Machine Learning

Lecture 4: Logistic Regression and Classification

Instructor: Dr. Farhad Pourkamali Anaraki

Introduction

- Logistic regression is used to estimate the probability that an instance belongs to a particular class (out of two classes, e.g., an email is spam or not)

- If the estimated probability is greater than 50%, the model predicts that the instance belongs to the positive class (labeled “1”)

  - otherwise, the model predicts that the instance belongs to the negative class (labeled “0”)

- Like linear regression, logistic regression computes a weighted sum of the input features

- Difference: It applies the logistic function to the output result

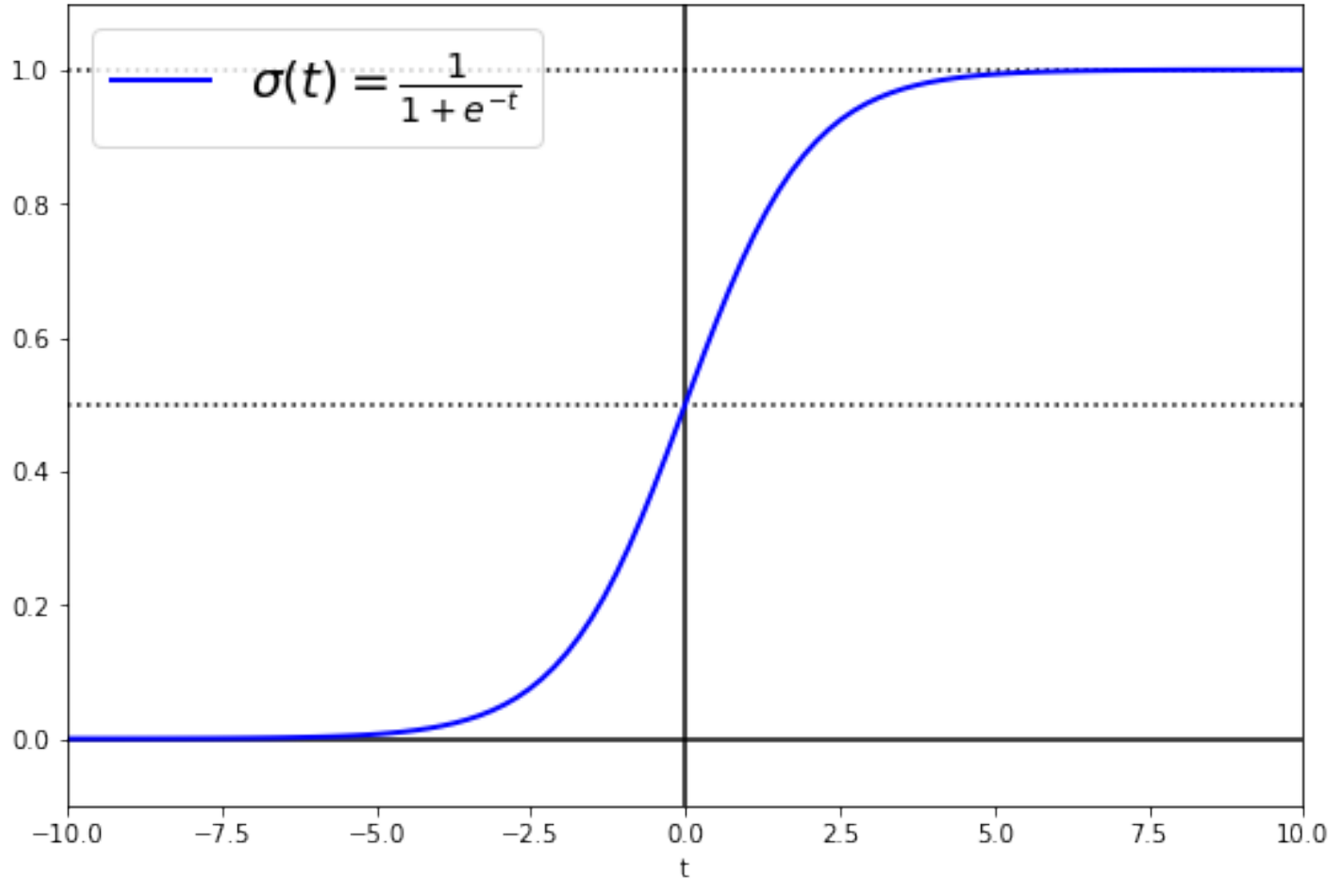
$$\hat{p} = h(\mathbf{x}) = \sigma\left(\theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d\right)$$

Logistic function

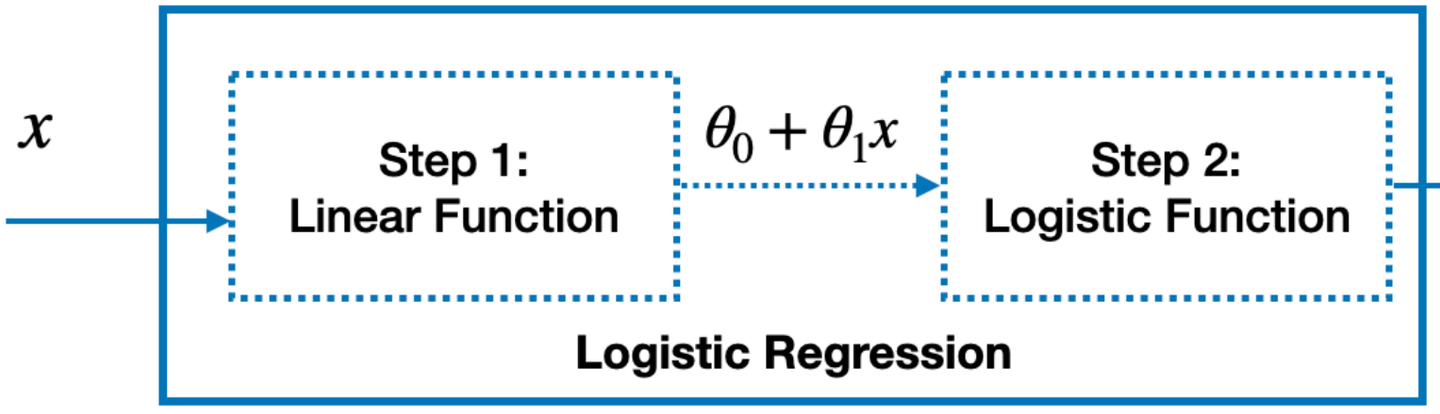
- The logistic or sigmoid function $\sigma(t)$ outputs a number between 0 and 1 as follows:

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

  - The score t is often called the logit.



