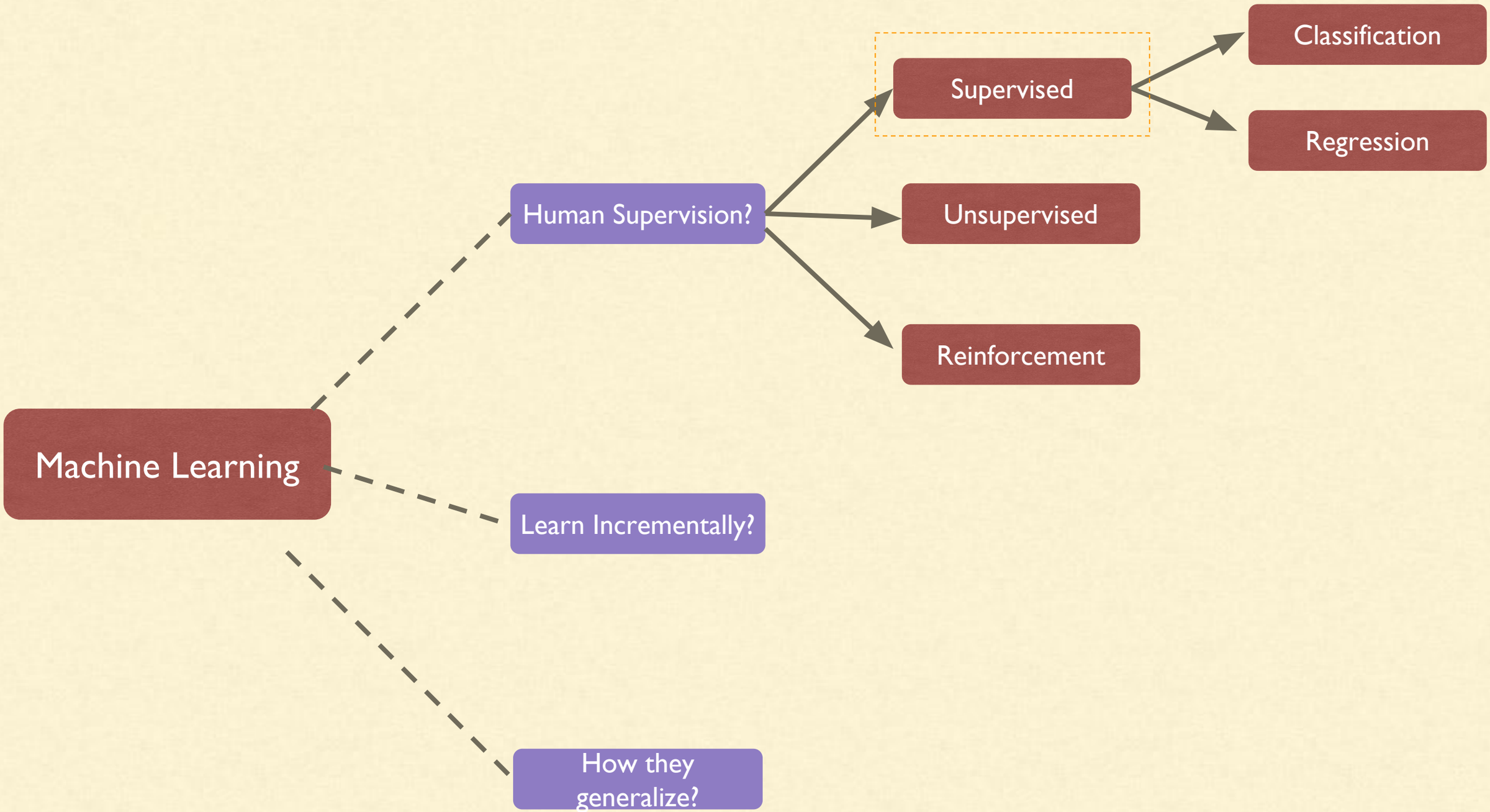
Classification

What is Classification ?

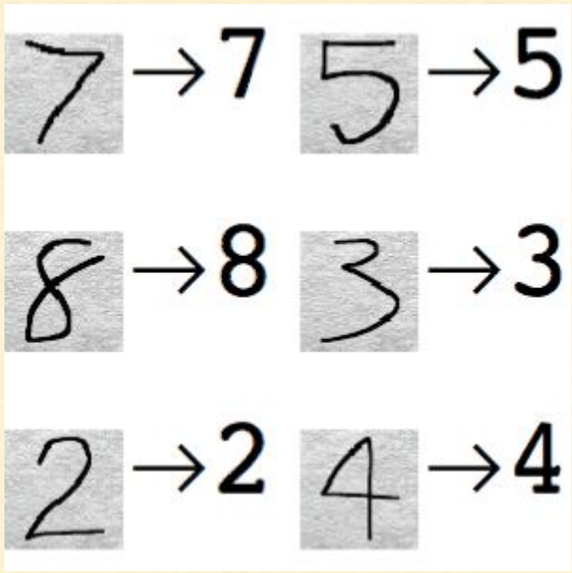


What is Classification ?

Identifying to which label something belongs to

5

Not 5



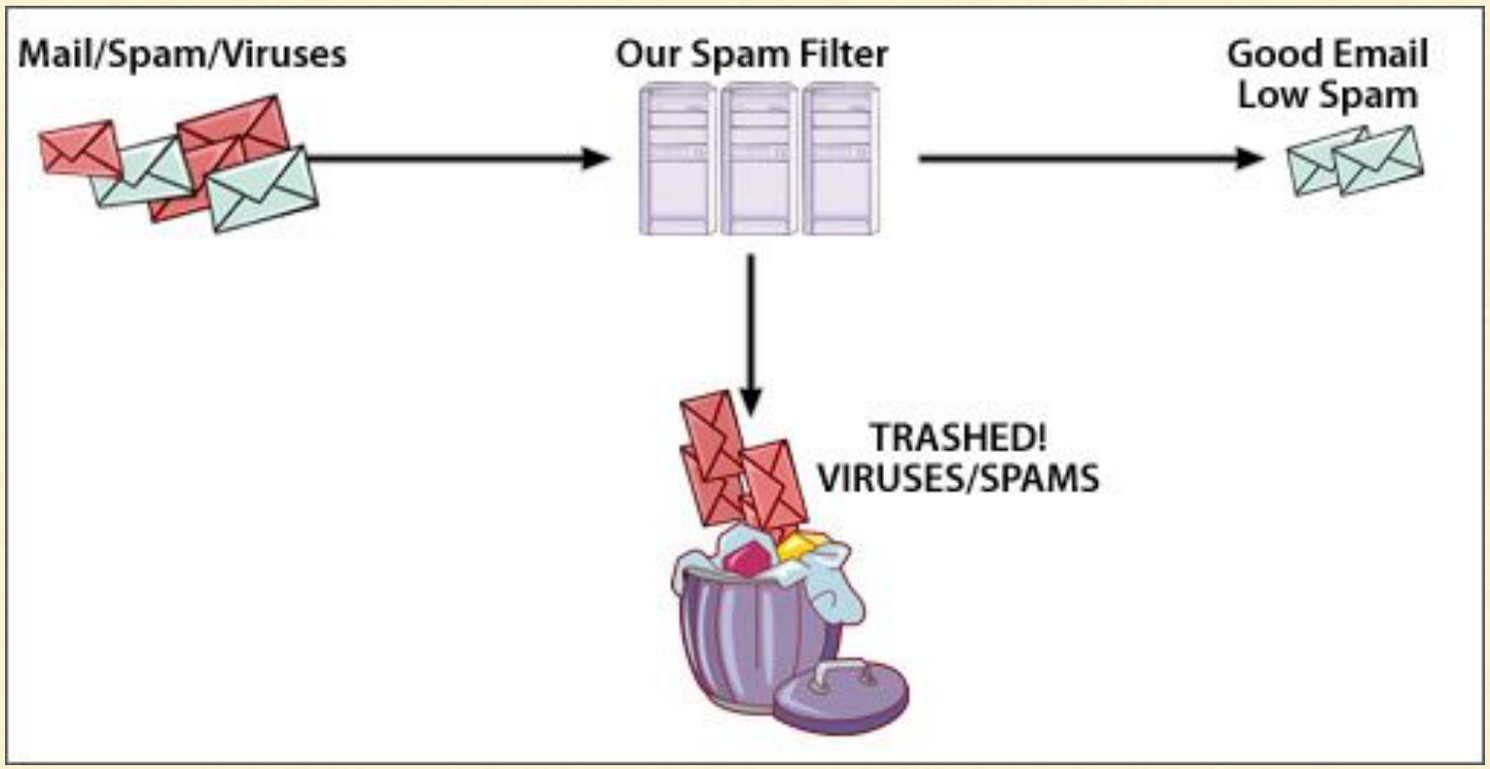
What is Classification ?

Classification is the task of

- Identifying to which category a new observation belongs
- On the basis of observations whose category is known

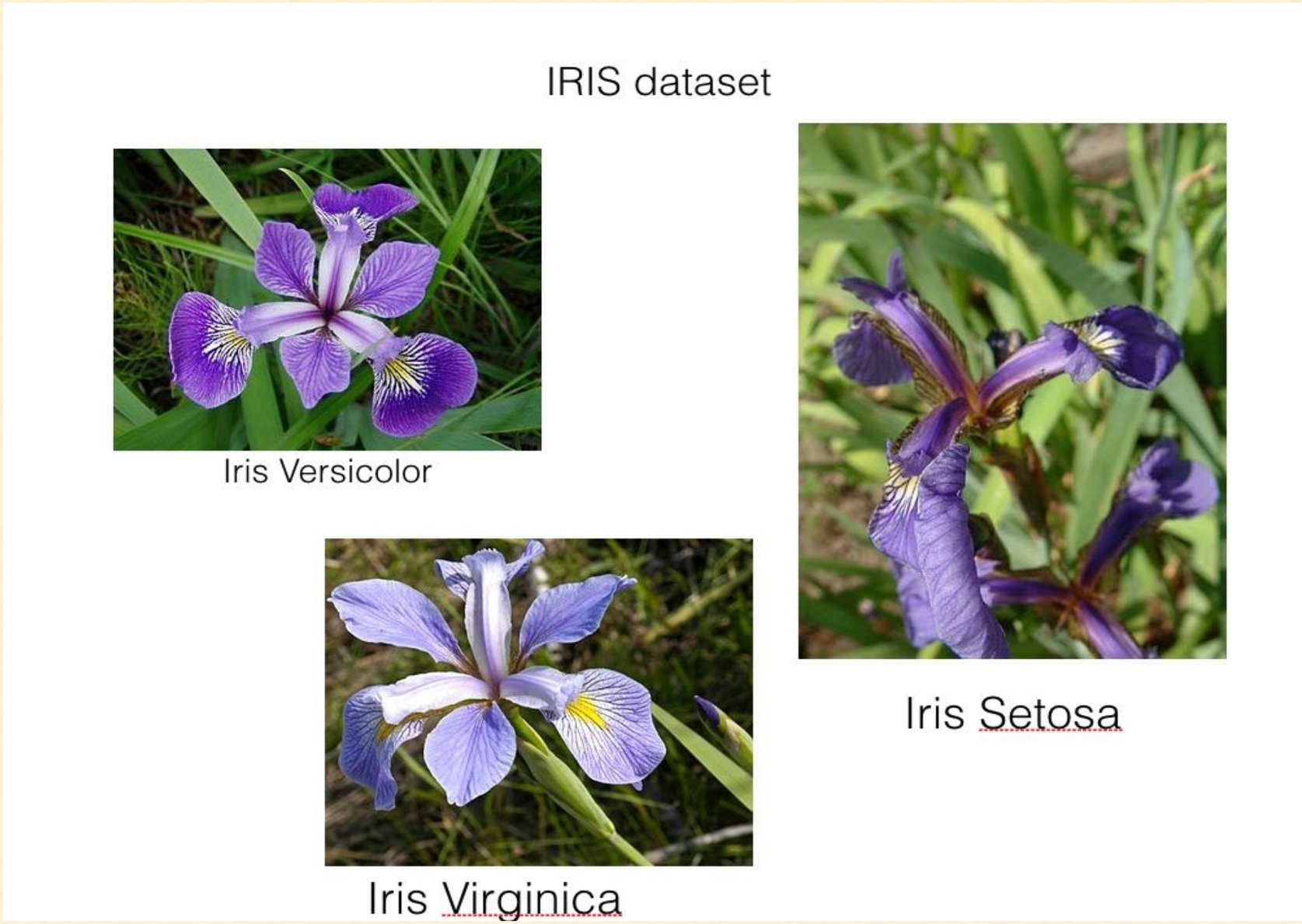
Examples of Classification

- Classifying emails as spam or not spam



Examples of Classification

- Classifying flowers of a particular species like the Iris Dataset

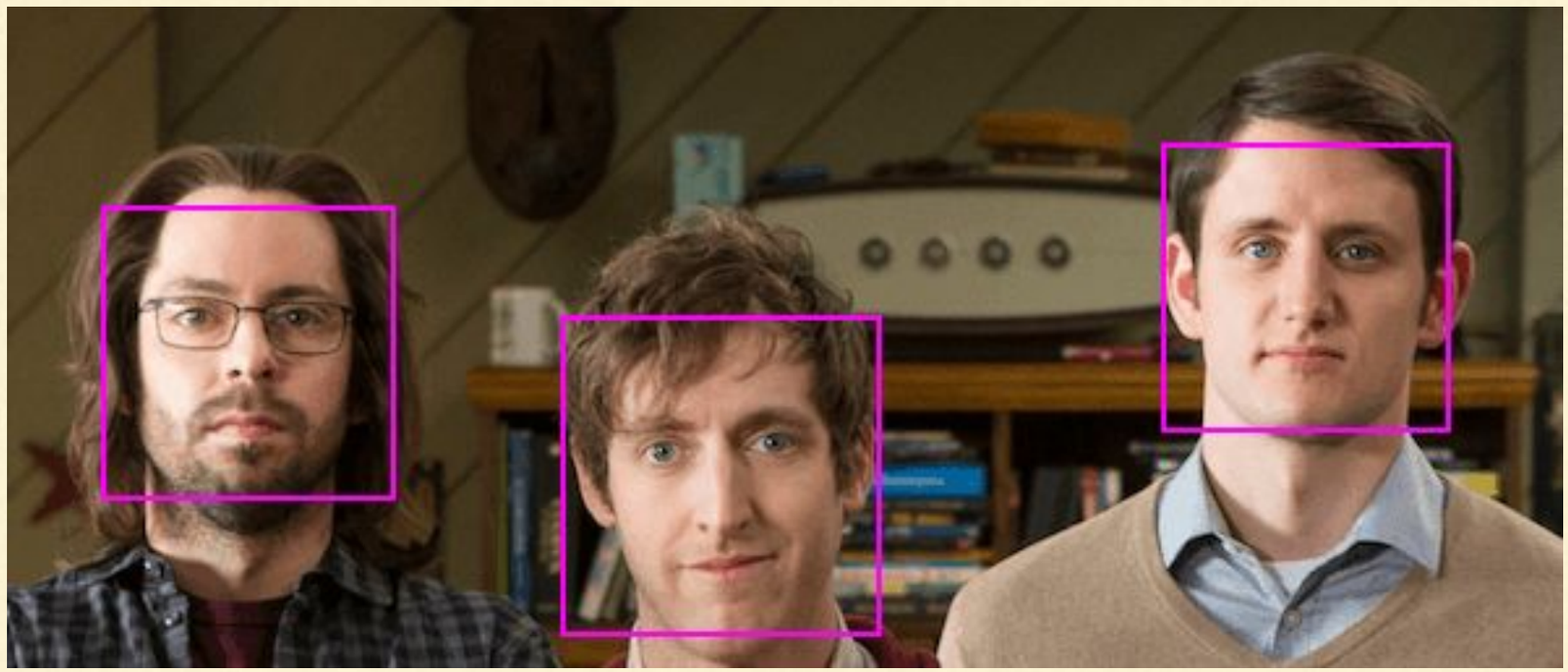


Examples of Classification

- Classifying a credit card transaction as fraudulent or genuine

Examples of Classification

- Face recognition

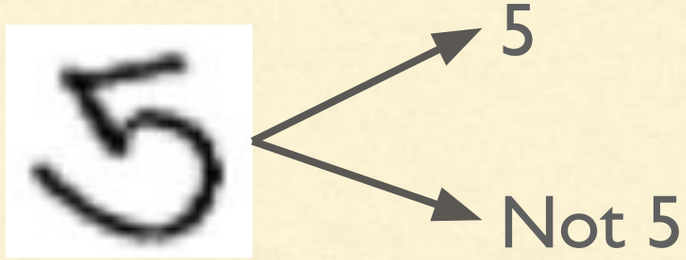


Goal - Project

The goal of this session is to classify handwritten digits

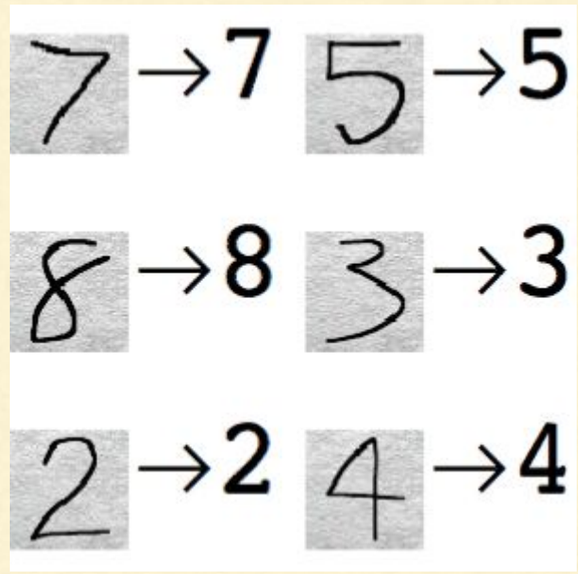
Binary and Multiclass Classification

Binary Classification



Classification is done between two classes

Multiclass Classification



Classification is done between multiple classes

Dataset

MNIST Dataset

Modified National Institute of Standards and Technology

- In this problem, we will use MNIST dataset
  - Set of 70, 000 small images of
  - Digits handwritten by high school students and
  - Employees of the US Census Bureau

Dataset

MNIST Dataset

- Each image is labeled with the digit it represents

Few Digits from the MNIST Dataset

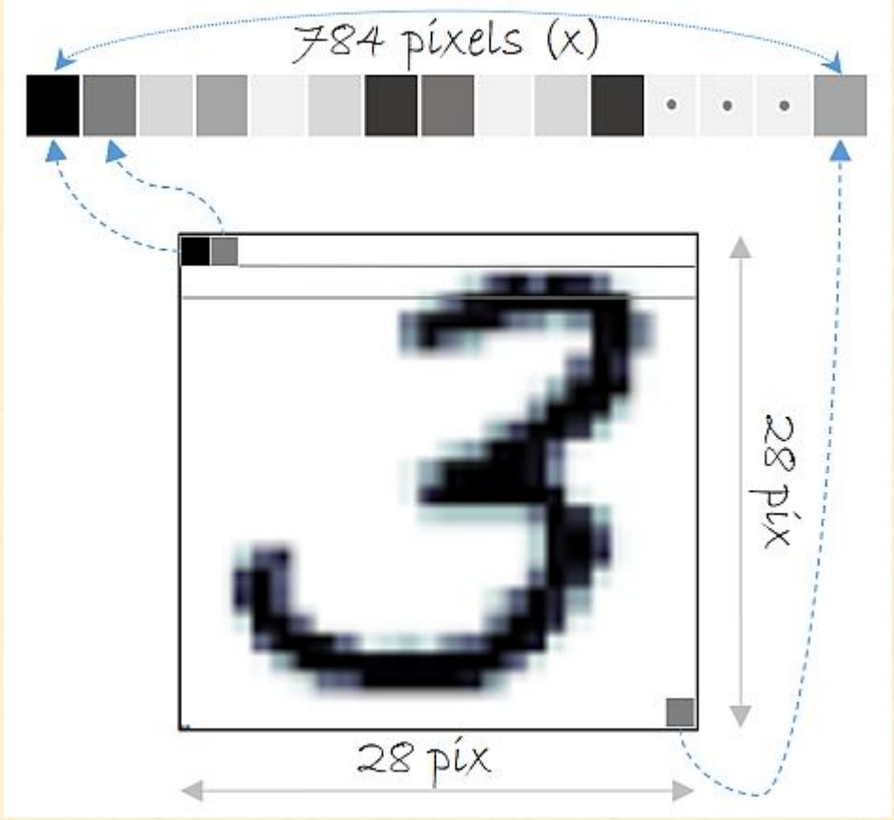
Dataset

MNIST Dataset

- MNIST dataset is also called
  - “Hello World” of Machine Learning
- Whenever people come up with new algorithm
  - They test it on MNIST dataset to see how it performs

Dataset

- Each image
  - 28 X 28 pixels
  - 784 features
- There are 70000 such images making the dataset dimension:
  - 70000 X 784



Fetching MNIST dataset in Scikit-Learn

Fetch the Data

```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1,
cache=True)
>>> X, y = mnist["data"], mnist["target"]
```

Switch to Notebook

Switch to Notebook

Fetching MNIST dataset in Scikit-Learn

Looking at one of the data samples

```
>>> %matplotlib inline
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> some_digit = X[36000]   # Selecting the 36,000th
image.
>>> some_digit_image = some_digit.reshape(28, 28)
>>> plt.imshow(some_digit_image, cmap =
matplotlib.cm.binary, interpolation="nearest")
>>> plt.axis("off")
>>> plt.show()
```

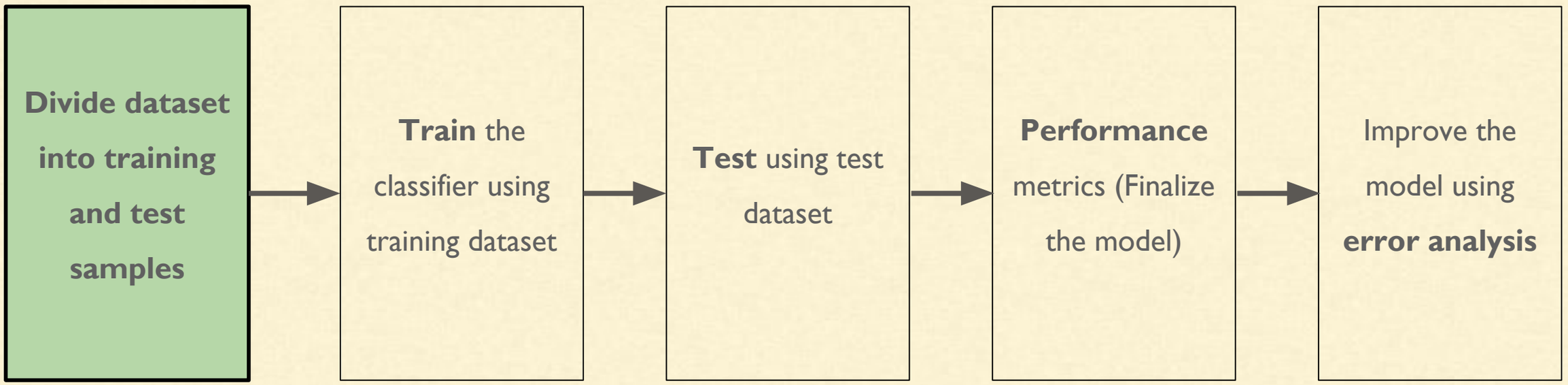
Switch to Notebook

Fetching MNIST dataset in Scikit-Learn

Looking at one of the data samples

Switch to Notebook

Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using test dataset
Performance metrics (Finalize the model)
Improve the model using **error analysis**

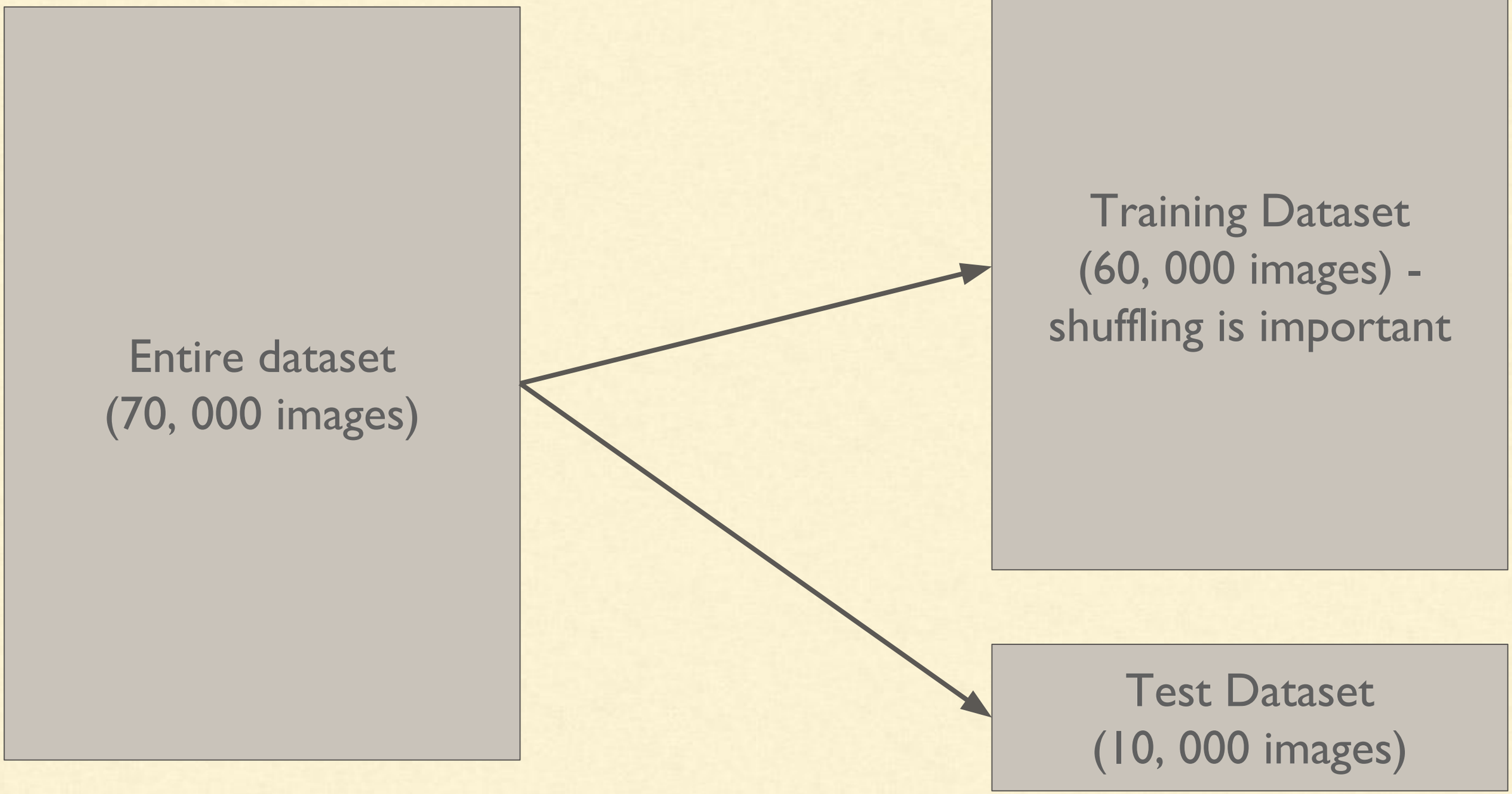
Training and Test dataset

- 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

Training and Test dataset



Dividing dataset into training and test samples

Dividing dataset into training and test in python:

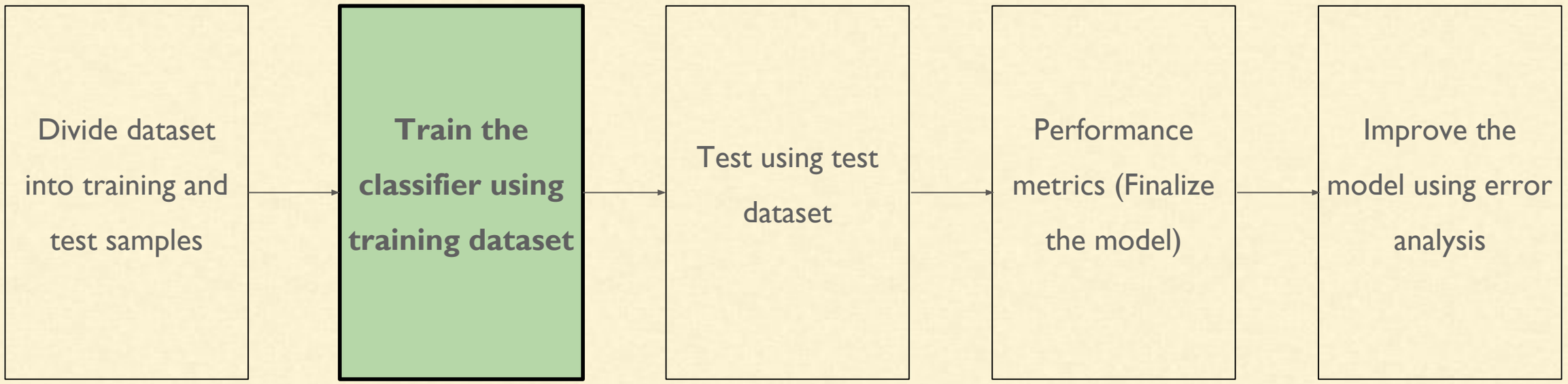
```
>>> X_train, X_test, y_train, y_test = X[:60000],
X[60000:], y[:60000], y[60000:]

>>> import numpy as np
>>> np.random.seed(42)
>>> shuffle_index = np.random.permutation(60000)
>>> X_train, y_train = X_train[shuffle_index],
y_train[shuffle_index]
```

Switch to Notebook

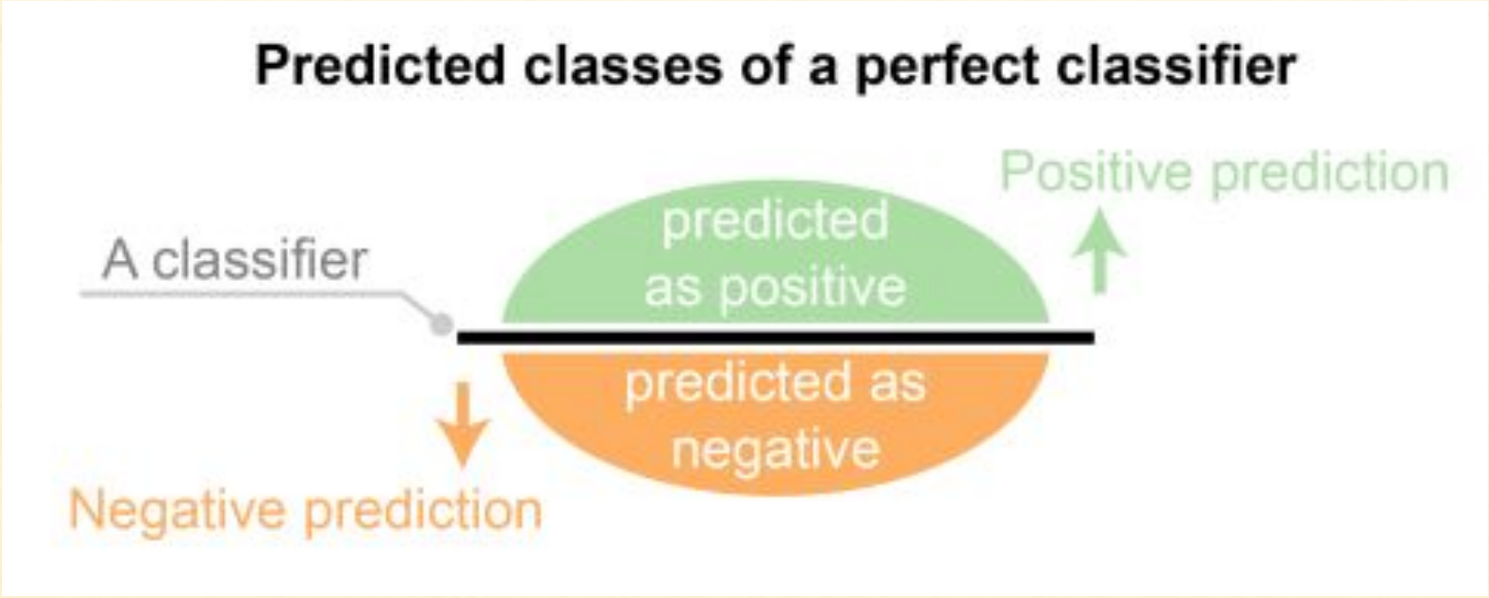
Switch to Notebook

Steps


Divide dataset into training and test samples
Train the classifier using training dataset
Test using test dataset
Performance metrics (Finalize the model)
Improve the model using error analysis

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.



