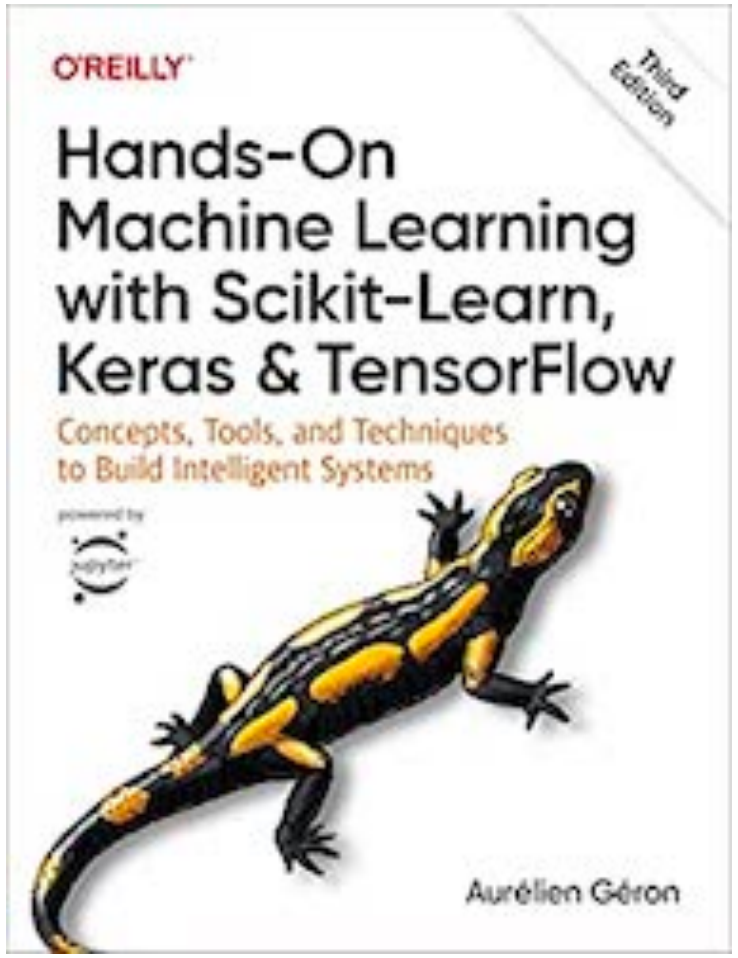
Machine Learning Security

3 Classification



Made Aug 26, 2023

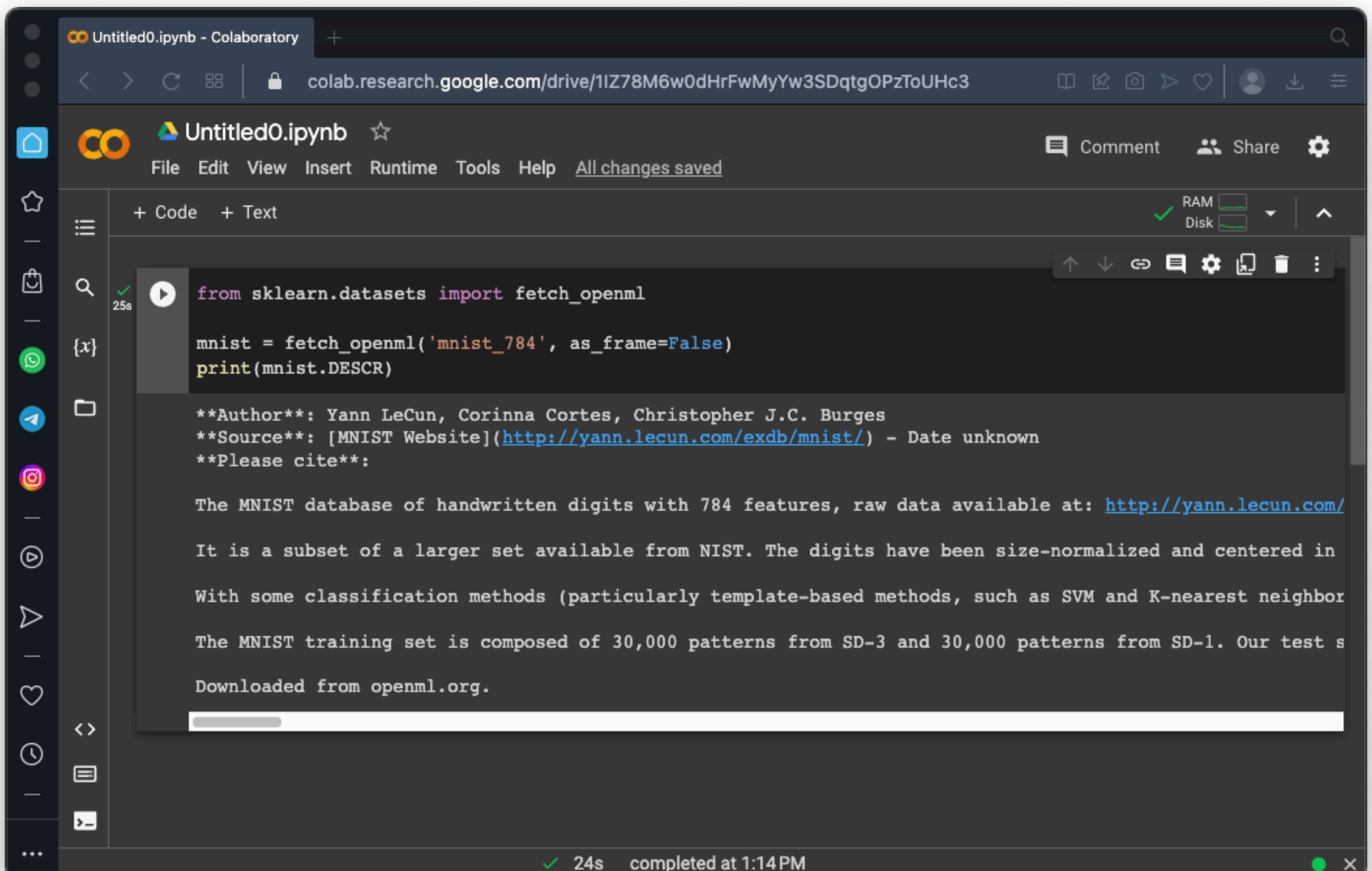
Topics

- Project ML 105
  - MNIST
  - Training a Binary Classifier
  - Performance Measures
  - Multiclass Classification
  - Error Analysis
  - Multilabel Classification
  - Multioutput Classification

MNIST

The "Hello World" of Machine Learning

- 70,000 images of handwritten digits



X and y

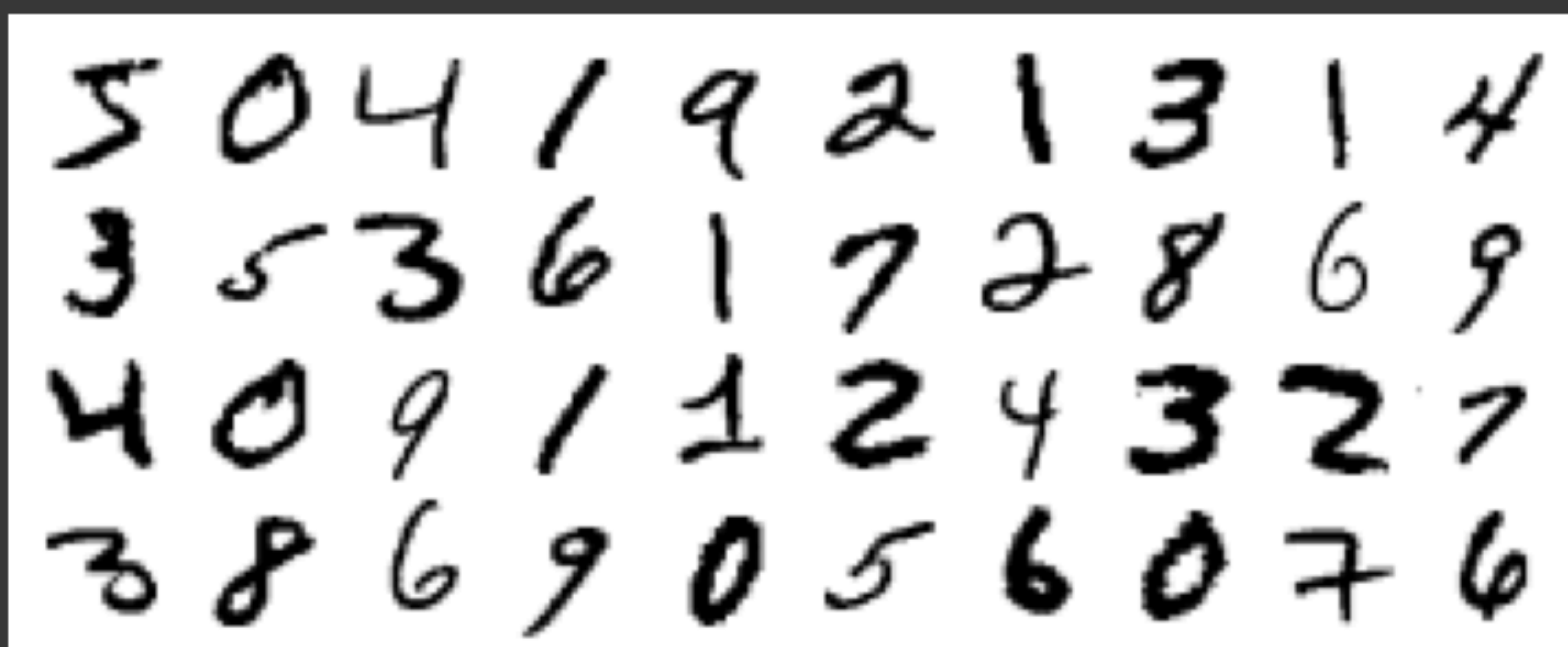
- X has the pixel values
  - 784 pixels
  - Each is a number from 0 to 255
- Y has the label
  - A digit from 0 to 9

```
X, y = mnist.data, mnist.target
print(X)
print(X.shape)
print(y)
print(y.shape)
```

```
[[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.]]
(70000, 784)
['5' '0' '4' ... '4' '5' '6']
(70000,)
```

Viewing the Images

```python
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image = image_data.reshape(28, 28)
    plt.imshow(image, cmap="binary")
    plt.axis("off")

plt.figure(figsize=(9, 9))
for idx, image_data in enumerate(X[:100]):
    plt.subplot(10, 10, idx + 1)
    plot_digit(image_data)
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```



Training and Testing Sets

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
print("Training set:", len(X_train))
print("Test set:", len(X_test))
```

```
Training set: 60000
Test set: 10000
```