Summary



$$P(C_1 \mid x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

- Let us find $P(C_0 \mid x)$
- There are two possible outcomes

$$P(C_0 \mid x) + P(C_1 \mid x) = 1 \Rightarrow P(C_0 \mid x) = 1 - P(C_1 \mid x)$$

- Next time, we discuss how to adjust the model's parameters θ_0, θ_1

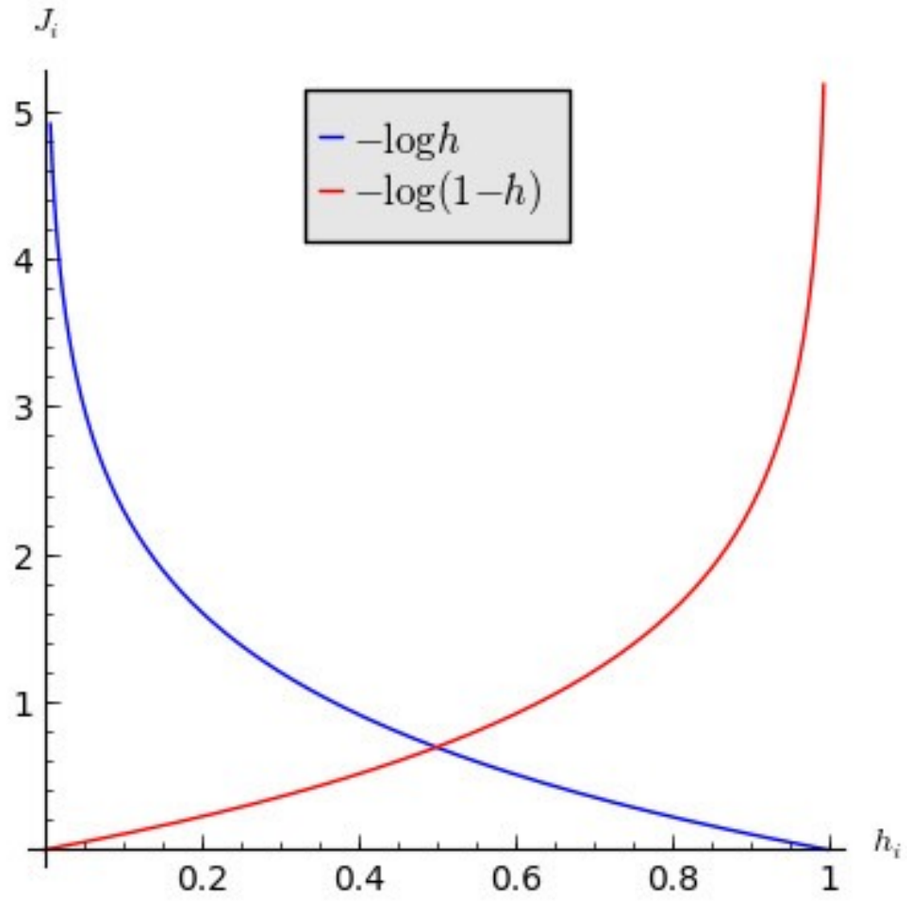
Training and cost function

- The cost function or loss function can be expressed in the general form

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \text{cost}(h(\mathbf{x}^{(i)}), y^{(i)})$$

- Measure of goodness of fit

$$\text{cost}(h(\mathbf{x}), y) = \begin{cases} -\log(h(\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - h(\mathbf{x})) & \text{if } y = 0 \end{cases}$$



Compact representation of cost function

- Note that $y \in \{0,1\}$:

$$\text{cost}(h(\mathbf{x}), y) = -y \log(h(\mathbf{x})) - (1-y)\log(1-h(\mathbf{x}))$$

- Thus, we get the same cost function we saw in the previous slide

- The cost function over the whole training set

$$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} \log(h(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h(\mathbf{x}^{(i)}))\right]$$

Optimization

- The bad news is that there is no known closed-form solution
- The good news is that we can use Gradient Descent to find the vector $\boldsymbol{\theta}$
- Thus, we need to find the gradient of cost function

$$\nabla J(\boldsymbol{\theta}) = \mathbf{X}^T\big(\sigma(\mathbf{X}\boldsymbol{\theta}) - \mathbf{y}\big)$$

$$\begin{bmatrix} \mathbf{x}^{(1)} \\ \cdots \\ \mathbf{x}^{(n)} \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Proof

- We don't go through all required steps to compute the gradient of cost function
- However, we focus on important property of the logistic function

$$\sigma(t)' = \sigma(t)\big(1 - \sigma(t)\big)$$

- Proof:

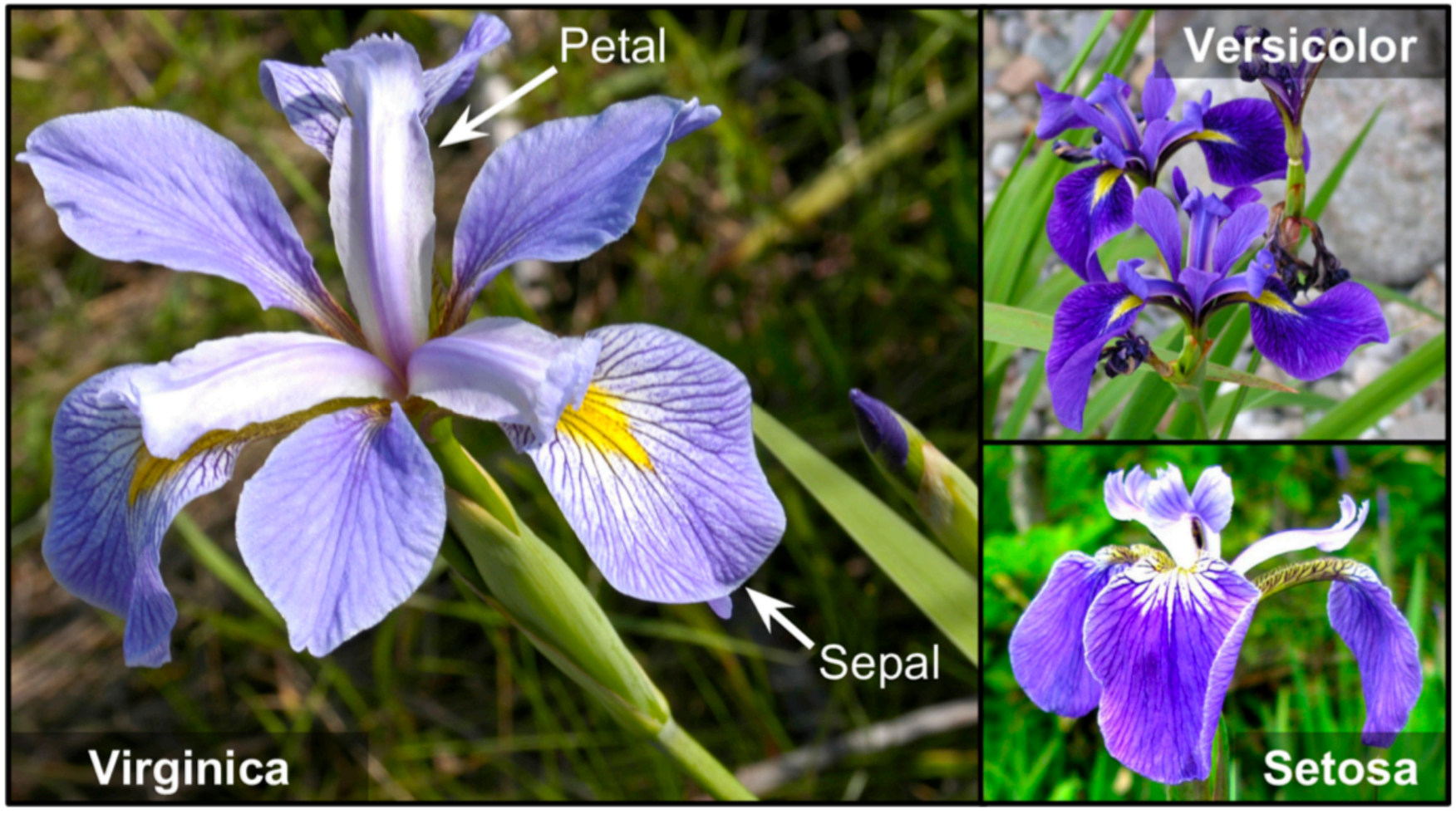
$$\Big((1 + e^{-t})^{-1}\Big)' = -(1 + e^{-t})^{-2}(-e^{-t}) = \frac{e^{-t}}{(1 + e^{-t})^2}$$

- We can write this term as

$$\frac{e^{-t}}{(1 + e^{-t})^2} = \underbrace{\left(\frac{1}{1 + e^{-t}}\right)}_{\sigma(t)} \underbrace{\left(\frac{e^{-t}}{1 + e^{-t}}\right)}_{1-\sigma(t)}$$