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.

Example:

Input: Image Output: Classification

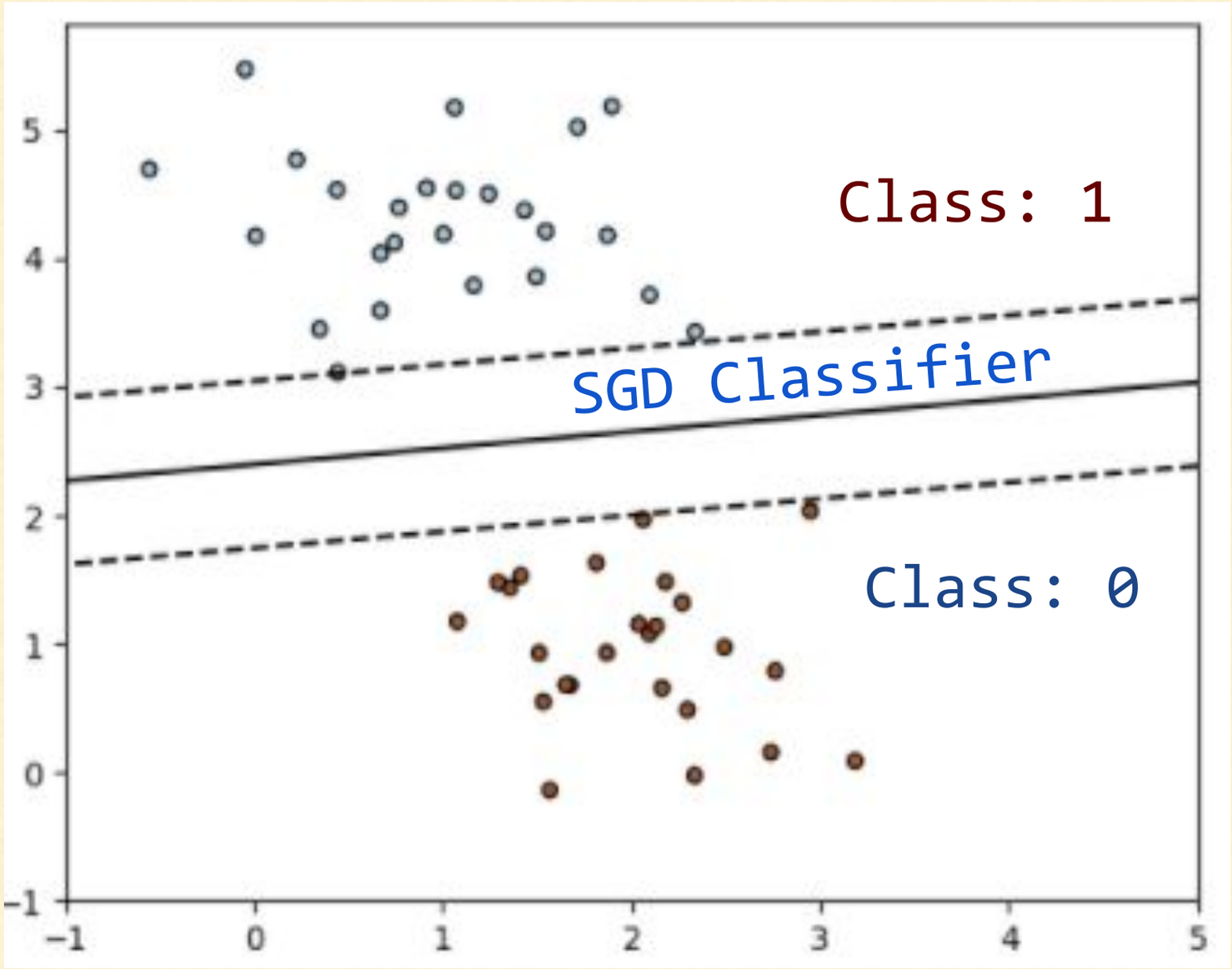
Stochastic Gradient Descent (SGD) Classifier

We are going to use: Stochastic Gradient Descent (SGD) Classifier

Stochastic Gradient Descent (SGD) Classifier

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

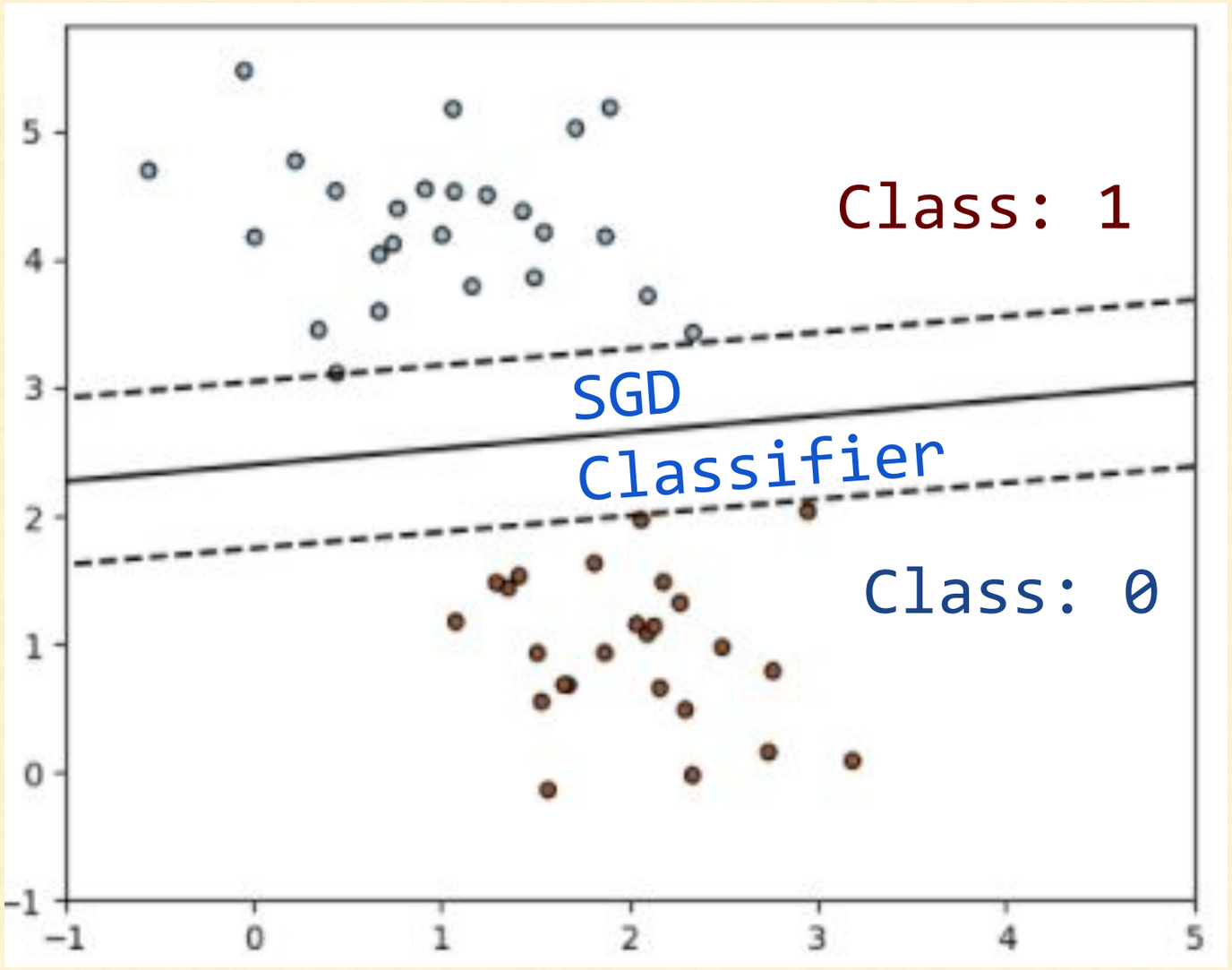
- Trying to estimate the coefficients of the line which can be
- Best-fitted dividing the two categories



Stochastic Gradient Descent (SGD) Classifier

The classifier SGDClassifier implements

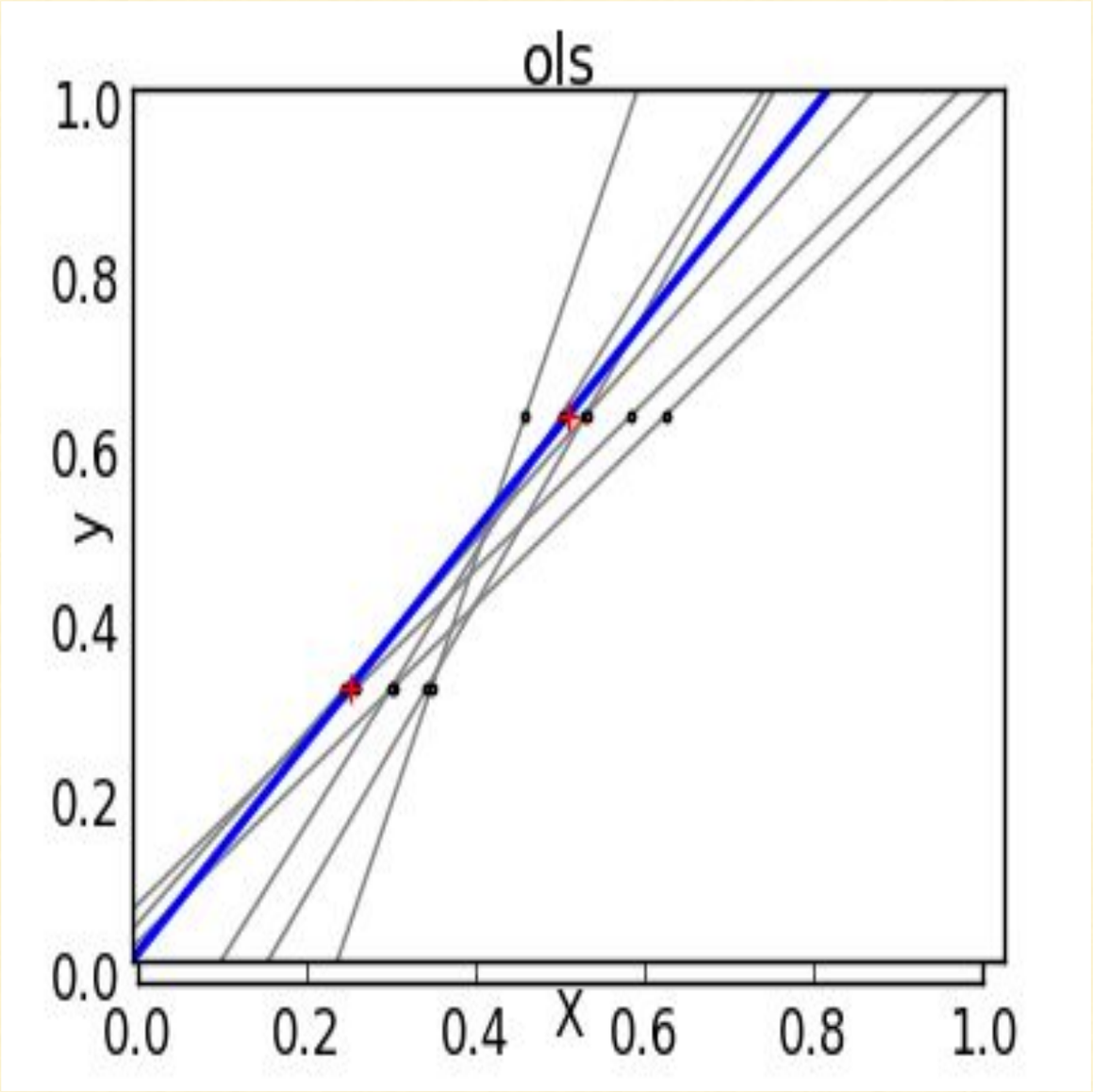
- a plain stochastic gradient descent learning routine
- supports different loss functions and
- penalties for classification



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

Machine Learning - Gradient Descent

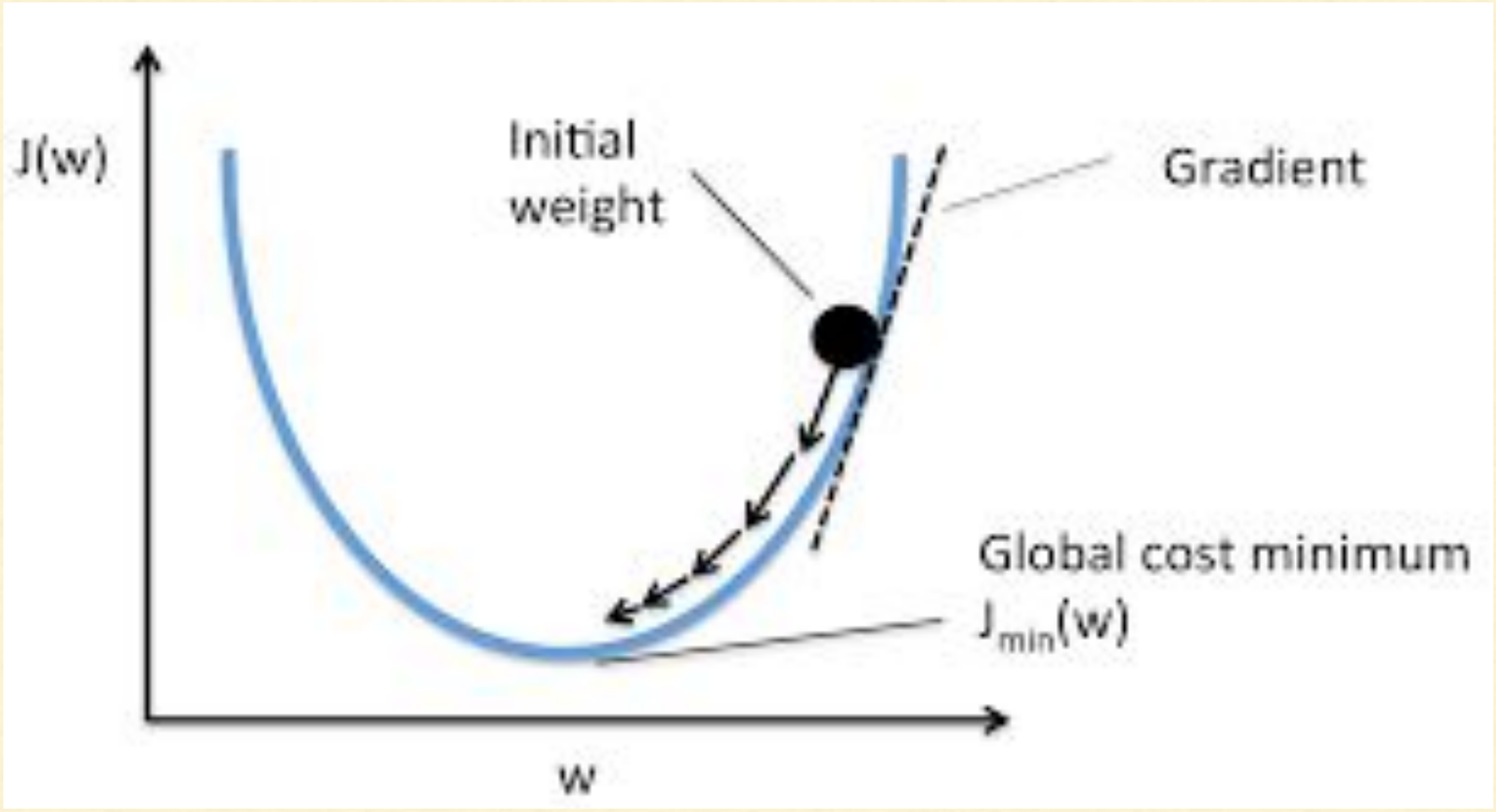


- Instead of trying all lines, go into the direction yielding better results

Machine Learning - Gradient Descent

- Imagine yourself blindfolded on the mountainous terrain
- And you have to find the best lowest point
- If your last step went higher, you will go in opposite direction
- Other, you will keep going just faster

Machine Learning - Gradient Descent



Stochastic Gradient Descent (SGD) Classifier

Example:

- Let Us try with a simple problem
  - Train the SGD Classifier dataset on two points (0,0) and (1,1)
  - Plot the classifier along with the training dataset
  - Check the classifier with a test dataset (2,2)

Stochastic Gradient Descent (SGD) Classifier

Forming the dataset and training the classifier

```
>>> from sklearn.linear_model import SGDClassifier
>>> X1 =
{"xcoord":pd.Series([0,1]),"ycoord":pd.Series([0,1])}
>>> X3 = pd.DataFrame(X1)
>>> y = [0, 1]
>>> clf = SGDClassifier(loss="hinge", penalty="l2")
clf.fit(X3, y)
```

Loss function: hinge for linear classifier

Penalties: L2 = least squares for distance calculation

Stochastic Gradient Descent (SGD) Classifier

- What is hinge loss?
  - A way to measure how badly a classifier is doing
  - We need to minimize this particular variable
  - Hinge loss is particularly used for linear classifiers

Stochastic Gradient Descent (SGD) Classifier

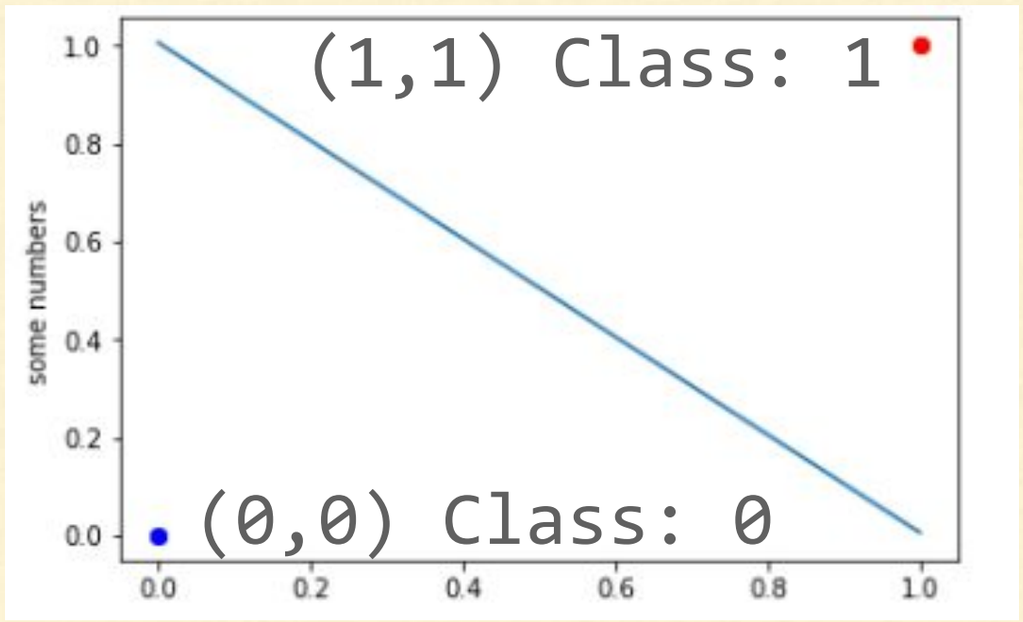
- Example code of using SGD Classifier
  - Result of the fitted classifier - the coefficients and intercept

```
>>> print(clf.coef_)
>>> print(clf.intercept_[0])
[[ 9.91080278  9.91080278]]
-9.97004991017
```

Stochastic Gradient Descent (SGD) Classifier

Plotting the dataset and the classifier for illustration

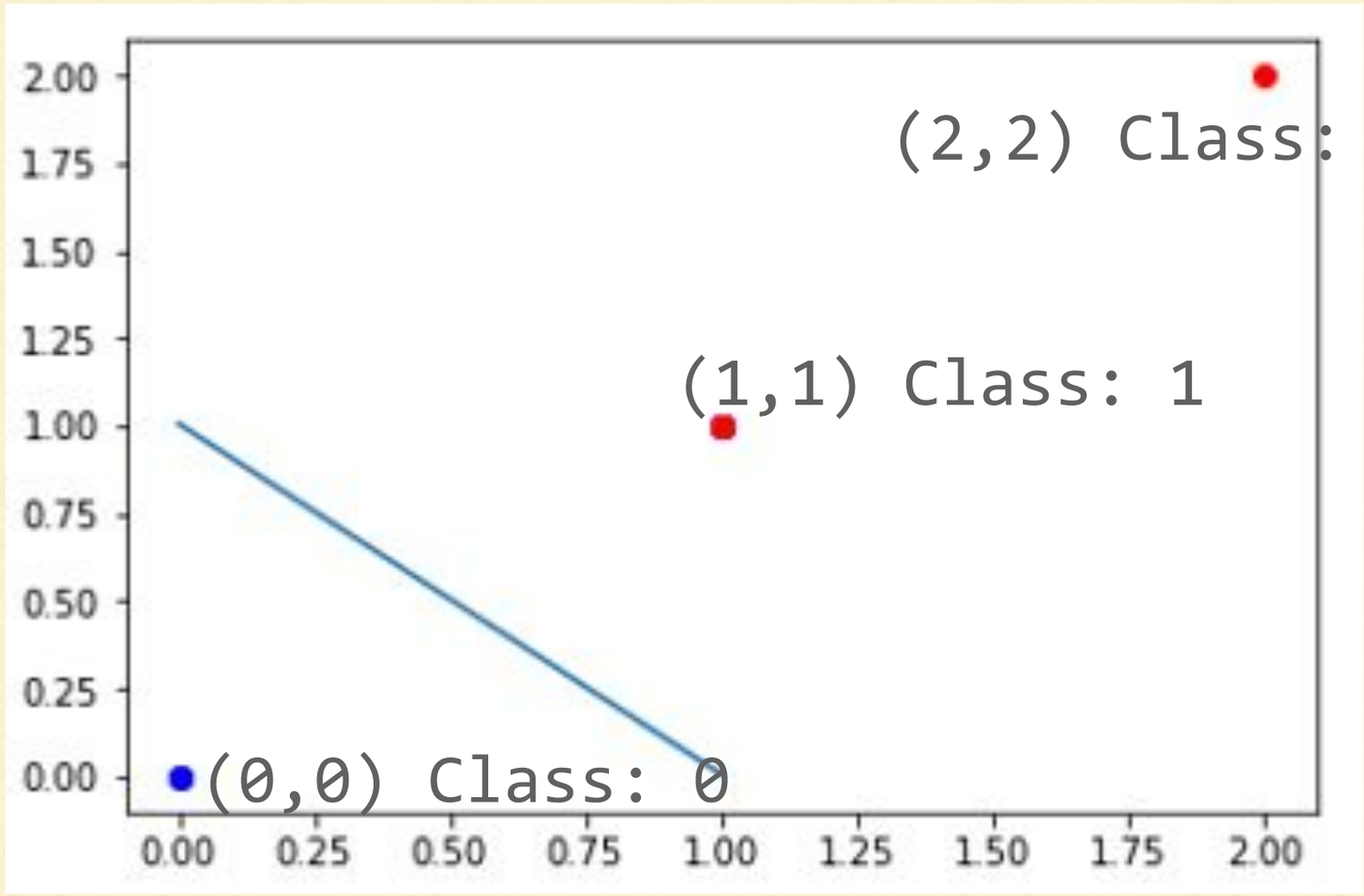
```
>>> import matplotlib.pyplot as plt
>>> plt.plot(X3[0:1],'bo')
>>> plt.plot(X3[1:],'ro')
>>> a=np.linspace(0.,1.,num=11)
>>>
b=(-a*clf.coef_[0,0]-clf.intercept_[0])/clf.coef_[0,1]
>>> plt.plot(a,b)
>>> plt.show()
```



Stochastic Gradient Descent (SGD) Classifier

Testing the classifier with a test data point (2,2)
- Classifier correctly identifies it to belong to class 1

```
>>> print(clf.predict([[2., 2.]]))
[1]
```



Switch to Notebook

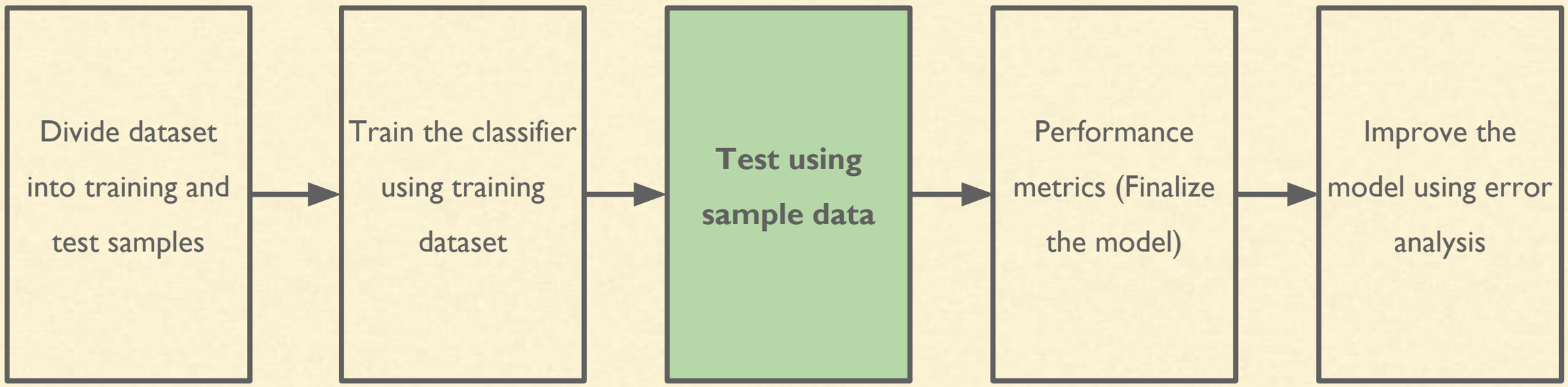
Training SGD Classifier in Scikit Learn

Original problem: '5' and 'Not 5' classifier

```
>>> from sklearn.linear_model import SGDClassifier
>>> sgd_clf = SGDClassifier(random_state=42, max_iter=10)
>>> sgd_clf.fit(X_train, y_train_5)
```

Switch to Notebook

Steps


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

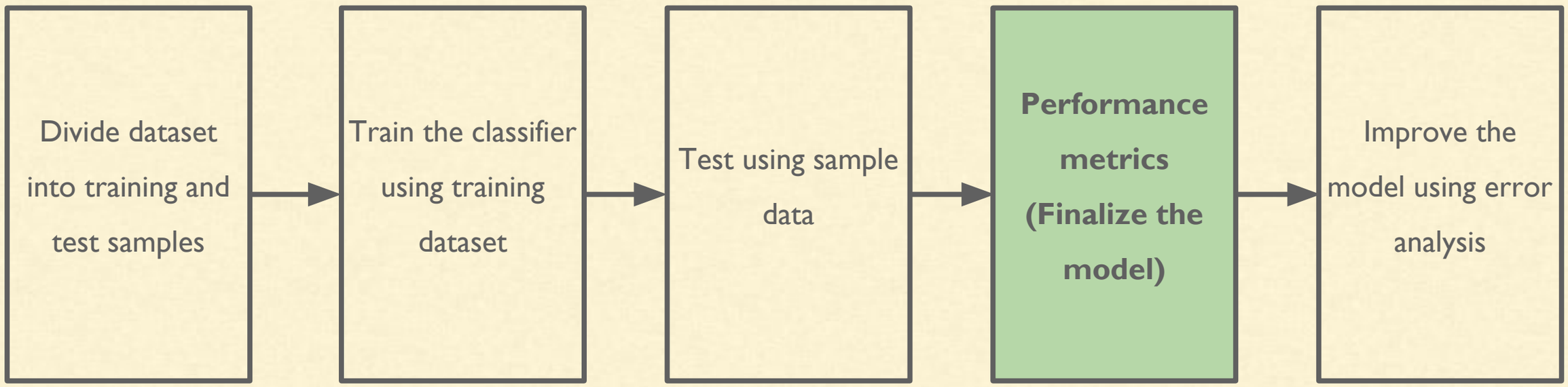
Testing SGD Classifier in Scikit Learn

- 36000th image stores the digit 5 and the classifier correctly classifier it as ‘True’

```
>>> some_digit = X[36000]  # Taking the 36,000th image
>>> sgd_clf.predict([some_digit])
array([True], dtype=bool)
```

Switch to Notebook

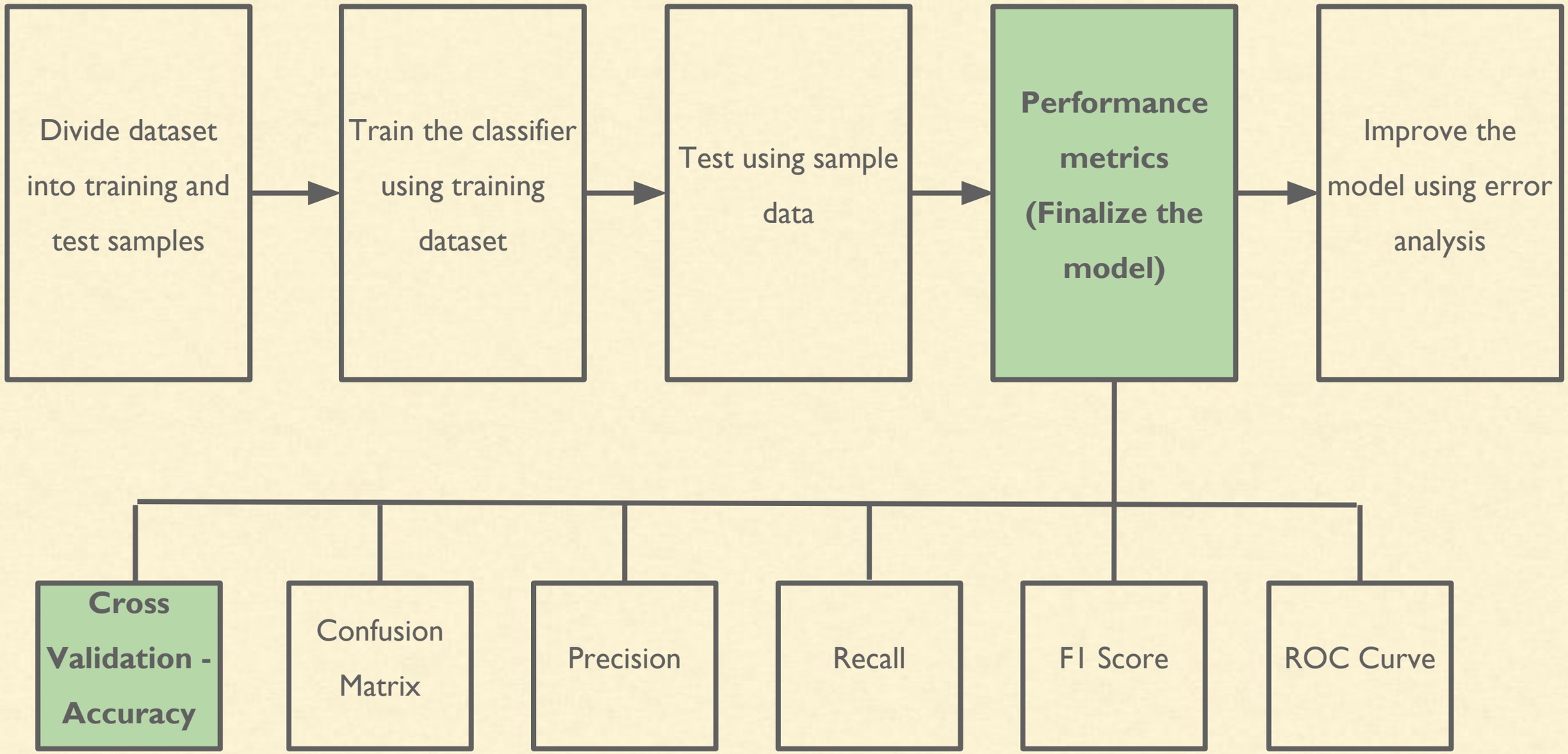
Steps


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

Performance measure - Methods

- Cross Validation - Accuracy
- Confusion Matrix
  - Precision
  - Recall
  - F1 score
- ROC Curve

Steps


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

Performance measure - Cross Validation

What is cross-validation?

