In [1]:

```
#获取数据集
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784',version=1,as_frame=False)
mnist
```

```
  'pixel47',
  'pixel48',
  'pixel49',
  'pixel50',
  'pixel51',
  'pixel52',
  'pixel53',
  'pixel54',
  'pixel55',
  'pixel56',
  'pixel57',
  'pixel58',
  'pixel59',
  'pixel60',
  'pixel61',
  'pixel62',
  'pixel63',
  'pixel64',
  'pixel65',
  'pixel66'
```

In [2]:

```
#查看数组
X,y = mnist["data"], mnist["target"]
X.shape
```

Out[2]:

```
(70000, 784)
```

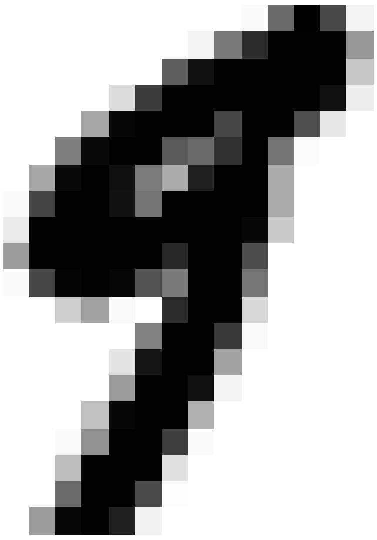
In [3]:

```
y.shape
```

Out[3]:

```
(70000,)
```

In [4]:
```
%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()
```



In [5]:
```
y[36000]
```

Out[5]:
```
'9'
```

In [6]:

```
import matplotlib as mpl

def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = mpl.cm.binary,
        interpolation="nearest")
    plt.axis("off")
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    n_rows = (len(instances) - 1) // images_per_row + 1

    # Append empty images to fill the end of the grid, if needed:
    n_empty = n_rows * images_per_row - len(instances)
    padded_instances = np.concatenate([instances, np.zeros((n_empty, size *
size))], axis=0)

    # Reshape the array so it's organized as a grid containing 28×28 images:
    image_grid = padded_instances.reshape((n_rows, images_per_row, size, size))

    big_image = image_grid.transpose(0, 2, 1, 3).reshape(n_rows * size,
    images_per_row * size)
    # Now that we have a big image, we just need to show it:
    plt.imshow(big_image, cmap = mpl.cm.binary, **options)
    plt.axis("off")

plt.figure(figsize=(9,9))
example_images = X[:100]
plot_digits(example_images, images_per_row=10)
plt.savefig("more_digits_plot")
plt.show()
```

In [7]:
```
y = y.astype(np.uint8)
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

In [8]:
```
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

In [9]:
```
y_train_5 = (y_train == 5) # True for all 5s, False for all other digits.
y_test_5 = (y_test == 5)
```

In [10]:
```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

Out[10]:
```
▾        SGDClassifier
SGDClassifier(random_state=42)
```

In [11]:
```
sgd_clf.predict([some_digit])
```

Out[11]:
```
array([False])
```

In [12]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
skfolds = StratifiedKFold(n_splits=3, random_state=42, shuffle=True)
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)) # prints 0.9502, 0.96565 and 0.96495
```

```
0.96775
0.9672
0.9601
```

In [13]:

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

Out[13]:

```
array([0.96055, 0.9585 , 0.96885])
```

In [14]:

```
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)
```

In [15]:

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

Out[15]:

```
array([0.91275, 0.90845, 0.90775])
```

In [16]:

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

In [17]:

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

Out[17]:

```
array([[53952,   627],
       [ 1615,  3806]], dtype=int64)
```

In [18]:

```
y_train_perfect_predictions = y_train_5 # pretend we reached perfection
confusion_matrix(y_train_5, y_train_perfect_predictions)
```

Out[18]:

```
array([[54579,     0],
       [    0,  5421]], dtype=int64)
```

In [19]:

```
from sklearn.metrics import precision_score, recall_score
recall_score(y_train_5, y_train_pred) # == 4344 / (4344 + 1077)
```

