Out[1]:

Click here to show or hide your raw code.

The MNIST Problem

The MNIST database (Modified National Institute of Standards and Technology database[1]) is a large database of handwritten digits that is commonly used for training and testing advanced machine learning algorithms. General references are:

MNIST database. Wikipedia. https://en.wikipedia.org/wiki/MNIST_database (https://en.wikipedia.org/wiki/MNIST_database).

THE MNIST DATABASE of handwritten digits. Yann LeCun, Courant Institute, NYU Corinna Cortes, Google Labs, New York Christopher J.C. Burges, Microsoft Research, Redmond. http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/)

Classification datasets results. Rodrigo Benenson. https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html (https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)

The MNIST database contains 60,000 training images and 10,000 testing images. In our dataset the images will be 32 x 32 greyscale digit rasters. In order to manage our computations in reasonable time, we are going to work only with the test subset, which we will further randomly split in a 20% train and validation subset and an 80% test subset.

Student contributions

- Student `Gupta, Kush` has done exploratory research and data analysis on the studied topics, and descriptive analysis of the obtained results.
- Student `Nemeth, Sebestyen` has implemented the data preprocessing and fitting methods and visualized the results.
- The students worked together on definining the workplan of the project, recapping the methods learnt on the laboratory sessions and analyzing the results.