- It involves splitting the training set into K distinct subsets called folds,
- Trains and evaluates 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.
    - There can be different evaluation methods like accuracy, average_precision, etc.

Performance measure - Cross Validation

Here k = 10

Performance measures - Cross Validation

cross_val_score

As discussed in end-to-end project session, cross_val_score() function in scikit-learn can be used to perform cross validation

Performing Cross validation in Scikit learn

Initiating cross validation method

```
>>> 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)

Switch to Notebook

Performing Cross validation in Scikit learn

Initiating cross validation method

```
>>> 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

array([0.9486, 0.9654, 0.957])

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

Performing Cross validation in Scikit learn

array([0.9486, 0.9654, 0.957])

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

The resulting accuracy of 95% is good enough?

- Accuracy may not be a good performance measure when dealing with skewed datasets
- Ours is a skewed dataset with only 10% of the data as digit 5.

Performing Cross validation in Scikit learn

Is the accuracy of 95% good enough?
Let us see another Classifier: Never5Classifier

```
>>> 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)
```

Switch to Notebook

Performing Cross validation in Scikit learn

Is the accuracy of 95% good enough?
Let us see another Classifier: Never5Classifier

- **What shall be the accuracy of Never5Classifier?**

```
>>> 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)
```

Switch to Notebook

Performing Cross validation in Scikit learn

Is the accuracy of 95% good enough?

Let us see another Classifier: Never5Classifier

- **What shall be the accuracy of Never5Classifier? 90%**

Calculating accuracy using cross_val_score

```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5,
cv=3, scoring="accuracy")
```

Switch to Notebook

Performing Cross validation in Scikit learn

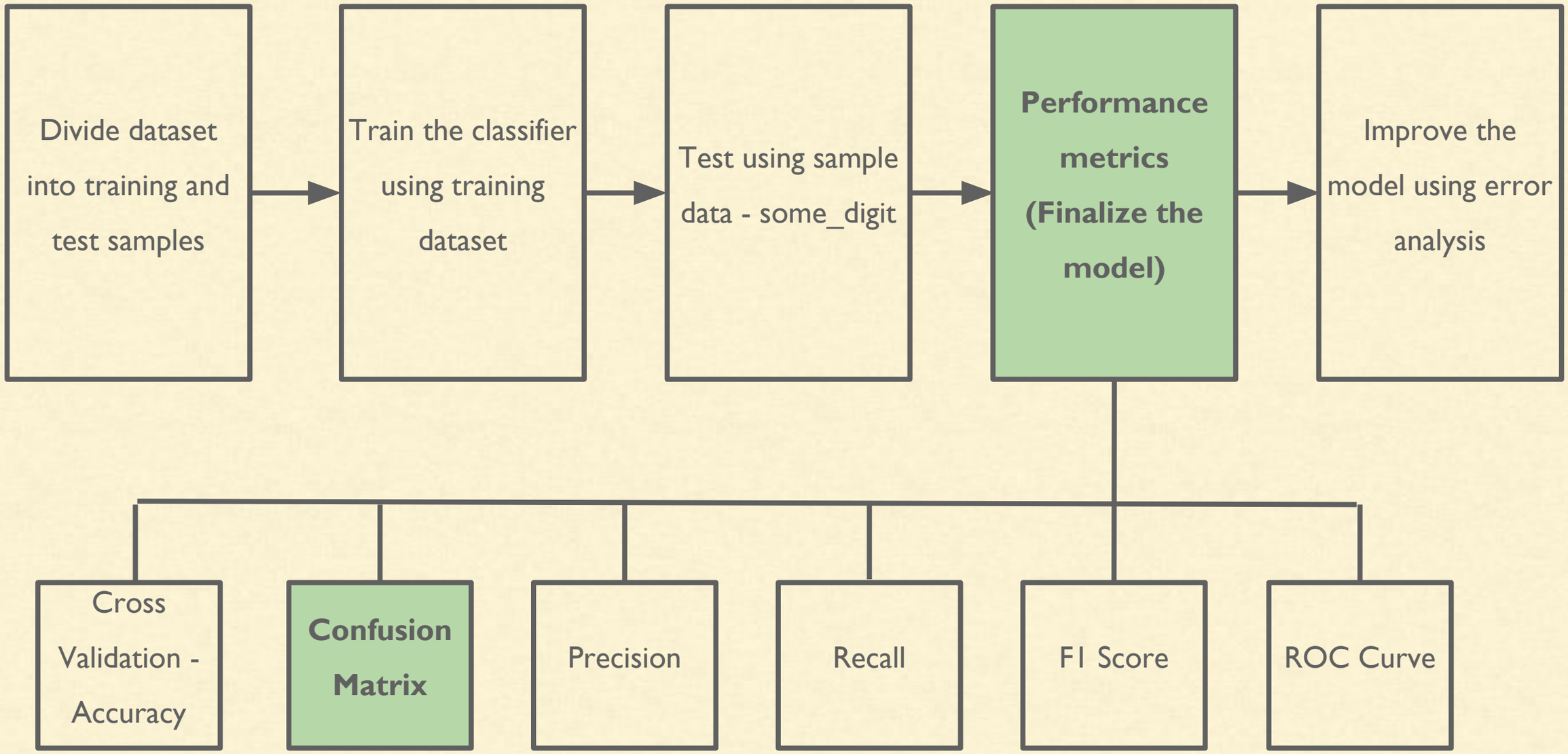
Is the accuracy of 95% for SGD Classifier good enough?

Probably Not!

Never5Classifier - a dumb classifier gave an accuracy of 90%

We need a better measure of performance for the classifier

Steps



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

Performance measures - Confusion Matrix

True Positive (TP)

False Negative (FN)

		Prediction: Positive	Prediction: Negative
Actual	Positive	TP	FN
Actual	Negative	FP	TN

False Positive (FP)

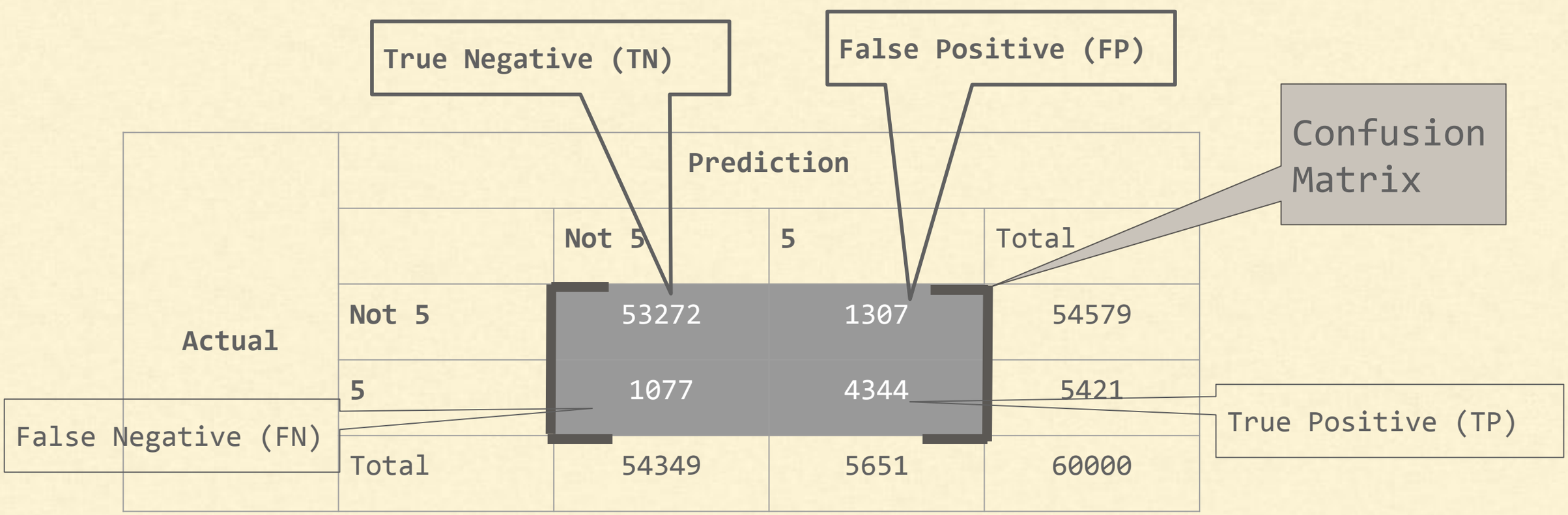
True Negative (TN)

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

Confusion Matrix - Example

For ‘5’ and ‘Not 5’ classifier

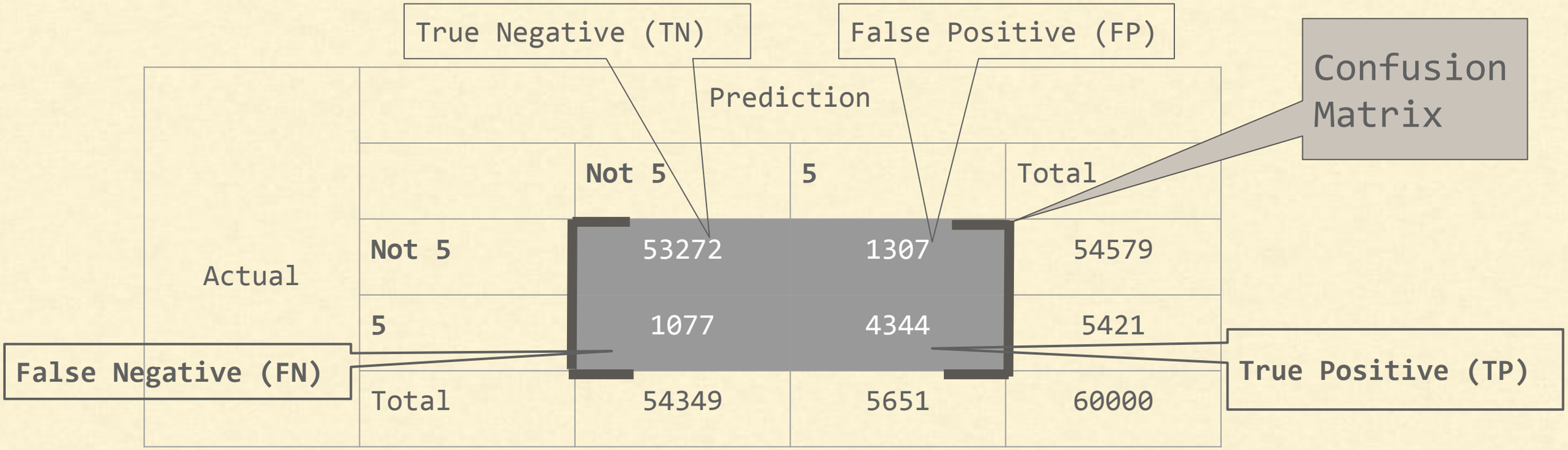
- The first row of this matrix considers non-5 images (the negative class):
  - 53,272 of them were correctly classified as non-5s (they are called true negatives)
  - The remaining 1,307 were wrongly classified as 5s (false positives).



Confusion Matrix - Example

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).



Performance measures - Confusion Matrix

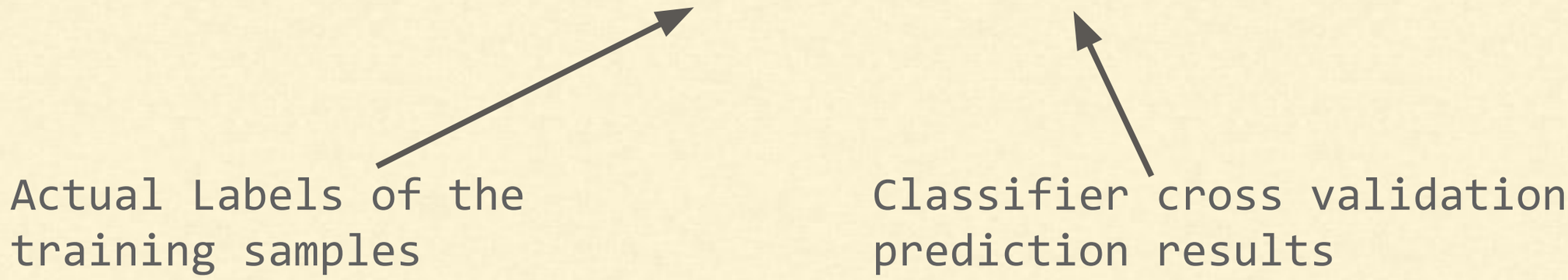
- Confusion matrix in Scikit Learn
  - first need to have a set of predictions,
  - Then, they can be compared to the actual targets.

```
>>> from sklearn.model_selection import
cross_val_predict
>>> y_train_pred = cross_val_predict(sgd_clf, X_train,
y_train_5, cv=3)
```

Performance measures - Confusion Matrix

- Confusion matrix in Scikit Learn
  - Then we compare the predicted value and the actual values

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



Performance measures - Confusion Matrix

- Confusion matrix in Scikit Learn

```
>>> from sklearn.model_selection import cross_val_predict
>>> y_train_pred = cross_val_predict(sgd_clf, X_train,
y_train_5, cv=3)
```

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

array([[53606,   973],
       [ 1607,  3814]])
```

Run it on Notebook

Performance measures - Confusion Matrix

- Confusion matrix in Scikit Learn

```
array([[53606,   973],
       [ 1607,  3814]])
```

The first row of this matrix considers non-5 images (the negative class):

+ 53,606 of them were correctly classified as non-5s (they are called true negatives)
+ The remaining 973 were wrongly classified as 5s (false positives).

Run it on Notebook

Performance measures - Confusion Matrix

- Confusion matrix in Scikit Learn

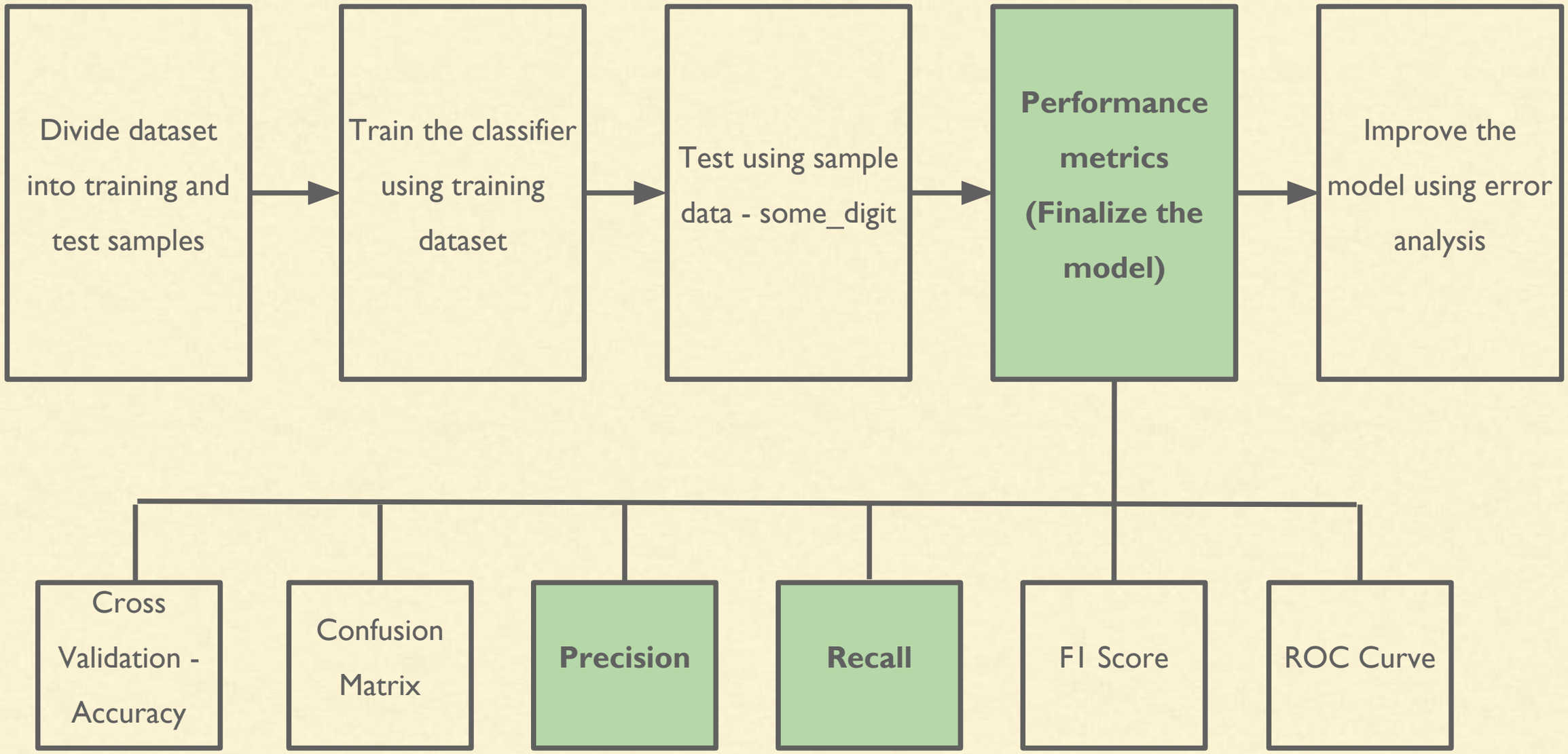
```
array([[53606,   973],
       [ 1607,  3814]])
```

The second row considers the images of 5s (the positive class):

+ 1,607 were wrongly classified as non-5s (false negatives)
+ The remaining 3,814 were correctly classified as 5s (true positives).

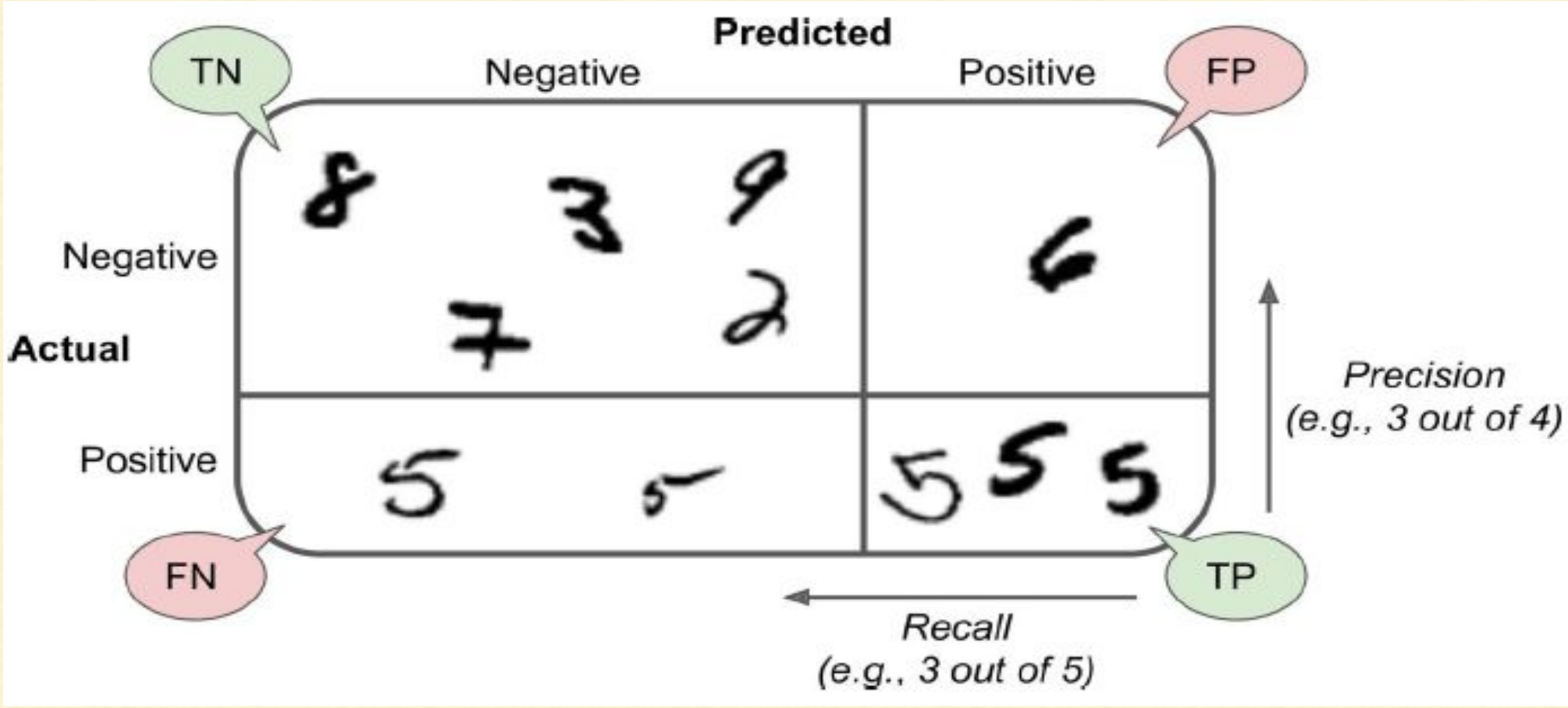
Run it on Notebook

Steps



Performance measures - Precision and recall

‘5’ and ‘Not 5’ classifier



Performance measures - Precision and recall

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

True - Positive means the classifier correctly classified the **Positive** class.

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

Performance measures - Precision and recall

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

True - Positive means the classifier correctly classified the **Positive** class.

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

Performance measures - Precision and recall

- '5' and 'Not 5' classifier

True Negative (TN)

False Positive (FP)

		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 %		

False Negative (FN)

True Positive (TP)

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

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

Performance measures - Precision and recall

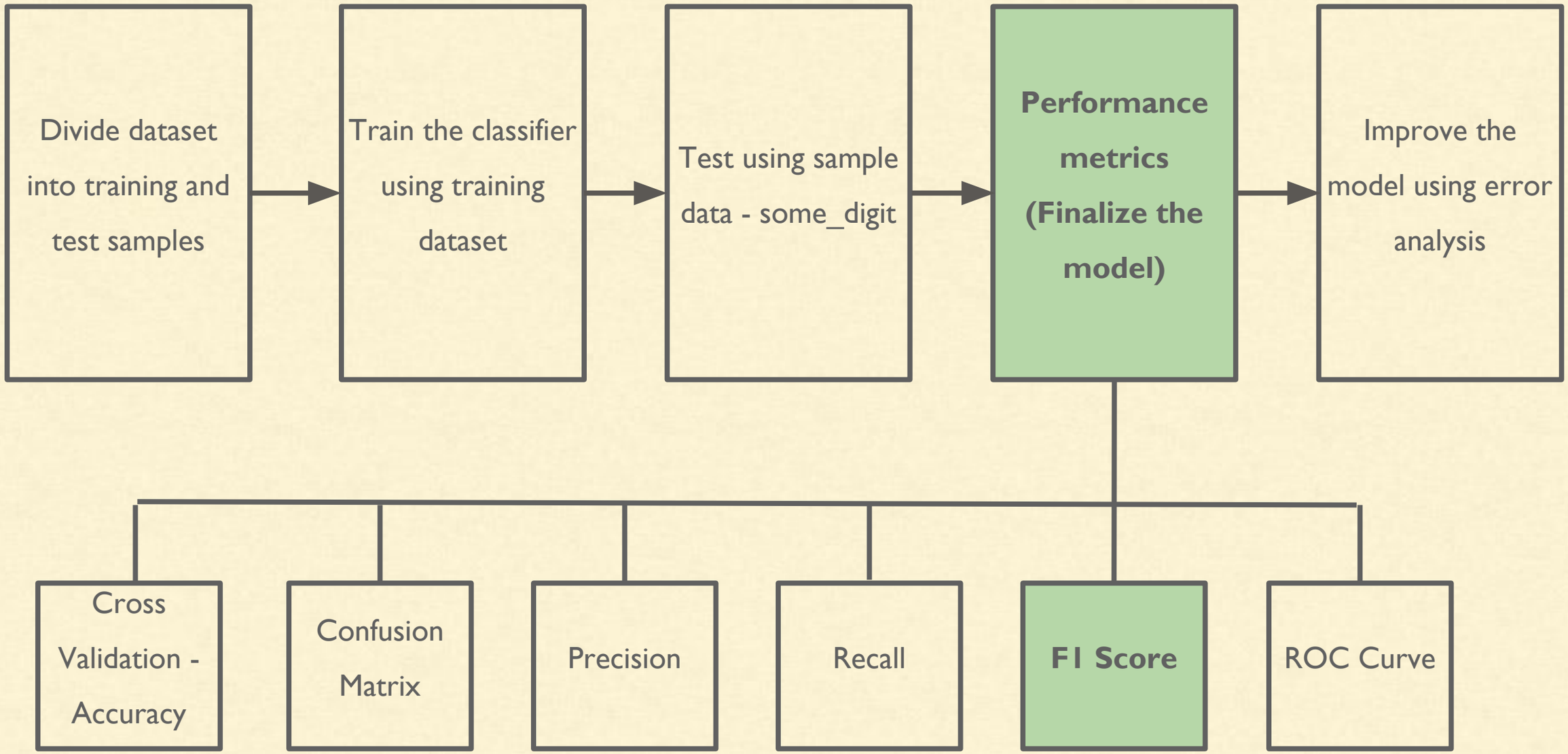
Precision and recall in Scikit-Learn

- **For 5 and Not 5 Classifier**

```
>>> from sklearn.metrics import precision_score,
recall_score
>>> precision_score(y_train_5, y_pred)
0.796741174013
>>> recall_score(y_train_5, y_train_pred)
0.70356022874
```

Switch to Notebook

Steps



Performance measures - F1 score

F1 score = is a measure of how close recall and precision are to each other.

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

Performance measures - F_1 score

- F1 score for '5' and 'Not 5' classifier

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

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

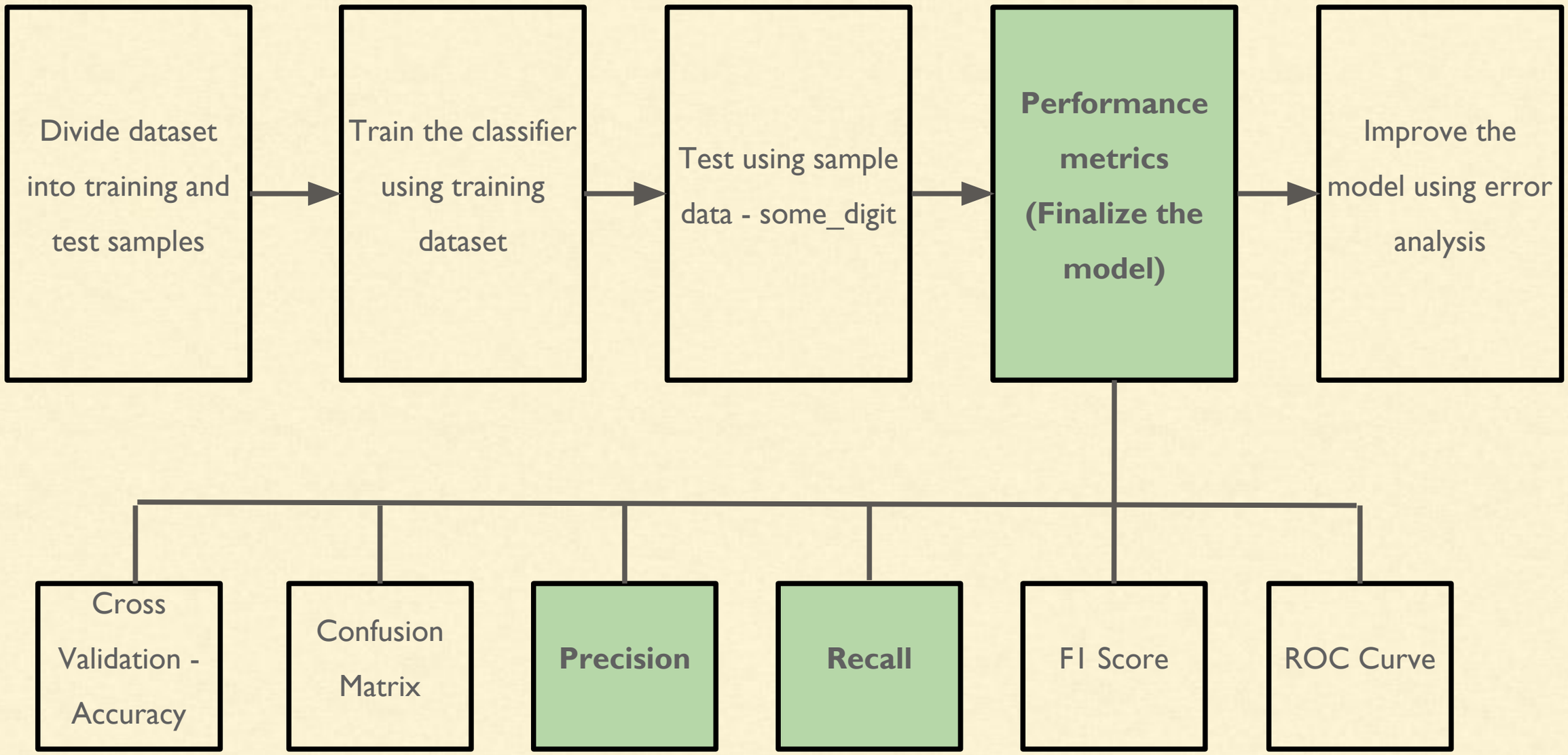
Performance measures - F1 score

- F1 score using Scikit-Learn
  - For 5 and Not 5 Classifier

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7472570532915361
```