Out[19]:

```
0.7020844862571481
```

In [20]:

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

Out[20]:

```
0.7724781814491578
```

In [21]:

```
y_scores = sgd_clf.decision_function([some_digit])
y_scores
```

Out[21]:

```
array([-7942.70054026])
```

In [22]:

```
threshold = 0
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred
```

Out[22]:

```
array([False])
```

In [23]:

```
threshold = 200000
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred
```

Out[23]:

```
array([False])
```

In [24]:

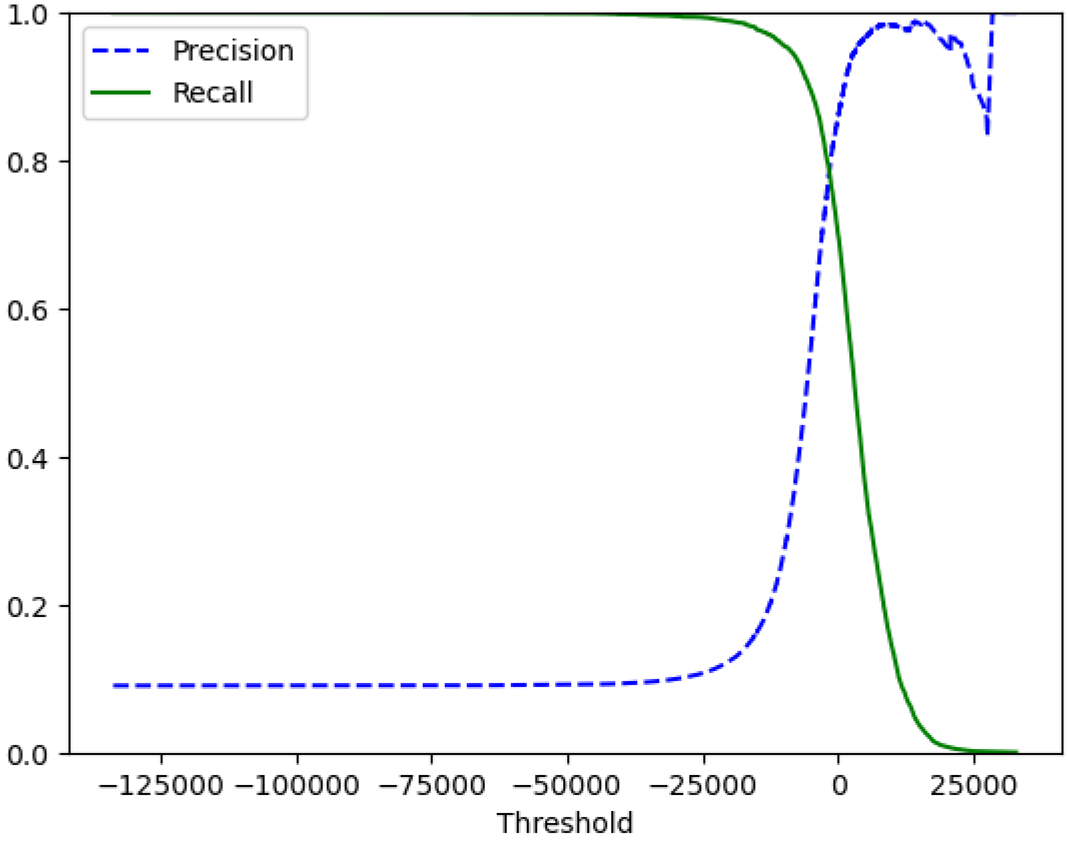
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
method="decision_function")
```

In [25]:

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

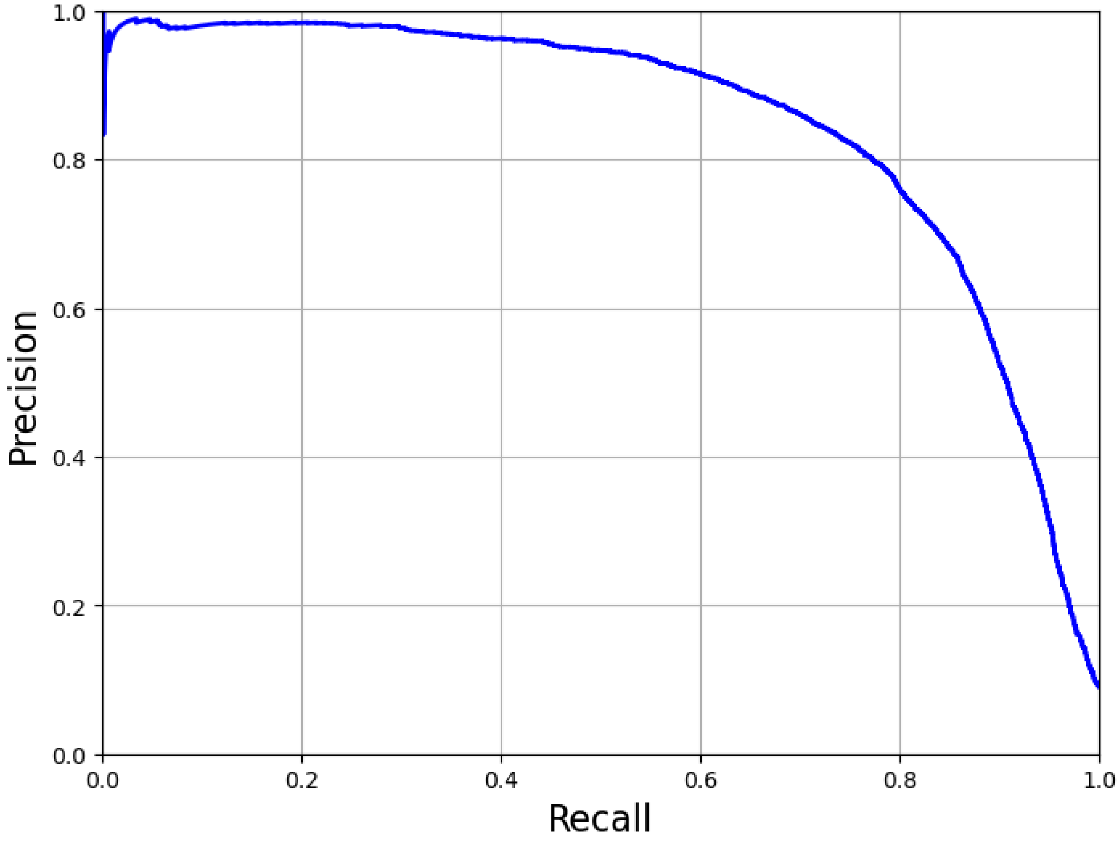
In [26]:

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")
    plt.ylim([0, 1])
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



In [27]:

```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])
    plt.grid(True)
plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.show()
```



In [28]:

```
y_train_pred_90 = (y_scores > 70000)
```

In [29]:

```
import warnings
warnings.filterwarnings("ignore")
precision_score(y_train_5, y_train_pred_90)
```

Out[29]:

```
0.0
```

In [30]:

```
recall_score(y_train_5, y_train_pred_90)
```

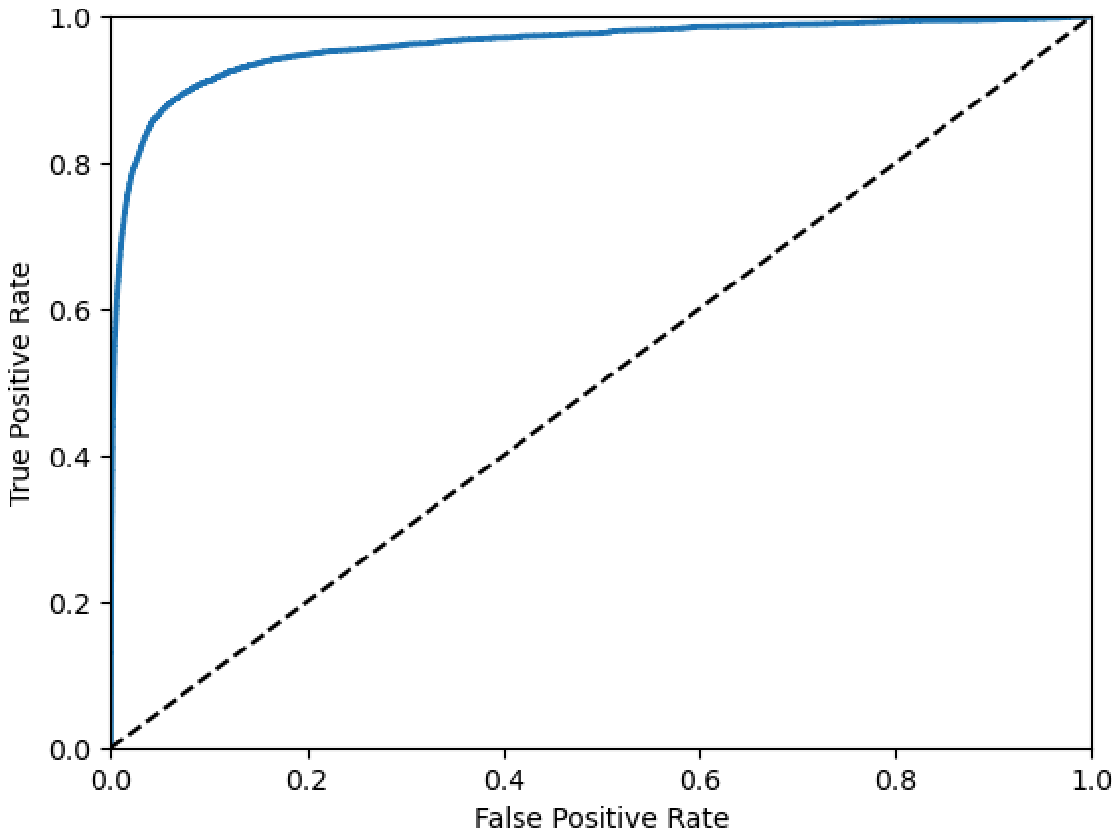
Out[30]:

```
0.0
```

In [31]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

In [32]:
```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
plot_roc_curve(fpr, tpr)
plt.show()
```



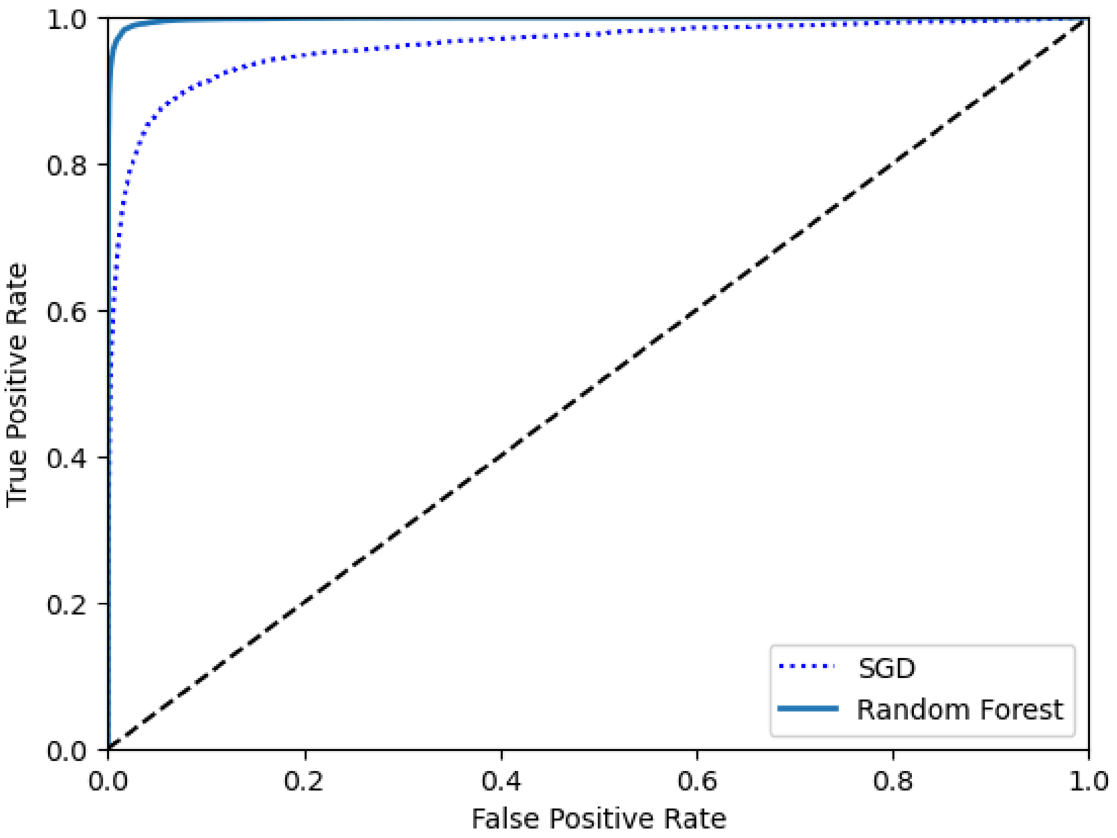
In [33]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_5, y_scores)
```