Loading Data

Notice that the shape of each pattern is given by a $32 \times 32 \times 1$ tensor. Thus, you may have to reshape it to either a matrix or a vector depending on the task you want to perform.

```
dict_keys(['DESCR', 'target', 'target_test', 'data', 'data_test'])
data_shape: (60000, 1024)
data_test_shape: (10000, 1024)
```

```
[7 2 1 ... 4 5 6]
```

```
['p_0_0', 'p_0_1', 'p_0_2', 'p_0_3', 'p_0_4', 'p_0_5', 'p_0_6', 'p_0_
7', 'p_0_8', 'p_0_9', 'p_0_10', 'p_0_11', 'p_0_12', 'p_0_13', 'p_0_1
4', 'p_0_15', 'p_0_16', 'p_0_17', 'p_0_18', 'p_0_19', 'p_0_20', 'p_0_2
1', 'p_0_22', 'p_0_23', 'p_0_24', 'p_0_25', 'p_0_26', 'p_0_27', 'p_0_2
8', 'p_0_29', 'p_0_30', 'p_0_31', 'p_1_0', 'p_1_1', 'p_1_2', 'p_1_3',
'p_1_4', 'p_1_5', 'p_1_6', 'p_1_7', 'p_1_8', 'p_1_9', 'p_1_10', 'p_1_1
1', 'p_1_12', 'p_1_13', 'p_1_14', 'p_1_15', 'p_1_16', 'p_1_17', 'p_1_1
8', 'p_1_19', 'p_1_20', 'p_1_21', 'p_1_22', 'p_1_23', 'p_1_24', 'p_1_2
5', 'p_1_26', 'p_1_27', 'p_1_28', 'p_1_29', 'p_1_30', 'p_1_31', 'p_2_
0', 'p_2_1', 'p_2_2', 'p_2_3', 'p_2_4', 'p_2_5', 'p_2_6', 'p_2_7', 'p_
2_8', 'p_2_9', 'p_2_10', 'p_2_11', 'p_2_12', 'p_2_13', 'p_2_14', 'p_2_
15', 'p_2_16', 'p_2_17', 'p_2_18', 'p_2_19', 'p_2_20', 'p_2_21', 'p_2_
22', 'p_2_23', 'p_2_24', 'p_2_25', 'p_2_26', 'p_2_27', 'p_2_28', 'p_2_
29', 'p_2_30', 'p_2_31', 'p_3_0', 'p_3_1', 'p_3_2', 'p_3_3', 'p_3_4',
'p_3_5', 'p_3_6', 'p_3_7', 'p_3_8', 'p_3_9', 'p_3_10', 'p_3_11', 'p_3_
12', 'p_3_13', 'p_3_14', 'p_3_15', 'p_3_16', 'p_3_17', 'p_3_18', 'p_3_
19', 'p_3_20', 'p_3_21', 'p_3_22', 'p_3_23', 'p_3_24', 'p_3_25', 'p_3_
26', 'p_3_27', 'p_3_28', 'p_3_29', 'p_3_30', 'p_3_31', 'p_4_0', 'p_4_
1', 'p_4_2', 'p_4_3', 'p_4_4', 'p_4_5', 'p_4_6', 'p_4_7', 'p_4_8', 'p_
4_9', 'p_4_10', 'p_4_11', 'p_4_12', 'p_4_13', 'p_4_14', 'p_4_15', 'p_4
_16', 'p_4_17', 'p_4_18', 'p_4_19', 'p_4_20', 'p_4_21', 'p_4_22', 'p_4
_23', 'p_4_24', 'p_4_25', 'p_4_26', 'p_4_27', 'p_4_28', 'p_4_29', 'p_4
_30', 'p_4_31', 'p_5_0', 'p_5_1', 'p_5_2', 'p_5_3', 'p_5_4', 'p_5_5',
'p_5_6', 'p_5_7', 'p_5_8', 'p_5_9', 'p_5_10', 'p_5_11', 'p_5_12', 'p_5
_13', 'p_5_14', 'p_5_15', 'p_5_16', 'p_5_17', 'p_5_18', 'p_5_19', 'p_5
_20', 'p_5_21', 'p_5_22', 'p_5_23', 'p_5_24', 'p_5_25', 'p_5_26', 'p_5
_27', 'p_5_28', 'p_5_29', 'p_5_30', 'p_5_31', 'p_6_0', 'p_6_1', 'p_6_
2', 'p_6_3', 'p_6_4', 'p_6_5', 'p_6_6', 'p_6_7', 'p_6_8', 'p_6_9', 'p_
6_10', 'p_6_11', 'p_6_12', 'p_6_13', 'p_6_14', 'p_6_15', 'p_6_16', 'p_
6_17', 'p_6_18', 'p_6_19', 'p_6_20', 'p_6_21', 'p_6_22', 'p_6_23', 'p_
6_24', 'p_6_25', 'p_6_26', 'p_6_27', 'p_6_28', 'p_6_29', 'p_6_30', 'p_
6_31', 'p_7_0', 'p_7_1', 'p_7_2', 'p_7_3', 'p_7_4', 'p_7_5', 'p_7_6',
'p_7_7', 'p_7_8', 'p_7_9', 'p_7_10', 'p_7_11', 'p_7_12', 'p_7_13', 'p_
7_14', 'p_7_15', 'p_7_16', 'p_7_17', 'p_7_18', 'p_7_19', 'p_7_20', 'p_
7_21', 'p_7_22', 'p_7_23', 'p_7_24', 'p_7_25', 'p_7_26', 'p_7_27', 'p_
7_28', 'p_7_29', 'p_7_30', 'p_7_31', 'p_8_0', 'p_8_1', 'p_8_2', 'p_8_
3', 'p_8_4', 'p_8_5', 'p_8_6', 'p_8_7', 'p_8_8', 'p_8_9', 'p_8_10', 'p
_8_11', 'p_8_12', 'p_8_13', 'p_8_14', 'p_8_15', 'p_8_16', 'p_8_17', 'p
_8_18', 'p_8_19', 'p_8_20', 'p_8_21', 'p_8_22', 'p_8_23', 'p_8_24', 'p
_8_25', 'p_8_26', 'p_8_27', 'p_8_28', 'p_8_29', 'p_8_30', 'p_8_31', 'p
_9_0', 'p_9_1', 'p_9_2', 'p_9_3', 'p_9_4', 'p_9_5', 'p_9_6', 'p_9_7',
'p_9_8', 'p_9_9', 'p_9_10', 'p_9_11', 'p_9_12', 'p_9_13', 'p_9_14', 'p
_9_15', 'p_9_16', 'p_9_17', 'p_9_18', 'p_9_19', 'p_9_20', 'p_9_21', 'p
_9_22', 'p_9_23', 'p_9_24', 'p_9_25', 'p_9_26', 'p_9_27', 'p_9_28', 'p
_9_29', 'p_9_30', 'p_9_31', 'p_10_0', 'p_10_1', 'p_10_2', 'p_10_3', 'p
_10_4', 'p_10_5', 'p_10_6', 'p_10_7', 'p_10_8', 'p_10_9', 'p_10_10',
'p_10_11', 'p_10_12', 'p_10_13', 'p_10_14', 'p_10_15', 'p_10_16', 'p_1
0_17', 'p_10_18', 'p_10_19', 'p_10_20', 'p_10_21', 'p_10_22', 'p_10_2
3', 'p_10_24', 'p_10_25', 'p_10_26', 'p_10_27', 'p_10_28', 'p_10_29',
'p_10_30', 'p_10_31', 'p_11_0', 'p_11_1', 'p_11_2', 'p_11_3', 'p_11_
4', 'p_11_5', 'p_11_6', 'p_11_7', 'p_11_8', 'p_11_9', 'p_11_10', 'p_11
_11', 'p_11_12', 'p_11_13', 'p_11_14', 'p_11_15', 'p_11_16', 'p_11_1
7', 'p_11_18', 'p_11_19', 'p_11_20', 'p_11_21', 'p_11_22', 'p_11_23',
'p_11_24', 'p_11_25', 'p_11_26', 'p_11_27', 'p_11_28', 'p_11_29', 'p_1
1_30', 'p_11_31', 'p_12_0', 'p_12_1', 'p_12_2', 'p_12_3', 'p_12_4', 'p
_12_5', 'p_12_6', 'p_12_7', 'p_12_8', 'p_12_9', 'p_12_10', 'p_12_11',
'p_12_12', 'p_12_13', 'p_12_14', 'p_12_15', 'p_12_16', 'p_12_17', 'p_1
2_18', 'p_12_19', 'p_12_20', 'p_12_21', 'p_12_22', 'p_12_23', 'p_12_2
4', 'p_12_25', 'p_12_26', 'p_12_27', 'p_12_28', 'p_12_29', 'p_12_30',
'p_12_31', 'p_13_0', 'p_13_1', 'p_13_2', 'p_13_3', 'p_13_4', 'p_13_5',
 'p_13_6', 'p_13_7', 'p_13_8', 'p_13_9', 'p_13_10', 'p_13_11', 'p_13_1
 2', 'p_13_13', 'p_13_14', 'p_13_15', 'p_13_16', 'p_13_17', 'p_13_18',
 'p_13_19', 'p_13_20', 'p_13_21', 'p_13_22', 'p_13_23', 'p_13_24', 'p_1
 3_25', 'p_13_26', 'p_13_27', 'p_13_28', 'p_13_29', 'p_13_30', 'p_13_3
 1', 'p_14_0', 'p_14_1', 'p_14_2', 'p_14_3', 'p_14_4', 'p_14_5', 'p_14_
 6', 'p_14_7', 'p_14_8', 'p_14_9', 'p_14_10', 'p_14_11', 'p_14_12', 'p_
 14_13', 'p_14_14', 'p_14_15', 'p_14_16', 'p_14_17', 'p_14_18', 'p_14_1
 9', 'p_14_20', 'p_14_21', 'p_14_22', 'p_14_23', 'p_14_24', 'p_14_25',
 'p_14_26', 'p_14_27', 'p_14_28', 'p_14_29', 'p_14_30', 'p_14_31', 'p_1
 5_0', 'p_15_1', 'p_15_2', 'p_15_3', 'p_15_4', 'p_15_5', 'p_15_6', 'p_1
 5_7', 'p_15_8', 'p_15_9', 'p_15_10', 'p_15_11', 'p_15_12', 'p_15_13',
 'p_15_14', 'p_15_15', 'p_15_16', 'p_15_17', 'p_15_18', 'p_15_19', 'p_1
 5_20', 'p_15_21', 'p_15_22', 'p_15_23', 'p_15_24', 'p_15_25', 'p_15_2
 6', 'p_15_27', 'p_15_28', 'p_15_29', 'p_15_30', 'p_15_31', 'p_16_0',
 'p_16_1', 'p_16_2', 'p_16_3', 'p_16_4', 'p_16_5', 'p_16_6', 'p_16_7',
 'p_16_8', 'p_16_9', 'p_16_10', 'p_16_11', 'p_16_12', 'p_16_13', 'p_16_
 14', 'p_16_15', 'p_16_16', 'p_16_17', 'p_16_18', 'p_16_19', 'p_16_20',
 'p_16_21', 'p_16_22', 'p_16_23', 'p_16_24', 'p_16_25', 'p_16_26', 'p_1
 6_27', 'p_16_28', 'p_16_29', 'p_16_30', 'p_16_31', 'p_17_0', 'p_17_1',
 'p_17_2', 'p_17_3', 'p_17_4', 'p_17_5', 'p_17_6', 'p_17_7', 'p_17_8',
 'p_17_9', 'p_17_10', 'p_17_11', 'p_17_12', 'p_17_13', 'p_17_14', 'p_17
 _15', 'p_17_16', 'p_17_17', 'p_17_18', 'p_17_19', 'p_17_20', 'p_17_2
 1', 'p_17_22', 'p_17_23', 'p_17_24', 'p_17_25', 'p_17_26', 'p_17_27',
 'p_17_28', 'p_17_29', 'p_17_30', 'p_17_31', 'p_18_0', 'p_18_1', 'p_18_
 2', 'p_18_3', 'p_18_4', 'p_18_5', 'p_18_6', 'p_18_7', 'p_18_8', 'p_18_
 9', 'p_18_10', 'p_18_11', 'p_18_12', 'p_18_13', 'p_18_14', 'p_18_15',
 'p_18_16', 'p_18_17', 'p_18_18', 'p_18_19', 'p_18_20', 'p_18_21', 'p_1
 8_22', 'p_18_23', 'p_18_24', 'p_18_25', 'p_18_26', 'p_18_27', 'p_18_2
 8', 'p_18_29', 'p_18_30', 'p_18_31', 'p_19_0', 'p_19_1', 'p_19_2', 'p_
 19_3', 'p_19_4', 'p_19_5', 'p_19_6', 'p_19_7', 'p_19_8', 'p_19_9', 'p_
 19_10', 'p_19_11', 'p_19_12', 'p_19_13', 'p_19_14', 'p_19_15', 'p_19_1
 6', 'p_19_17', 'p_19_18', 'p_19_19', 'p_19_20', 'p_19_21', 'p_19_22',
 'p_19_23', 'p_19_24', 'p_19_25', 'p_19_26', 'p_19_27', 'p_19_28', 'p_1
 9_29', 'p_19_30', 'p_19_31', 'p_20_0', 'p_20_1', 'p_20_2', 'p_20_3',
 'p_20_4', 'p_20_5', 'p_20_6', 'p_20_7', 'p_20_8', 'p_20_9', 'p_20_10',
 'p_20_11', 'p_20_12', 'p_20_13', 'p_20_14', 'p_20_15', 'p_20_16', 'p_2
 0_17', 'p_20_18', 'p_20_19', 'p_20_20', 'p_20_21', 'p_20_22', 'p_20_2
 3', 'p_20_24', 'p_20_25', 'p_20_26', 'p_20_27', 'p_20_28', 'p_20_29',
 'p_20_30', 'p_20_31', 'p_21_0', 'p_21_1', 'p_21_2', 'p_21_3', 'p_21_
 4', 'p_21_5', 'p_21_6', 'p_21_7', 'p_21_8', 'p_21_9', 'p_21_10', 'p_21
 _11', 'p_21_12', 'p_21_13', 'p_21_14', 'p_21_15', 'p_21_16', 'p_21_1
 7', 'p_21_18', 'p_21_19', 'p_21_20', 'p_21_21', 'p_21_22', 'p_21_23',
 'p_21_24', 'p_21_25', 'p_21_26', 'p_21_27', 'p_21_28', 'p_21_29', 'p_2
 1_30', 'p_21_31', 'p_22_0', 'p_22_1', 'p_22_2', 'p_22_3', 'p_22_4', 'p
 _22_5', 'p_22_6', 'p_22_7', 'p_22_8', 'p_22_9', 'p_22_10', 'p_22_11',
 'p_22_12', 'p_22_13', 'p_22_14', 'p_22_15', 'p_22_16', 'p_22_17', 'p_2
 2_18', 'p_22_19', 'p_22_20', 'p_22_21', 'p_22_22', 'p_22_23', 'p_22_2
 4', 'p_22_25', 'p_22_26', 'p_22_27', 'p_22_28', 'p_22_29', 'p_22_30',
 'p_22_31', 'p_23_0', 'p_23_1', 'p_23_2', 'p_23_3', 'p_23_4', 'p_23_5',
 'p_23_6', 'p_23_7', 'p_23_8', 'p_23_9', 'p_23_10', 'p_23_11', 'p_23_1
 2', 'p_23_13', 'p_23_14', 'p_23_15', 'p_23_16', 'p_23_17', 'p_23_18',
 'p_23_19', 'p_23_20', 'p_23_21', 'p_23_22', 'p_23_23', 'p_23_24', 'p_2
 3_25', 'p_23_26', 'p_23_27', 'p_23_28', 'p_23_29', 'p_23_30', 'p_23_3
 1', 'p_24_0', 'p_24_1', 'p_24_2', 'p_24_3', 'p_24_4', 'p_24_5', 'p_24_
 6', 'p_24_7', 'p_24_8', 'p_24_9', 'p_24_10', 'p_24_11', 'p_24_12', 'p_
 24_13', 'p_24_14', 'p_24_15', 'p_24_16', 'p_24_17', 'p_24_18', 'p_24_1
 9', 'p_24_20', 'p_24_21', 'p_24_22', 'p_24_23', 'p_24_24', 'p_24_25',
 'p_24_26', 'p_24_27', 'p_24_28', 'p_24_29', 'p_24_30', 'p_24_31', 'p_2
 5_0', 'p_25_1', 'p_25_2', 'p_25_3', 'p_25_4', 'p_25_5', 'p_25_6', 'p_2
 5_7', 'p_25_8', 'p_25_9', 'p_25_10', 'p_25_11', 'p_25_12', 'p_25_13',
 'p_25_14', 'p_25_15', 'p_25_16', 'p_25_17', 'p_25_18', 'p_25_19', 'p_2
```

```
5_20', 'p_25_21', 'p_25_22', 'p_25_23', 'p_25_24', 'p_25_25', 'p_25_2
6', 'p_25_27', 'p_25_28', 'p_25_29', 'p_25_30', 'p_25_31', 'p_26_0',
'p_26_1', 'p_26_2', 'p_26_3', 'p_26_4', 'p_26_5', 'p_26_6', 'p_26_7',
'p_26_8', 'p_26_9', 'p_26_10', 'p_26_11', 'p_26_12', 'p_26_13', 'p_26_
14', 'p_26_15', 'p_26_16', 'p_26_17', 'p_26_18', 'p_26_19', 'p_26_20',
'p_26_21', 'p_26_22', 'p_26_23', 'p_26_24', 'p_26_25', 'p_26_26', 'p_2
6_27', 'p_26_28', 'p_26_29', 'p_26_30', 'p_26_31', 'p_27_0', 'p_27_1',
'p_27_2', 'p_27_3', 'p_27_4', 'p_27_5', 'p_27_6', 'p_27_7', 'p_27_8',
'p_27_9', 'p_27_10', 'p_27_11', 'p_27_12', 'p_27_13', 'p_27_14', 'p_27
_15', 'p_27_16', 'p_27_17', 'p_27_18', 'p_27_19', 'p_27_20', 'p_27_2
1', 'p_27_22', 'p_27_23', 'p_27_24', 'p_27_25', 'p_27_26', 'p_27_27',
'p_27_28', 'p_27_29', 'p_27_30', 'p_27_31', 'p_28_0', 'p_28_1', 'p_28_
2', 'p_28_3', 'p_28_4', 'p_28_5', 'p_28_6', 'p_28_7', 'p_28_8', 'p_28_
9', 'p_28_10', 'p_28_11', 'p_28_12', 'p_28_13', 'p_28_14', 'p_28_15',
'p_28_16', 'p_28_17', 'p_28_18', 'p_28_19', 'p_28_20', 'p_28_21', 'p_2
8_22', 'p_28_23', 'p_28_24', 'p_28_25', 'p_28_26', 'p_28_27', 'p_28_2
8', 'p_28_29', 'p_28_30', 'p_28_31', 'p_29_0', 'p_29_1', 'p_29_2', 'p_
29_3', 'p_29_4', 'p_29_5', 'p_29_6', 'p_29_7', 'p_29_8', 'p_29_9', 'p_
29_10', 'p_29_11', 'p_29_12', 'p_29_13', 'p_29_14', 'p_29_15', 'p_29_1
6', 'p_29_17', 'p_29_18', 'p_29_19', 'p_29_20', 'p_29_21', 'p_29_22',
'p_29_23', 'p_29_24', 'p_29_25', 'p_29_26', 'p_29_27', 'p_29_28', 'p_2
9_29', 'p_29_30', 'p_29_31', 'p_30_0', 'p_30_1', 'p_30_2', 'p_30_3',
'p_30_4', 'p_30_5', 'p_30_6', 'p_30_7', 'p_30_8', 'p_30_9', 'p_30_10',
'p_30_11', 'p_30_12', 'p_30_13', 'p_30_14', 'p_30_15', 'p_30_16', 'p_3
0_17', 'p_30_18', 'p_30_19', 'p_30_20', 'p_30_21', 'p_30_22', 'p_30_2
3', 'p_30_24', 'p_30_25', 'p_30_26', 'p_30_27', 'p_30_28', 'p_30_29',
'p_30_30', 'p_30_31', 'p_31_0', 'p_31_1', 'p_31_2', 'p_31_3', 'p_31_
4', 'p_31_5', 'p_31_6', 'p_31_7', 'p_31_8', 'p_31_9', 'p_31_10', 'p_31
_11', 'p_31_12', 'p_31_13', 'p_31_14', 'p_31_15', 'p_31_16', 'p_31_1
7', 'p_31_18', 'p_31_19', 'p_31_20', 'p_31_21', 'p_31_22', 'p_31_23',
'p_31_24', 'p_31_25', 'p_31_26', 'p_31_27', 'p_31_28', 'p_31_29', 'p_3
1_30', 'p_31_31']
```

