Classification

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly). Copyright 2017 Aurélien Géron, 978-1-491-96229-9.

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
- Scikit-Learn provides download functions.

1.1. Get the Data

```
>>> from sklearn.datasets import fetch_mldata
>>> mnist = fetch_mldata('MNIST original')
>>> mnist
{'COL_NAMES': ['label', 'data'],
 'DESCR': 'mldata.org dataset: mnist-original',
 'data': array([[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]], dtype=uint8),
 'target': array([ 0.,  0.,  0., ...,  9.,  9.,  9.])}
```

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).

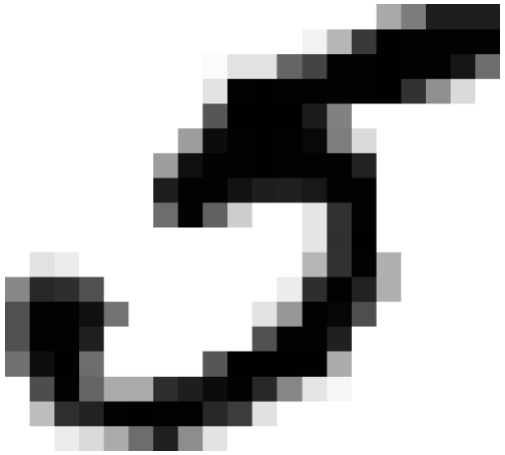
1.3. Examine One Image

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

some_digit = X[36000]
some_digit_image = some_digit.reshape(28, 28)

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

```
>>> y[36000]
5.0
```



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).
- You need to shuffle the training set to guarantee that all cross-validation folds are similar.

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

import numpy as np

shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

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

2. Training a Binary Classifier

- Note that a better classifier for this problem is the *Random Forest Classifier.*

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(random_state=42)
forest_clf.fit(X_train, y_train_5)
```

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
- 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))
```

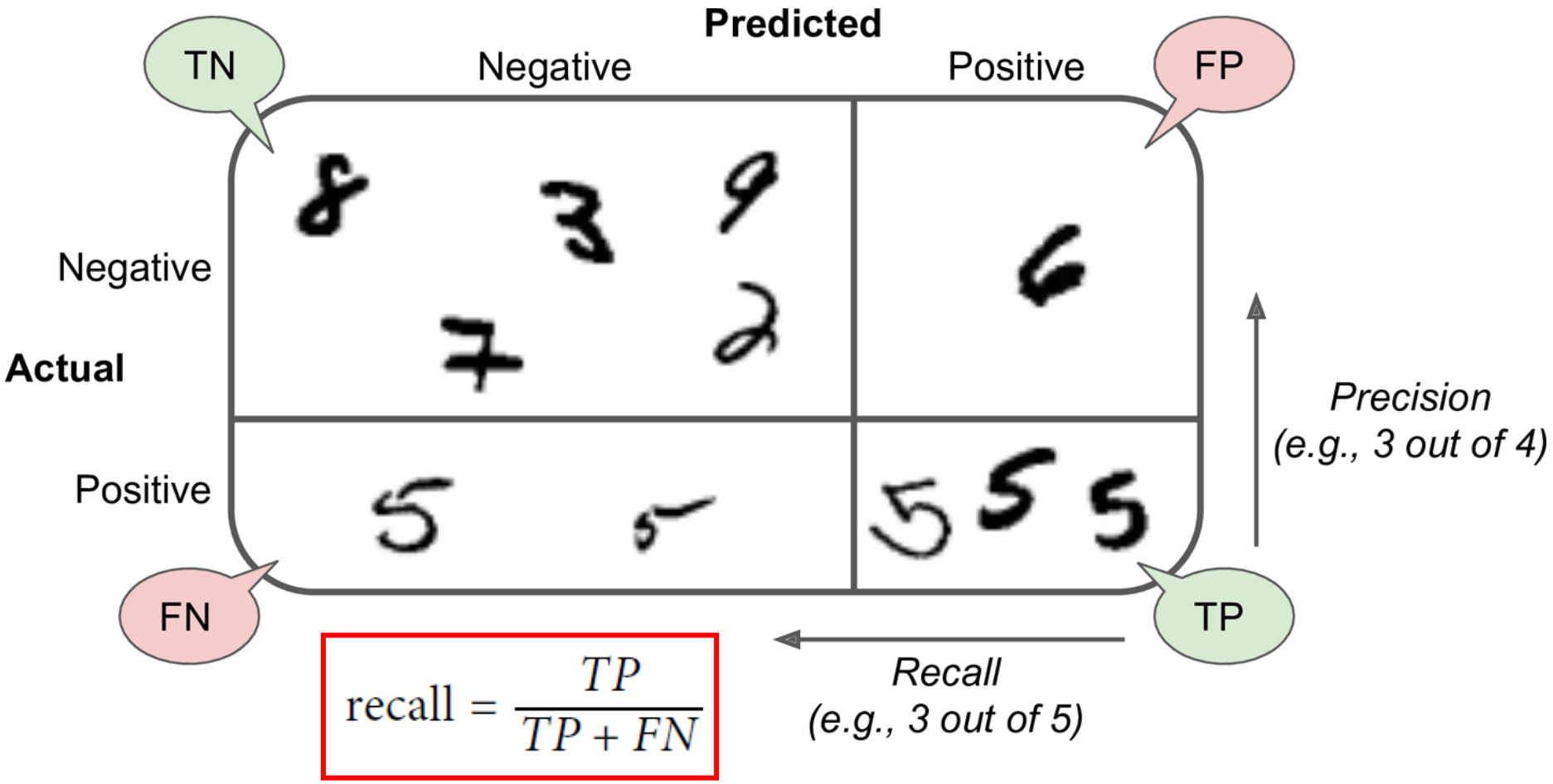
Example how to find the accuracy.

