate the coefficients of the line which can be
- Best-fitted dividing the two categories



Steps

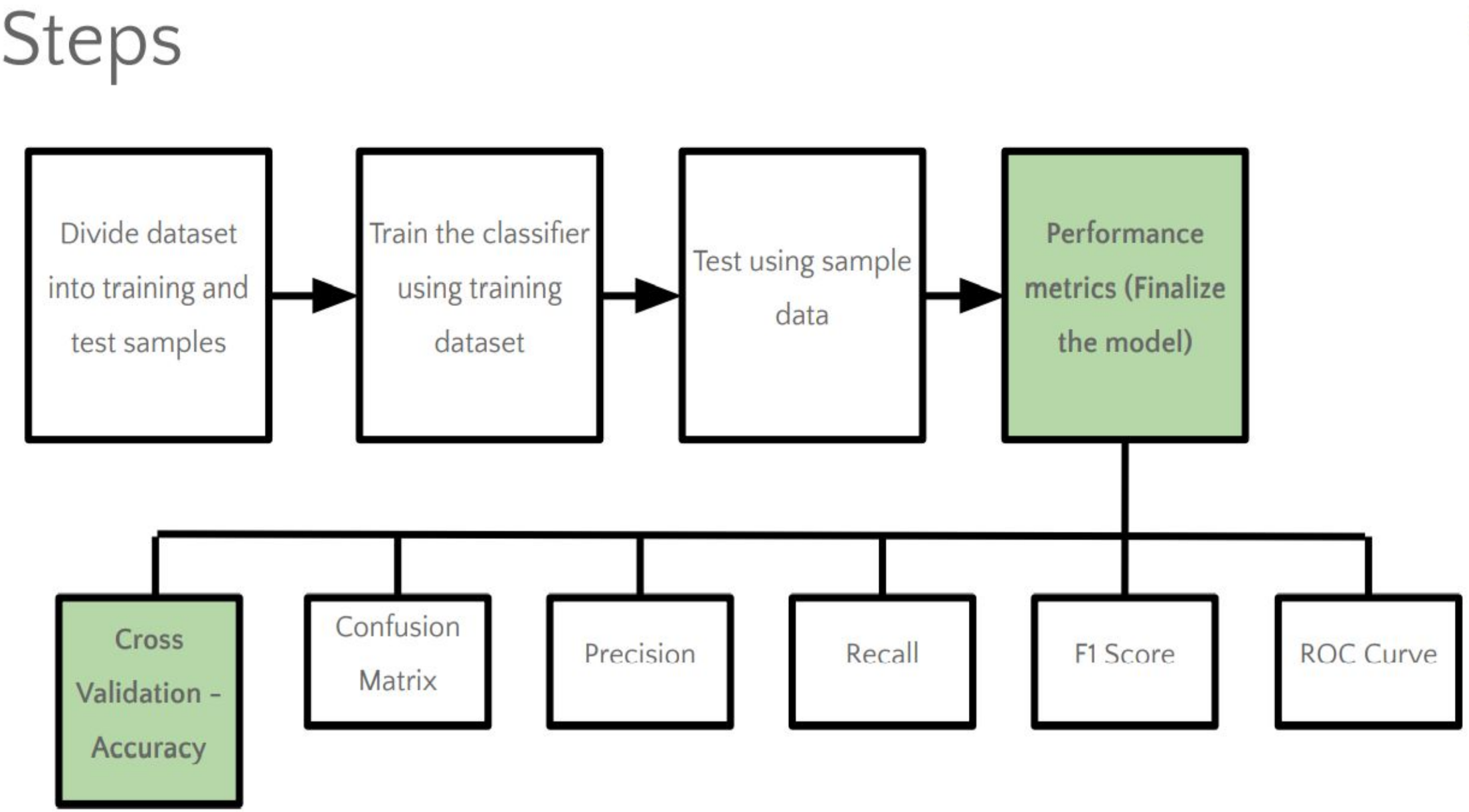

Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data
Performance metrics (Finalize the model)

Testing SGD Classifier in Scikit Learn

```
>>> some_digit = X[0] # Taking the 11th image
>>>
sgd_clf.predict([some_digit])
array([True])
```

The classifier guesses that this image represents a 5 (True).

Performance measure - Methods



Performance measure - Cross Validation

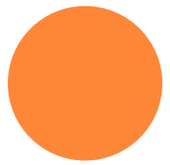
What is cross-validation?

- It involves splitting the training set into K distinct subsets called folds, then training and evaluating the model K times, picking a different fold for evaluation every time and training on the other K-1 folds.

- The result is an array containing K evaluation scores.

cross_val_score

cross_val_score() function in scikit-learn can be used to perform cross validation.



Performance measure - Cross Validation

Here k = 10

Performing Cross validation in Scikit learn

```
>>> from sklearn.model_selection import
cross_val_score

>>> cross_val_score(sgd_clf, X_train, y_train_5,
cv=3, scoring="accuracy")
```

Scoring parameter

Classifier object

Training data

Labels

No.of folds

(Here, scoring parameter is accuracy)

Performance Measures – Cross Validation

```
>>> array([0.95035, 0.96035, 0.9604 ])
```

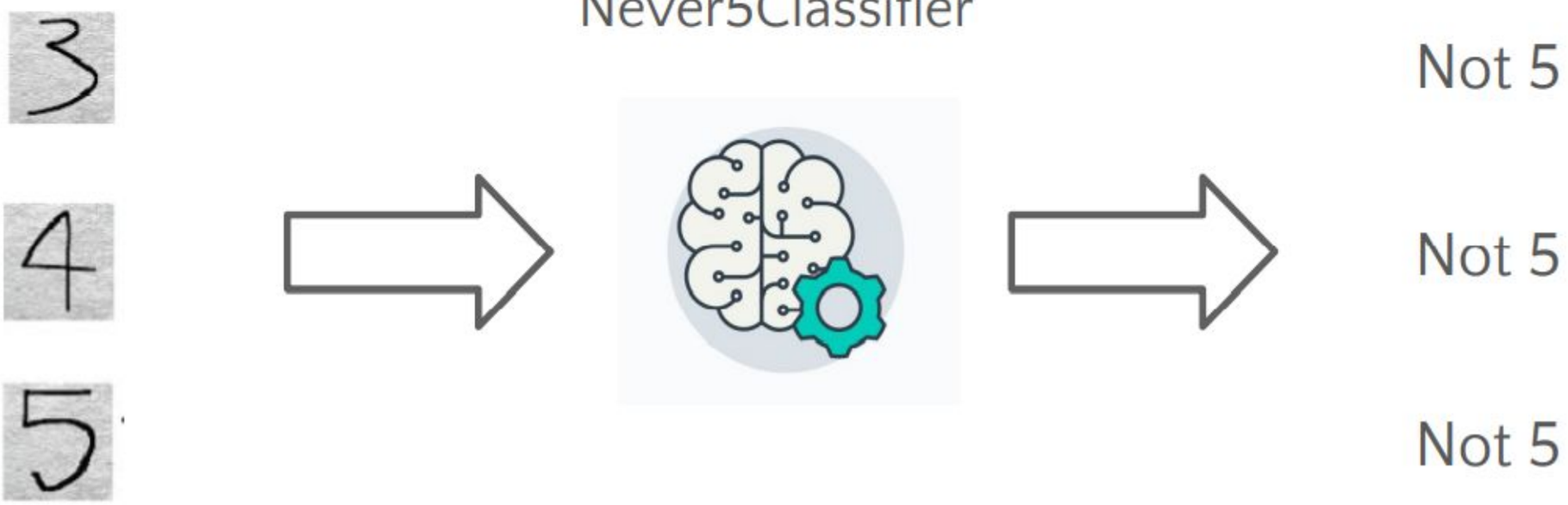
- The resulting accuracy is above 95 % for each of the folds

Is the accuracy of 95% good enough?

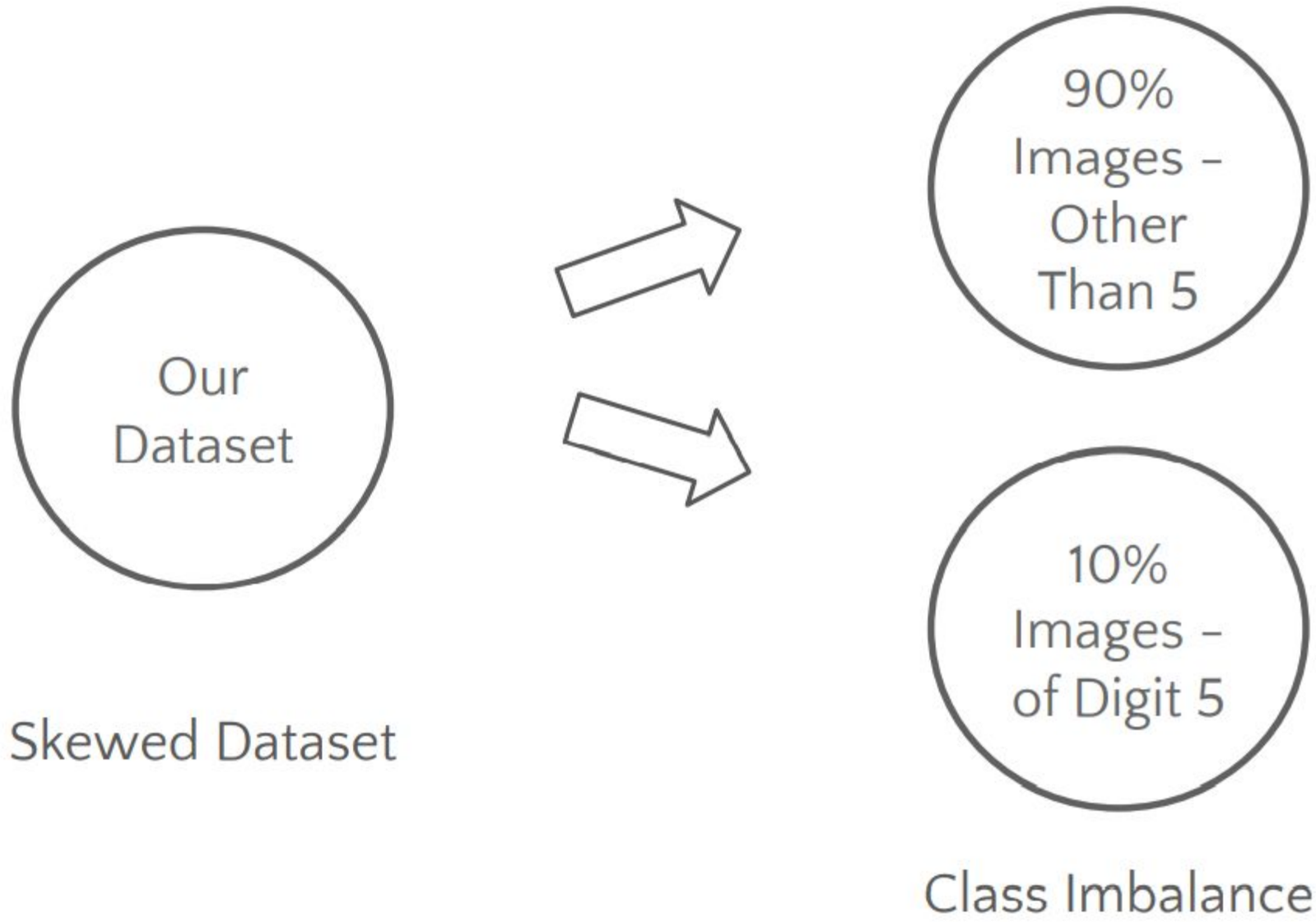
Let's see

Performance Measures – Cross Validation

For Now Let's Build Another Classifier – Never5Classifier – Which classifies every image as "Not 5"



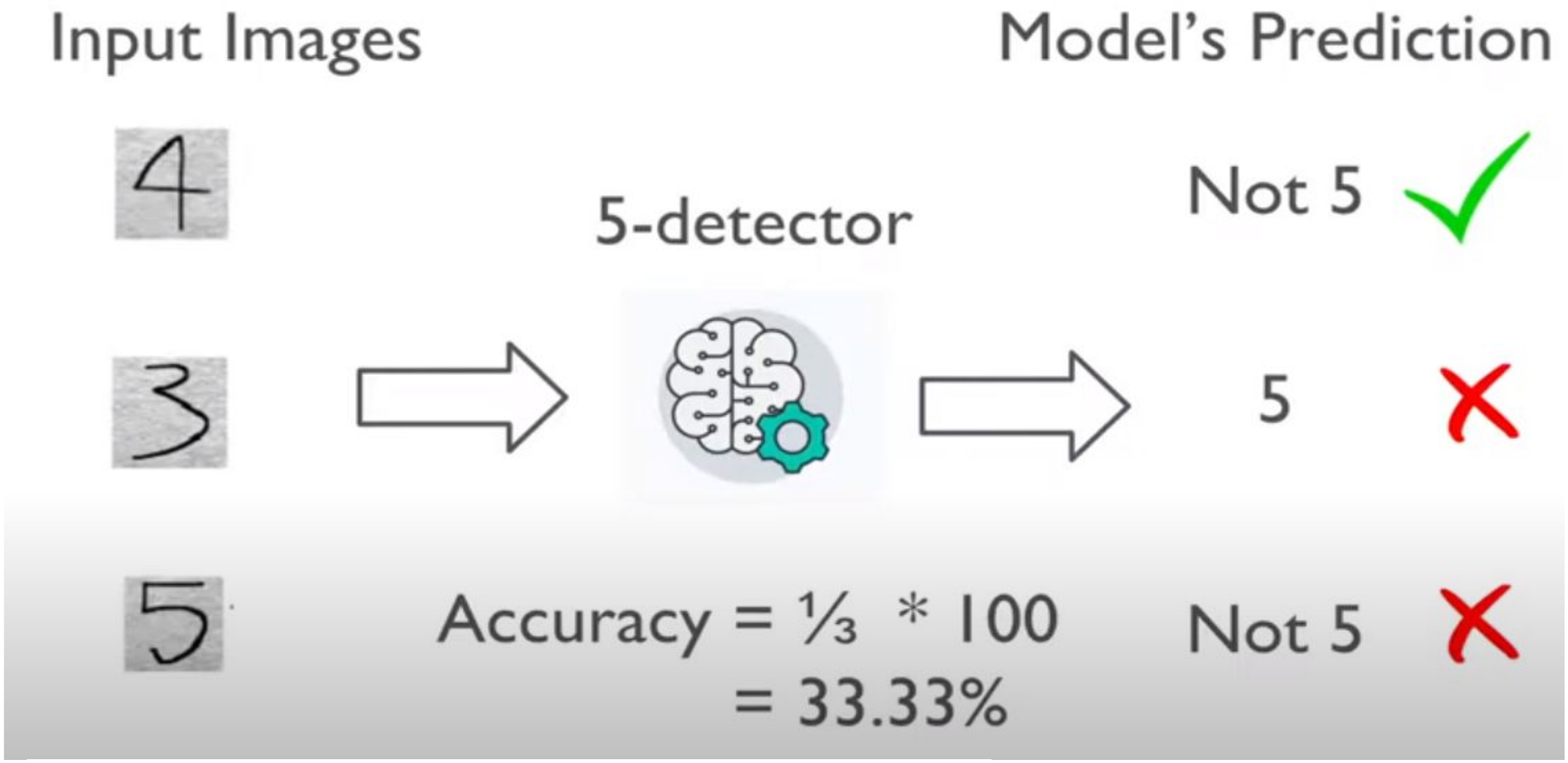
Skewed Dataset



Skewed Dataset

Class Imbalance

Accuracy

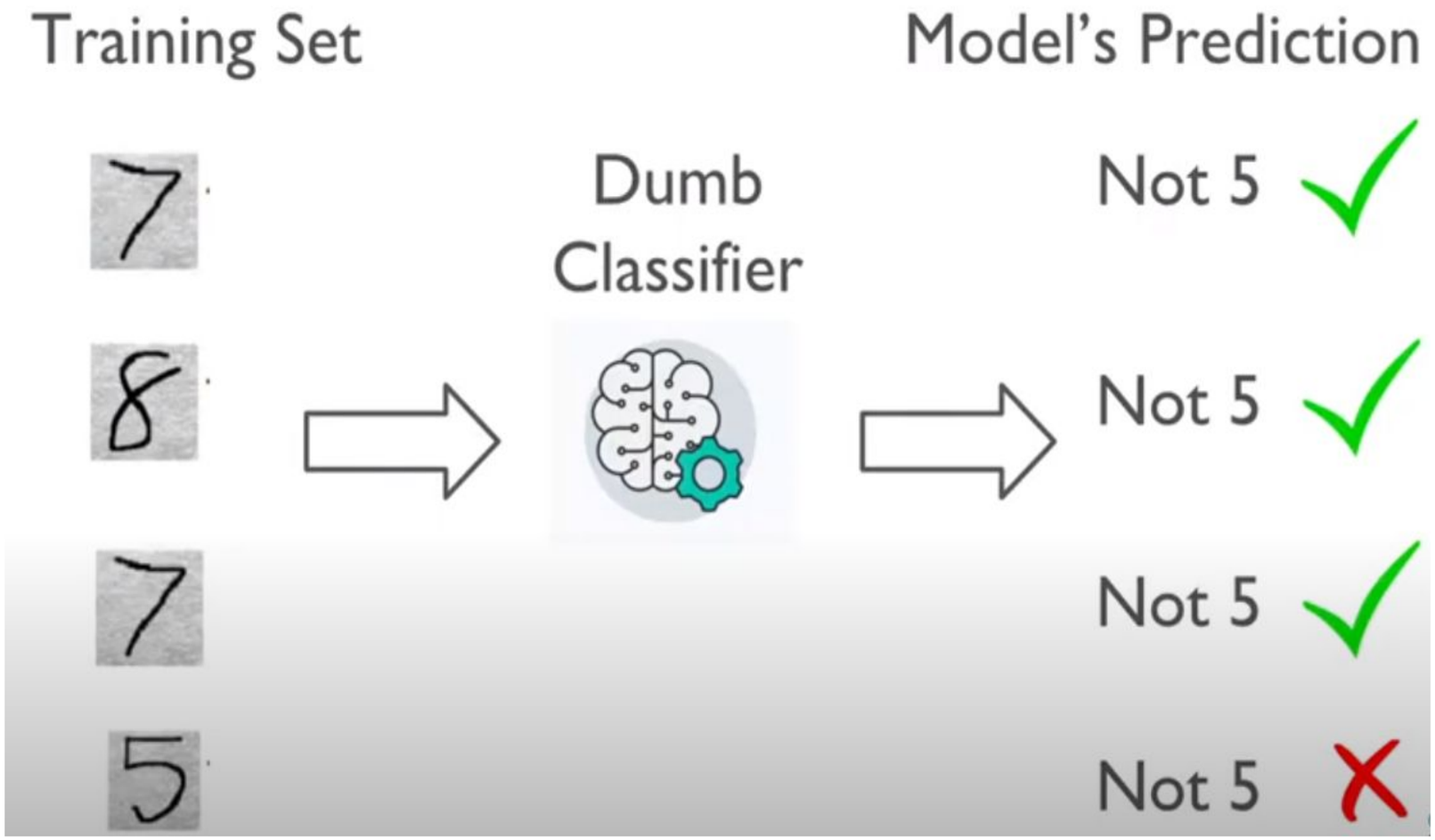


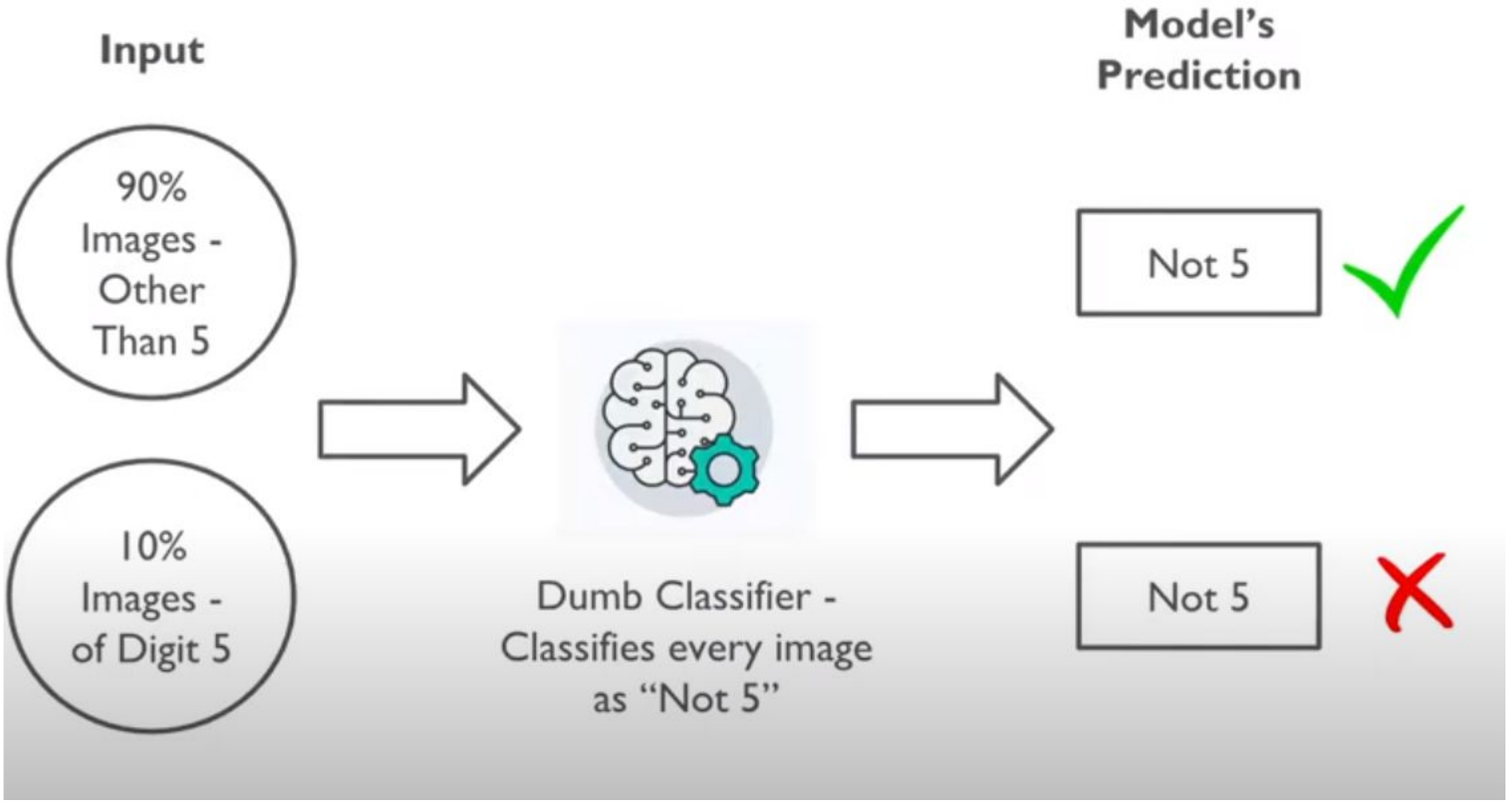
$$Accuracy = \frac{No.Samples\ Predicted\ Correctly}{Total\ No.of\ Samples}$$

Accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (when some classes are much more frequent than others).