Switch to Notebook

Steps



TRADE-OFF

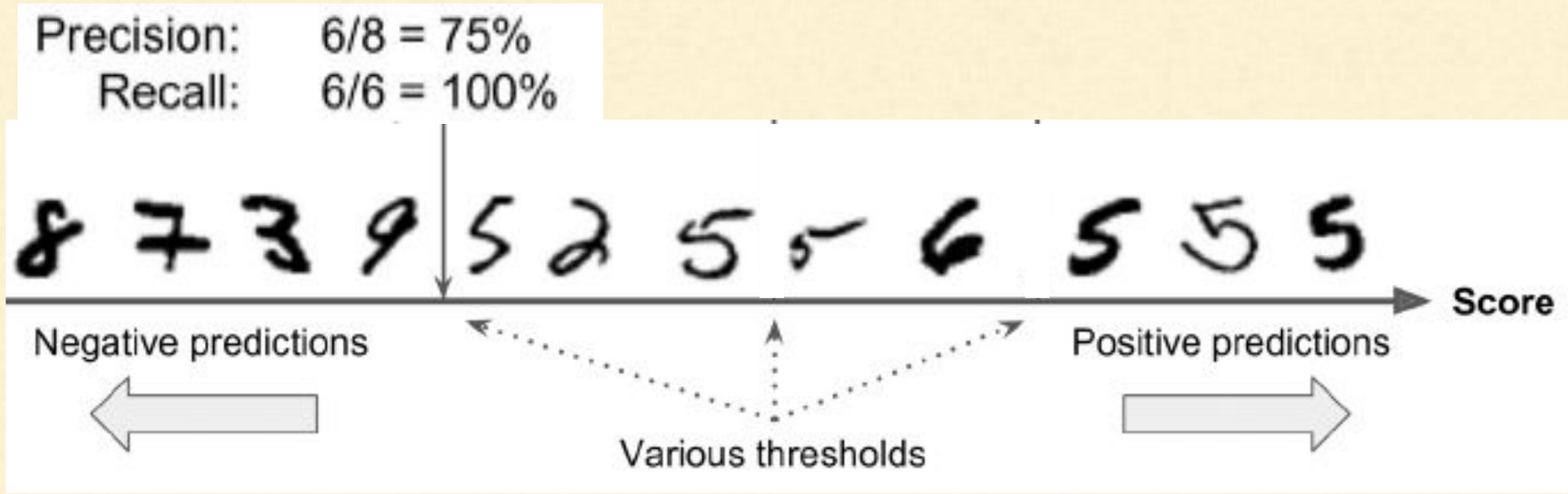
Performance measures - Precision vs Recall

- Different use cases may require different precision and recall

Performance measure ↓	**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.
Raising the threshold decreases Recall and increases Precision

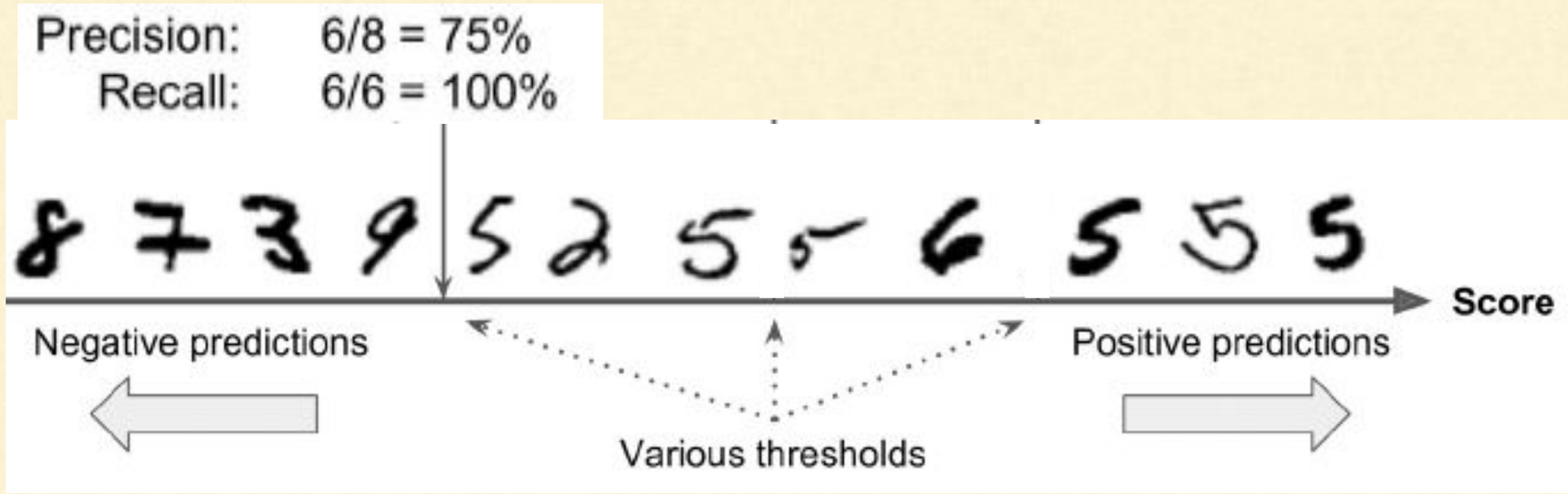
Precision / Recall Tradeoff - Thresholds



How does SGDClassifier work?

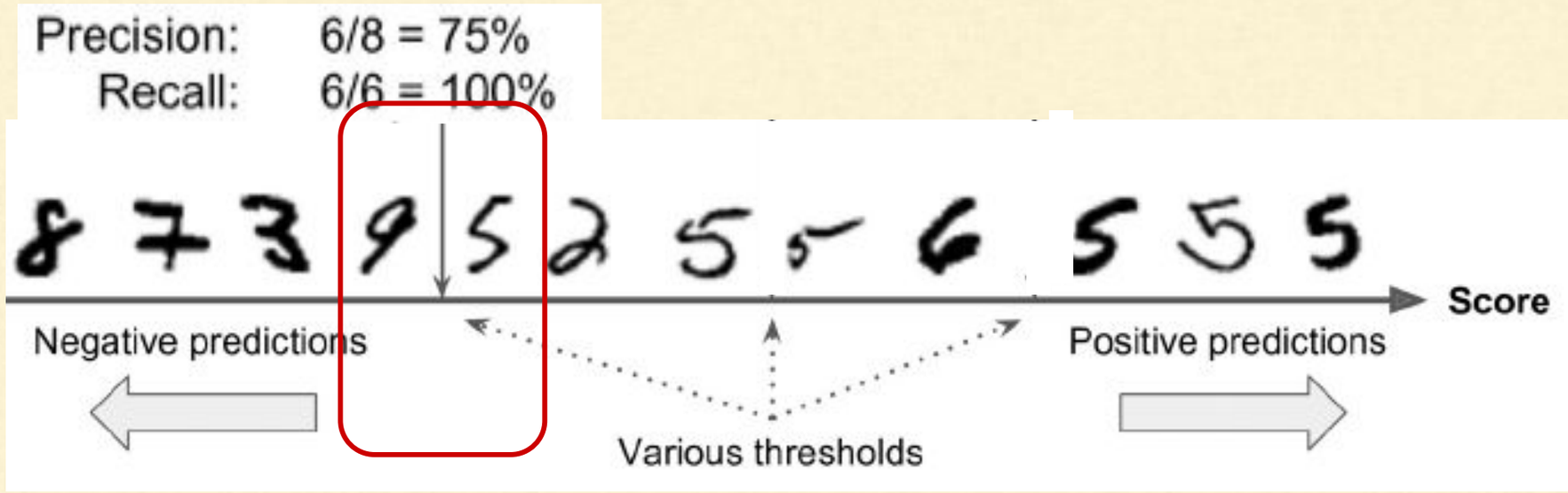
- Classification is done on the basis of a *score* as calculated by the decision function in SGD Classifier
  - Score above a certain *threshold* is classified as positive class
  - Score below a certain *threshold* is classified as negative class

Precision / Recall Tradeoff - Thresholds



- Thresholds can be set to achieve certain precision and recall as required.
- Let us observe the above example

Precision / Recall Tradeoff



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

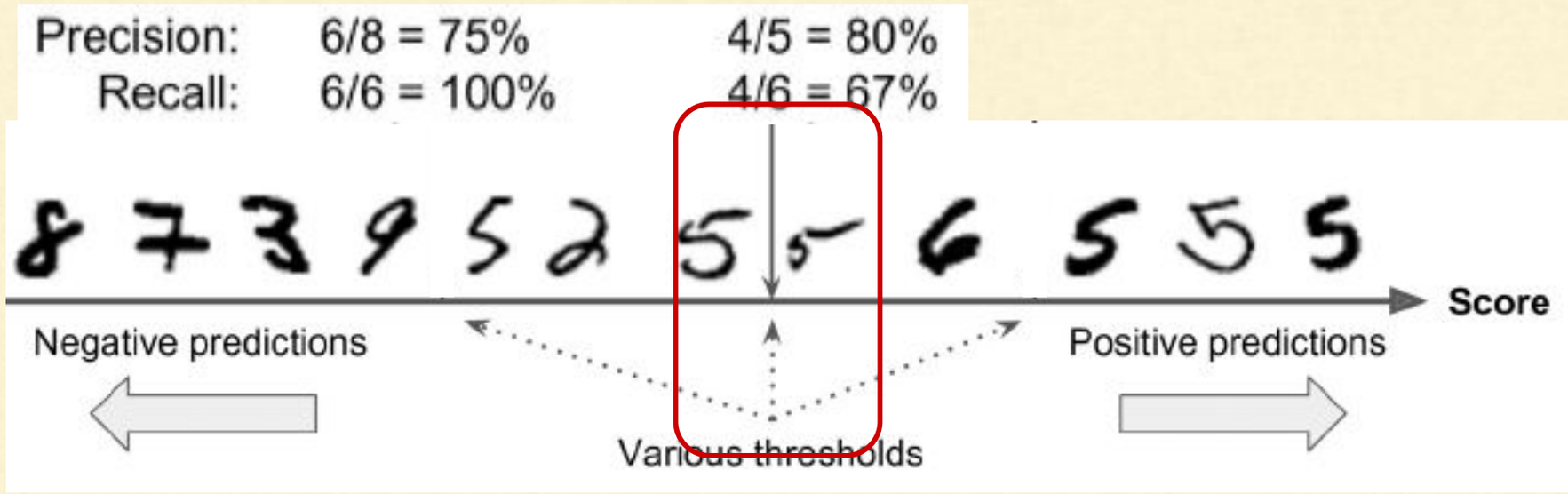
TN = 4, TP = 6, FN = 0, FP = 2

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

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

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

Precision / Recall Tradeoff



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

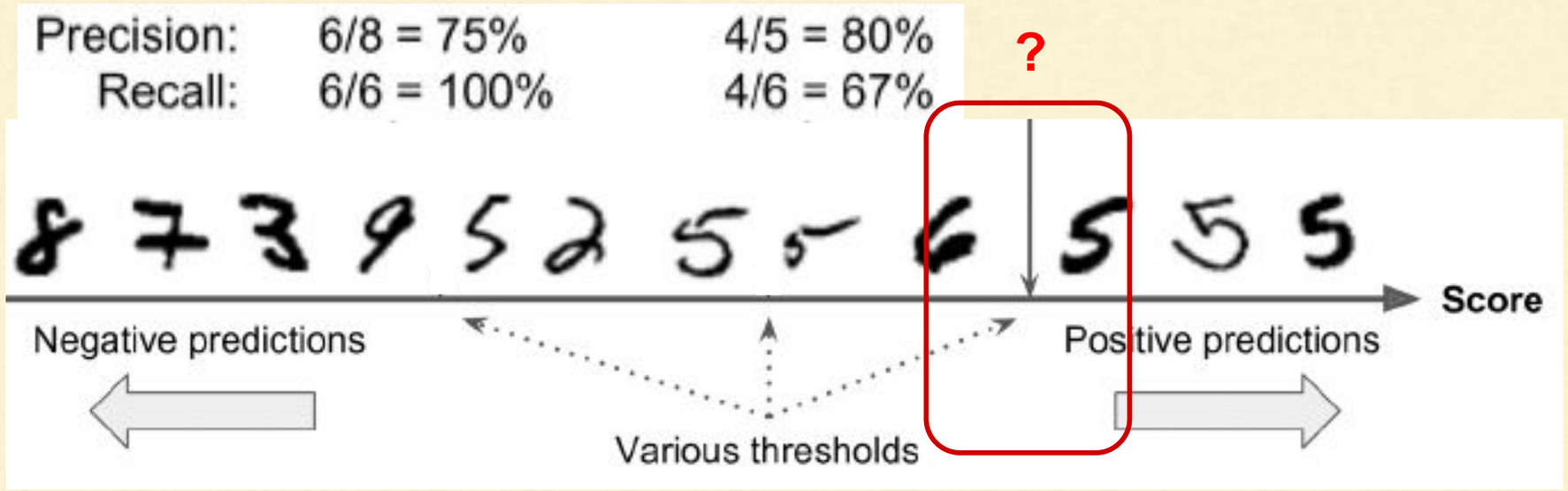
TN = 5, TP = 4, FN = 2, FP = 1

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

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

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

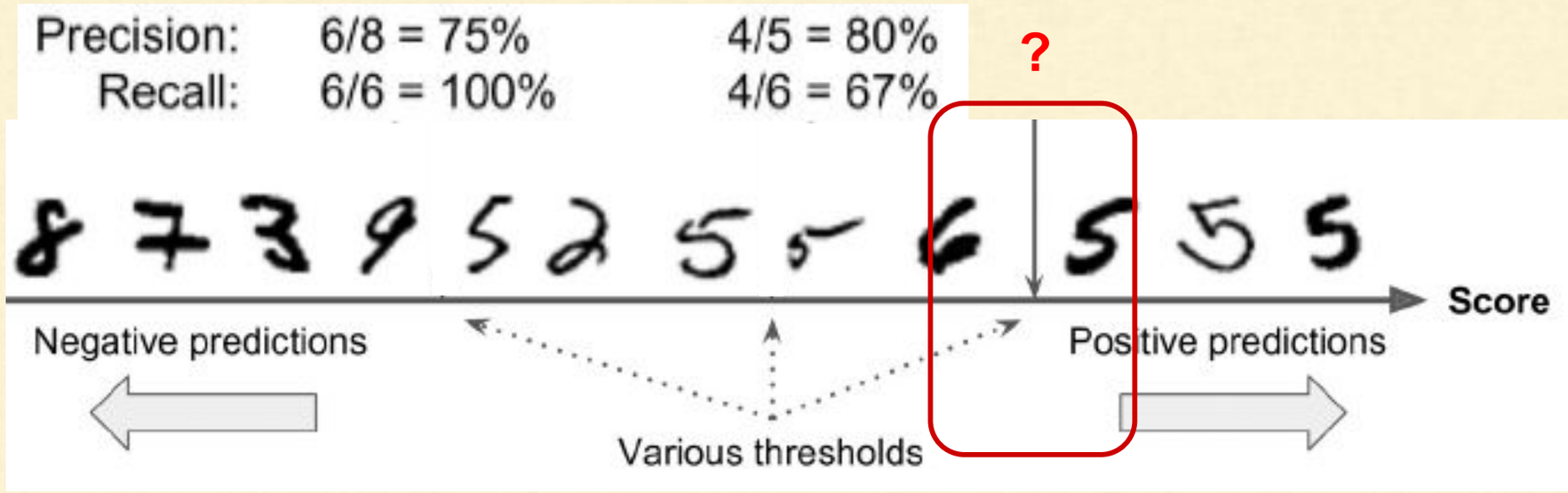
Precision / Recall Tradeoff



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

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

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%

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

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

Precision / Recall Tradeoff

Increasing precision reduces recall and vice-versa
Raising the threshold, increases precision and decreases recall

- For ‘5’ and ‘Not 5’ classifier
  - For threshold = 0, classifier correctly classifies 5 (36000th image) as 5
  - For threshold = 20000, classifier incorrectly classifies digit 5 (36000th image) as not 5

Precision / Recall Tradeoff - Thresholds

- Scikit enables the user to get the scores from the classifier

Calculating the decision function score for the SGDClassifier

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([ 1206.46829305])
```

Precision / Recall Tradeoff

Increasing precision reduces recall and vice-versa
Raising the threshold, decreases recall

- For '5' and 'Not 5' classifier
  - For threshold = 0, classifier correctly classifies 5 as 5
  - For threshold = 20000, classifier incorrectly classifies digit 5 as not 5

```
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([ True], dtype=bool)
```

Run it on Notebook

Precision / Recall Tradeoff

Increasing precision reduces recall and vice-versa
Raising the threshold, decreases recall

- For '5' and 'Not 5' classifier
  - For threshold = 0, classifier correctly classifies 5 as 5
  - For threshold = 20000, classifier incorrectly classifies digit 5 as not 5

```
# Setting the threshold to 20000
>>> threshold = 20000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False], dtype=bool)
```

Run it on Notebook

Review of Precision/ Recall Tradeoff

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

Precision / Recall Curve

How to decide the best threshold?

- Get the scores of all the training dataset using cross_val_predict with decision_function as function
- Compute the precision and recall for all possible thresholds using precision_recall_curve()
- Plot both precision and recall for the thresholds using matplotlib
- Select the threshold value that gives the best precision/ recall tradeoff to the task at hand.

Precision / Recall Tradeoff

- Calculating precision/ recall curve using Scikit-Learn

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

>>> from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds =
precision_recall_curve(y_train_5, y_scores)
```

Run it on Notebook

Precision / Recall Tradeoff

- Plotting precision/ recall curve using Scikit-Learn

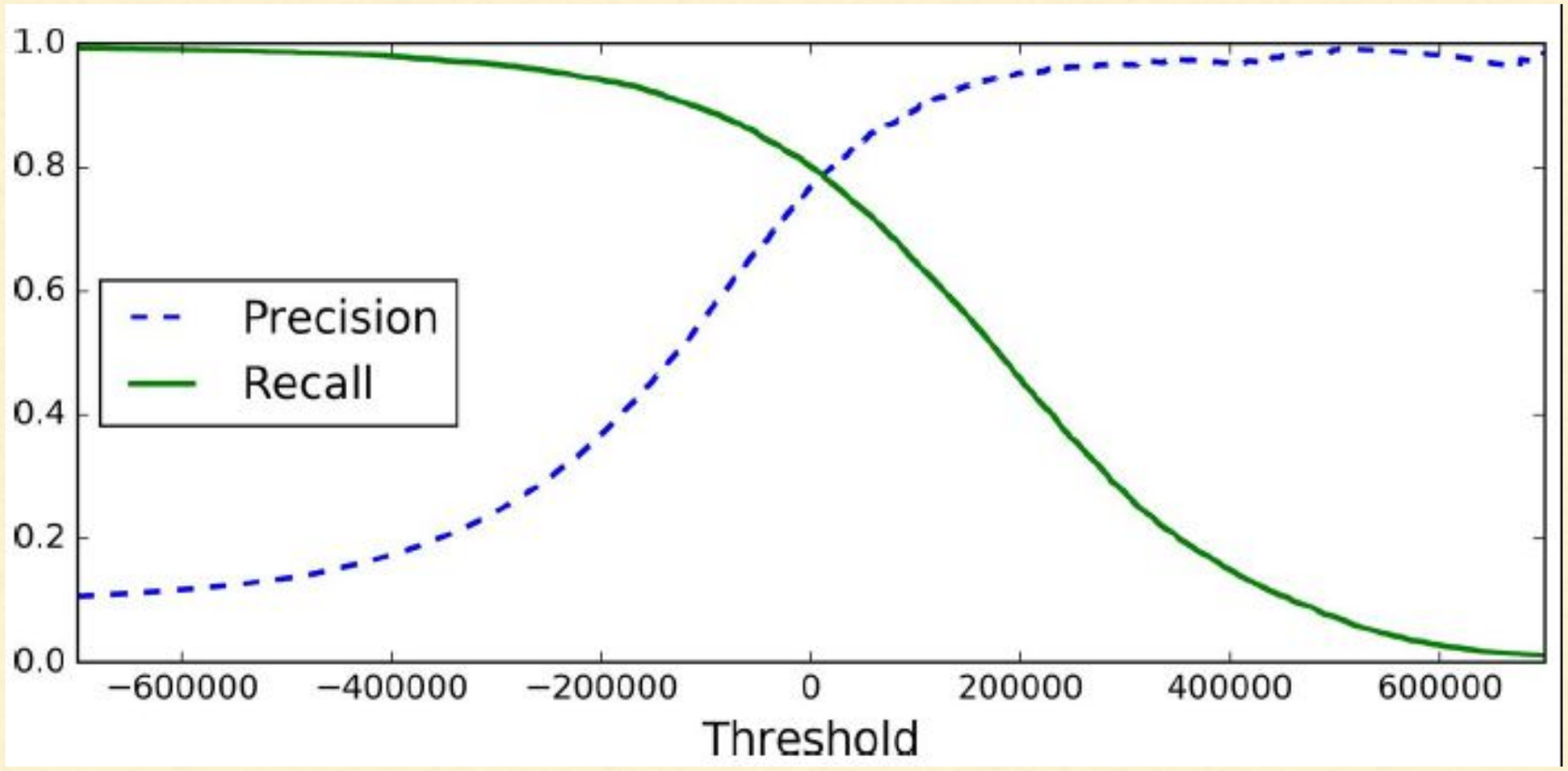
```
>>> def plot_precision_recall_vs_threshold(precisions, recalls,
thresholds):
    plt.figure(figsize=(18,7))
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")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

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

Run it on Notebook

Precision / Recall Curve

- Plotting precision/ recall curve using Scikit-Learn



Precision vs Recall Curve

- Another way to select a good precision/recall tradeoff is to
  - plot **precision directly against recall**.

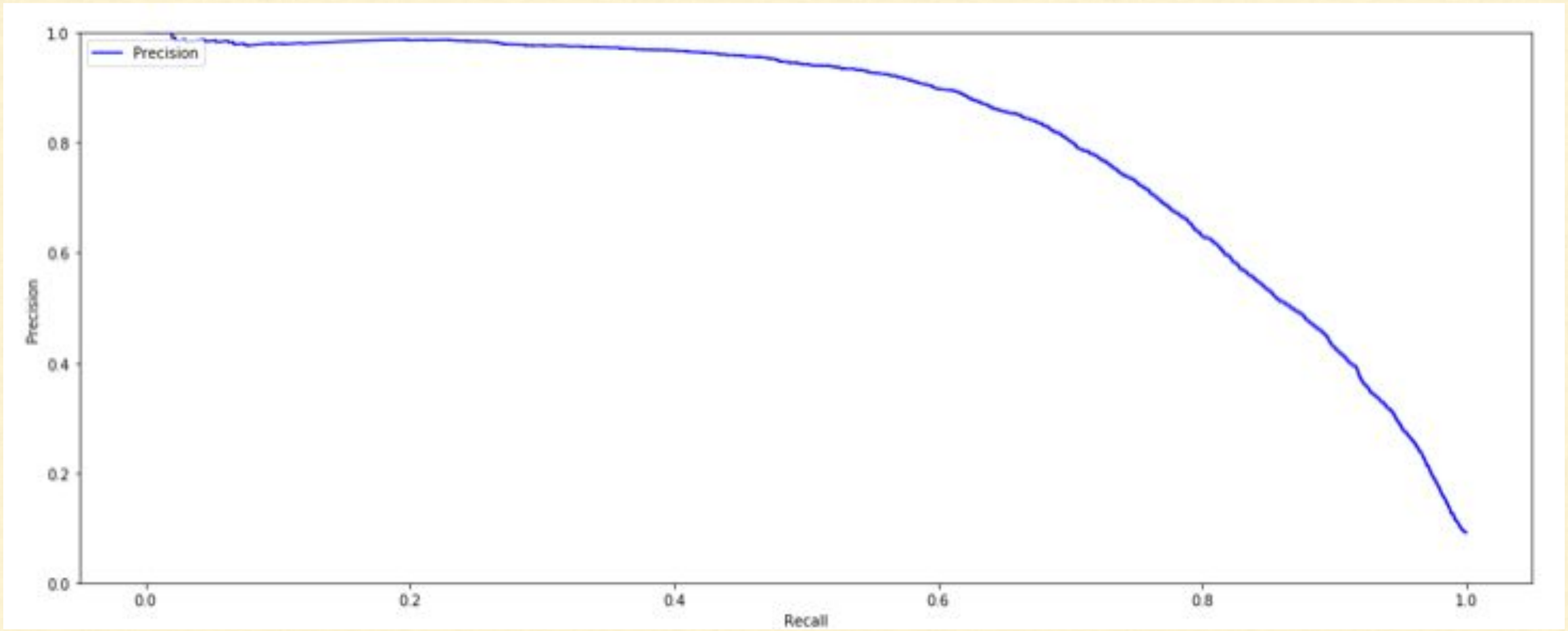
```
>>> def plot_precision_vs_recall(precisions, recalls):
    plt.figure(figsize=(18,7))
    plt.plot(recalls[:-1], precisions[:-1], "b-", label="Precision")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.legend(loc="upper left")
    plt.ylim([0, 1])

>>> plot_precision_vs_recall(precisions, recalls)
>>> plt.show()
```

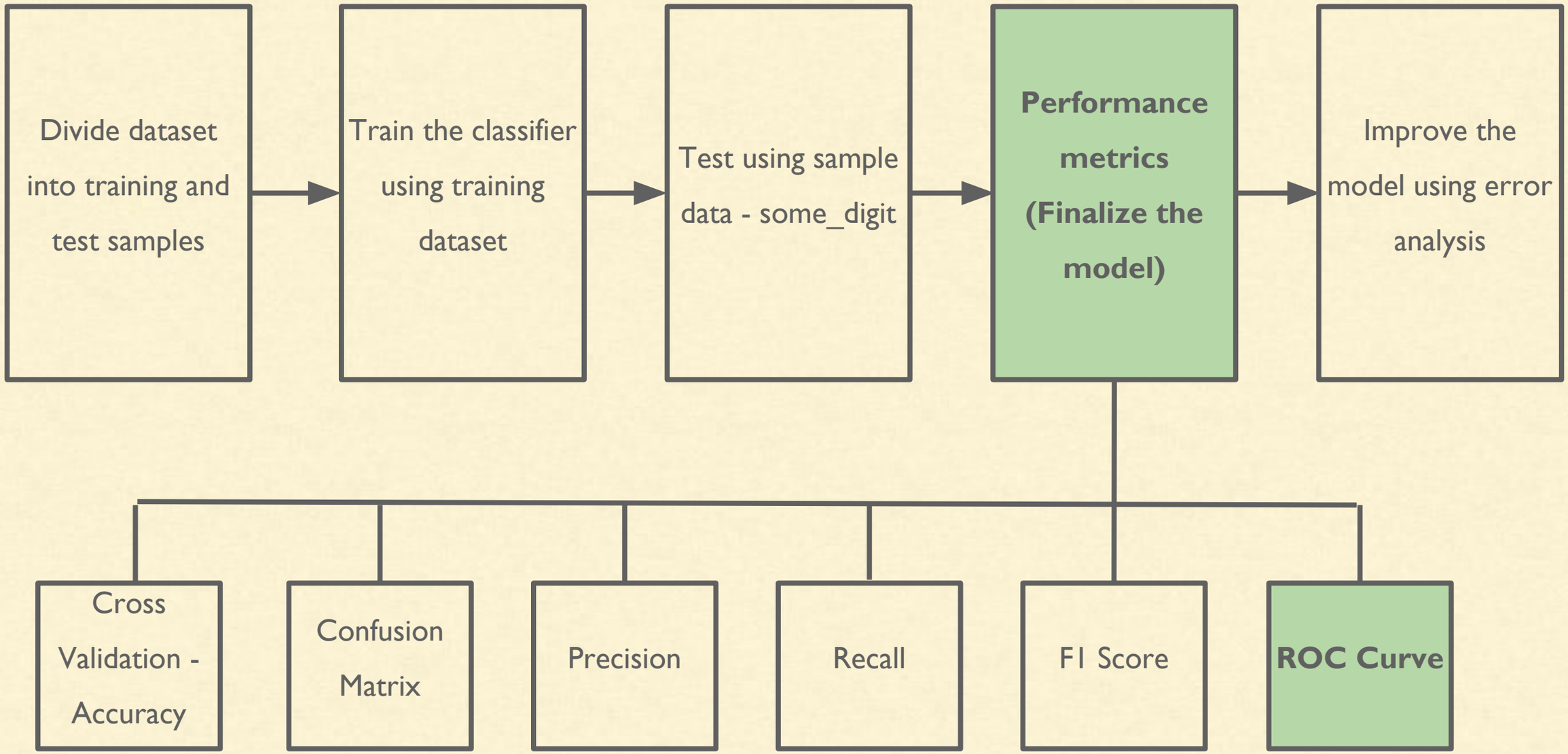
Run it on Notebook

Precision vs Recall Curve

- Another way to select a good precision/recall tradeoff is to
  - plot precision directly against recall directly.



Steps



Performance measures - ROC Curve

- 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)
- TNR = TN / (FP + TN)

Receiver Operating Characteristics (ROC) plots:
TPR versus FPR

Performance measures - ROC Curve

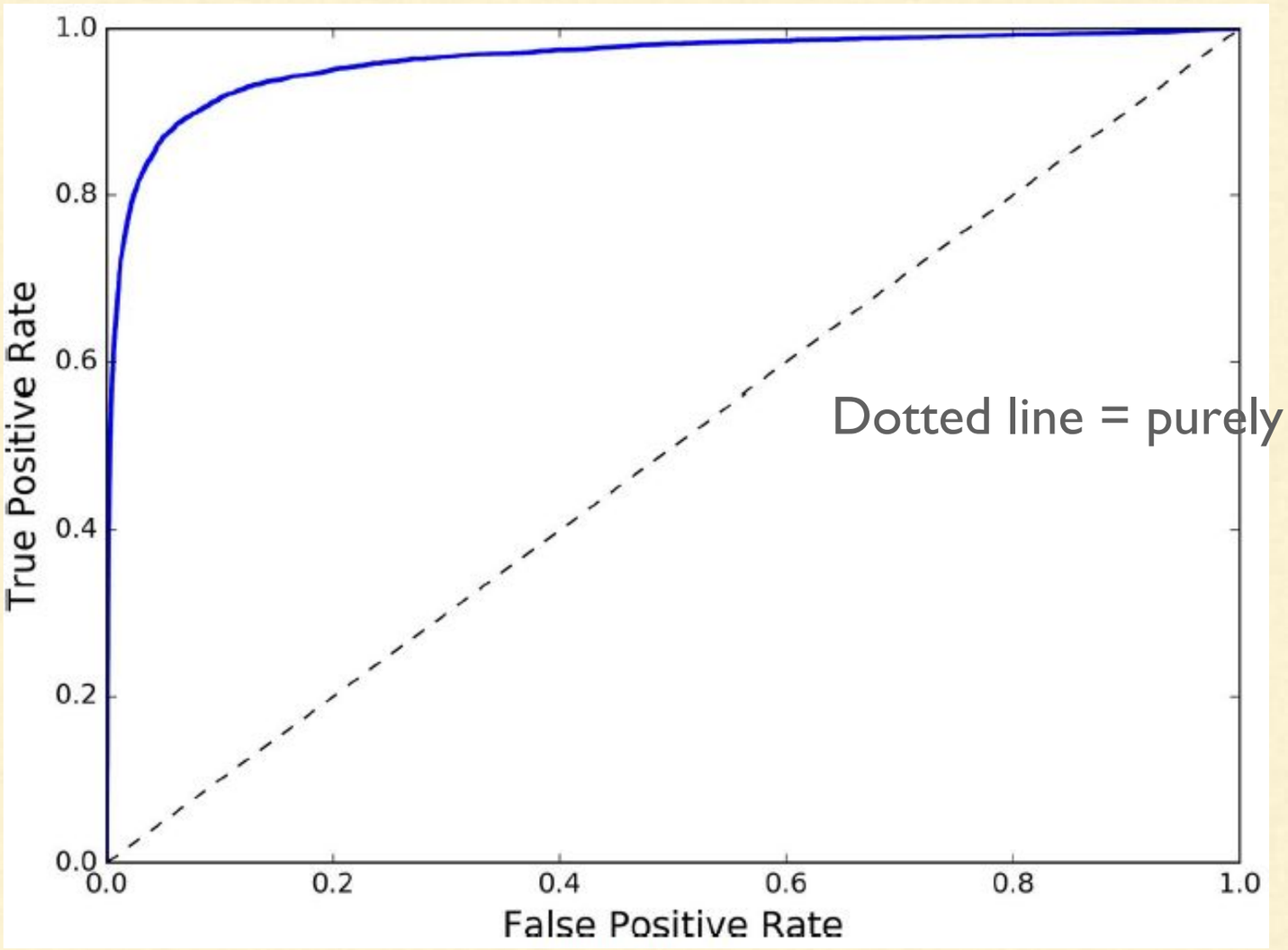
- Making ROC Curve using sklearn