Example

- Iris dataset: sepal and petal length and width of 150 iris flowers
  - Three species: setosa, versicolor, and virginica
- Available from sklearn: https://scikit-learn.org/stable/datasets/index.html



Example

- Let us consider one feature (petal width) and detect the Iris virginica type

```
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
```

```
['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
```

```
# features and labels
X = iris["data"][:, 3:]  # petal width
y = (iris["target"] == 2).astype(np.int)  # 1 if Iris virginica, else 0
print(X.shape, y.shape)
```

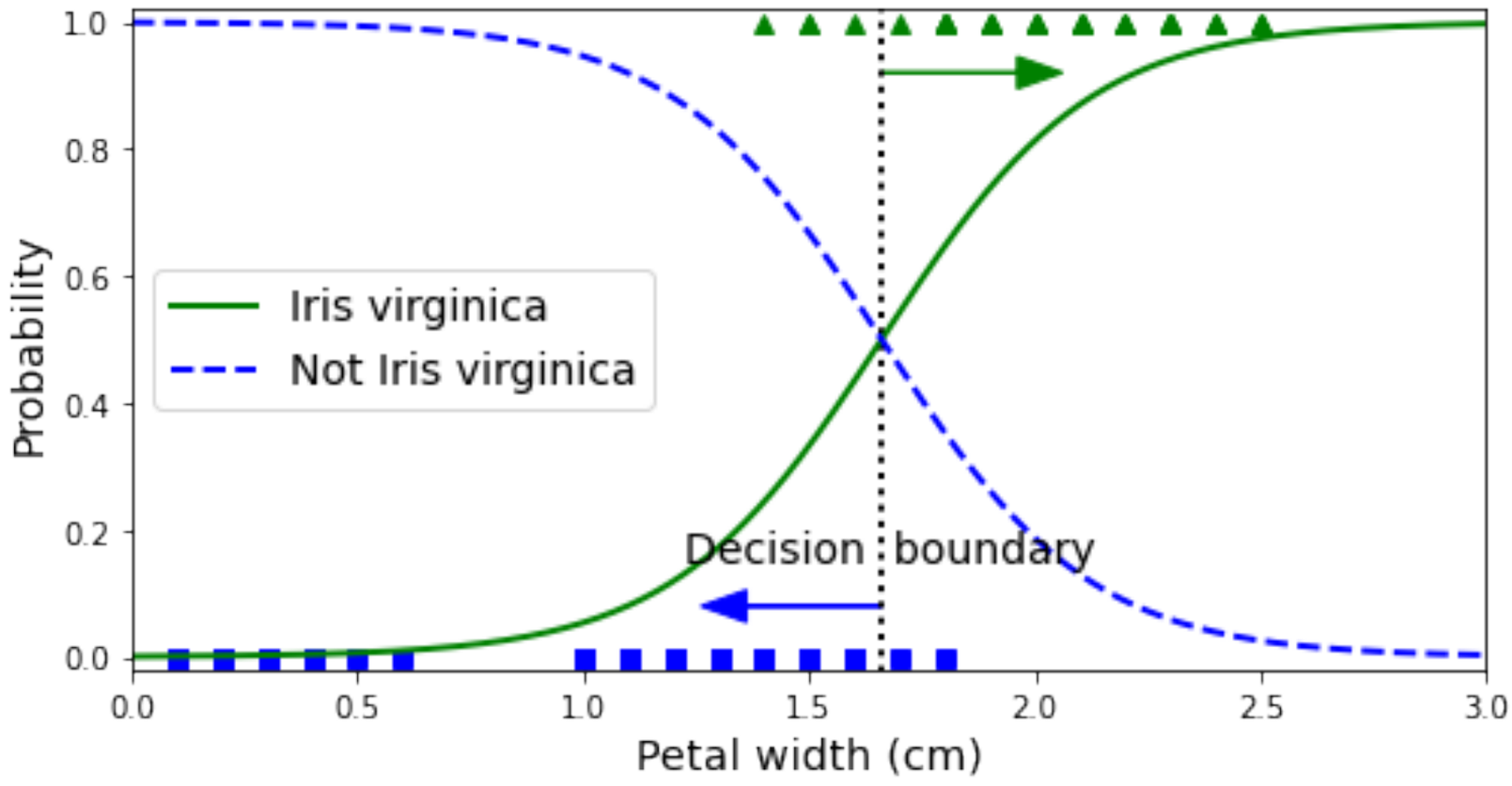
```
(150, 1) (150,)
```

```
# training logistic regression
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="lbfgs", random_state=42)
log_reg.fit(X, y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Decision boundary

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
```

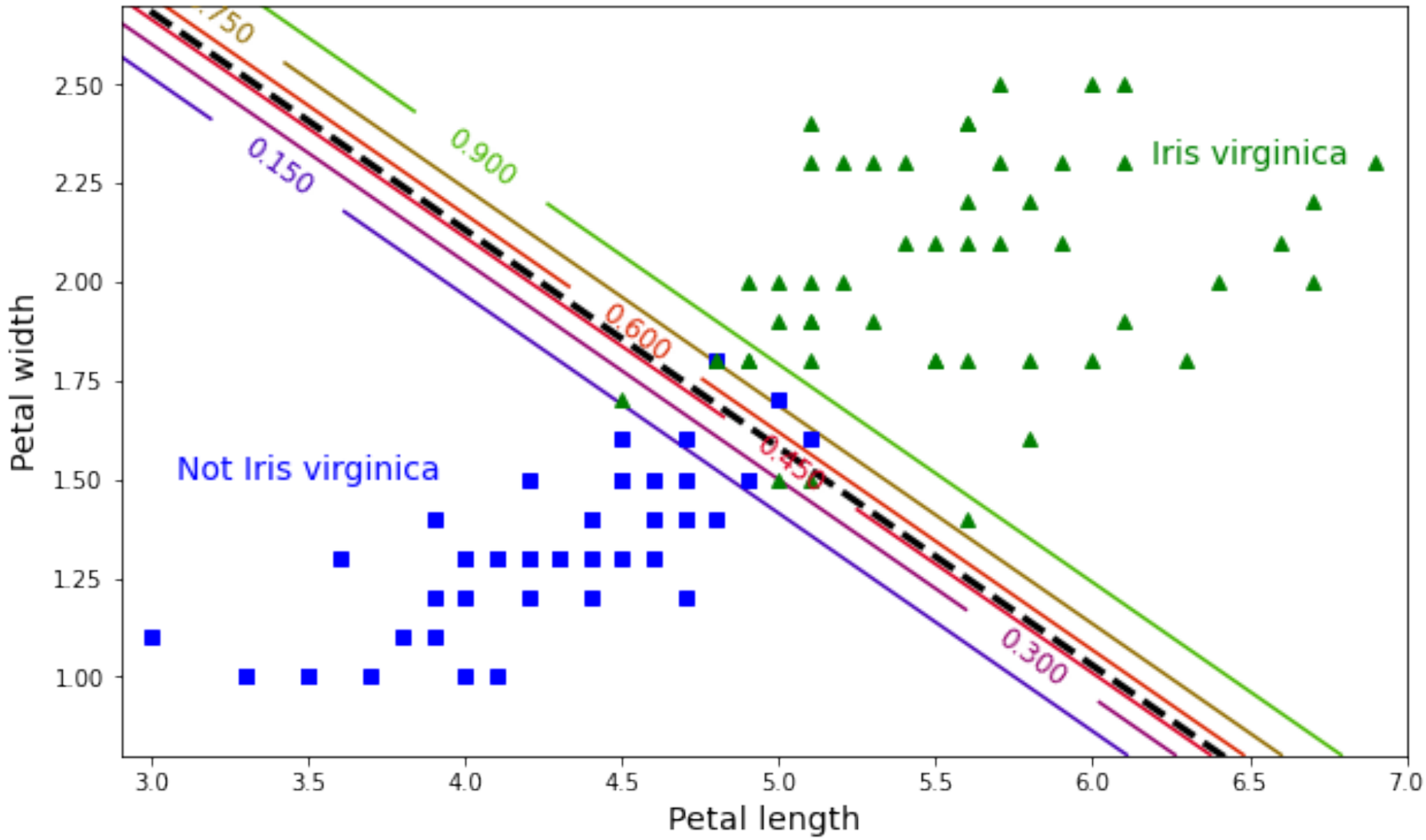



```
log_reg.predict([[1.7], [1.5]])
```

```
array([1, 0])
```