DUMB CLASSIFIER

- Dumb classifier that just classifies every single image in the “not-5” class.




Input
Model's Prediction
90% Images - Other Than 5
10% Images - of Digit 5
Dumb Classifier - Classifies every image as "Not 5"
Not 5
Not 5

```python
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.91125, 0.90855, 0.90915])
```

Performing Cross validation in Scikit learn

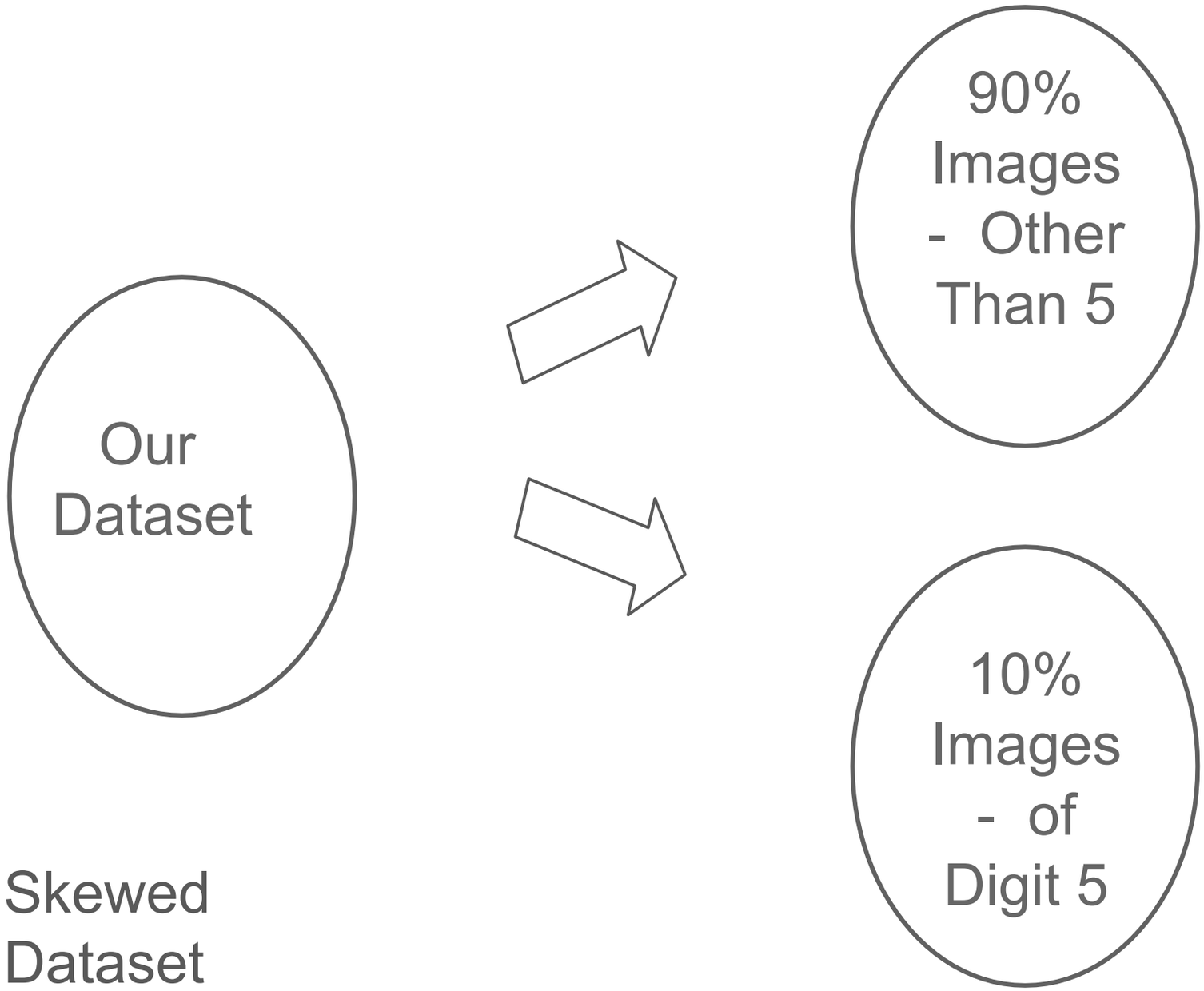
Accuracy = 95 %

SGDClassifier

Accuracy = 90 %

Dumb Classifier - Never5 Classifier

Skewed Dataset



Skewed Dataset

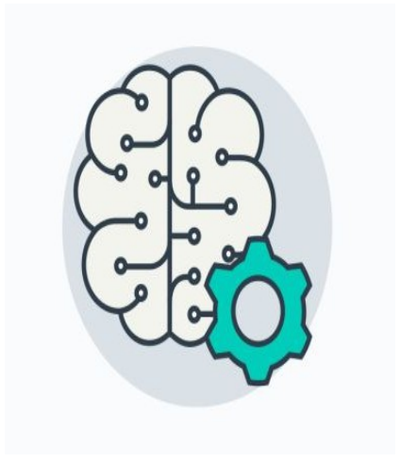
Class Imbalance

Input

Model's Prediction

90% Images - Other Than 5

10% Images - of Digit 5



Dumb Classifier - Classifies every image as "Not 5"

Accuracy → 90%

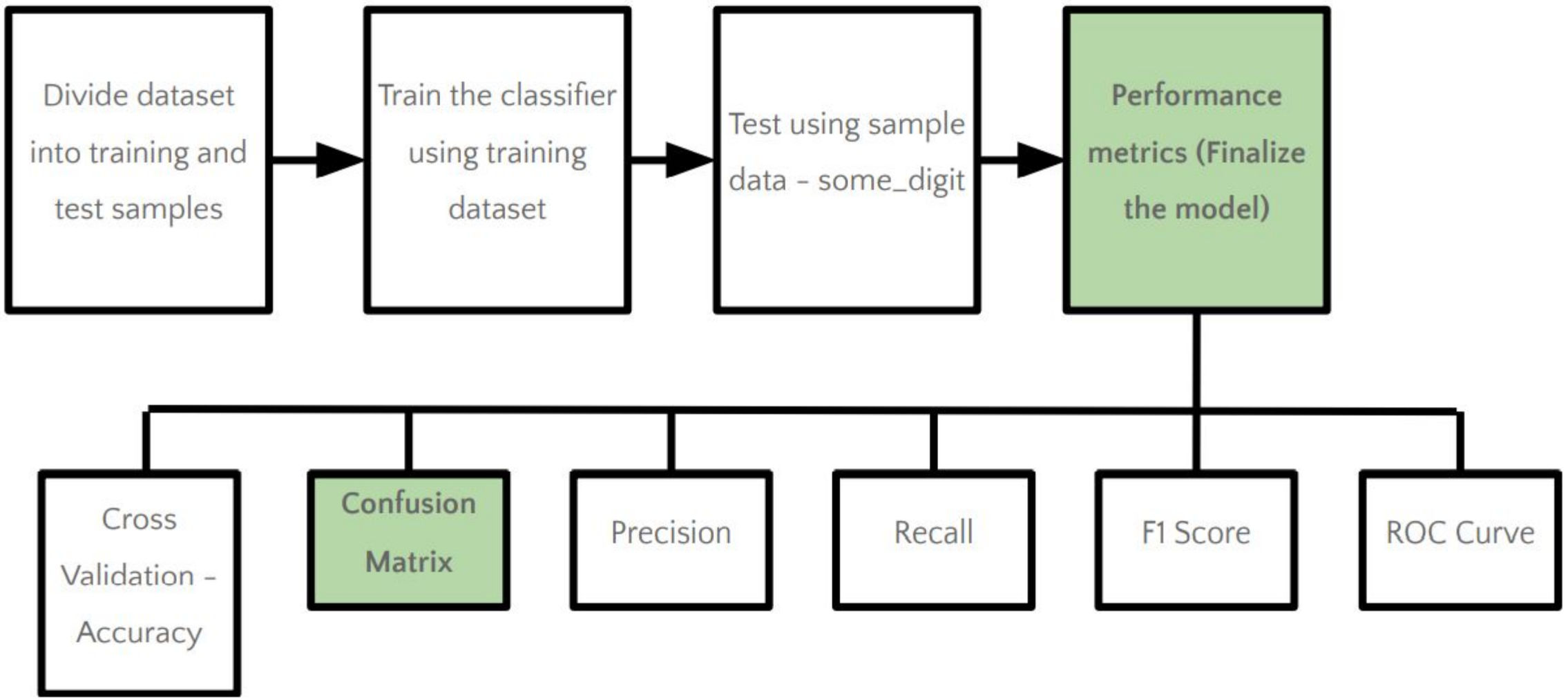
Not 5

Not 5

Learnings??

- Accuracy may not be a good performance measure when dealing with skewed datasets

Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data - some_digit
Performance metrics (Finalize the model)
Cross Validation - Accuracy
Confusion Matrix
Precision
Recall
F1 Score
ROC Curve

Performance measures - Confusion Matrix

- What is confusion matrix?
  - The general idea is to count the number of times instances of class A are classified as class B.
  - Can be better than simple accuracy.
  - For example: to know the number of times the classifier confused images of 5s with 3s

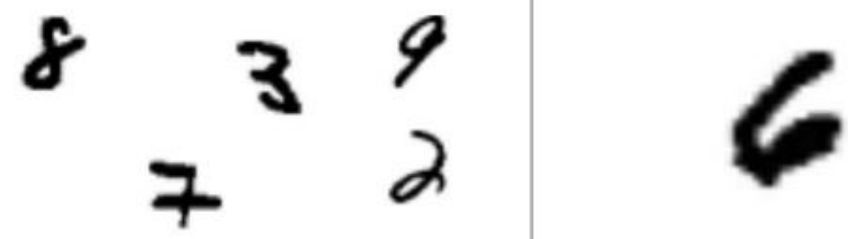
	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Outline of a confusion matrix for 2-way classification

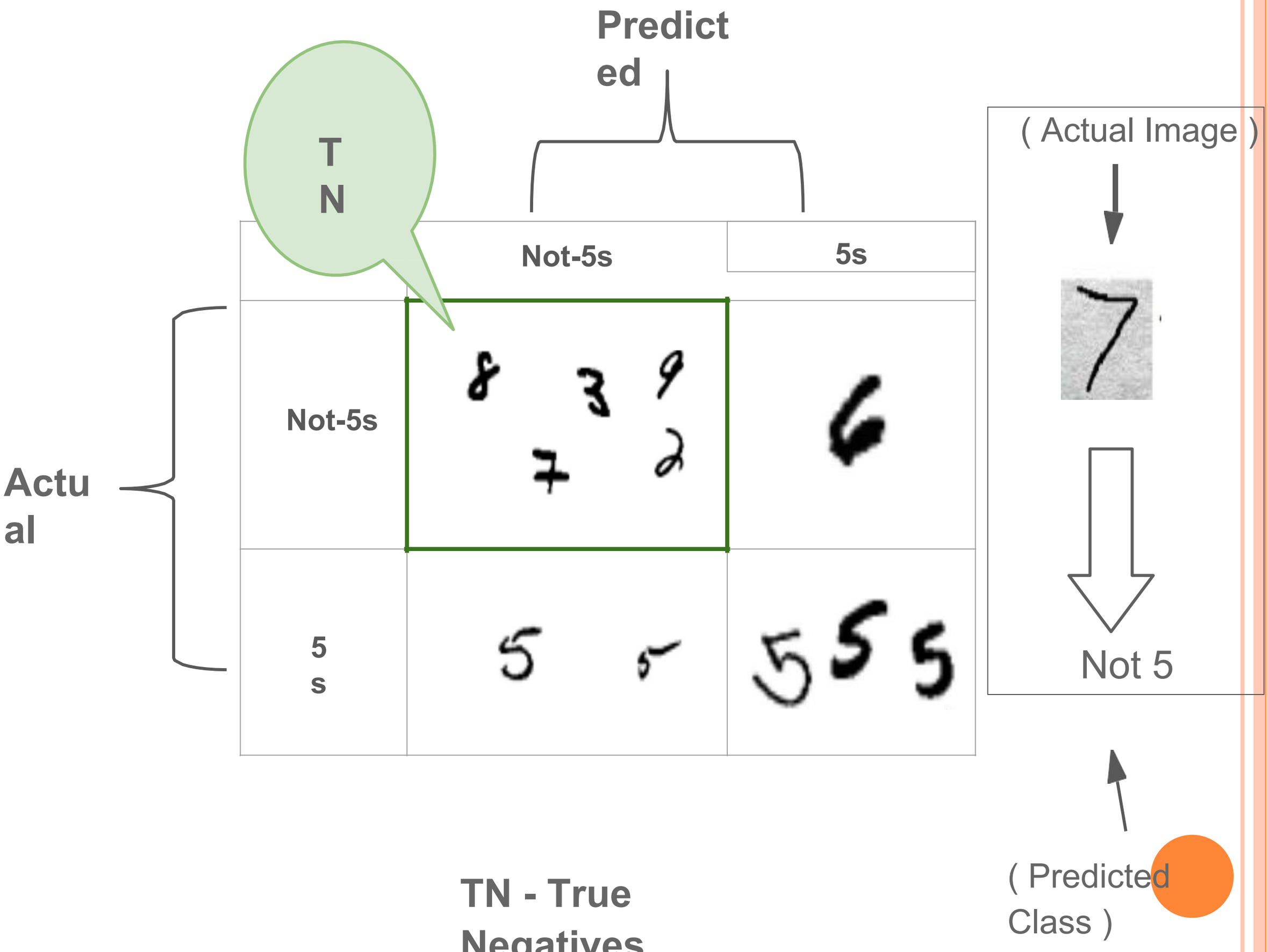
TN = True Negative
TP = True Positive
FN = False Negative
FP = False Positive

Predicted

Actual

	Not-5s	5s
Not-5s		
5s		

Sample Confusion Matrix of 5-Detector



TN - True Negatives

Predicted

FP

Actual	Not-5s	5s
Not-5s	8 3 9 7 2	6
5s	5 5	5 5 5

(Actual Image)

5

(Predicted Class)

FP - False Positives

Predicted

Actual	Not-5s	5s
Not-5s	8 3 9 7 2	6
5	5 5	5 5 5

FN

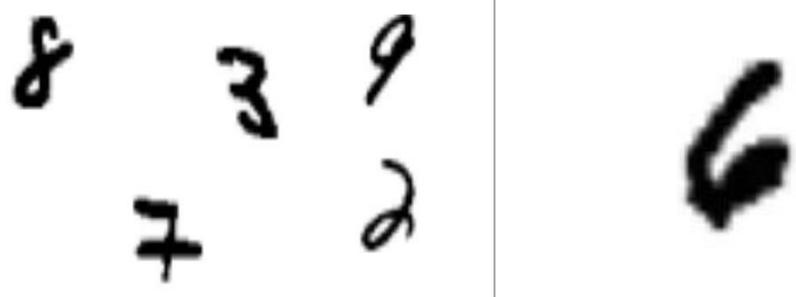
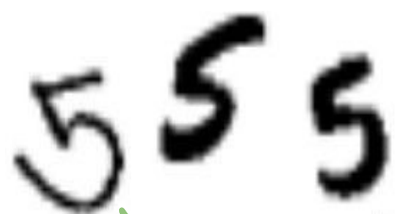
FN - False Negatives

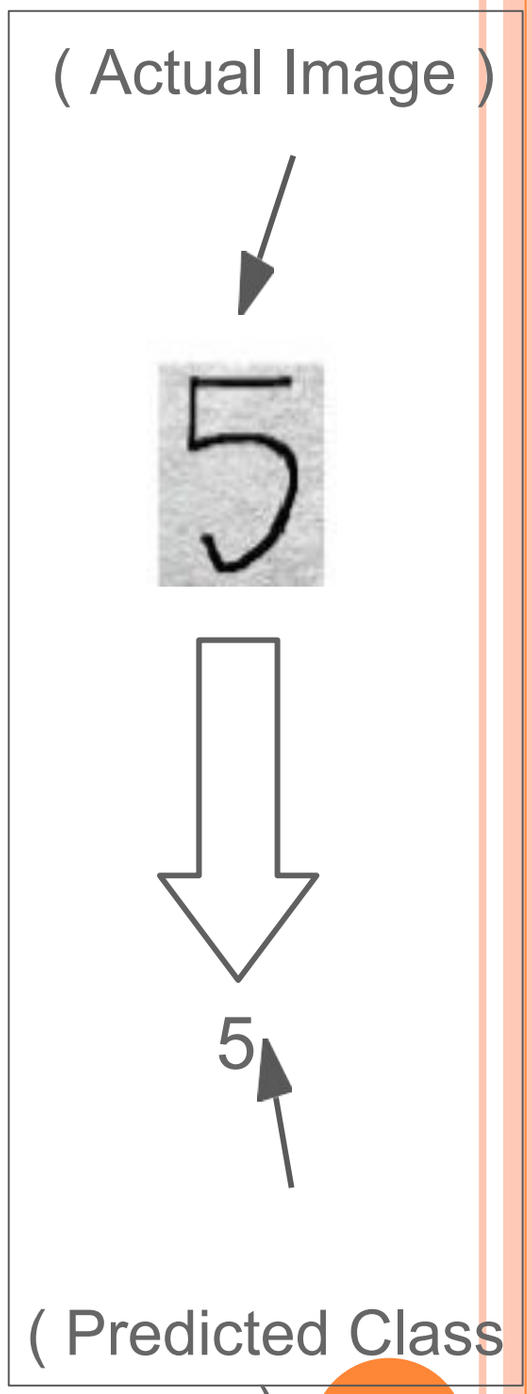
(Actual Image)

Not 5

(Predicted Class)

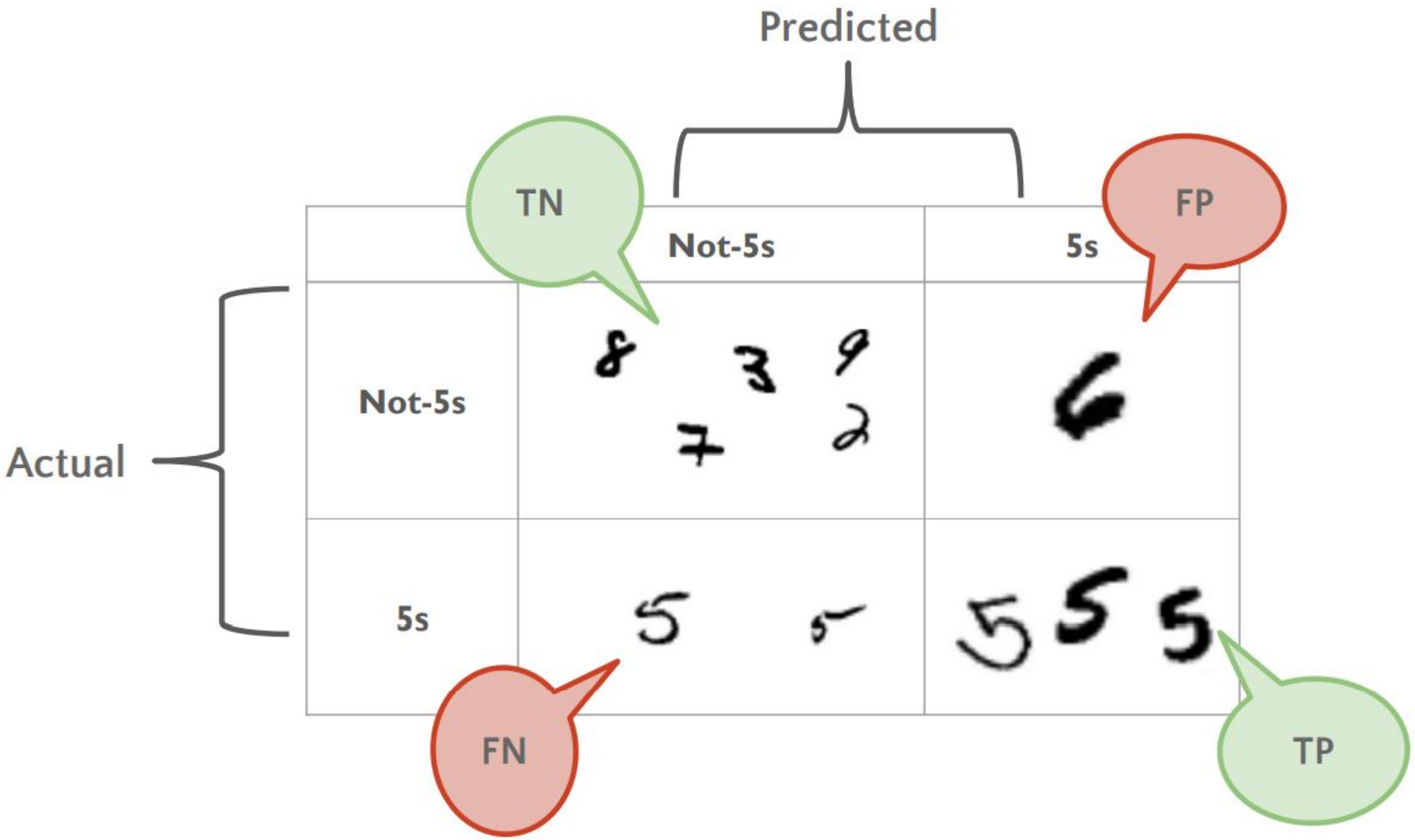
Predicted

Actual		Not-5s	5s
	Not-5s		
	5s		



TP - True Positives

T P


Predicted
TN
FP
Not-5s
5s
Actual
Not-5s
5s
FN
TP

- In-order to compute the confusion matrix we would consider set of predictions so that they can be compared to the actual targets.
- Keep test set aside and apply cross_val_predict() function on train set.

