Classification using MNIST.ipynb

文件 修改 视图 插入 代码执行程序 工具 帮助 已保存所有更改

评论 分享

+ 代码 + 文本

RAM 磁盘 修改

Chapter 3 – Classification

This notebook contains all the sample code and solutions to the exercises in chapter 3.

Open in Colab Open in Kaggle

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20.

[1]

```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Is this notebook running on Colab or Kaggle?
IS_COLAB = "google.colab" in sys.modules
IS_KAGGLE = "kaggle_secrets" in sys.modules

# Scikit-Learn ≥0.20 is required
```

```
# 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)
os.makedirs(IMAGES_PATH, exist_ok=True)

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

MNIST

Warning: since Scikit-Learn 0.24, `fetch_openml()` returns a Pandas `DataFrame` by default. To avoid this and keep the same code as in the book, we use `as_frame=False`.

[2]

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

```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

[3]

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

```
(70000, 784)
```

[4]

```
y.shape
```

```
(70000,)
```

[5]

```
28 * 28
```

```
784
```

```
%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")

save_fig("some_digit_plot")
plt.show()
```

Saving figure some_digit_plot

+ 代码 + 文本

```
[7] y[0]
```

'5'

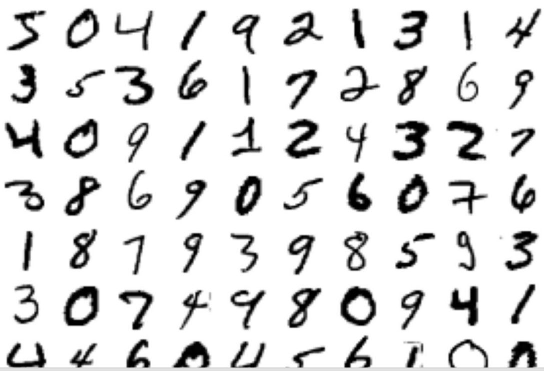
```
[8] y = y.astype(np.uint8)
```

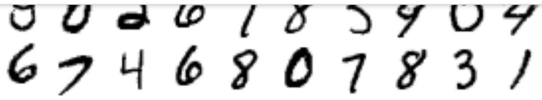
```
[9] def plot_digit(data):
        image = data.reshape(28, 28)
        plt.imshow(image, cmap = mpl.cm.binary,
                   interpolation="nearest")
        plt.axis("off")
```

```
# EXTRA
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    # This is equivalent to n_rows = ceil(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))

    # Combine axes 0 and 2 (vertical image grid axis, and vertical image axis),
```

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

Saving figure more_digits_plot





```
[12] y[0]
```

```
5
```

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

Training a Binary Classifier

```
[14] y_train_5 = (y_train == 5)
     y_test_5 = (y_test == 5)
```

Note: some hyperparameters will have a different defaut value in future versions of Scikit-Learn, such as `max_iter` and `tol`. To be future-proof, we explicitly set these hyperparameters to their future default values. For simplicity, this is not shown in the book.

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

```
     sgd_clf.fit(X_train, y_train_5)
```

```
SGDClassifier(random_state=42)
```

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

```
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")
```

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

Performance Measures

Measuring Accuracy Using Cross-Validation

```
[18] from sklearn.model_selection import StratifiedKFold
     from sklearn.base import clone

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

     for train_index, test_index in skfolds.split(X_train, y_train_5):
```

```
         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))
```

```
0.9669
0.91625
0.96785
```

Note: `shuffle=True` was omitted by mistake in previous releases of the book.

```
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)
```

```
[20] 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])
```

+ 代码 + 文本

Confusion Matrix

```
[21] from sklearn.model_selection import cross_val_predict

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

```
[22] from sklearn.metrics import confusion_matrix

confusion_matrix(y_train_5, y_train_pred)
```

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

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

```
array([[54579,     0],
       [    0,  5421]])
```

Precision and Recall

```
[24] from sklearn.metrics import precision_score, recall_score
```

+ 代码 + 文本

