03_classification

September 29, 2019

[1]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "classification"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)


def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    os.makedirs(os.path.join(IMAGES_PATH), exist_ok=True)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
    print('Figure saved as', fig_id + '.png')

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

[2]:
```
'''
Fetch the MNIST dataset which is what we will be working on in this chapter
the old method of fetching data is deprecated so you must use new method,
but it returns the data unsorted which is fine, but the function below
will ensure results are same as in book
'''
def sort_by_target(mnist):
    reorder_train = np.array(sorted([(target, i) for i, target in␣
 ↪enumerate(mnist.target[:60000])]))[:, 1]
    reorder_test = np.array(sorted([(target, i) for i, target in␣
 ↪enumerate(mnist.target[60000:])]))[:, 1]
    mnist.data[:60000] = mnist.data[reorder_train]
    mnist.target[:60000] = mnist.target[reorder_train]
    mnist.data[60000:] = mnist.data[reorder_test + 60000]
    mnist.target[60000:] = mnist.target[reorder_test + 60000]
```

[3]:
```
try:
    from sklearn.datasets import fetch_openml
    mnist = fetch_openml('mnist_784', version=1, cache=True)
    mnist.target = mnist.target.astype(np.int8) # fetch_openml() returns␣
 ↪targets as strings
    sort_by_target(mnist) # fetch_openml() returns an unsorted dataset
except ImportError:
    from sklearn.datasets import fetch_mldata
    mnist = fetch_mldata('MNIST original')
mnist["data"], mnist["target"]
```

[3]:
```
(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.]]),
 array([0, 0, 0, ..., 9, 9, 9], dtype=int8))
```

[4]:
```
mnist.data.shape
```

[4]:
```
(70000, 784)
```

[5]:
```
X, y = mnist['data'], mnist['target']
X.shape
```

[5]:
```
(70000, 784)
```

[6]:
```
y.shape
```

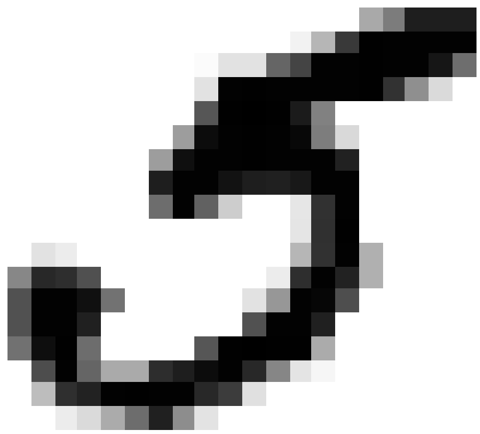
[6]:
```
(70000,)
```

[7]:
```
#lets take a look at a few digits
some_digit = X[36000]
```

```
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap = mpl.cm.binary, interpolation = 'nearest')
save_fig('some_digit_plot')
plt.axis('off')
```

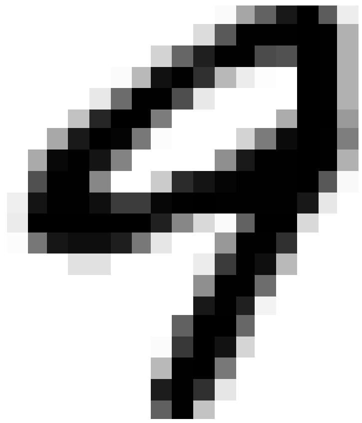
```
Saving figure some_digit_plot
Figure saved as some_digit_plot.png
```

[7]:
```
(-0.5, 27.5, 27.5, -0.5)
```



[8]:
```
some_other_digit = X[69999]
some_other_digit_img = some_other_digit.reshape(28, 28)
plt.imshow(some_other_digit_img, cmap = mpl.cm.binary, interpolation =␣
 ↪'nearest')
plt.axis('off')
```

[8]:
```
(-0.5, 27.5, 27.5, -0.5)
```



[9]:
```
def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = mpl.cm.binary,
               interpolation = 'nearest')
    plt.axis('off')
```

[10]:
```
#EXTRA
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size, size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = mpl.cm.binary, **options)
    plt.axis('off')
```

[11]:
```
plt.figure(figsize = (9,9))
example_images = np.r_[X[:12000:600], X[13000:30600:600], X[30600:60000:590]]
plot_digits(example_images, images_per_row=10)
save_fig('more_digits_plot')
plt.show()
```

```
Saving figure more_digits_plot
```

Figure saved as more_digits_plot.png

[12]:
```
'''create test set and train set and set aside, however,
this dataset is already split (1st 60,000 images are for training),
and the last 10,000 are for testing'''

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

[13]:
```
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