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

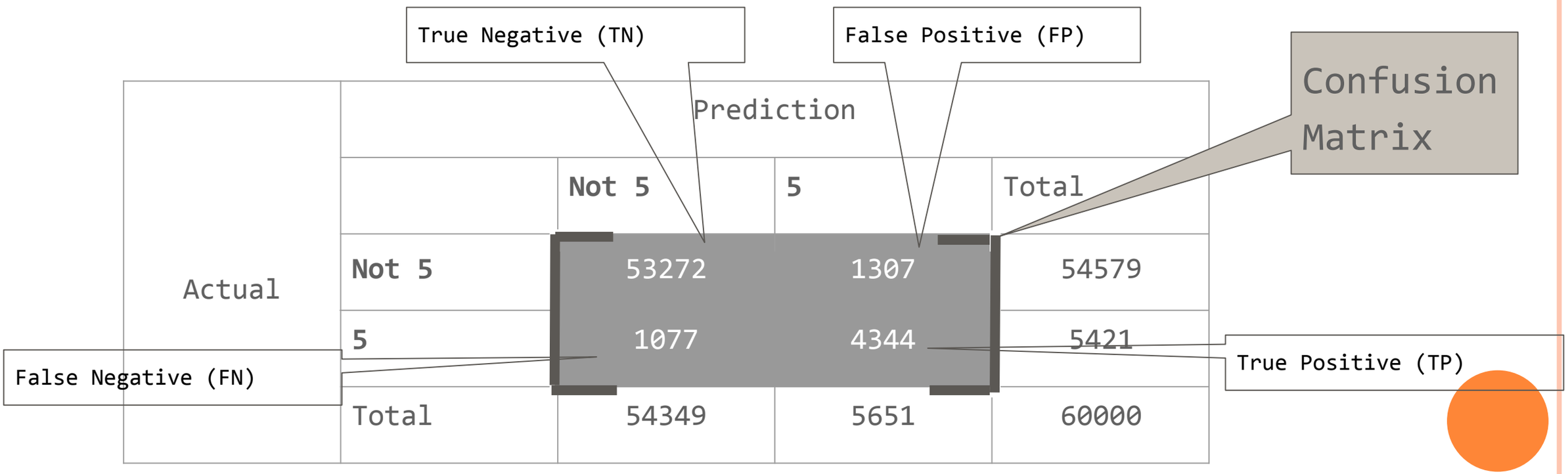
cross_val_predict() performs K-fold cross-validation, and returns the predictions made on each fold.
Confusion matrix is printed using confusion_matrix() function. To this we pass the target classes (y_train_5) and the predicted classes (y_train_pred):

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

Confusion Matrix - Example

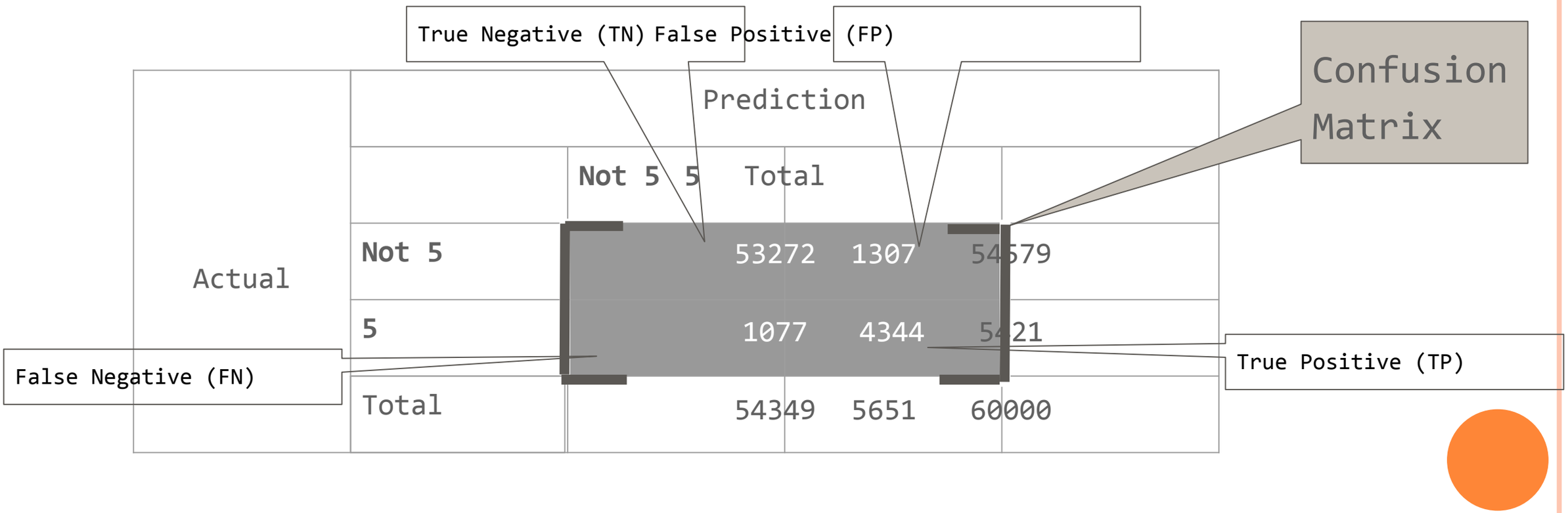
For '5' and 'Not 5' classifier

- Each row in a confusion matrix represents an actual class, while each column represents a predicted class.
- The first row of this matrix considers non-5 images (the negative class):
  - 53,272 of them were correctly classified as non-5s (they are called true negatives)
  - The remaining 1,307 were wrongly classified as 5s (false positives).



For ‘5’ and ‘Not 5’ classifier

- The second row considers the images of 5s (the positive class):
  - 1,077 were wrongly classified as non-5s (false negatives)
  - The remaining 4,344 were correctly classified as 5s (true positives).

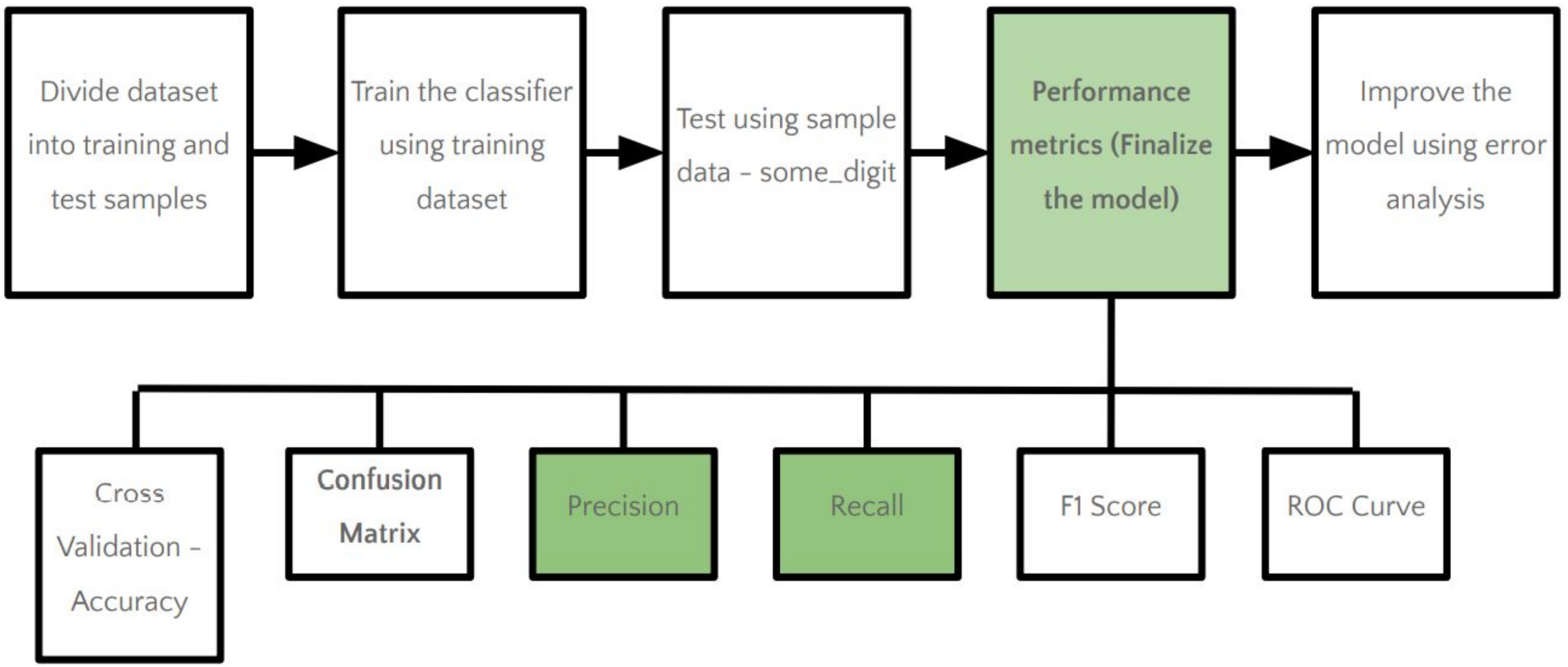


- A perfect classifier would have only true positives and true negatives, so its confusion matrix would have nonzero values only on its main diagonal (top left to bottom right):

```
>>> y_train_perfect_predictions = y_train_5  # pretend we reached perfection
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579,     0],
       [    0,  5421]])
```

- The confusion matrix gives a lot of information, but sometimes we may prefer a more concise metric. An interesting one to look at is the accuracy of the positive predictions; this is called the precision of the classifier

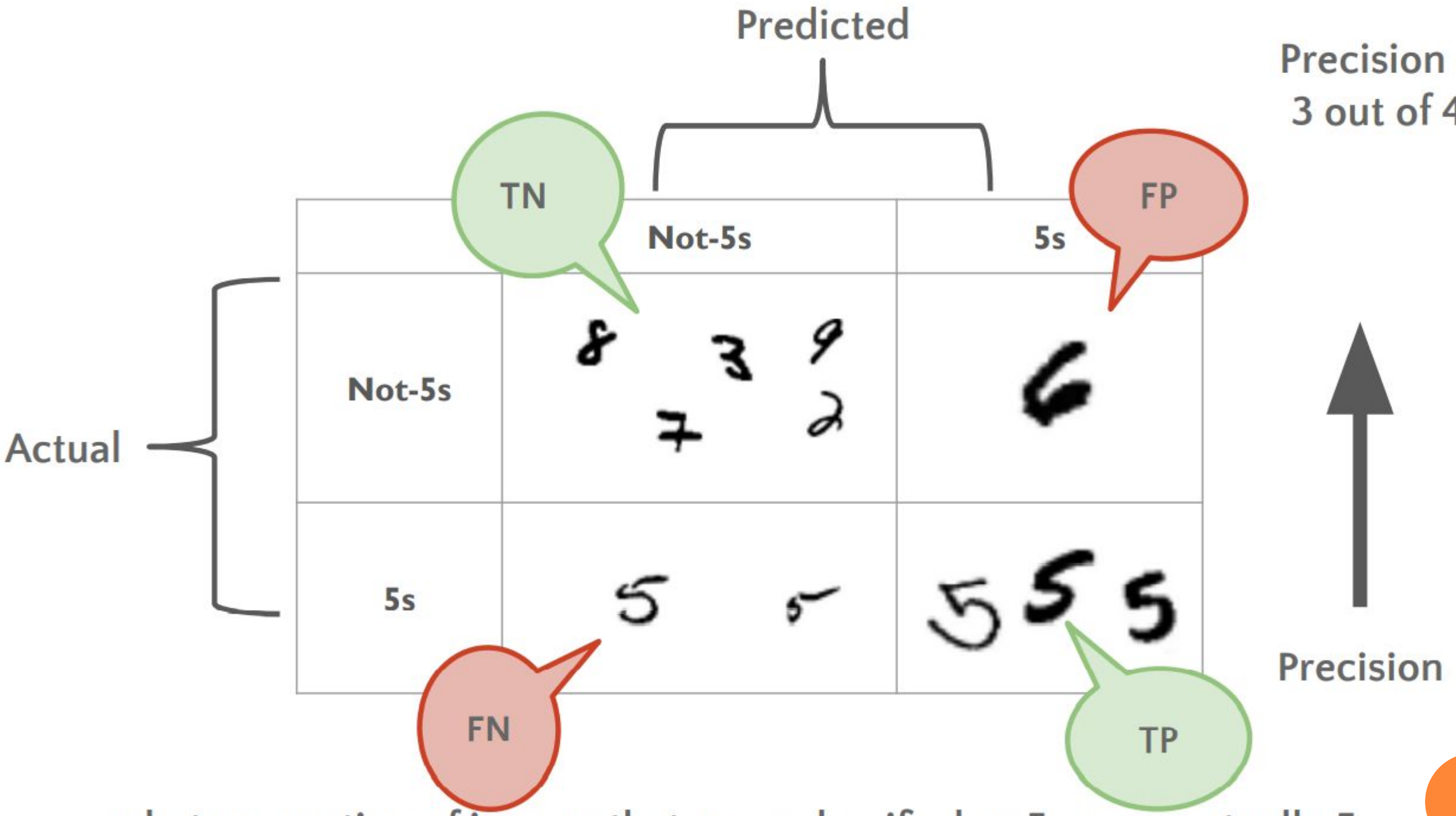
Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data - some_digit
Performance metrics (Finalize the model)
Improve the model using error analysis
Cross Validation - Accuracy
Confusion Matrix
Precision
Recall
F1 Score
ROC Curve

Performance Measures – Precision

Precision means lack of mistakes

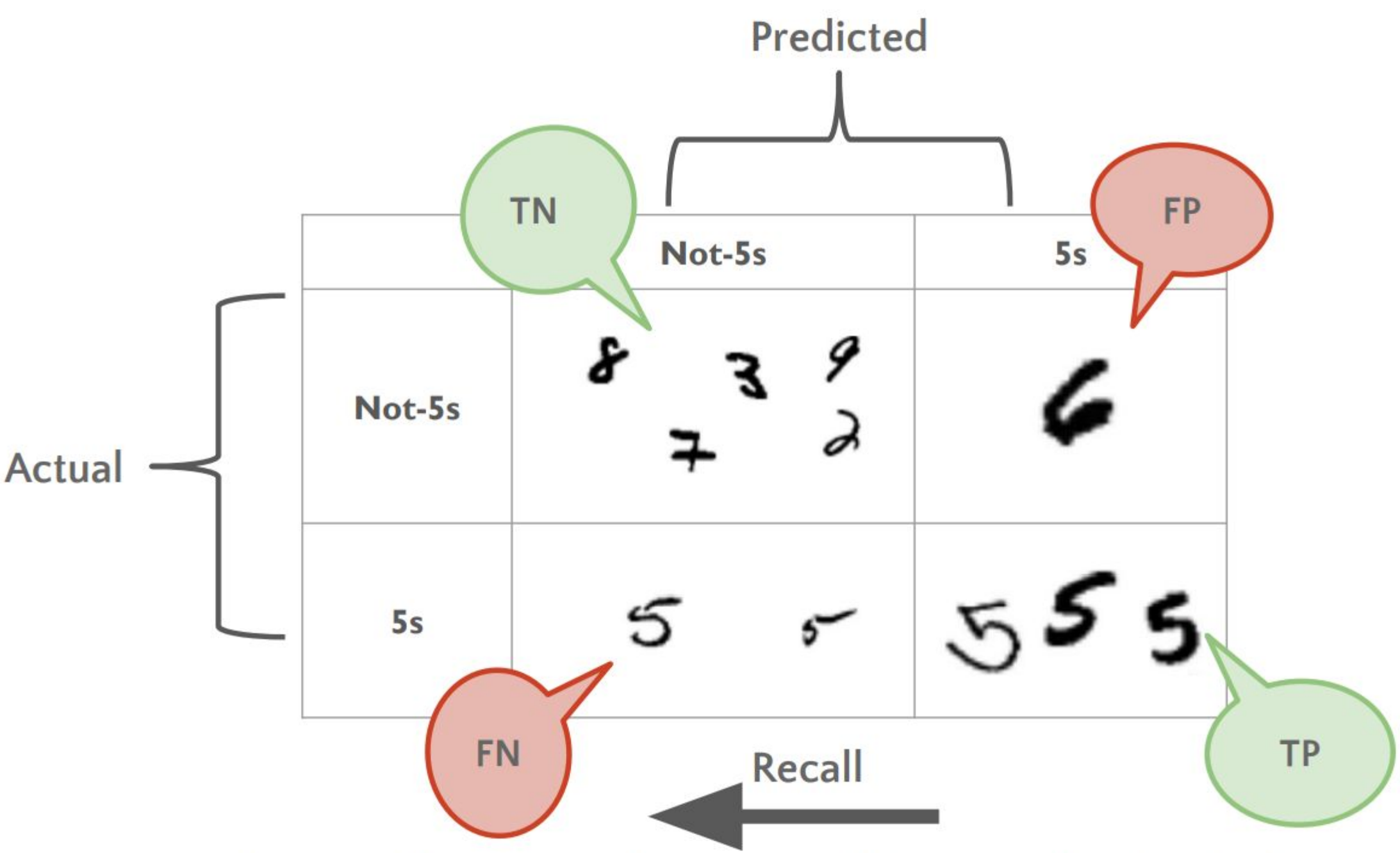
$$\text{precision} = \frac{TP}{TP + FP}$$



what proportion of images that were classified as 5s were actually 5

Performance Measures – Recall

Recall means remember something learnt in past



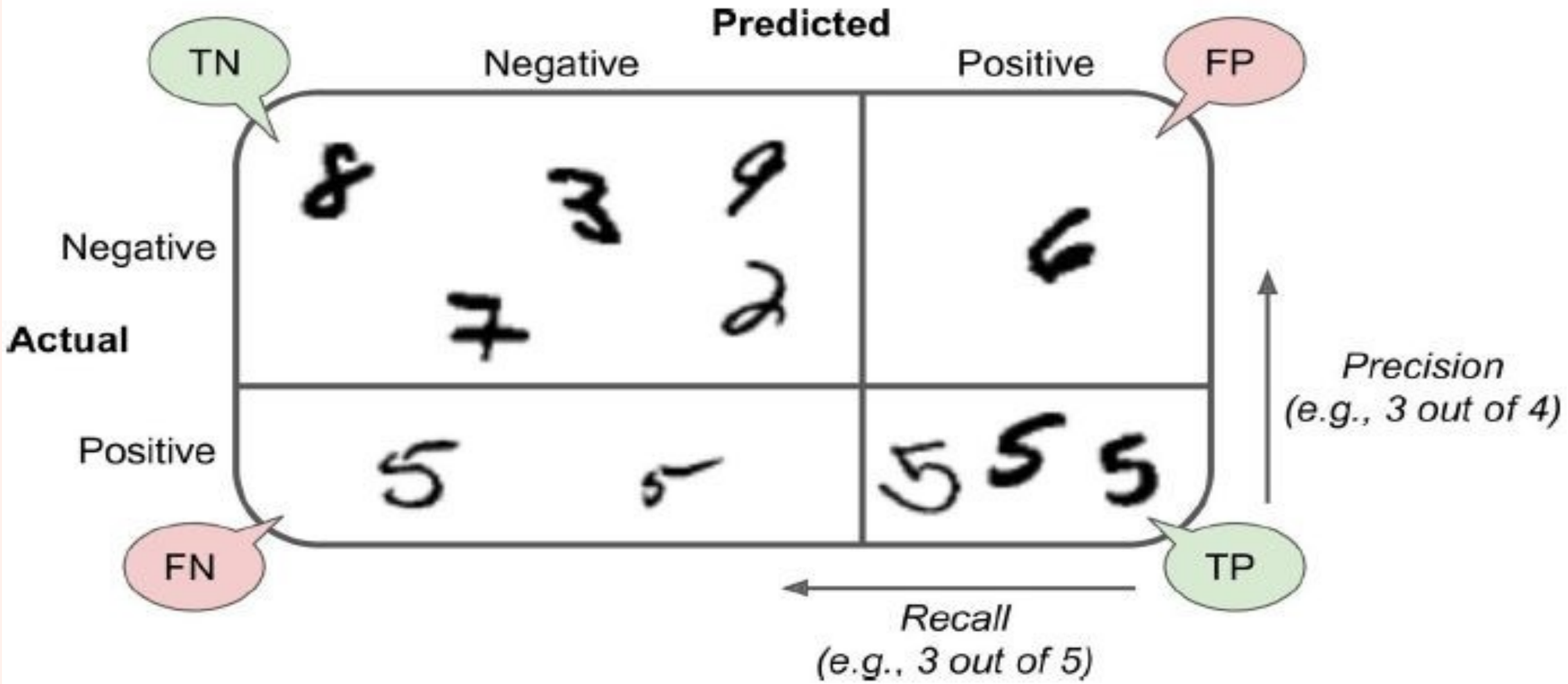
what proportion of images that were actually 5 were predicted as class 5

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

Recall – 3 out of 5

Performance measures - Precision and recall

'5' and 'Not 5' classifier



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

Recall, also called **sensitivity** or the **true positive rate** (TPR):

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

- **True - Positive** means the classifier correctly classified the **Positive** class.
- **True - Negative** means the classifier correctly classified the **Negative** class.
- **False - Positive** means the classifier incorrectly classified a **Negative** class as **Positive** Class.
- **False - Negative** means the classifier incorrectly classified a **Positive** class as **Negative** Class.