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.9502 ,  0.96565,  0.96495])
```

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

3.2. Confusion Matrix

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



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

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([[53272,  1307],
       [ 1077,  4344]])
```

- The first row is for the non-5s (the negative class):
  - 53,272 correctly classified (*true negatives*)
  - 1,307 wrongly classified (*false positives*)
- The second row is for the 5s (the positive class):
  - 1,077 wrongly classified (*false negatives*)
  - 4,344 correctly classified (*true positives*)

3.3. Precision and Recall

Precision

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

Recall

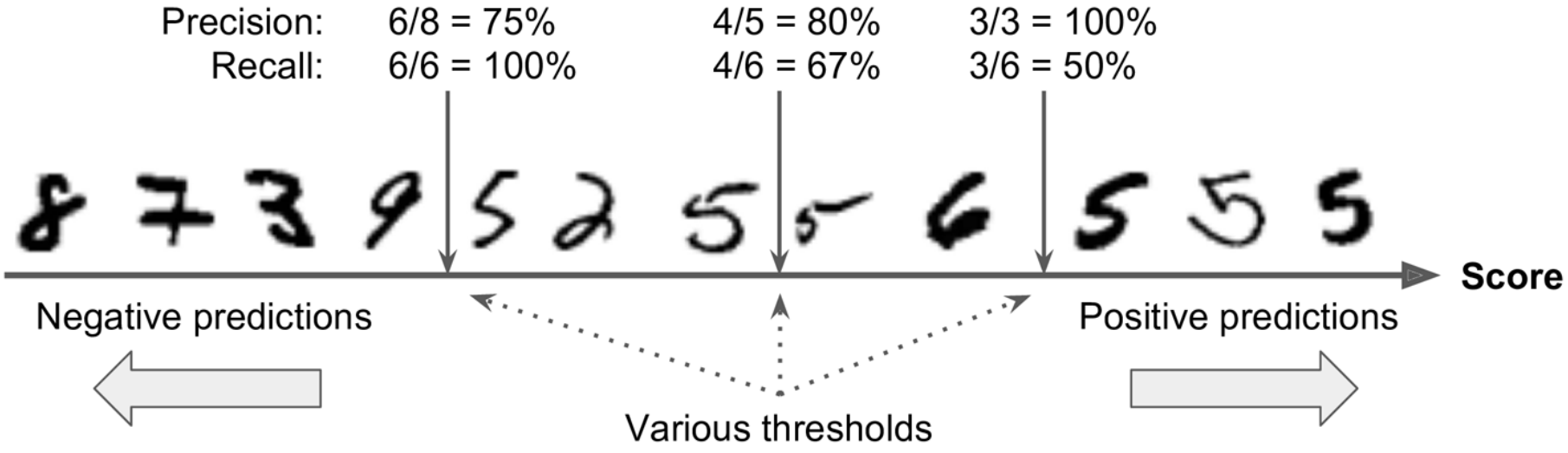
$$\text{recall} = \frac{TP}{TP + FN}$$

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_pred)     # == 4344 / (4344 + 1307)
0.76871350203503808
>>> recall_score(y_train_5, y_train_pred)  # == 4344 / (4344 + 1077)