```
>>> from sklearn.metrics import roc_curve
>>> fpr, tpr, thresholds = roc_curve(y_train_5,
y_scores)
>>> def plot_roc_curve(fpr, tpr, label=None):
        plt.figure(figsize=(18,7))
        plt.plot(fpr, tpr, linewidth=2, label=label)
        plt.plot([0, 1], [0, 1], 'k--')
        plt.axis([0, 1, 0, 1])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
>>> plot_roc_curve(fpr, tpr)
>>> plt.show()
```

Run it on Notebook

Performance measures - ROC Curve

- Making ROC Curve using sklearn



Dotted line = purely random classifier

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.

Performance measures - ROC Curve

- Making ROC Curve using sklearn

 # 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)
0.9550544284581809
```

Run it on Notebook

Performance measures - ROC Curve

- RandomForestClassifier versus SGDClassifier comparison for digit 5 binary classifier

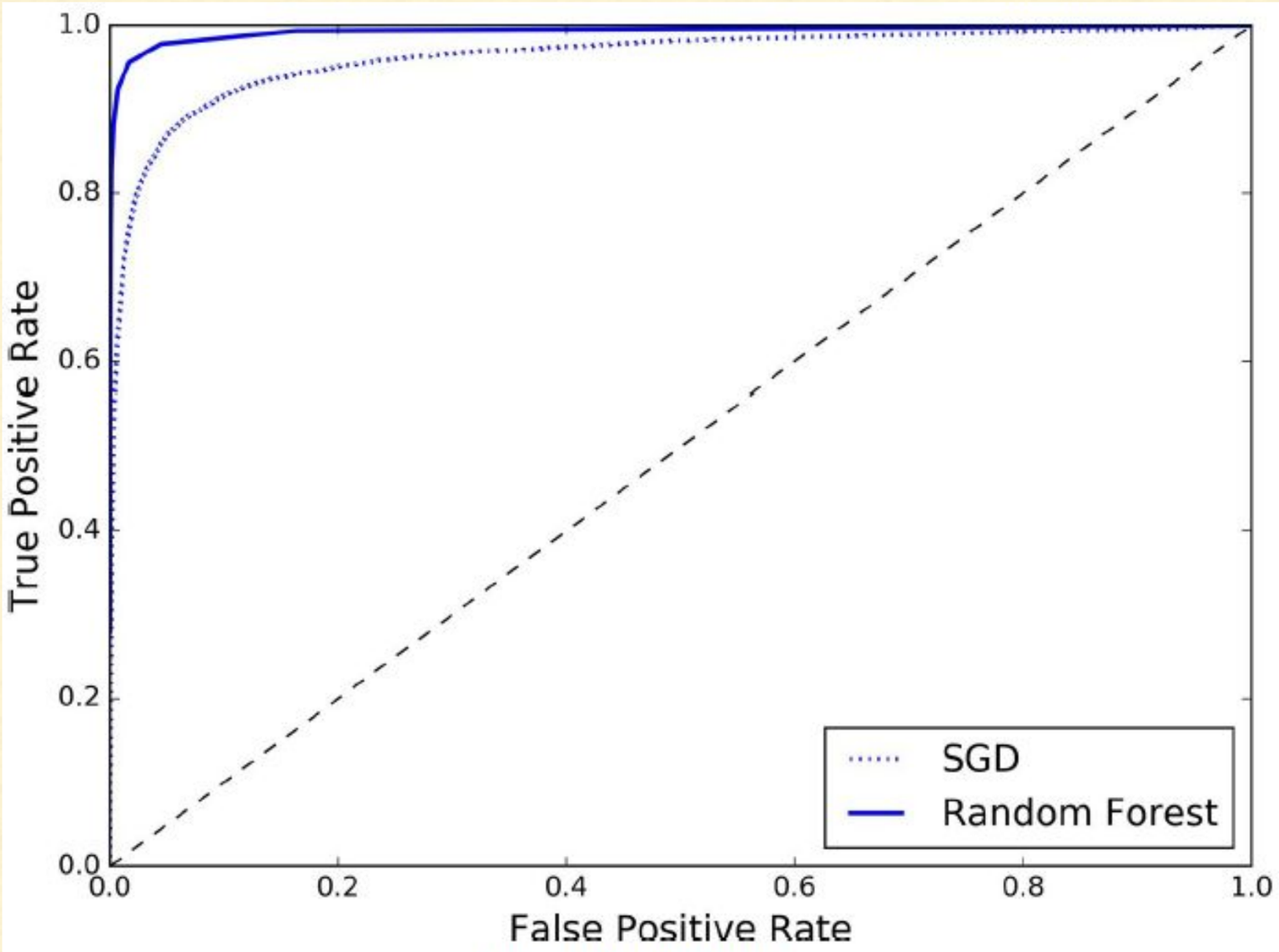


Figure 3-7. Comparing ROC curves

Performance measures - ROC Curve

- RandomForestClassifier versus SGDClassifier comparison for digit 5 classifier
- RandomForestClassifier uses predict_proba() instead of decision_function() for calculating the score of the classifier

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> forest_clf = RandomForestClassifier(random_state=42)

>>> y_probas_forest = cross_val_predict(forest_clf, X_train,
y_train_5, cv=3, method="predict_proba")
```

Run it on Notebook

Performance measures - ROC Curve

- Plotting the ROC Curve for RandomForestClassifier
  - Treating the probability of the positive class as score

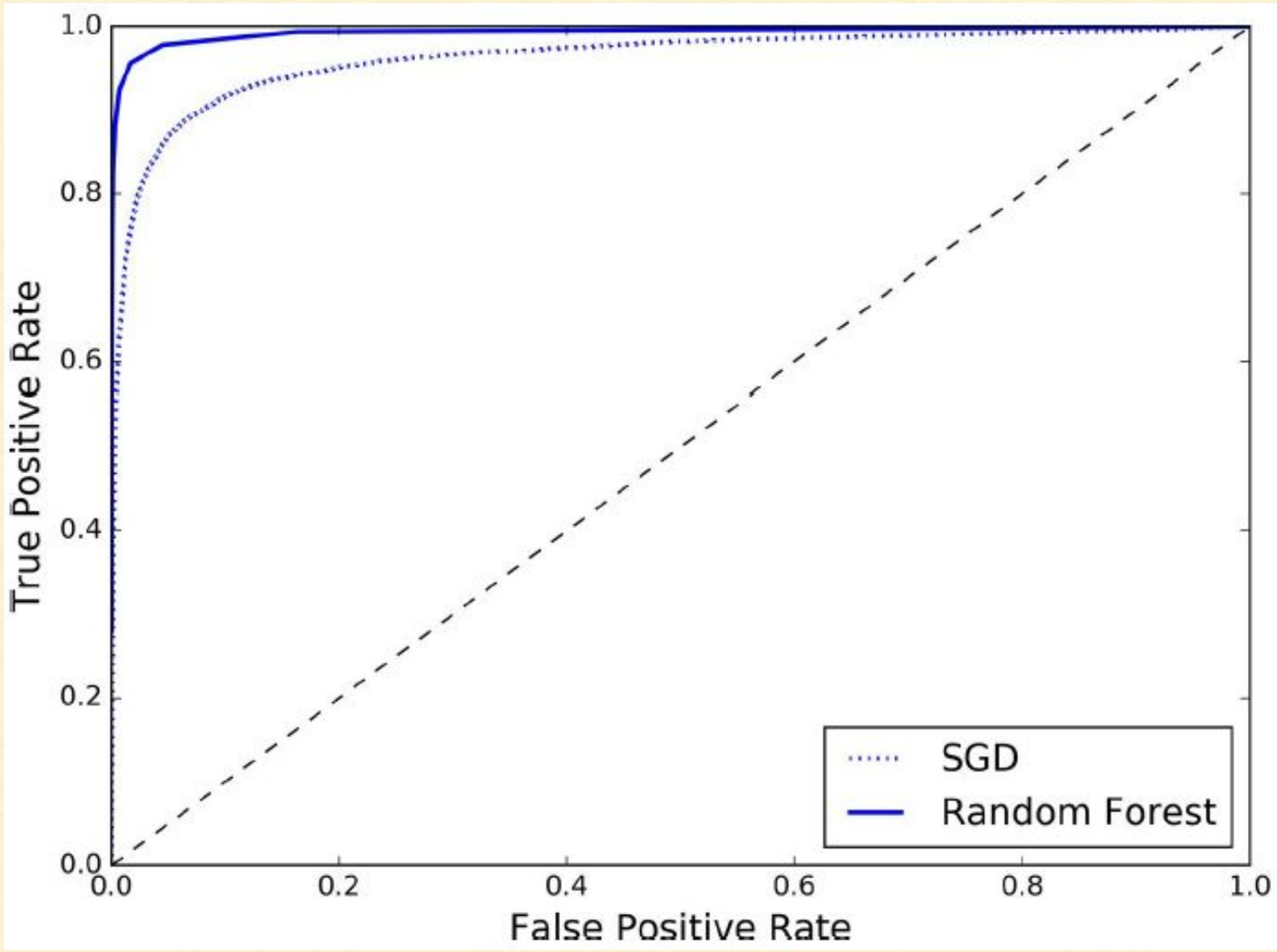
```
>>> y_scores_forest = y_probas_forest[:, 1]
>>> fpr_forest, tpr_forest, thresholds_forest =
roc_curve(y_train_5,y_scores_forest)
```

```
>>> fig, ax = plt.subplots()
>>> ax.plot(fpr, tpr, "b:", label="SGD")
>>> ax.plot(fpr_forest, tpr_forest, linewidth=2, label="Random
Forest")
>>> ax.plot([0, 1], [0, 1], 'k--')
>>> ax.legend(loc=4);
>>> ax.set_title('ROC curves');
```

Run it on Notebook

Performance measures - ROC Curve

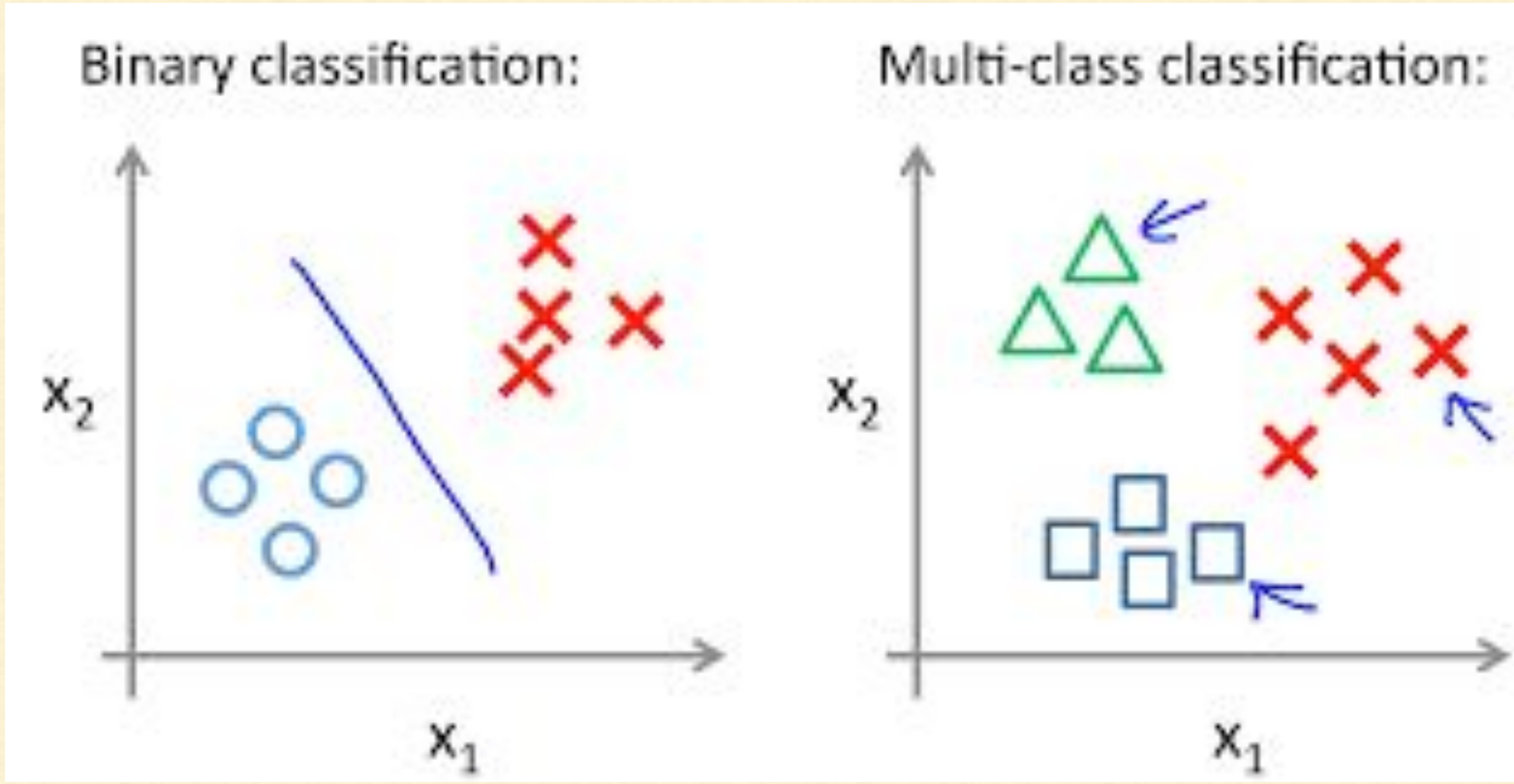
- ROC Curve for RandomForestClassifier versus SGDClassifier
  - RandomForestClassifier significantly better



Review - Binary Classification

- 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



Multi-class Classification

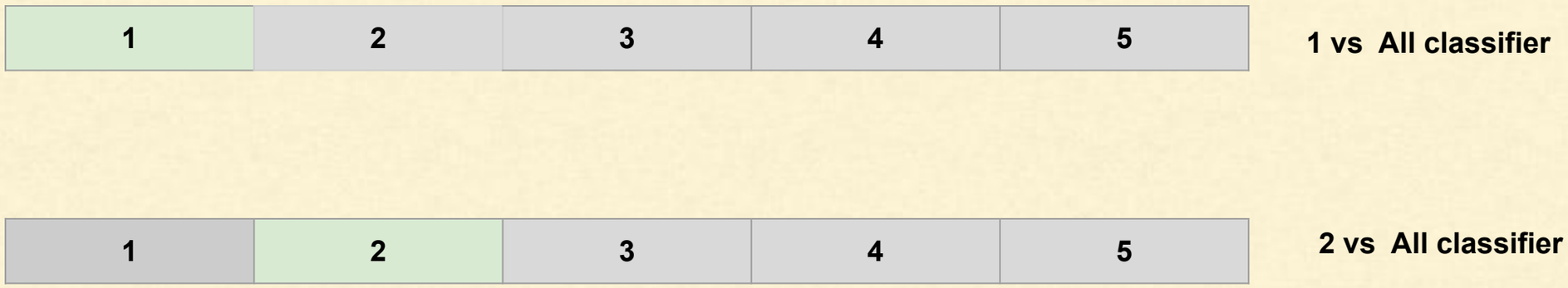
- Binary classifiers distinguish between two classes
- While multi-class classifiers (also called multinomial classifiers) can distinguish
  - Between more than two classes
  - Can distinguish between multiple classes
  - Eg. Random Forest classifiers , naive Bayes classifiers etc

Multiclass Classification

A Binary Classifier can be used for Multiclass Classification. There are basically two strategies for doing this.

1. **One-versus-all (OvA) strategy** also called one-versus-the-rest - 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, select the class whose classifier outputs the highest score.

Multiclass Classification - OvA strategy



Multi-class Classification

- Doing multi-class classification using OvA Classifier
  - **By default**, scikit-learn SGDClassifier assumes OvA

```
# By default scikit learn SGDClassifier assumes OvA
>>> sgd_clf.fit(X_train, y_train)
# Under the hood it trained 10 classifiers

>>> sgd_clf.predict([some_digit])
array([ 5.])
```

Run it on Notebook

Multiclass Classification

1. **One-versus-all (OvA) strategy**
   a. Under the hood, for the multiclass SGDClassifier example,
      i. the classifier trains 10 binary classifiers and
      ii. selects the class with highest score

Multi-class Classification

- Doing multi-class classification using OvA Classifier
  - 10 classifier scores is obtained
  - Index of the maximum score is 5

```
>>> some_digit_scores = sgd_clf.decision_function([some_digit])
>>> some_digit_scores
array([[ -86880.99526464, -237672.82817877, -166400.74040071,
         -131594.15873554, -191845.64360797,    1206.46829305,
         -330563.87557331, -241281.56362563, -308221.30939119,
         -314369.41736179]])
# The highest score is indeed the one corresponding to class 5:
```

Run it on Notebook

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 × (N – 1) / 2 classifiers.

Multiclass Classification

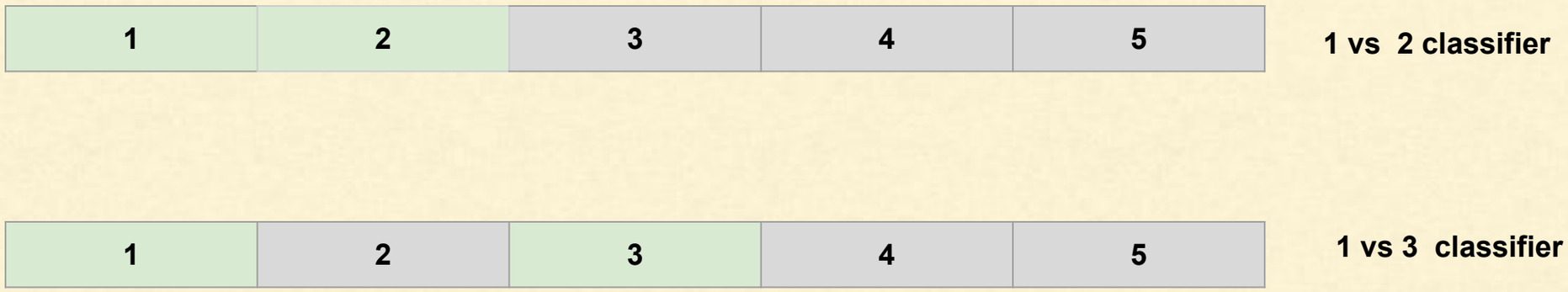
How many classifiers do we need to train in MNIST for OvO?

Multiclass Classification

How many classifiers do we need to train in MNIST for OvO?

10*9/2 = 45

Multiclass Classification - OvO Strategy



Multi-class Classification

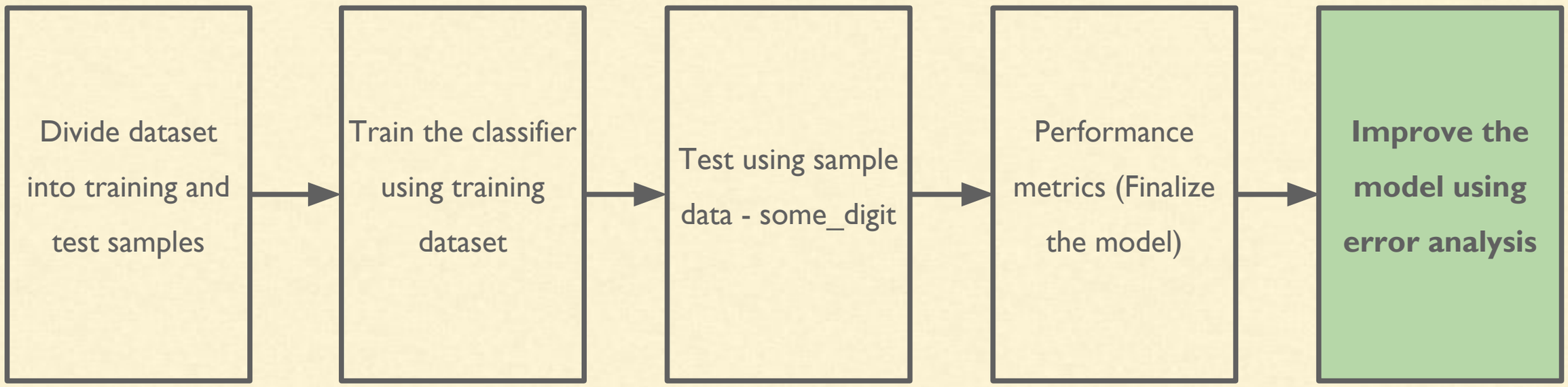
- Doing multi-class classification using OvO Classifier
  - SGD uses OVA by default
  - Need to specifically mention OvO to do OvO classification
  - Number of estimators trained for MNIST dataset = 45

```
>>> from sklearn.multiclass import OneVsOneClassifier
>>> ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42,
max_iter=20))
>>> ovo_clf.fit(X_train, y_train)
>>> ovo_clf.predict([some_digit])
array([ 5.])
>>> len(ovo_clf.estimators_)
45
```

Run it on Notebook

Switch to Notebook

Steps

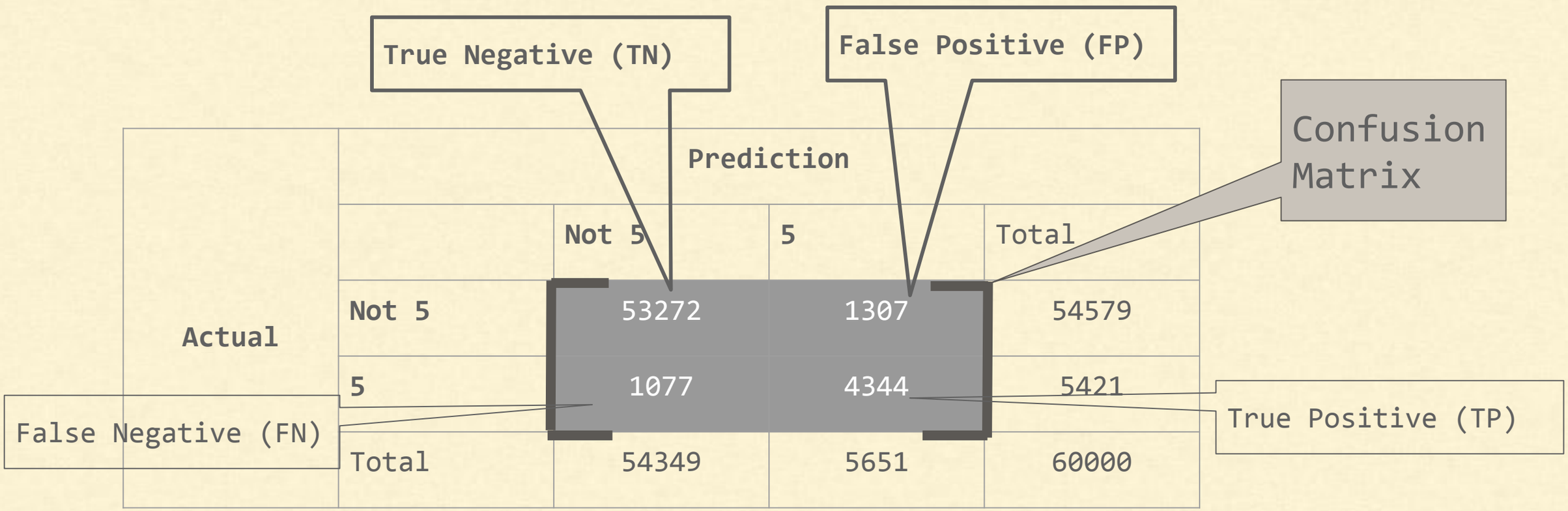

Divide dataset into training and test samples
Train the classifier using training dataset
Test using sample data - some_digit
Performance metrics (Finalize the model)
Improve the model using error analysis

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

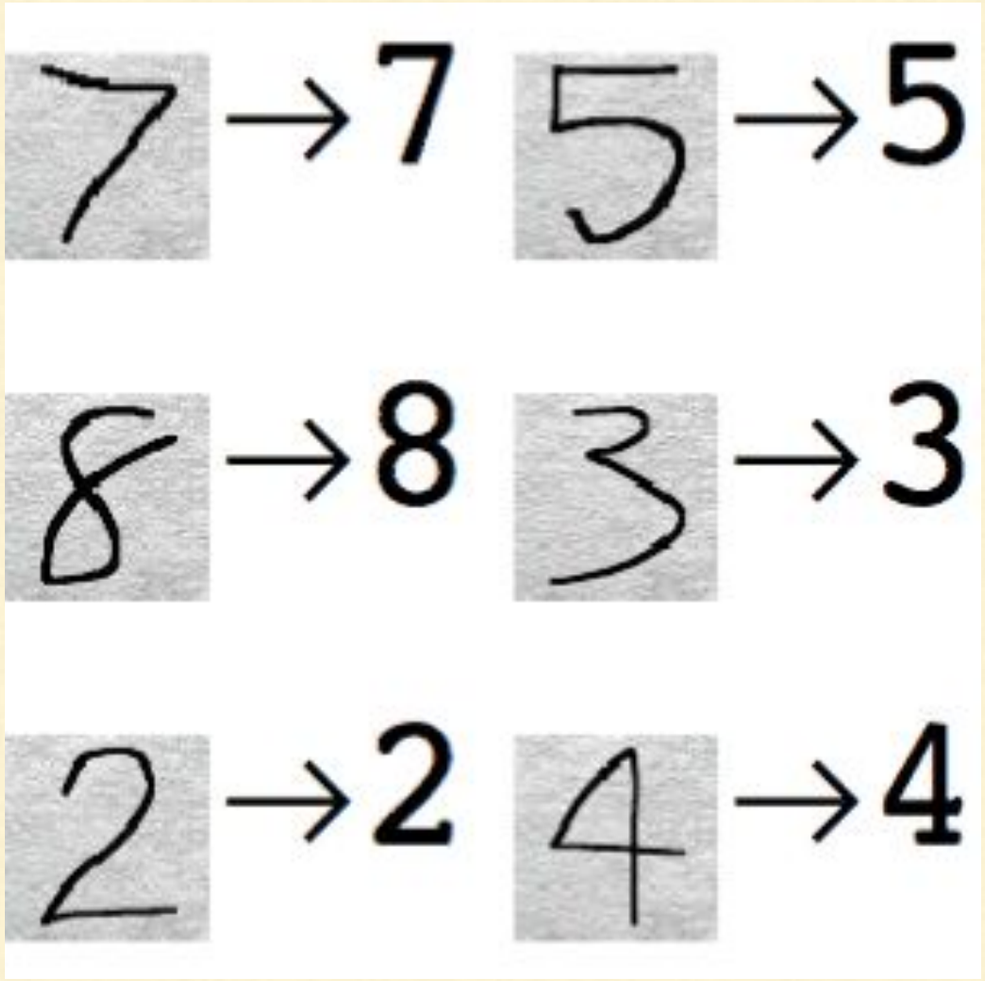
Confusion Matrix - Recap

For ‘5’ and ‘Not 5’ classifier



Multi-class Classification - Review

Identifying to which label something belongs to



Confusion Matrix - Multiclass

Confusion matrix for Multiclass classifier

Number of instances actually 0 and correctly predicted as 0

		Prediction						
		0	1	2	3	...	9	Total
Actual	0	5725	..	...	..	..	4	
	1	..	..	..	..	..	..	
	2	..	..	..	..	..	..	
	...	..	..	..	..	..	..	
	9	43	..	..	..	..	5240	
	Total							60000

Number of instances actually 0 and incorrectly predicted as 9

Error Analysis

- Calculating confusion matrix for multiclass using Scikit-Learn

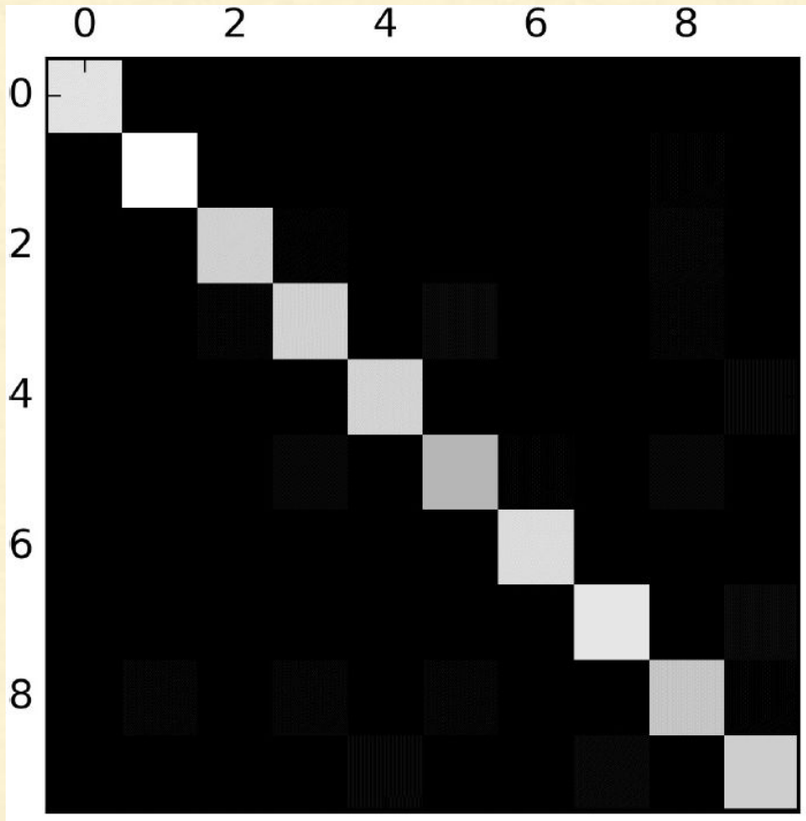
```
>>> y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train,
cv=3)
>>> conf_mx = confusion_matrix(y_train, y_train_pred)
>>> conf_mx
array([[5730,    2,   23,    8,   11,   43,   51,    8,   43,    4],
       [   1, 6472,   46,   25,    6,   45,    5,   12,  119,   11],
       [  47,   35, 5351,  101,   85,   24,   89,   53,  160,   13],
       [  47,   39,  136, 5359,    1,  223,   34,   54,  142,   96],
       [  14,   27,   40,   11, 5356,    8,   54,   33,   94,  205],
       [  65,   40,   36,  186,   68, 4610,  102,   32,  195,   87],
       [  30,   24,   45,    2,   45,   98, 5614,    5,   54,    1],
       [  25,   17,   74,   29,   54,   13,    6, 5798,   19,  230],
       [  45,  139,   68,  154,   10,  150,   57,   25, 5080,  123],
       [  39,   32,   30,   86,  167,   34,    2,  203,   88, 5268]],
dtype=int64)
```

Run it on Notebook

Error Analysis

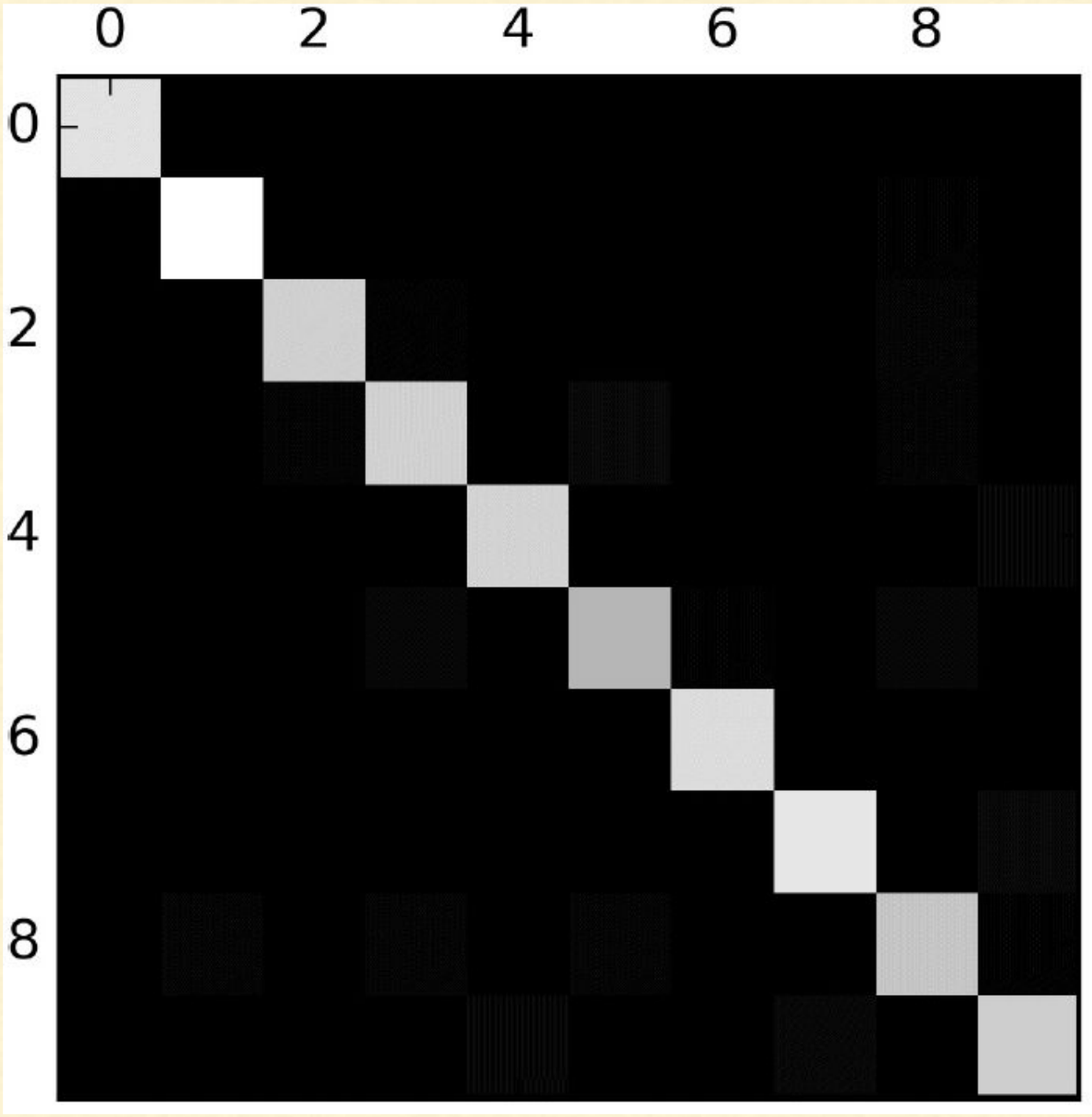
- Plotting confusion matrix for multiclass using Scikit-Learn