Precision and recall in scikit-learn

```
>>> from sklearn.metrics import precision_score,
recall_score
>>> precision_score(y_train_5,
y_train_pred)
 0.7687135020350380 8
>>> recall_score(y_train_5, y_train_pred)
0.7913669064748201 1
```

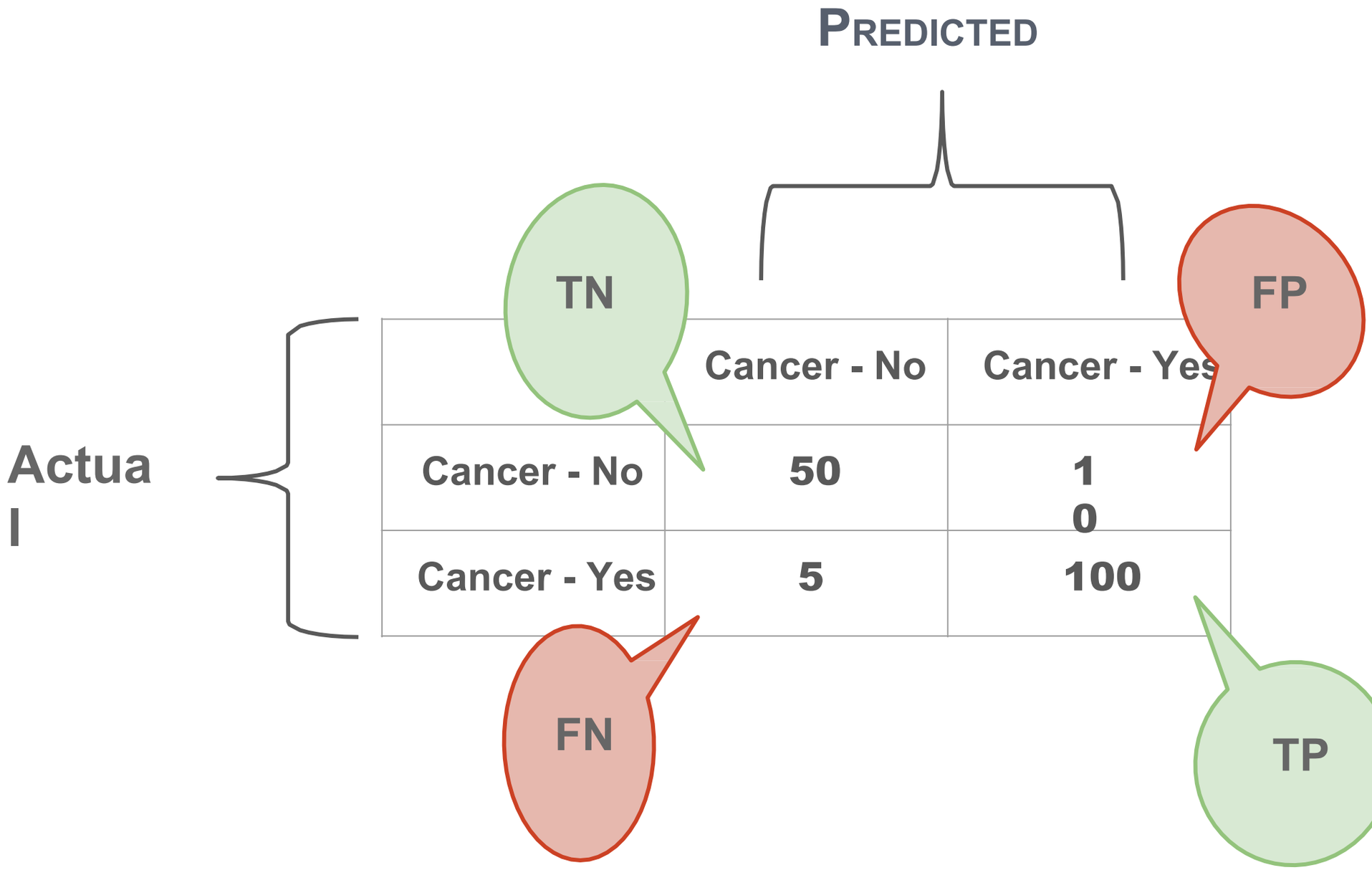
Performance Measures – Confusion Matrix

Precision = 83 %	Recall = 65 %
Correct only 83% of time	Detects only 65% of 5s

Precision / Recall Score of 5-detector

Let's see one more example of confusion matrix of model predicting if someone has cancer or not

Performance Measures - Confusion Matrix



Performance Measures - Confusion Matrix

Predicted

Actual

	Cancer - No	Cancer - Yes
Cancer - No	**50**	**10**
Cancer - Yes	**5**	**100**

Total Predictions = 110

Performance Measures - Confusion Matrix

Predicted

Actual

	Cancer - No	Cancer - Yes
Cancer - No	**50**	**10**
Cancer - Yes	**5**	**100**

Correct Predictions = 100

Precision - 100 out of 110 = 0.91

Performance Measures - Confusion Matrix

Predicted

Actual

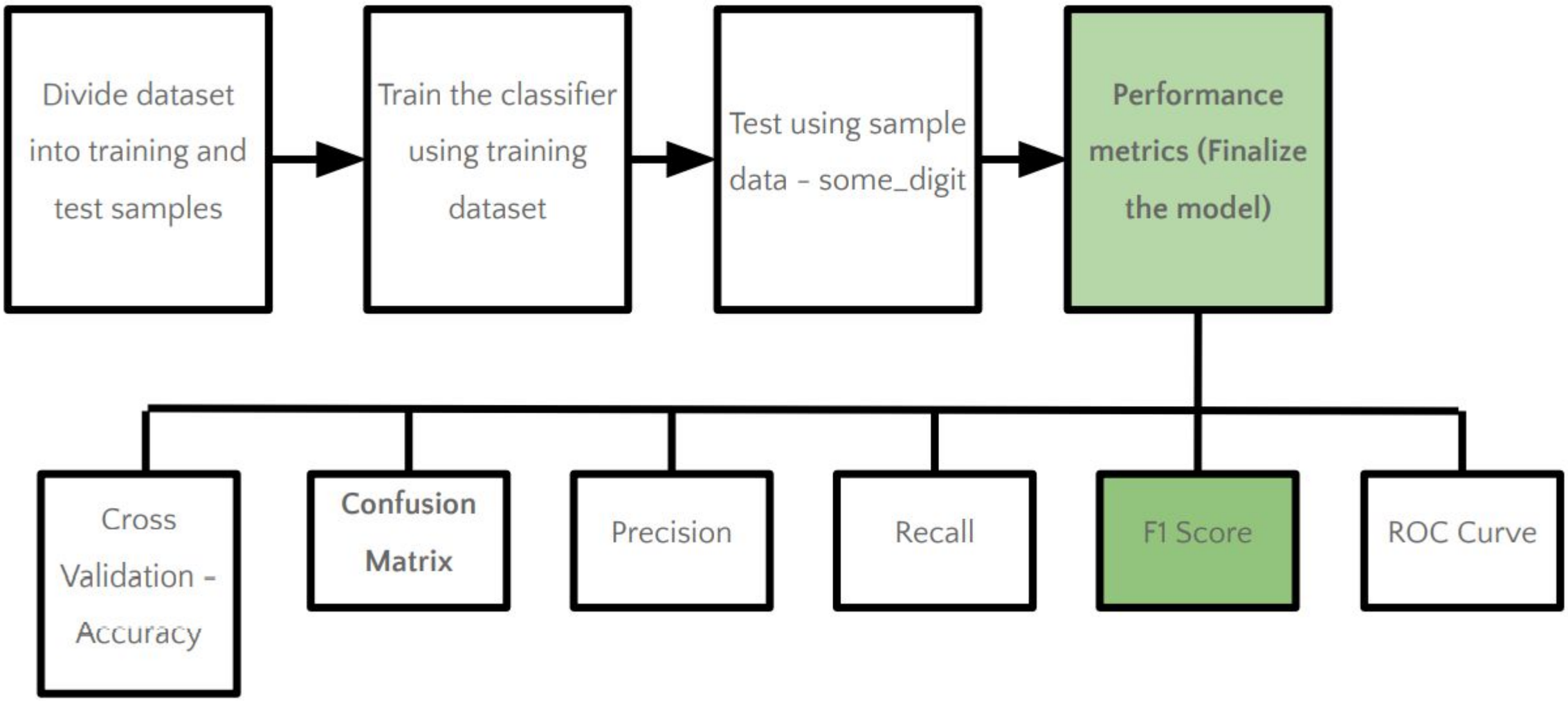
	Cancer - No	Cancer - Yes
Cancer - No	**50**	**10**
Cancer - Yes	**5**	**100**

Total Patients Having Cancer = 105

Model's Predictions as Having Cancer = 100

Recall - 100 out of 105 = 0.95

Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data - some_digit
Performance metrics (Finalize the model)
Cross Validation - Accuracy
Confusion Matrix
Precision
Recall
F1 Score
ROC Curve

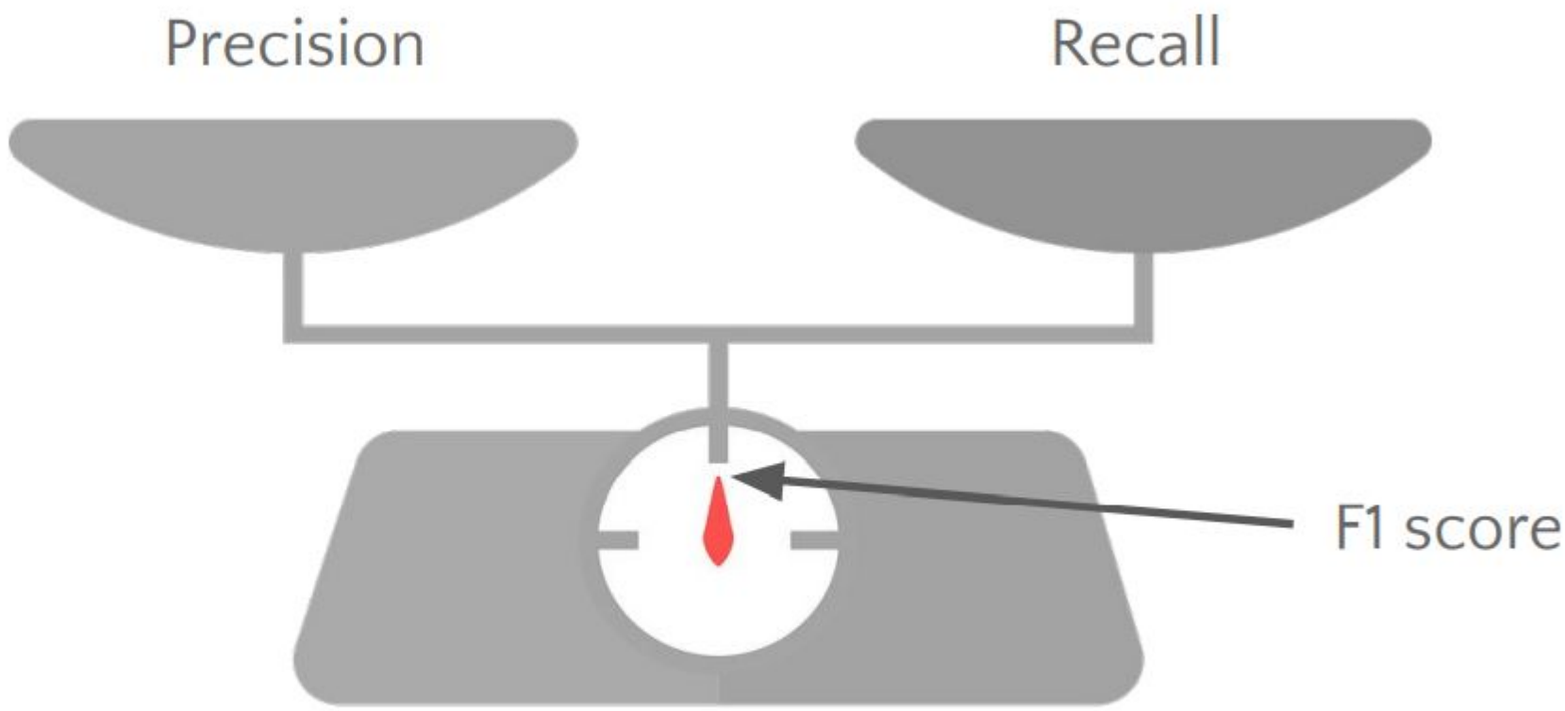
Performance Measures – F1 Score

- Instead of computing precision and recall every time
- We prefer a single metric which combines both precision and recall
- This single metric is f1 score

- F1 score is harmonic mean of precision and recall

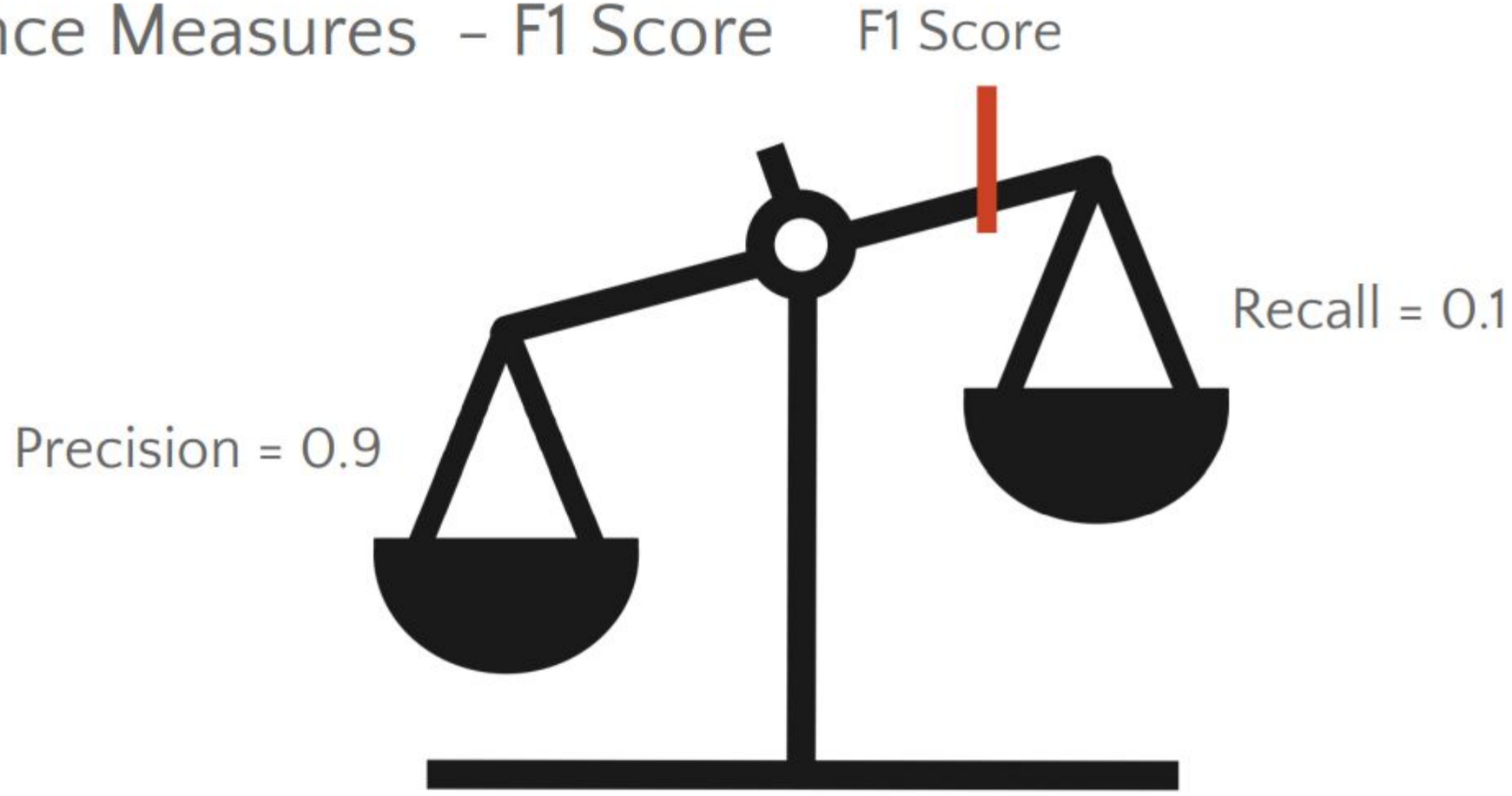
$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

Performance Measures – F1 Score



F1 score is high when both **Precision** and **Recall** are almost similar

Performance Measures – F1 Score

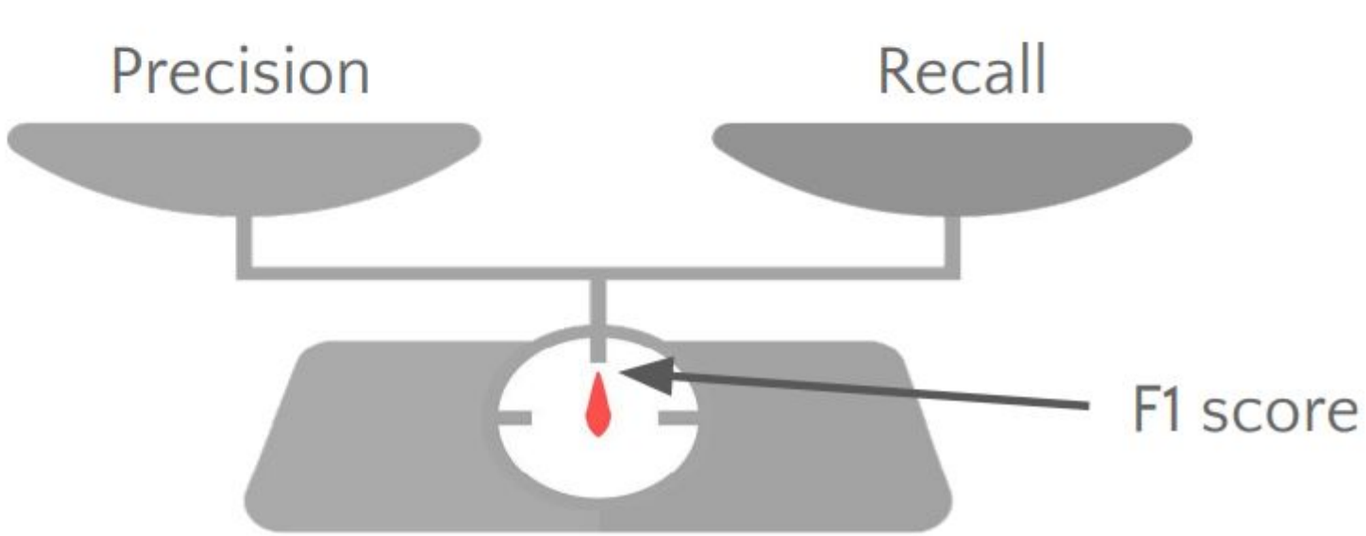


Recall is really small compared to **Precision**

F1 score will be closer to the smaller number than the bigger one

Performance Measures – F1 Score

- F1 score favors models that have similar precision and recall
- But depending on the problem which we are solving
  - We may go for higher precision or higher recall



- ‘5’ and ‘Not 5’ classifier

		Prediction			
		Not 5	5	Total	*Recall*
Actual	**Not 5**	53272	1307	54579	
	5	1077	4344	5421	= 4344 / 5421
	Total	54349	5651	60000	
	Precision		= 4344/5651 = 76.87 %		F1 score = 0.78468

F1 score = harmonic mean of precision and recall

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

- F1 score using scikit-learn

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

Say we have to build a model which detects if a video is safe for kids or not.

Question - High Precision or High Recall?

Performance Measures - F1 Score

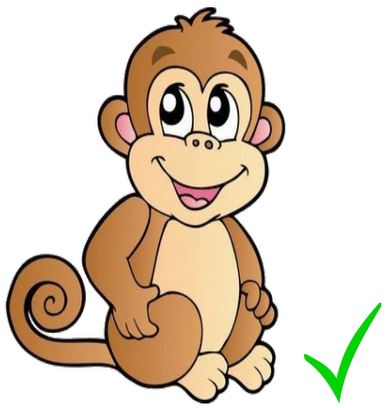
High precision means if the model classifies video 4 and video 6 as safe for kids, they are actually safe for kids.

In high precision, we are okay if the model is not able to classify video 2 as safe for kids but whichever videos it classifies as safe for kids they are actually safe.

1 - Not Safe

2 - Safe

3 - Not Safe



4 - Safe

5 - Not Safe

6 - Safe

High **Precision**

Performance Measures - F1 Score

High recall means the model will try to maximize the number of videos that are classified as safe.

1 - Not Safe

2 - Safe ✓

3 - Not Safe

4 - Safe ✓

5 - Not Safe

6 - Safe ✓

High
Recall

Performance Measures - F1 Score

In high recall model may mistake while classifying video as safe. This is because recall is more about classifying all the “safe for kids” videos as “safe” rather than classifying all the videos correctly.

1 - Not Safe

2 - Safe ✓

3 - Not Safe

4 - Safe ✓

5 - Not Safe

6 - Safe ✓

High **Recall**

Performance Measures - F1 Score

High **Precision** or High **Recall**?

1 - Not Safe

2 - Safe

3 - Not Safe

4 - Safe

5 - Not Safe

6 - Safe

Performance Measures - F1 Score

We would prefer a model which has high precision and low recall. It is okay if the model rejects many good videos but keeps only really safe ones

High **Precision**, Low **Recall**

1 - Not Safe

2 - Safe ✓

3 - Not Safe

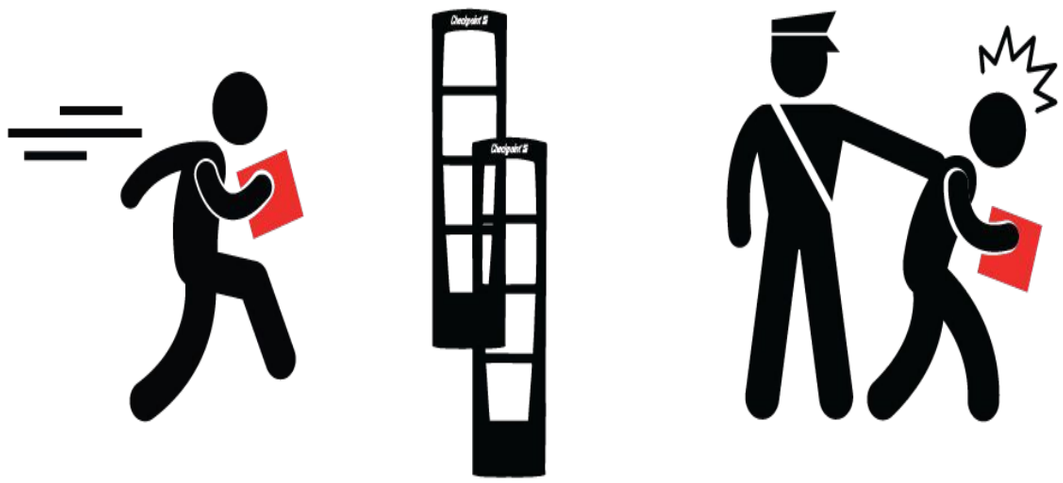
4 - Safe ✓

5 - Not Safe

6 - Safe ✓

Say we have to build a model which detects shoplifters on the basis of surveillance image. In case, someone is marked as shoplifter, we manually examine.

