Classification

May 3, 2021

1 Classification

1.1 MNIST

Fetching the MNIST dataset

[1]:
```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1)
mnist.keys()
```

[1]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names',
'target_names', 'DESCR', 'details', 'url'])
```

Analyzing the dataset

[2]:
```
import numpy as np

X, y = mnist["data"], mnist["target"]
y = y.astype(np.uint8)
```

[3]:
```
print(mnist.data, end="\n \n")
print(mnist.target, end="\n \n")
print(mnist.feature_names, end="\n \n")
print(mnist.target_names, end="\n \n")
print(mnist.DESCR, end="\n \n")
print(mnist.details, end="\n \n")
print(mnist.url, end="\n \n")

print(mnist.data.shape)
print(mnist.target.shape)
```

```
[[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.]]

['5' '0' '4' … '4' '5' '6']

['pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6', 'pixel7', 'pixel8',
'pixel9', 'pixel10', 'pixel11', 'pixel12', 'pixel13', 'pixel14', 'pixel15',
'pixel16', 'pixel17', 'pixel18', 'pixel19', 'pixel20', 'pixel21', 'pixel22',
'pixel23', 'pixel24', 'pixel25', 'pixel26', 'pixel27', 'pixel28', 'pixel29',
'pixel30', 'pixel31', 'pixel32', 'pixel33', 'pixel34', 'pixel35', 'pixel36',
'pixel37', 'pixel38', 'pixel39', 'pixel40', 'pixel41', 'pixel42', 'pixel43',
'pixel44', 'pixel45', 'pixel46', 'pixel47', 'pixel48', 'pixel49', 'pixel50',
'pixel51', 'pixel52', 'pixel53', 'pixel54', 'pixel55', 'pixel56', 'pixel57',
'pixel58', 'pixel59', 'pixel60', 'pixel61', 'pixel62', 'pixel63', 'pixel64',
'pixel65', 'pixel66', 'pixel67', 'pixel68', 'pixel69', 'pixel70', 'pixel71',
'pixel72', 'pixel73', 'pixel74', 'pixel75', 'pixel76', 'pixel77', 'pixel78',
'pixel79', 'pixel80', 'pixel81', 'pixel82', 'pixel83', 'pixel84', 'pixel85',
'pixel86', 'pixel87', 'pixel88', 'pixel89', 'pixel90', 'pixel91', 'pixel92',
'pixel93', 'pixel94', 'pixel95', 'pixel96', 'pixel97', 'pixel98', 'pixel99',
'pixel100', 'pixel101', 'pixel102', 'pixel103', 'pixel104', 'pixel105',
'pixel106', 'pixel107', 'pixel108', 'pixel109', 'pixel110', 'pixel111',
'pixel112', 'pixel113', 'pixel114', 'pixel115', 'pixel116', 'pixel117',
'pixel118', 'pixel119', 'pixel120', 'pixel121', 'pixel122', 'pixel123',
'pixel124', 'pixel125', 'pixel126', 'pixel127', 'pixel128', 'pixel129',
'pixel130', 'pixel131', 'pixel132', 'pixel133', 'pixel134', 'pixel135',
'pixel136', 'pixel137', 'pixel138', 'pixel139', 'pixel140', 'pixel141',
'pixel142', 'pixel143', 'pixel144', 'pixel145', 'pixel146', 'pixel147',
'pixel148', 'pixel149', 'pixel150', 'pixel151', 'pixel152', 'pixel153',
'pixel154', 'pixel155', 'pixel156', 'pixel157', 'pixel158', 'pixel159',
'pixel160', 'pixel161', 'pixel162', 'pixel163', 'pixel164', 'pixel165',
'pixel166', 'pixel167', 'pixel168', 'pixel169', 'pixel170', 'pixel171',
'pixel172', 'pixel173', 'pixel174', 'pixel175', 'pixel176', 'pixel177',
'pixel178', 'pixel179', 'pixel180', 'pixel181', 'pixel182', 'pixel183',
'pixel184', 'pixel185', 'pixel186', 'pixel187', 'pixel188', 'pixel189',
'pixel190', 'pixel191', 'pixel192', 'pixel193', 'pixel194', 'pixel195',
'pixel196', 'pixel197', 'pixel198', 'pixel199', 'pixel200', 'pixel201',
'pixel202', 'pixel203', 'pixel204', 'pixel205', 'pixel206', 'pixel207',
'pixel208', 'pixel209', 'pixel210', 'pixel211', 'pixel212', 'pixel213',
'pixel214', 'pixel215', 'pixel216', 'pixel217', 'pixel218', 'pixel219',
'pixel220', 'pixel221', 'pixel222', 'pixel223', 'pixel224', 'pixel225',
'pixel226', 'pixel227', 'pixel228', 'pixel229', 'pixel230', 'pixel231',
'pixel232', 'pixel233', 'pixel234', 'pixel235', 'pixel236', 'pixel237',
'pixel238', 'pixel239', 'pixel240', 'pixel241', 'pixel242', 'pixel243',
'pixel244', 'pixel245', 'pixel246', 'pixel247', 'pixel248', 'pixel249',
'pixel250', 'pixel251', 'pixel252', 'pixel253', 'pixel254', 'pixel255',
'pixel256', 'pixel257', 'pixel258', 'pixel259', 'pixel260', 'pixel261',
'pixel262', 'pixel263', 'pixel264', 'pixel265', 'pixel266', 'pixel267',
'pixel268', 'pixel269', 'pixel270', 'pixel271', 'pixel272', 'pixel273',
'pixel274', 'pixel275', 'pixel276', 'pixel277', 'pixel278', 'pixel279',
```