```
>>> plt.matshow(conf_mx, cmap=plt.cm.gray)
>>> plt.show()
```



Run it on Notebook

Error Analysis



Observations:

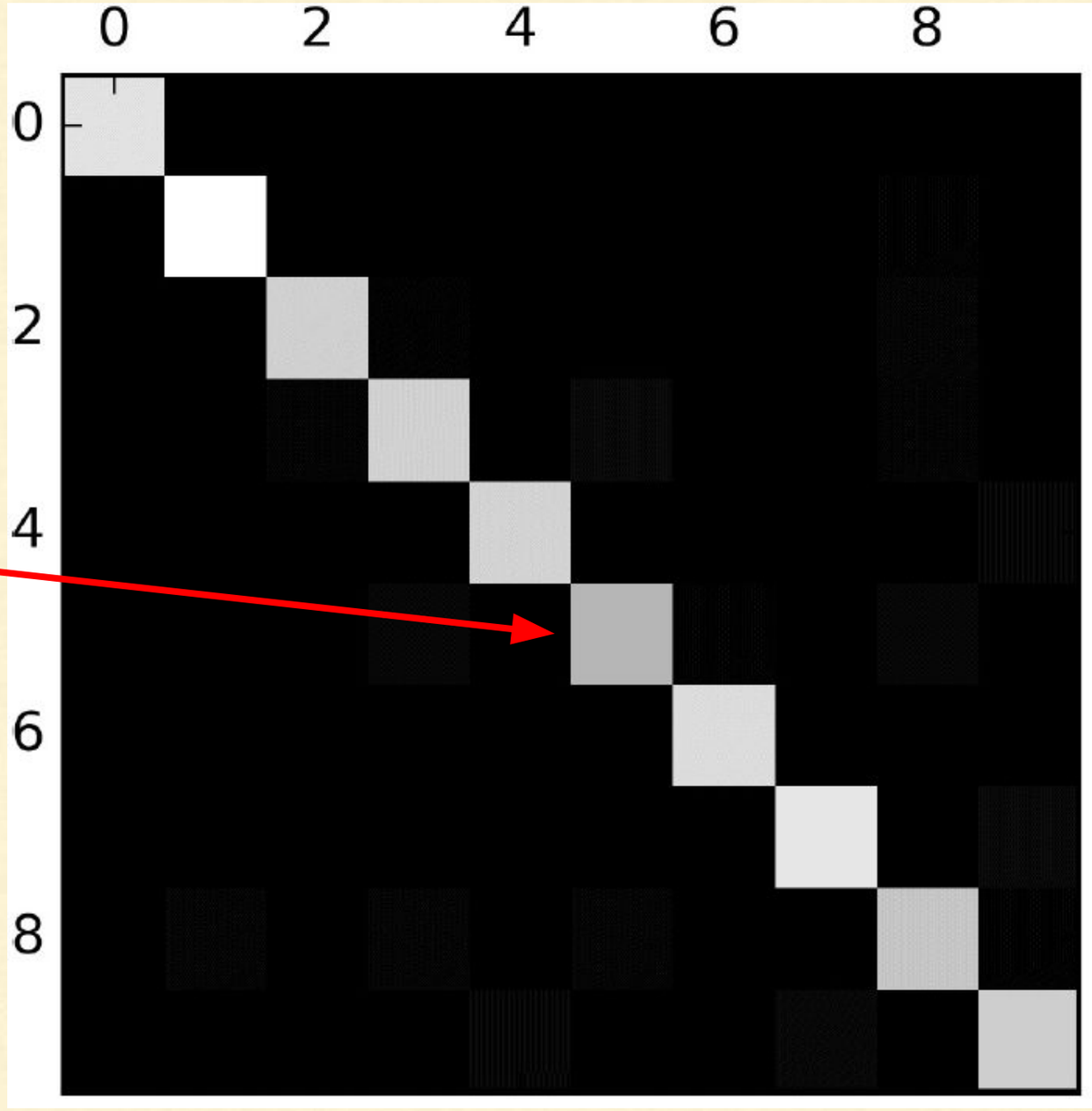
- ???

Error Analysis

Observations:

A. Looks fairly good since most images are on the diagonal

B. 5s looks darker than other digits, why?
   a. Fewer 5s in the dataset
   b. Classifier does not perform well on 5s
   c. Both

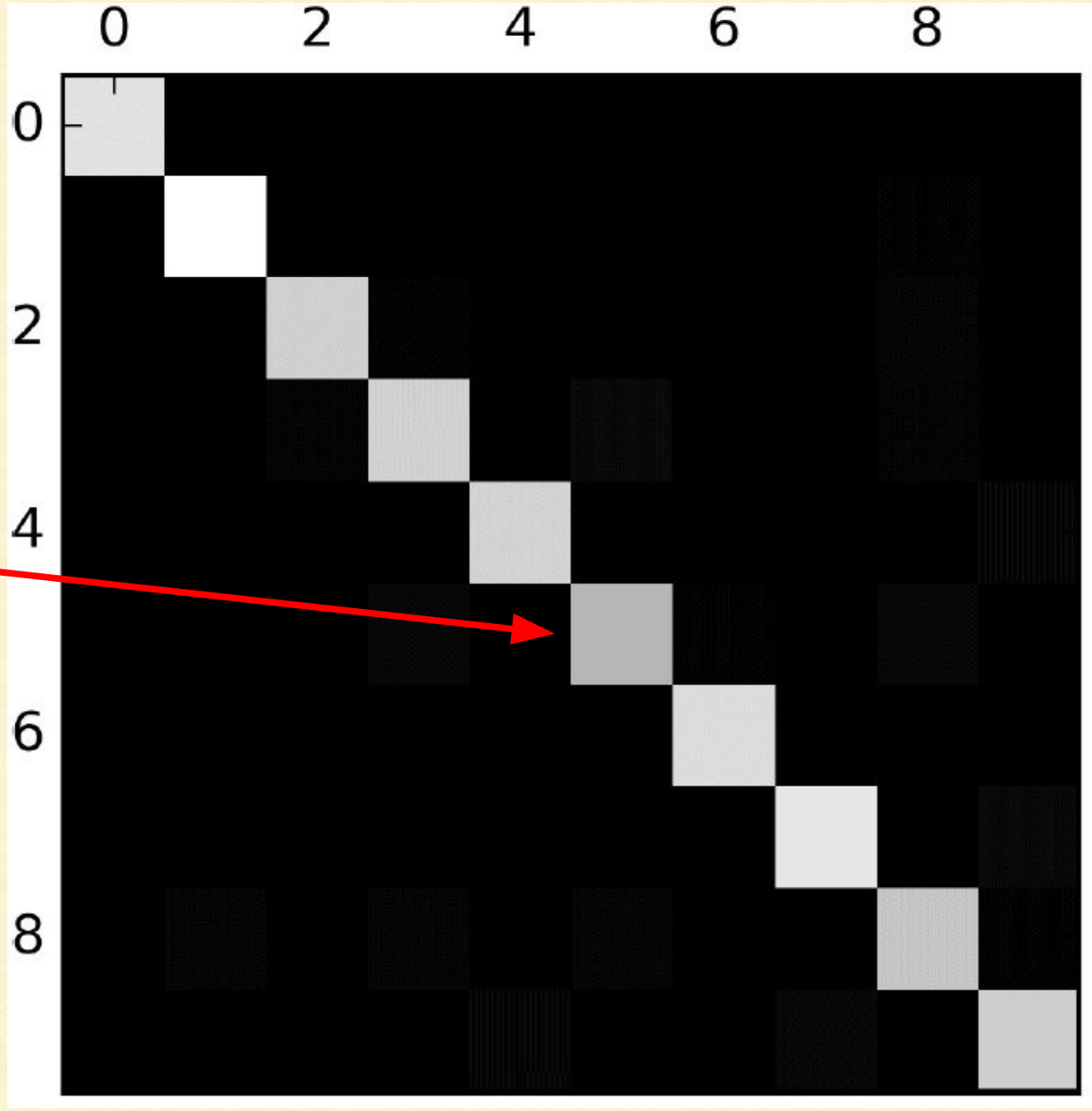


Error Analysis

Observations:

A. Looks fairly good since most images are on the diagonal

B. 5s looks darker than other digits, why?
   a. Fewer 5s in the dataset
   b. Classifier does not perform well on 5s
   c. **Both**

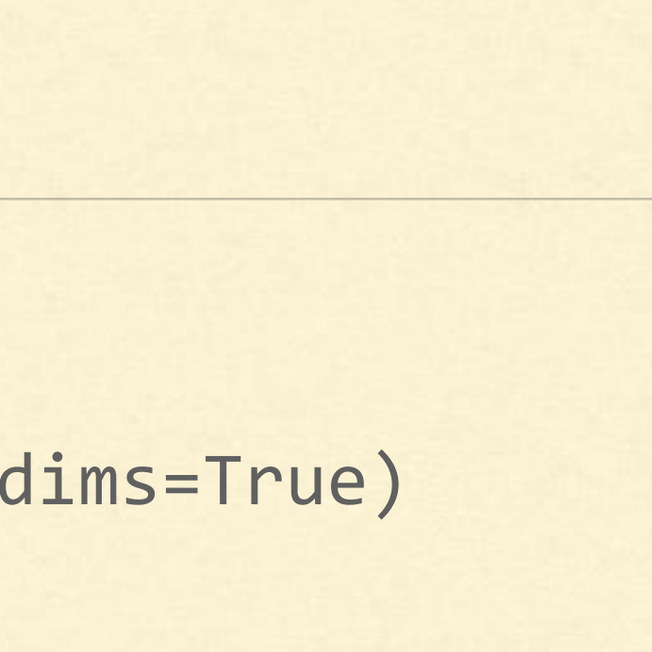


Error Analysis

Lets take a closer Look:

```
# Highlight the errors by normalizing
>>> row_sums = conf_mx.sum(axis=1, keepdims=True)
>>> norm_conf_mx = conf_mx / row_sums

#  Remove the correct predictions
>>> np.fill_diagonal(norm_conf_mx, 0)
>>> plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
>>> plt.show()
```

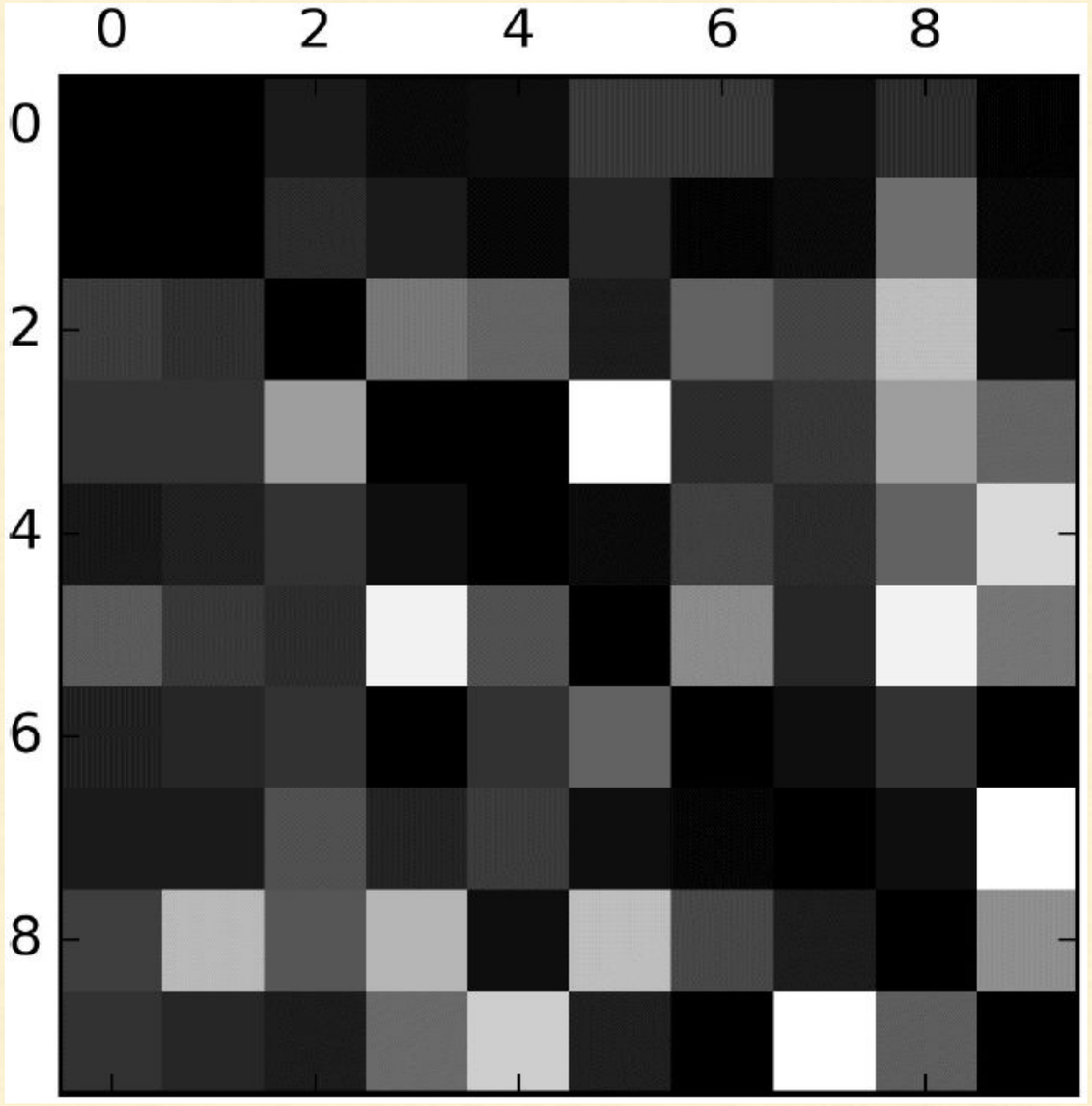


Run it on Notebook

Error Analysis

Observations:

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



Error Analysis

Observations:

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

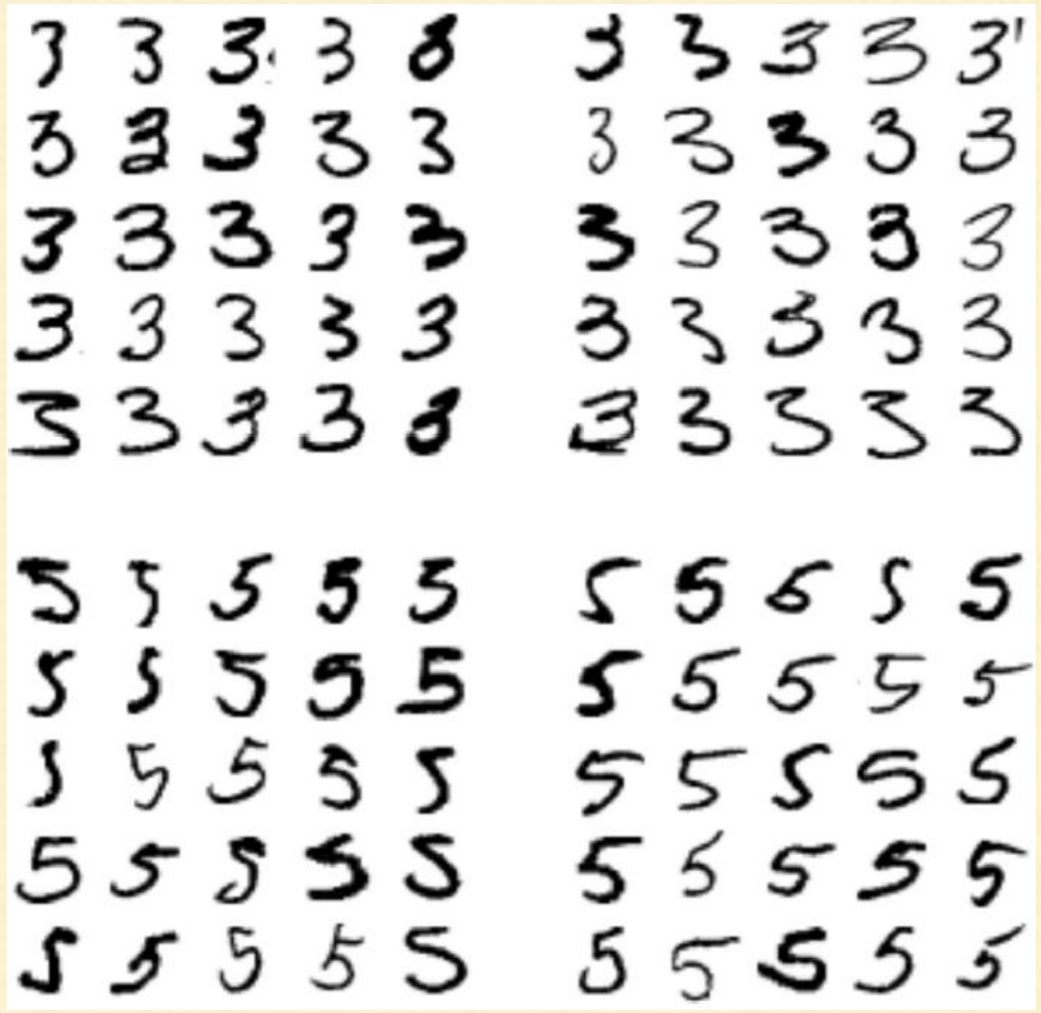
Possible solutions:

1. Gather more training data for 8s and 9s
2. Engineer new feature that would help the classifier like number of loops in the image of the digit
3. Preprocess images
   a. using Scikit-Image, Pillow, or OpenCV ...
   b. to make certain features stand out more
4. Analyzing individual errors

Error Analysis

Analyzing individual errors (plotting 3s and 5s)

Actual 3,
Predicted 3



Actual 3,
Predicted 5

Actual 5,
Predicted 3

Actual 5,
Predicted 5

Error Analysis

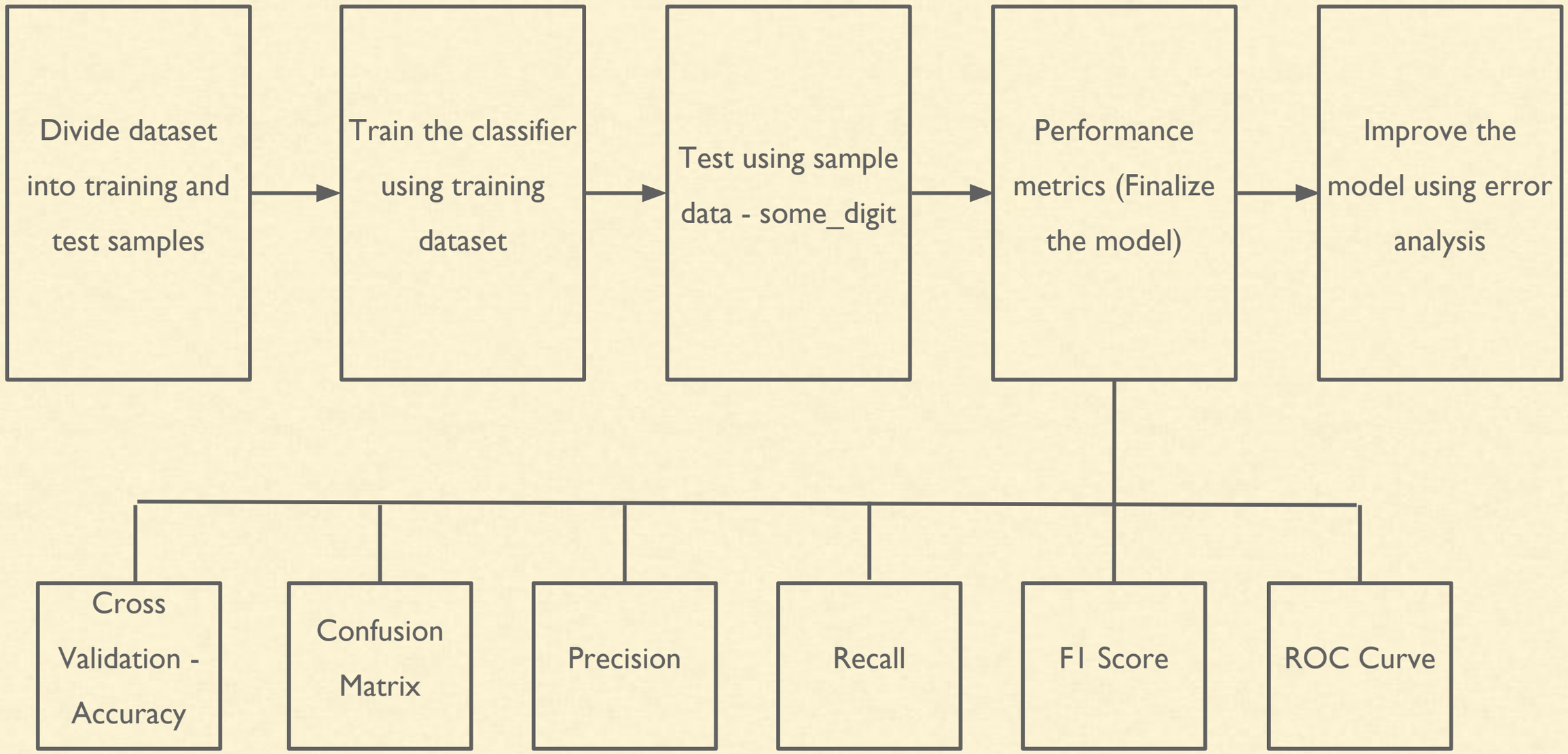
Observation:

- 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

Error Analysis - 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

Review till now?



What do we do next?

Completed:

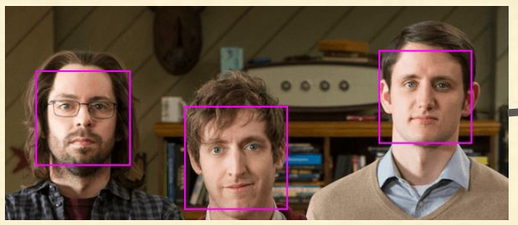
- Binary Classification
- Multi-class Classification

Next:

- Multi-label Classification
- Multi-output Classification

Multilabel Classification

- Explanation by example 1



Is Bob there in the image?

Is Alice there in the image?

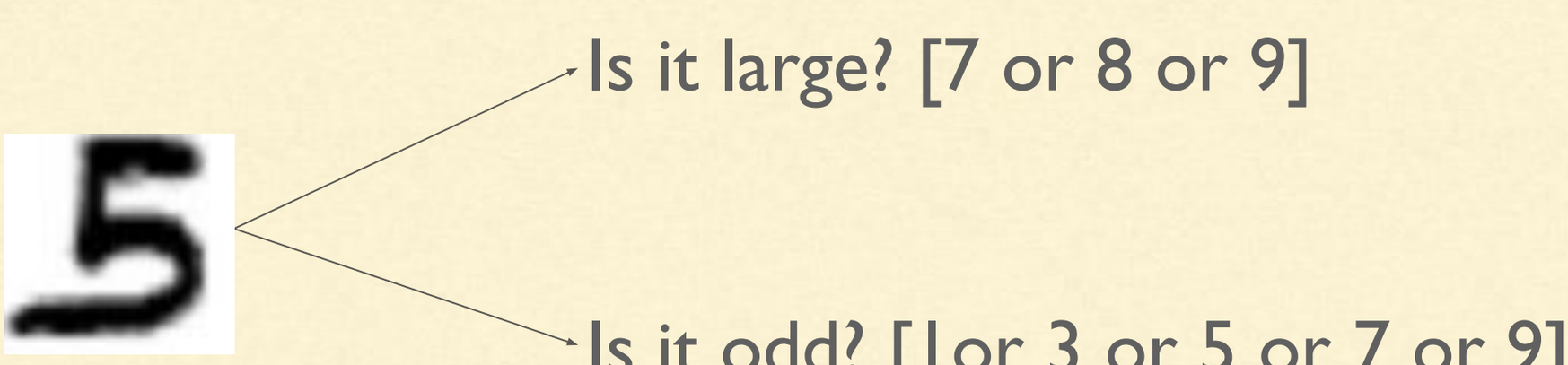
Is Charlie there in the image?

Multi-labels

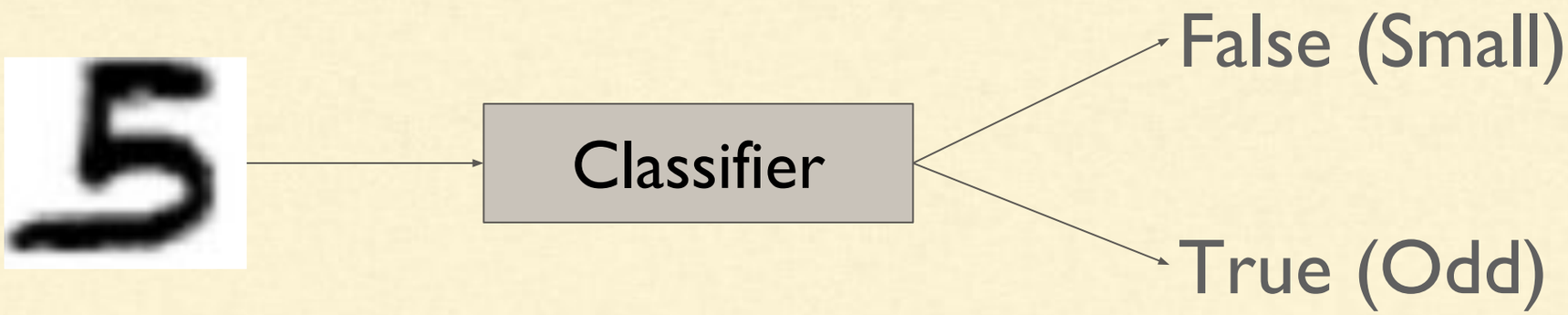
- So there are two classifications in the same image

Multilabel Classification

- Explanation by example 2

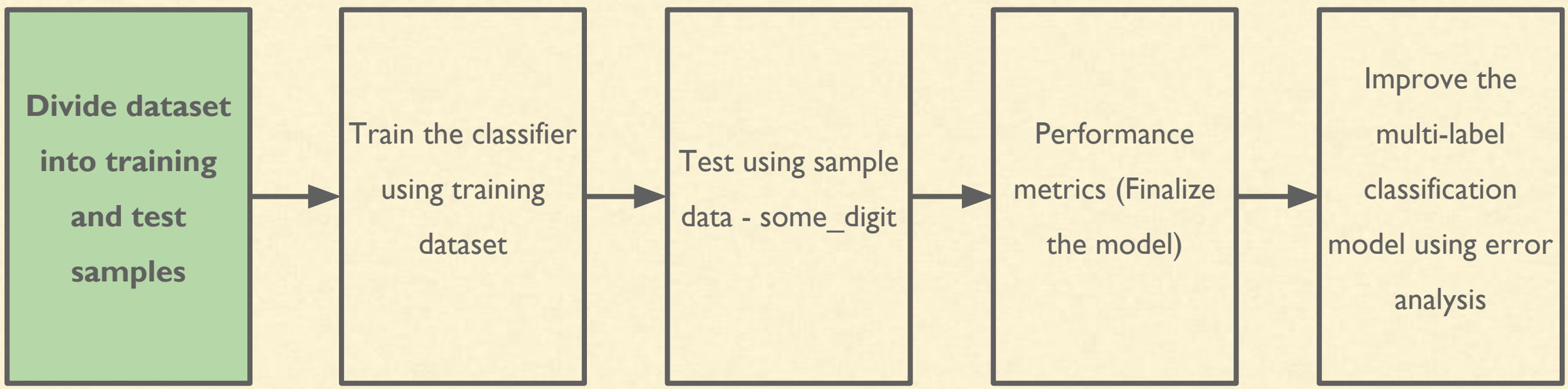


- So there are two classifications in the same image



Multilabel Classification - Example

- Classification of image into whether [large, odd], how do we do it?
  - Step 0: Fetch the data set (already done previously)