Out[8]:

	p_0_0	p_0_1	p_0_2	p_0_3	p_0_4	p_0_5	p_0_6	p_0_7	p_0_8	p_0_9	...	p_31_23	p_31_24
0	0	0	0	0	0	0	0	0	0	0	...	0	(
1	0	0	0	0	0	0	0	0	0	0	...	0	(
2	0	0	0	0	0	0	0	0	0	0	...	0	(
3	0	0	0	0	0	0	0	0	0	0	...	0	(
4	0	0	0	0	0	0	0	0	0	0	...	0	(

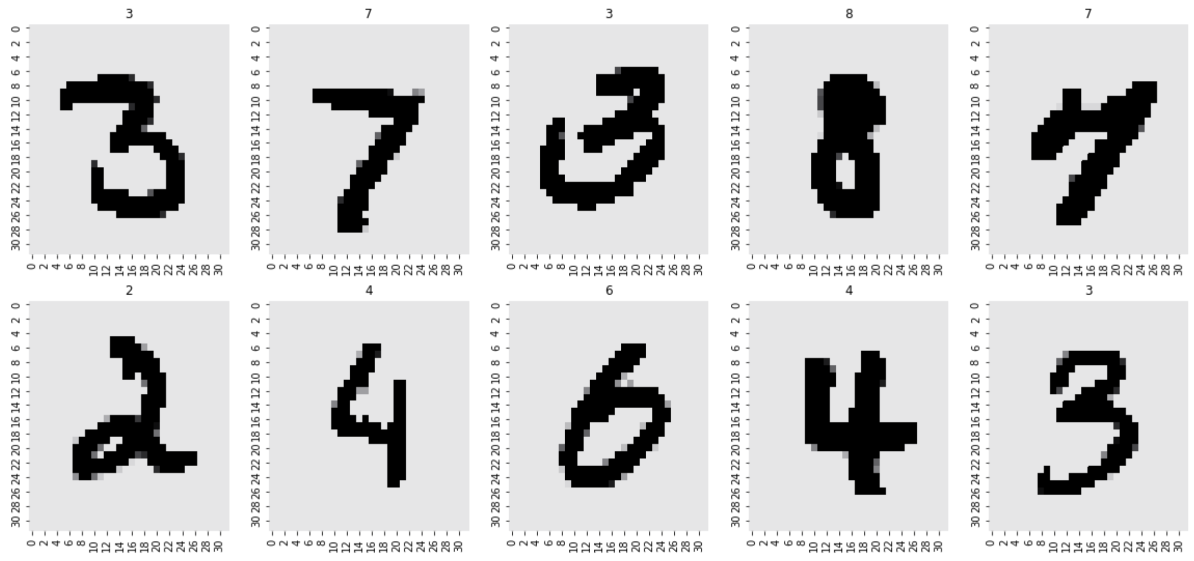
5 rows × 1025 columns

Data Exploration, Visualization and Correlations

Descriptive statistics, boxplots and histograms.