Training a Binary Classifier

Preparing Binary Data

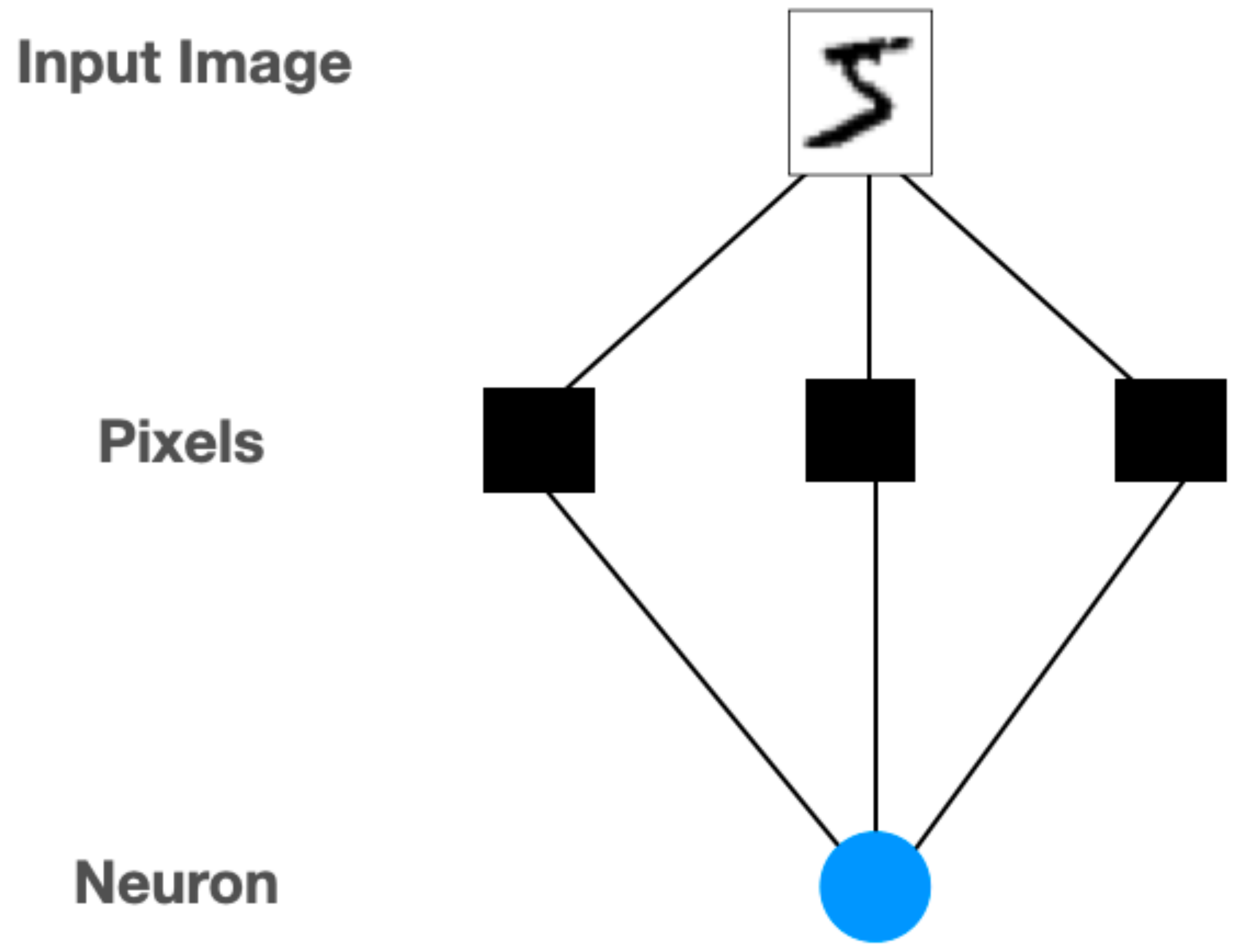
- True = "5"; False = "Not 5"
- Both sets are mostly "False"

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

print("Training set:", y_train_5)
print("True:", numpy.count_nonzero(y_train_5 == True))
print("False:", numpy.count_nonzero(y_train_5 == False))

print()
print("Test set:", y_test_5)
print("True:", numpy.count_nonzero(y_test_5 == True))
print("False:", numpy.count_nonzero(y_test_5 == False))
```

```
Training set: [ True False False ...  True False False]
True: 5421
False: 54579

Test set: [False False False ... False  True False]
True: 892
False: 9108
```

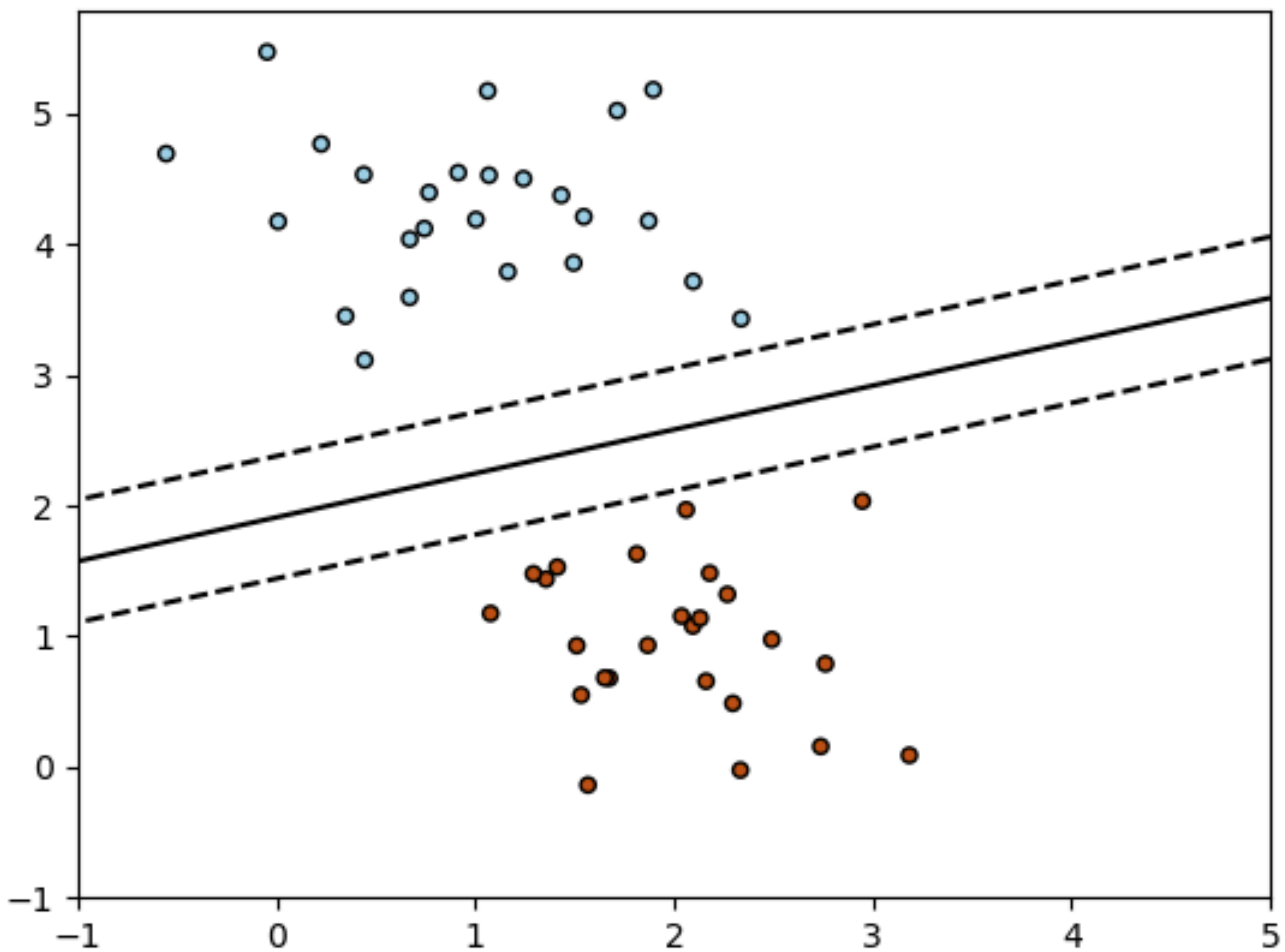
Linear Model

- The image provides 784 ***pixels***
    - The brightness is a number from 0 to 255
- The neuron calculates an output by multiplying each pixel by a ***weight*** and adding them together
- There are 784 parameters



Stochastic Gradient Descent

- A simple, efficient way to fit linear classifiers
- See https://scikit-learn.org/stable/modules/sgd.html



Training a Binary Classifier

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42, verbose=2)
sgd_clf.fit(X_train, y_train_5)

print("Prediction for image 0 (",y[0],"):", sgd_clf.predict([X[0]]))
print("Prediction for image 1 (",y[1],"):", sgd_clf.predict([X[1]]))
print("Prediction for image 2 (",y[2],"):", sgd_clf.predict([X[2]]))
```

- Predicts first three training images correctly

```
-- Epoch 237
Norm: 152.72, NNZs: 674, Bias: 79.301728, T: 14220000, Avg. loss: 78.698093
Total training time: 27.14 seconds.
-- Epoch 238
Norm: 152.05, NNZs: 674, Bias: 79.342428, T: 14280000, Avg. loss: 77.607774
Total training time: 27.24 seconds.
-- Epoch 239
Norm: 151.52, NNZs: 674, Bias: 79.387858, T: 14340000, Avg. loss: 77.870948
Total training time: 27.34 seconds.
Convergence after 239 epochs took 27.34 seconds
Prediction for image 0 ( 5 ): [ True]
Prediction for image 1 ( 0 ): [False]
Prediction for image 2 ( 4 ): [False]
```

Performance Measures

Measuring Accuracy Using Cross-Validation

```
from sklearn.model_selection import cross_val_score

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

- "cv = 3" means three folds
- Train model three times on 2/3 of the training data
- Evaluate it on the other 1/3 of the data each time
- Accuracy > 95%

```
-- Epoch 131
Norm: 340.06, NNZs: 657, Bias: 91.122958, T: 5240000, Avg. loss: 202.672238
Total training time: 8.84 seconds.
Convergence after 131 epochs took 8.84 seconds
array([0.95035, 0.96035, 0.9604 ])
```

How Good is 95%?

- 90% of the data is False (Not "5")
- So simply classifying everything as False would be 90% correct
- Accuracy alone is not a preferred performance measure

Confusion Matrices

```
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

cm = confusion_matrix(y_train_5, y_train_pred)
cm
```

- FIrst row is the False images (Not "5")
  - 687 incorrectly classified as True ("5")
- Second row is the True images ("5")
  - 1891 incorrectly classified as False (Not "5")
-

```
Convergence after 131 epochs took 8.21 seconds
array([[53892,   687],
       [ 1891,  3530]])
```

Precision

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

- TP = True Positives
- FP = False Positives
- Measures accuracy of positive predictions

```
array([[53892,   687],
       [ 1891,  3530]])
```

From the matrix in the image above:

- 687 non-5's were identified as 5's -- these are *False Positives*
- 3530 5's were identified as 5's -- these are *True Positives*

So the Precision is

Precision = 3530 / (3530 + 687) = 0.837 = 83.7%

Precision

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

- **BUT** a classifier can get perfect precision by classifying everything as False except one which it can accurately classify as True
  - No False Positives, but many False Negatives

Recall

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

- Also called ***sensitivity*** or the ***true positive rate*** **(TPR)**
- TP = True Positives
- FN = False Negatives
- Measures the ratio of positive instances that are correctly detected

