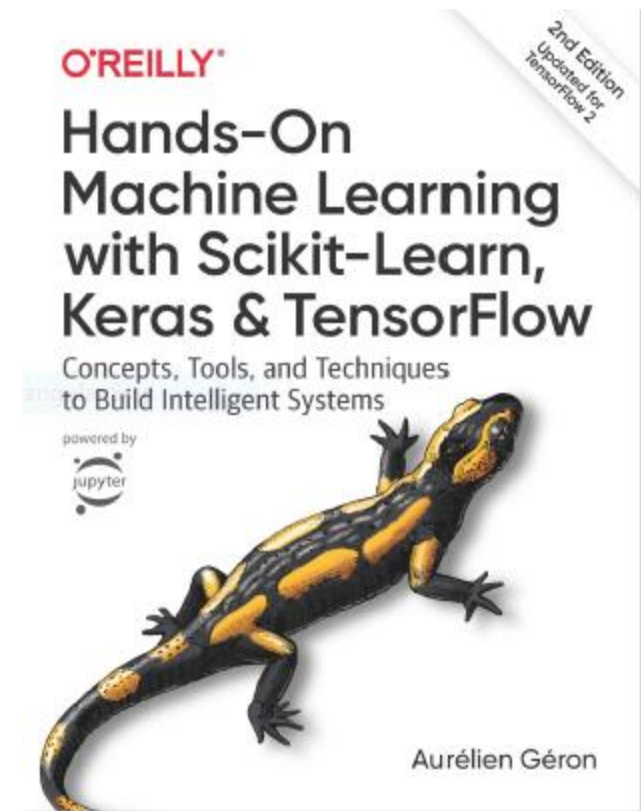


# **Classification**

**Prof. Gheith Abandah**

# Reference

- Chapter 3: **Classification**



- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 2nd Edition, 2019
  - Material: <https://github.com/ageron/handson-ml2>

# Introduction

- YouTube Video: **Machine Learning - Supervised Learning Classification** from Cognitive Class