[14]:
```
'''BINARY CLASSIFIER'''
y_train_5 = (y_train == 5) #This is true for all 5's, false for all other␣
↪digits.
y_test_5 = (y_test == 5)
```

[15]:
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(max_iter=5, tol=-np.infty, random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

[15]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=5,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=42, shuffle=True, tol=-inf, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

[16]:
```
sgd_clf.predict([some_digit])
```

[16]:
```
array([ True])
```

[17]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring='accuracy')
```

[17]:
```
array([0.96225, 0.9645 , 0.94765])
```

[18]:
```
#sometimes you need more control over the cross-validation process so you may␣
 ↪want to
#implement it yourself like below

from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

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))
```

```
0.96225
0.9645
0.94765
```

[19]:
```
'''95% accuracy above, but accuracy is not always a good way of measuring
performance in classifiers, specially if the data is skewed'''

from sklearn.model_selection import cross_val_predict
```

```
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

[20]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5, y_train_pred)
```

[20]:
```
array([[53417,  1162],
       [ 1350,  4071]], dtype=int64)
```

[21]:
```
y_train_perfect_predictions = y_train_5
confusion_matrix(y_train_5, y_train_perfect_predictions)
```

[21]:
```
array([[54579,     0],
       [    0,  5421]], dtype=int64)
```

[22]:
```
'''PRECISION AND RECALL'''
from sklearn.metrics import precision_score, recall_score
precision_score(y_train_5, y_train_pred)# == 4344 / (4344 + 1307)
```

[22]:
```
0.7779476399770686
```

[23]:
```
recall_score(y_train_5, y_train_pred)# == 4344 / (4344 + 1077)
```

[23]:
```
0.7509684560044272
```

[24]:
```
from sklearn.metrics import f1_score
f1_score(y_train_5, y_train_pred)
```

[24]:
```
0.7642200112633752
```

[25]:
```
y_scores = sgd_clf.decision_function([some_digit])
y_scores
```

[25]:
```
array([150526.40944343])
```

[26]:
```
threshold = 0
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred
```

[26]:
```
array([ True])
```

[27]:
```
#to decide which threshold to use, you need to get scores of all the cross val␣
↪results but specify you want decision scores instead of predictions
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method = 'decision_function')
```

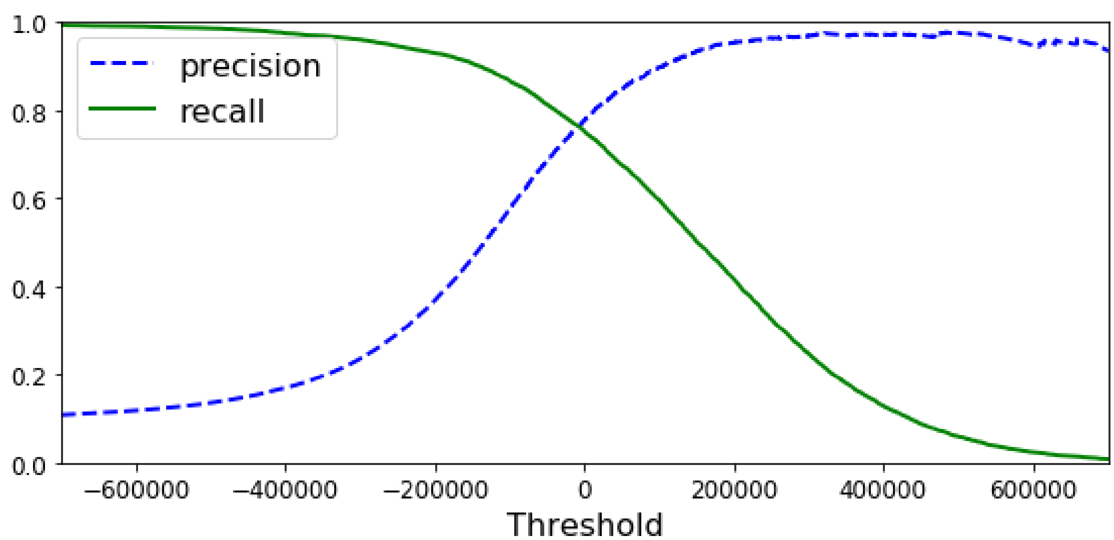
[28]:
```
#now with above scores, you can compute precision and recall for all possible␣
↪threshold using the precision_recal_curve() function
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

[29]:
```
#now, you can plot precision and recall as functions of the threshold value
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], 'b--', label = 'precision',␣
↪linewidth=2)
    plt.plot(thresholds, recalls[:-1], 'g-', label='recall', linewidth=2)
```

```
    plt.xlabel('Threshold', fontsize=16)
    plt.legend(loc='upper left', fontsize=16)
    plt.ylim([0,1])

plt.figure(figsize=(8,4))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.xlim([-700000, 700000])
save_fig('precision_recall_vs_threshold_plot')
plt.show()
```

```
Saving figure precision_recall_vs_threshold_plot
Figure saved as precision_recall_vs_threshold_plot.png
```