```
'pixel280', 'pixel281', 'pixel282', 'pixel283', 'pixel284', 'pixel285',
'pixel286', 'pixel287', 'pixel288', 'pixel289', 'pixel290', 'pixel291',
'pixel292', 'pixel293', 'pixel294', 'pixel295', 'pixel296', 'pixel297',
'pixel298', 'pixel299', 'pixel300', 'pixel301', 'pixel302', 'pixel303',
'pixel304', 'pixel305', 'pixel306', 'pixel307', 'pixel308', 'pixel309',
'pixel310', 'pixel311', 'pixel312', 'pixel313', 'pixel314', 'pixel315',
'pixel316', 'pixel317', 'pixel318', 'pixel319', 'pixel320', 'pixel321',
'pixel322', 'pixel323', 'pixel324', 'pixel325', 'pixel326', 'pixel327',
'pixel328', 'pixel329', 'pixel330', 'pixel331', 'pixel332', 'pixel333',
'pixel334', 'pixel335', 'pixel336', 'pixel337', 'pixel338', 'pixel339',
'pixel340', 'pixel341', 'pixel342', 'pixel343', 'pixel344', 'pixel345',
'pixel346', 'pixel347', 'pixel348', 'pixel349', 'pixel350', 'pixel351',
'pixel352', 'pixel353', 'pixel354', 'pixel355', 'pixel356', 'pixel357',
'pixel358', 'pixel359', 'pixel360', 'pixel361', 'pixel362', 'pixel363',
'pixel364', 'pixel365', 'pixel366', 'pixel367', 'pixel368', 'pixel369',
'pixel370', 'pixel371', 'pixel372', 'pixel373', 'pixel374', 'pixel375',
'pixel376', 'pixel377', 'pixel378', 'pixel379', 'pixel380', 'pixel381',
'pixel382', 'pixel383', 'pixel384', 'pixel385', 'pixel386', 'pixel387',
'pixel388', 'pixel389', 'pixel390', 'pixel391', 'pixel392', 'pixel393',
'pixel394', 'pixel395', 'pixel396', 'pixel397', 'pixel398', 'pixel399',
'pixel400', 'pixel401', 'pixel402', 'pixel403', 'pixel404', 'pixel405',
'pixel406', 'pixel407', 'pixel408', 'pixel409', 'pixel410', 'pixel411',
'pixel412', 'pixel413', 'pixel414', 'pixel415', 'pixel416', 'pixel417',
'pixel418', 'pixel419', 'pixel420', 'pixel421', 'pixel422', 'pixel423',
'pixel424', 'pixel425', 'pixel426', 'pixel427', 'pixel428', 'pixel429',
'pixel430', 'pixel431', 'pixel432', 'pixel433', 'pixel434', 'pixel435',
'pixel436', 'pixel437', 'pixel438', 'pixel439', 'pixel440', 'pixel441',
'pixel442', 'pixel443', 'pixel444', 'pixel445', 'pixel446', 'pixel447',
'pixel448', 'pixel449', 'pixel450', 'pixel451', 'pixel452', 'pixel453',
'pixel454', 'pixel455', 'pixel456', 'pixel457', 'pixel458', 'pixel459',
'pixel460', 'pixel461', 'pixel462', 'pixel463', 'pixel464', 'pixel465',
'pixel466', 'pixel467', 'pixel468', 'pixel469', 'pixel470', 'pixel471',
'pixel472', 'pixel473', 'pixel474', 'pixel475', 'pixel476', 'pixel477',
'pixel478', 'pixel479', 'pixel480', 'pixel481', 'pixel482', 'pixel483',
'pixel484', 'pixel485', 'pixel486', 'pixel487', 'pixel488', 'pixel489',
'pixel490', 'pixel491', 'pixel492', 'pixel493', 'pixel494', 'pixel495',
'pixel496', 'pixel497', 'pixel498', 'pixel499', 'pixel500', 'pixel501',
'pixel502', 'pixel503', 'pixel504', 'pixel505', 'pixel506', 'pixel507',
'pixel508', 'pixel509', 'pixel510', 'pixel511', 'pixel512', 'pixel513',
'pixel514', 'pixel515', 'pixel516', 'pixel517', 'pixel518', 'pixel519',
'pixel520', 'pixel521', 'pixel522', 'pixel523', 'pixel524', 'pixel525',
'pixel526', 'pixel527', 'pixel528', 'pixel529', 'pixel530', 'pixel531',
'pixel532', 'pixel533', 'pixel534', 'pixel535', 'pixel536', 'pixel537',
'pixel538', 'pixel539', 'pixel540', 'pixel541', 'pixel542', 'pixel543',
'pixel544', 'pixel545', 'pixel546', 'pixel547', 'pixel548', 'pixel549',
'pixel550', 'pixel551', 'pixel552', 'pixel553', 'pixel554', 'pixel555',
'pixel556', 'pixel557', 'pixel558', 'pixel559', 'pixel560', 'pixel561',
'pixel562', 'pixel563', 'pixel564', 'pixel565', 'pixel566', 'pixel567',
```