Some examples

Plot 10 randomly chosen digit images as 5 x 2 subplots.



Descriptive analysis

Build a DataFrame to make easier the exploratory analysis.

Out[10]:

	count	mean	std	min	25%	50%	75%	max
p_0_0	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_0_1	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_0_2	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_0_3	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_0_4	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
p_31_28	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_31_29	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_31_30	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
p_31_31	10000.0	0.00	0.0	0.0	0.0	0.0	0.0	0.0
target	10000.0	4.44	2.9	0.0	2.0	4.0	7.0	9.0

1025 rows × 8 columns

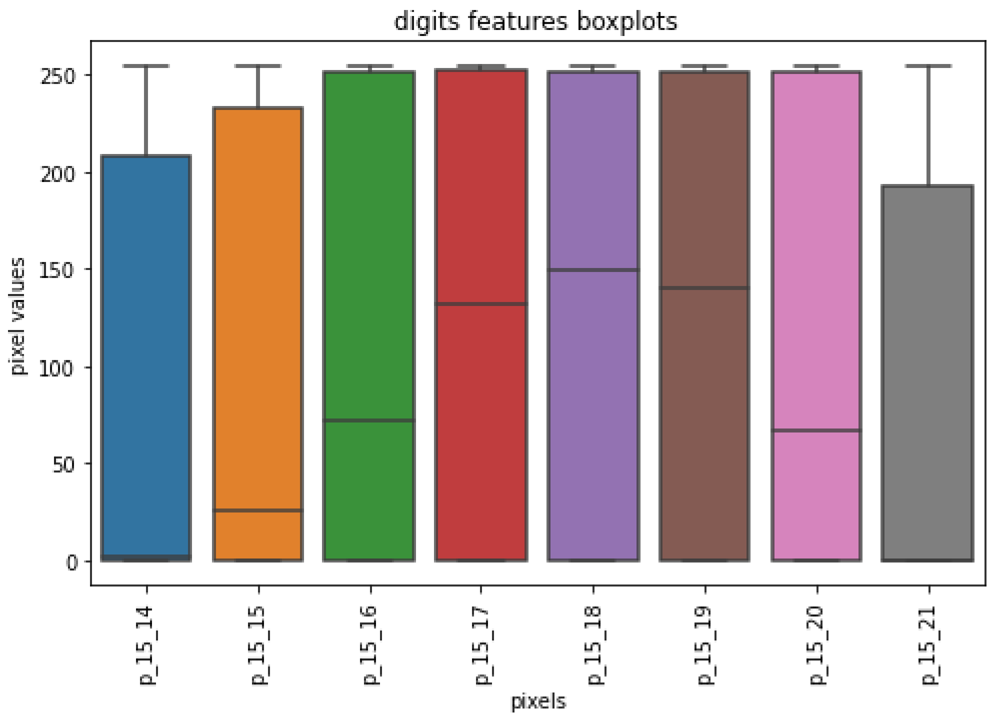
Describe the basic statistics of the pixels on the positions in the range `[494 : 502]` of the reshaped patterns.

Out[11]:

	count	mean	std	min	25%	50%	75%	max
p_15_14	10000.0	84.41	106.12	0.0	0.0	2.0	208.0	255.0
p_15_15	10000.0	96.37	109.08	0.0	0.0	25.0	233.0	255.0
p_15_16	10000.0	113.71	114.30	0.0	0.0	72.0	252.0	255.0
p_15_17	10000.0	127.31	112.31	0.0	0.0	132.0	253.0	255.0
p_15_18	10000.0	132.80	109.88	0.0	0.0	149.0	252.0	255.0
p_15_19	10000.0	128.99	112.47	0.0	0.0	140.0	252.0	255.0
p_15_20	10000.0	109.99	111.90	0.0	0.0	67.0	252.0	255.0
p_15_21	10000.0	80.92	104.67	0.0	0.0	0.0	193.0	255.0

Boxplots

Compute and display the boxplots of pixels in the range `[494 : 502]`.



Histograms and scatterplots

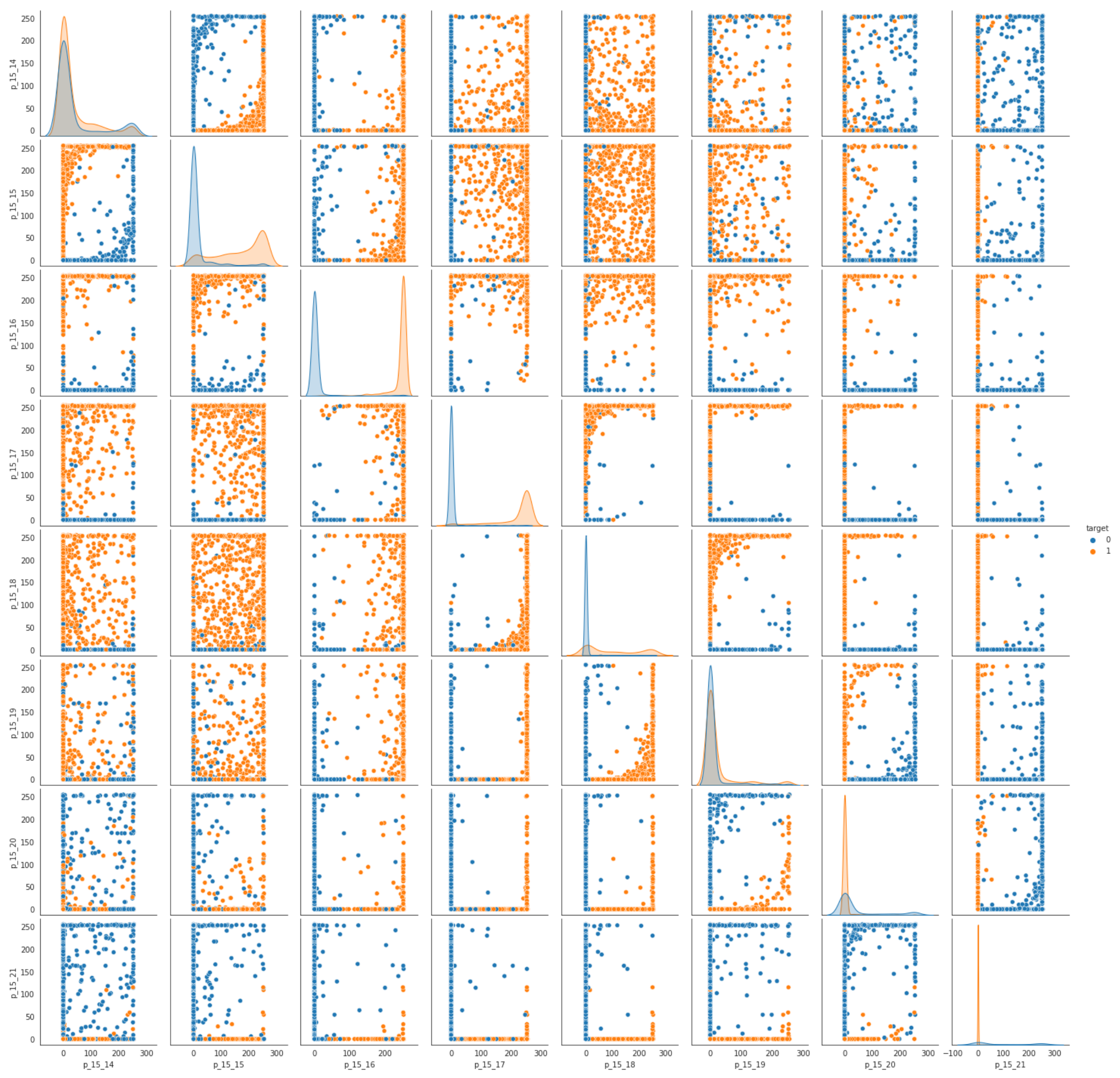
Plot pairplots and histograms over the previous pixel range using `sns.pairplot`. To do so select first two target digits (e.g., 2 and 7) and apply `pairplot` only on patterns from those two targets.