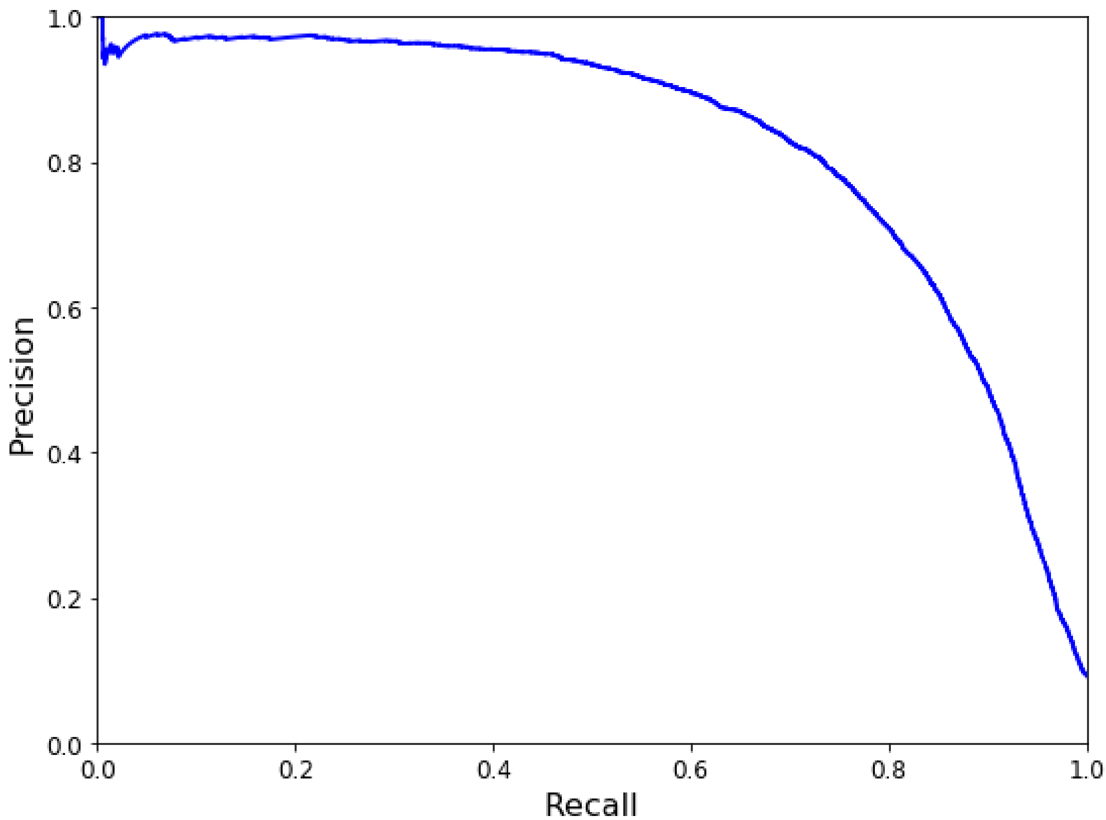


[30]:

```
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.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
save_fig("precision_vs_recall_plot")
plt.show()
```

```
Saving figure precision_vs_recall_plot
Figure saved as precision_vs_recall_plot.png
```