```
'pixel568', 'pixel569', 'pixel570', 'pixel571', 'pixel572', 'pixel573',
'pixel574', 'pixel575', 'pixel576', 'pixel577', 'pixel578', 'pixel579',
'pixel580', 'pixel581', 'pixel582', 'pixel583', 'pixel584', 'pixel585',
'pixel586', 'pixel587', 'pixel588', 'pixel589', 'pixel590', 'pixel591',
'pixel592', 'pixel593', 'pixel594', 'pixel595', 'pixel596', 'pixel597',
'pixel598', 'pixel599', 'pixel600', 'pixel601', 'pixel602', 'pixel603',
'pixel604', 'pixel605', 'pixel606', 'pixel607', 'pixel608', 'pixel609',
'pixel610', 'pixel611', 'pixel612', 'pixel613', 'pixel614', 'pixel615',
'pixel616', 'pixel617', 'pixel618', 'pixel619', 'pixel620', 'pixel621',
'pixel622', 'pixel623', 'pixel624', 'pixel625', 'pixel626', 'pixel627',
'pixel628', 'pixel629', 'pixel630', 'pixel631', 'pixel632', 'pixel633',
'pixel634', 'pixel635', 'pixel636', 'pixel637', 'pixel638', 'pixel639',
'pixel640', 'pixel641', 'pixel642', 'pixel643', 'pixel644', 'pixel645',
'pixel646', 'pixel647', 'pixel648', 'pixel649', 'pixel650', 'pixel651',
'pixel652', 'pixel653', 'pixel654', 'pixel655', 'pixel656', 'pixel657',
'pixel658', 'pixel659', 'pixel660', 'pixel661', 'pixel662', 'pixel663',
'pixel664', 'pixel665', 'pixel666', 'pixel667', 'pixel668', 'pixel669',
'pixel670', 'pixel671', 'pixel672', 'pixel673', 'pixel674', 'pixel675',
'pixel676', 'pixel677', 'pixel678', 'pixel679', 'pixel680', 'pixel681',
'pixel682', 'pixel683', 'pixel684', 'pixel685', 'pixel686', 'pixel687',
'pixel688', 'pixel689', 'pixel690', 'pixel691', 'pixel692', 'pixel693',
'pixel694', 'pixel695', 'pixel696', 'pixel697', 'pixel698', 'pixel699',
'pixel700', 'pixel701', 'pixel702', 'pixel703', 'pixel704', 'pixel705',
'pixel706', 'pixel707', 'pixel708', 'pixel709', 'pixel710', 'pixel711',
'pixel712', 'pixel713', 'pixel714', 'pixel715', 'pixel716', 'pixel717',
'pixel718', 'pixel719', 'pixel720', 'pixel721', 'pixel722', 'pixel723',
'pixel724', 'pixel725', 'pixel726', 'pixel727', 'pixel728', 'pixel729',
'pixel730', 'pixel731', 'pixel732', 'pixel733', 'pixel734', 'pixel735',
'pixel736', 'pixel737', 'pixel738', 'pixel739', 'pixel740', 'pixel741',
'pixel742', 'pixel743', 'pixel744', 'pixel745', 'pixel746', 'pixel747',
'pixel748', 'pixel749', 'pixel750', 'pixel751', 'pixel752', 'pixel753',
'pixel754', 'pixel755', 'pixel756', 'pixel757', 'pixel758', 'pixel759',
'pixel760', 'pixel761', 'pixel762', 'pixel763', 'pixel764', 'pixel765',
'pixel766', 'pixel767', 'pixel768', 'pixel769', 'pixel770', 'pixel771',
'pixel772', 'pixel773', 'pixel774', 'pixel775', 'pixel776', 'pixel777',
'pixel778', 'pixel779', 'pixel780', 'pixel781', 'pixel782', 'pixel783',
'pixel784']