```
array([[53892,   687],
       [ 1891,  3530]])
```

- 1891 5's were identified as non-5's -- these are *False Negatives*
- 3530 5's were identified as 5's -- these are *True Positives*

So the Recall is

Recall = 3530 / (3530 + 1891) = 0.651 = 65.1%

Precision

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

- TP = True Positives
- FP = False Positives
- Measures accuracy of positive predictions
- **BUT** a classifier can get perfect precision by classifying everything as False except one which it can accurately classify as True
  - No False Positives, but many False Negatives

Precision

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

- TP = True Positives
- FP = False Positives
- Measures accuracy of positive predictions
- **BUT** a classifier can get perfect precision by classifying everything as False except one which it can accurately classify as True
    - No False Positives, but many False Negatives

A Simple Example

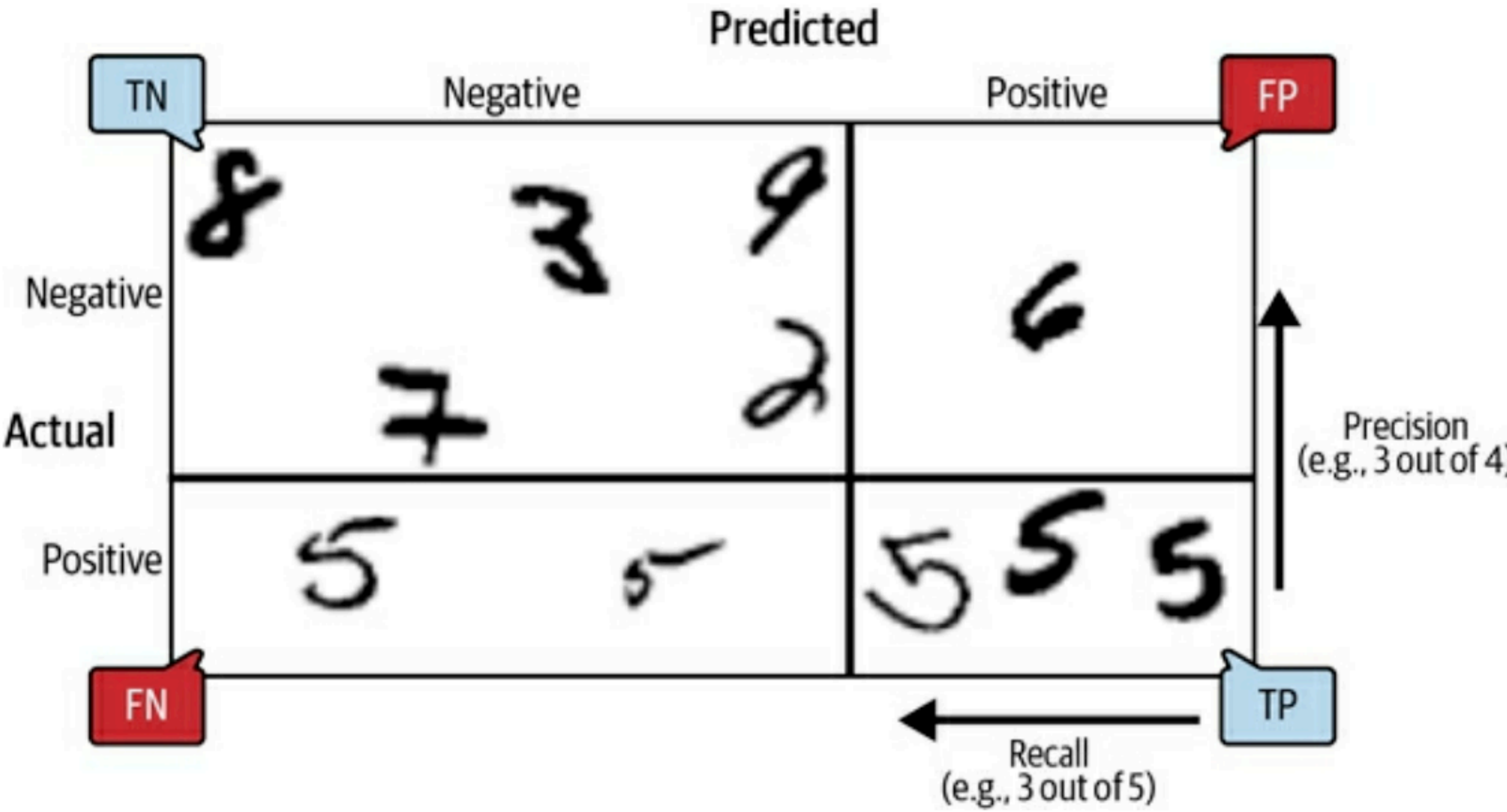


Figure 3-3. An illustrated confusion matrix showing examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)

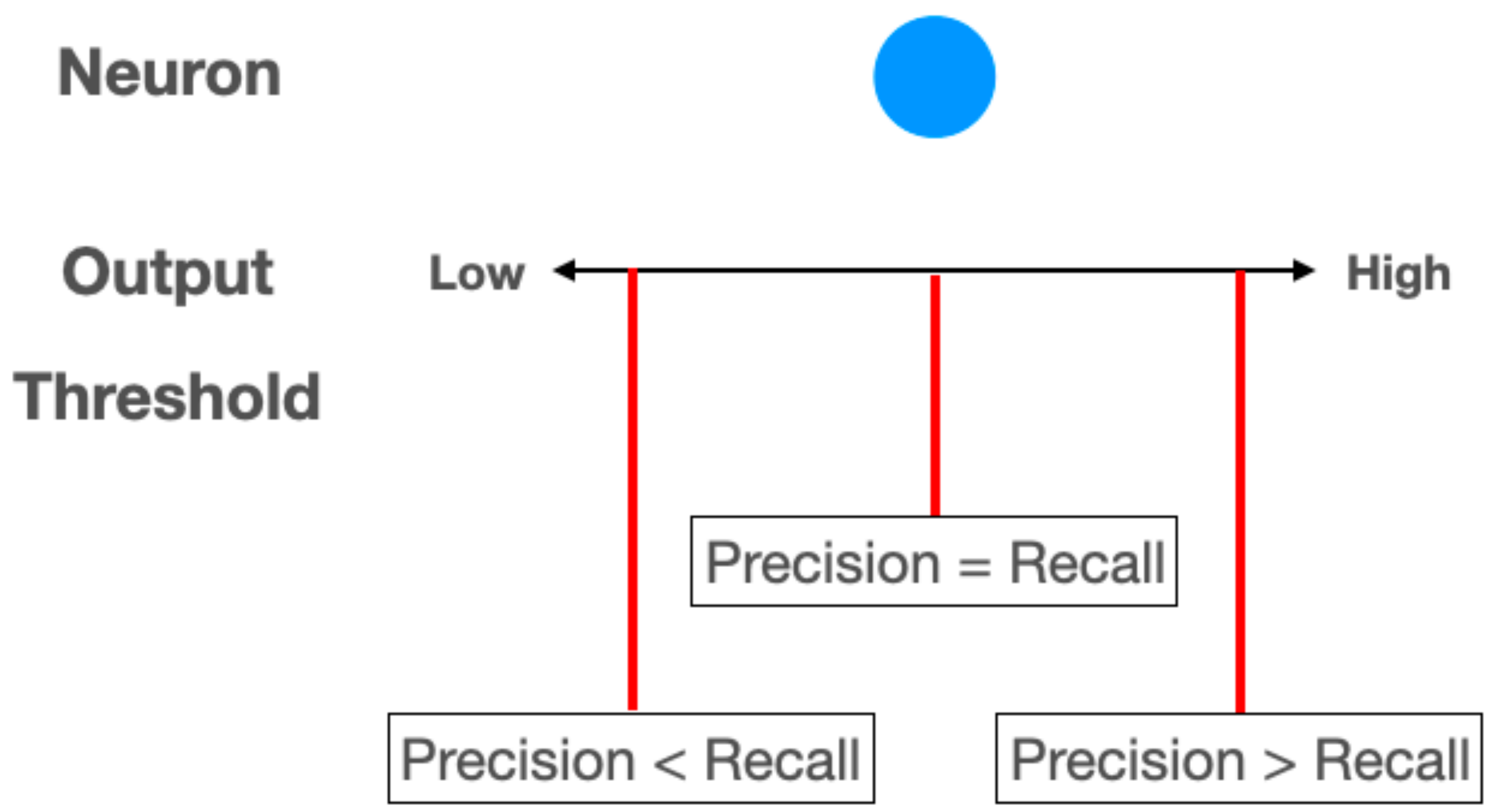
F1 Score

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

- ***F_1*** is high only if both precision and recall are high
- It favors classifiers with similar precision and recall

The Precision/Recall Trade-off

- The neuron puts out a signal
    - Signal above ***threshold*** = "True"



The Precision/Recall Trade-off

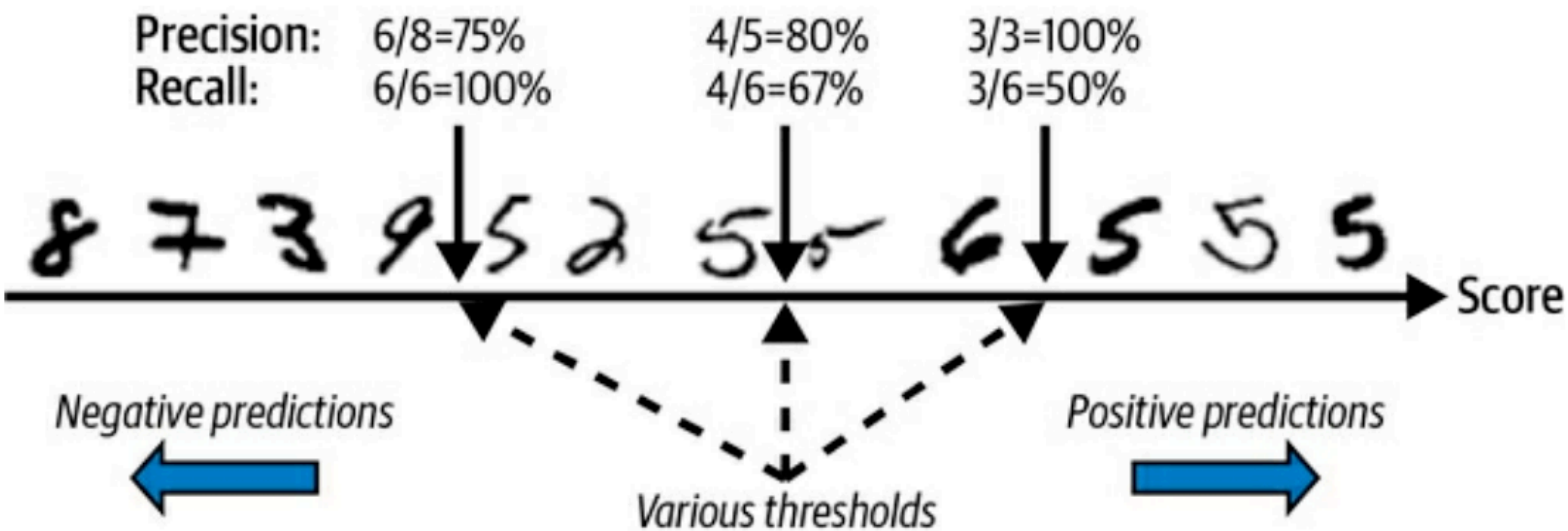


Figure 3-4. The precision/recall trade-off: images are ranked by their classifier score, and those above the chosen decision threshold are considered positive; the higher the threshold, the lower the recall, but (in general) the higher the precision

Effect of Threshold

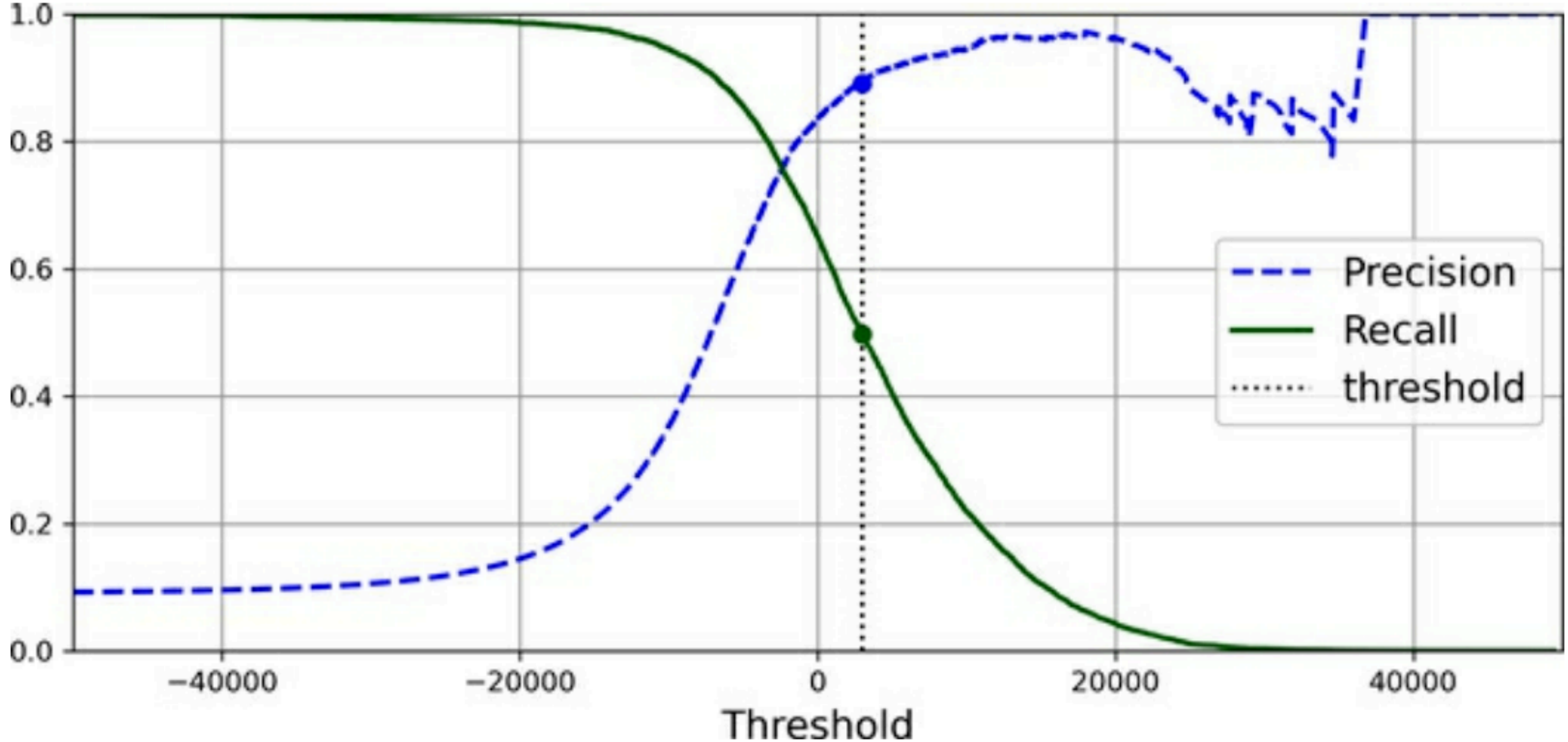
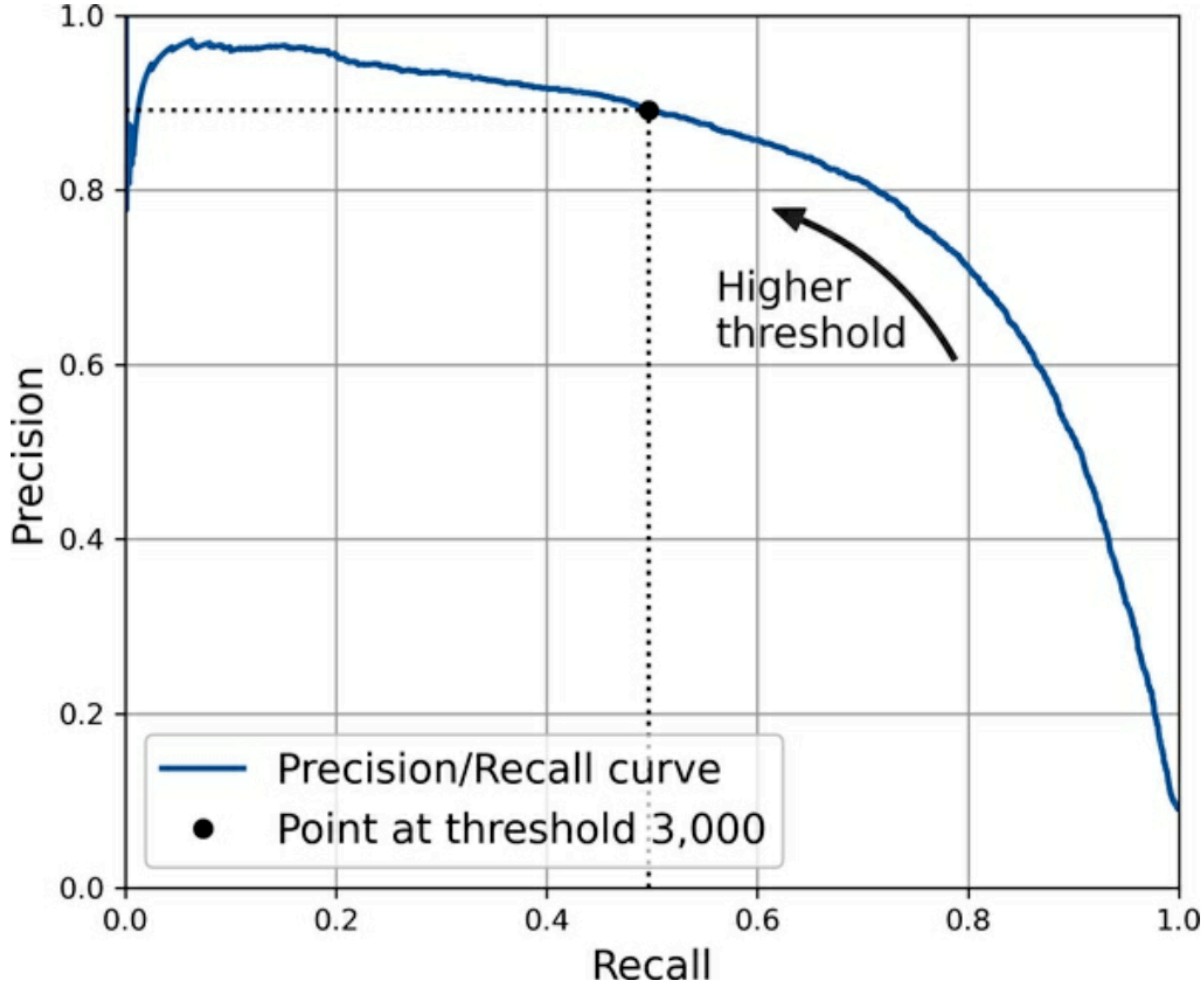


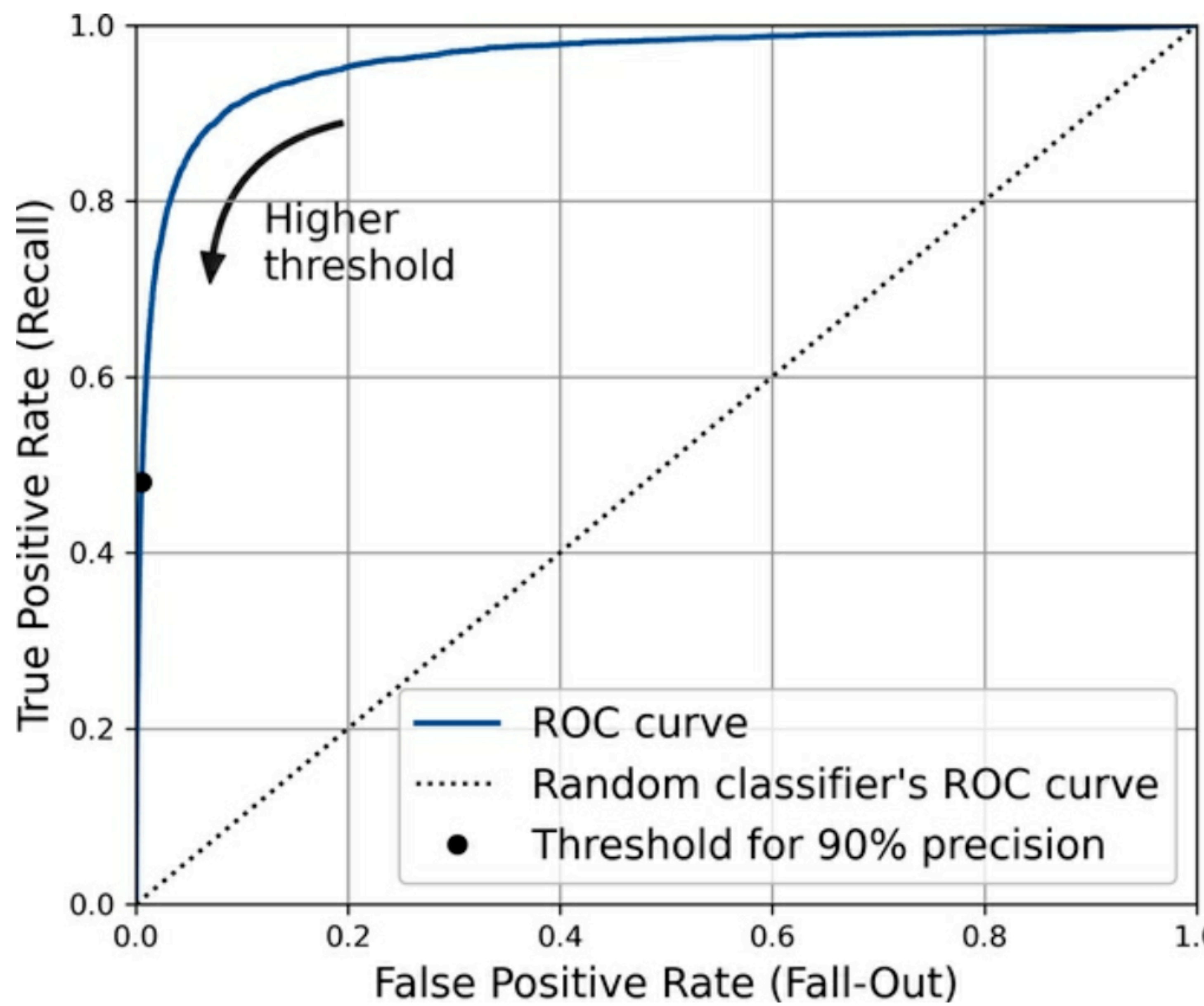