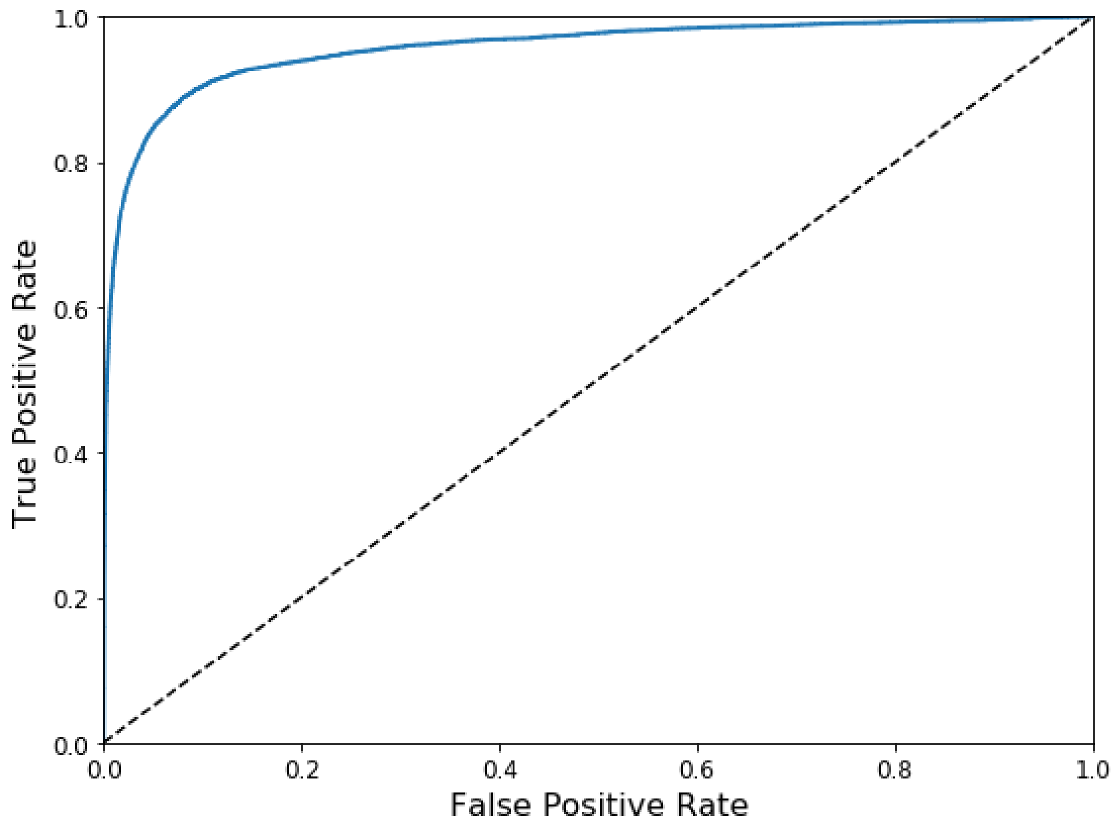
[31]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

[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', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)

plt.figure(figsize=(8,6))
plot_roc_curve(fpr, tpr)
save_fig('roc_curve_plot')
plt.show
```

```
Saving figure roc_curve_plot
Figure saved as roc_curve_plot.png
```

[32]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[33]:
```python
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

[33]: 0.9562435587387078

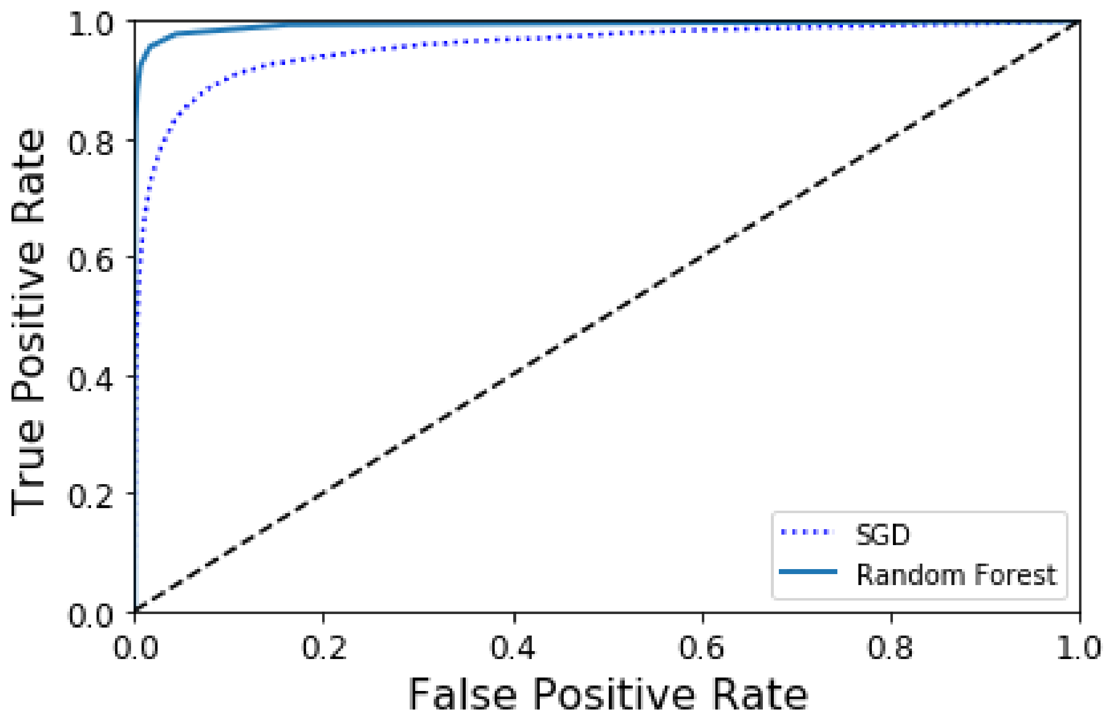
[34]:
```python
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators=10, random_state = 42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method = 'predict_proba')
```