['class']

**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
**Please cite**:

The MNIST database of handwritten digits with 784 features, raw data available
at: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the
first 60,000 examples, and a test set of 10,000 examples
```

It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

With some classification methods (particularly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications. The MNIST database was constructed from NIST's NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint. SD-1 contains 58,527 digit images written by 500 different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample training set is available.

Downloaded from openml.org.

{'id': '554', 'name': 'mnist_784', 'version': '1', 'format': 'ARFF', 'creator': ['Yann LeCun', 'Corinna Cortes', 'Christopher J.C. Burges'], 'upload_date': '2014-09-29T03:28:38', 'language': 'English', 'licence': 'Public', 'url': 'https://www.openml.org/data/v1/download/52667/mnist_784.arff', 'file_id':

```
'52667', 'default_target_attribute': 'class', 'tag': ['AzurePilot', 'OpenML-
CC18', 'OpenML100', 'study_1', 'study_123', 'study_41', 'study_99', 'vision'],
'visibility': 'public', 'minio_url':
'http://openml1.win.tue.nl/dataset554/dataset_554.pq', 'status': 'active',
'processing_date': '2020-11-20 20:12:09', 'md5_checksum':
'0298d579eb1b86163de7723944c7e495'}

https://www.openml.org/d/554

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

Visualizing one image

[4]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
from random import randint

# index = randint(0, 70000)
index = 0

some_digit = X[index]
label = y[index]

some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap="binary")
plt.title(label)
plt.axis("off")
plt.show()
```

Creating training and test sets

[5]:
```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

1.2 Training a Binary Classifier

Create the target vectors for this binary classification task

[6]:
```
y_train_5 = (y_train == 5) # True for all 5s, False for all other digits
y_test_5 = (y_test == 5)
```