```
>>> from sklearn.datasets import fetch_mldata
>>> mnist = fetch_mldata('MNIST original')
>>> X, y = mnist["data"], mnist["target"]
>>> some_digit = X[36000]
```

Multilabel Classification - Example

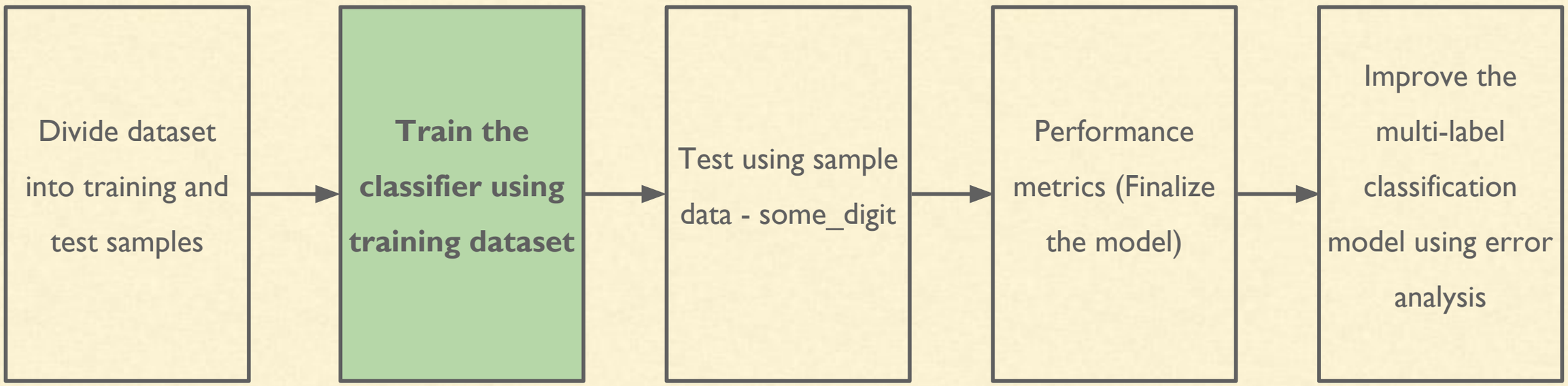
- Step 1: Divide the dataset into training and test samples (new labels according to the requirement)
  - Shuffling of dataset already done previously

```
>>> y_train_large = (y_train >= 7)
>>> y_train_odd = (y_train % 2 == 1)
>>> y_multilabel = np.c_[y_train_large, y_train_odd]  # np.c_ is used
to concatenate the two arrays element wise
```

Run it on Notebook

Multilabel Classification

- Step 2: Train the classifier
  - KNeighbours Classifier supports multi-label, classification

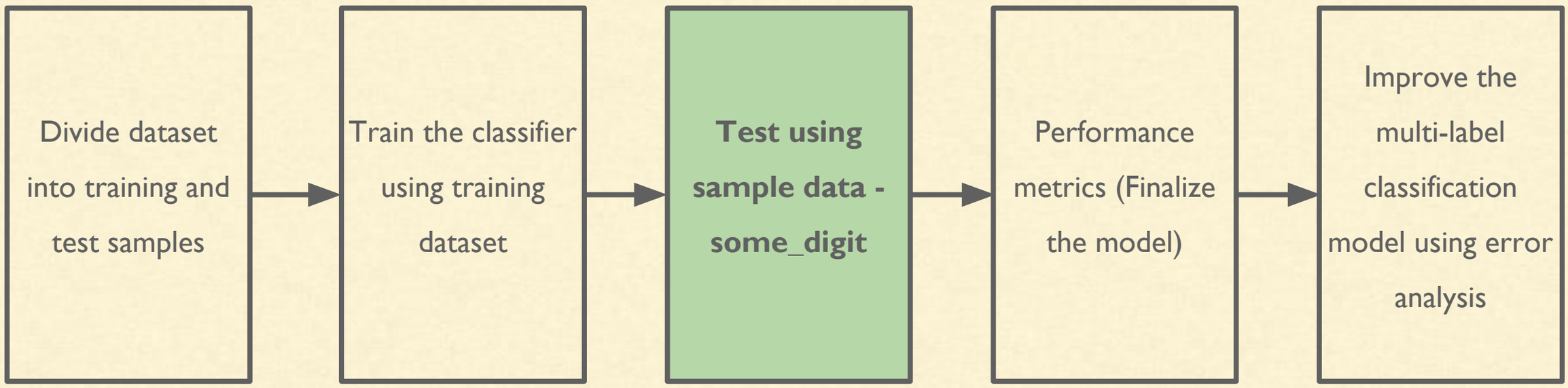



```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn_clf = KNeighborsClassifier()
>>> knn_clf.fit(X_train, y_multilabel)
```

Run it on Notebook

Multilabel Classification

- Step 3: Test the dataset

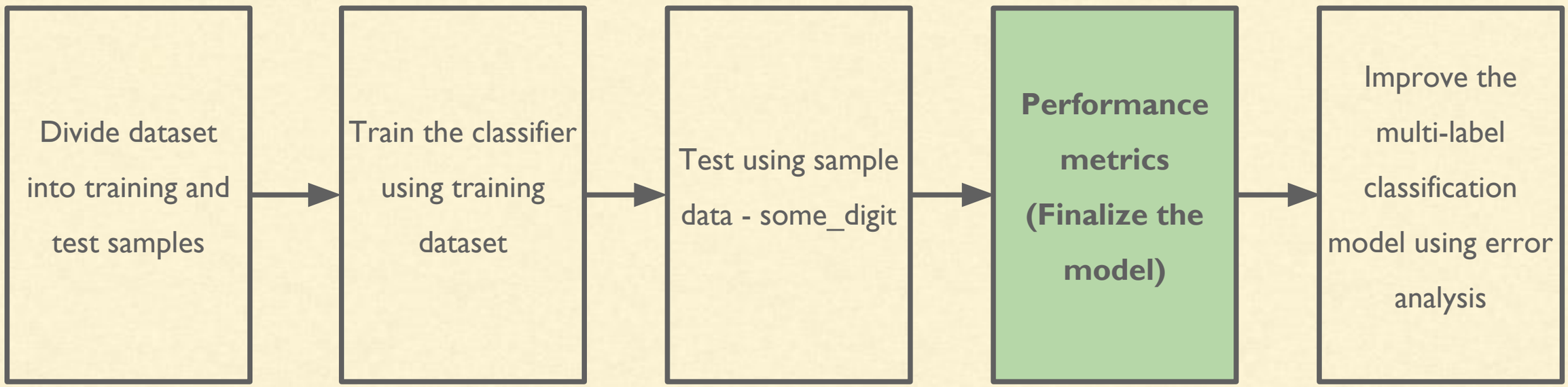



```
>>> knn_clf.predict([some_digit])
array([[False, True]], dtype=bool)
```

- Digit 5 is not large (i.e. it not 7 or 8 or 9) but is odd

Multilabel Classification

- Classification of image into whether [large, odd], how do we do it?




```
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_train,
cv=3)
>>> f1_score(y_train, y_train_knn_pred, average="macro")
0.9684554018028022 1
```

Run it on Notebook

Multi-Output Classification

- It is simply a generalization of multilabel classification where each label can be multiclass (i.e., it can have more than two possible values).

- Multi-output classification example:
  - **Removing noise from images**

- We'll build a system that removes noise from images. It will take as input a noisy digit image, and it will output a clean digit image, represented as an array of pixel intensities, just like the MNIST images.

Multi-Output Classification

- Notice that the classifier's output is
  - **Multilabel (one label per pixel) - so 784 labels**
  - **Multiclass**: And each label can have **multiple values** (pixel intensity ranges from 0 to 255) - 256 classes.

It is thus an example of a **Multioutput classification system**.

Multi-Output Classification

144 One label	200 One label	64 One label
94 One label	105 One label	74 One label
149 One label	200 One label	123 One label

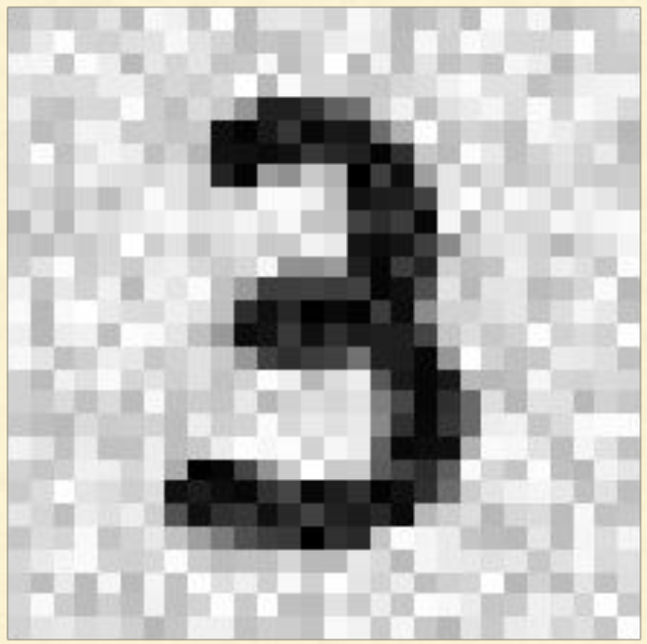
28 columns

0 to 255 values possible

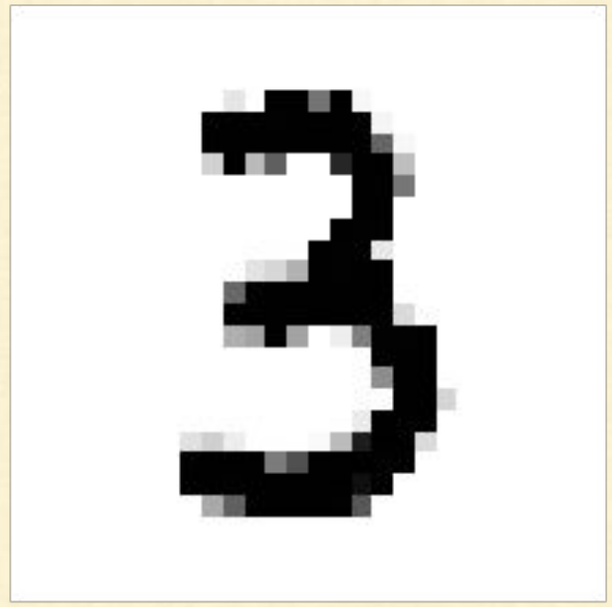
28 rows

Multi-Output Classification

Our system will remove noise from images. Input will be a noisy digit image, and it will output a clean digit image, represented as an array of pixel intensities, just like the MNIST images.



Input Image



Output Image

Multi-Output Classification

Demonstrating multi-output classifier: digit image noise removal

- Adding noise to the training set

```
>>> import numpy.random as rnd
>>> noise_train = rnd.randint(0, 100,
(len(X_train), 784))
>>> X_train_mod = X_train + noise_train
```

Run it on Notebook

Multi-Output Classification

- Adding noise to the test set

```
>>> noise_test = rnd.randint(0, 100, (len(X_test),
784))
>>> X_test_mod = X_test + noise_test
```

Run it on Notebook

Multi-Output Classification

- Setting clean image as the label (y_train and y_test)

```
>>> y_train_mod = X_train
>>> y_test_mod = X_test
```

Run it on Notebook

Multi-Output Classification

- Let us view the noisy image

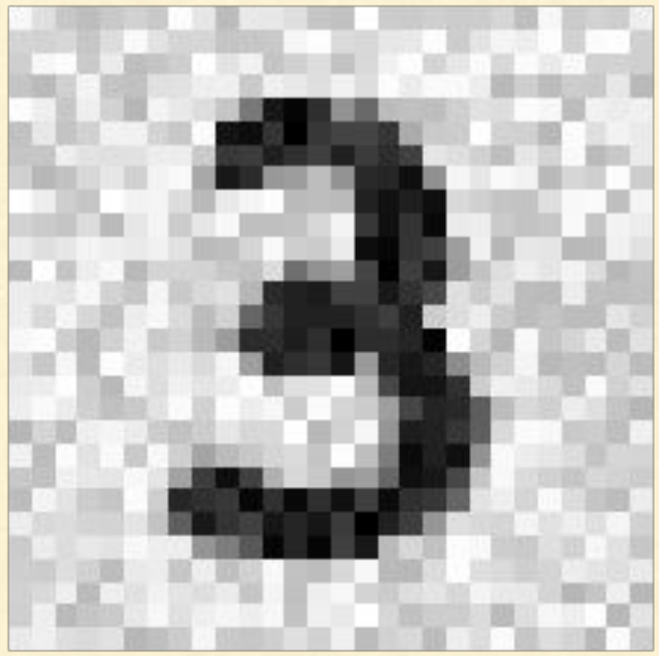
```
>>> def plot_digit(array):
    array_image = array.reshape(28, 28)
    plt.imshow(array_image, cmap = matplotlib.cm.binary,
interpolation="nearest")
    plt.axis("off")
    plt.show()

>>> plot_digit(X_test_mod[4000])
```

Run it on Notebook

Multi-Output Classification

- Let us view the noisy image



`Run it on Notebook`

Multi-Output Classification

- Let us clean the image using the Classifier

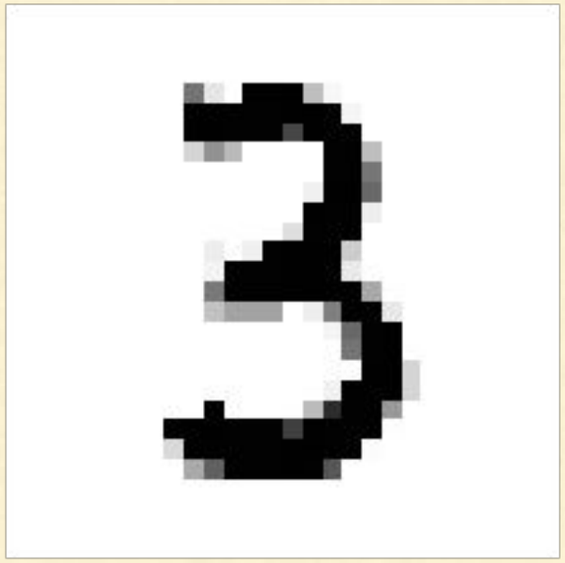
```
>>> knn_clf.fit(X_train_mod, y_train_mod)
>>> clean_digit = knn_clf.predict([X_test_mod[4000]])
```

Run it on Notebook

Multi-Output Classification

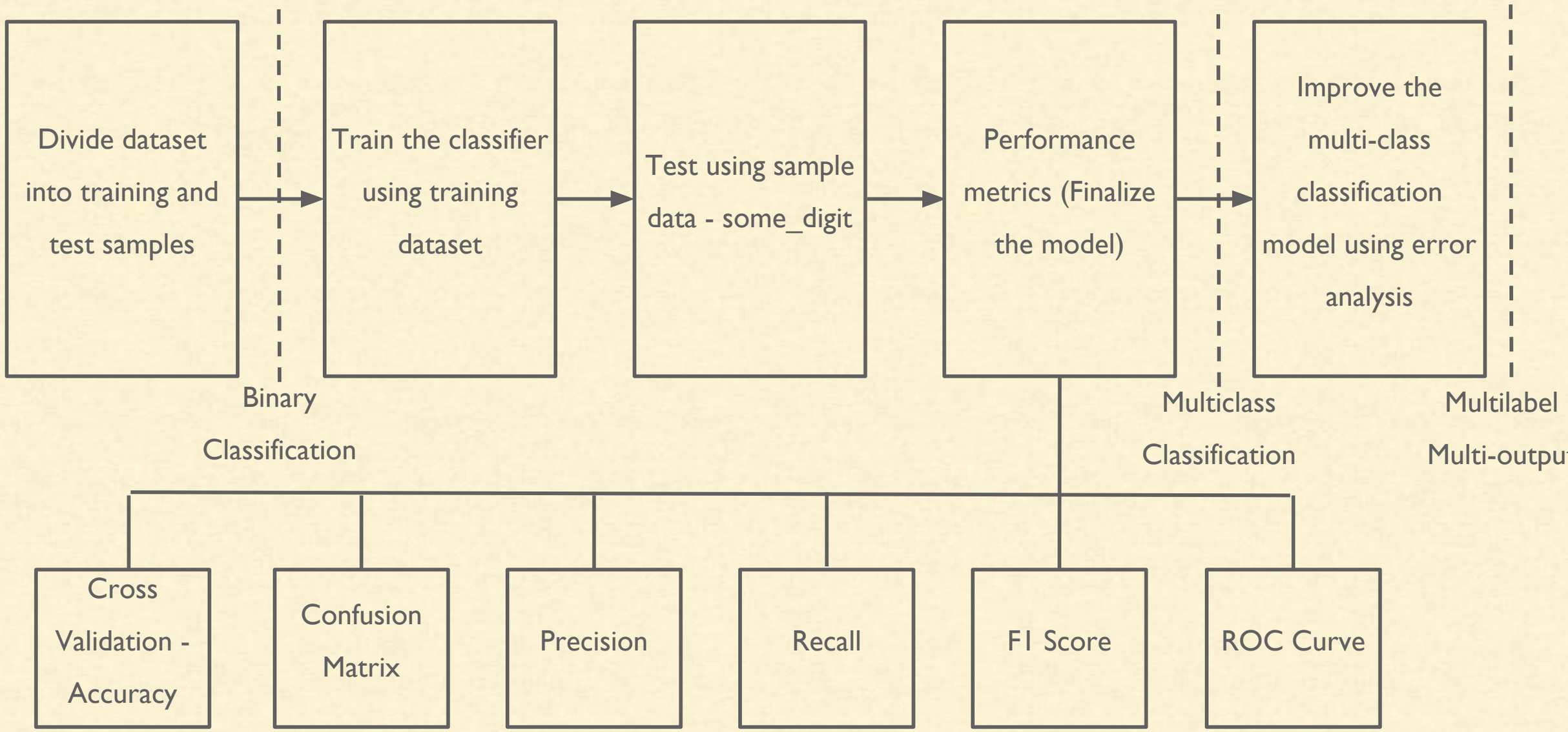
- Plotting the clean image

```
>>> plot_digit(clean_digit)
```



Run it on Notebook

Review till now?



Questions?

https://discuss.cloudxlab.com

reachus@cloudxlab.com

Archives

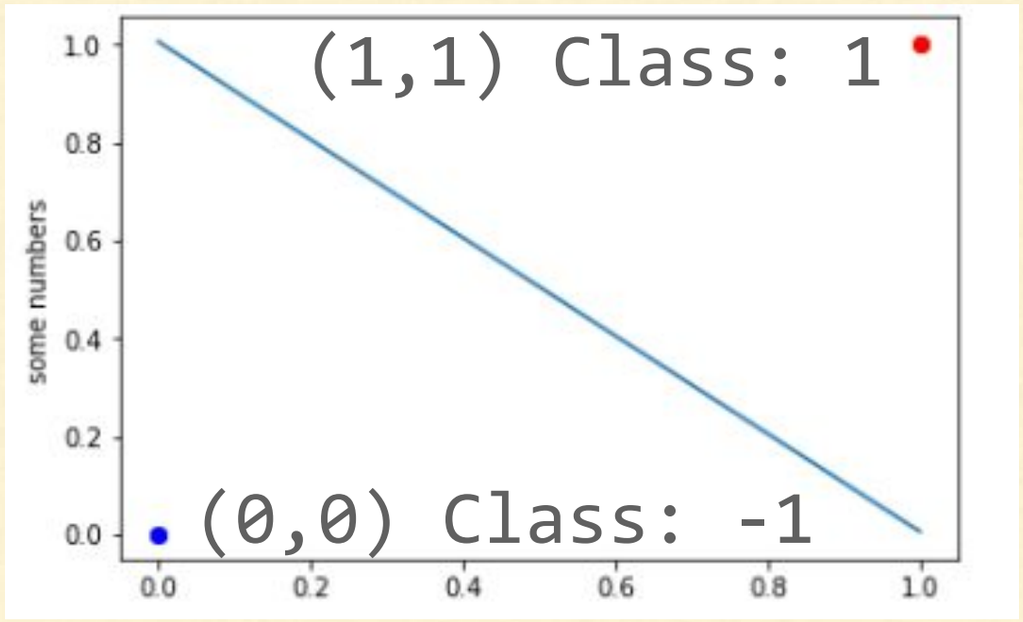
Stochastic Gradient Descent (SGD) Classifier

What is hinge loss?

Stochastic Gradient Descent (SGD) Classifier

- What is hinge loss?

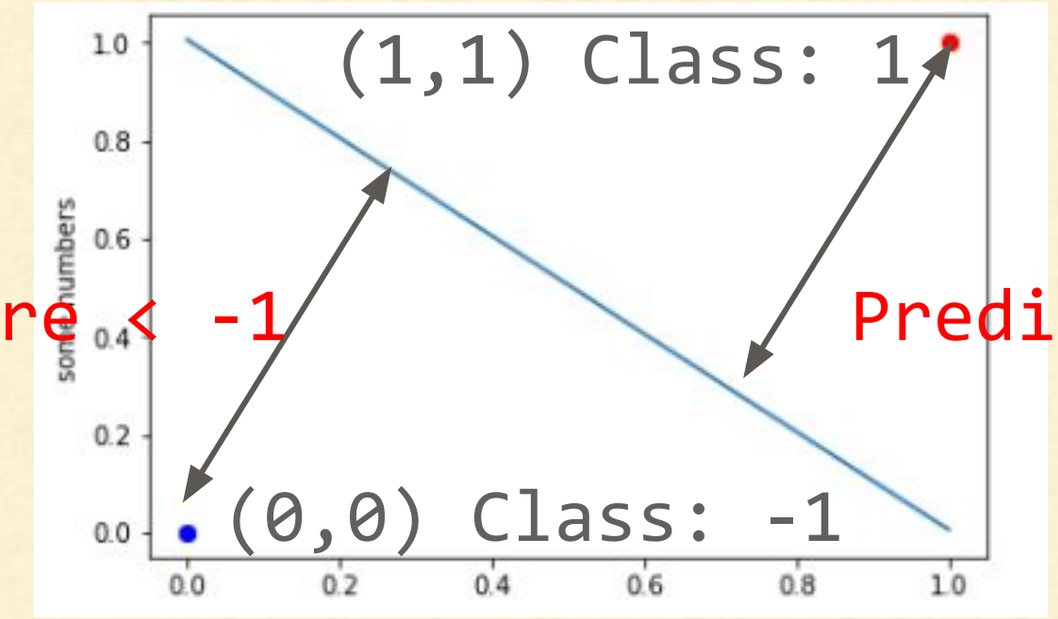
  - For the training data, there are two labels:
    - y_true = 1 or
    - y_true = -1



Stochastic Gradient Descent (SGD) Classifier

- Prediction_score denotes how far is the training set from the classifier
  - +ve: on the positive side
  - -ve: on the negative side
  - | prediction_score | > 1

prediction_score < -1

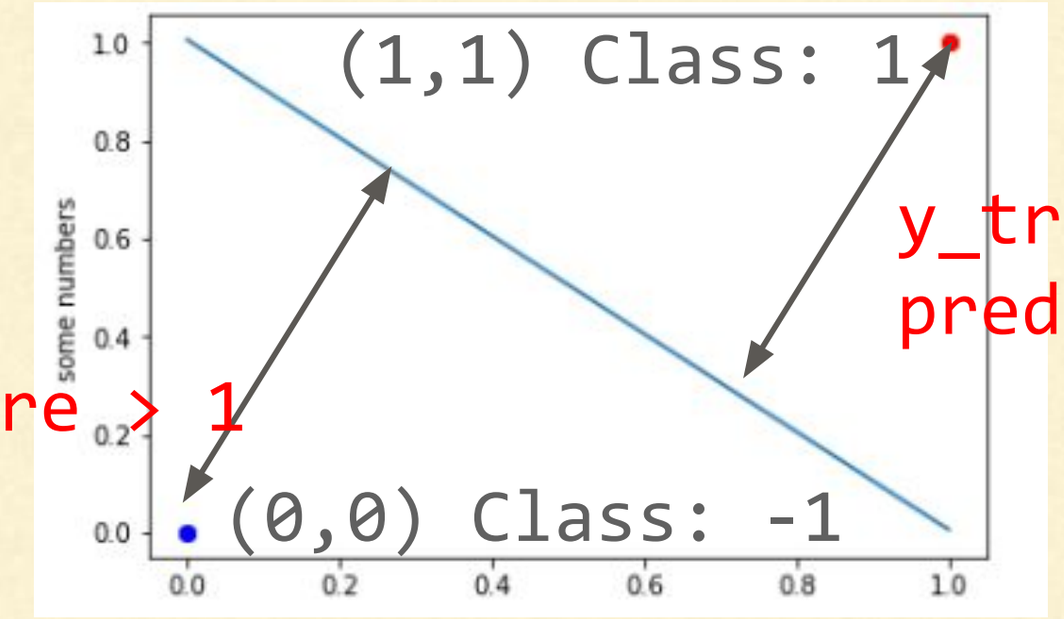


Prediction_score > 1

Stochastic Gradient Descent (SGD) Classifier

- y_true * prediction_score becomes
  - negative and less than -1 for incorrect prediction
  - Positive and greater than 1 for correct prediction

y_true *
prediction_score > 1

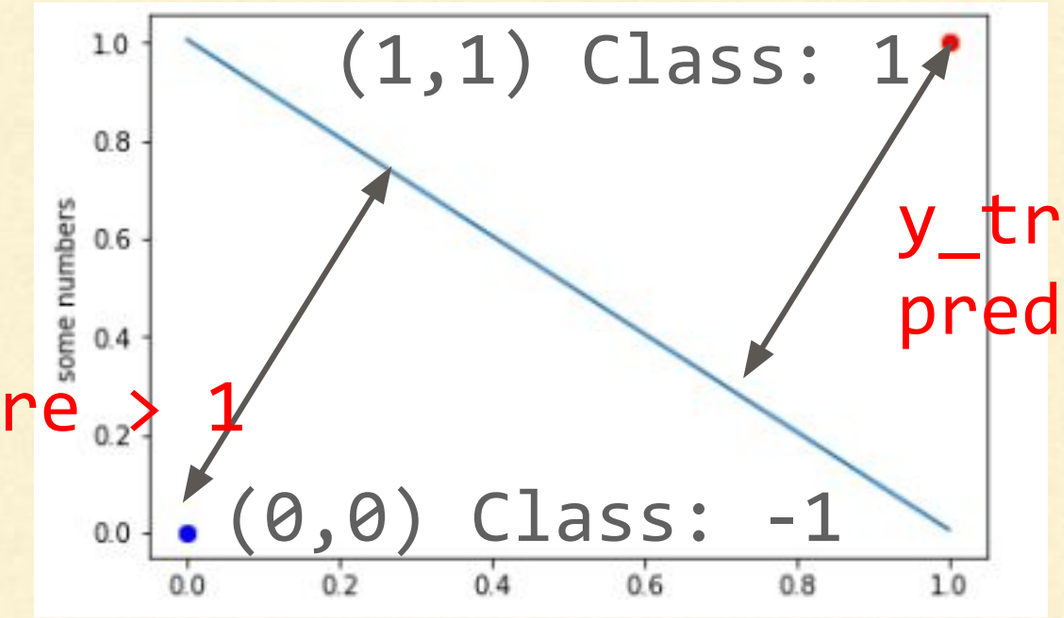


y_true *
prediction_score > 1

Stochastic Gradient Descent (SGD) Classifier

- y_true * prediction_score becomes
  - negative and less than -1 for incorrect prediction
  - positive and greater than 1 for correct prediction

y_true *
prediction_score > 1

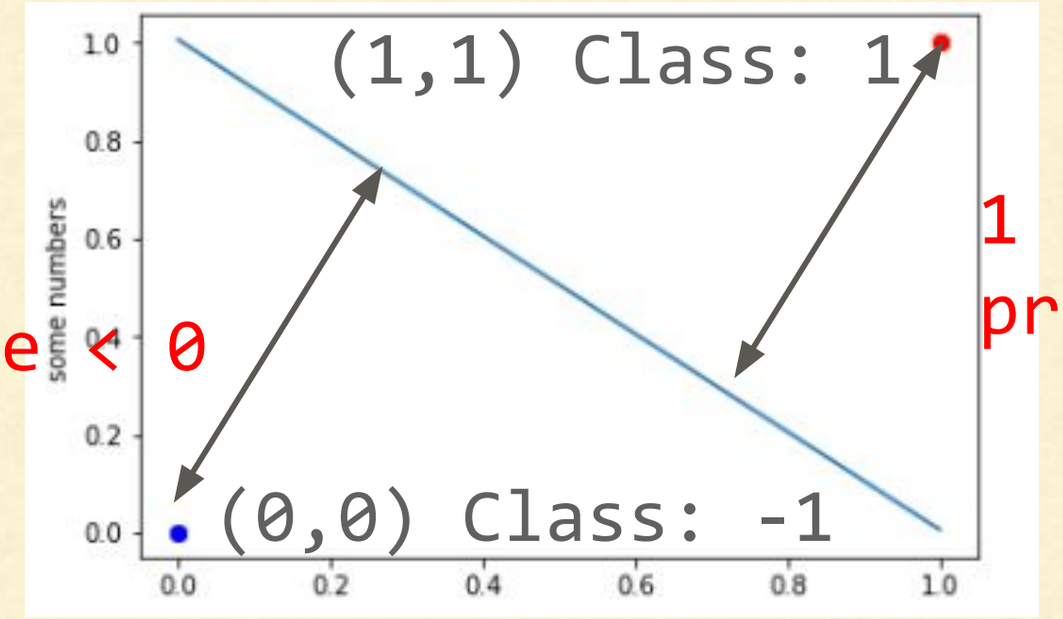


y_true *
prediction_score > 1

Stochastic Gradient Descent (SGD) Classifier

- 1 - y_true * prediction_score becomes
  - negative and less than -1 for correct prediction
  - positive and greater than 1 for incorrect prediction

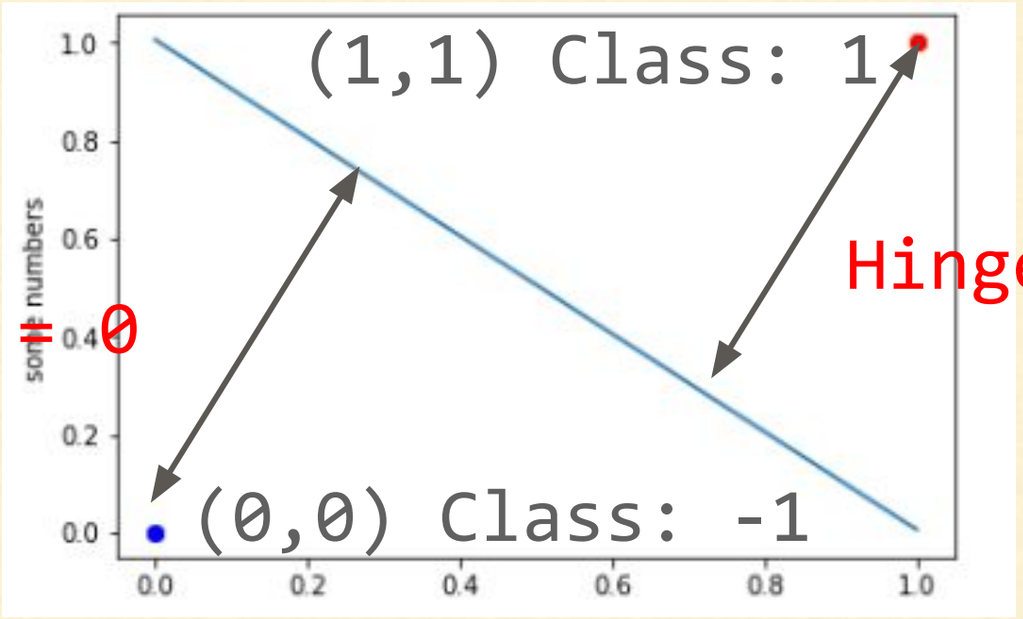
1 - y_true * prediction_score < 0



1 - y_true * prediction_score < 0

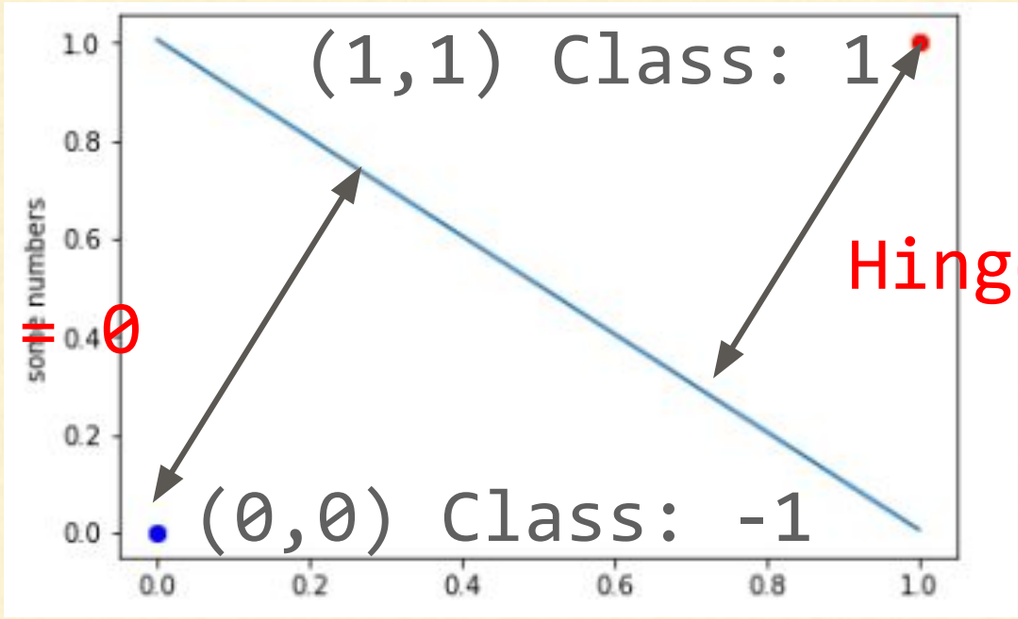
Stochastic Gradient Descent (SGD) Classifier

- Hinge Loss = max(0,1 - y_true * prediction_score) becomes
  - 0 and less than -1 for correct prediction
  - positive and greater than 1 for incorrect prediction



Stochastic Gradient Descent (SGD) Classifier

- Hinge Loss = max(0,1 - y_true * prediction_score) becomes
  - 0 and less than -1 for correct prediction
  - positive and greater than 1 for incorrect prediction
- Minimize the hinge loss

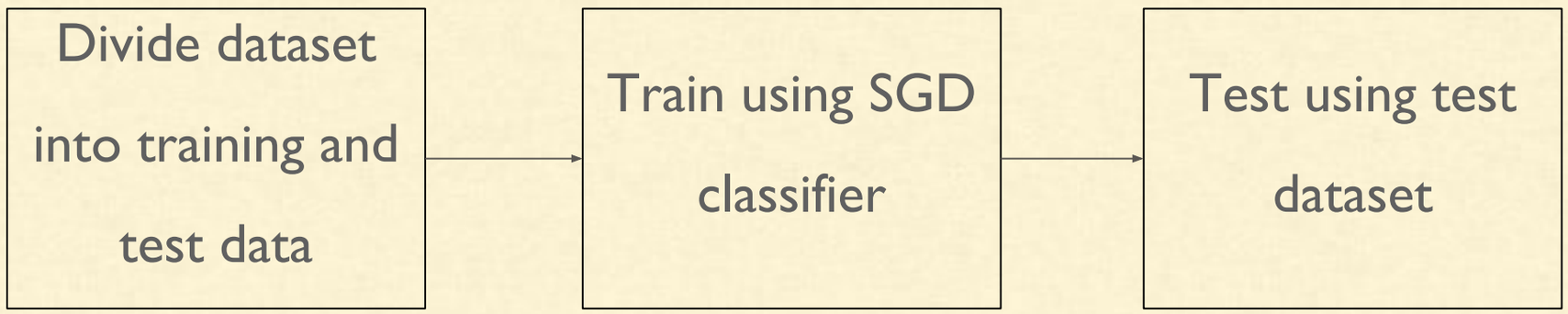


Hinge loss = 0

Hinge loss = 0

Performance measure - Cross Validation

- Cross Validation using StratifiedKFold of scikit-learn involves:
  - Custom code to split the training dataset into K folds
  - Evaluate the performance on the test data and print the performance

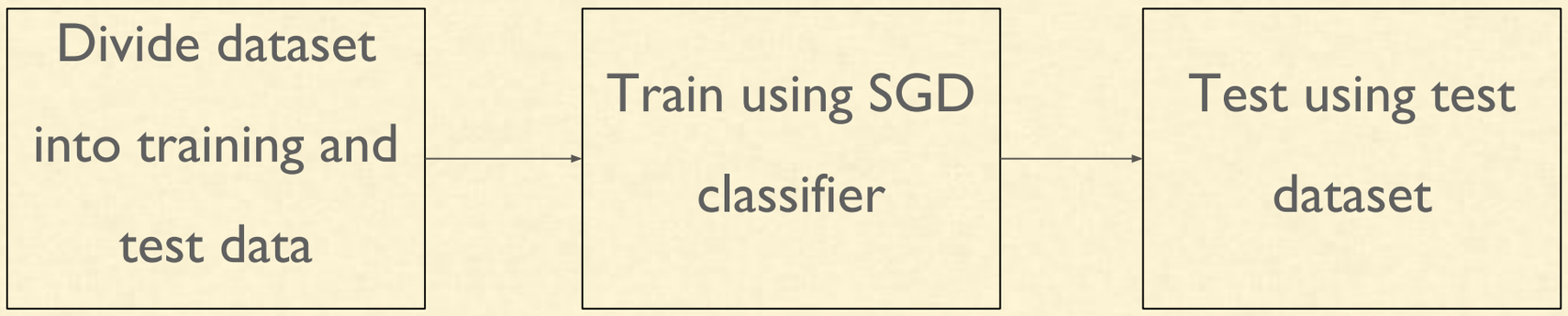


Performance measures - Never5Classifier

- '5' and 'Not 5' classifier
- *Cross_val_score*: As in the previous lecture, use cross_val_score to perform cross validation

