Lab1

September 2, 2021

1 Introduction to Python and Scikit-Learn

Note: This lab has been generated using material from Chapter 3 in ”Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. The following link (https://github.com/ageron/handson-ml) contains the extended Jupyter notebook as well as more tasks so you can better familiarize yourself with Python and Scikit-learn.

2 1. MNIST

MNIST dataset, is a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. Each image is labeled with the digit it represents. For the first task download and display some of the digits in the dataset.

[410]:
```
import pandas as pd
import os
import numpy as np
# to make this notebook's output stable across runs
np.random.seed(42)
import sklearn # ML in python

# 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)
```

[411]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
```

[411]:
```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'DESCR',
'details', 'categories', 'url'])
```

[412]:
```
mnist.values()
```

[412]:
```
dict_values([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(['5', '0', '4', ..., '4', '5',
'6'], dtype=object), None, ['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 \n**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown \n**Please cite**: \n\nThe 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 \n\nIt 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. \n\nWith 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. \n\nThe 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.\n\nDownloaded from openml.org.", {'id': '554', 'name': 'mnist_784',
'version': '1', 'description_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', 'status': 'active', 'processing_date': '2020-11-20
20:12:09', 'md5_checksum': '0298d579eb1b86163de7723944c7e495'}, {},
'https://www.openml.org/d/554'])
```


[413]:
```python
# set up the feature data and labels
X, y = mnist["data"], mnist["target"]
X.shape
print(X[0]) # X is 2D array
```

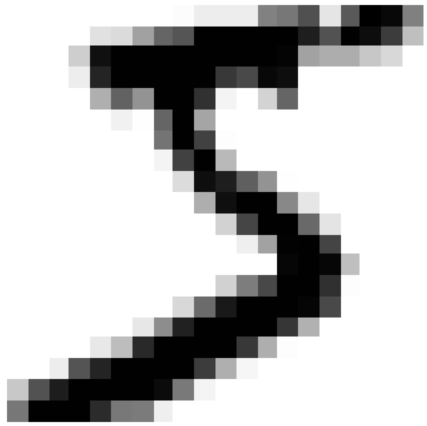
```
[  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   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.   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.   0.   0.   0.   0.   0.   0.   0.   0.   3.  18.
  18.  18. 126. 136. 175.  26. 166. 255. 247. 127.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.  30.  36.  94. 154. 170. 253.
 253. 253. 253. 253. 225. 172. 253. 242. 195.  64.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.  49. 238. 253. 253. 253. 253. 253.
 253. 253. 253. 251.  93.  82.  82.  56.  39.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.  18. 219. 253. 253. 253. 253. 253.
 198. 182. 247. 241.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.  80. 156. 107. 253. 253. 205.
  11.   0.  43. 154.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.  14.   1. 154. 253.  90.
   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. 139. 253. 190.
   2.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  11. 190. 253.
  70.   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.  35. 241.
 225. 160. 108.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  81.
 240. 253. 253. 119.  25.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
```

```
  45. 186. 253. 253. 150.  27.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.  16.  93. 252. 253. 187.   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. 249. 253. 249.  64.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  46. 130. 183. 253. 253. 207.   2.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  39. 148.
 229. 253. 253. 253. 250. 182.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  24. 114. 221. 253.
 253. 253. 253. 201.  78.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.  23.  66. 213. 253. 253. 253.
 253. 198.  81.   2.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.  18. 171. 219. 253. 253. 253. 253. 195.
  80.   9.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.  55. 172. 226. 253. 253. 253. 253. 244. 133.  11.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0. 136. 253. 253. 253. 212. 135. 132.  16.   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.   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.
   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.]
```

[414]:
```
y.shape
#print(len(y))
print(y[0]) # y is 1D array
```

```
5
```

[415]:
```
some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap=mpl.cm.binary)
plt.axis("off")
plt.show()
plt.savefig("1.png")
```



