Classification

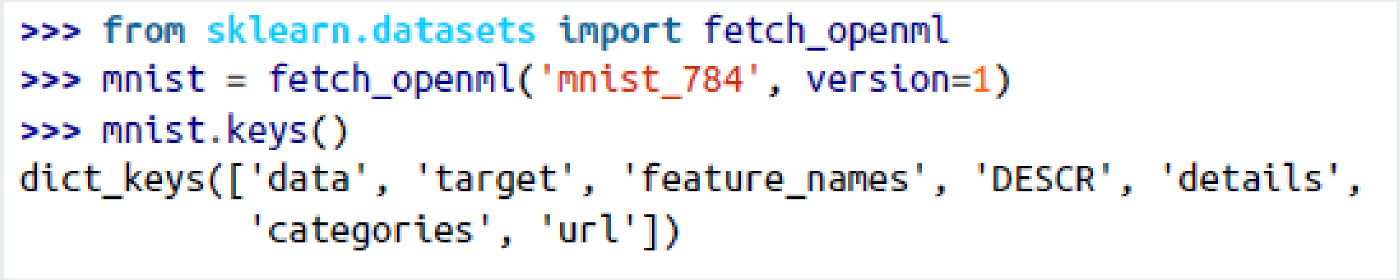
Binary Classifiers

Outline

MNIST

- 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau.

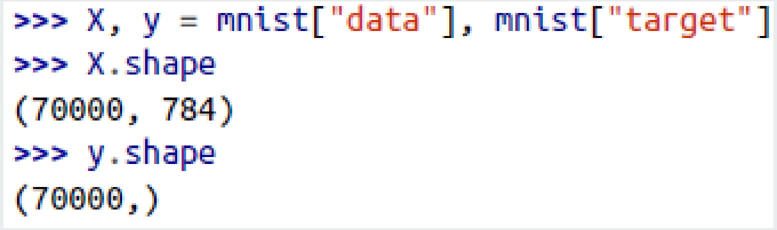
- It has been studied a lot that is often called the “hello world” of Machine Learning: whenever people come up with a new classification algorithm the are curious to see how it will perform on MNIST



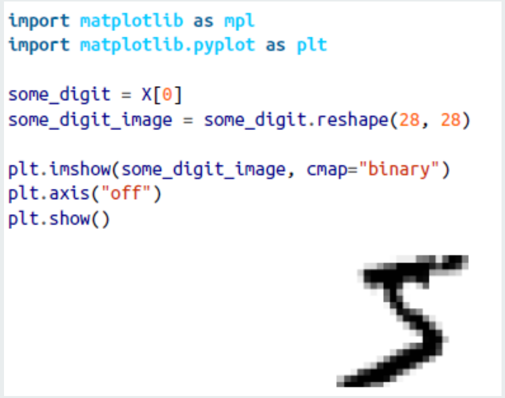
```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1)
>>> mnist.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details',
           'categories', 'url'])
```

Datasets

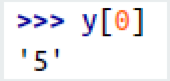
- Usually contains DESCR key describing the dataset
- A data key containing an array with one row per instance and one column per feature
- A target key containing an array with the labels



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



```
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="binary")
plt.axis("off")
plt.show()
```



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

Training a Binary Classifier

- Identify one digit – for example, the number 5.

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

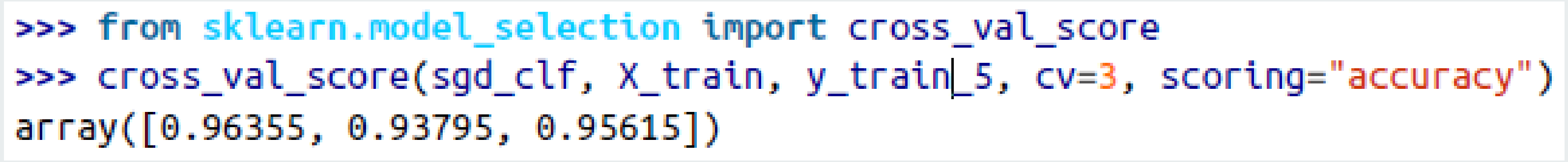
- Stochastic Gradient Descent (SGD) classifier, using Scikit-Learn’s SGDClassifier class.

```
from sklearn.linear_model import SGDClassifier

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

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

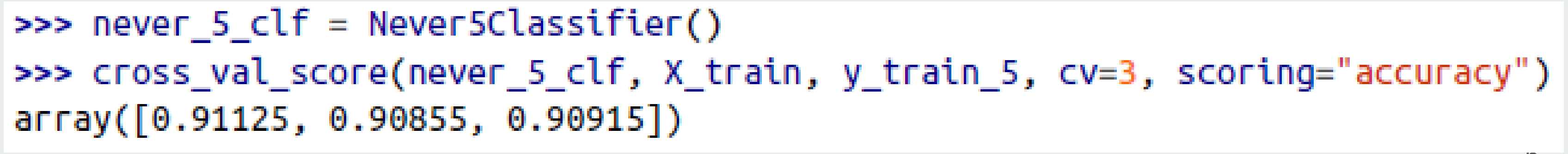
Performance Measures

- Evaluate a model 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")
array([0.96355, 0.93795, 0.95615])
```

- Accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets.



```
>>> 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])
```

Confusion Matrix

- Evaluate the performance using confusion matrix: it is a matrix to check if the prediction/algorithm is correctly predicted the label of the actuals.

```
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([[53057,  1522],
       [ 1325,  4096]])
```

Actual/Predition	Not 5	It is 5
Not 5	TN	FP
It is 5	FN	TP

Confusion Matrix

- Precision

Equation 3-1. Precision

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

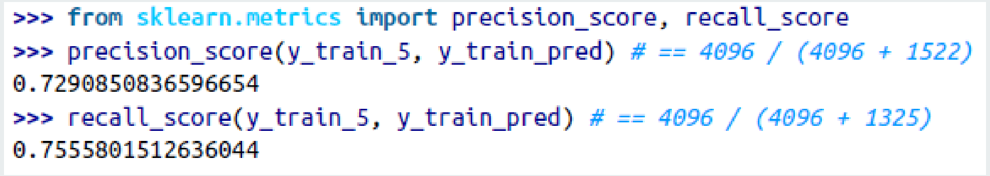
- Recall:this is the ratio of positive instances that are correctly detected by the classifier

Equation 3-2. Recall

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

Actual/Predition	Not 5	It is 5	
Not 5	TN	FP	
It is 5	FN	TP	Recall/Sensitivity/TPR
		Precision	

Confusion Matrix



```
>>> 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
```

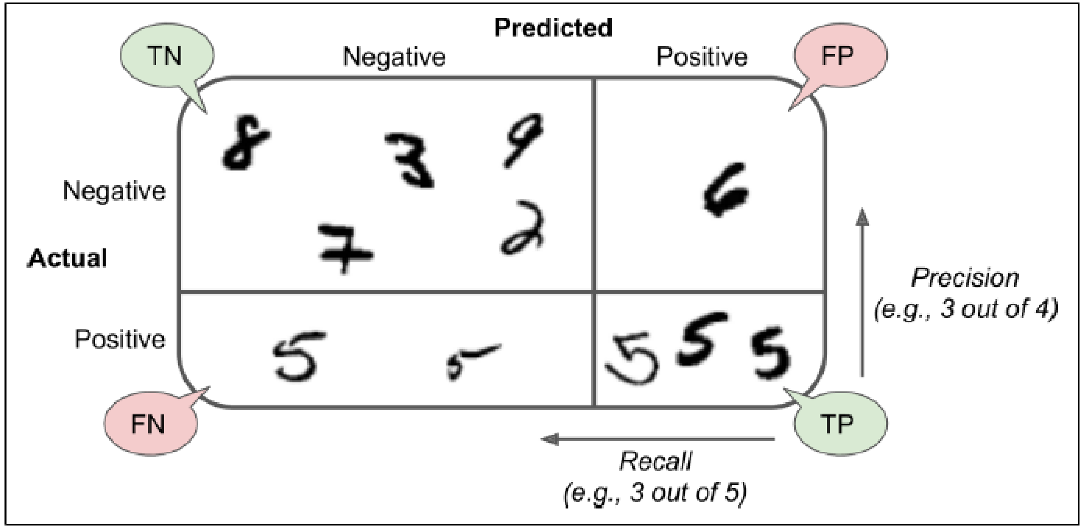


Figure 3-2. An illustrated confusion matrix shows examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)

Confusion Matrix: F-1 Score

- Precision

Equation 3-1. Precision

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

Actual/Predition	Not 5	It is 5	
Not 5	TN	FP	
It is 5	FN	TP	Recall/Sensitivity/TPR
		Precision	

- Recall: this is the ratio of positive instances that are correctly detected by the classifier

Equation 3-2. Recall

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

- Harmonic mean of precision and recall.