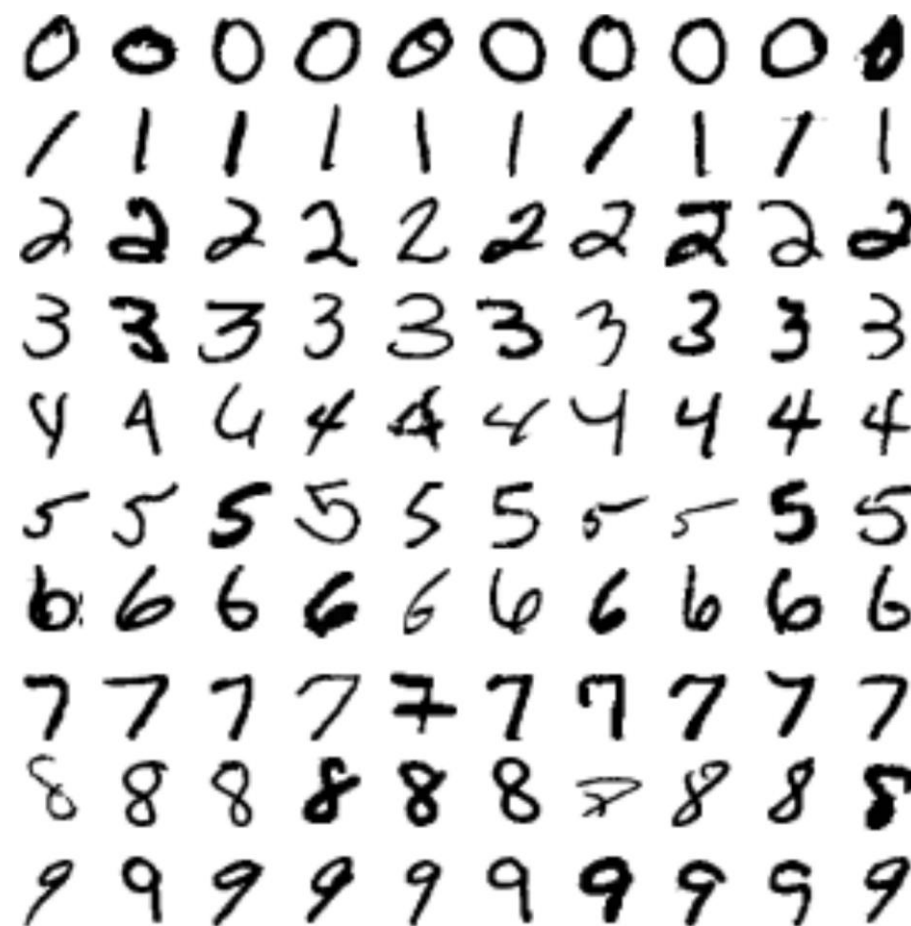
<https://youtu.be/Lf2bCQIktTo>

# Outline

1. MNIST dataset
2. Training a binary classifier
3. Performance measures
4. Multiclass classification
5. Multilabel classification
6. Exercise

# 1. MNIST Dataset

- **MNIST** is a set of 70,000 small images of **handwritten digits**.
- Available from [mldata.org](http://mldata.org)
- **Scikit-Learn** provides **download** functions.



# 1.1. Get the Data

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

## 1.2. Extract Features and Labels

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

There are 70,000 images, and each image has **784** features.

This is because each image is **28×28** pixels, and each feature simply represents one pixel's intensity, from **0 (white)** to **255 (black)**.

# 1.3. Examine One Image

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)
```

```
plt.imshow(some_digit_image, cmap = mpl.cm.binary, interpolation="nearest")
plt.axis("off")
plt.show()
```

```
>>> y[0]
'5'
```





# 1.4. Split the Data

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

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

# Outline

1. MNIST dataset
2. Training a binary classifier
3. Performance measures
4. Multiclass classification
5. Multilabel classification
6. Exercise

## 2. Training a Binary Classifier

- A binary classifier can classify **two classes**.
- For example, classifier for the number 5, capable of distinguishing between two classes, **5** and **not-5**.

```
y_train_5 = (y_train == 5)  
y_test_5 = (y_test == 5)
```

True for all 5s, False for all other digits.

```
from sklearn.linear_model import SGDClassifier
```

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

**Stochastic Gradient Descent** (SGD) classifier

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

# Outline

1. MNIST dataset
2. Training a binary classifier
3. Performance measures
4. Multiclass classification
5. Multilabel classification
6. Exercise

# 3. Performance Measures

- **Accuracy**: Ratio of correct predictions
- **Confusion matrix**
- **Precision** and **recall**
- **F1 Score**
- **Precision/recall tradeoff**

# 3.1. Accuracy

```
y_pred = clone_clf.predict(X_test_fold)
n_correct = sum(y_pred == y_test_fold)
print(n_correct / len(y_pred))
```

Example how to find the accuracy.

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])
```