Out[33]:
```
0.9606586694924489
```

In [34]:
```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
method="predict_proba")
```

In [35]:
```
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)
```

In [36]:
```
plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right")
plt.show()
```



In [37]:
```
roc_auc_score(y_train_5, y_scores_forest)
```

Out[37]:
```
0.9984573419278522
```

In [38]:
```
sgd_clf.fit(X_train, y_train) # y_train, not y_train_5
sgd_clf.predict([some_digit])
```

Out[38]:
```
array([3], dtype=uint8)
```

In [39]:
```
some_digit_scores = sgd_clf.decision_function([some_digit])
some_digit_scores
```

Out[39]:
```
array([[-38451.98135063, -27346.42156278, -30213.6866118 ,
          2354.83581704,   -522.53770922,  -4292.47256279,
        -42491.8010252 , -13236.88812634,  -3844.91582026,
         -2470.16305866]])
```

In [40]:
```
np.argmax(some_digit_scores)
```

Out[40]:
```
3
```

In [41]:
```
sgd_clf.classes_
```

Out[41]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

In [42]:

```
sgd_clf.classes_[5]
```

Out[42]:

```
5
```

In [43]:

```
from sklearn.multiclass import OneVsOneClassifier
ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42))
ovo_clf.fit(X_train, y_train)
ovo_clf.predict([some_digit])
```

Out[43]:

```
array([4], dtype=uint8)
```

In [44]:

```
len(ovo_clf.estimators_)
```

Out[44]:

```
45
```

In [45]:

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

Out[45]:

```
array([9], dtype=uint8)
```

In [46]:

```
forest_clf.predict_proba([some_digit])
```

Out[46]:

```
array([[0.  , 0.02, 0.  , 0.  , 0.1 , 0.03, 0.02, 0.01, 0.01, 0.81]])
```

In [47]:

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

Out[47]:

```
array([0.8833, 0.8761, 0.8477])
```

In [48]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```

Out[48]:

```
array([0.90315, 0.9061 , 0.9034 ])
```

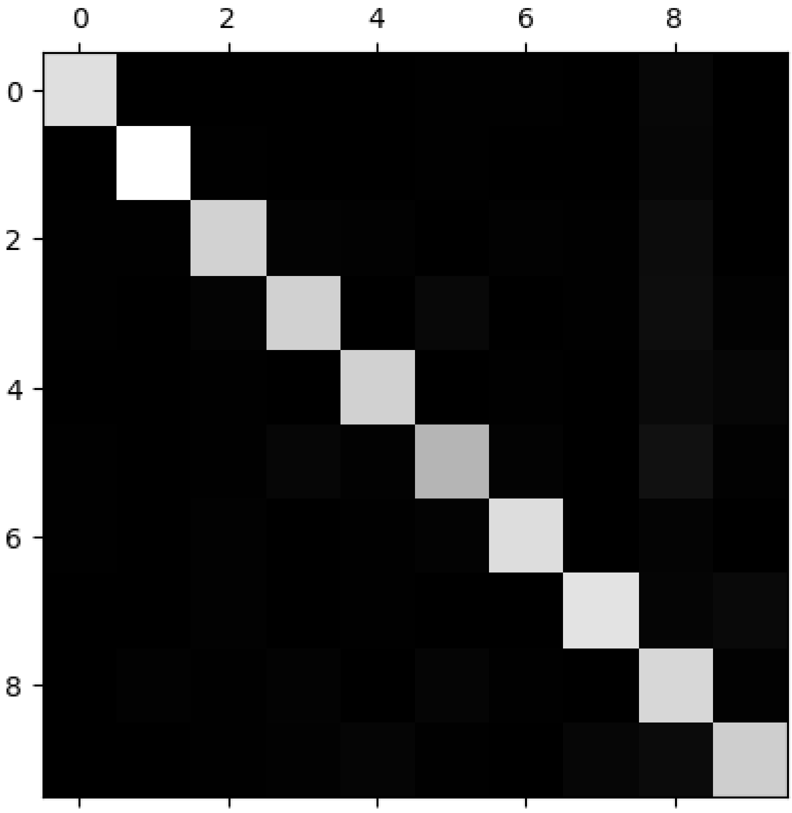
In [49]:

```
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
```

Out[49]:

```
array([[5604,    0,   16,    6,    9,   50,   35,    6,  196,    1],
       [   1, 6425,   44,   20,    4,   49,    5,    8,  173,   13],
       [  28,   28, 5284,   87,   72,   25,   69,   38,  317,   10],
       [  28,   22,  111, 5260,    2,  223,   25,   42,  350,   68],
       [   7,   12,   44,   11, 5259,   11,   43,   20,  263,  172],
       [  28,   17,   27,  155,   55, 4551,   77,   15,  428,   68],
       [  27,   18,   55,    3,   43,   95, 5556,    5,  116,    0],
       [  18,   14,   51,   22,   50,   17,    3, 5717,  144,  229],
       [  17,   66,   47,   88,    2,  131,   36,    7, 5406,   51],
       [  23,   23,   28,   63,  126,   41,    1,  171,  282, 5191]],
      dtype=int64)
```

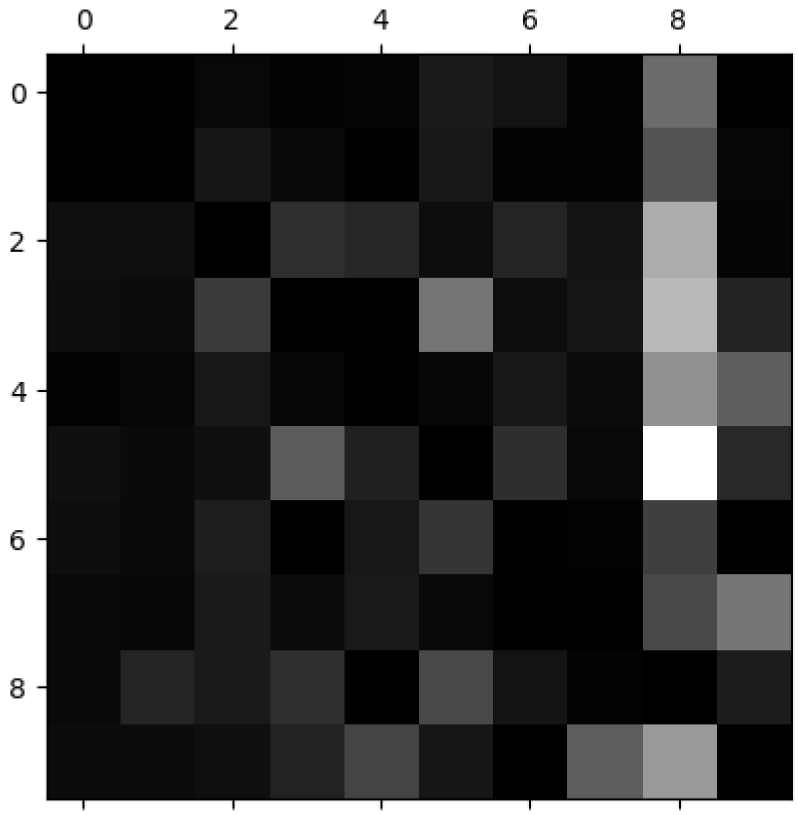
In [50]:
```
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show()
#可见分类效果非常的好
```