Training a Stochastic Gradient Descent (SGD) classifier

[7]:
```
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])
```

[7]:
```
array([ True])
```

1.3 Performance Measures

1.3.1 Measuring Accuracy Using Cross-Validation

[8]:
```
from sklearn.model_selection import cross_val_score

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

[8]:
```
array([0.95035, 0.96035, 0.9604 ])
```

Implementing Cross-Validation

[9]:
```
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))
```

```
/home/denys/.conda/envs/handson-ml2-denys/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:297: FutureWarning: Setting a
random_state has no effect since shuffle is False. This will raise an error in
0.24. You should leave random_state to its default (None), or set shuffle=True.
  FutureWarning
0.95035
0.96035
0.9604
```

1.3.2 Confusion Matrix

[10]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
print(confusion_matrix(y_train_5, y_train_pred))

y_train_perfect_predictions = y_train_5
print(confusion_matrix(y_train_5, y_train_perfect_predictions))
```

```
[[53892   687]
 [ 1891  3530]]
[[54579     0]
 [    0  5421]]
```

1.3.3 Precision and Recall

[11]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

print(precision_score(y_train_5, y_train_pred))
print(recall_score(y_train_5, y_train_pred))
print(f1_score(y_train_5, y_train_pred))
```

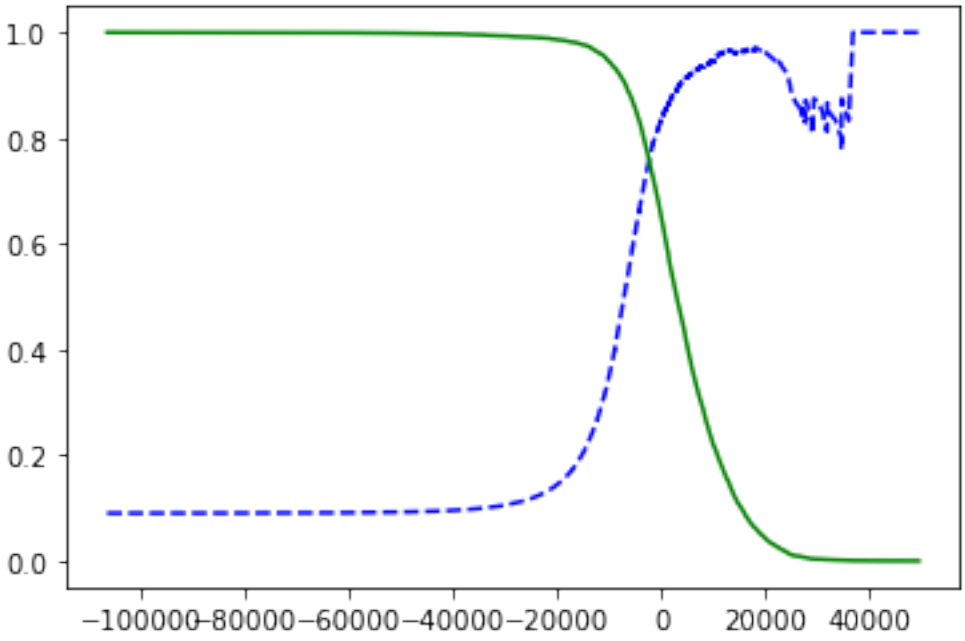
```
0.8370879772350012
0.6511713705958311
0.7325171197343846
```

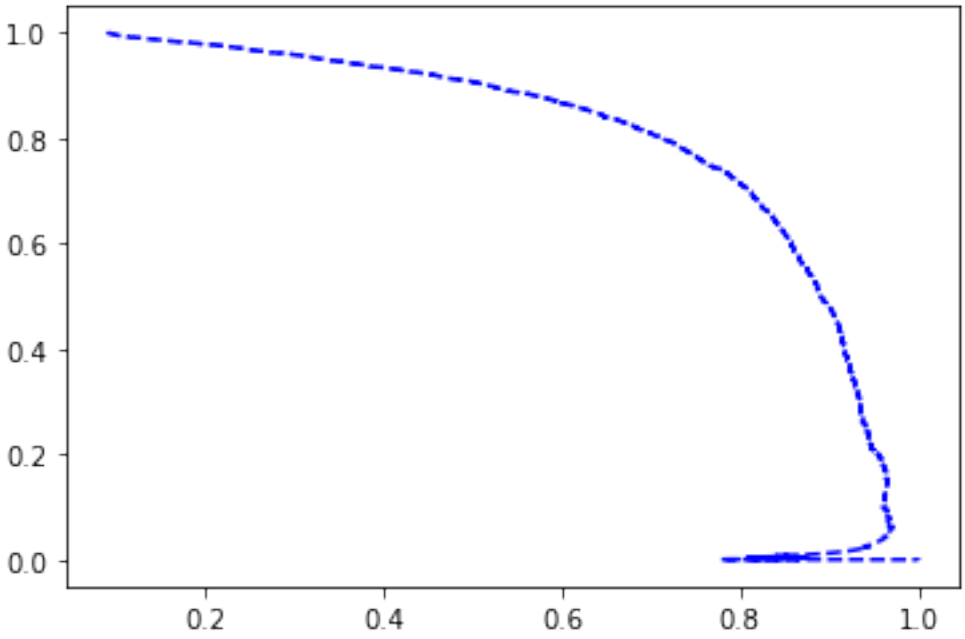
1.3.4 Precision/Recall Trade-off

[12]:
```
y_scores = sgd_clf.decision_function([some_digit])
print(y_scores)
threshold = 0
y_some_digit_pred = (y_scores > threshold)
print(y_some_digit_pred)