Equation 3-3. F_1 score

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

Confusion Matrix: Precision and Recall Tradeoffs

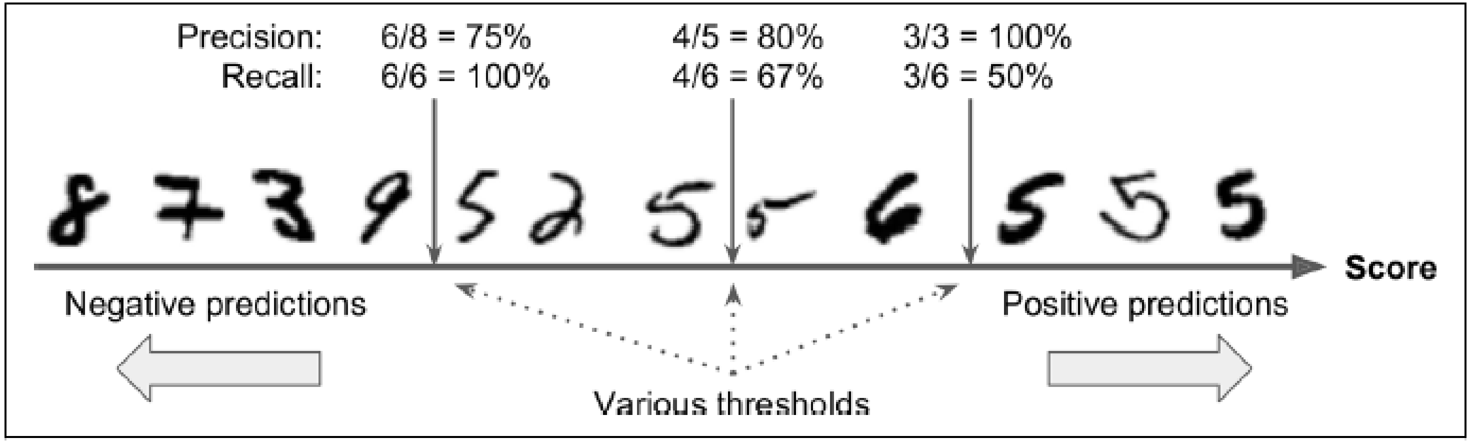

Precision: 6/8 = 75% 4/5 = 80% 3/3 = 100%
Recall: 6/6 = 100% 4/6 = 67% 3/6 = 50%
Score
Negative predictions
Positive predictions
Various thresholds

Figure 3-3. In this 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

Confusion Matrix: Precision and Recall Tradeoffs

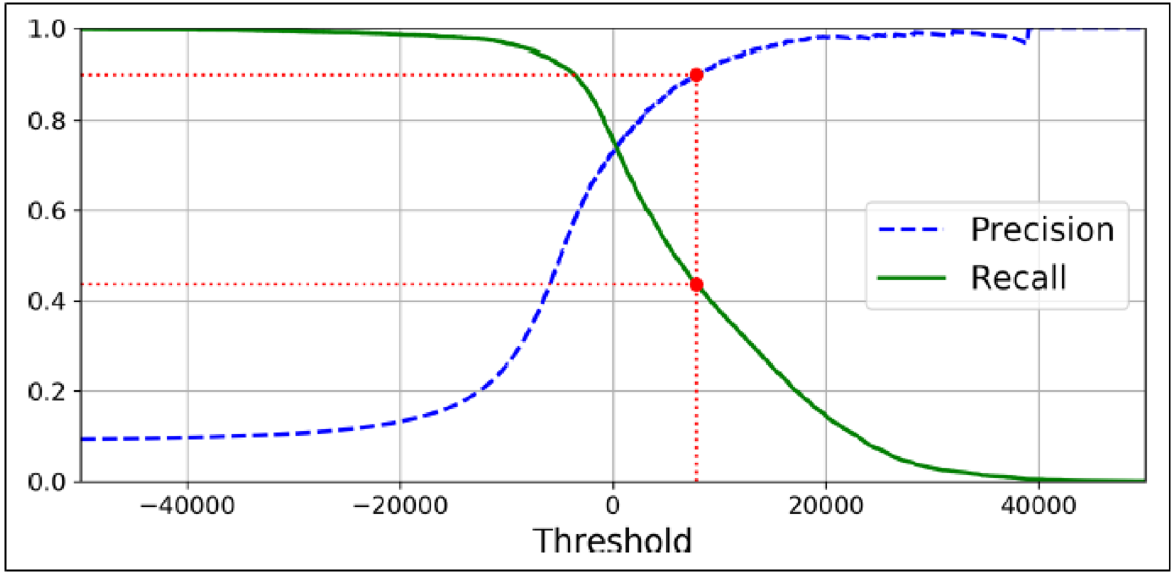


Figure 3-4. Precision and recall versus the decision threshold

Confusion Matrix: Precision and Recall Tradeoffs

- Great, you have a 90% precision classifier! As you can see, it is fairly easy to create a classifier with virtually any precision you want: just set a high enough threshold, and you're done. But wait, not so fast. A high-precision classifier is not very useful if its recall is too low!