0.79136690647482011
```

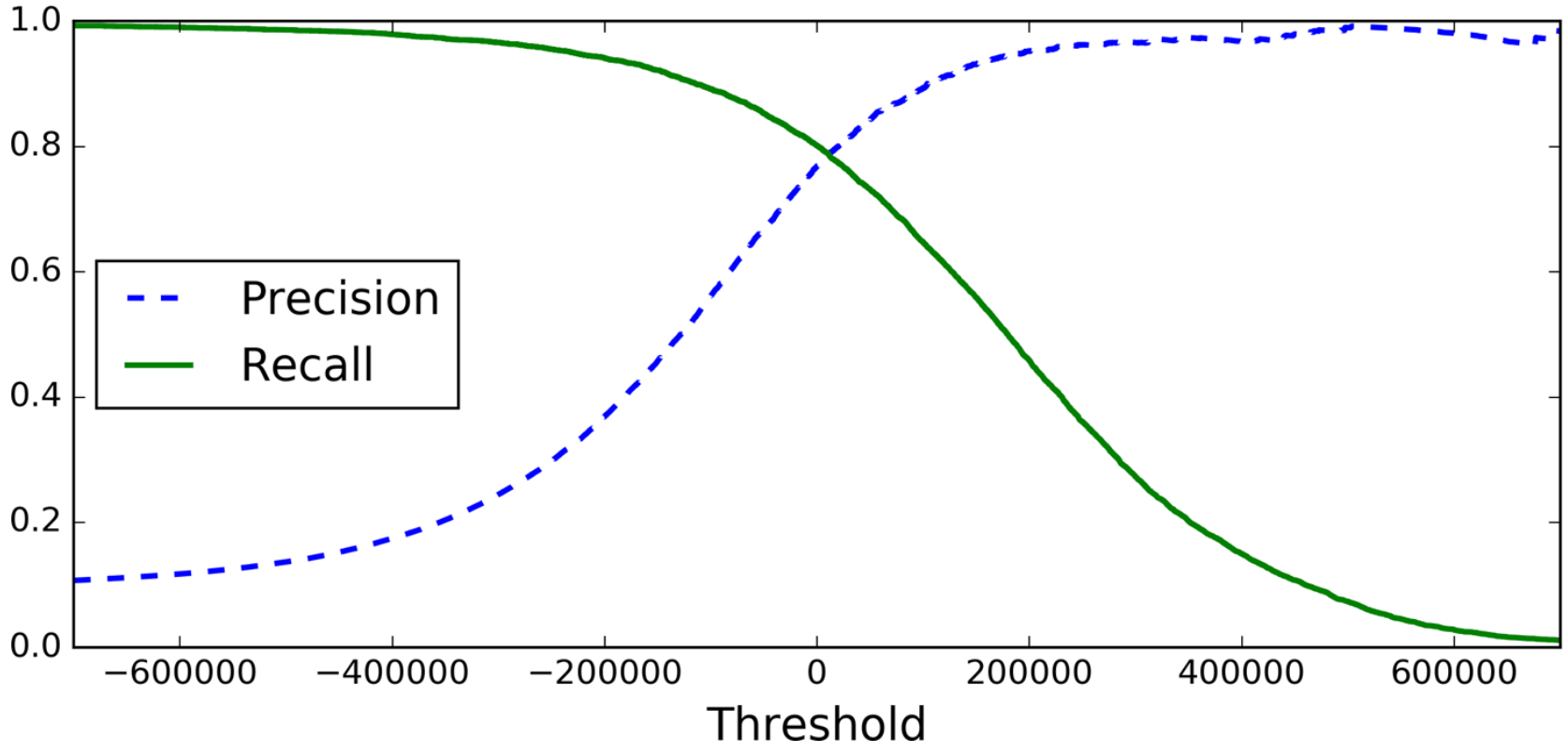
The precision and recall are smaller than the accuracy.
Why?

3.4. 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.



3.4. Precision/Recall Tradeoff



3.4. 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")
```

- When using larger decision threshold, we increase the precision and decrease the recall.

```
y_train_pred_90 = (y_scores > 70000)
>>> precision_score(y_train_5, y_train_pred_90)
0.8998702983138781
>>> recall_score(y_train_5, y_train_pred_90)
0.6399188341634385 3
```

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 × (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.])

>>> forest_clf.fit(X_train, y_train)
>>> forest_clf.predict([some_digit])
array([ 5.])
```

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([ 0.9502 ,  0.96565,  0.96495])
```

- **Multiclass task:**

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([ 0.84063187,  0.84899245,  0.86652998])
```

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).