Figure 3-5. Precision and recall versus the decision threshold

Precision/Recall Curve

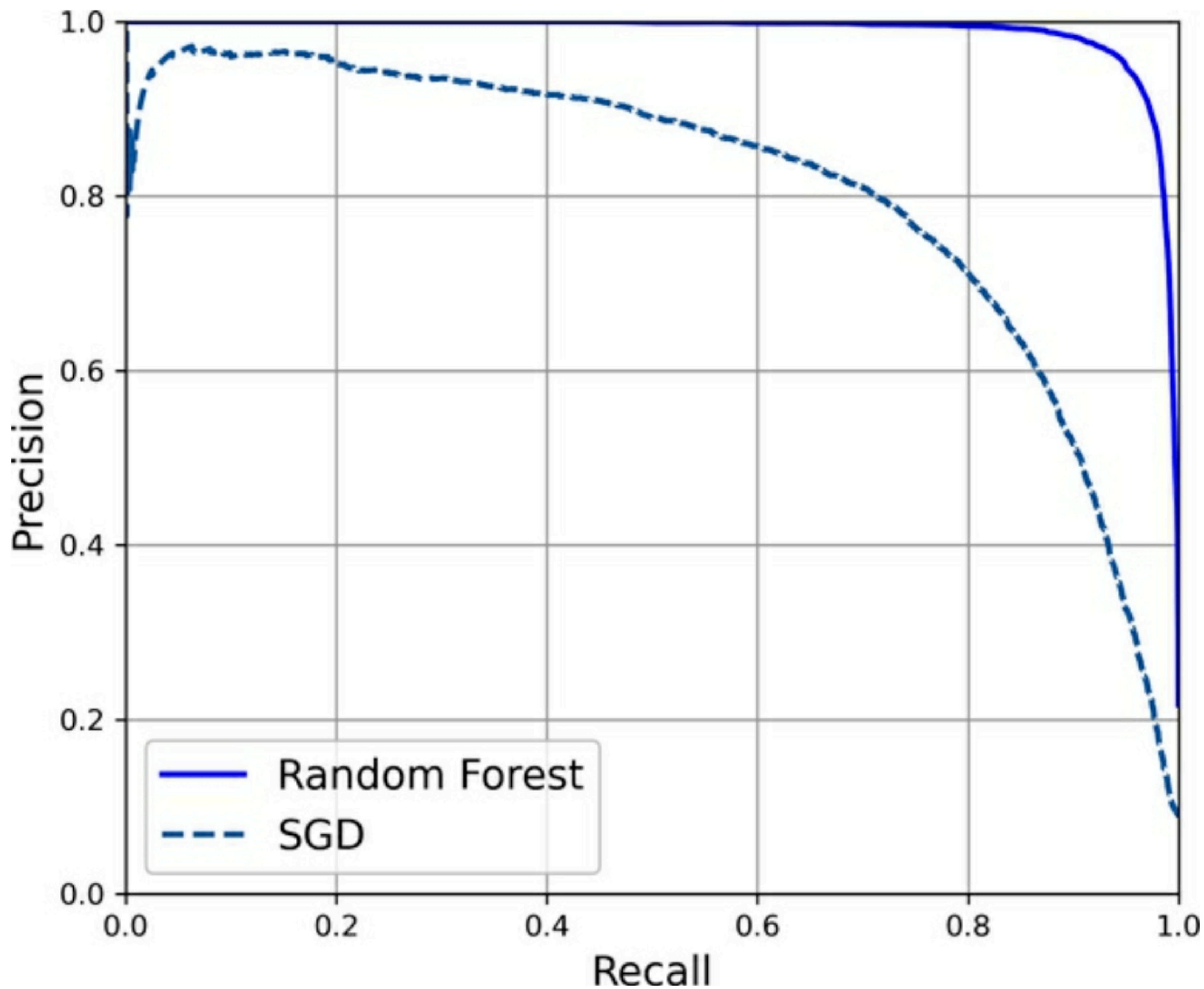


The ROC Curve

- Reciever Operating Characteristic (ROC) Curve
- Area Under the Curve (AUC)
  - 0.5 for a random classifier
  - 1.0 for a perfect classifier



Comparing Models



Random Forest v. Linear Model

- Linear model
  - Precision 83.7%, Recall 65.1%
  - F_1 0.732, AOC 0.960
- Random Forest
  - Precision 99.1%, Recall 86.6%
  - F_1 0.924, AOC 0.998

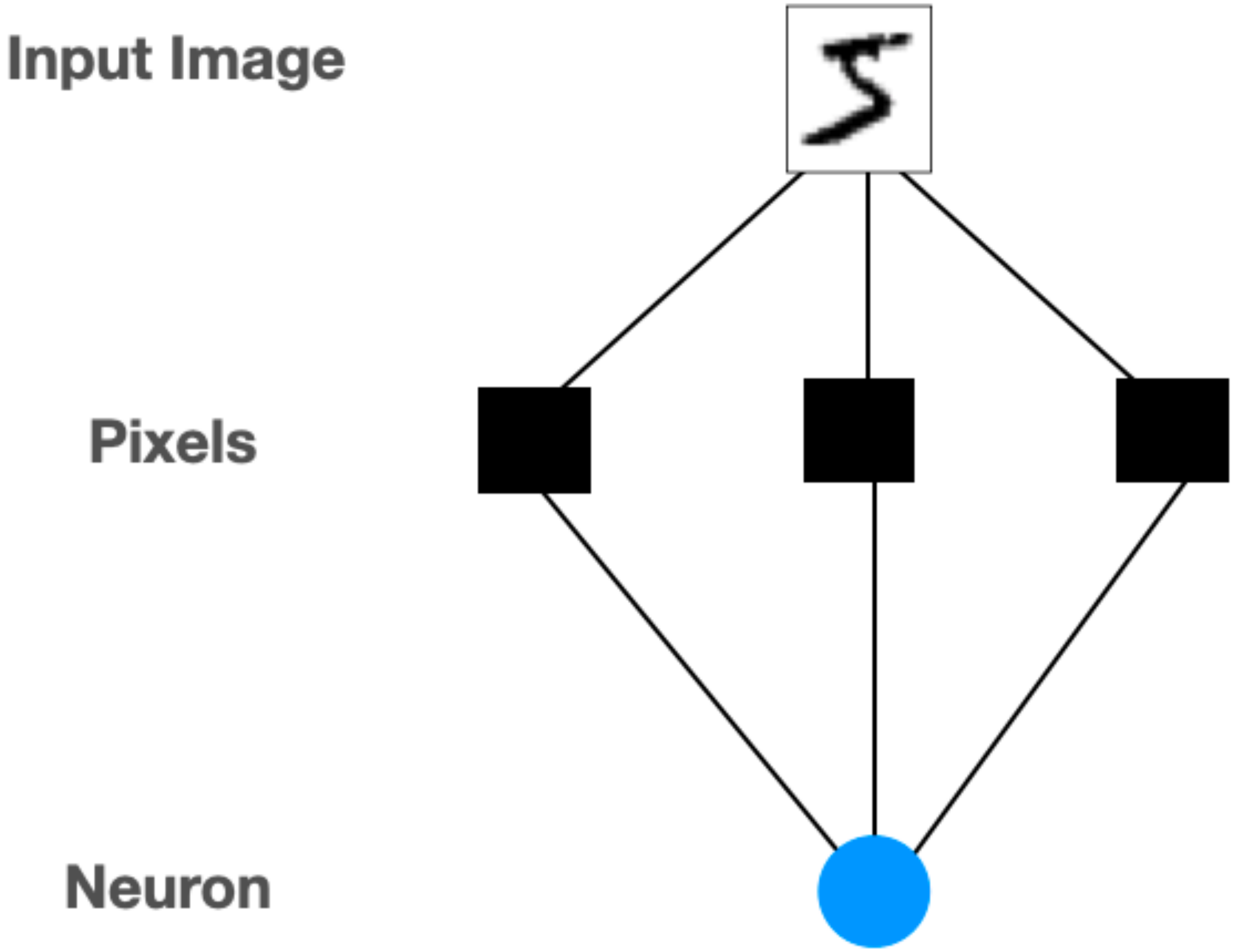
Kahoot!

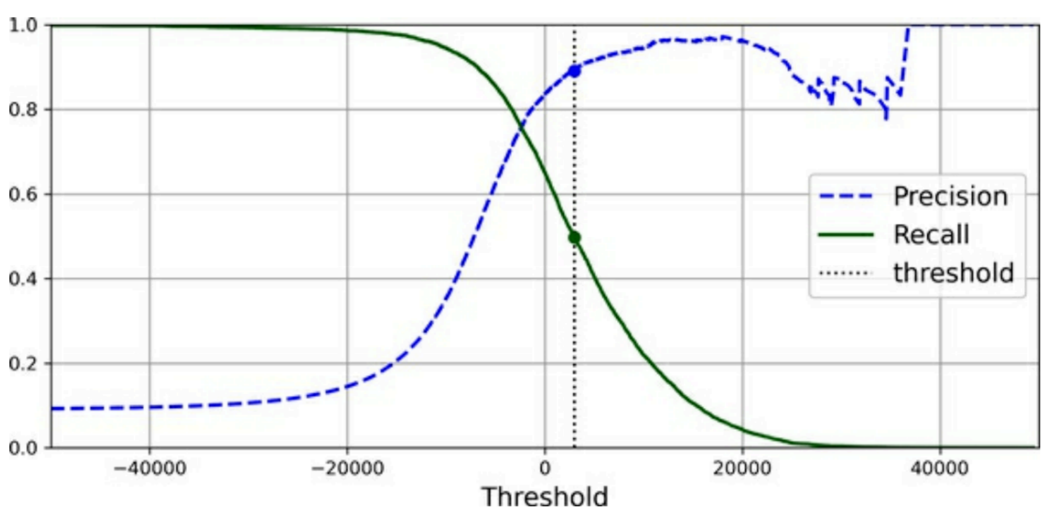
Ch 3a

Multiclass Classification

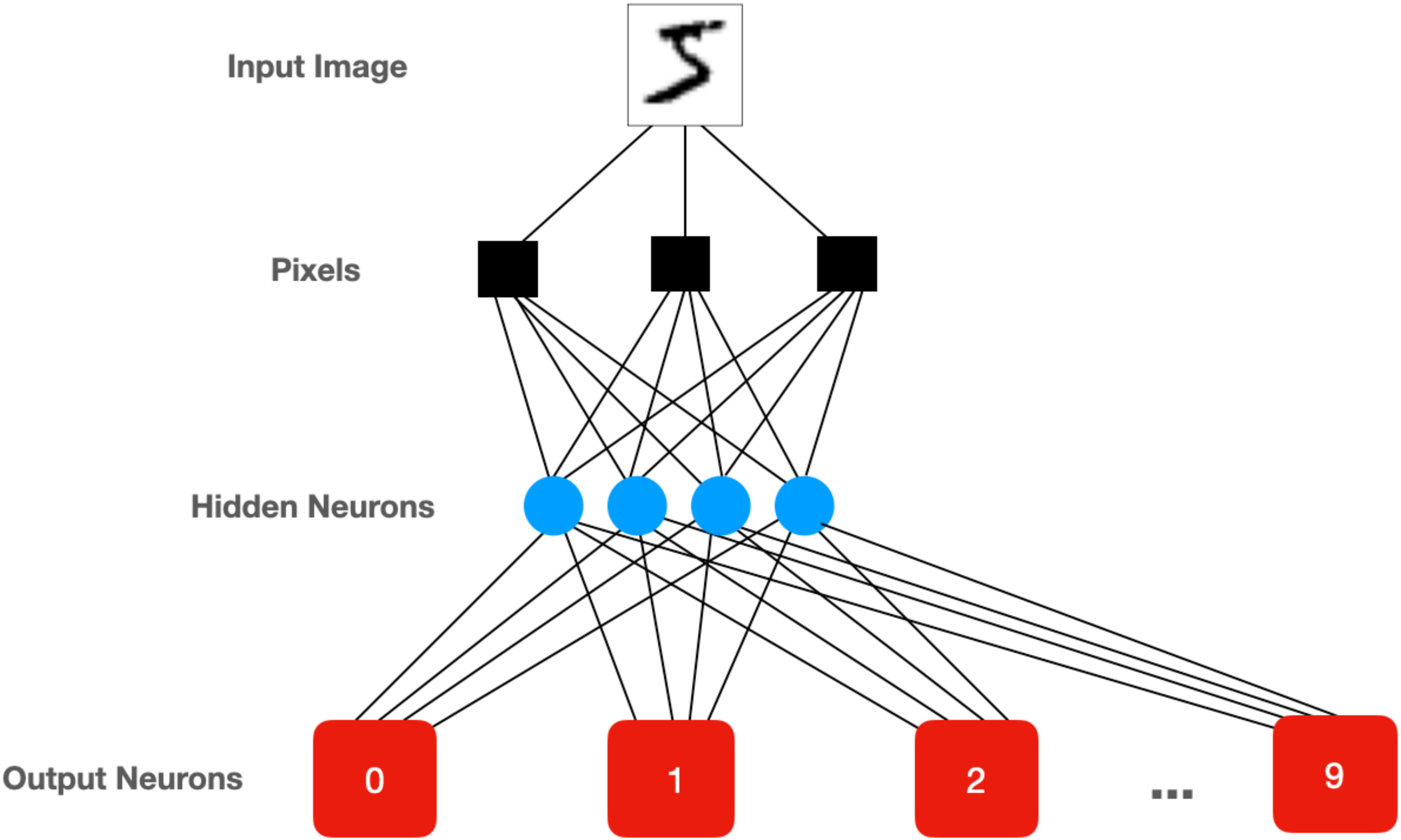
Binary Classifier

- Output is True or False
- "5" or "Not 5"



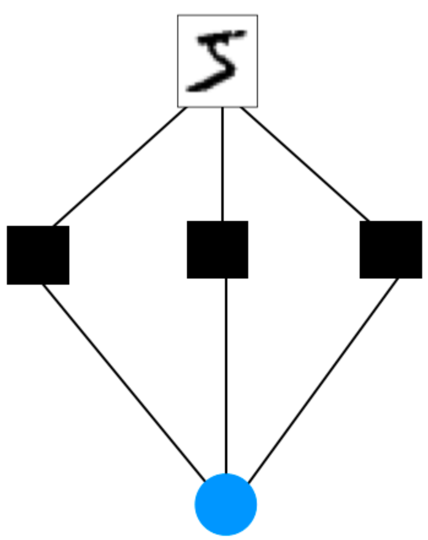


Multiclass Classifier

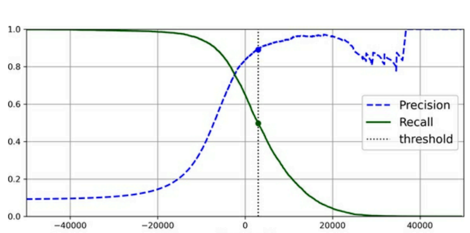


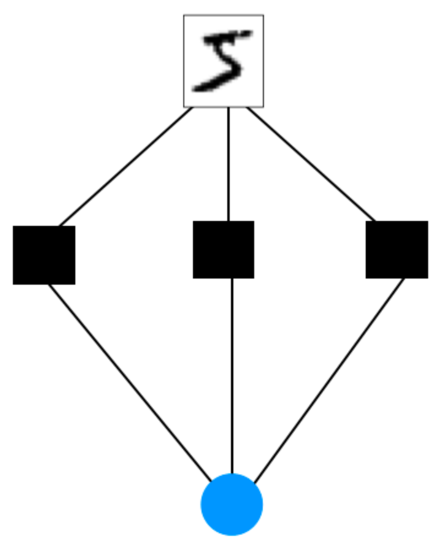
One-versus-the-Rest (OvR)

- Ten Binary Classifiers