Using the `cross_val_score()` function to find the accuracy on three folds

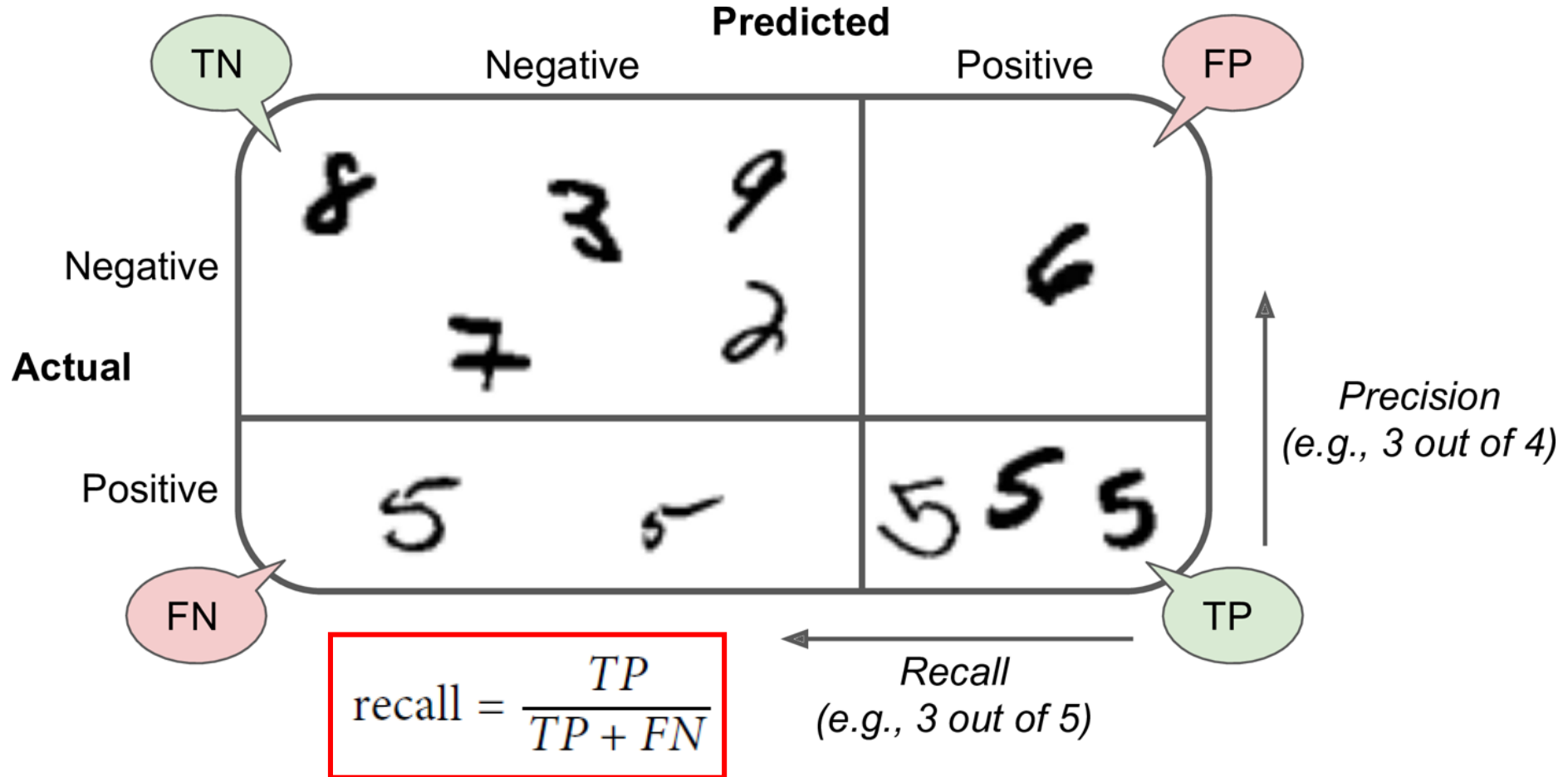
# 3.1. Accuracy

- Use `cross_val_predict()` to predict the targets of the entire training set.

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

# 3.2. Confusion Matrix

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





## 3.2. Confusion Matrix

- Scikit Learn has a function for finding the **confusion matrix**.

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

- The first row is for the non-5s (the **negative** class):
  - 53,057 correctly classified (**true negatives**)
  - 1,522 wrongly classified (**false positives**)
- The second row is for the 5s (the **positive** class):
  - 1,325 wrongly classified (**false negatives**)
  - 4,096 correctly classified (**true positives**)

## 3.3. Precision and Recall

### Precision

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

### Recall

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

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

The precision and recall are smaller than the accuracy.  
Why?

## 3.4. F1 Score

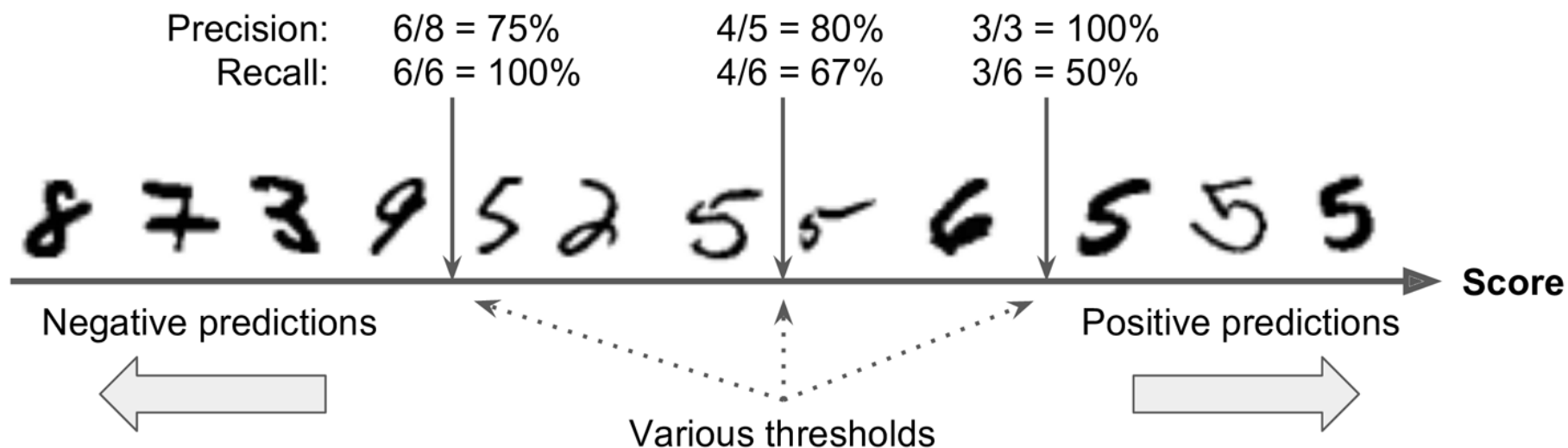
- The F1 Score combines the precision and recall in one metric (**harmonic mean**).