[35]:
```python
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)
```

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

```
Saving figure roc_curve_comparison_plot
Figure saved as roc_curve_comparison_plot.png
```



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

[37]: 0.9931243366003829

[38]:
```
y_train_pred_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3)
precision_score(y_train_5, y_train_pred_forest)
```

[38]: 0.9852973447443494

[39]:
```
recall_score(y_train_5, y_train_pred_forest)
```

[39]: 0.8282604685482383

[40]:
```
'''MULTICLASS CLASSIFICATIONS'''
sgd_clf.fit(X_train, y_train) #y_train not y_train_5
sgd_clf.predict([some_digit])
```

[40]: array([5], dtype=int8)

[41]:
```
some_digit_scores = sgd_clf.decision_function([some_digit])
some_digit_scores
```

[41]:
```
array([[-152619.46799791, -441052.22074349, -249930.3138537 ,
        -237258.35168498, -447251.81933158,  120565.05820991,
        -834139.15404835, -188142.48490477, -555223.79499145,
        -536978.92518594]])
```

[42]:
```
np.argmax(some_digit_scores)
```

[42]: 5

[43]:
```
sgd_clf.classes_
```

[43]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)

[44]:
```
sgd_clf.classes_[5]
```

[44]: 5

[45]:
```
from sklearn.multiclass import OneVsOneClassifier
ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42))
ovo_clf.fit(X_train, y_train)
ovo_clf.predict([some_digit])
```

[45]: array([5], dtype=int8)

[46]:
```
len(ovo_clf.estimators_)
```

[46]: 45

[47]:
```
forest_clf.fit(X_train, y_train)
forest_clf.predict([some_digit])
```

[47]: array([5], dtype=int8)

[48]:
```
forest_clf.predict_proba([some_digit])
```

[48]: array([[0.1, 0. , 0. , 0.1, 0. , 0.8, 0. , 0. , 0. , 0.]])

[49]:
```
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring='accuracy')
```

[49]: array([0.84993001, 0.81769088, 0.84707706])

[50]:
```
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')
```

[50]: array([0.91211758, 0.9099955 , 0.90643597])

[51]:
```
'''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
```

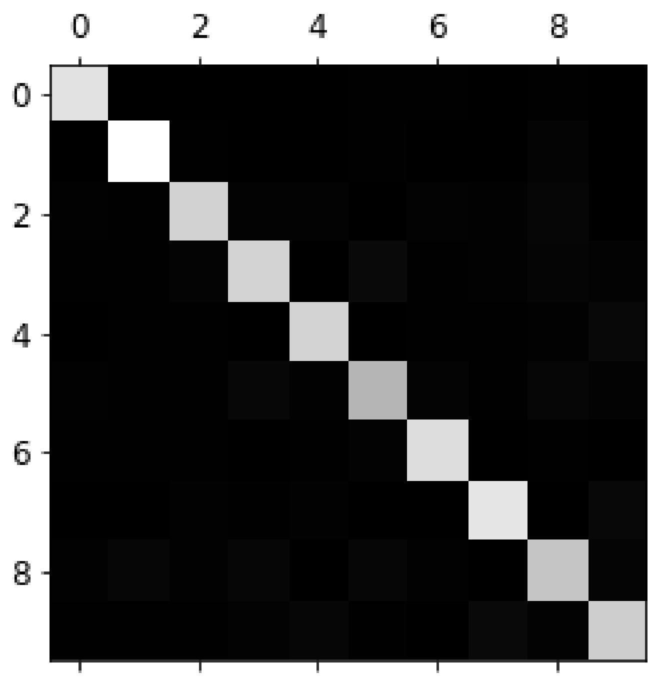
[51]:
```
array([[5749,    4,   22,   11,   11,   40,   36,   11,   36,    3],
       [   2, 6490,   43,   24,    6,   41,    8,   12,  107,    9],
       [  53,   42, 5330,   99,   87,   24,   89,   58,  159,   17],
       [  46,   41,  126, 5361,    1,  241,   34,   59,  129,   93],
       [  20,   30,   35,   10, 5369,    8,   48,   38,   76,  208],
       [  73,   45,   30,  194,   64, 4614,  106,   30,  170,   95],
       [  41,   30,   46,    2,   44,   91, 5611,    9,   43,    1],
       [  26,   18,   73,   30,   52,   11,    4, 5823,   14,  214],
       [  63,  159,   69,  168,   15,  172,   54,   26, 4997,  128],
       [  39,   39,   27,   90,  177,   40,    2,  230,   78, 5227]],
```

```
      dtype=int64)
```