- Combine them to form a multiclass classifier
- Each classifier reports a result and a "decision score"
  - The signal from the neuron
- Select the class with the highest decision score

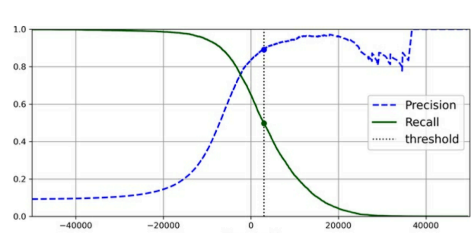


"0" or "Not 0"

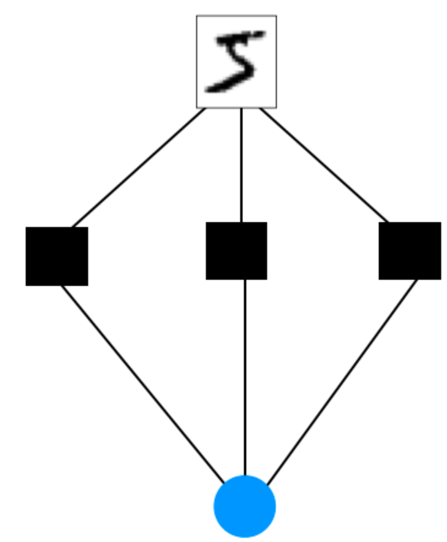




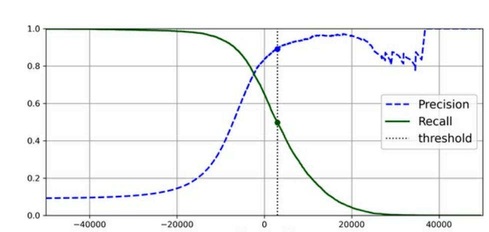
"1" or "Not 1"



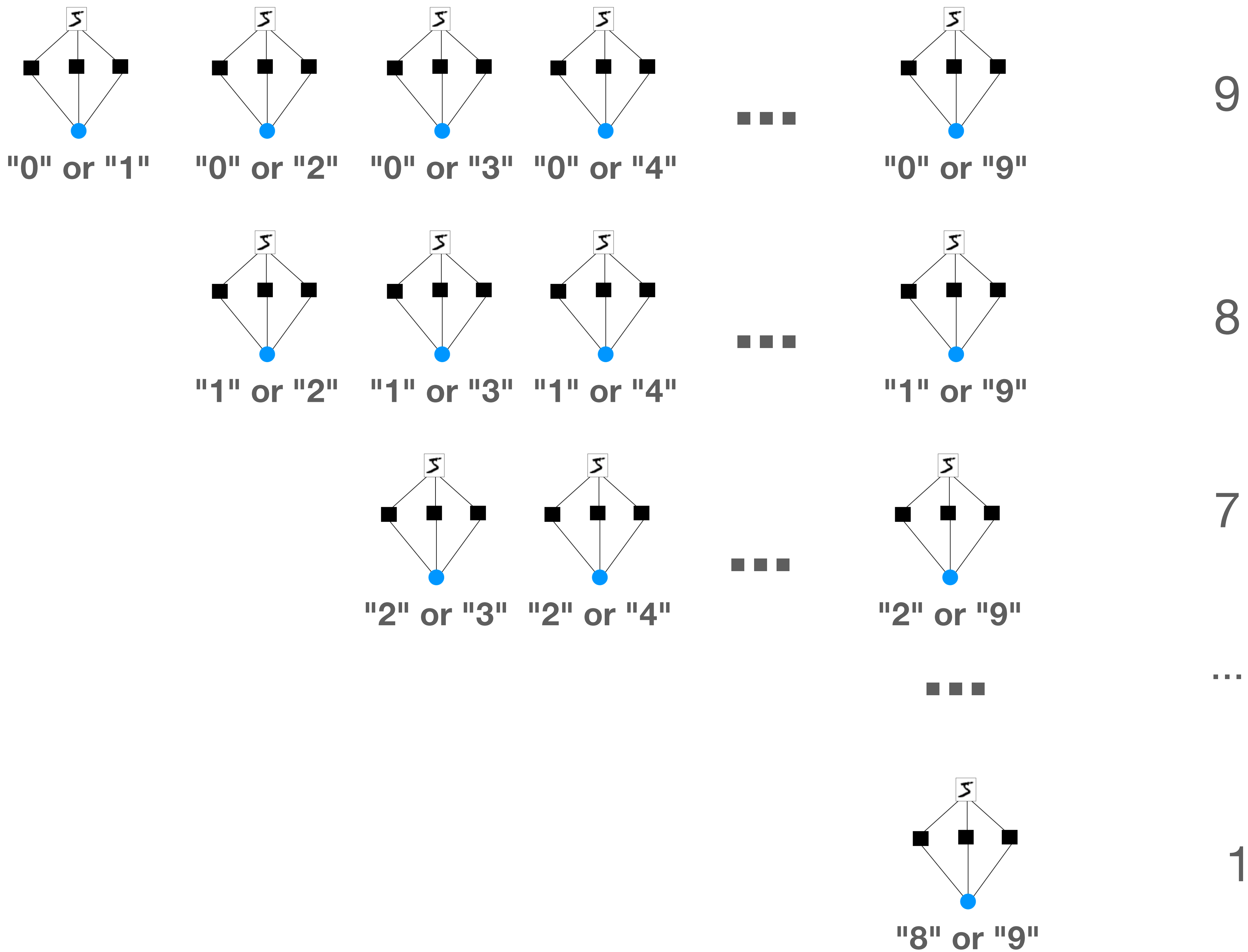
...



"9" or "Not 9"



One-versus-One (OvO)



One-versus-One (OvO)

- Number of binary classifiers:
  - 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45
- For N categories,
  - N x (N - 1) / 2
- Run the image through all 45 binary classifiers
- Select the digit that wins the most duels

- Each classifier only needs to be trained on those two digits of training data
- This is an advantage for algorithms that scale poorly with the size of the training set
  - Like Support Vector Machines
- Most of the time, OvR is preferred

Support Vector Machine: OvO