Question - High Precision or High Recall?



Performance Measures - F1 Score

We would prefer the model to have high recall even if the precision is low because our goal is to catch almost all the shoplifters.

In the high recall, the security guard might catch and examine some non shoplifters also but we will achieve our goal of catching almost all the shoplifters.

High **Recall**

Performance Measures - F1 Score

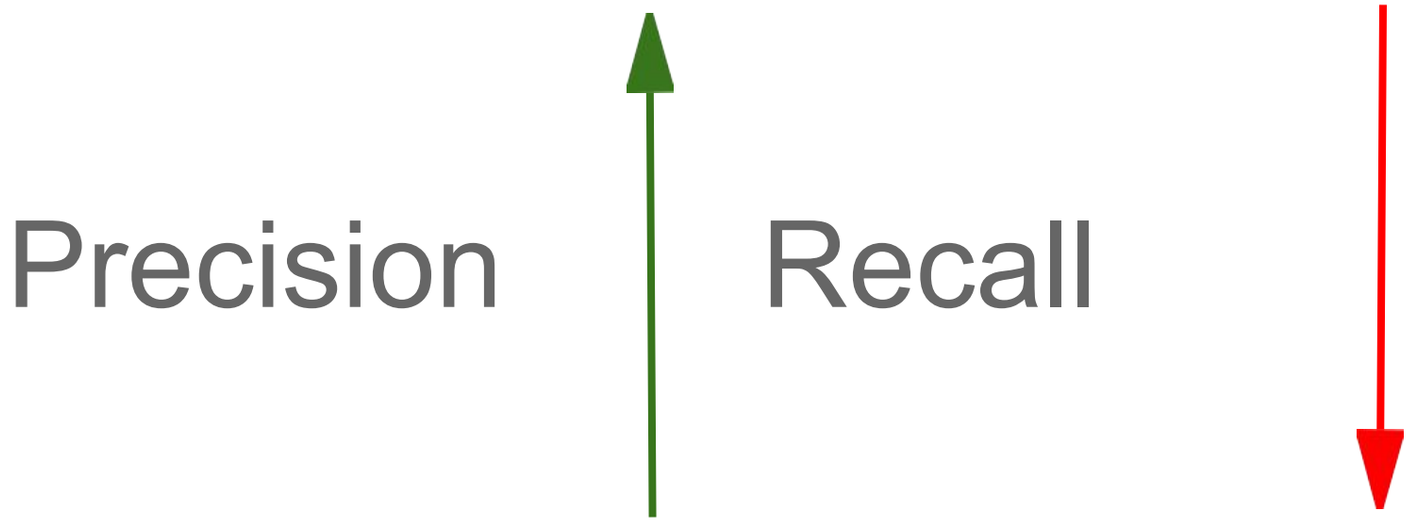
Now you may think that we can have both high precision and high recall in a good model.
But unfortunately, we can't have both high precision and high recall at the same time.

Precision ↑

Recall ↑

Performance Measures - F1 Score

Increasing the precision reduces recall and



Performance Measures - F1 Score

Vice versa

Performance measures - Precision vs Recall

- Different use cases may require different precision and recall

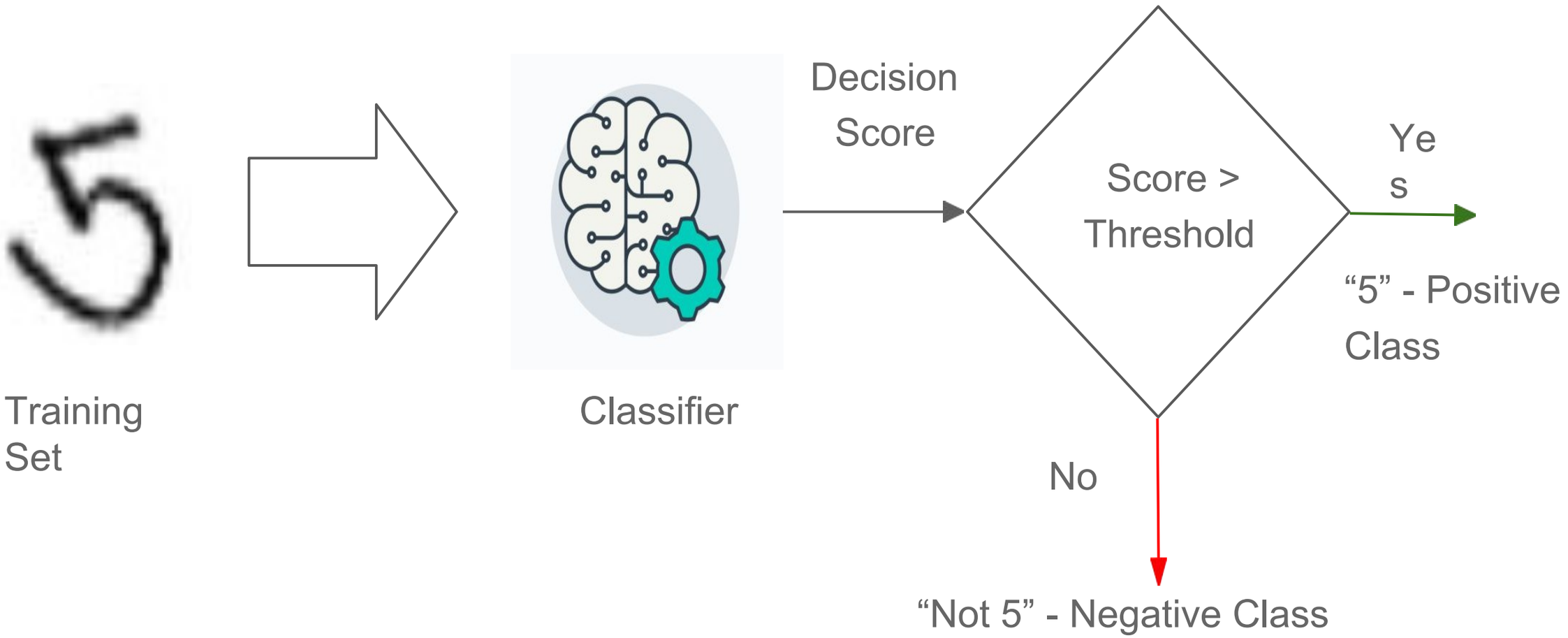
Performance measure (down)	**Detect videos that are unsafe for kids**	**Detect shoplifters in surveillance images**
Precision	High	Low
Recall	Low	High
	FP should be low, FN can be high	FP can be high, FN should be low

Increasing precision reduces recall, and vice versa. This is called the precision/recall tradeoff.

Performance Measures- Precision / Recall Tradeoff

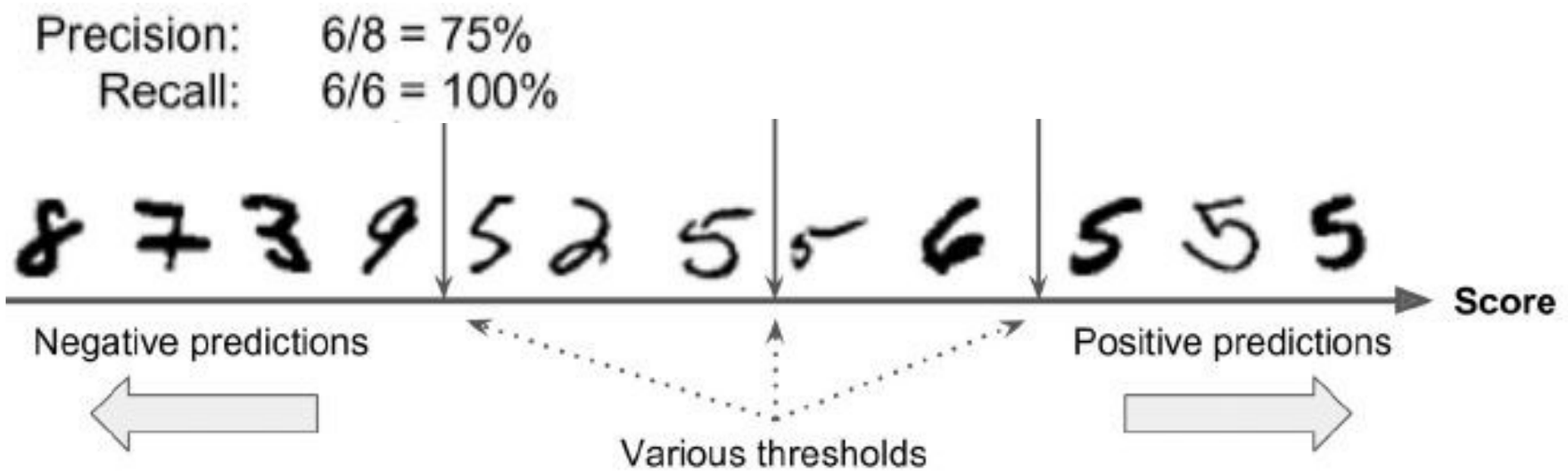
To understand this tradeoff, lets see how SGDClassifier works

Performance Measures - Precision / Recall Tradeoff



Decision Threshold - decided by the classification algorithm

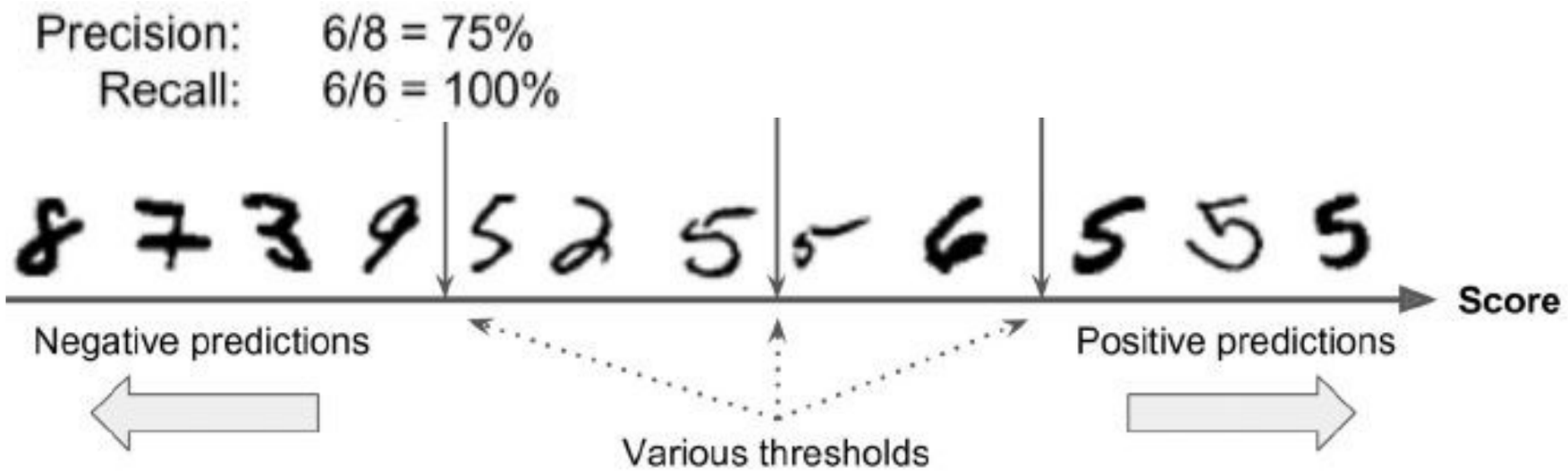
Precision / Recall Tradeoff - Thresholds



- SGD Classifier makes its classification decision, for instance it computes a *score* based on the *decision function*
  - Score above a certain *threshold* is classified as positive class
  - Score below a certain *threshold* is classified as negative class
- Thresholds can be set to achieve certain precision and recall.

Let us observe the above example : shows a few digits positioned from the lowest score on the left to the highest score on the right.

Precision / Recall Tradeoff



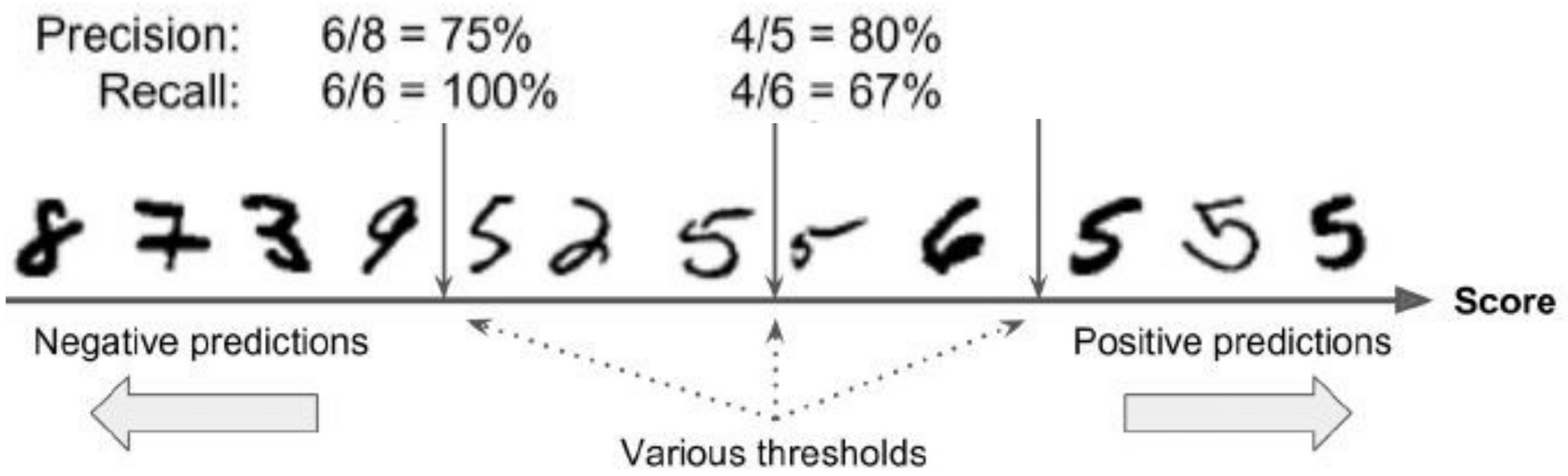
TN - TN - TN - TN - TP - FP - TP - TP - FP - TP - TP - TP

TN = 6, TP = 6, FN = 0, FP = 2

Precision = 6/ (6+2) = 75% Recall = 6/(6+0) = 100%

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

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



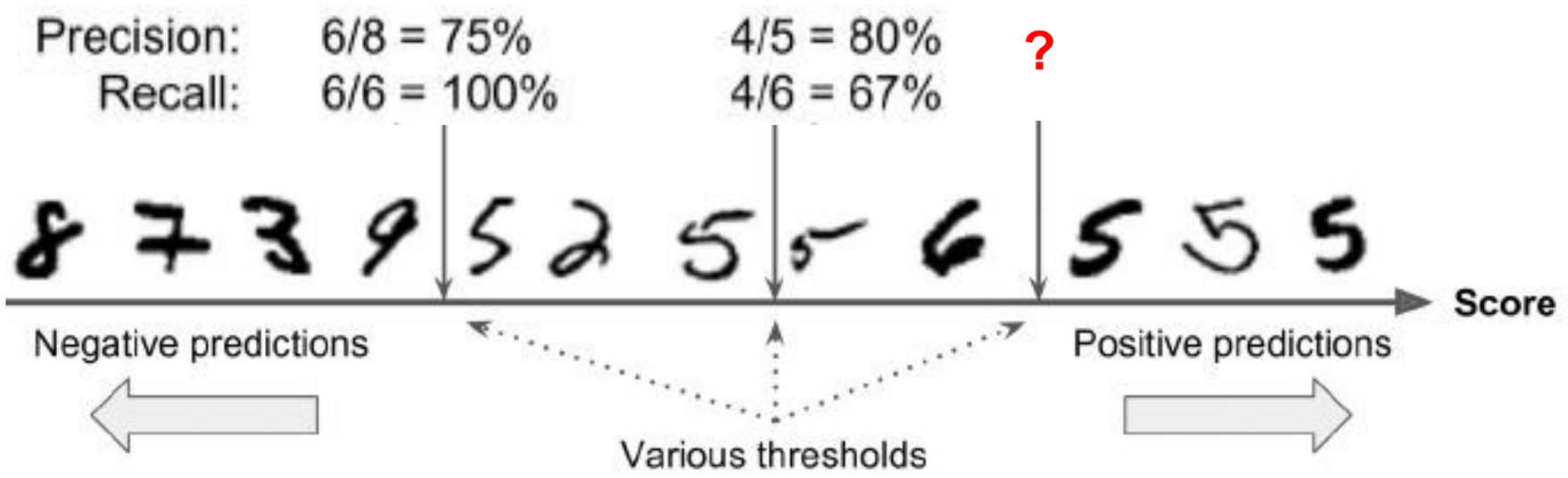
TN - TN - TN - TN - FN - TN - FN - TP - FP - TP - TP - TP

TN = 5, TP = 4, FN = 2, FP = 1

Precision = 4/ (4+1) = 80%
Recall = 4/ (4+2) = 67%

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{recall} = \frac{TP}{TP + FN}$$

Precision / Recall Tradeoff



TN - TN - TN - TN - FN - TN - FN - FN - TN - TP - TP - TP

TN = 6, TP = 3, FN = 2, FP = 0

Precision = 3/ (3+0) =
100% Recall = 3/(3+3) =
50%

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

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

Scikit-Learn does not allow to set the threshold directly, but it does provide access to the decision scores to make predictions.
Decision_function() method returns a score for each instance.
Then use any threshold for which we want to make predictions based on those scores:

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([2412.53175101])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True])
```

The SGDClassifier uses a threshold equal to 0 and returns True.
Let's raise the threshold:

```
>>> threshold = 8000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

This confirms that raising the threshold decreases recall. The image actually represents a 5, and the classifier detects it when the threshold is 0, but it misses it when the threshold is increased to 8,000.

How to decide the best threshold?

- Get the scores of all the training dataset using cross_val_predict with decision_function as function

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

- Compute the precision and recall for all possible thresholds using precision_recall_curve()

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

- Plot both precision and recall for the thresholds using matplotlib.

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold and add the legend, axis label, and grid

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```

- Select the threshold value that gives the best precision/ recall tradeoff.

- Plotting precision/ recall curve using Scikit-Learn

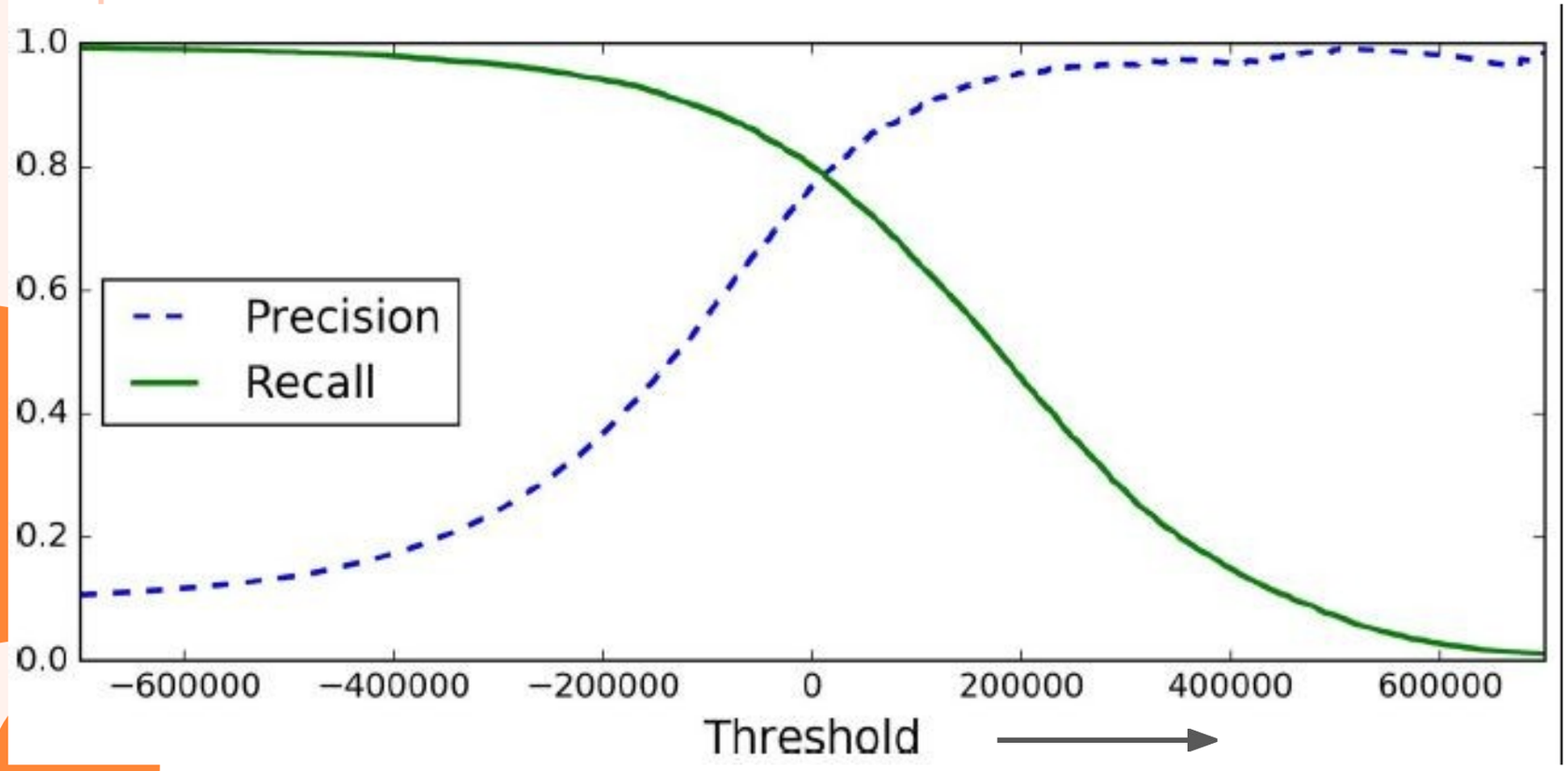
```
>>> def plot_precision_recall_vs_threshold(precisions,
recalls,  thresholds):
    plt.figure(figsize=(18,7))
    plt.plot(thresholds, precisions[:-1], "b--",
    label="Precision")  plt.plot(thresholds, recalls[:-1], "g-",
    label="Recall")  plt.xlabel("Threshold")
    plt.legend(loc="upper
    left")  plt.ylim([0, 1])

>>> plot_precision_recall_vs_threshold(precisions,
recalls,  thresholds)
>>> plt.show()
```