threshold = 8000
y_some_digit_pred = (y_scores > threshold)
print(y_some_digit_pred)
```

```
[2164.22030239]
[ True]
[False]
```

[13]:
```
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)

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")

def plot_precision_vs_recall(precisions, recalls):
    plt.plot(precisions[:-1], recalls[:-1], "b--")

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()

plot_precision_vs_recall(precisions, recalls)
plt.show()
```





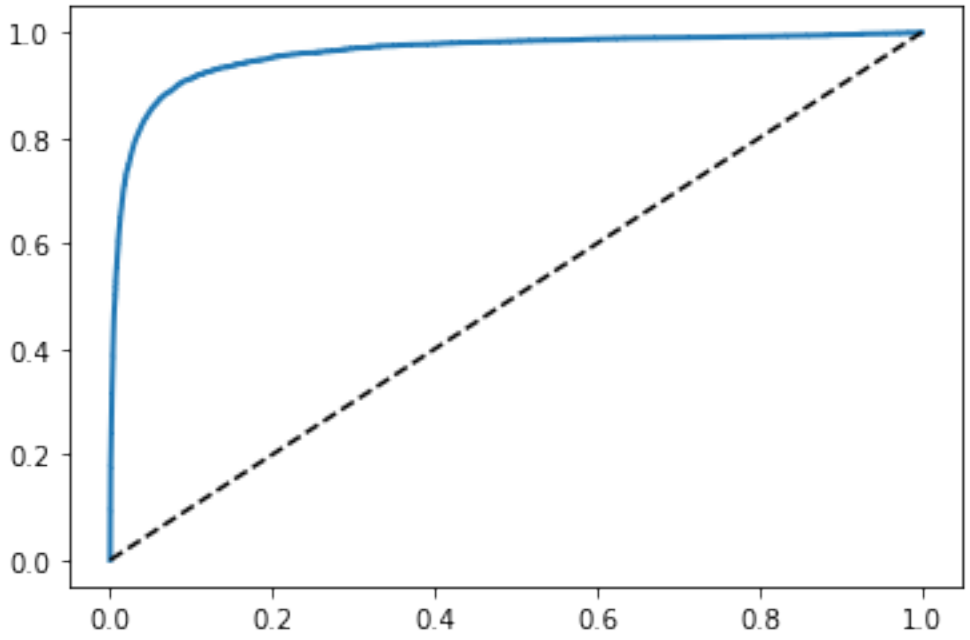
[14]:
```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816
print(threshold_90_precision)

y_train_pred_90 = (y_scores >= threshold_90_precision)
```

```
print(precision_score(y_train_5, y_train_pred_90))

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

```
3370.0194991439557
0.9000345901072293
0.4799852425751706
```