```
from sklearn.svm import SVC

svm_clf = SVC(random_state=42)
svm_clf.fit(X_train[:2000], y_train[:2000])  # y_train, not y_train_5
```

- Scikit-Learn automatically runs OvO for the SVC model
- The classifier scores are highest for "5"

```
>>> some_digit_scores = svm_clf.decision_function([some_digit])
>>> some_digit_scores.round(2)
array([[ 3.79,  0.73,  6.06,  8.3 , -0.29,  9.3 ,  1.75,  2.77,  7.21,
         4.82]])
```

Support Vector Machine: OvR

```
from sklearn.multiclass import OneVsRestClassifier

ovr_clf = OneVsRestClassifier(SVC(random_state=42))
ovr_clf.fit(X_train[:2000], y_train[:2000])
```

- OneVsRestClassifier forces it to use OvR

Support Vector Machine: OvR

```
>>> sgd_clf = SGDClassifier(random_state=42)
>>> sgd_clf.fit(X_train, y_train)
>>> sgd_clf.predict([some_digit])
array(['3'], dtype='<U1')
```

- Stochastic Gradient Descent (SGD)
  - Used OvR; 10 binary classifiers
- Incorrectly predicted "3", but "5" was almost as strong

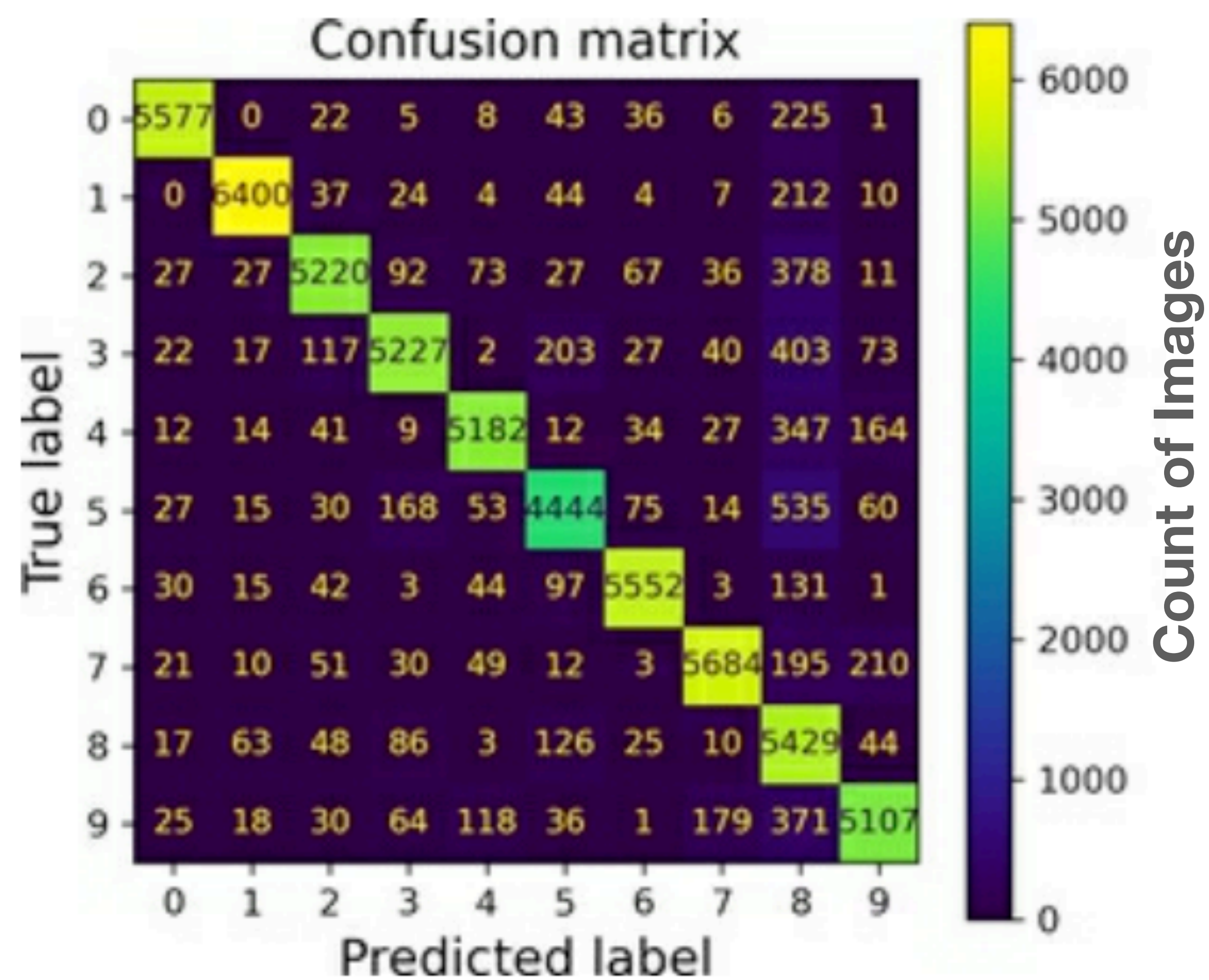
```
>>> sgd_clf.decision_function([some_digit]).round()
array([[-31893., -34420.,  -9531.,   1824., -22320.,  -1386., -26189.,
        -16148.,  -4604., -12051.]])
```

Error Analysis

Confusion Matrix

```python
from sklearn.metrics import ConfusionMatrixDisplay

y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred)
plt.show()
```

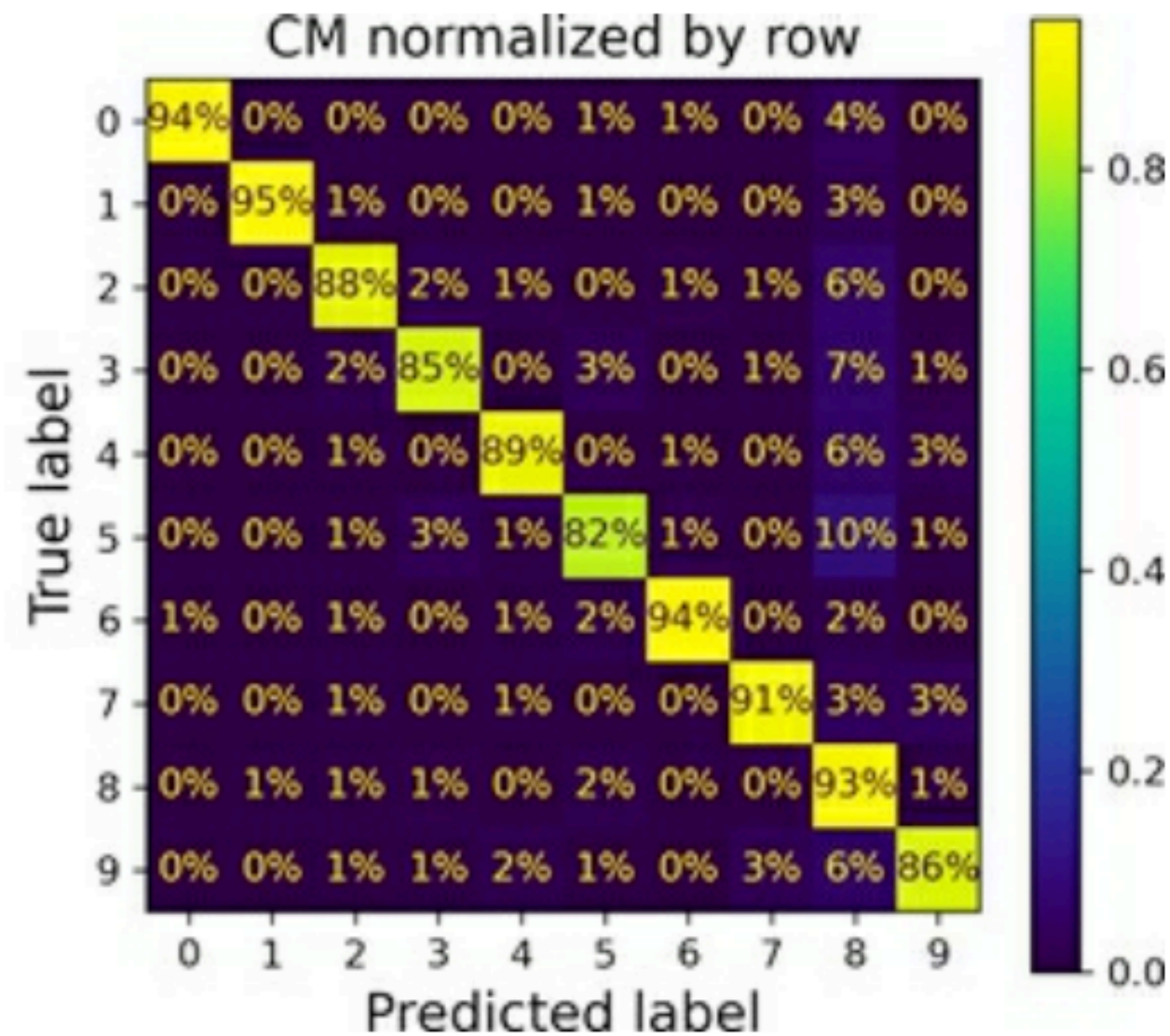
- Cell 5 - 5 has a low number of images
- Could mean the model made more errors there
- But it could also mean there are few 5's in the data set



Normalized Confusion Matrix