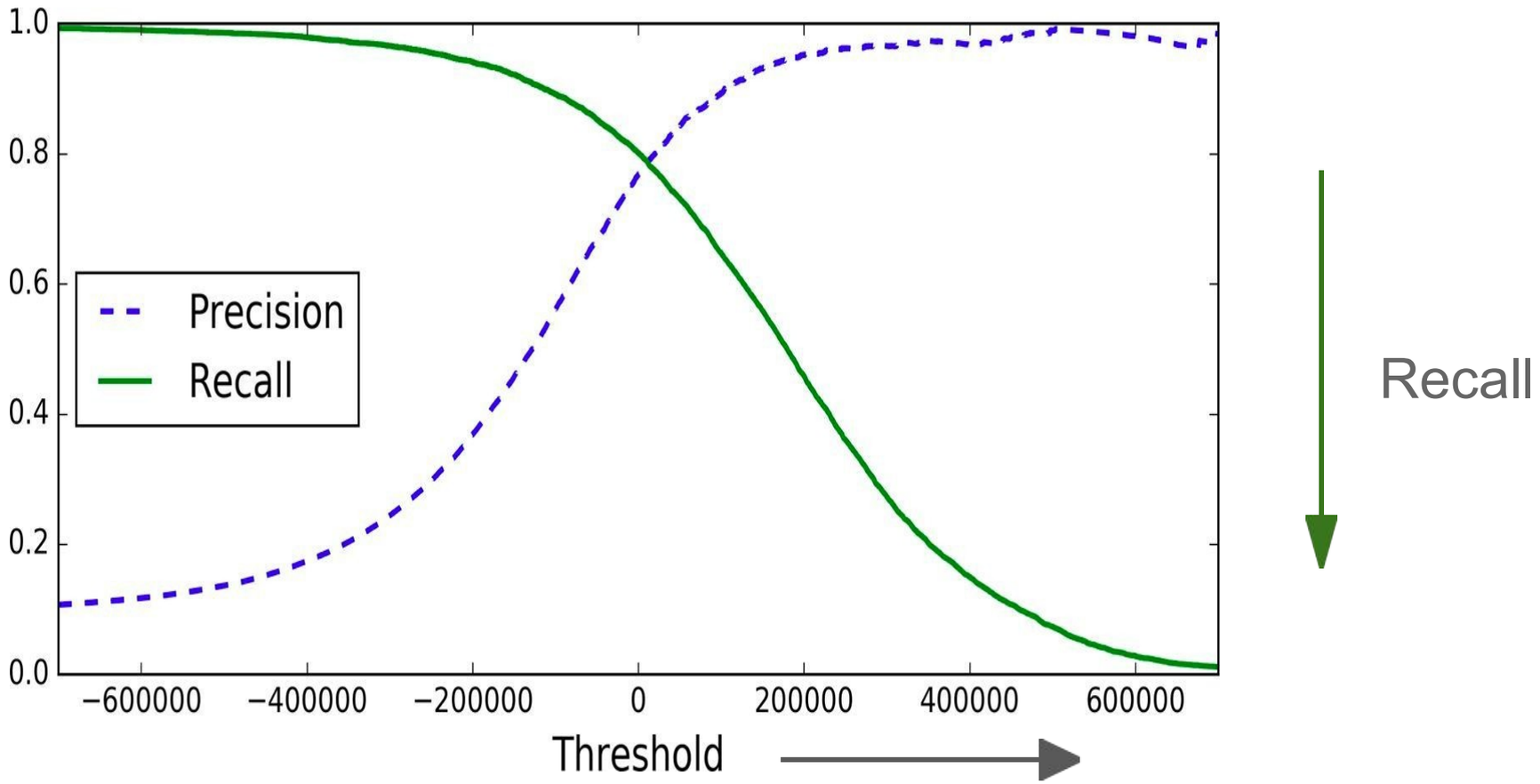
- Precision increases while recall decreases with increase in threshold
- Scikit enables the user to get the scores from the classifier

Precision / Recall Tradeoff

Q. How to decide the best threshold? What is the best threshold for below?

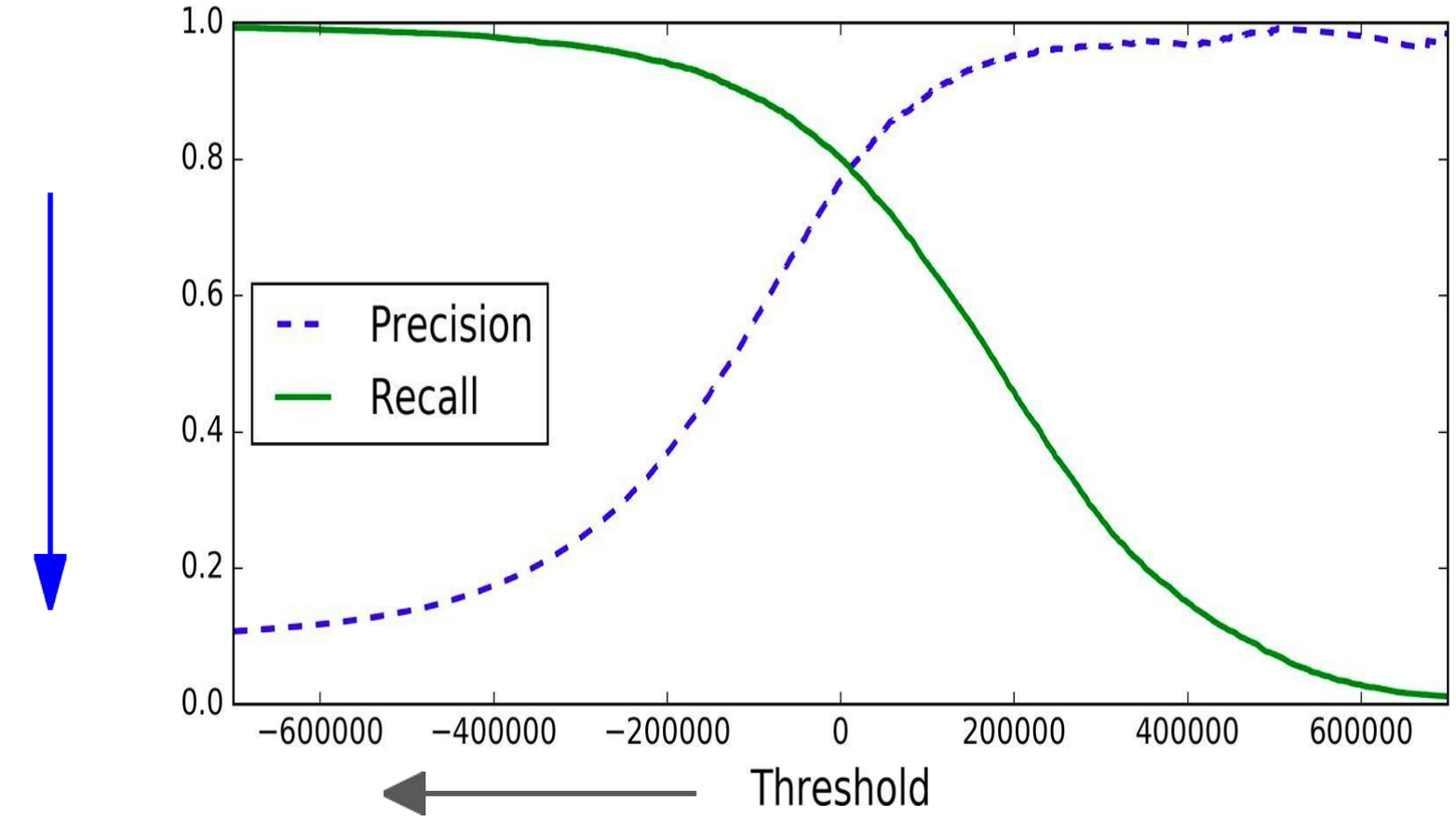


Precision / Recall Curve



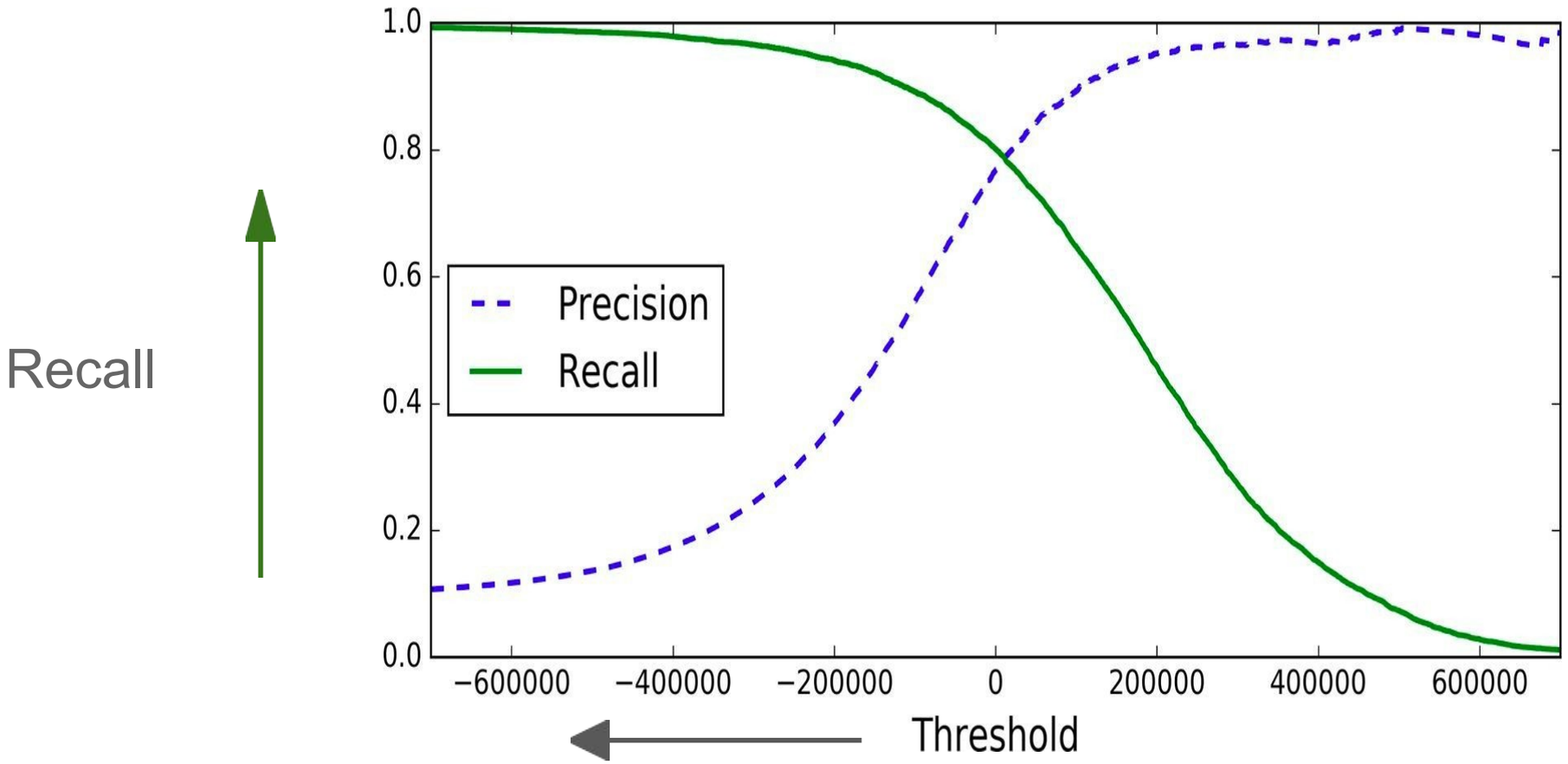
Precision and recall versus the decision threshold

Precision / Recall Curve



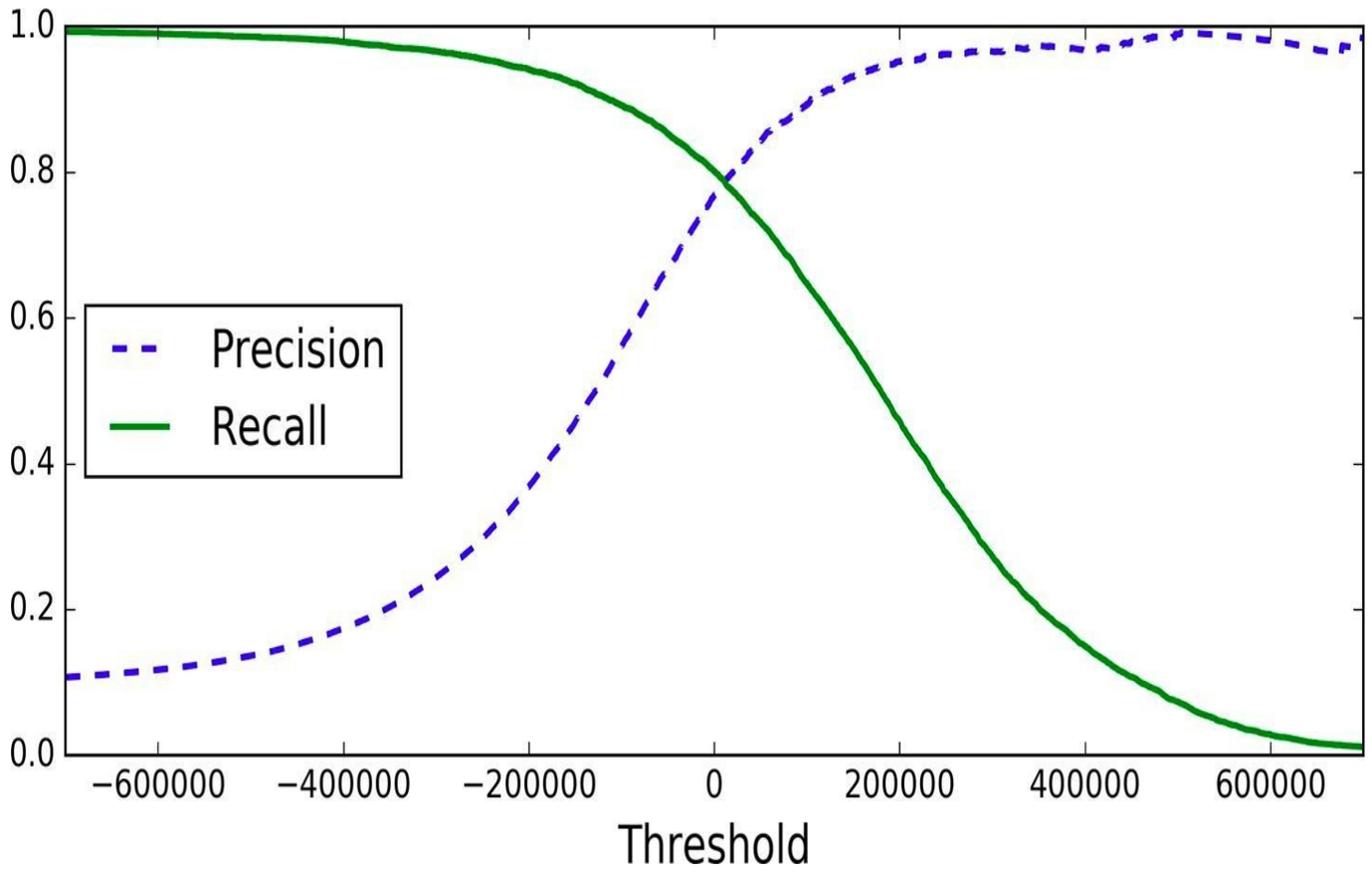
Precision and recall versus the decision threshold

Precision / Recall Curve



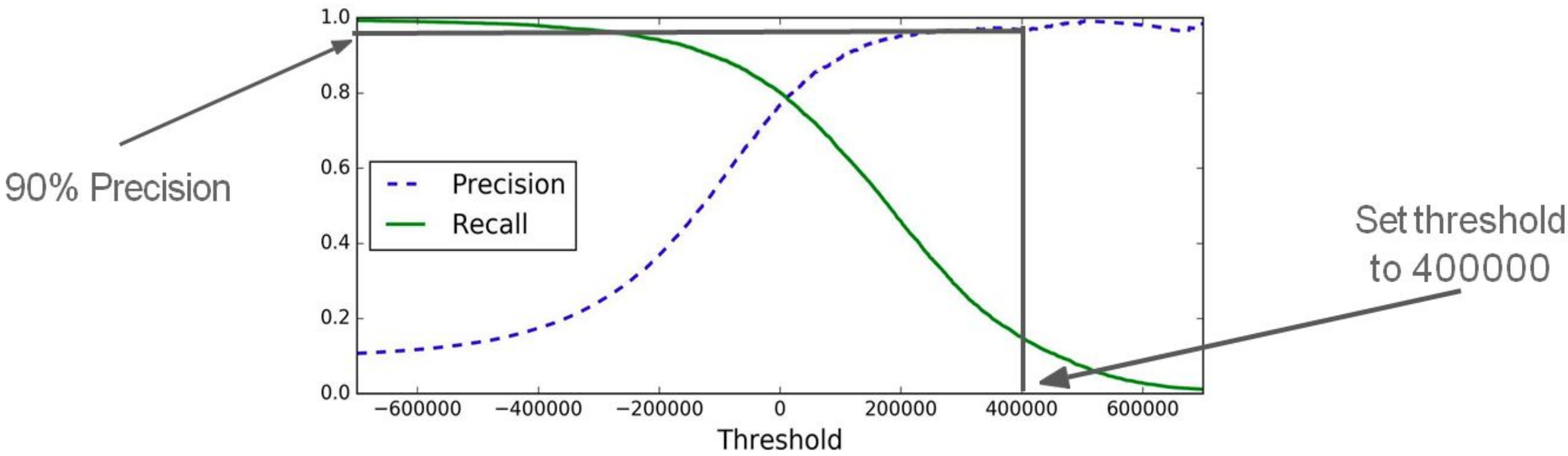
Precision and recall versus the decision threshold

Precision / Recall Curve



Which threshold value to use?

Precision / Recall Curve

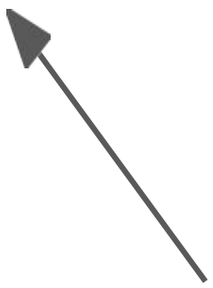


Select the threshold to achieve 90% precision

Building model with desired precision

- So let's say you want to build a classifier with 90% precision
  - Then first select the threshold value which gives you 90% precision
  - Then build classifier using this threshold

```
>>> threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
>>> y_train_pred_90 = (y_scores >= threshold_90_precision)
```



This classifier will give 90% precision

Building model with desired precision

- Verify it

```
>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
```

```
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

Boss

I want a model with 99% precision

ML Engineer

Boss

At what recall? :)

ML Engineer

Revew of Precision/ Recall Tradeoff

- The user can subsequently set the threshold and obtain the classification by a simple comparison.