[52]:
```python
plt.matshow(conf_mx, cmap=plt.cm.gray)
save_fig('confusion_matrix_plot', tight_layout=False)
plt.show
```

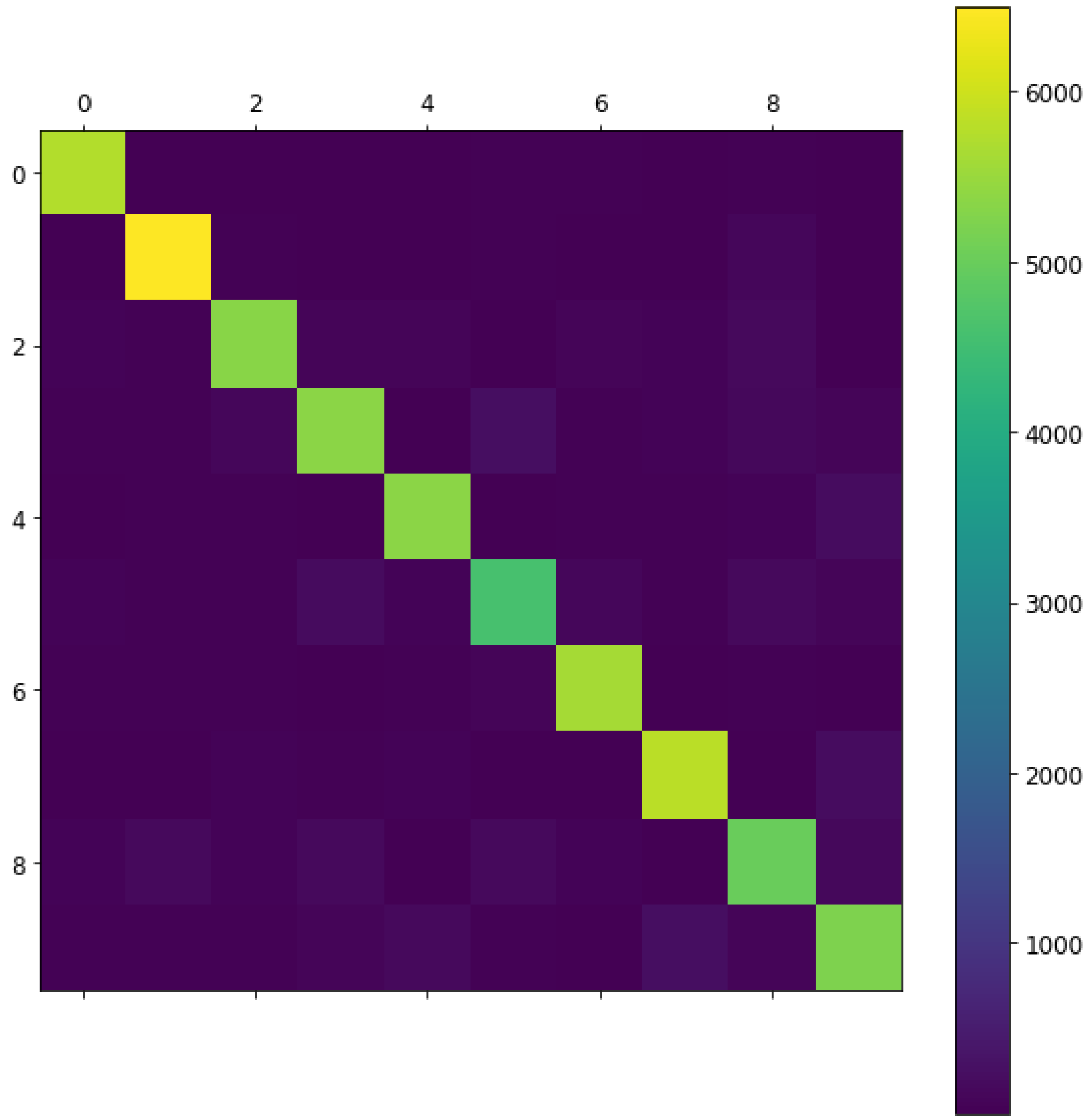
```
Saving figure confusion_matrix_plot
Figure saved as confusion_matrix_plot.png
```

[52]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[53]:
```python
def plot_confusion_matrix(matrix):
    """If you prefer color and a colorbar"""
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)
plot_confusion_matrix(conf_mx)
save_fig('conf_matrix_colored')
```