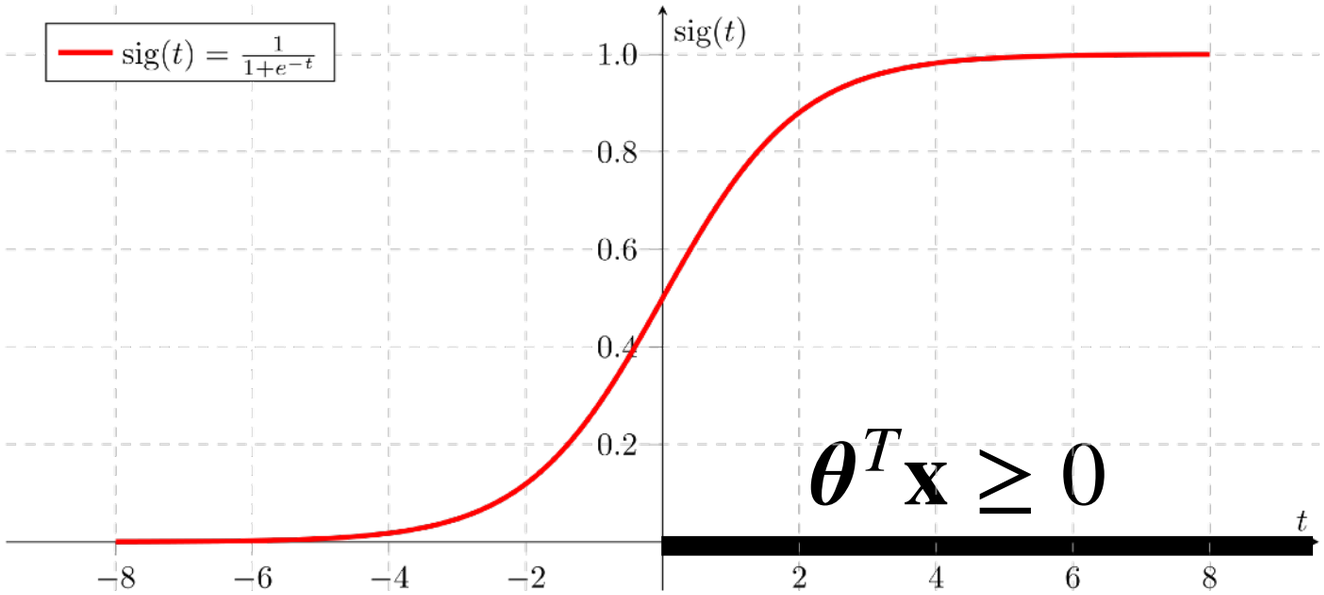
Decision boundary

- In this example, we consider two features: petal width and length

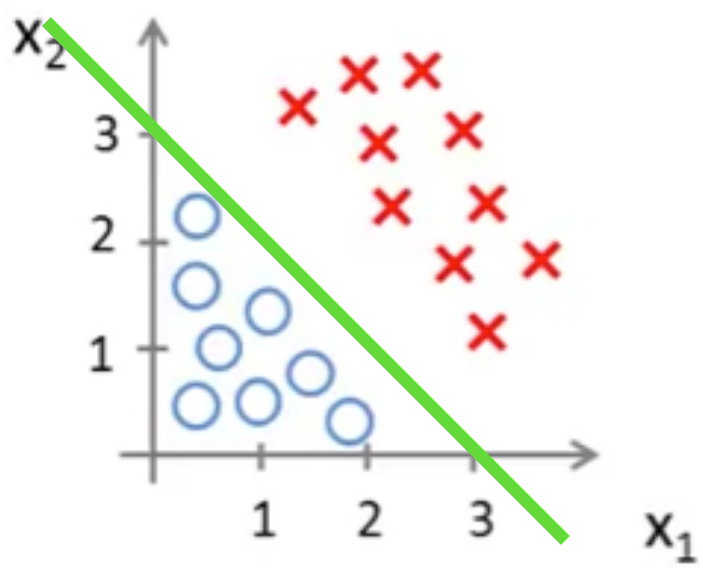


How to find decision boundary?

- Recall that we predict $y = 1$ if $\hat{p} = h(\mathbf{x}) = \sigma(\boldsymbol{\theta}^T \mathbf{x}) \geq 0.5$



- Example: consider $h(\mathbf{x}) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ with $\theta_0 = -3$, $\theta_1 = \theta_2 = 1$



Softmax regression

- The logistic regression model can be generalized to support multiple classes
- Let K be the number of classes and compute a score $s_k(\mathbf{x})$ for each $k \in \{1,\ldots,K\}$

$$s_k(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\theta}^{(k)}$$

- Estimate the probability that the instance $\mathbf{x}$ belongs to class k

$$\widehat{p}_k = \sigma(s(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$$

Prediction

- Softmax regression predicts the class with the highest estimated probability

$$\hat{p} = \text{argmax } \sigma(s(\mathbf{x}))_k = \text{argmax } s_k(\mathbf{x}) = \text{argmax } \mathbf{x}^T \boldsymbol{\theta}^{(k)}$$

- Cross entropy cost function

$$J(\boldsymbol{\Theta}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$$

- where $y_k^{(i)} \in \{0,1\}$ is the target probability

Softmax regression in Scikit-Learn

```
from sklearn.linear_model import LogisticRegression
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = iris["target"]

softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs"
softmax_reg.fit(X, y)
```

```
softmax_reg.predict([[5, 2]])
```

```
array([2])
```

```
softmax_reg.predict_proba([[5, 2]])
```

```
array([[6.38014896e-07, 5.74929995e-02, 9.42506362e-01]])
```

multi_class : *{'auto', 'ovr', 'multinomial'}, default='auto'*

If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, *even when the data is binary*. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.

New in version 0.18: Stochastic Average Gradient descent solver for 'multinomial' case.

Changed in version 0.22: Default changed from 'ovr' to 'auto' in 0.22.