```
0.8370879772350012
```

```
[25] cm = confusion_matrix(y_train_5, y_train_pred)
cm[1, 1] / (cm[0, 1] + cm[1, 1])
```

```
0.8370879772350012
```

```
[26] recall_score(y_train_5, y_train_pred)
```

```
0.6511713705958311
```

```
[27] cm[1, 1] / (cm[1, 0] + cm[1, 1])
```

```
0.6511713705958311
```

```
[28] from sklearn.metrics import f1_score

f1_score(y_train_5, y_train_pred)
```

```
0.7325171197343846
```

```
[29] cm[1, 1] / (cm[1, 1] + (cm[1, 0] + cm[0, 1]) / 2)
```

```
0.7325171197343847
```

Precision/Recall Trade-off

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

```
array([2164.22030239])
```

```
[31] threshold = 0
y_some_digit_pred = (y_scores > threshold)
```

```
[32] y_some_digit_pred
```

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

```
[33] threshold = 8000
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred
```

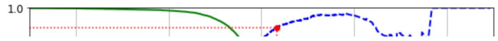
```
array([False])
```

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

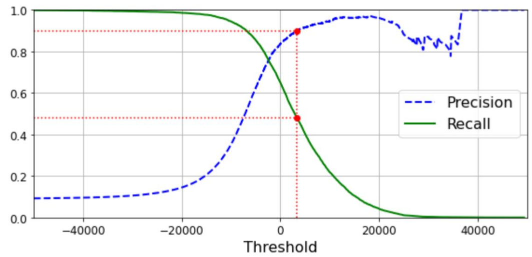
```
[35] from sklearn.metrics import precision_recall_curve
```

```python
[36] 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.legend(loc="center right", fontsize=16)  # Not shown in the book
         plt.xlabel("Threshold", fontsize=16)         # Not shown
         plt.grid(True)                               # Not shown
         plt.axis([-50000, 50000, 0, 1])              # Not shown



     recall_90_precision = recalls[np.argmax(precisions >= 0.90)]
     threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]


     plt.figure(figsize=(8, 4))                                                     # No
     plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
     plt.plot([threshold_90_precision, threshold_90_precision], [0., 0.9], "r:")    # Not shown
     plt.plot([-50000, threshold_90_precision], [0.9, 0.9], "r:")                    # Not shown
     plt.plot([-50000, threshold_90_precision], [recall_90_precision, recall_90_precision], "r:")# Not shown
     plt.plot([threshold_90_precision], [0.9], "ro")                                  # Not shown
     plt.plot([threshold_90_precision], [recall_90_precision], "ro")                  # Not shown
     save_fig("precision_recall_vs_threshold_plot")                                   # Not shown
     plt.show()
```

Saving figure precision_recall_vs_threshold_plot



Saving figure precision_recall_vs_threshold_plot



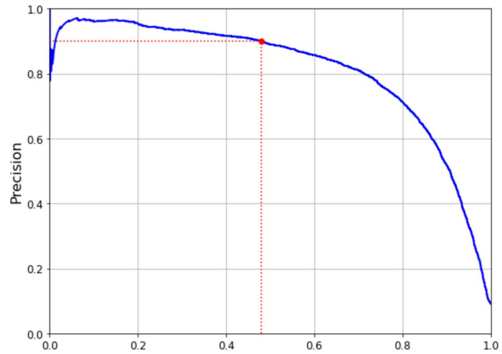

```python
[37] (y_train_pred == (y_scores > 0)).all()
```

True

```python
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)
```

```python
plt.plot([recall_90_precision, recall_90_precision], [0., 0.9], "r:")
plt.plot([0.0, recall_90_precision], [0.9, 0.9], "r:")
plt.plot([recall_90_precision], [0.9], "ro")
save_fig("precision_vs_recall_plot")
plt.show()
```

Saving figure precision_vs_recall_plot