```
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                        normalize="true", values_format=".0%")
plt.show()
```

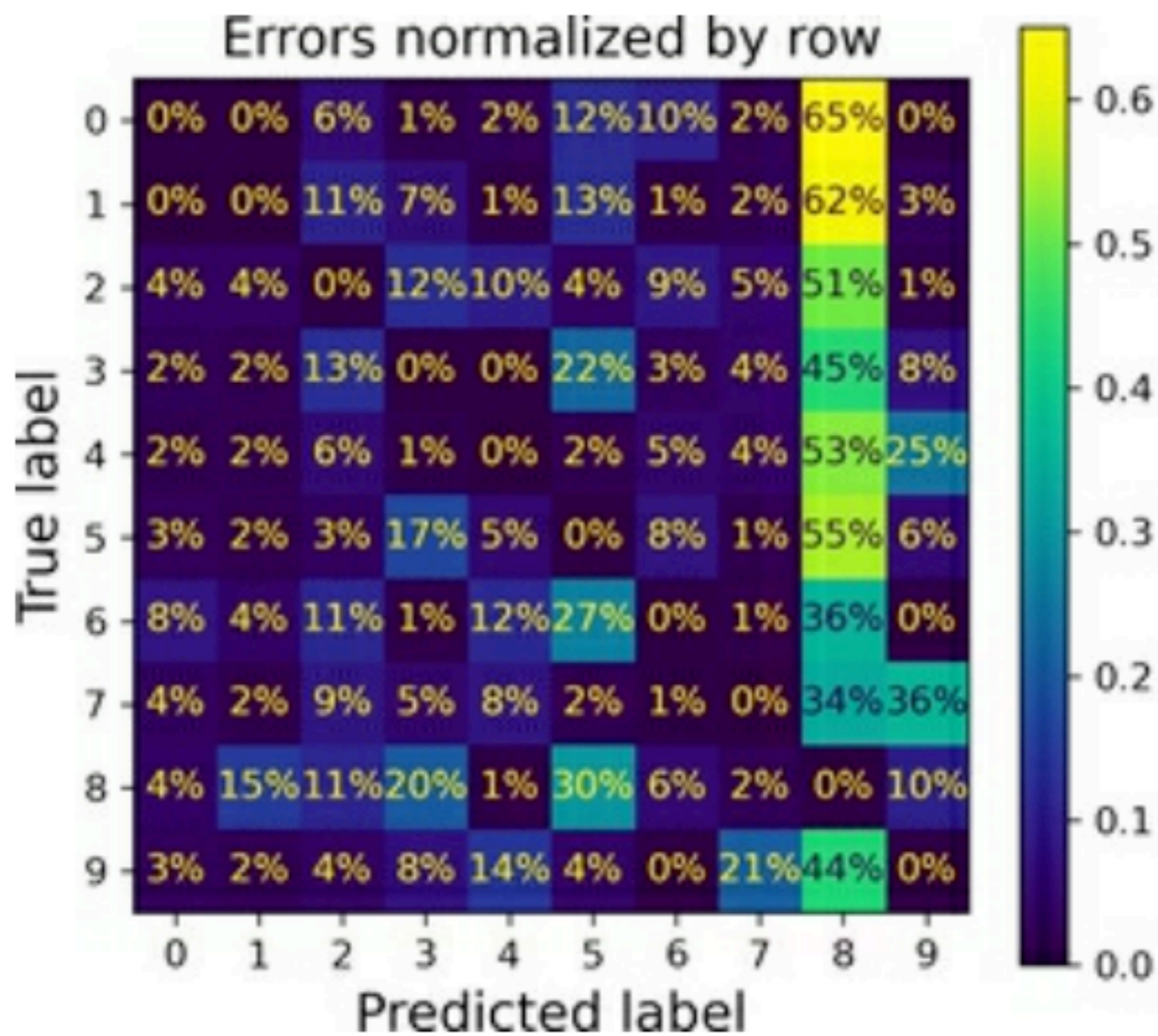
- 5's do have more errors
- Often classified as 8's



Errors Normalized by Row

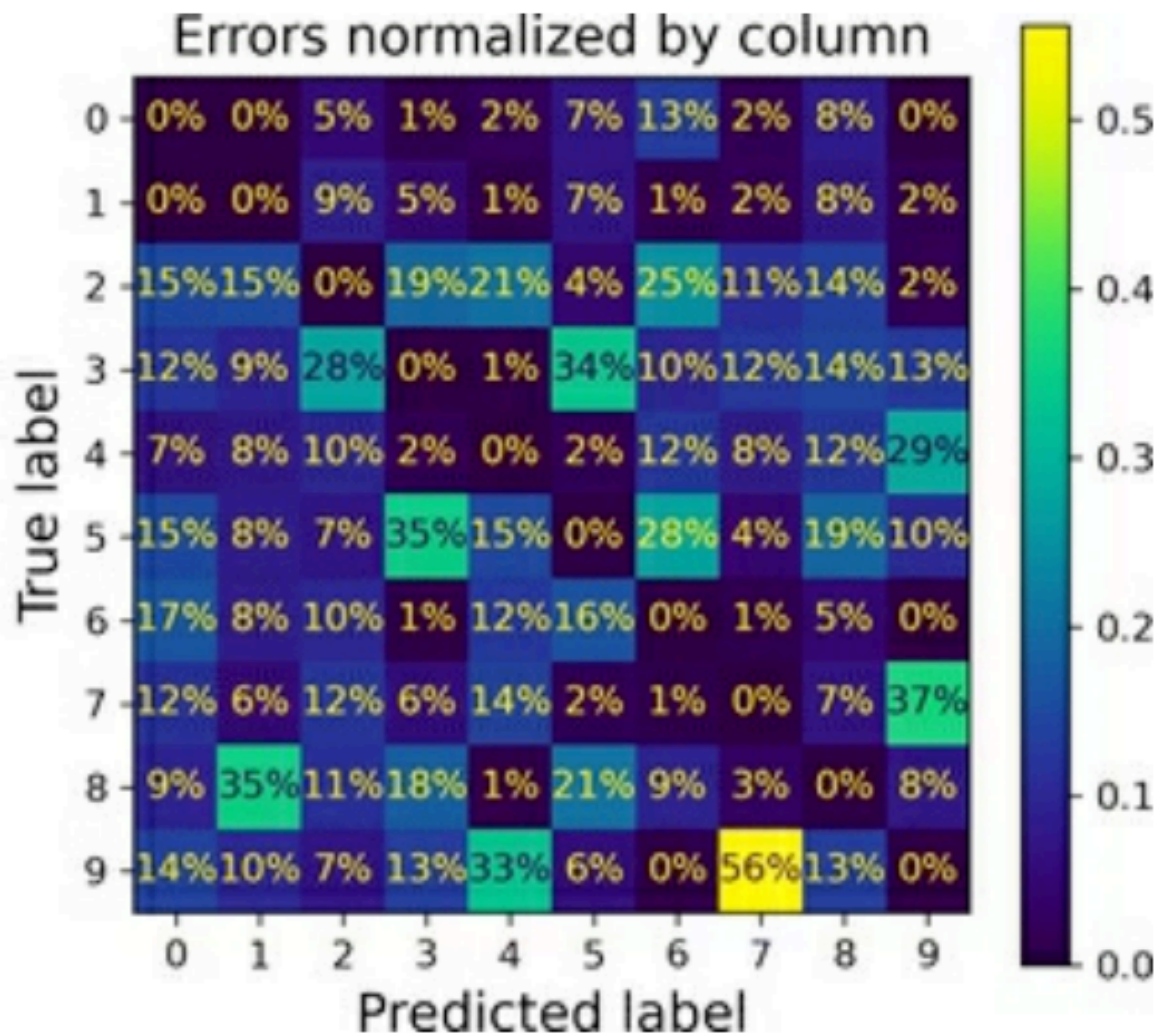
```
sample_weight = (y_train_pred != y_train)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                        sample_weight=sample_weight,
                                        normalize="true", values_format=".0%")
plt.show()
```

- Puts zero weight on the correct predictions
- Many images are incorrectly labelled "8"



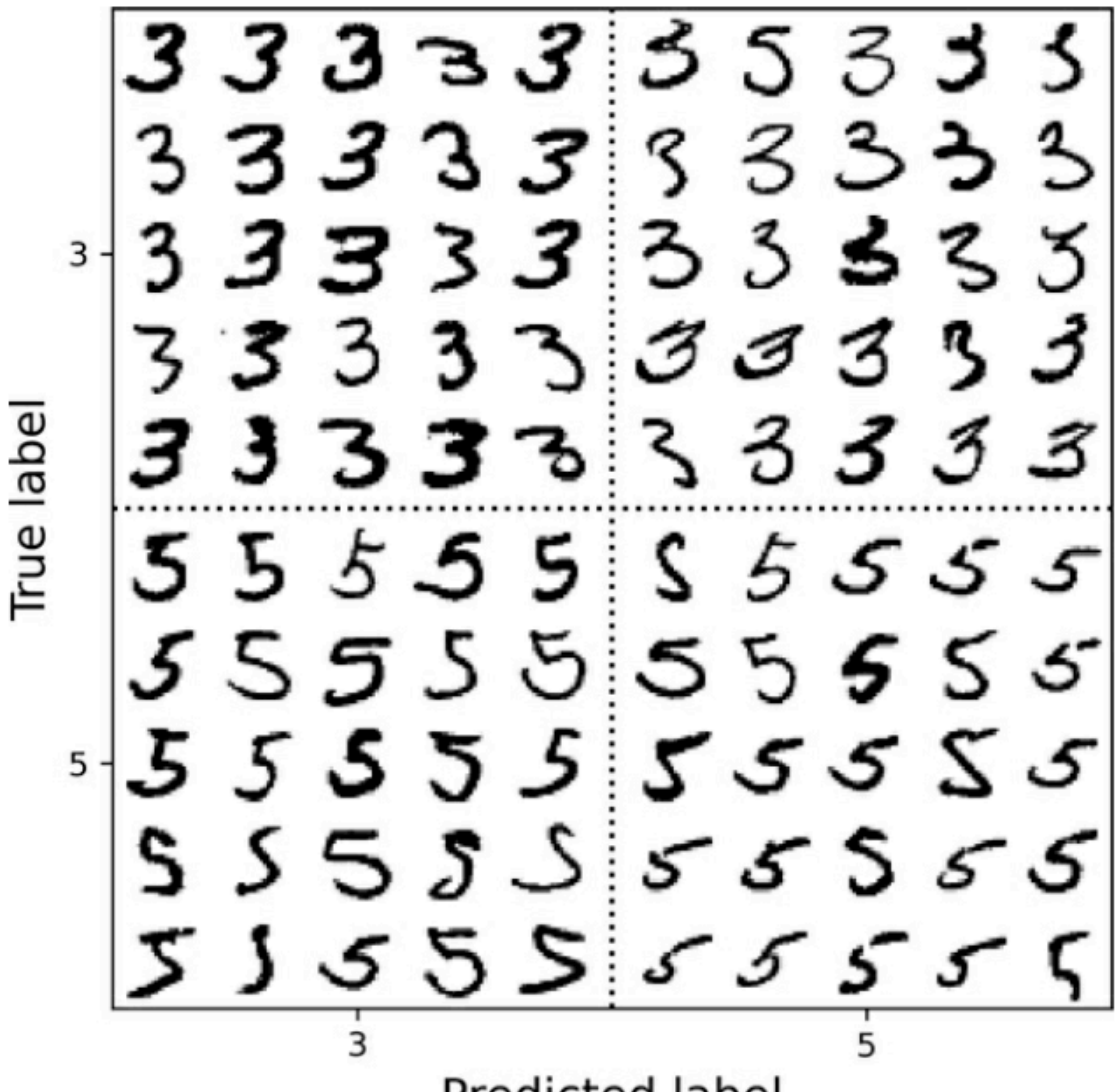
Errors Normalized by Column

- 56% of misclassified 7's are actually 9's



Viewing Example Images

- If the upper vertical line in a 3 is shifted to the left, the model calls it a 5
- ***Data augmentation***
    - Adding shifted and rotated images to the training set will make the model more tolerant of these variations



Multilabel Classification

Multilabel Classification

- **Multiclass Classification**
  - Sort images in to categories
  - One category per image
- **Multilabel Classification**
  - Apply labels to images
  - May apply multiple labels on the same image

Multilabel Classification

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= '7')
y_train_odd = (y_train.astype('int8') % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

- Two labels per image
  - Digit is large (7, 8, or 9)
  - Digit is odd (1, 3, 5, 7, or 9)
- It correctly predicts that 5 is not large, but is odd

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

Evaluating a Multilabel Classifier

```
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
>>> f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.976410265560605
```

- Average F_1 score across all labels
- Appropriate if all labels are equally important

Multilabel Classification with SVC

- SVC does not natively support multilabel classification
- One strategy would be to train one model per label
- But that would miss dependency between the labels
  - Large (7, 8, or 9) images are more likely to also be Odd
- ChainClassifier can train models in sequence
  - Feeding each model labels from previous models

Multioutput Classification

Multioutput Classification

- A generalization of multilabel classification
- Each label can have more than two possible values
- Example: removing noise from images
- Input is many pixels, each from 0 to 255
- Output is many pixels, each from 0 to 255

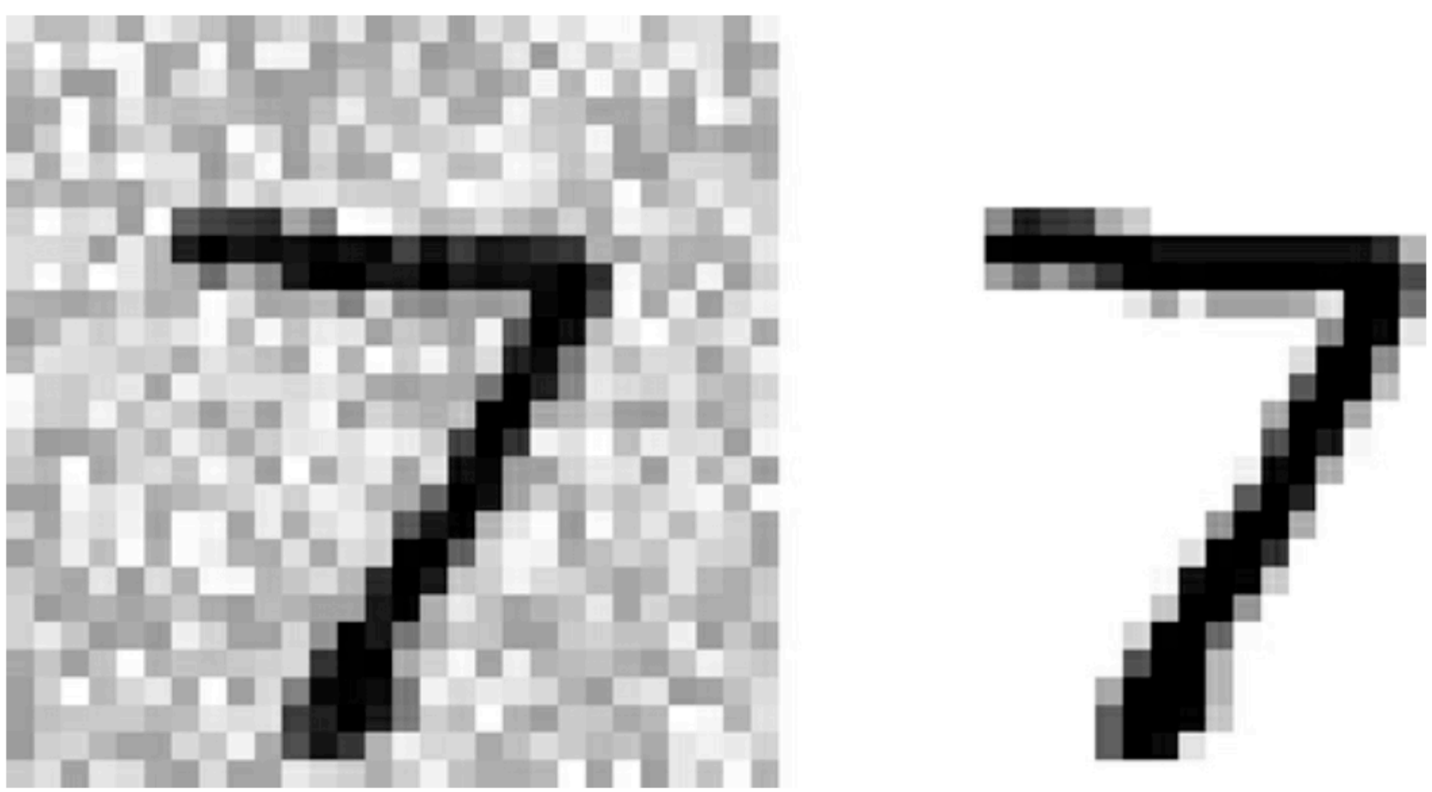


Figure 3-12. A noisy image (left) and the target clean image (right)

Kahoot!

Ch 3b