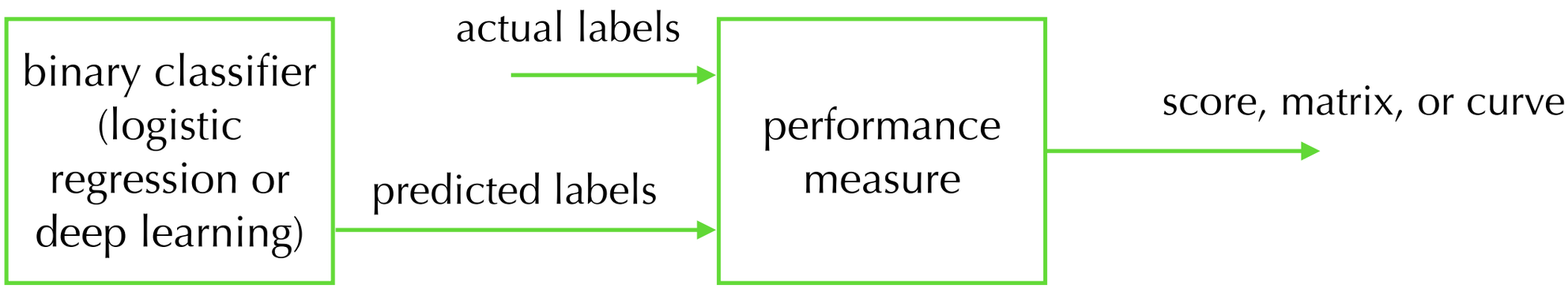
Reading Assignment: Chapter 4 of textbook "Training Models"

Pages 142-151

Performance measures for classifiers

Evaluating performance of classifiers

- In this discussion, we focus on general binary classification tasks



- We work with MNIST dataset consisting of 70,000 images of handwritten digits

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
mnist.keys()
```

```
dict_keys(['data', 'target', 'frame', 'feature_names',
```

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

```
(70000, 784) (70000,)
```

Visualization

```
%matplotlib inline
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)
plt.axis("off")
plt.show()
```

- Let's look at the label

```
y[0]
```

```
'5'
```

→

```
type(y[0])
```

```
str
```

→

```
y = y.astype(np.uint8)
y[0]
```

```
5
```

Visualization

Converting to binary classification

- We simplify the problem by identifying one digit (the number 5)
  - Binary classifier with two classes: 5 or not-5

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)
y_train_5[:12]
```

```
array([ True, False, False, False, False, False, False, False, False,
       False, False,  True])
```

- Training classifier

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

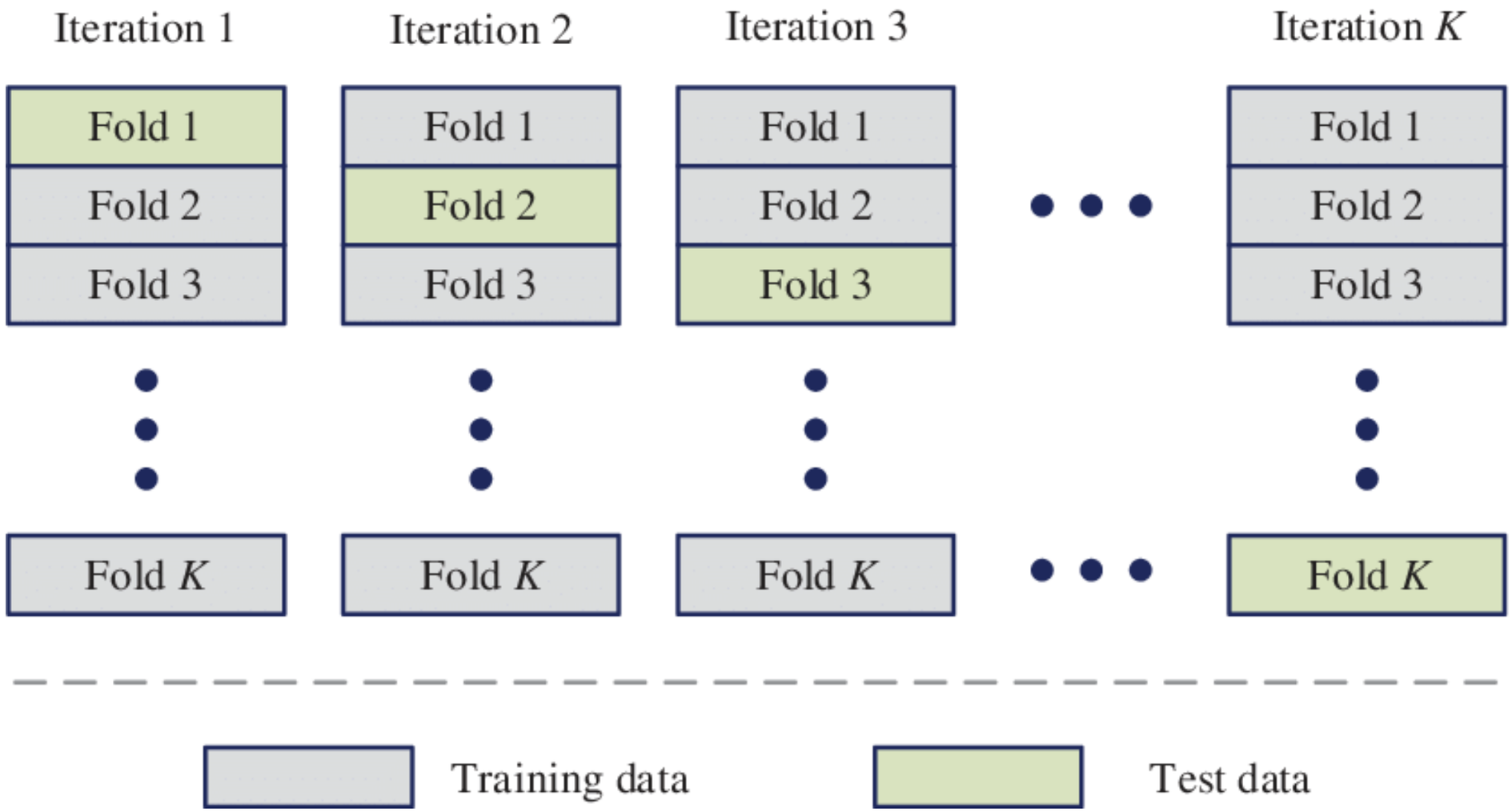
- Predicted labels

```
sgd_clf.predict([some_digit])
```

```
array([ True])
```

Implementing cross-validation

- Recall that we divide the data into K sets or “folds” and use one for testing



- To implement cross-validation, we use (1) our own code and (2) off the shelf scikit-learn function

Two implementations

- Our implementation

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone      (constructs a new estimator with the same parameters)

skfolds = StratifiedKFold(n_splits=3)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))   (ratio of correct predictions)
```

```
0.95035
0.96035
0.9604
```

- Scikit-learn version

```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

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

Analyzing our results

- The accuracy of our classifier is above 95%.
  - Did we really find an accurate classifier?
  - Let's do a simple experiment and define a classifier that never returns 5

```
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)
```