```
[39] threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
```

```
[40] threshold_90_precision
```

```
3370.0194991439557
```

```
[41] y_train_pred_90 = (y_scores >= threshold_90_precision)
```

```
[42] precision_score(y_train_5, y_train_pred_90)
```

```
0.9000345901072293
```

```
[43] recall_score(y_train_5, y_train_pred_90)
```

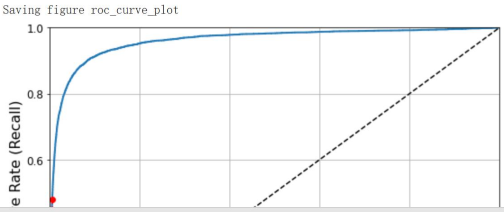
```
0.4799852425751706
```

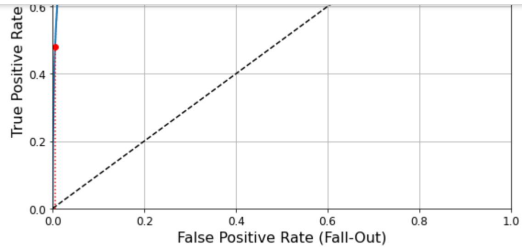
The ROC Curve

```
[44] from sklearn.metrics import roc_curve

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

```
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
    plt.axis([0, 1, 0, 1])                                    # Not shown in the book
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16) # Not shown
    plt.ylabel('True Positive Rate (Recall)', fontsize=16)    # Not shown
    plt.grid(True)                                            # Not shown

plt.figure(figsize=(8, 6))                                    # Not shown
plot_roc_curve(fpr, tpr)
fpr_90 = fpr[np.argmax(tpr >= recall_90_precision)]           # Not shown
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:")  # Not shown
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:")  # Not shown
plt.plot([fpr_90], [recall_90_precision], "ro")               # Not shown
save_fig("roc_curve_plot")                                    # Not shown
plt.show()
```

```
Saving figure roc_curve_plot
```






```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

```
0.9604938554008616
```

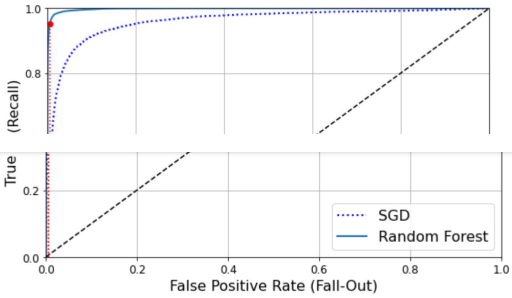
Note: we set `n_estimators=100` to be future-proof since this will be the default value in Scikit-Learn 0.22.

```
[47] 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,
```

```
[48] 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)
```

```
recall_for_forest = tpr_forest[np.argmax(fpr_forest >= fpr_90)]

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:")
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:")
plt.plot([fpr_90], [recall_90_precision], "ro")
plt.plot([fpr_90, fpr_90], [0., recall_for_forest], "r:")
plt.plot([fpr_90], [recall_for_forest], "ro")
plt.grid(True)
plt.legend(loc="lower right", fontsize=16)
save_fig("roc_curve_comparison_plot")
plt.show()
```

```
Saving figure roc_curve_comparison_plot
```




```
roc_auc_score(y_train_5, y_scores_forest)
```

```
0.9983436731328145
```

```
[51] 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)
```

```
0.9905083315756169
```

```
[52] recall_score(y_train_5, y_train_pred_forest)
```

```
0.8662608374838591
```

Multiclass Classification

```
[53] from sklearn.svm import SVC

     svm_clf = SVC(gamma="auto", random_state=42)
     svm_clf.fit(X_train[:1000], y_train[:1000]) # y_train, not y_train_5
     svm_clf.predict([some_digit])
```

```
array([5], dtype=uint8)
```

```
[54] some_digit_scores = svm_clf.decision_function([some_digit])
     some_digit_scores
```

```
array([[ 2.81585438,  7.09167958,  3.82972099,  0.79365551,  5.8885703 ,
         9.29718395,  1.79862509,  8.10392157, -0.228207  ,  4.83753243]])
```

```
[55] np.argmax(some_digit_scores)
```

```
5
```

```
[56] svm_clf.classes_
```

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

```
[57] svm_clf.classes_[5]
```

```
5
```