Out[13]:

	p_15_14	p_15_15	p_15_16	p_15_17	p_15_18	p_15_19	p_15_20	p_15_21	target
0	0	0	0	0	0	129	254	238	7
1	253	233	35	0	0	0	0	0	2
2	0	57	237	205	8	0	0	0	1
3	0	0	0	0	0	0	31	200	0
4	0	0	0	0	0	134	252	211	4

Out[14]:

	count	mean	std	min	25%	50%	75%	max
p_15_14	2115.0	43.98	78.39	0.0	0.0	0.0	53.5	255.0
p_15_15	2115.0	96.71	104.53	0.0	0.0	50.0	208.5	255.0
p_15_16	2115.0	133.51	122.65	0.0	0.0	211.0	253.0	255.0
p_15_17	2115.0	117.20	118.02	0.0	0.0	87.0	253.0	255.0
p_15_18	2115.0	56.72	89.26	0.0	0.0	0.0	98.0	255.0
p_15_19	2115.0	19.23	53.17	0.0	0.0	0.0	0.0	255.0
p_15_20	2115.0	22.89	62.94	0.0	0.0	0.0	0.0	255.0
p_15_21	2115.0	45.14	87.89	0.0	0.0	0.0	15.0	255.0
target	2115.0	0.54	0.50	0.0	0.0	1.0	1.0	1.0


p_15_14
p_15_15
p_15_16
p_15_17
p_15_18
p_15_19
p_15_20
p_15_21
target
0
1

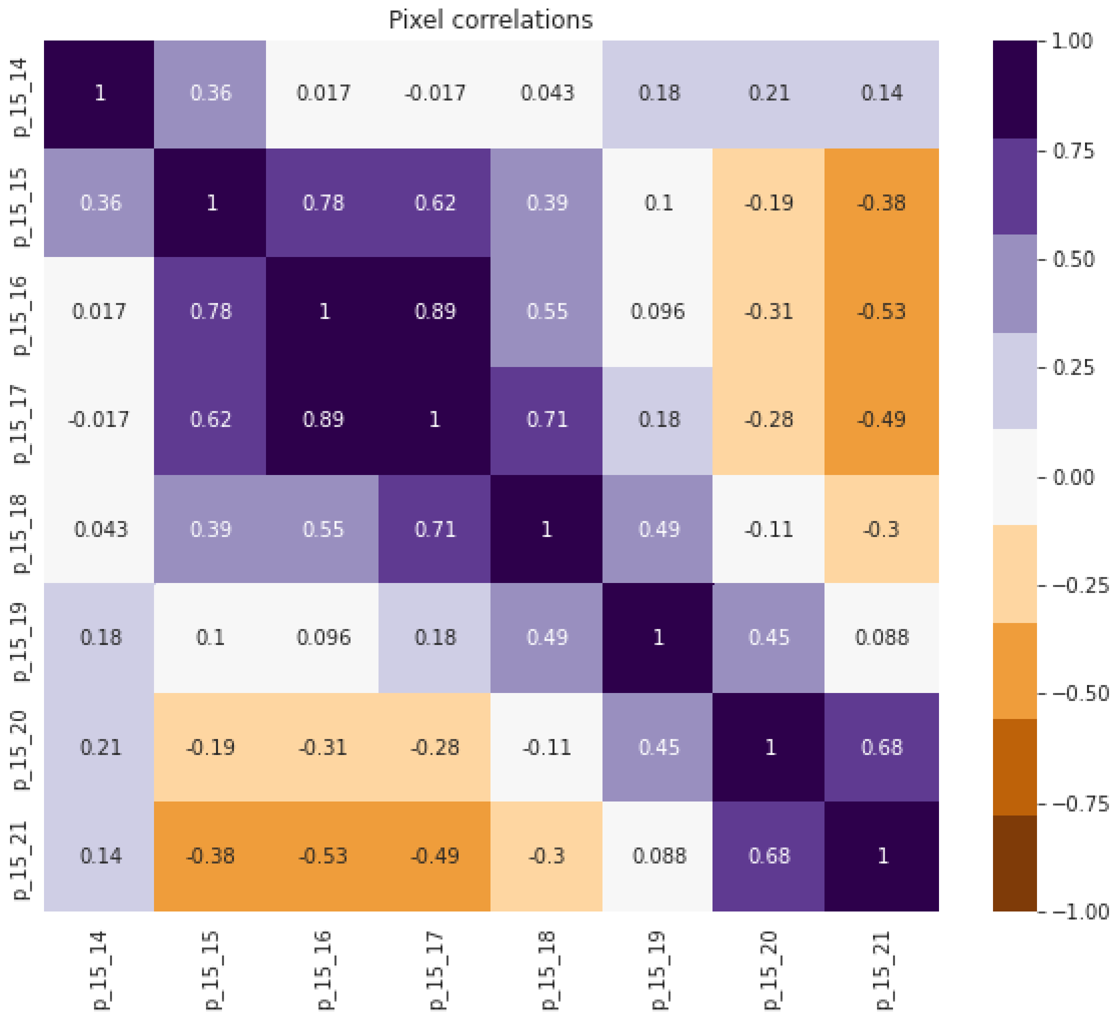
Correlations

Use the previous pixel range but drop the `target` column.

Use directly a heatmap to display the correlations.

```
Out[16]:
```

	p_15_14	p_15_15	p_15_16	p_15_17	p_15_18	p_15_19	p_15_20	p_15_21
p_15_14	1.000	0.363	0.017	-0.017	0.043	0.176	0.208	0.144
p_15_15	0.363	1.000	0.777	0.620	0.386	0.104	-0.194	-0.377
p_15_16	0.017	0.777	1.000	0.893	0.548	0.096	-0.314	-0.534
p_15_17	-0.017	0.620	0.893	1.000	0.712	0.175	-0.281	-0.494
p_15_18	0.043	0.386	0.548	0.712	1.000	0.486	-0.108	-0.301
p_15_19	0.176	0.104	0.096	0.175	0.486	1.000	0.449	0.088
p_15_20	0.208	-0.194	-0.314	-0.281	-0.108	0.449	1.000	0.679
p_15_21	0.144	-0.377	-0.534	-0.494	-0.301	0.088	0.679	1.000



Data Analysis Conclusions

Write down here a summary of your conclusions after the basic data analysis

1. Upon loading the data we analyzed the structure and distribution of the data. As there were no labels for the columns. So, we created the column labels for it and used describe function to observe other aspects of provided data.
2. We observed that each digit was written in several different styles. So we choose specific region of interset(pixels) in the whole image. We could see that the most interesting region was from the column 494 label(p_15_14) to 502 column label (*p_15_21*).
3. In the pair plot we could observe that at p_15_16, the pixel values for digit 0 and 1 have a very distinctive distribution where as at p_15_14 and at p_15_19 they have quite similar pixel distribution region.
4. Hence, for better analysis we created a new data frame. Using the heat map we observed that in the image the regions from P_15_15 to P_15_18 are highly corelated. Also, we tried to analyze the digits 0 & 1 and we found that using the pair plots and pixel correlation plots we could distinguish between 0 & 1. As for 0, the centeroid pixels are white and dark for 1.