- Perform cross-validation as before

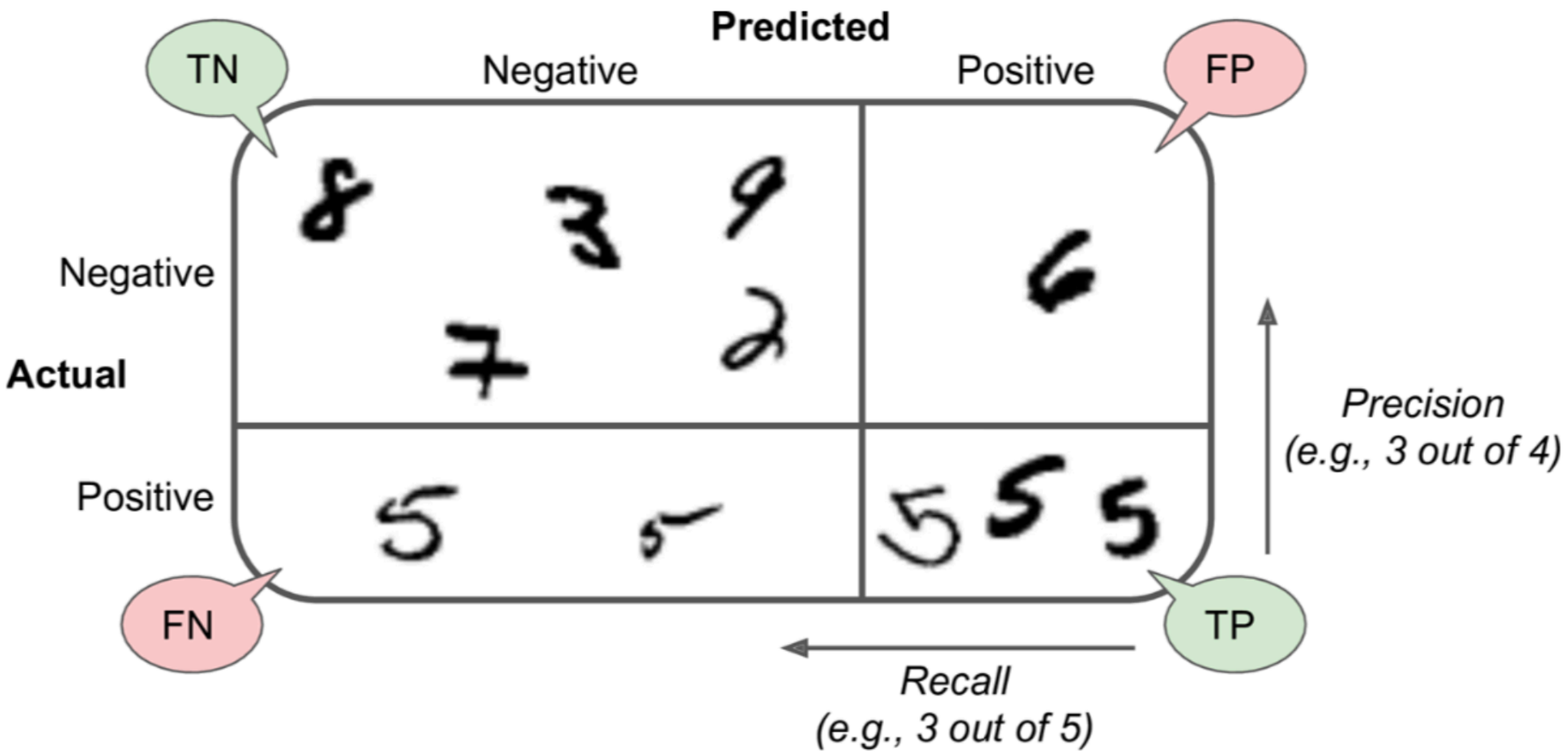
```
never_5_clf = Never5Classifier()
cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

```
array([0.91125, 0.90855, 0.90915])
```

- Accuracy is above 90% without learning

Confusion matrix

- A much better way to evaluate the performance of a classifier is to look at the confusion matrix



- How to create the confusion matrix?

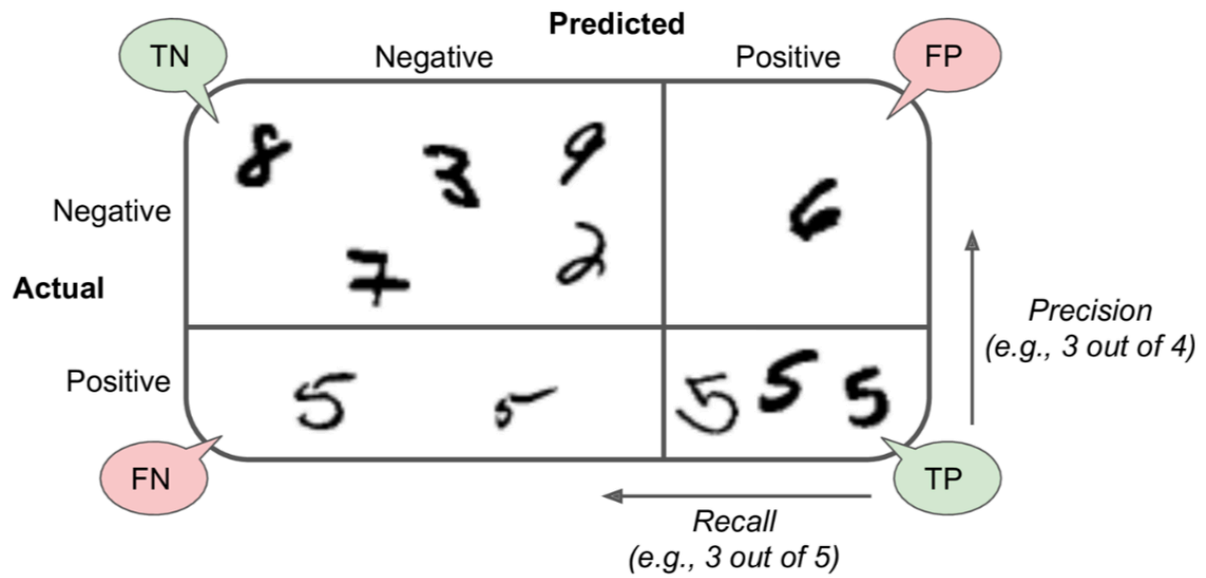
```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

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

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

Precision and recall

- Confusion matrix gives you a lot of information, but sometimes you prefer a more concise metric



- Precision:

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

- Recall or sensitivity:

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

Computing precision and recall

- We use off the shelf functions to compute these scores

```
from sklearn.metrics import precision_score, recall_score

precision_score(y_train_5, y_train_pred)
```

```
0.8370879772350012
```

```
recall_score(y_train_5, y_train_pred)
```

```
0.6511713705958311
```

- Why do we see a significant difference?