```
>>> threshold = 200000
>>> y_some_digit_pred =                    > threshold)
(y_scores
>>> y_some_digit_pred
array([False],
      dtype=bool)
```

- Hence, by selecting an appropriate threshold, the user can obtain the desired precision. However, the best precision may not have the best recall.

- Another way to select a good precision/recall trade-off is to plot precision directly against recall, as shown in figure.

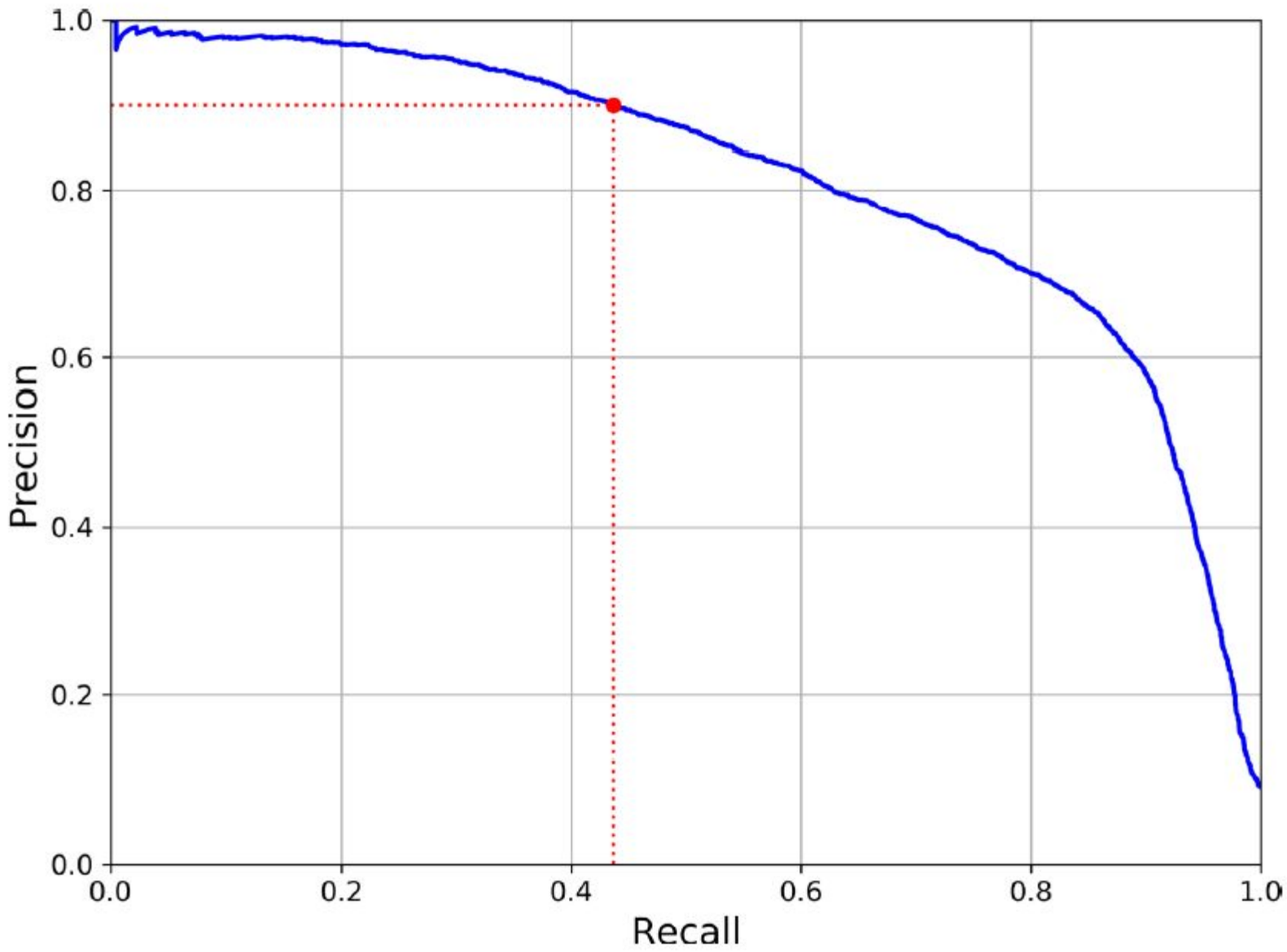


Figure 3-5. Precision versus recall

We see that precision really starts to fall sharply around 80% recall.
We probably want to select a precision/recall trade-off just before that drop—for example, at around 60% recall. But of course, the choice depends on your project.

Suppose you decide to aim for 90% precision.
We look the first plot and search for the lowest threshold that gives you at least 90% precision.
(np.argmax() will give you the first index of the maximum value, which in this case means the first True value):

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816
```

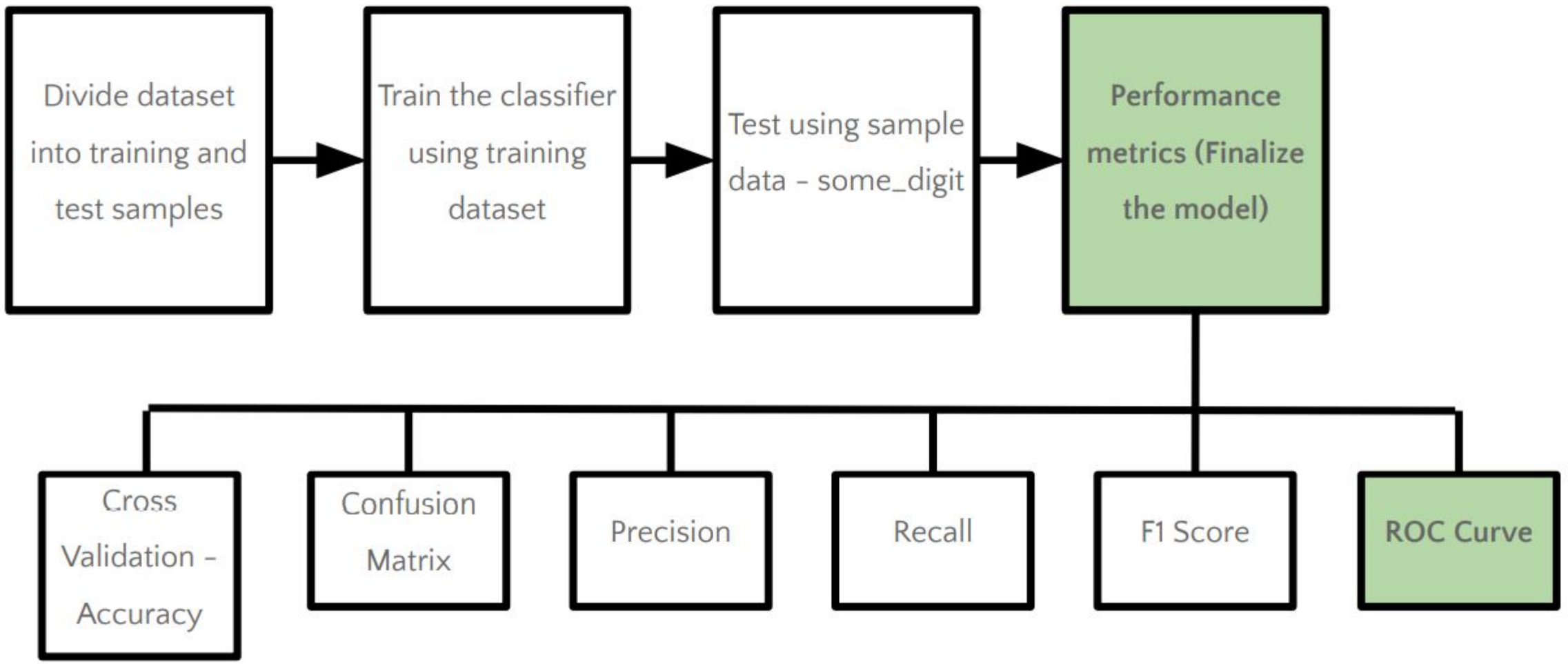
- To make predictions (on the training set for now), instead of calling the classifier's predict() method, you can run this code:

```
y_train_pred_90 = (y_scores >= threshold_90_precision)
```

Let's check these predictions' precision and recall:

```
>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data - some_digit
Performance metrics (Finalize the model)
Cross Validation - Accuracy
Confusion Matrix
Precision
Recall
F1 Score
ROC Curve

THE ROC CURVE

- The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers.
- the ROC curve plots the true positive rate (another name for recall) against the false positive rate (FPR).

- ROC Curve Similar to F1 score but uses a different metric
- Uses True Positive Rate (TPR) = Recall = TP / (TP + FN)
- False Positive Rate (FPR) = FP / (FP + TN)
  = 1 - True Negative Rate (TNR)
- FPR is the ratio of negative instances that are incorrectly classified as positive.
- True Negative Rate (TNR) = TN / (FP + TN)
- The TNR is also called specificity.
- Receiver Operating Characteristics (ROC) plots: TPR versus FPR

Performance measures - ROC Curve

- ROC curve plots recall versus 1-specificity.

To plot the ROC curve, you first use the roc_curve() function to compute the TPR and FPR for various threshold values:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

Then you can plot the FPR against the TPR using Matplotlib.

```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal
    [...] # Add axis labels and grid

plot_roc_curve(fpr, tpr)
plt.show()
```

Performance measures - ROC Curve

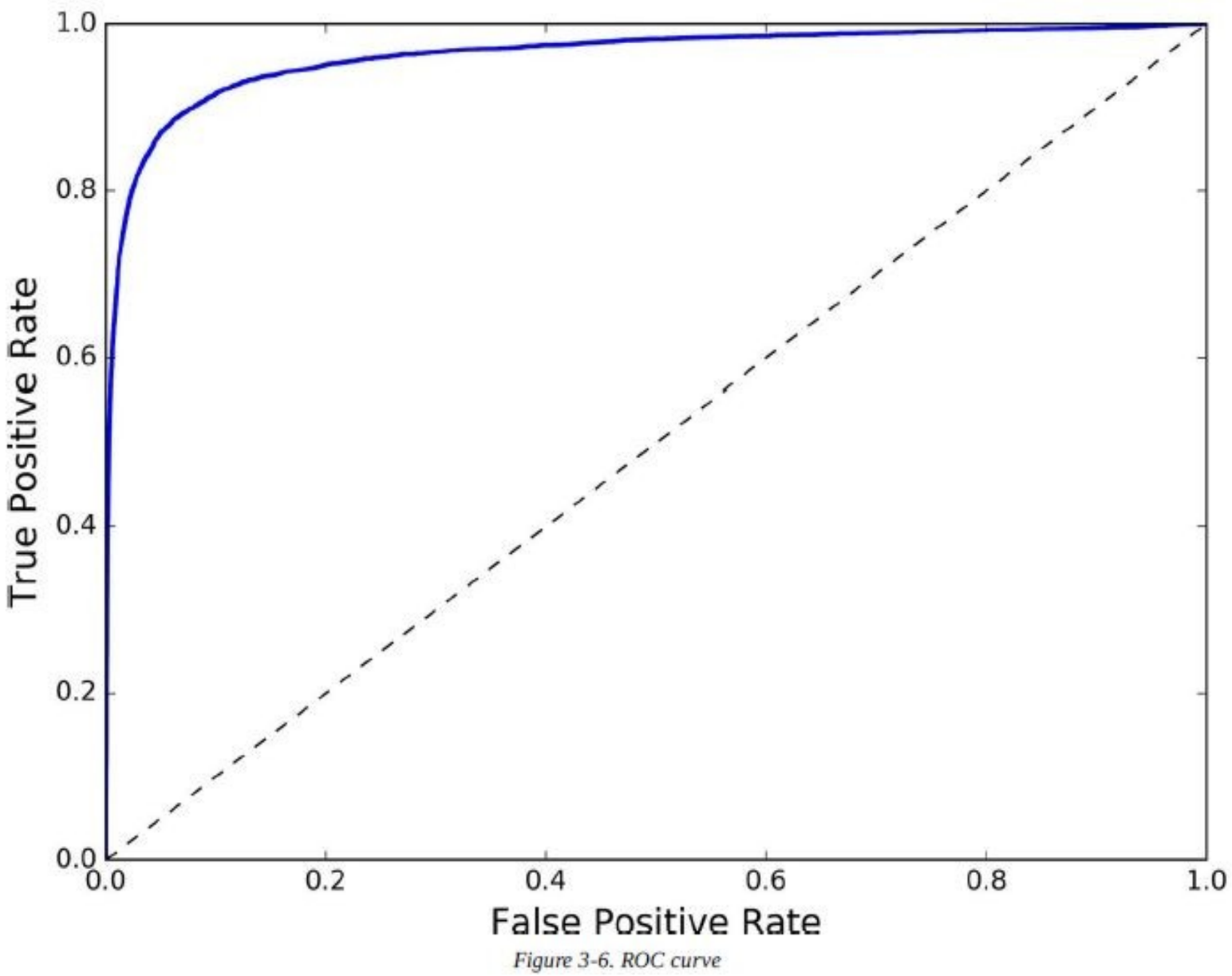
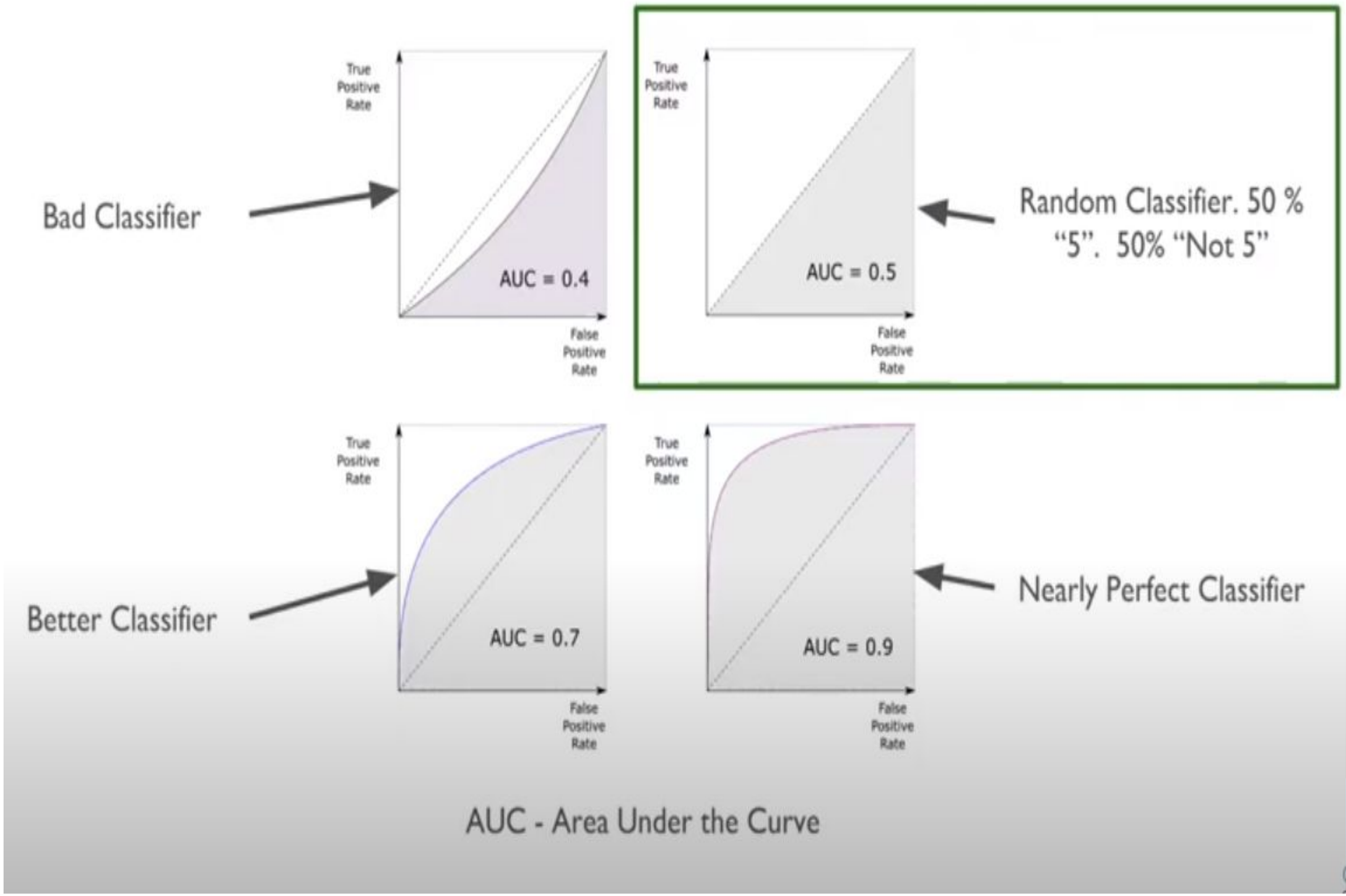


Figure 3-6. ROC curve

Dotted line = purely random classifier


Bad Classifier
True Positive Rate
False Positive Rate
AUC = 0.4
True Positive Rate
False Positive Rate
AUC = 0.5
Random Classifier. 50 % "5". 50% "Not 5"
Better Classifier
True Positive Rate
False Positive Rate
AUC = 0.7
True Positive Rate
False Positive Rate
AUC = 0.9
Nearly Perfect Classifier
AUC - Area Under the Curve

Performance measures - ROC Curve

- Higher the recall (TPR True Positive Rate), higher is the FPR

  (False Positive Rate)

- Dotted line = purely random classifier
- Good classifier stays away from the dotted line towards top-left corner
- A perfect classifier shall have a ROC Area Under the Curve (AUC) equal

  to 1 whereas a purely random classifier shall have ROC AUC = 0.5.

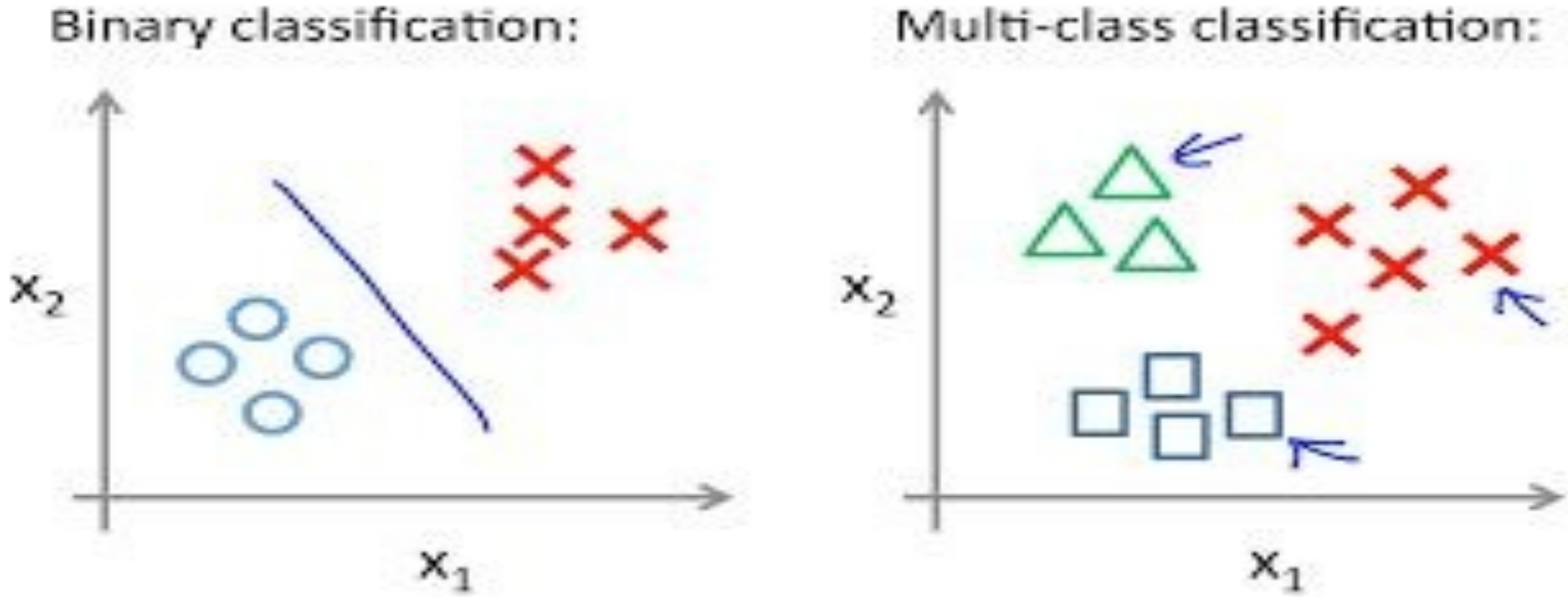
Scikit-Learn provides a function to compute the ROC AUC:

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

Review of the ML process - Binary Classifier

- Divide the dataset into training and test samples
- Train the binary classifier
- Choose the appropriate metric for the task (recall, precision, F1, ROC)
- Select the precision/ recall tradeoff that fits the needs
- Compare various models using ROC curves and ROC AUC curves

Multiclass Classification(MultiNomial classification)



- Binary classifiers distinguish between two classes
- Algorithm such as Logistic Regression or Support Vector Machine classifiers are strictly binary classifiers.
- While multi-class classifiers (also called multinomial classifiers) can distinguish between more than two classes.
- Algorithm such as SGD classifiers, Random Forest classifiers, and naive Bayes classifiers can handle multiple classes.

Multiclass classification

Email foldering/tagging: Work, Friends, Family, Hobby

$y=1$ $y=2$ $y=3$ $y=4$

Medical diagrams: Not ill, Cold, Flu

$y=1$ 2 3

Weather: Sunny, Cloudy, Rain, Snow

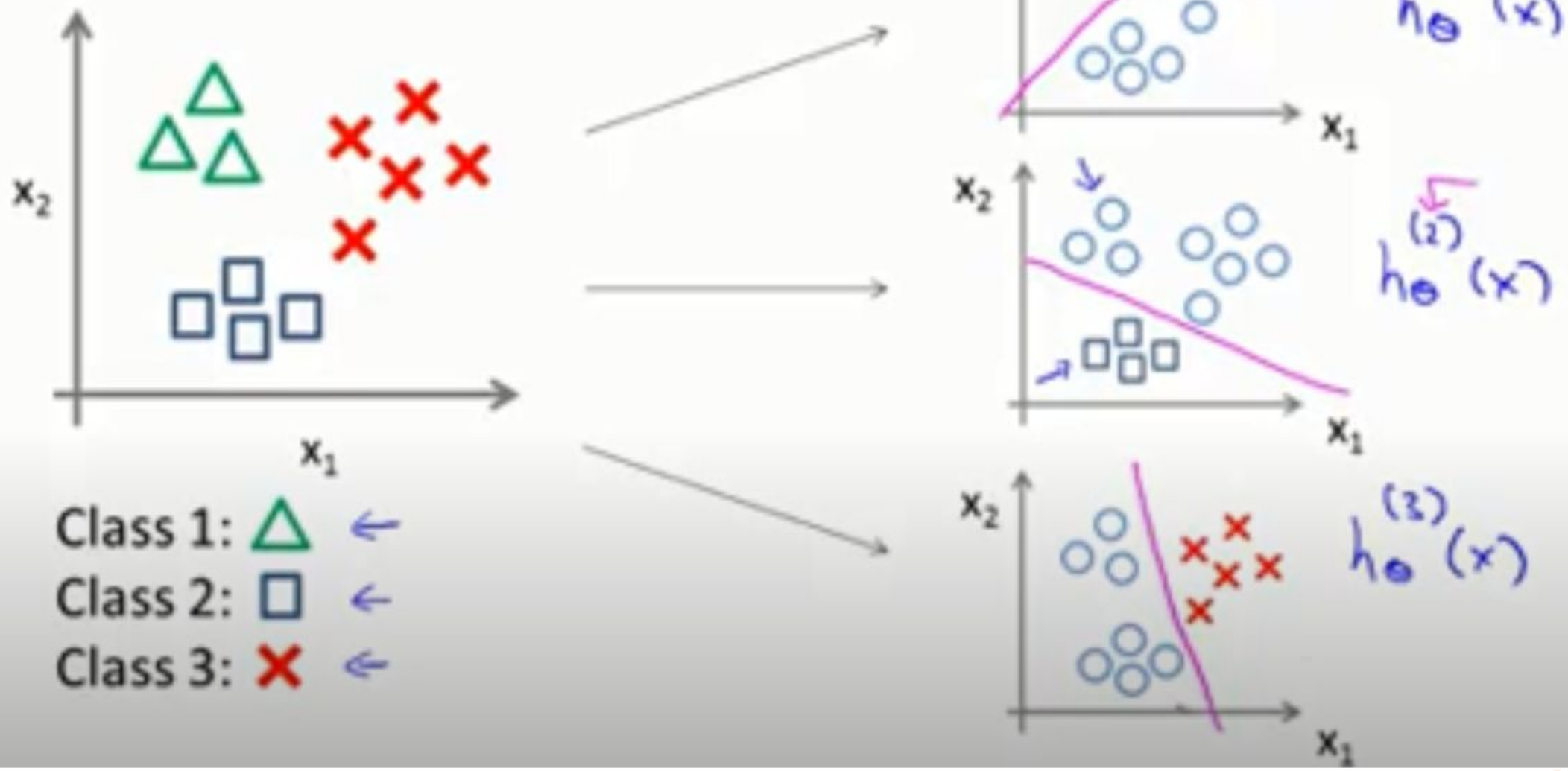
$y=1$ 2 3 4

Multiclass Classification

The various strategies that we can use to perform multiclass classification with multiple binary classifiers.

I. **One-versus-all (OvA) strategy** also called **one-versus-the-rest(OvR)** -
   for example,
   a. For eg. to classify the digit images into 10 classes (from 0 to 9) one way is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).
   b. Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score.

One-vs-all (one-vs-rest):



Multiclass Classification

0	1	2	3	4

1 vs All classifier

0	1	2	3	4

2 vs All classifier

0	1	2	3	4

3 vs All classifier

0	1	2	3	4

4 vs All classifier

0	1	2	3	4

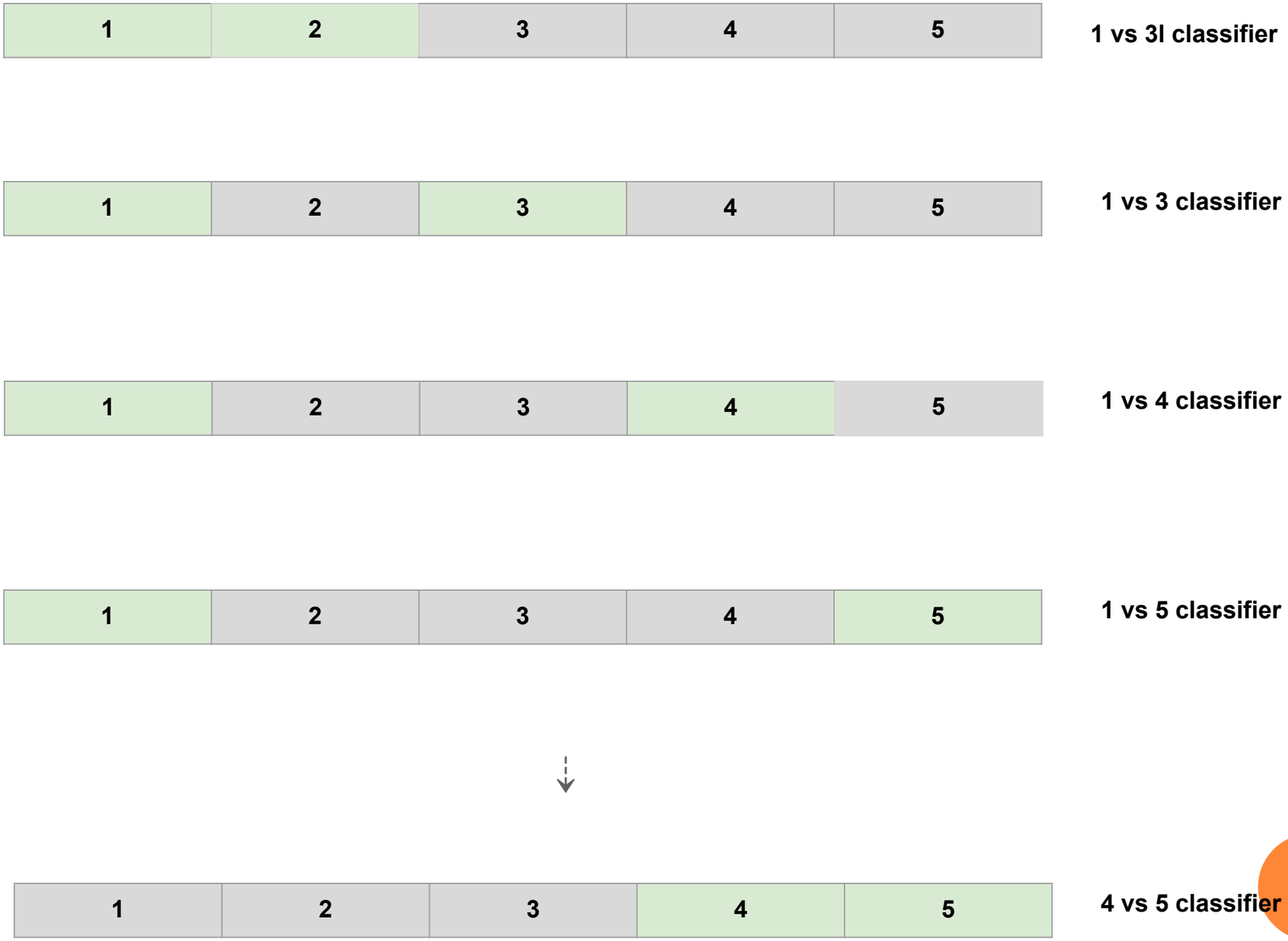
5 vs All classifier

Multiclass Classification

2. One-versus-one (OvO) strategy

a. This is another strategy in which we train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on.

b. If there are N classes, you need to train $N \times (N - 1) / 2$ classifiers.


1
2
3
4
5
1 vs 3l classifier
1
2
3
4
5
1 vs 3 classifier
1
2
3
4
5
1 vs 4 classifier
1
2
3
4
5
1 vs 5 classifier
1
2
3
4
5
4 vs 5 classifier

Error Analysis

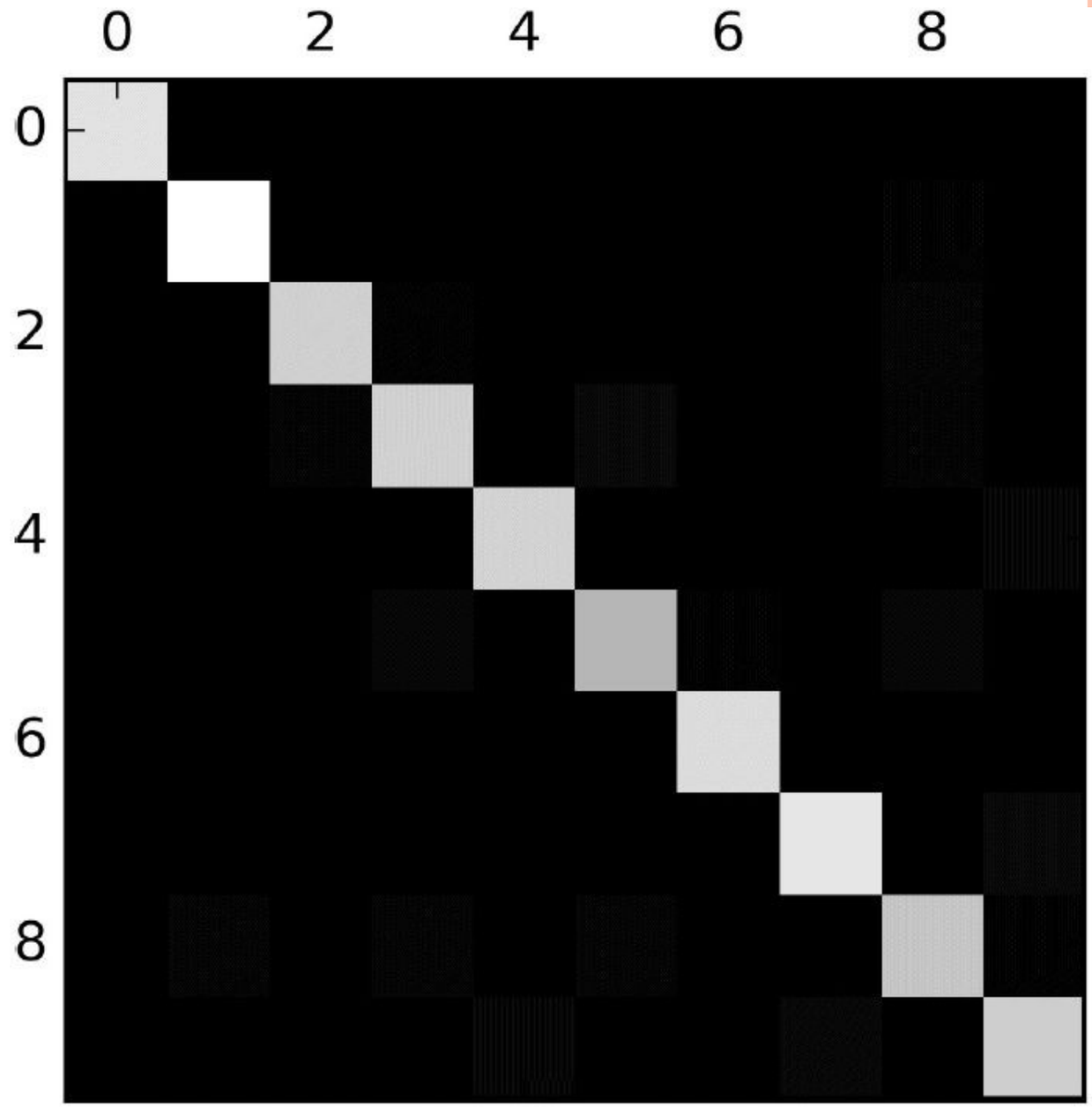
- Once the model (classifier) is identified, it can be improved by analyzing the types of errors it makes
- For the previous multiclass classification example of classifying images of digits into digit labels, it can be done by observing the confusion matrix and plotting it on the graph