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

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

## 3.5. Precision/Recall Tradeoff

- **Increase** the **decision threshold** to improve the precision when it is **bad** to have FP.
- **Decrease** the decision threshold to improve the recall when it is **important not to miss FN**.



## 3.5. Precision/Recall Tradeoff

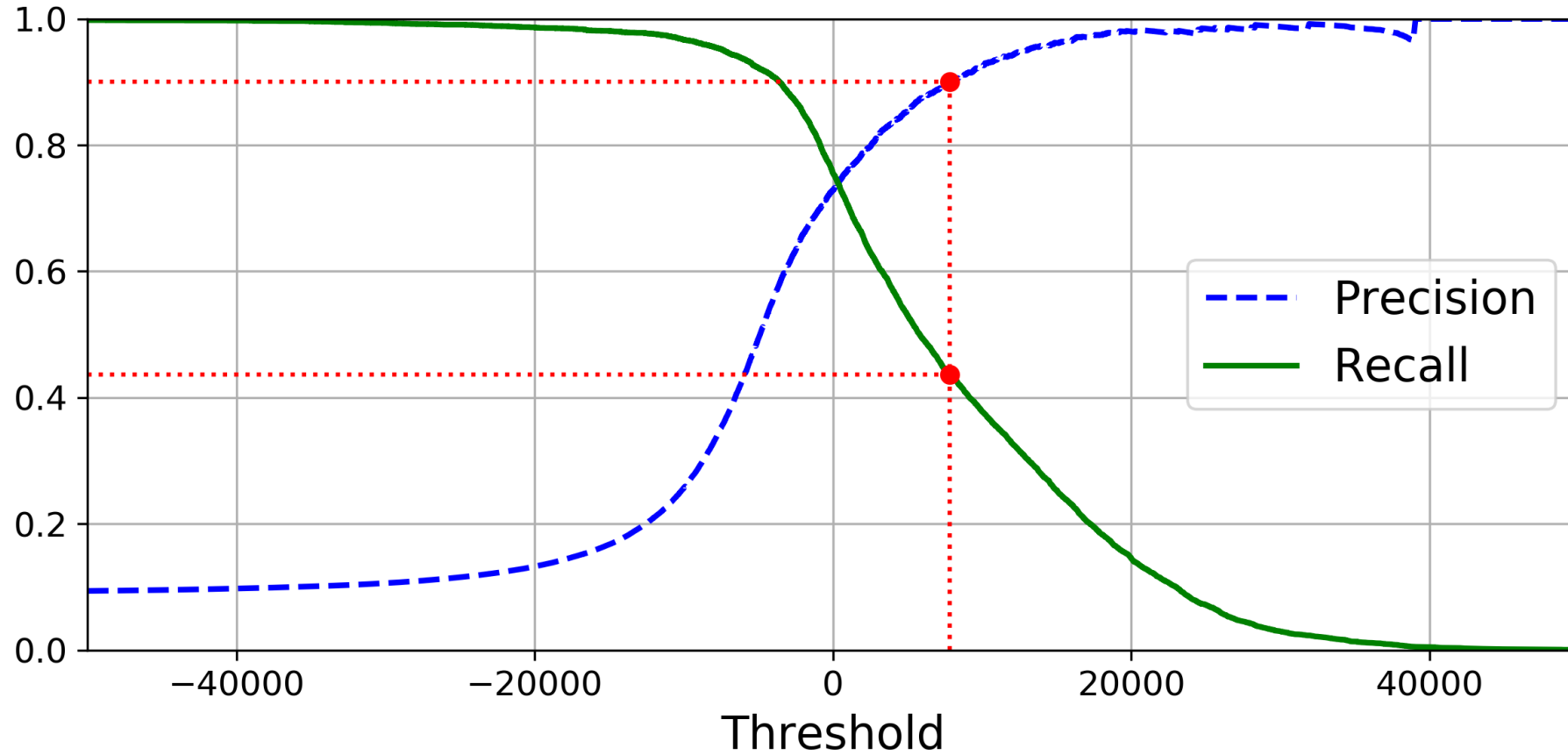
- The function `cross_val_predict()` can return **decision scores** instead of predictions.