- We can combine these two scores into a single metric

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

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

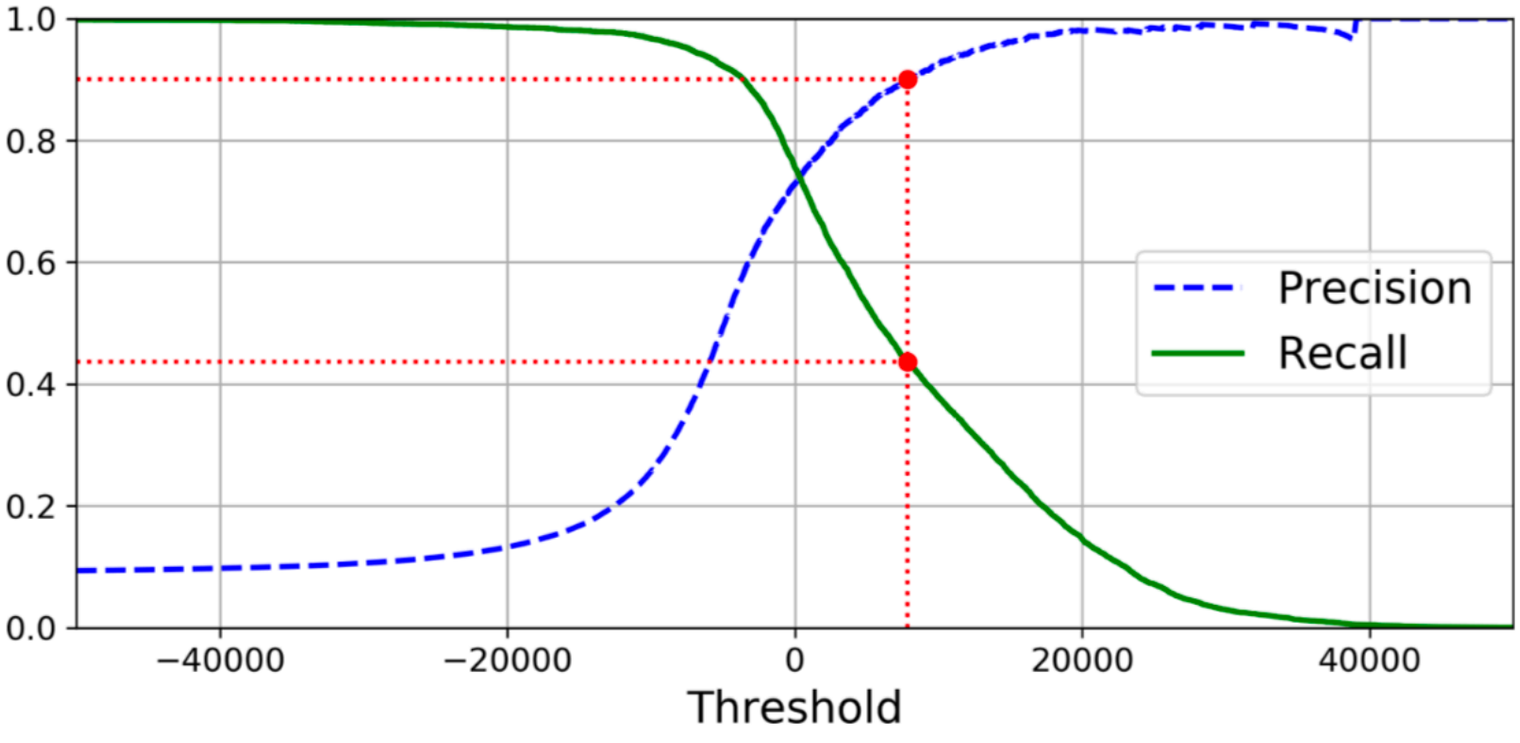
```
0.7325171197343846
```

Precision/recall trade-off

- F_1 score favors classifiers that have similar precision and recall
  - Sometimes we care about precision, and sometime recall
- We can use decision scores instead of calling the classifier's predict() method

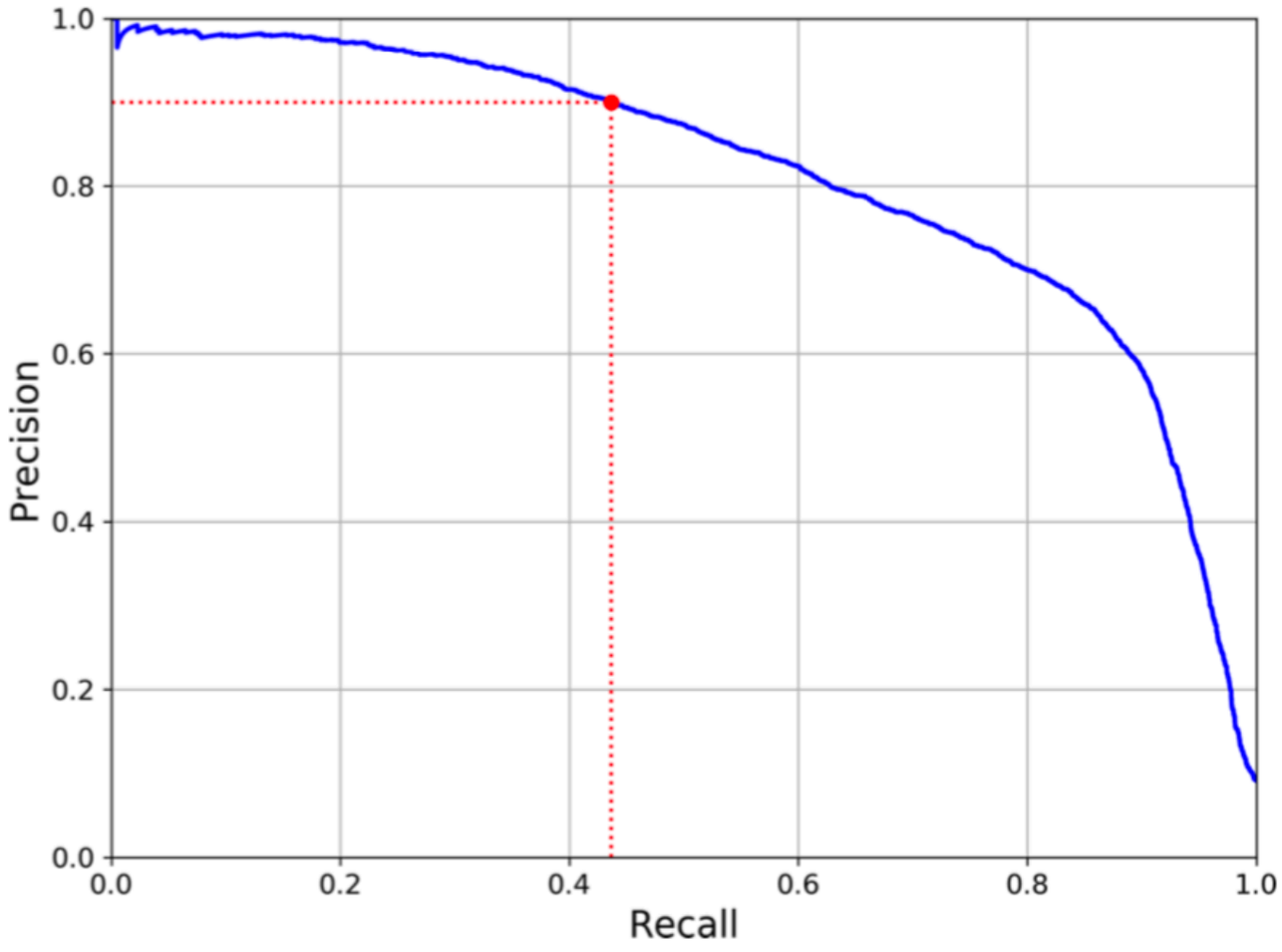
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