```
<Figure size 432x288 with 0 Axes>
```

3 2. Binary Classifier

2.1 Identify one digit for example, the number 5. This "5-detector" will be an example of a binary classifier, capable of distinguishing between just two classes, 5 and not-5. For this task pick the Stochastic gradient descent classifier from the Scikit-Learn's SGDClassifier class.

2.2 Evaluate the performance of your classifier by
(a) Measuring accuracy using cross-validation.
(b) The use of the confusion matrix.
(c) Understanding the precision/recall trade-off.
(d) The use of the ROC curve.

2.3 Compare the ROC curve generated by the RandomForestClassifier with the ROC curve generated by the SGDClassifier

[416]:
```
# change y to integer
y = y.astype(np.uint8)
type(y)
```

[416]:
```
numpy.ndarray
```

[417]:
```
# create a training and testing set
# The MNIST dataset is actually already split into a training set (the first␣
↪60,000 images) and a test set (the last 10,000 images), so:
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
shuffle_index = np.random.permutation(60000)
#print(shuffle_index)
```

```
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
print(X_train)
```

```
[[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.]]
```

[418]:
```
# Distinguish between two classes, 5 and not-5. Create the target vectors
y_train_5 = (y_train == 5) # make it as binary
y_test_5 = (y_test == 5)
print(y_train_5)
```

```
[False False False ... False False False]
```

[419]:
```
#X_train.shape
y_train_5
```

[419]:
```
array([False, False, False, ..., False, False, False])
```

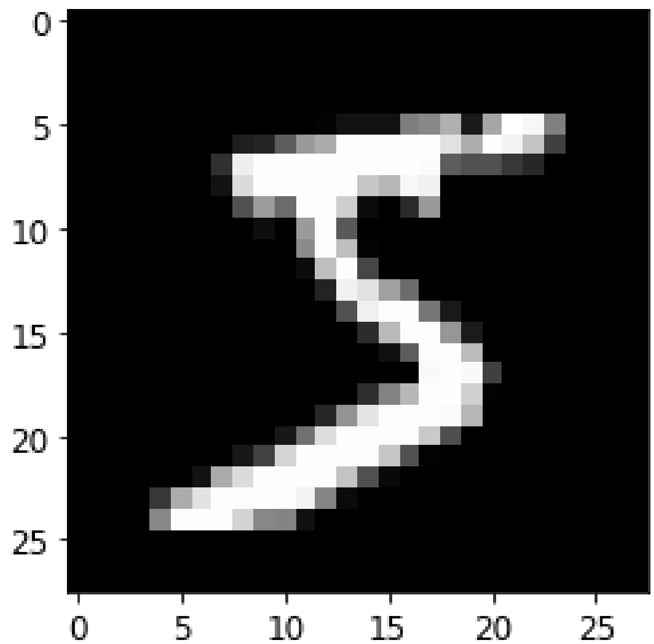
[420]:
```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(max_iter = 5, tol = -np.infty, random_state = 42) #␣
 ↪reproducible results so set a random_state
sgd_clf.fit(X_train, y_train_5)
```

[420]:
```
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)
```

[421]:
```
sgd_clf.predict([some_digit])
```

[421]:
```
array([ True])
```

[422]:
```
plt.imshow(some_digit_image, cmap="gray")
plt.savefig("2.png")
```



3.1 Measuring Accuracy Using Cross-Vaildation

[423]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Let's use the `cross_val_score()` function to evaluate our SGDClassifier model, using K-fold cross-validation with three folds. Remember that K-fold cross-validation means splitting the training set into K folds (in this case, three), then making predictions and evaluating them on each fold using a model trained on the remaining folds

[424]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

[424]:
```
array([0.964 , 0.9579, 0.9571])
```

3.2 Confusion Matrix

Each row in a confusion matrix represents an actual class, while each column represents a predicted class.

[425]:
```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)#␣
↪regression, data, target
print(len(y_train_pred))
```

60000

[426]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5, y_train_pred)
```

[426]:
```
array([[54058,   521],
       [ 1899,  3522]])
```

3.3 Precision/Recall trade-off

3.3.1 Precision = TP/ (TP + FP)

- TP is the number of true positives, and FP is the number of false positives.

3.3.2 Recall (Sensitivity) = TP / (TP + FN)

- FN is, of course, the number of false negatives.