In [51]:
```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
```

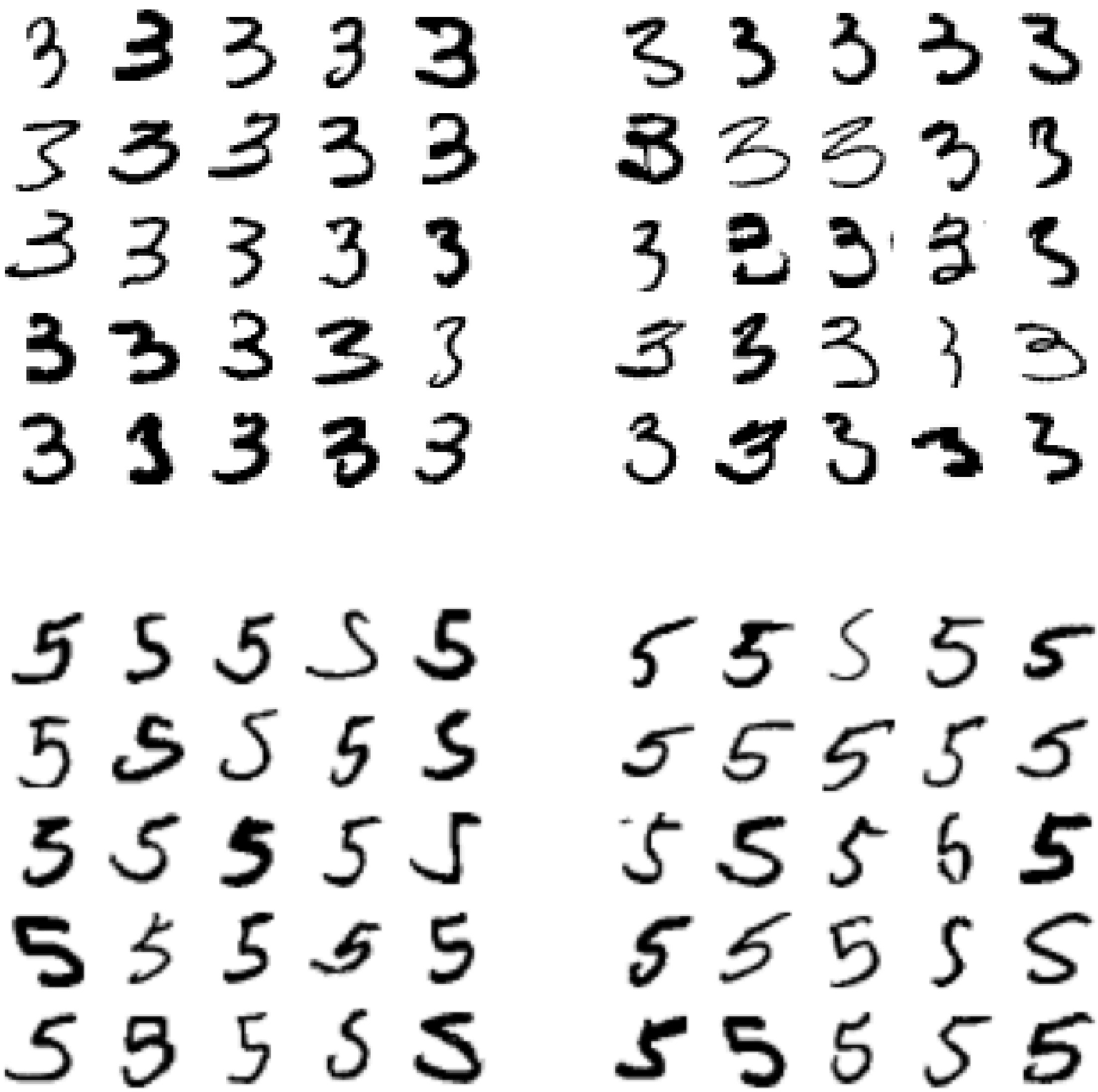
In [52]:

```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
#分析误差，其他数字被误分类成8的可能性最大
```