Classiffiers

We are going to build a k-NN and an MLP classifier **over the test dataset**. But before working with any classifier, we split first the test dataset into a train-validation and a test subset.
Use for this the class `StratifiedShuffleSplit` from scikit-learn. Set the `test_size` parameter to either `0.5` or `0.75`.

Splitting the test dataset

```
Train samples:  7500
Test samples:   2500
```

k-NN Classification

Grid Search of Optimal Number of neighbors

```
GridSearch over a list of neighbors: [1, 2, 3, 5, 9, 17, 33]
Fitting 10 folds for each of 7 candidates, totalling 70 fits
```

Out[19]:

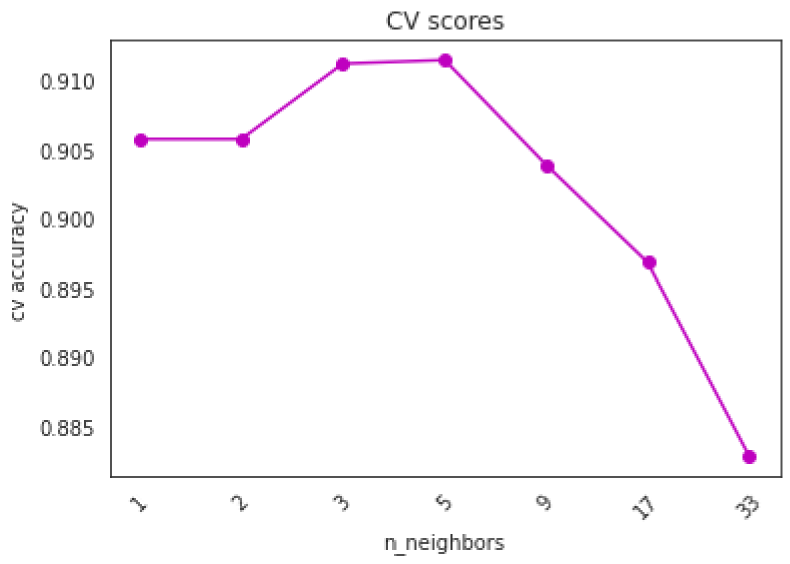
```
GridSearchCV(cv=StratifiedKFold(n_splits=10, random_state=None, shuffl
e=True),
             estimator=Pipeline(steps=[('std_sc', StandardScaler()),
                                       ('knn',
                                        KNeighborsClassifier(weights
='distance'))]),
             n_jobs=-1,
             param_grid={'knn__n_neighbors': [1, 2, 3, 5, 9, 17, 33]},
             return_train_score=True, scoring='accuracy', verbose=1)
```

Search Results

We first examine the test scores of the 5 best hyperparameters.

	param_knn__n_neighbors	mean_test_score
3	5	0.911467
2	3	0.911200
0	1	0.905733
1	2	0.905733
4	9	0.903867

We analyze the CV results to check whether the CV ranges used are correct.



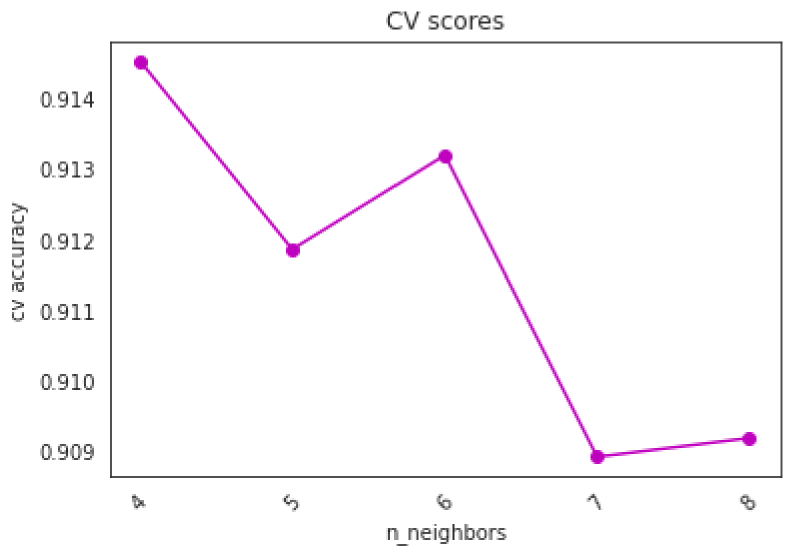
Fine tune selection of K

```
GridSearch over a list of neighbors: range(4, 9)
Fitting 10 folds for each of 5 candidates, totalling 50 fits
```

Out[22]:

```
GridSearchCV(cv=StratifiedKFold(n_splits=10, random_state=None, shuffl
e=True),
             estimator=Pipeline(steps=[('std_sc', StandardScaler()),
                                       ('knn',
                                        KNeighborsClassifier(weights
='distance'))]),
             n_jobs=-1, param_grid={'knn__n_neighbors': range(4, 9)},
             return_train_score=True, scoring='accuracy', verbose=1)
```

	param_knn__n_neighbors	mean_test_score
0	4	0.914533
2	6	0.913200
1	5	0.911867
4	8	0.909200
3	7	0.908933



Test Accuracy and Confusion Matrix

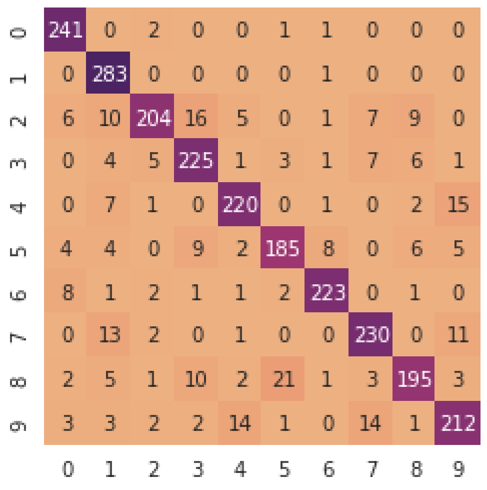
Precision and recall can also be defined for multiclass problems but we will skip them.

```
best_k: 4
acc: 0.887

confusion matrix:
 [[241   0   2   0   0   1   1   0   0   0]
 [  0 283   0   0   0   0   1   0   0   0]
 [  6  10 204  16   5   0   1   7   9   0]
 [  0   4   5 225   1   3   1   7   6   1]
 [  0   7   1   0 220   0   1   0   2  15]
 [  4   4   0   9   2 185   8   0   6   5]
 [  8   1   2   1   1   2 223   0   1   0]
 [  0  13   2   0   1   0   0 230   0  11]
 [  2   5   1  10   2  21   1   3 195   3]
 [  3   3   2   2  14   1   0  14   1 212]]
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc86a933f10>
```



	0	1	2	3	4	5	6	7	8	9
0	241	0	2	0	0	1	1	0	0	0
1	0	283	0	0	0	0	1	0	0	0
2	6	10	204	16	5	0	1	7	9	0
3	0	4	5	225	1	3	1	7	6	1
4	0	7	1	0	220	0	1	0	2	15
5	4	4	0	9	2	185	8	0	6	5
6	8	1	2	1	1	2	223	0	1	0
7	0	13	2	0	1	0	0	230	0	11
8	2	5	1	10	2	21	1	3	195	3
9	3	3	2	2	14	1	0	14	1	212

Conclusions on the k-NN classifier

1. Initally, for K-NN classifier we used 7 number of neighbours (1, 2, 3, 5, 9, 17, 33) and Euclidean distance measure for nearest neighbour classification and after CV grid search observed that the highest mean scores were between 3 to 9 neighbours, i.e we get the best result considering this number of neighbours.
2. The highest score belonged to $k = 5$, but since not all k values were tested, for further analysis we used another grid search with optimized 4 to 8 neighbour pixels and found that 4 neighbour had the highest mean score 0.9145, implying most of the pixels were surrounded by 4 neighbours of the right class.
3. Finally, we used 4 nearest neighbours to build a classifier and observed that the model showed an accuracy of 89%. It classified digit '1' most accurately. Also, it mostly gets confused when classifing between 5 and 8 & 4 and 9.