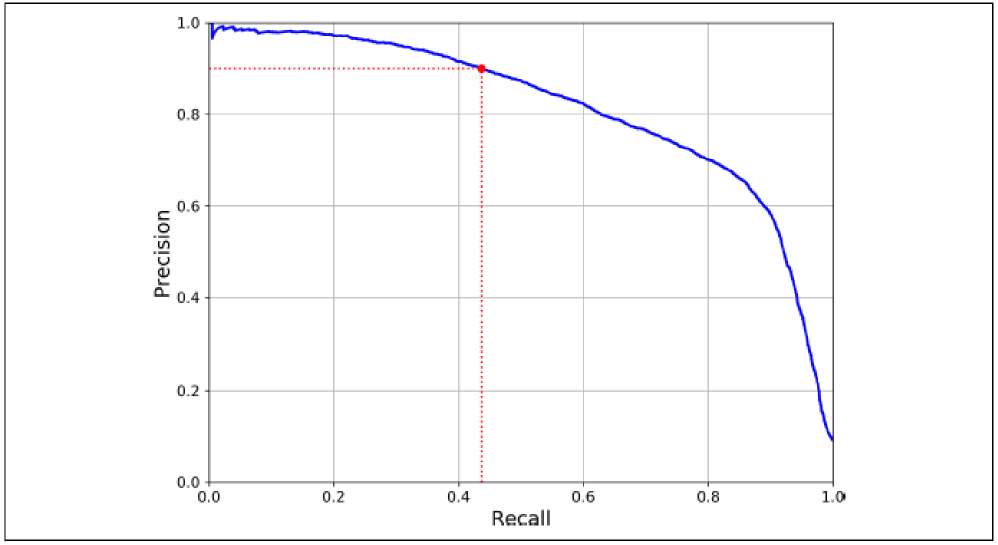
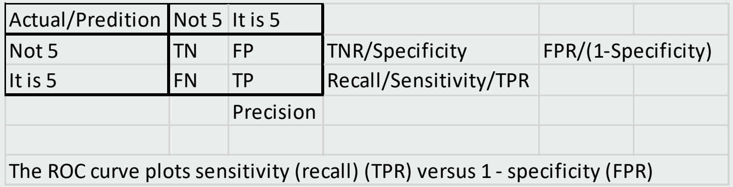


Figure 3-5. Precision versus recall

The ROC Curve

- The *receiver operating characteristic* (ROC) curve is another common tool used with binary classifiers.
- It is very similar to the precision/recall curve, but instead of plotting precision versus recall, the ROC curve plots the *true positive rate* (another name for recall) against the *false positive rate* (FPR).



Actual/Predition	Not 5	It is 5			
Not 5	TN	FP	TNR/Specificity	FPR/(1-Specificity)	
It is 5	FN	TP	Recall/Sensitivity/TPR		
		Precision			
The ROC curve plots sensitivity (recall) (TPR) versus 1 - specificity (FPR)					