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



Precision vs recall

- We can plot precision directly against recall

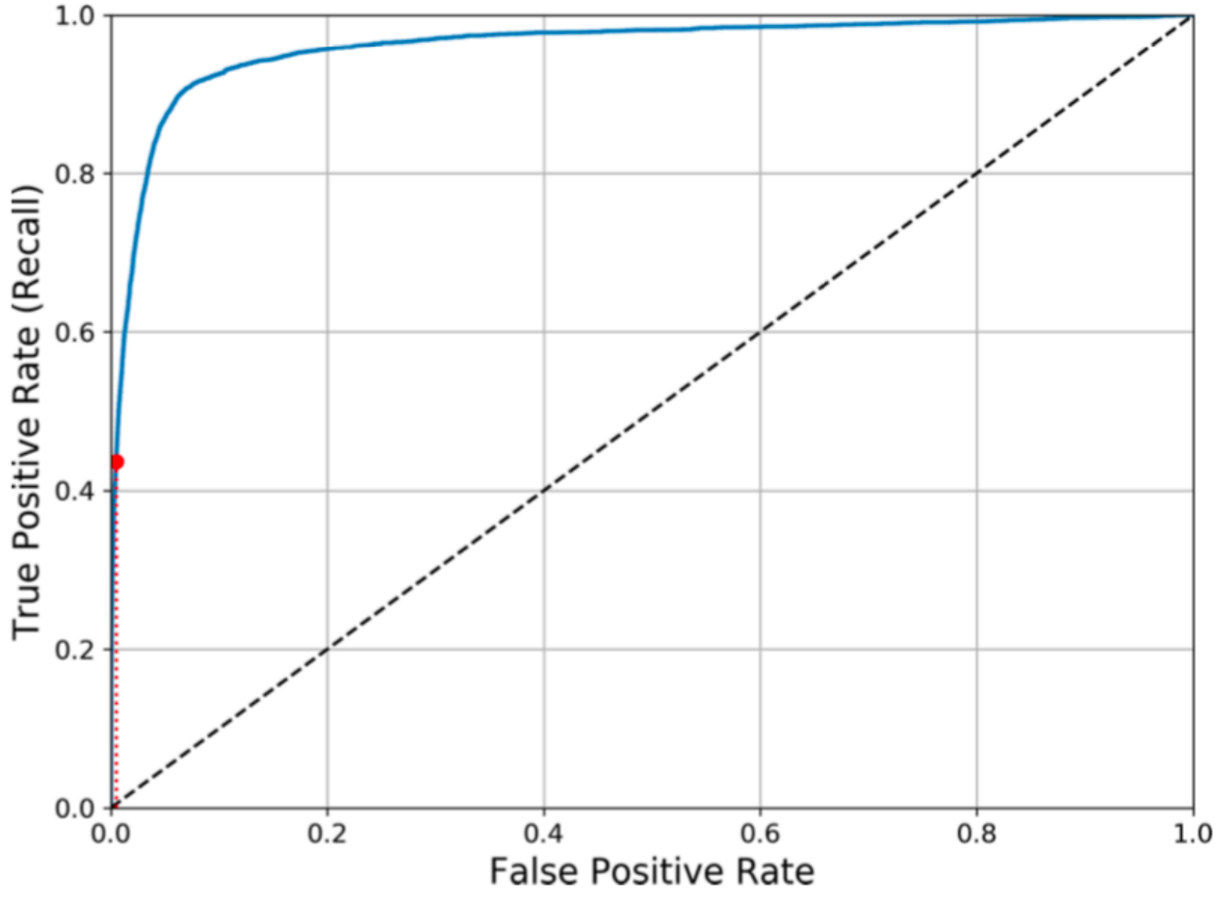


ROC curve

- Receiver Operating Characteristic (ROC) curve plots the true positive rate (TPR) as a function false positive rate (FPR)

$$TPR = \frac{TP}{TP + FN} \text{ (same as recall)}$$

$$FPR = \frac{FP}{FP + TN} = 1 - \frac{TN}{TN + FP} = 1 - TNR$$



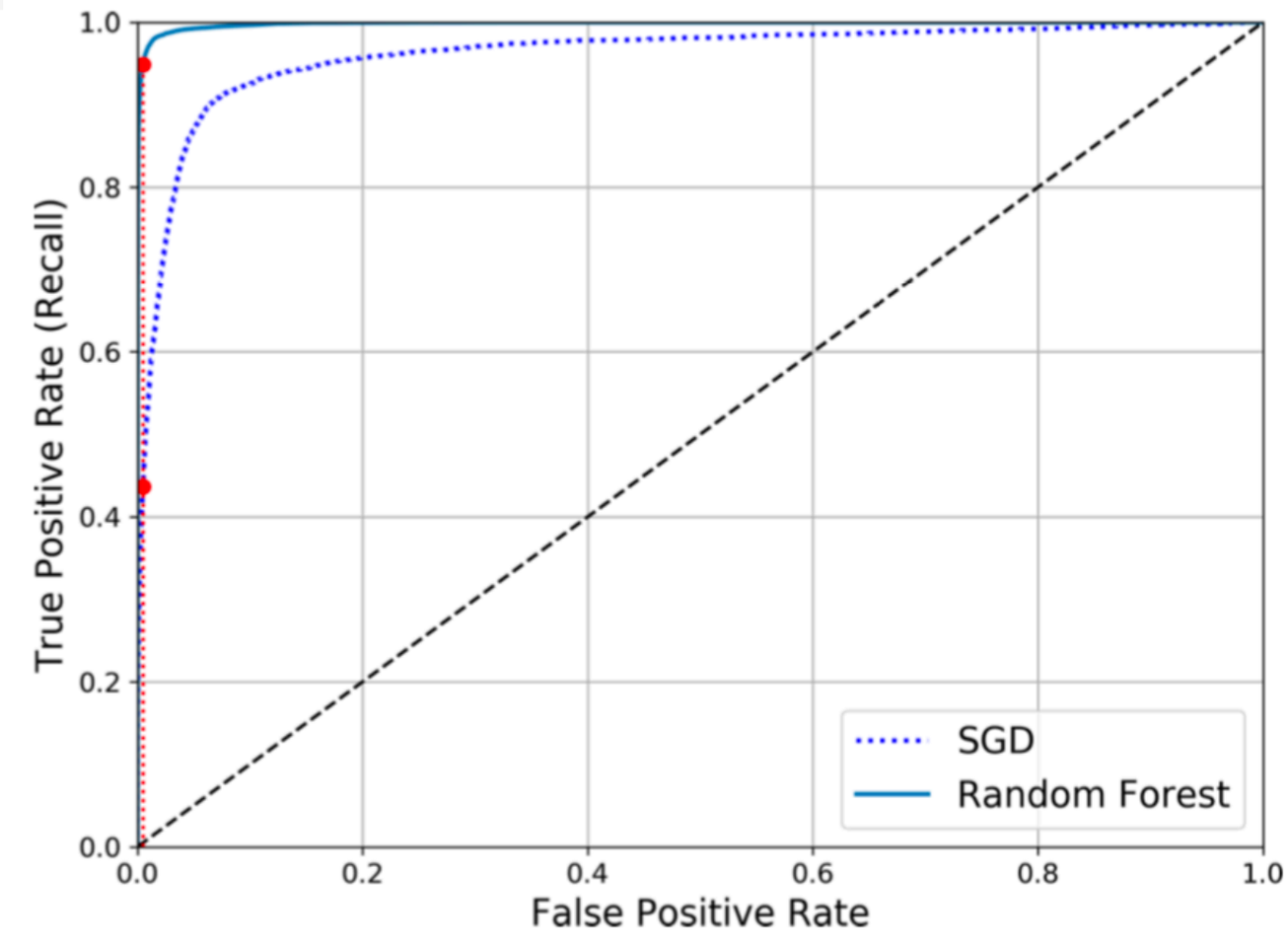
Comparing classifiers

```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

```
from sklearn.metrics import roc_curve
y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)
```

```
print(fpr_forest.shape,tpr_forest.shape)
```

```
(101,) (101,)
```



Multiclass classification

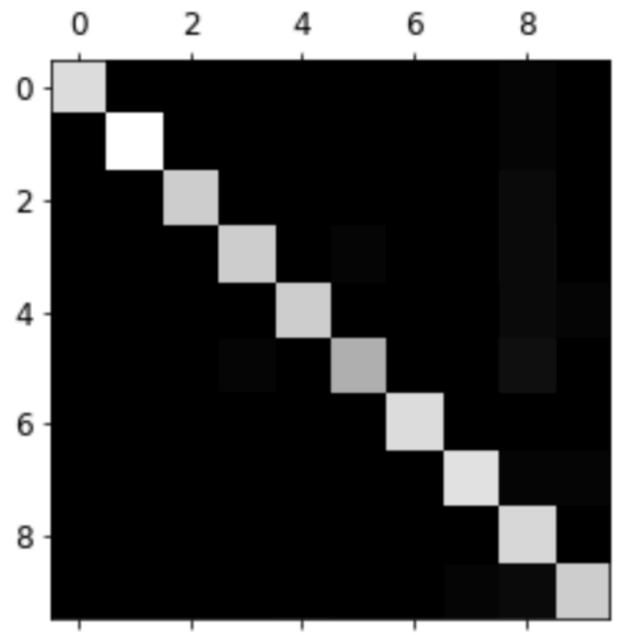
- Two main techniques:
  - One-versus-the-rest (OVR): train 10 binary classifiers, one for each digit (0-detector,…,10-detector) and select the class whose classifier outputs the highest score
  - One-versus-one (OVO): train a binary classifier for every pair of digits
    - For example, one classifier to distinguish 0s and 1s, etc.
    - We need to train $\frac{K(K-1)}{2}$ classifiers
    - However, each classifier needs to be trained on part of the training data

Example

- Train a classifier using all ten digits

```
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]])
```



Reading Assignment: Chapter 3 of textbook "Classification"