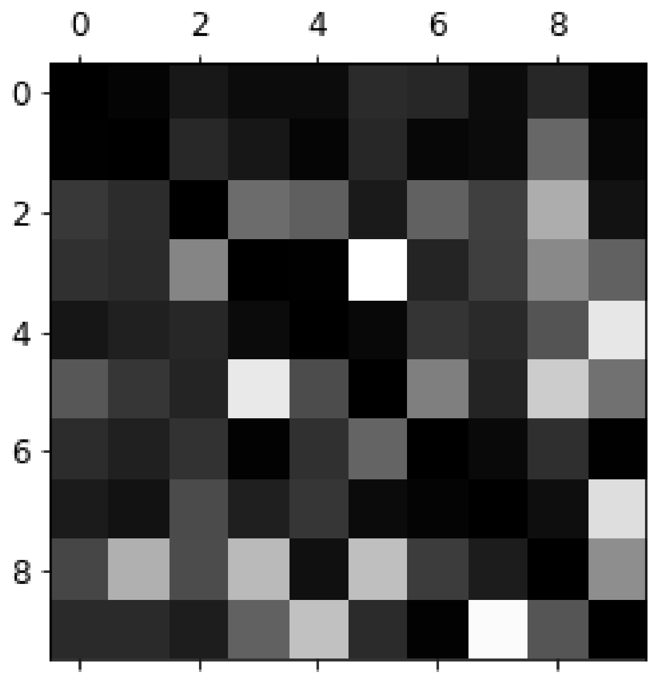
```
Saving figure conf_matrix_colored
Figure saved as conf_matrix_colored.png
```



[54]:
```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
```

[55]:
```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
save_fig('confusion_matrix_errors_plot', tight_layout=False)
plt.show()
```

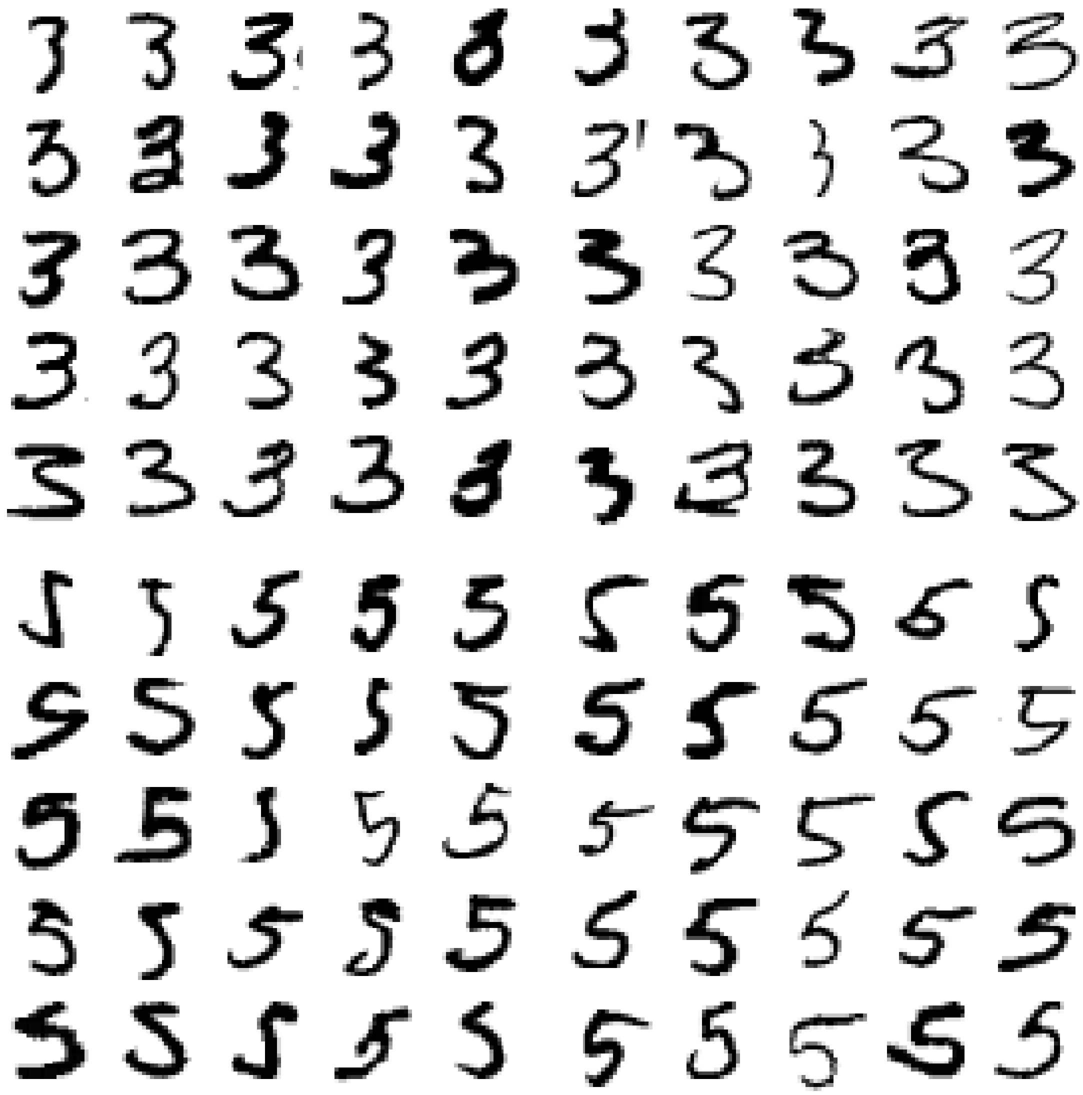
```
Saving figure confusion_matrix_errors_plot
Figure saved as confusion_matrix_errors_plot.png
```