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

- These scores can be used to compute precision and recall for all possible thresholds using the **precision\_recall\_curve()** function.

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

# 3.5. Precision/Recall Tradeoff



## 3.5. Precision/Recall Tradeoff

- For **larger precision**, **increase the threshold**, and **decrease it** for **larger recall**.
- **Example**: To get 90% precision.

The first threshold with precision  $\geq 90\%$

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816
y_train_pred_90 = (y_scores >= threshold_90_precision)
>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

True when score  
 $\geq$  new threshold

# Outline

1. MNIST dataset
2. Training a binary classifier
3. Performance measures
4. Multiclass classification
5. Multilabel classification
6. Exercise



# 4. Multiclass Classification

- Multiclass classifiers can distinguish between **more than two classes**.
- Some **algorithms** (such as Random Forest classifiers or Naive Bayes classifiers) are **capable of handling multiple classes** directly.
- **Others** (such as Support Vector Machine classifiers or Linear classifiers) are **strictly binary classifiers**.
- There are **two main strategies** to perform multiclass classification using multiple binary classifiers.

# 4.1. One-versus-All (OvA) Strategy

- For example, classify the digit images into 10 classes (from 0 to 9) to **train 10 binary classifiers**, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).
- Then to classify an image, get the decision score from each classifier for that image and select the class whose classifier outputs the **highest score**.

## 4.2. One-versus-One (OvO) Strategy

- Train a binary classifier **for every pair** of digits.
- If there are  $N$  classes, need  $N \times (N - 1) / 2$  classifiers. For MNIST, **need 45 classifiers**.
- To classify an image, run the image through all 45 classifiers and see which class **wins the most duels**.
- The main advantage of **OvO** is that each classifier only needs to be **trained on a subset** of the training set.
- OvO is preferred for algorithms (such as **Support Vector Machine**) that scale poorly with the size of the training set.

## 4.3. Scikit Learn Support of Multiclass Classification

- **Scikit-Learn** detects when you try to use a binary classification algorithm for a multiclass classification task, and it automatically runs **OvA** (except for **SVM** classifiers for which it uses **OvO**).

```
>>> sgd_clf.fit(X_train, y_train) # y_train, not y_train_5
>>> sgd_clf.predict([some_digit])
array([5], dtype=uint8)
```

```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)
>>> forest_clf.fit(X_train, y_train)
>>> forest_clf.predict([some_digit])
array([5], dtype=uint8)
```

Better  
classifier than  
SGD

## 4.3. Scikit Learn Support of Multiclass Classification

- Note that the multiclass task is harder than the binary task.

- **Binary task**

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])
```

- **Multiclass task**

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.8489802 , 0.87129356, 0.86988048])
```

## 4.4. Error Analysis

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

Many images are misclassified as 8s.

# Outline

1. MNIST dataset
2. Training a binary classifier
3. Performance measures
4. Multiclass classification
5. **Multilabel classification**
6. Exercise

# 5. Multilabel Classification

- Classifiers that output **multiple classes for each instance**.

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

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

Popular algorithm

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



# Summary

1. MNIST dataset
2. Training a binary classifier
3. Performance measures
4. Multiclass classification
5. Multilabel classification
6. Exercise

# Exercise

- Try to build a classifier for the MNIST dataset that achieves over 97% accuracy on the test set. Hint: the **KNeighborsClassifier** works quite well for this task; you just need to find good hyperparameter values (try a grid search on the **weights** and **n\_neighbors** hyperparameters).