```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.909 , 0.90715, 0.9128 ])
```

```
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)
```



Stochastic Gradient Descent (SGD) Classifier

The class SGDClassifier implements

- a plain stochastic gradient descent learning routine
- supports different loss functions and
  - Loss function defines the type of classifier: hinge (linear), etc.
  - Other options to be discussed later are: ‘log’, ‘squared_hinge’, ‘perceptron’

$$y = w_1 + w_2 x$$

Loss function: hinge

Stochastic Gradient Descent (SGD) Classifier

The class SGDClassifier implements

- a plain stochastic gradient descent learning routine
- supports different loss functions and
- penalties for classification
  - L2: least squares
  - L1: mod of the error

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2.$$

Penalties: L2

Stochastic Gradient Descent (SGD) Classifier

- Start with a certain weight vector [w1, w2]
- And a certain learning factor (eta)
- Iterate and update the weight vector using the training dataset until further change is minimum

$$y = w_1 + w_2 x$$

Loss function: hinge

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2.$$

Penalties: L2

$$w := w - \eta \nabla Q_i(w).$$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \eta \begin{bmatrix} 2(w_1 + w_2 x_i - y_i) \\ 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}$$

Learning rate: eta

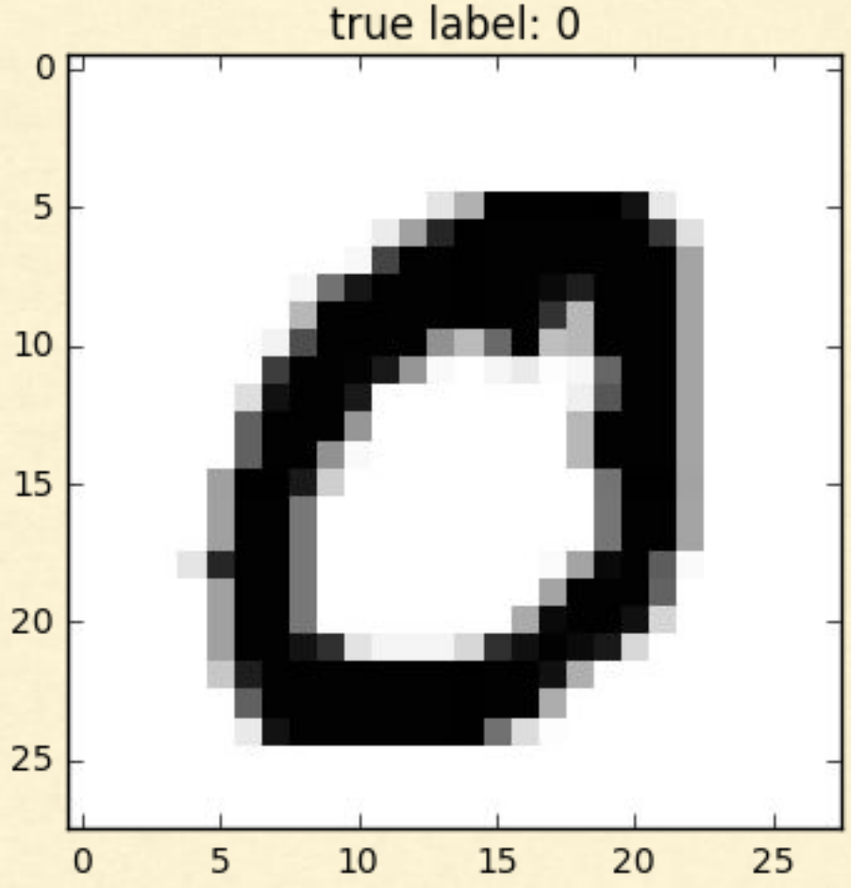
Dataset

MNIST dataset

- Set of 70,000 small handwritten images of digits
- Labelled by the digits it represents

Sample images

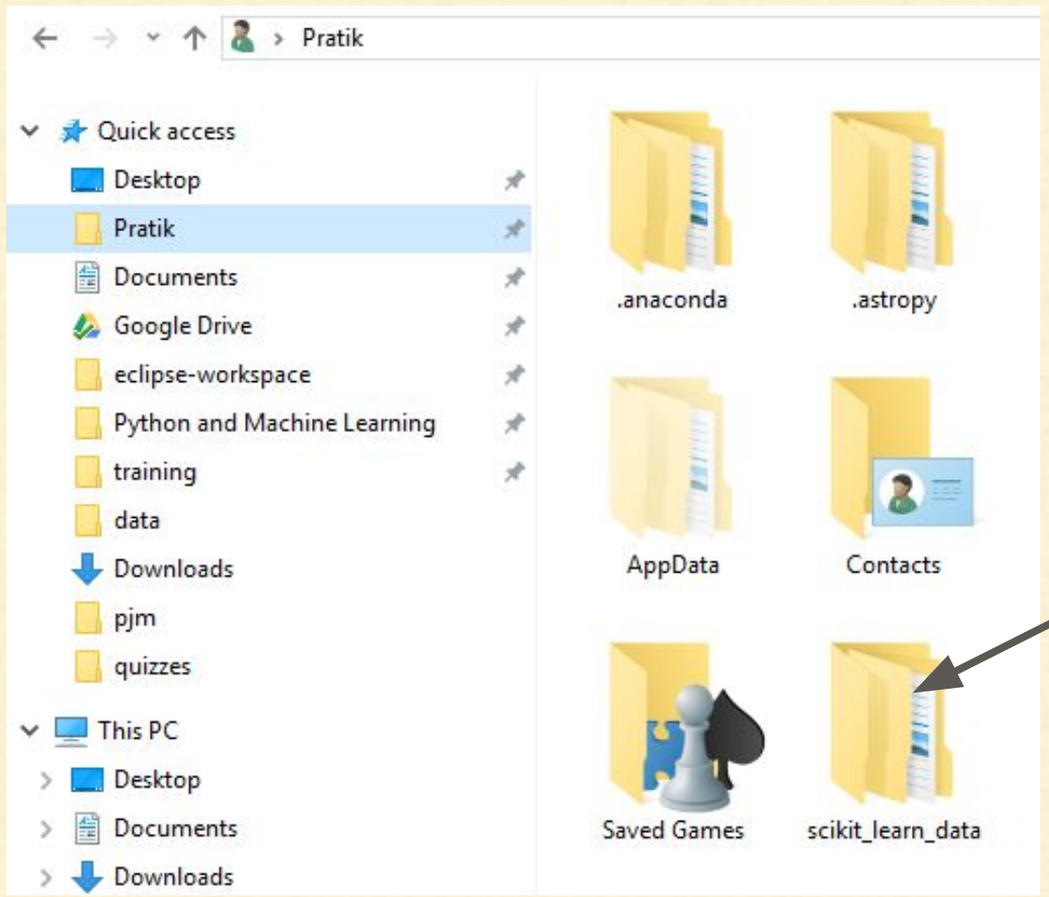
Dataset image



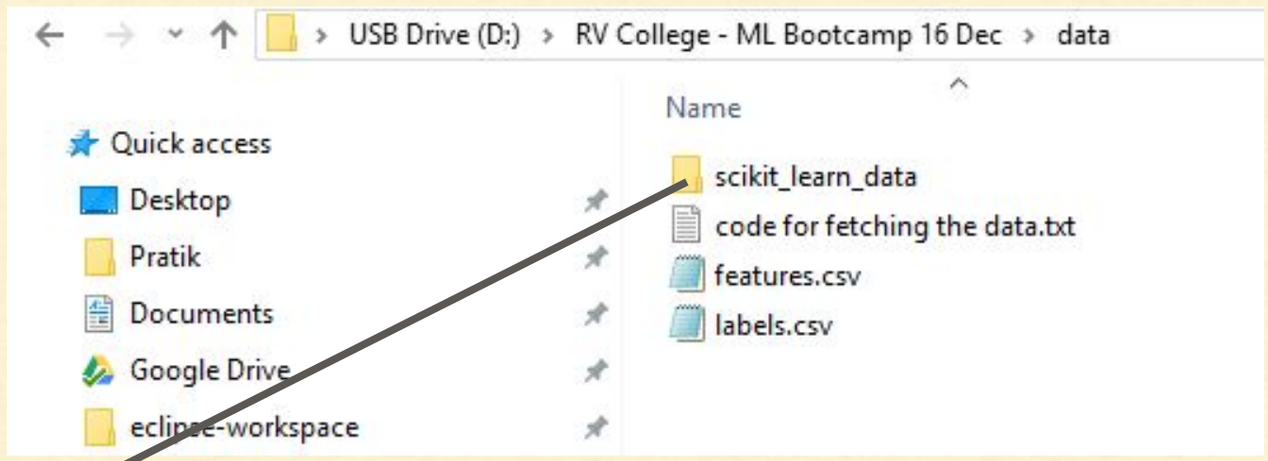
Setting Up!

Copy ‘scikit_learn_data’ from the usb to the home folder.

Local Home Folder

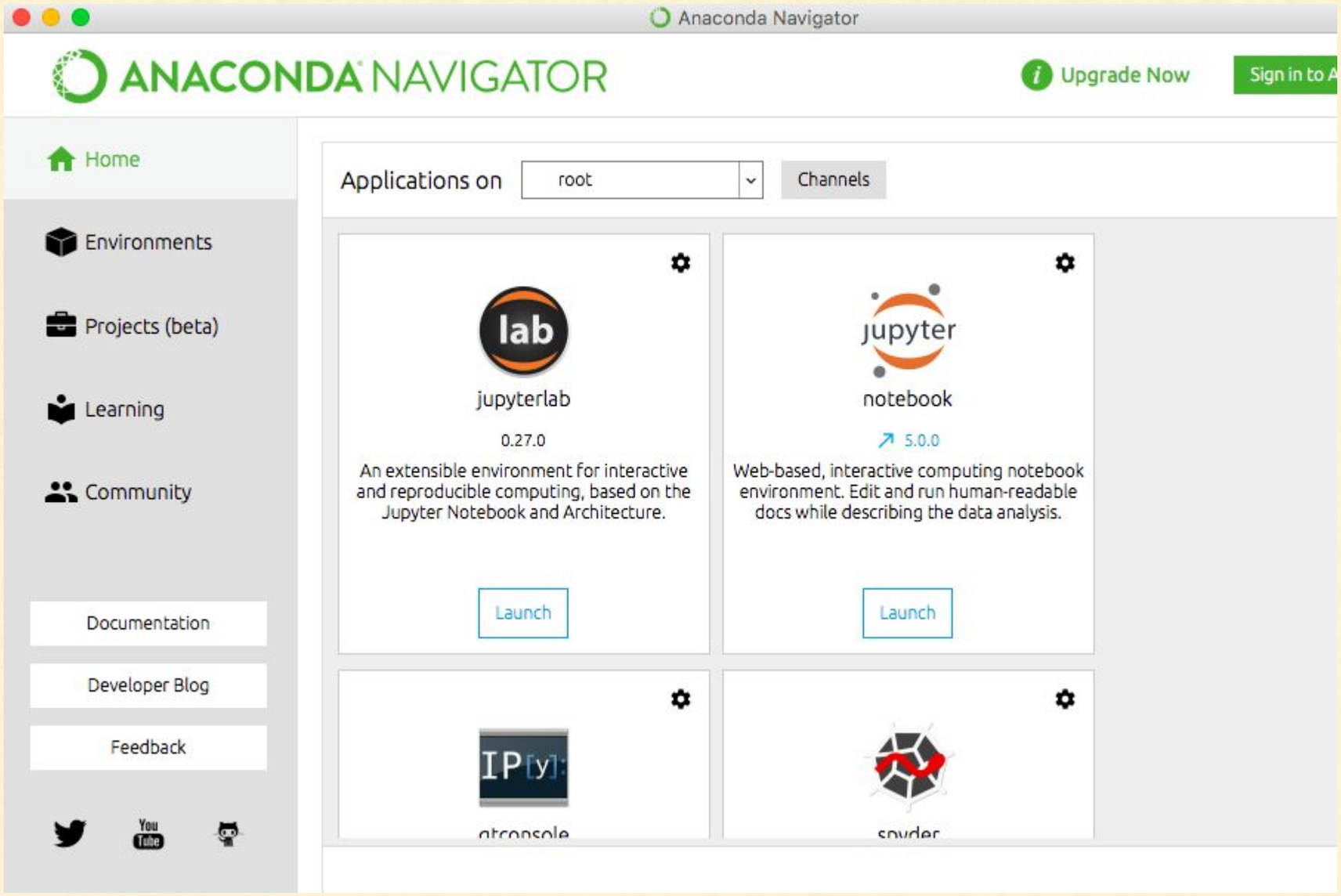


USB Drive



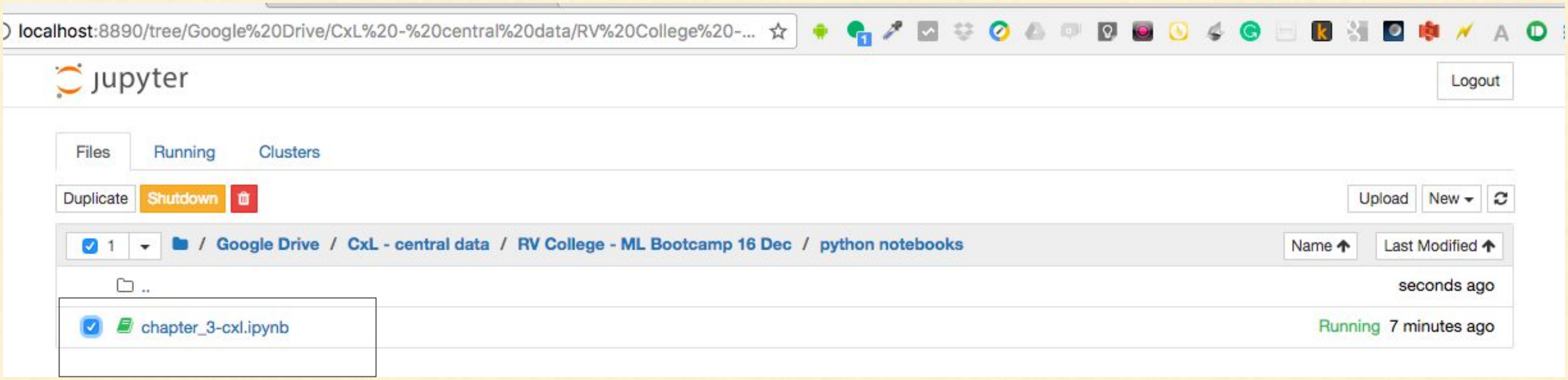
Setting Up!

Start Anaconda Launcher - Open Jupyter notebok



Setting Up!

Navigate to the folder in which you copied the whole USB



Multi-class Classification

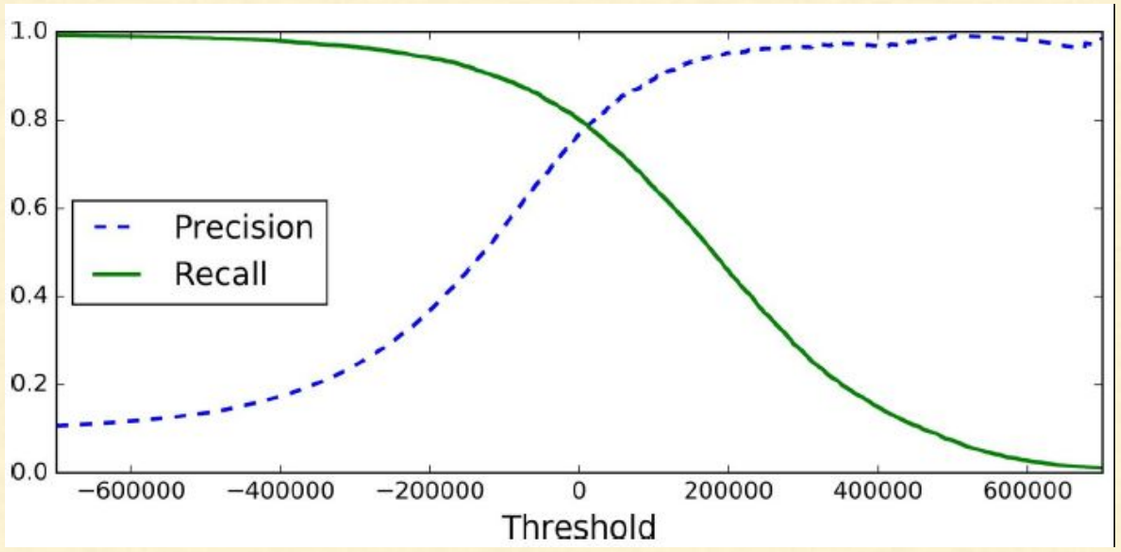
- Doing multi-class classification using OvO Classifier
  - Random Forest classifier directly does multi-class classification
    - No OvO or OvA strategy
  - Result: probability of the digit being 5 is the highest: 0.8

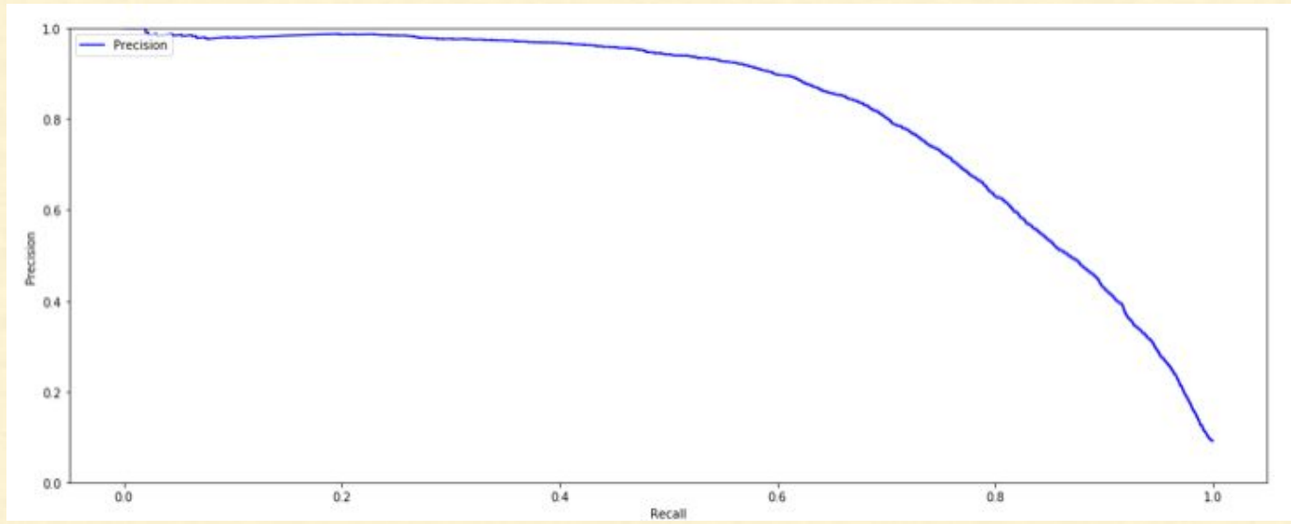
```
>>> forest_clf.fit(X_train, y_train)
>>> forest_clf.predict([some_digit])
>>> forest_clf.predict_proba([some_digit])
array([[ 0.1,  0. ,  0. ,  0.1,  0. ,  0.8,  0. ,  0. ,  0. ,  0. ]])
```

Run it on Notebook

Precision / Recall Curve

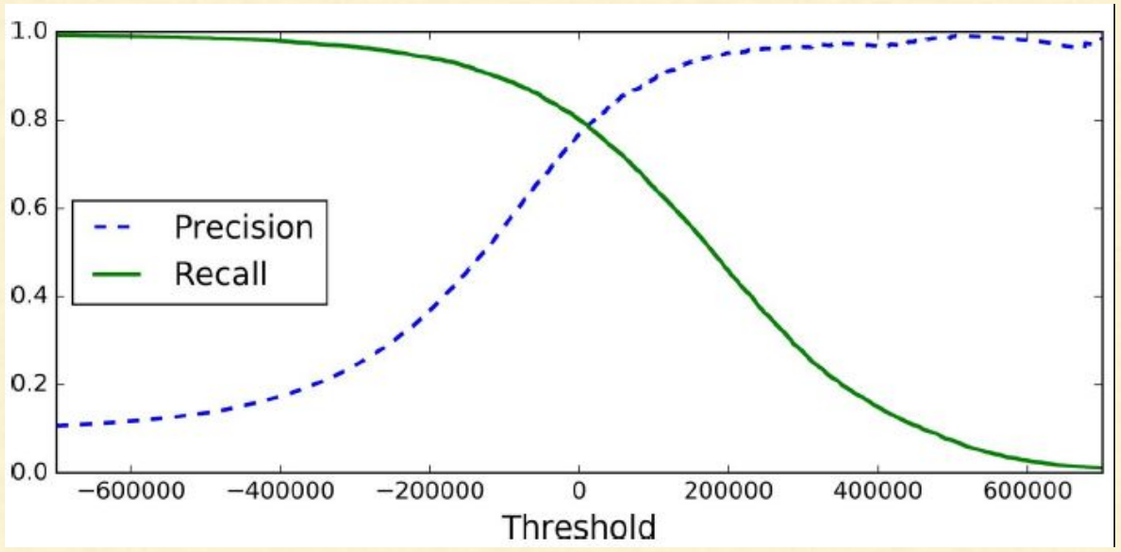
Q. How to decide the best threshold? What is the best threshold for below?

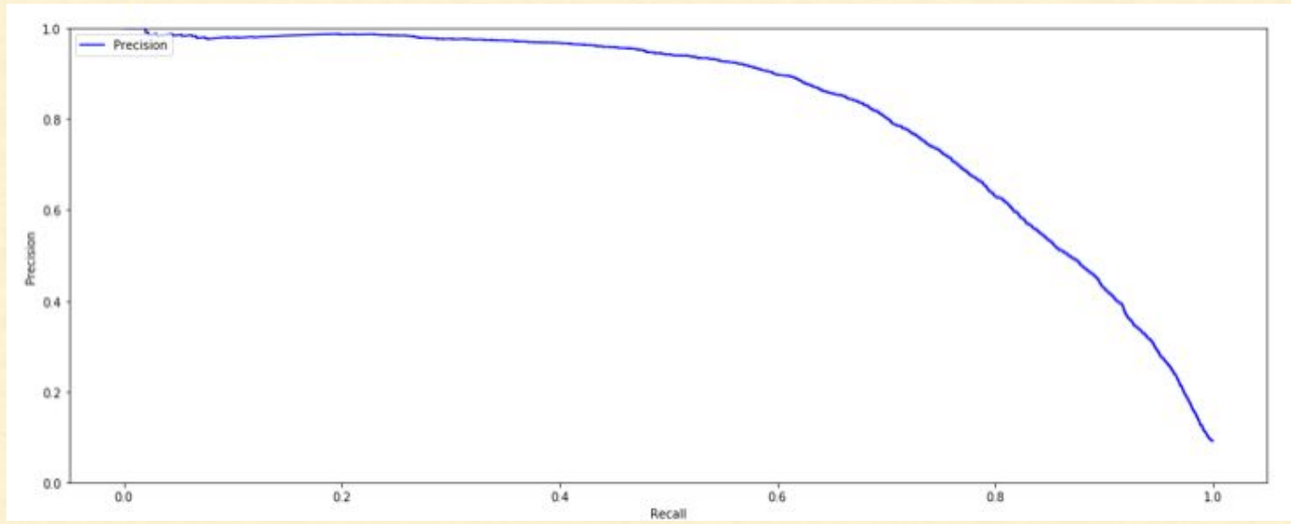




Precision / Recall Curve

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





Ans. **Depends on the requirement of the project.**