```
>>> y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
>>> conf_mx = confusion_matrix(y_train, y_train_pred)
>>> conf_mx
array([[5578,    0,   22,    7,    8,   45,   35,    5,  222,    1],
       [   0, 6410,   35,   26,    4,   44,    4,    8,  198,   13],
       [  28,   27, 5232,  100,   74,   27,   68,   37,  354,   11],
       [  23,   18,  115, 5254,    2,  209,   26,   38,  373,   73],
       [  11,   14,   45,   12, 5219,   11,   33,   26,  299,  172],
       [  26,   16,   31,  173,   54, 4484,   76,   14,  482,   65],
       [  31,   17,   45,    2,   42,   98, 5556,    3,  123,    1],
       [  20,   10,   53,   27,   50,   13,    3, 5696,  173,  220],
       [  17,   64,   47,   91,    3,  125,   24,   11, 5421,   48],
       [  24,   18,   29,   67,  116,   39,    1,  174,  329, 5152]])
```

Error Analysis

- Looks fairly good since most images are on the diagonal

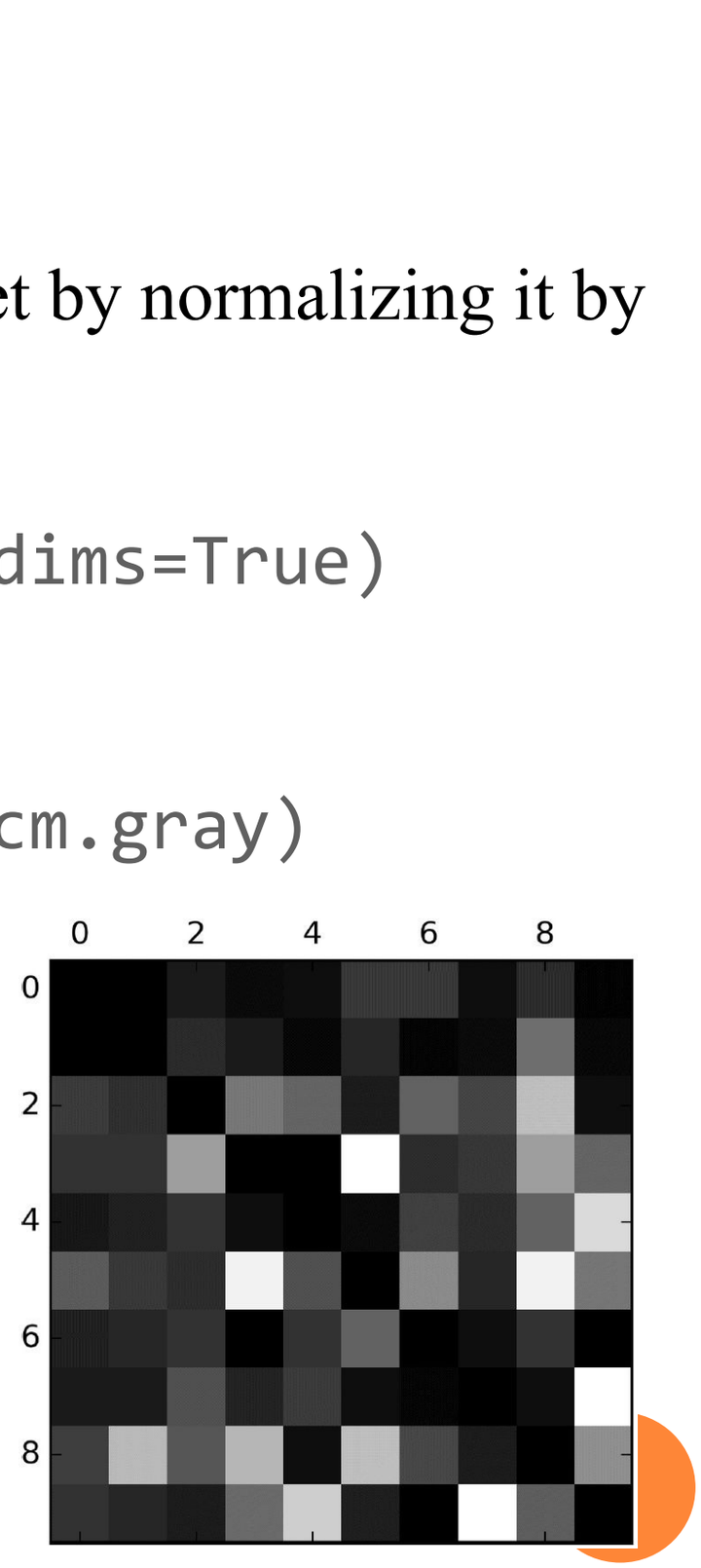
- 5s looks darker than other digits
  - Q. What are the possible reasons?
    - Fewer 5s in the dataset
    - Classifier does not perform well on 5s
    - Both **(Ans)**



Error Analysis

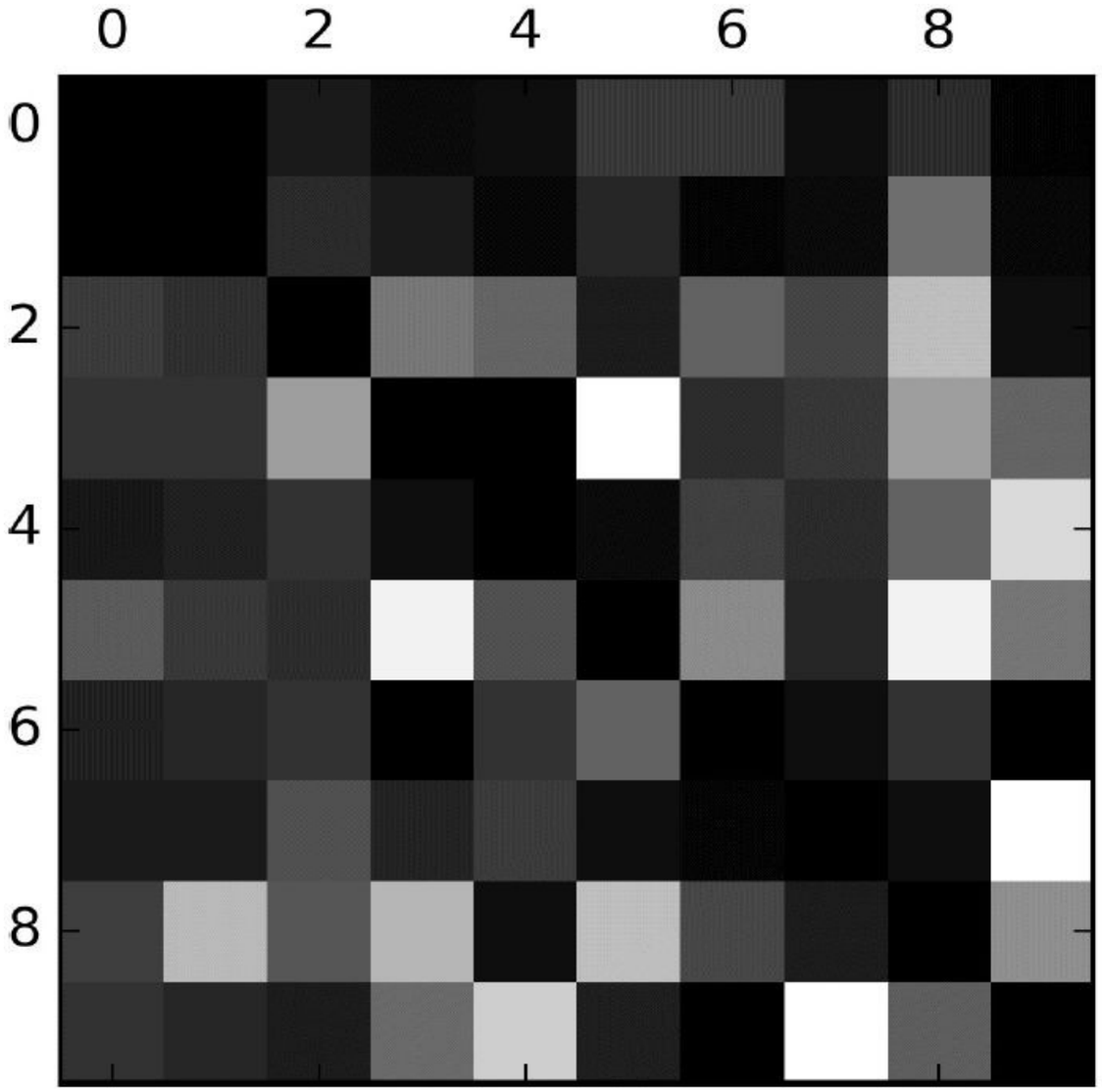
- We remove the possibility of fewer 5s in the dataset by normalizing it by the number of samples in each dataset

```
>>> row_sums = conf_mx.sum(axis=1, keepdims=True)
>>> norm_conf_mx = conf_mx / row_sums
>>> np.fill_diagonal(norm_conf_mx, 0)
>>> plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
>>> plt.show()
```



Observations:

1. 8 and 9 columns are bright:
    1. many digits misclassified as 8 and 9
    2. many 8 and 9 misclassified as other digits
3. (3,5) and (5,3) are abnormally bright: confusion between 3 and 5

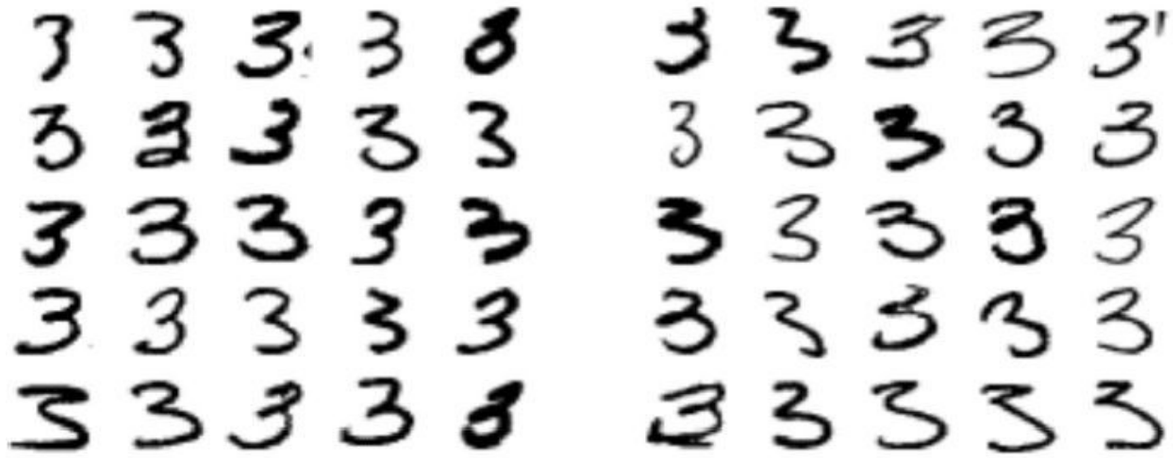


Possible solutions:

1. **Gather more training data for 8s and 9s**
2. **Pre-process images to make certain features stand out more**
3. **Analyzing individual errors**

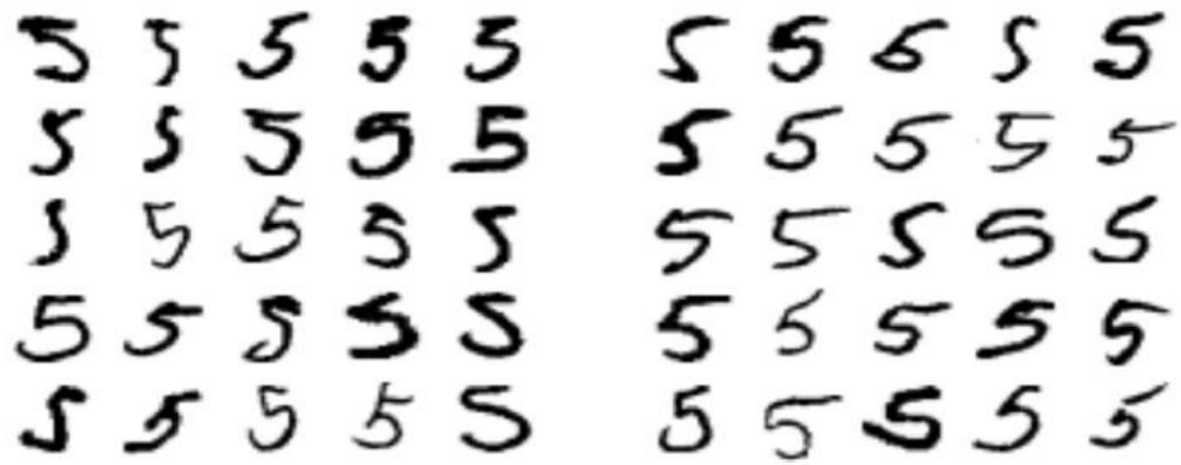
Analyzing individual errors (plotting 3s and 5s)

Actual 3,
Predicted 3



Actual 3,
Predicted 5

Actual 5,
Predicted 3



Actual 5,
Predicted 5

Some written badly, most are errors. What can we do about the errors?

Observation:

- Some digits are written badly while most are errors in classification (between 3s and 5s).
- SGDClassifier assigns a weight per class to each pixel and calculate the total score for a particular class
- Since 3s and 5s differ by a few pixels, the classifier is easily confused - Main difference between 3 and 5 is the position of the straight line

Possible Solution

- Classifier is quite sensitive to image shifting and rotation
- Solution: Preprocess the image to ensure that they are well-centered and not rotated