[427]:
```python
# Precision
sklearn.metrics.precision_score(y_train_5, y_train_pred) # equal to 4096 /␣
↪(4096 + 1522)
```

[427]: 0.8711352955725946

[428]:
```python
# Recall
sklearn.metrics.recall_score(y_train_5, y_train_pred) # equal to 4096 / (4096 +␣
↪1325)
```

[428]: 0.6496956281128943

3.4 F1 Score

Combining **precision** and **recall** into a single metric called the F1 score

[429]:
```python
sklearn.metrics.f1_score(y_train_5, y_train_pred)
```

[429]: 0.7442941673710904

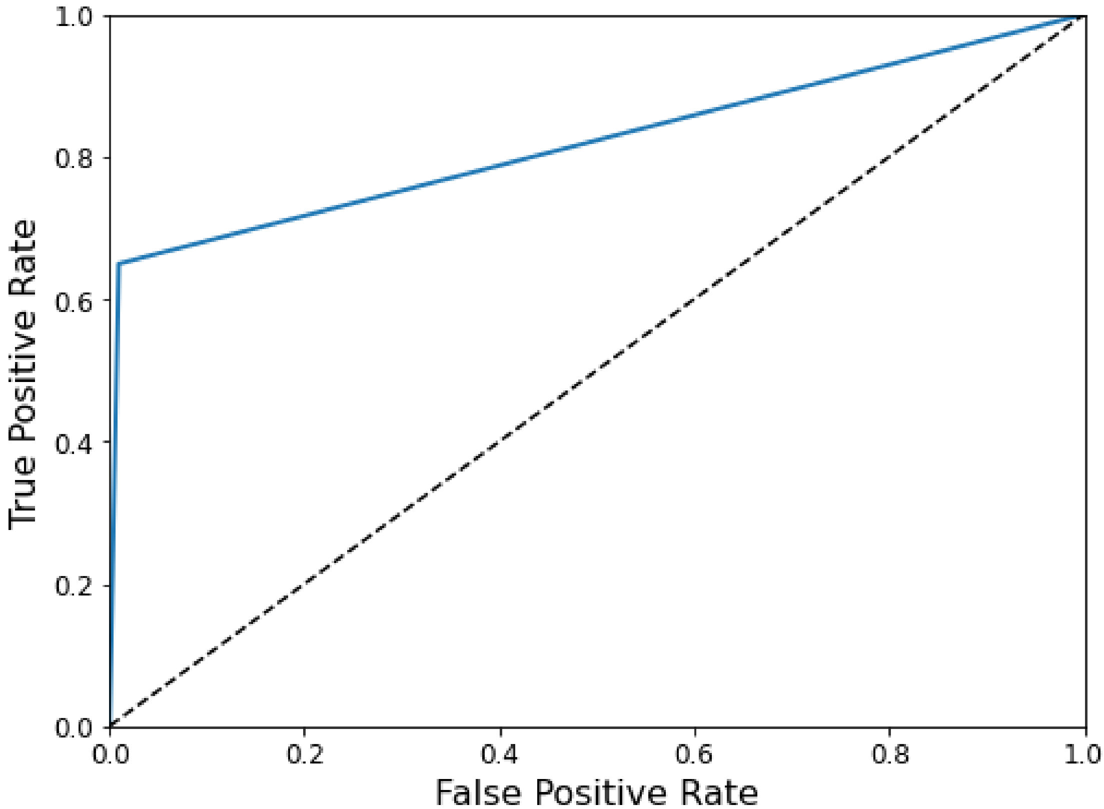
4 The ROC Curve

ROC curve plots the true positive rate (another name for recall aka TPR) against the false positive rate (FPR).

[430]:
```python
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_train_pred)
```

[431]:
```python
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)
plt.show()
plt.savefig("3.png")
```




```
<Figure size 432x288 with 0 Axes>
```

5 Random Forest Classifier

[ ]:
```
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")
```

[ ]:
```
y_probas_forest[:,1] # take all the row and only the second column
```

[ ]:
```
y_probas_forest # The predict_proba() method returns an array containing a row␣
 ↪per instance and a column per class
```

[ ]:
```
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)
```

[ ]:
```
# Compare the ROC curve generated by the RandomForestClassifier with the ROC␣
 ↪curve geenrated by the SGDClassifier
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right", fontsize=16)
plt.show()
plt.savefig("4.png")
```

6 Conclusion

Random Forest model performs better.

[ ]:
```
# !jupyter nbconvert --to html "/content/drive/MyDrive/American_University/
 ↪2021_Fall/DATA-642-001_Advanced Machine Learning/GitHub/Labs/Lab1.ipynb"
!jupyter nbconvert --to pdf "/content/drive/MyDrive/American_University/
 ↪2021_Fall/DATA-642-001_Advanced Machine Learning/GitHub/Labs/Lab1.ipynb"
```