MLP Classifier

CV Hyperparametrization

Define an appropriate `MLPClassifier` and perform CV to select proper `alpha` and `hidden_layer_sizes` hyperparameters.

```
Fitting 2 folds for each of 12 candidates, totalling 24 fits
```

Out[27]:

```
GridSearchCV(cv=StratifiedKFold(n_splits=2, random_state=None, shuffle
=True),
             estimator=Pipeline(steps=[('std_sc', StandardScaler()),
                                       ('mlpc', MLPClassifier(max_iter
=1000))]),
             n_jobs=-1,
             param_grid={'mlpc__alpha': [1e-06, 1e-05, 0.0001, 0.001],
                         'mlpc__hidden_layer_sizes': [(128, 128), (12
8, 64),
                                                      (64, 32)]},
             return_train_score=True, scoring='accuracy', verbose=3)
```

Search Results

We first examine the test scores of the 5 best hyperparameters.

	param_mlpc__alpha	param_mlpc__hidden_layer_sizes	mean_test_score
7	0.0001	(128, 64)	0.933867
6	0.0001	(128, 128)	0.933067
1	0.000001	(128, 64)	0.931867
9	0.001	(128, 128)	0.931867
3	0.00001	(128, 128)	0.931200

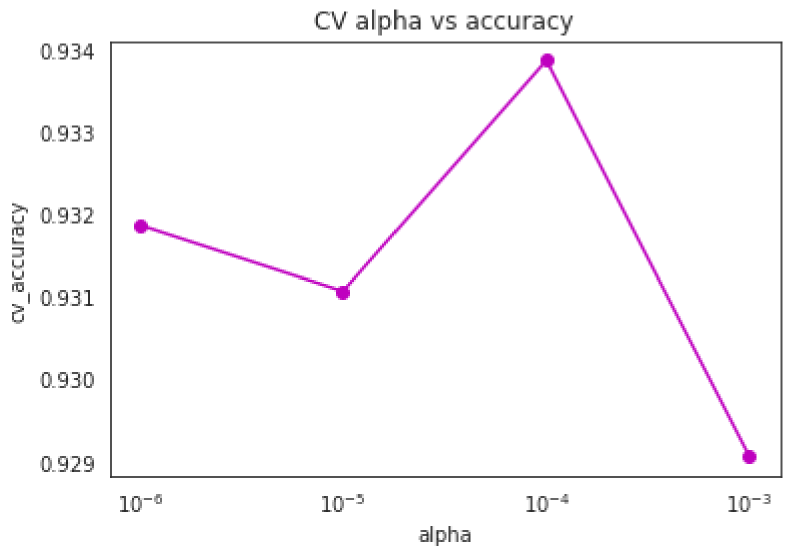
We analyze the CV results to check whether the CV ranges used are correct.

```
best alpha: 0.000100
alpha_min: 0.000001     alpha_max: 0.001000
best_hidden_layer_sizes (128, 64)
acc: 0.934

 1      0.931867
4      0.931067
7      0.933867
10     0.929067
Name: mean_test_score, dtype: float64
```

Out[29]:

```
[<matplotlib.lines.Line2D at 0x7fc869f47c50>]
```



Test MLPC Performance

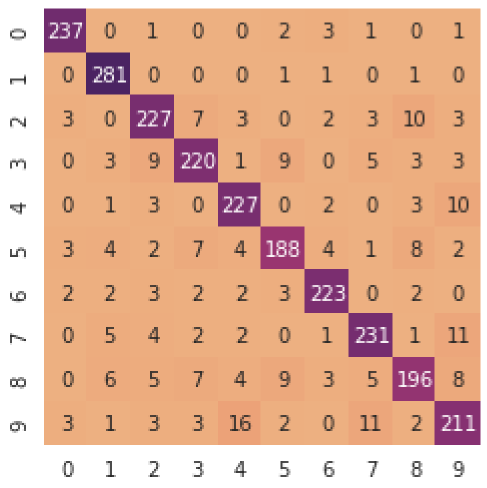
We check the test accuracy and confusion matrix.

```
acc: 0.896

confusion matrix:
 [[237   0   1   0   0   2   3   1   0   1]
 [  0 281   0   0   0   1   1   0   1   0]
 [  3   0 227   7   3   0   2   3  10   3]
 [  0   3   9 220   1   9   0   5   3   3]
 [  0   1   3   0 227   0   2   0   3  10]
 [  3   4   2   7   4 188   4   1   8   2]
 [  2   2   3   2   2   3 223   0   2   0]
 [  0   5   4   2   2   0   1 231   1  11]
 [  0   6   5   7   4   9   3   5 196   8]
 [  3   1   3   3  16   2   0  11   2 211]]
```

```
Out[31]:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc869d68b10>
```



Conclusions on the MLP classifier

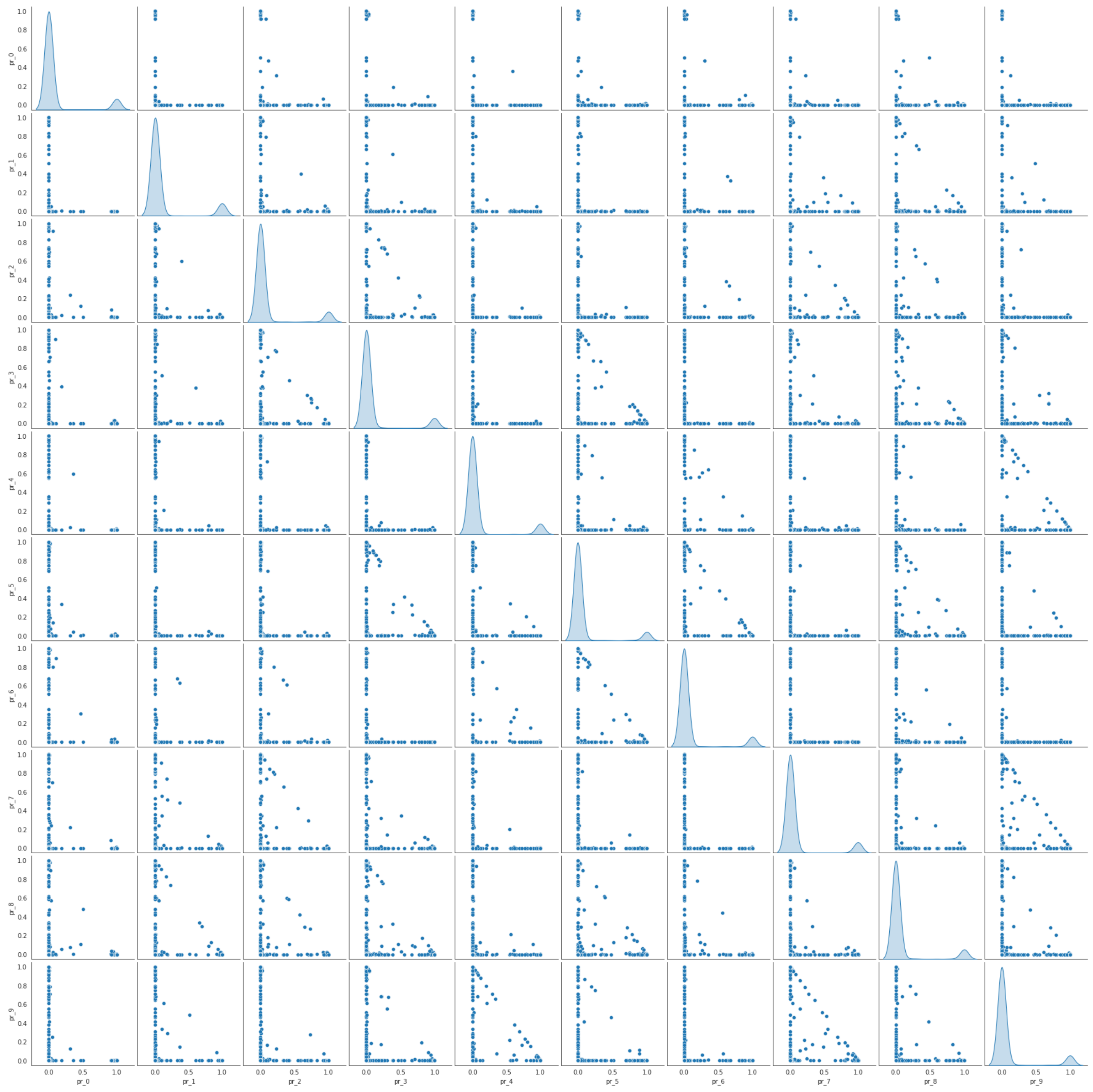
1. The K-NN classifier results were good but not very impressive. So, in order to acheive better results we built a classifier using Multi-layer perceptron. For the mlp classifier we found that adam was a decent solver for weight optimization. It refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba.
2. We used 'relu' rectified linear unit which returns f(x) = max(0, x) as the activation function. The 2 fold CV and the following number of layers (128, 128), (128,64), (64, 32) were complete in 12 iterations. 3. We observed that the best hidden layer size was (128,64) with a learning rate $alpha$ of 0.0001 which gave a mean test score of 0.9338.
3. However, we found that the score of layer size (128,128) with a learning rate of 0.0001 was 0.9330. We assumed that the (128,128) layer would show better results due to more denser structure, but it was not the case.
4. The MLPC out performed the K-NN classifier by 0.9%. MLPC showed an accuracy of 89.6% and proved that using a MLP classifier we can acheive slightly better results. The confusion matrix for the MLP also shows that the decisions made by it are pretty accurate.