In [53]:

```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]
plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
plt.show()
```



In [54]:

```
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)
```

Out[54]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [55]:
```
knn_clf.predict([some_digit])
```

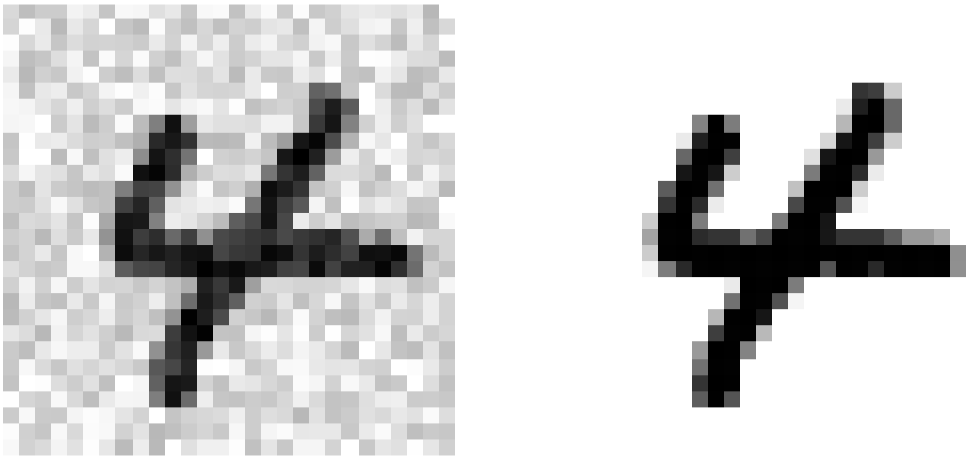
Out[55]:
```
array([[ True,  True]])
```

In [56]:
```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_train, cv=3)
f1_score(y_train, y_train_knn_pred, average="macro")
```

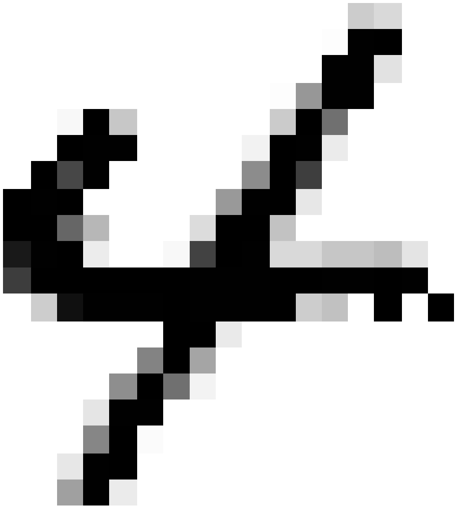
Out[56]:
```
0.96801255083268
```

In [57]:
```
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
```

In [58]:
```
some_index = 5500
plt.subplot(121); plot_digit(X_test_mod[some_index])
plt.subplot(122); plot_digit(y_test_mod[some_index])
plt.show()
```



In [59]:
```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
```



In [ ]:
```
```