1.3.5 The ROC Curve

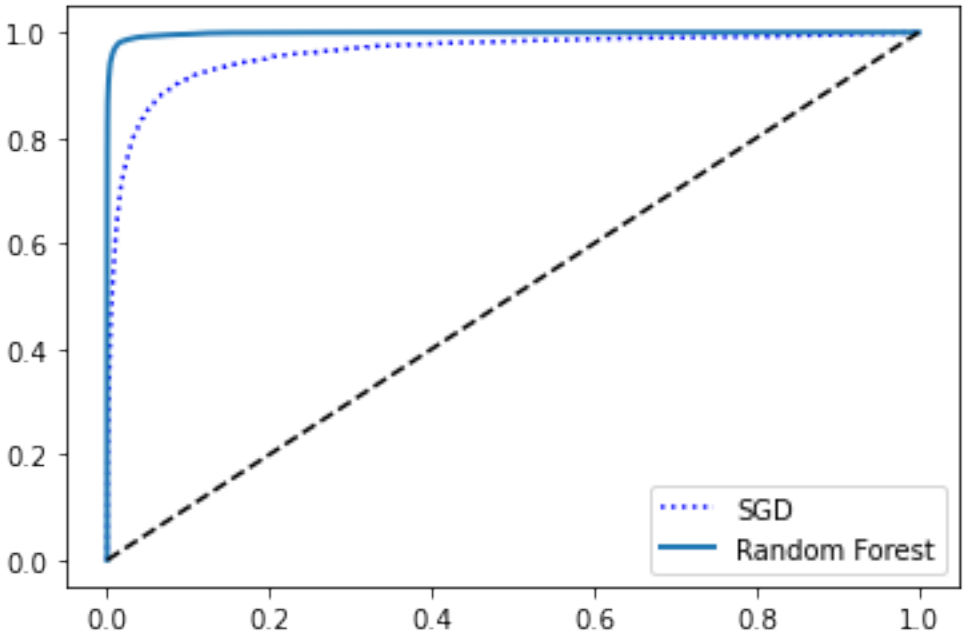
[15]:
```
from sklearn.metrics import roc_curve

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--')

plot_roc_curve(fpr, tpr)
plt.show()

from sklearn.metrics import roc_auc_score

print("Area Under the Curve - AUC : ", roc_auc_score(y_train_5, y_scores))
```



```
Area Under the Curve - AUC :  0.9604938554008616
```

Training a RandomForestClassifier and compare its ROC curve and ROC AUC score to those of the SGDClassifier

[16]:
```
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")
y_scores_forest = y_probas_forest[:, 1]
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)

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

print("Area Under the Curve - AUC for the RFC: ", roc_auc_score(y_train_5,␣
 ↪y_scores_forest))
```




```
Area Under the Curve - AUC for the RFC:  0.9983436731328145
```

1.3.6 Multiclass Classification

[17]:
```
from sklearn.svm import SVC

svm_clf = SVC()
svm_clf.fit(X_train, y_train)
svm_clf.predict([some_digit])


some_digit_scores = svm_clf.decision_function([some_digit])
print(some_digit_scores)


print(np.argmax(some_digit_scores))
print(svm_clf.classes_)
print(svm_clf.classes_[5])
```

```
[[ 1.72501977  2.72809088  7.2510018   8.3076379  -0.31087254  9.3132482
   1.70975103  2.76765202  6.23049537  4.84771048]]
5
[0 1 2 3 4 5 6 7 8 9]
5
```

Creating a multiclass classifier using the OvR strategy, based on an SVC

[18]:
```
from sklearn.multiclass import OneVsRestClassifier

ovr_clf = OneVsRestClassifier(SVC())
ovr_clf.fit(X_train, y_train)
ovr_clf.predict([some_digit])
print(len(ovr_clf.estimators_))
```

```
10
```

Training an SGDClassifier

[19]:
```
sgd_clf.fit(X_train, y_train)
sgd_clf.predict([some_digit])
sgd_clf.decision_function([some_digit])
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")

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")
```

[19]:
```
array([0.8983, 0.891 , 0.9018])
```