Predicting probabilities

We compute class probabilities over the test subset and pairplot them over the 10 classes.

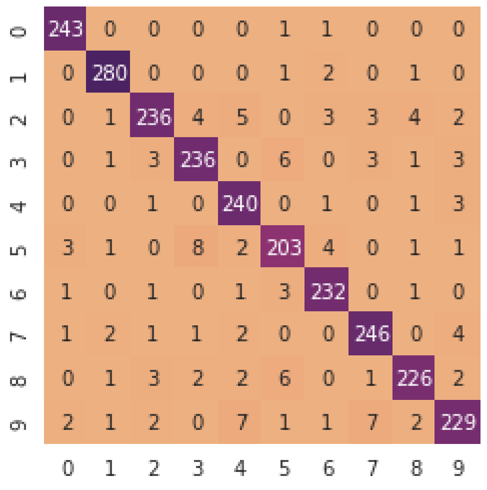
```
Out[32]:
```

	count	mean	std	min	25%	50%	75%	max
target	2500.0	4.44	2.90	0.00	2.0	4.0	7.0	9.0
max_prob	2500.0	0.98	0.07	0.31	1.0	1.0	1.0	1.0
pred_target	2500.0	4.42	2.89	0.00	2.0	4.0	7.0	9.0



```
Out[34]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fc86815a310>
```



	0	1	2	3	4	5	6	7	8	9
0	243	0	0	0	0	1	1	0	0	0
1	0	280	0	0	0	1	2	0	1	0
2	0	1	236	4	5	0	3	3	4	2
3	0	1	3	236	0	6	0	3	1	3
4	0	0	1	0	240	0	1	0	1	3
5	3	1	0	8	2	203	4	0	1	1
6	1	0	1	0	1	3	232	0	1	0
7	1	2	1	1	2	0	0	246	0	4
8	0	1	3	2	2	6	0	1	226	2
9	2	1	2	0	7	1	1	7	2	229

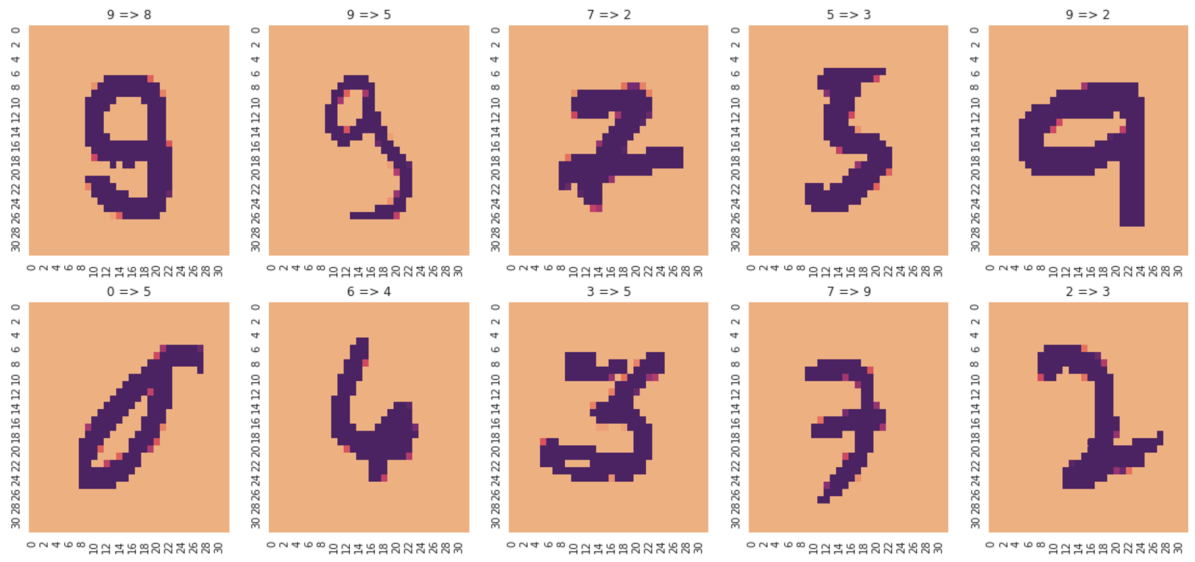
Conclusions on the probability pairplots

1. The pair plot shows a clear picture of the model's performance. As we can see in the pair plots the model did a nice job of classifying digits properly. However, there are a few cases (around 10%) where the MLPC is confused as the digits were written in a very similar way and the pixels on the diagnoal takes a value of 0.5 for probability distribution.
2. The confusion matrix displays the number of correctly classified and misclassified digits. As we can see that the number of misclassified digits had dropped significantly for MLPC incomparison to K-NN. It was able to differentiate between 5 & 8 distinctively. However, for MLPC it's still sometimes difficult to differentiate between 4 & 9 as they are written in a really similar way.

What went wrong?

Probably your classification results are not very impressive.

As a first step to interpret them, find 10 cases of wrongly classified numbers, plot them and discuss your findings.



Out[35]:

	label	pred_label	confidence
0	9	8	0.478492
1	9	5	0.886433
2	7	2	0.998427
3	5	3	0.893568
4	9	2	0.973790
5	0	5	0.989275
6	6	4	0.998197
7	3	5	0.862002
8	7	9	0.551302
9	2	3	0.782943

Discussion

1. As seen in the above figure and table the model sometimes misclassify the digits with a high confidence. This implies that the model has incorrectly learned the pixel arrangment for digits in some extreme cases.
2. One of the possible reason for that is, as we are using a smaller data set. The model wasn't able to learn all the possible scenarios in which the digits can be written.
3. Other, plausible explaination for the misclassification can be the similarity in the way of writting the digits. For example, some times the distribution of pixels for digit 4, is very similar to that of pixel distribution on digit 9. Implying that 4 and 9 were written in a very similar way. Other example of such case is digit 5 and digit 8.

How to improve your results?

1. Using the $K-NN$ classifier we got an accuracy of 88.7%. Inorder to acheive, better results we tried MLP classifier and our results improved by 0.9% and we got an final accuracy of 89.6%. The training time for the model was less than 10 mins with (128,64) hidden layer size.
2. We can improve the obtained results probably by using a large dataset for training. As using a larger dataset will help our model to learn more variations in writing of digits.
3. Much better results can be obtained by doing a more exhaustive grid search in order to find a optimum set of parameter. However, we weren't able to perform a extensive grind search due to GPU constraints.
4. We can use decision tree models like random forest or ensemble models like XGboost or adaboost for improved results.
5. Finally, to improve the overall classification accuracy CNN models can also be implemented.