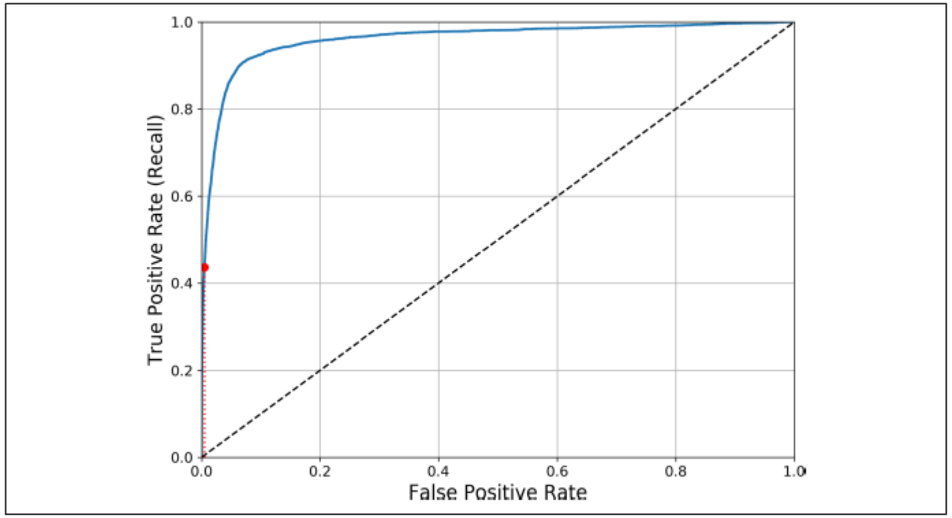
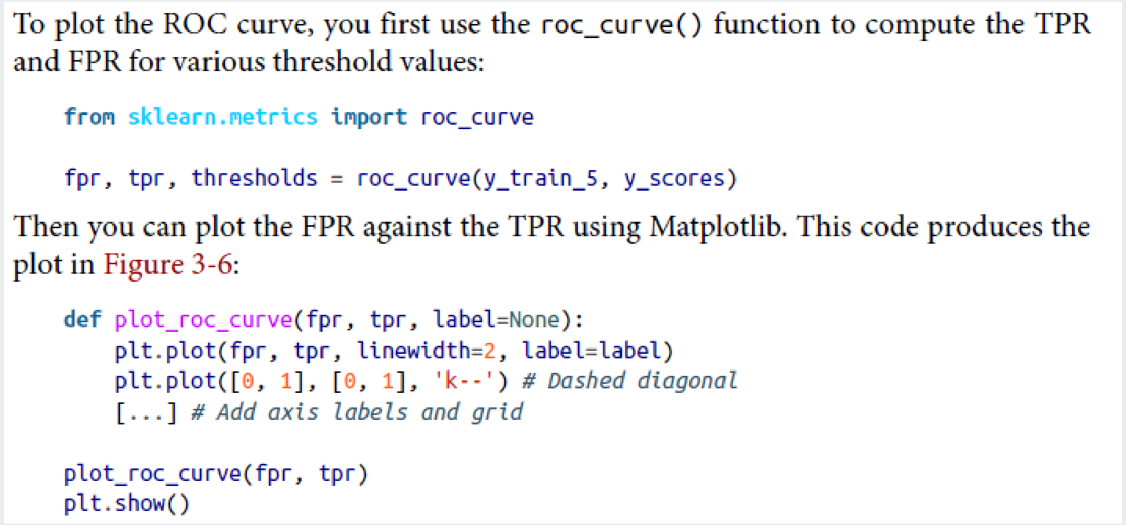


Figure 3-6. This ROC curve plots the false positive rate against the true positive rate for all possible thresholds; the red circle highlights the chosen ratio (at 43.68% recall)

The ROC Curve: Scikit-Learn

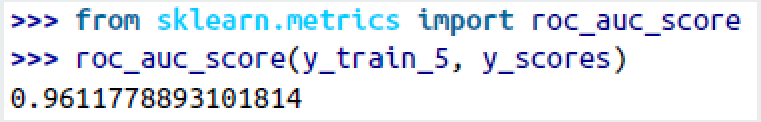


To plot the ROC curve, you first use the `roc_curve()` function to compute the TPR and FPR for various threshold values:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

Then you can plot the FPR against the TPR using Matplotlib. This code produces the plot in Figure 3-6:

```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal
    [...] # Add axis labels and grid

plot_roc_curve(fpr, tpr)
plt.show()
```



```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.9611778893101814
```

The ROC Curve: Comparisons

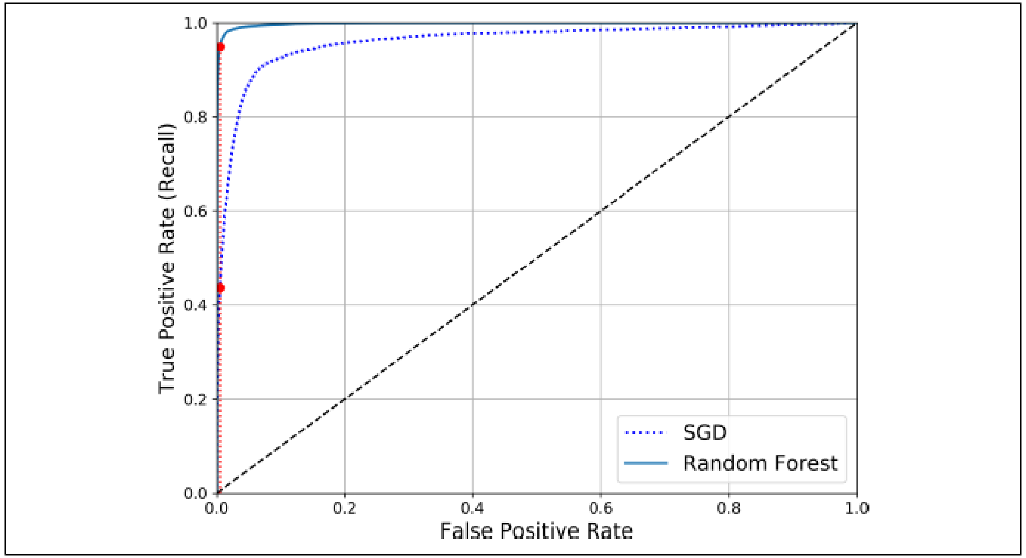


Figure 3-7. Comparing ROC curves: the Random Forest classifier is superior to the SGD classifier because its ROC curve is much closer to the top-left corner, and it has a greater AUC

Outline

- MNIST
- Training a Binary Classifier
- Performance Measures
- **Multiclass Classification**
- **Error Analysis**
- **Multilabel Classification**
- **Multioutput Classification**

Multiclass Classification

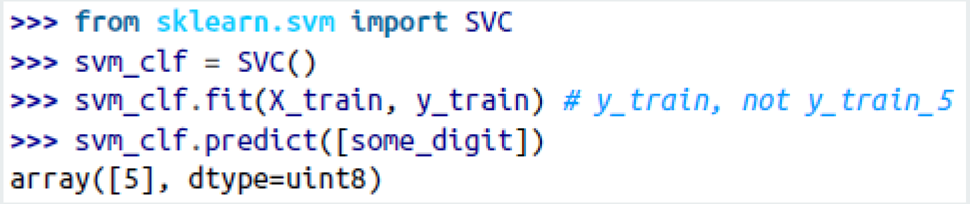
- Multiclass classifier can distinguish between more than two classes
- Multiclass classifiers: Logistic Regression, Random Forest, Naïve Bayes
- Binary classifiers: SGD classifier, Support Vector Machine classifier

Multiclass Classification

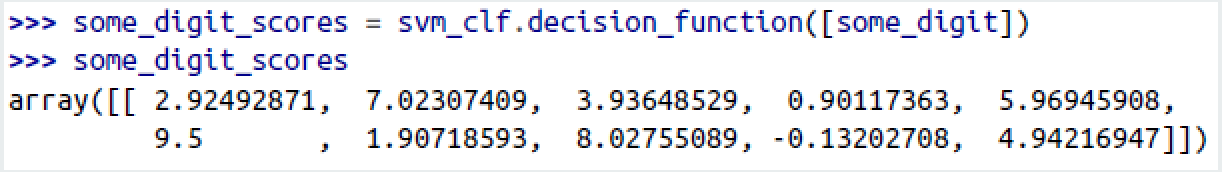
- One-versus-the-rest (OvR) strategy
  - Get multiple binary classifiers
  - Output the highest score based on each binary classifier

- One-versus-one (OvO) strategy
  - One to distinguish 0s and 1s
  - One to distinguish 0s and 2s
  - And so on…
  - N*(N-1)/2 classifiers
  - SVC is not scalable with the size of the training set, it is using OvO strategy
  - Most other classifers are using OvR strategy

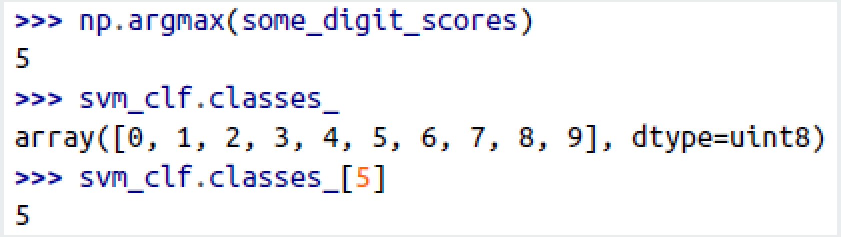
Multiclass Classification: Support Vector Classifier (SVC)



```
>>> from sklearn.svm import SVC
>>> svm_clf = SVC()
>>> svm_clf.fit(X_train, y_train) # y_train, not y_train_5
>>> svm_clf.predict([some_digit])
array([5], dtype=uint8)
```



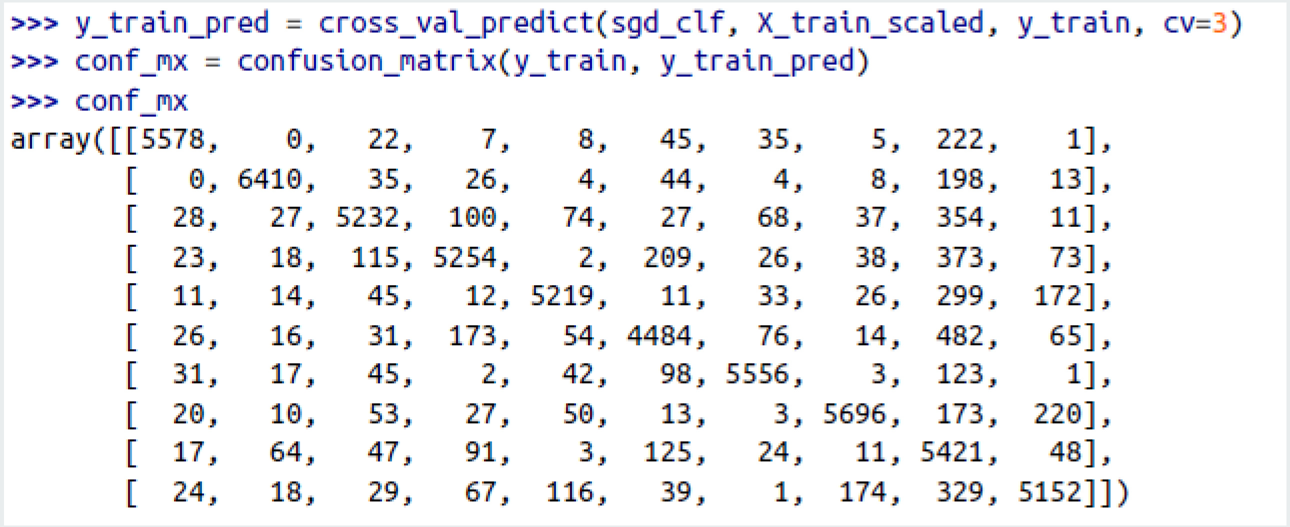
```
>>> some_digit_scores = svm_clf.decision_function([some_digit])
>>> some_digit_scores
array([[ 2.92492871,  7.02307409,  3.93648529,  0.90117363,  5.96945908,
         9.5       ,  1.90718593,  8.02755089, -0.13202708,  4.94216947]])
```



```
>>> np.argmax(some_digit_scores)
5
>>> svm_clf.classes_
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
>>> svm_clf.classes_[5]
5
```

- Scikit-Learn detects when you try to use a binary classification algorithm for a multiclass classification task, and it automatically runs OvR or OvO, depending on the algorithm.

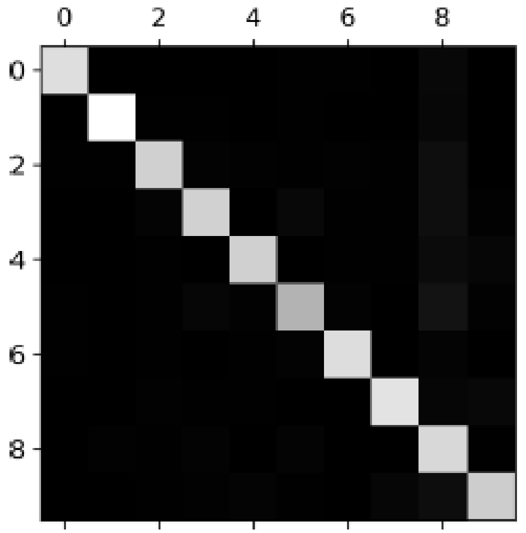
Error Analysis



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

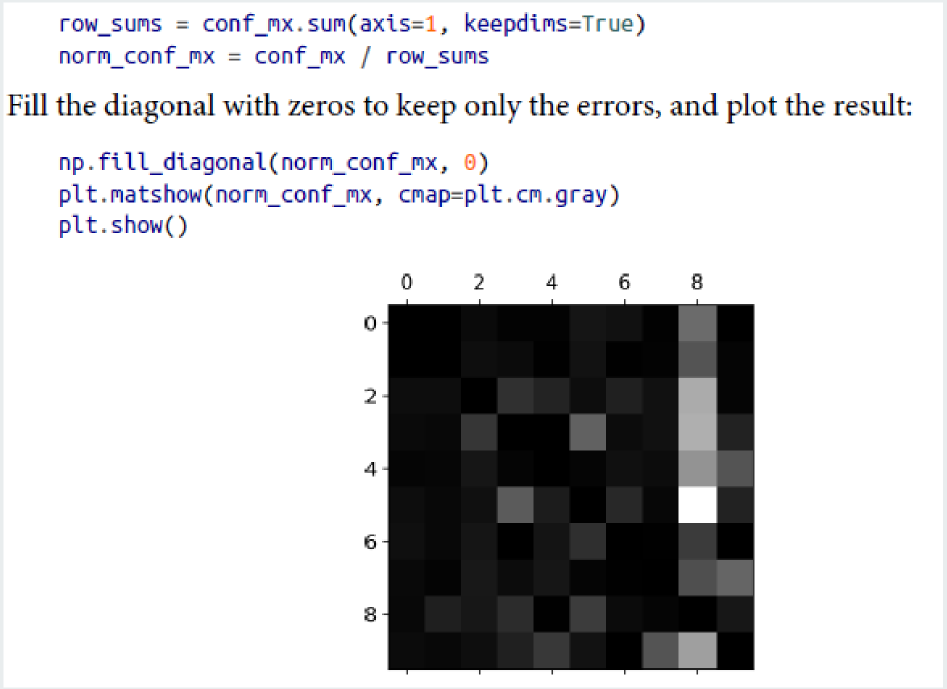
Error Analysis

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



Error Analysis

- The column for class 8 is quite bright, indicating many images get misclassified as 8s.
- Confusion matrix may not be symmetric
- Help you enhance your algorithm by focusing on where the errors are



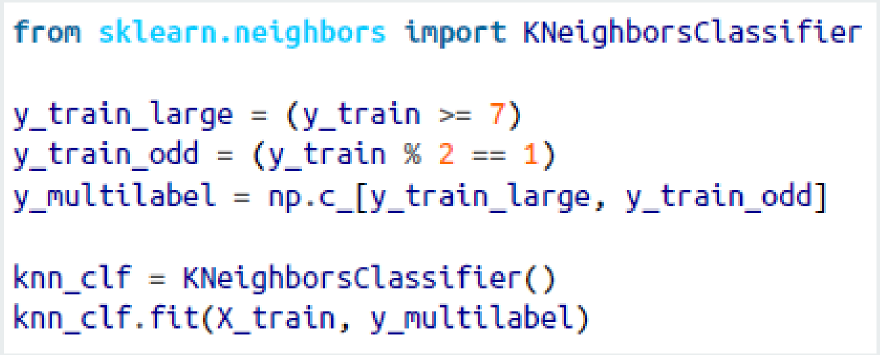
```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
```

Fill the diagonal with zeros to keep only the errors, and plot the result:

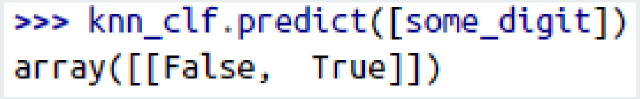
```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```

Multilabel Classification

- Large or not
- Even or not
- Use 5 to test



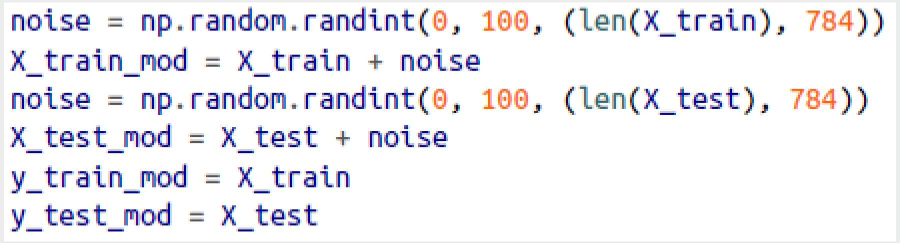
```python
from sklearn.neighbors import KNeighborsClassifier

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)
```



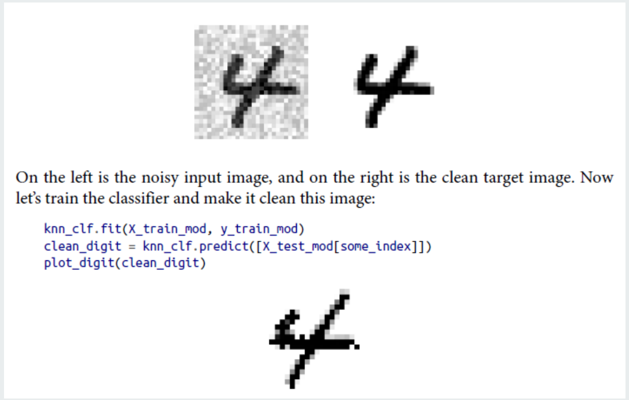
```
>>> knn_clf.predict([some_digit])
array([[False,  True]])
```

Multioutput Classification

- To illustrate this, let’s build a system that removes noise from images. It will take as input a noisy digit image, and it will (hopefully) output a clean digit image, represented as an array of pixel intensities, just like the MNIST images. Notice that the classifier’s output is multilabel (one label per pixel) and each label can have multiple values (pixel intensity ranges from 0 to 255). It is thus an example of a multioutput classification system.



```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```



On the left is the noisy input image, and on the right is the clean target image. Now let’s train the classifier and make it clean this image:

```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
```