[56]:
```
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)
save_fig("error_analysis_digits_plot")
plt.show()
```

```
Saving figure error_analysis_digits_plot
Figure saved as error_analysis_digits_plot.png
```



[57]:

```python
'''MULTILABEL CLASSIFICATION'''
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)
```

[57]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[58]:
```
knn_clf.predict([some_digit])
```

[58]:
```
array([[False,  True]])
```

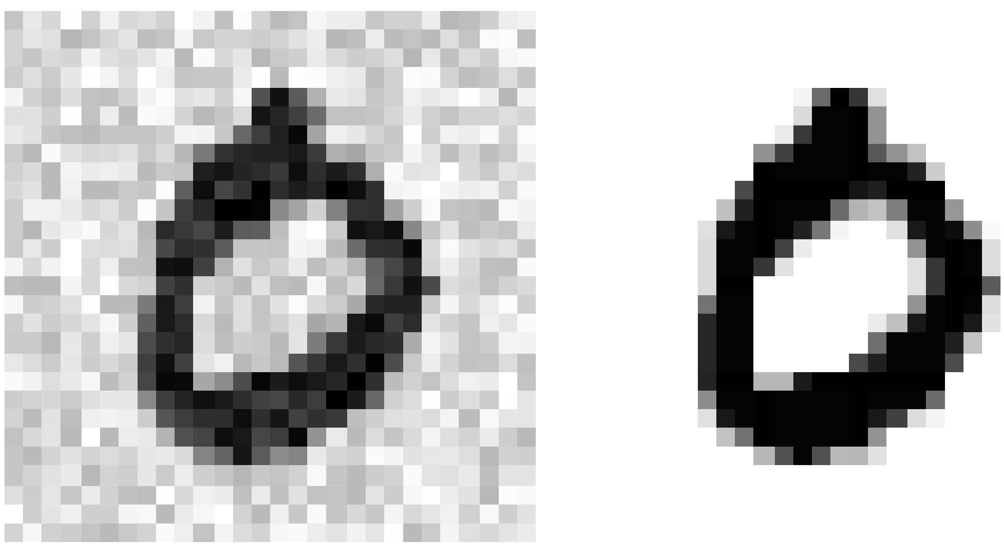
[59]:
```
'''THIS CODE MAY TAKE A VERY LONG TIME'''
#y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
#f1_score(y_multilabel, y_train_knn_pred, average='macro')
```

[59]:
```
'THIS CODE MAY TAKE A VERY LONG TIME'
```

[62]:
```
'''MULTIOUTPUT CLASSIFICATION'''
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
```

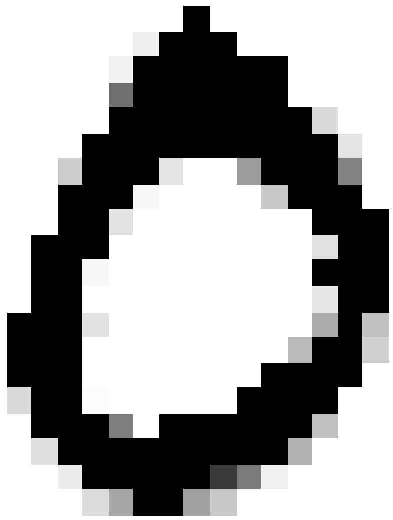
[67]:
```
some_index = 0
plt.subplot(121); plot_digit(X_test_mod[some_index])
plt.subplot(122); plot_digit(y_test_mod[some_index])
save_fig('noisy_digit_example_plot')
plt.show()
```

```
Saving figure noisy_digit_example_plot
Figure saved as noisy_digit_example_plot.png
```



[68]:
```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
save_fig('cleaned_digit_example_plot')
```

```
Saving figure cleaned_digit_example_plot
Figure saved as cleaned_digit_example_plot.png
```



[ ]: