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

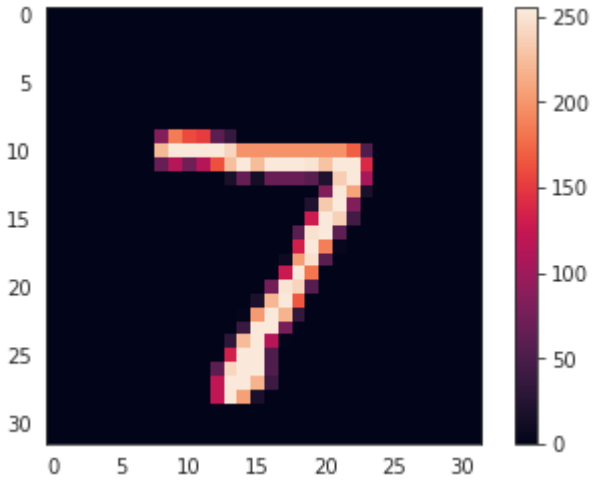
- Student `Mohammed Hussein` has worked over the model implementations, KNN and MLP: hypertuning and design.
- Student `Sebastian Cajas` has worked over the Data exploration, documentation/testing of the models.

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

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'])
1. DESCR: MNIST data set from Lecun site: http://yann.lecun.com/exdb/
mnist/
2. Target test: (10000,)
3. data_test_shape: (10000, 1024)


(10000, 1024)
(10000, 1)


Series([], dtype: int64)
```

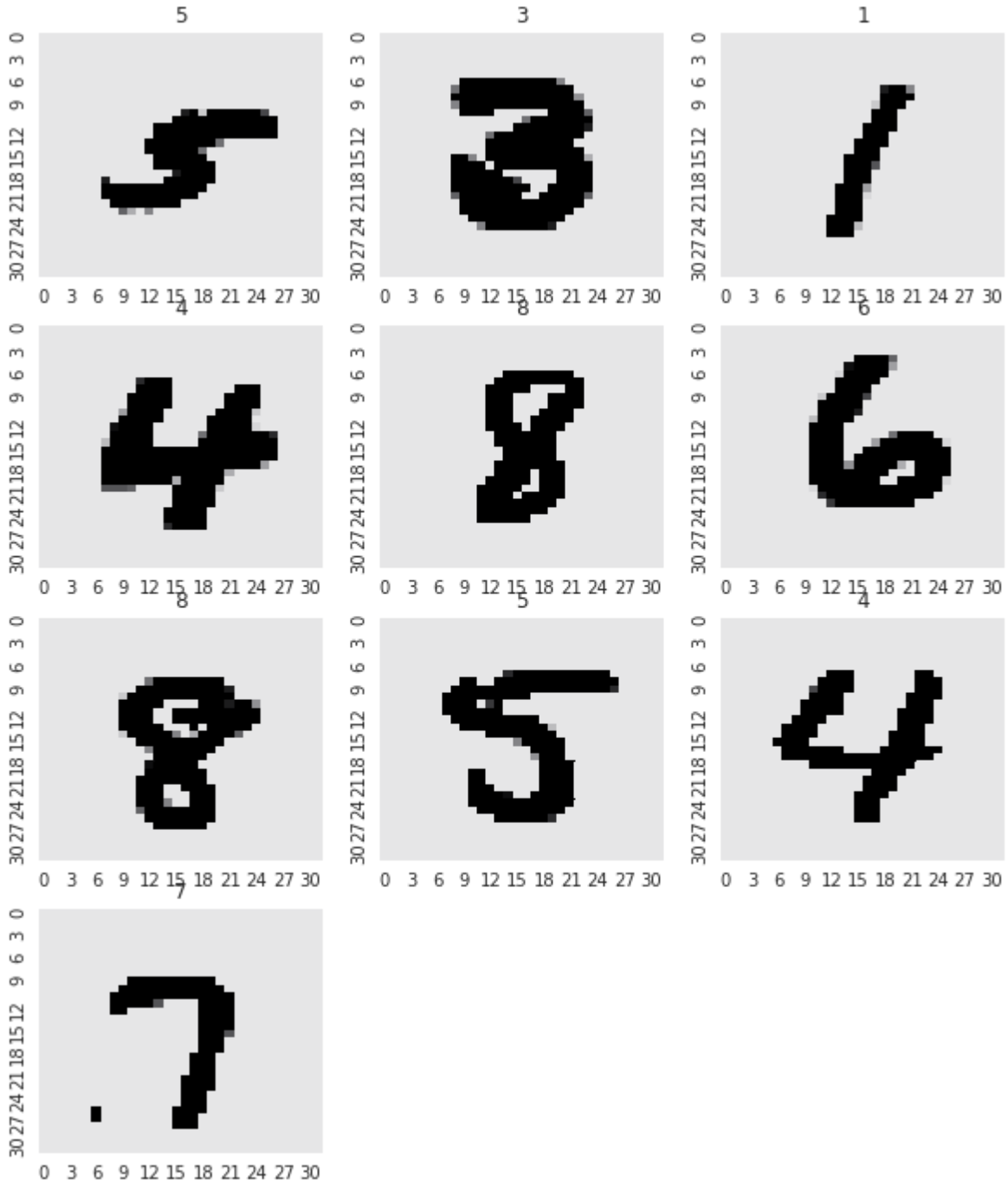


Data Exploration, Visualization and Correlations

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.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 1025 entries, 0 to target
dtypes: uint8(1025)
memory usage: 9.8 MB
```

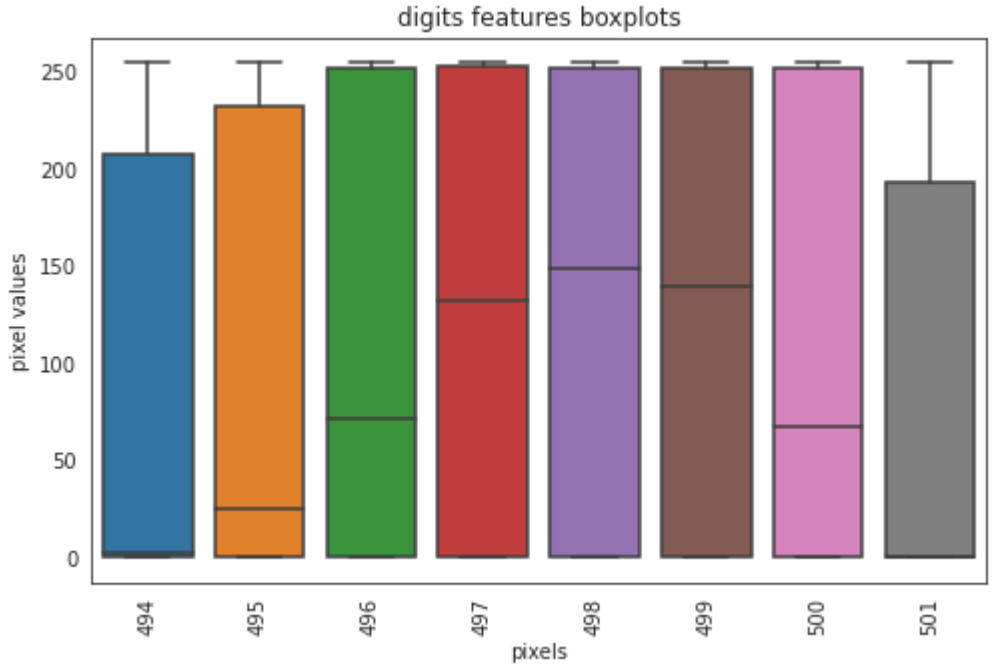
Out[36]:

	494	495	496	497	498	499	1(
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	
mean	84.414300	96.372400	113.705900	127.309600	132.804900	128.993700	
std	106.120561	109.081183	114.304354	112.306614	109.878282	112.465829	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	2.000000	25.000000	72.000000	132.000000	149.000000	140.000000	
75%	208.000000	233.000000	252.000000	253.000000	252.000000	252.000000	
max	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	

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

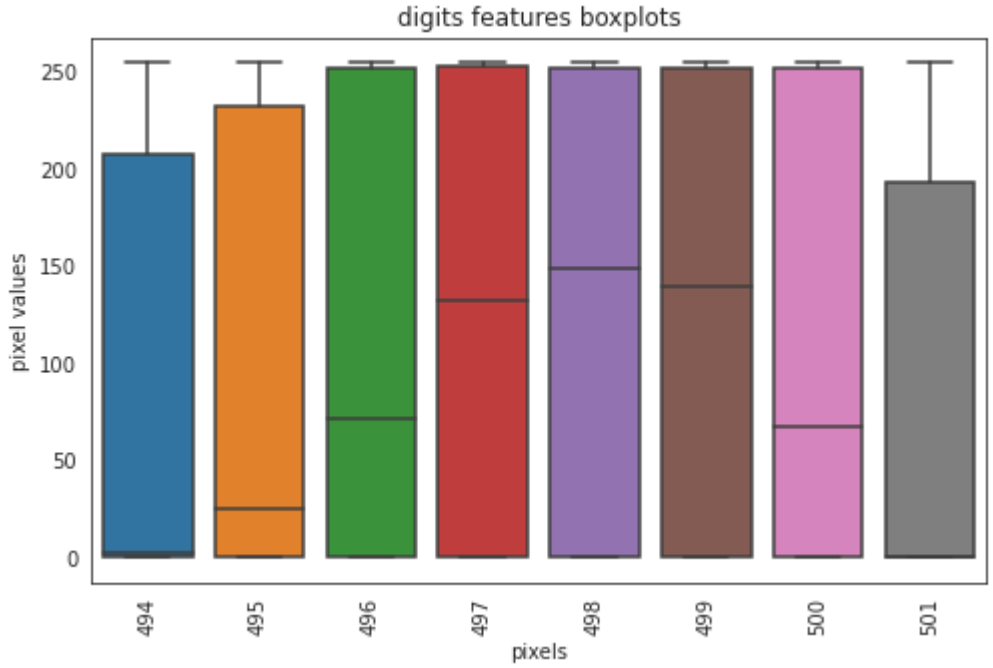
Boxplots

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



Out[38]:

	494	495	496	497	498	499	500
0	0	0	0	0	0	129	254
1	253	233	35	0	0	0	0
2	0	57	237	205	8	0	0
3	0	0	0	0	0	0	31
4	0	0	0	0	0	134	252
5	0	111	254	254	132	0	0
6	253	253	253	254	253	253	219
7	0	0	234	253	254	135	0
8	0	0	0	0	0	0	0
9	178	178	169	210	251	231	254



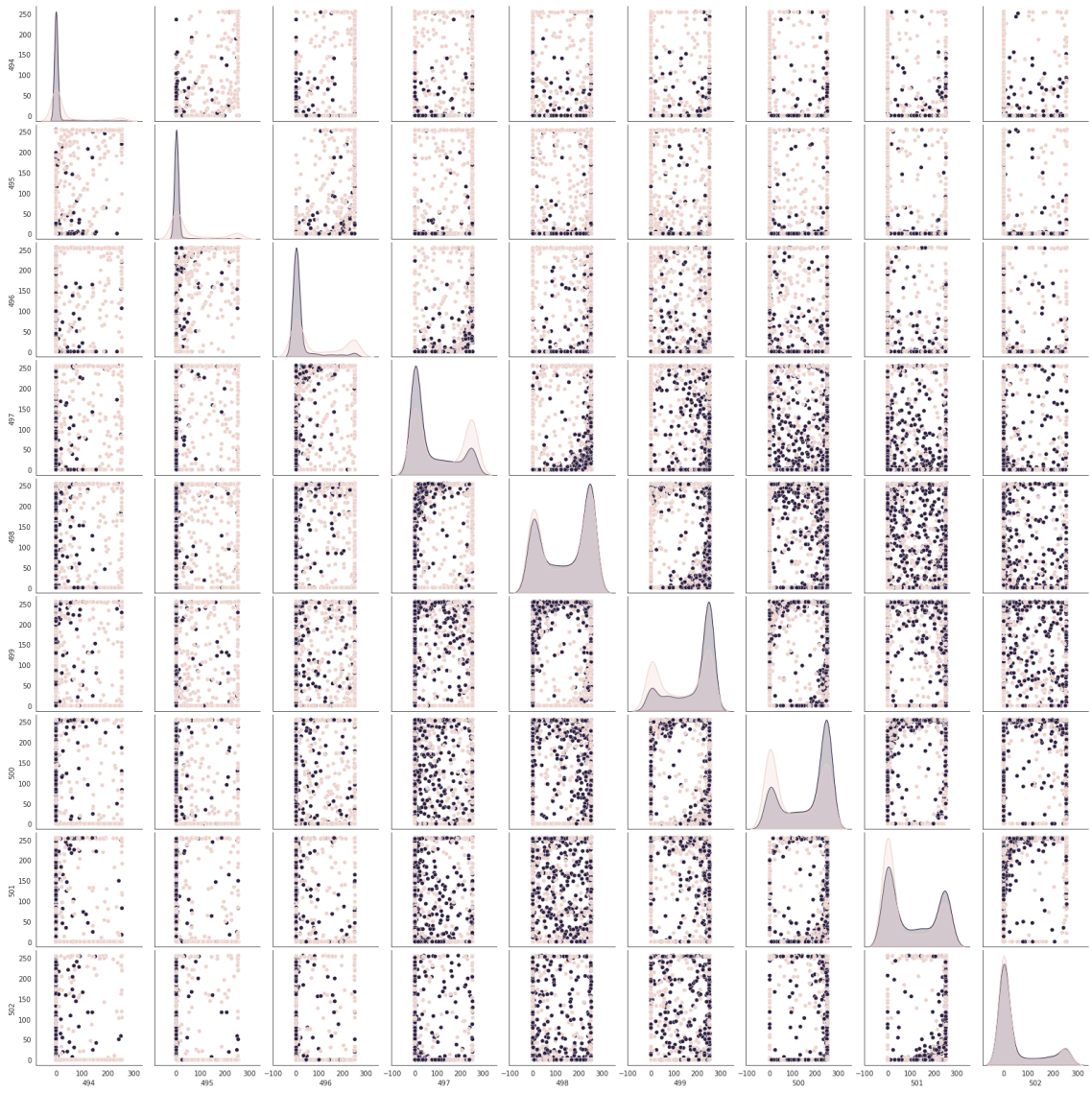
Histograms and scatterplots

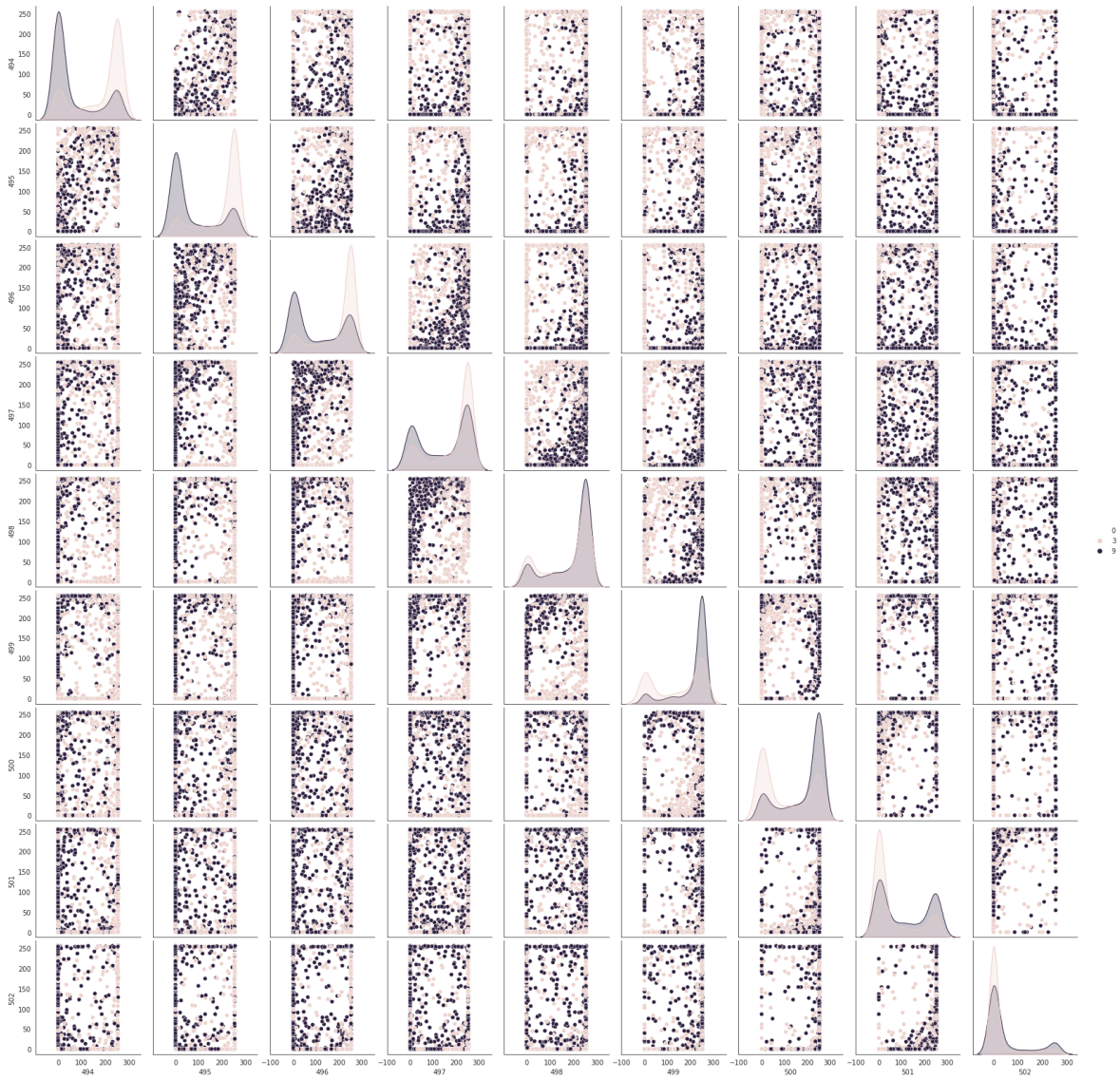
Plot pairplots and histograms over the previous pixel range using `sns.pairplot` .
To do so select first two target digits (e.g., 2 and 7) and apply `pairplot` only on patterns from those two targets.

Out[41]:

	494	495	496	497	498	499	500	501	502	0
0	0	0	0	0	0	129	254	238	44	7
1	253	233	35	0	0	0	0	0	0	2
17	0	0	0	0	0	0	115	254	254	7
26	0	0	0	0	0	162	253	151	0	7
34	0	0	0	13	241	252	252	126	24	7
...	...	...	...	...	...	...	...	...	...	...
9979	0	0	0	0	0	52	232	252	239	7
9980	1	85	198	253	253	253	197	253	253	2
9985	0	0	13	176	253	253	224	0	0	2
9990	0	0	44	240	253	253	227	0	0	7
9995	0	0	0	191	255	255	255	0	0	2

2060 rows × 10 columns


494
495
496
497
498
499
500
501
502
0
2
7



Correlations

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

Use directly a heatmap to display the correlations.

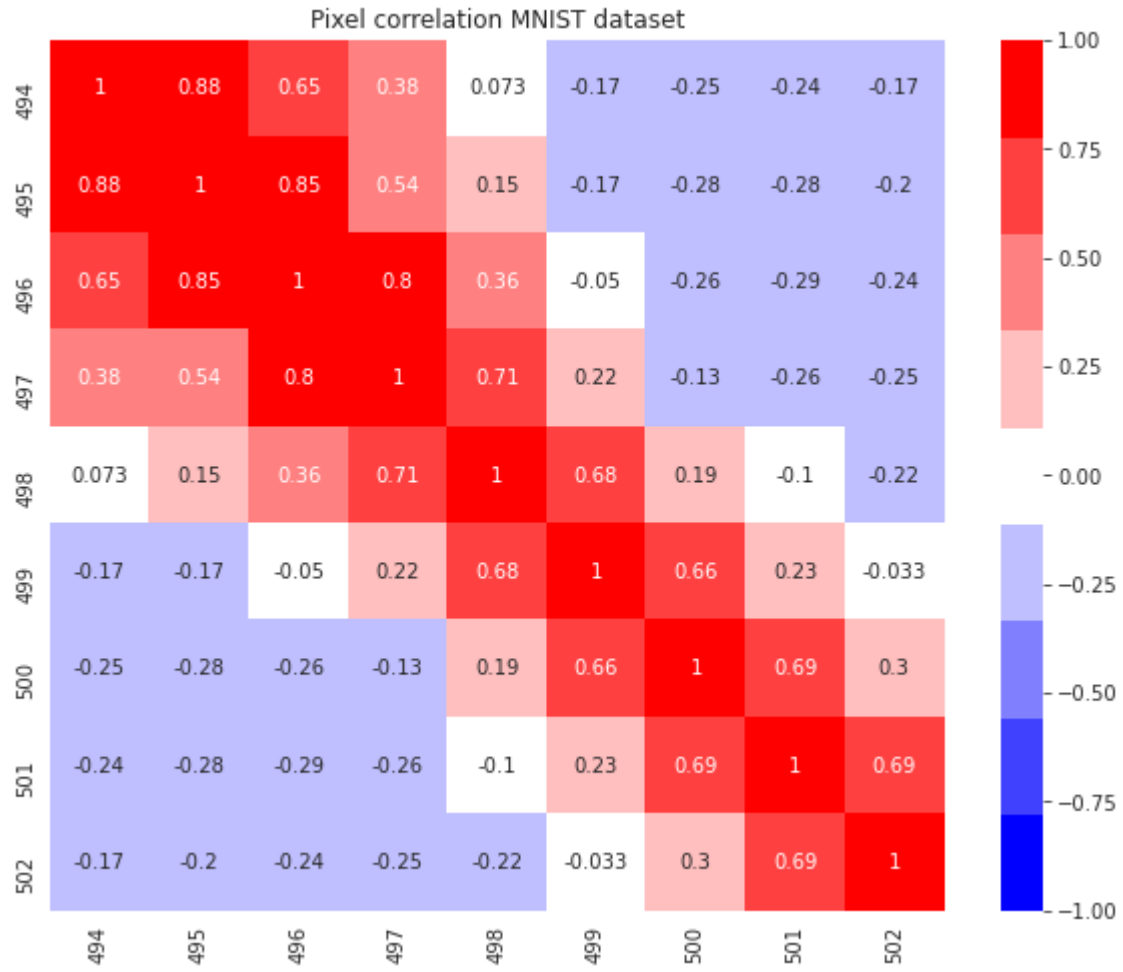
Out[44]:

	494	495	496	497	498	499	500	501	502
7	0	0	234	253	254	135	0	0	0
9	178	178	169	210	251	231	254	254	254
12	0	0	0	5	136	252	252	64	0
16	0	0	0	0	6	219	253	241	31
18	253	253	253	253	253	253	253	244	89
...	...	...	...	...	...	...	...	...	...
9973	0	0	0	17	103	236	253	224	39
9975	0	0	17	185	254	254	62	0	0
9986	45	200	253	253	253	133	0	0	0
9992	0	0	0	112	229	253	253	254	253
9996	0	77	185	230	254	254	254	239	45

2019 rows × 9 columns

Out[45]:

	494	495	496	497	498	499	500	501	502
494	1.000	0.878	0.652	0.381	0.073	-0.174	-0.250	-0.243	-0.167
495	0.878	1.000	0.853	0.544	0.151	-0.166	-0.283	-0.281	-0.202
496	0.652	0.853	1.000	0.803	0.362	-0.050	-0.261	-0.292	-0.238
497	0.381	0.544	0.803	1.000	0.706	0.218	-0.134	-0.256	-0.250
498	0.073	0.151	0.362	0.706	1.000	0.679	0.193	-0.103	-0.219
499	-0.174	-0.166	-0.050	0.218	0.679	1.000	0.664	0.233	-0.033
500	-0.250	-0.283	-0.261	-0.134	0.193	0.664	1.000	0.687	0.300
501	-0.243	-0.281	-0.292	-0.256	-0.103	0.233	0.687	1.000	0.690
502	-0.167	-0.202	-0.238	-0.250	-0.219	-0.033	0.300	0.690	1.000



Data Analysis Conclusions

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

Remarks:

The columns 494-500 have 0-255 set-values on grayscale, since the images were set on 1 channel, this means that there is a thresholding change from black to white or grayscale. The mean starts in 86, increases up to 129 and decreases again until 55. Meaning that in most of the images there is a high information content fo white color in this area. Considering the standard deviation, the 50% and 75% percentil, this areas are mostly white on all images.

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

Out[535]: 5

```
Train Data shape:
(5000, 1024) (5000, 1)

Test Data shape:
(5000, 1024) (5000, 1)
```

k-NN Classification

Grid Search of Optimal Number of neighbors

Search Results

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

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] n_neighbors=1
..................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurre
nt workers.

[CV] ................................... n_neighbors=1, total=   8.3
s
[CV] n_neighbors=1
..................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    8.3s remainin
g:    0.0s
```

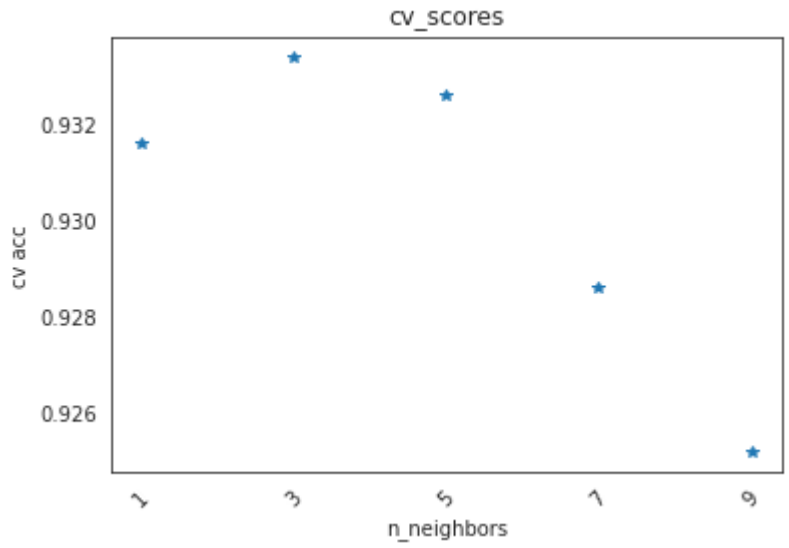
```
[CV] ..................................... n_neighbors=1, total=   8.7
s
[CV] n_neighbors=1
.................................................
[CV] ..................................... n_neighbors=1, total=   8.8
s
[CV] n_neighbors=1
.................................................
[CV] ..................................... n_neighbors=1, total=   9.0
s
[CV] n_neighbors=1
.................................................
[CV] ..................................... n_neighbors=1, total=   8.8
s
[CV] n_neighbors=3
.................................................
[CV] ..................................... n_neighbors=3, total=   9.0
s
[CV] n_neighbors=3
.................................................
[CV] ..................................... n_neighbors=3, total=   9.0
s
[CV] n_neighbors=3
.................................................
[CV] ..................................... n_neighbors=3, total=   8.8
s
[CV] n_neighbors=3
.................................................
[CV] ..................................... n_neighbors=3, total=   8.7
s
[CV] n_neighbors=3
.................................................
[CV] ..................................... n_neighbors=3, total=   8.8
s
[CV] n_neighbors=5
.................................................
[CV] ..................................... n_neighbors=5, total=   9.2
s
[CV] n_neighbors=5
.................................................
[CV] ..................................... n_neighbors=5, total=   9.1
s
[CV] n_neighbors=5
.................................................
[CV] ..................................... n_neighbors=5, total=   9.1
s
[CV] n_neighbors=5
.................................................
[CV] ..................................... n_neighbors=5, total=   8.9
s
[CV] n_neighbors=5
.................................................
[CV] ..................................... n_neighbors=5, total=   8.9
s
[CV] n_neighbors=7
.................................................
[CV] ..................................... n_neighbors=7, total=   9.0
```

```
s
[CV] n_neighbors=7
.................................................
[CV] ..................................... n_neighbors=7, total=   9.2
s
[CV] n_neighbors=7
.................................................
[CV] ..................................... n_neighbors=7, total=   9.4
s
[CV] n_neighbors=7
.................................................
[CV] ..................................... n_neighbors=7, total=   9.4
s
[CV] n_neighbors=7
.................................................
[CV] ..................................... n_neighbors=7, total=   9.7
s
[CV] n_neighbors=9
.................................................
[CV] ..................................... n_neighbors=9, total=   9.9
s
[CV] n_neighbors=9
.................................................
[CV] ..................................... n_neighbors=9, total=  10.1
s
[CV] n_neighbors=9
.................................................
[CV] ..................................... n_neighbors=9, total=   9.9
s
[CV] n_neighbors=9
.................................................
[CV] ..................................... n_neighbors=9, total=   9.9
s
[CV] n_neighbors=9
.................................................
[CV] ..................................... n_neighbors=9, total=   9.9
s

[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:  3.8min finished
```

Out[57]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 3, 5, 7, 9]}, scoring='ac
curacy',
             verbose=2)
```

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

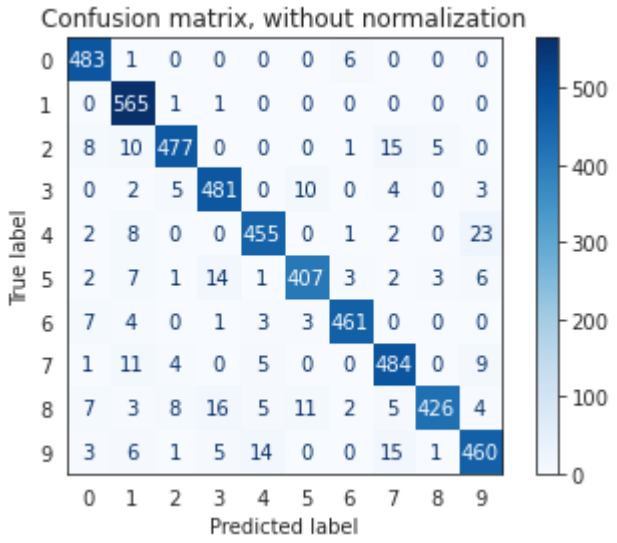
```
1  :  mean_test_score    93.34
Name: 1, dtype: float64 {'n_neighbors': 1}
2  :  mean_test_score    93.26
Name: 2, dtype: float64 {'n_neighbors': 3}
3  :  mean_test_score    93.16
Name: 0, dtype: float64 {'n_neighbors': 5}
4  :  mean_test_score    92.86
Name: 3, dtype: float64 {'n_neighbors': 7}
5  :  mean_test_score    92.52
Name: 4, dtype: float64 {'n_neighbors': 9}
```



Test Accuracy and Confusion Matrix

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

```
accuracy= 93.98%
KNeighborsClassifier(n_neighbors=3)
----------------

Confusion matrix, without normalization
[[483   1   0   0   0   0   6   0   0   0]
 [  0 565   1   1   0   0   0   0   0   0]
 [  8  10 477   0   0   0   1  15   5   0]
 [  0   2   5 481   0  10   0   4   0   3]
 [  2   8   0   0 455   0   1   2   0  23]
 [  2   7   1  14   1 407   3   2   3   6]
 [  7   4   0   1   3   3 461   0   0   0]
 [  1  11   4   0   5   0   0 484   0   9]
 [  7   3   8  16   5  11   2   5 426   4]
 [  3   6   1   5  14   0   0  15   1 460]]
```




Pred: 9 Pred: 9 Pred: 3 Pred: 0 Pred: 3 Pred: 3 Pred: 8 Pred: 4 Pred: 9 Pred: 3
Pred: 3 Pred: 4 Pred: 7 Pred: 0 Pred: 3 Pred: 2 Pred: 6 Pred: 5 Pred: 1 Pred: 7
Pred: 0 Pred: 9 Pred: 9 Pred: 8 Pred: 1 Pred: 9 Pred: 1 Pred: 1 Pred: 2 Pred: 6
Pred: 3 Pred: 1 Pred: 0 Pred: 7 Pred: 1 Pred: 8 Pred: 6 Pred: 0 Pred: 4 Pred: 6
Pred: 1 Pred: 8 Pred: 3 Pred: 5 Pred: 3 Pred: 2 Pred: 7 Pred: 0 Pred: 2 Pred: 7
Pred: 4 Pred: 7 Pred: 0 Pred: 2 Pred: 9 Pred: 6 Pred: 5 Pred: 4 Pred: 3 Pred: 6
Pred: 8 Pred: 5 Pred: 8 Pred: 9 Pred: 4 Pred: 2 Pred: 7 Pred: 7 Pred: 4 Pred: 0
Pred: 5 Pred: 6 Pred: 2 Pred: 6 Pred: 6 Pred: 9 Pred: 5 Pred: 6 Pred: 9 Pred: 4
Pred: 1 Pred: 6 Pred: 5 Pred: 4 Pred: 8 Pred: 5 Pred: 4 Pred: 5 Pred: 2 Pred: 7
Pred: 4 Pred: 7 Pred: 6 Pred: 5 Pred: 8 Pred: 6 Pred: 1 Pred: 9 Pred: 9 Pred: 8

Conclusions on the k-NN classifier

Discuss your results here

° From the Grid search results we have considered 5 different numbers, from which we have extracted the mean test score for each case and have ordered in desceding order, weighted by its score values. From the graph above, we can see that when we have 3 neighbors we obtain the best accuracy. This means that for the KNN, we require 3 neighboring pixels to obtain the best pixel prediction, in this case, to represent a pixel that fits with the corresponding number structure, either 0 or 255.

° From the confusion_matrix we can deduce the following statements:

```
- # 1 is the best predicted true label.
- # 5 is the worest predicted true label.
```

° It can also be seen that there are slight biases over pixel values, for example, on the number 2 we have that there are 477 true positive labels, but there are also 39 missmatches. Continuing with these observations, the number 1, the best predicted true label, also has some mismatches were it was confused as false positive, which explaines why the prediction is not perfect. On the other hand we have the worst predicted true label, the number 5, on which there were 39 mismatches with other labels and therefore the lefting predictions were totally missed, meaning that there were as well False Negatives during the prediction task.

° From here we can see that the number 5 is confused with the number 9 in one ocasion.

MLP Classifier

CV Hyperparametrization

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

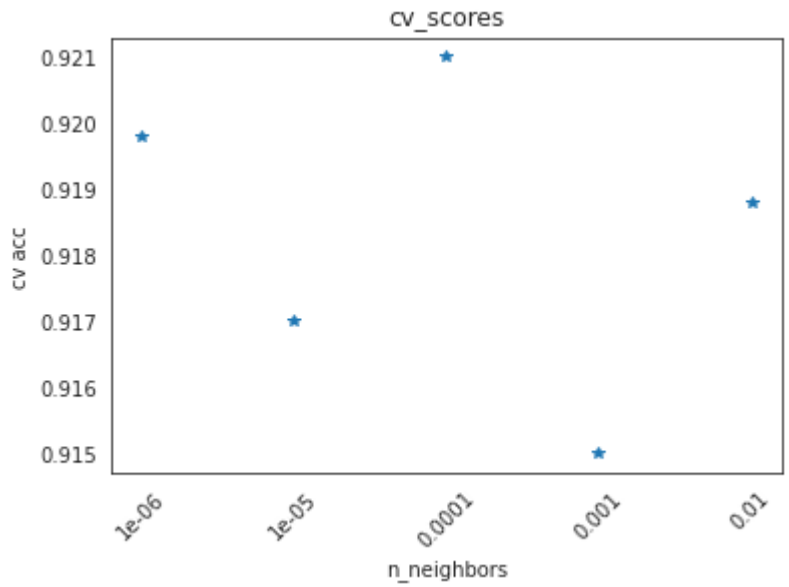
```
(5000, 1024)
Fitting 10 folds for each of 20 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  8.9min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:  9.2min finishe
d

mlp_grid_search_time: 9.6f
```

Search Results

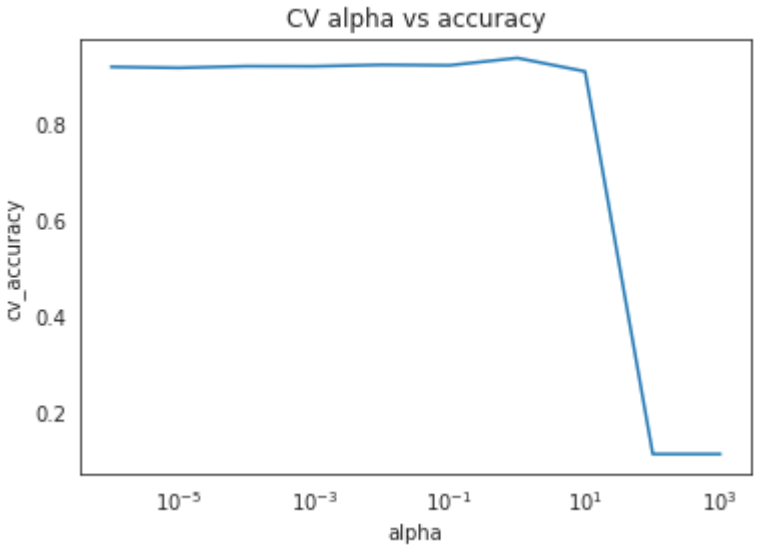
We first examine the test scores of the 5 best hyperparameters.

	param_mlpc__alpha	param_mlpc__hidden_layer_sizes	mean_test_score
13	1.0	(20, 20)	0.9354
12	1.0	(20,)	0.9304
10	0.1	(20,)	0.9240
6	0.001	(20,)	0.9222
9	0.01	(20, 20)	0.9212

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



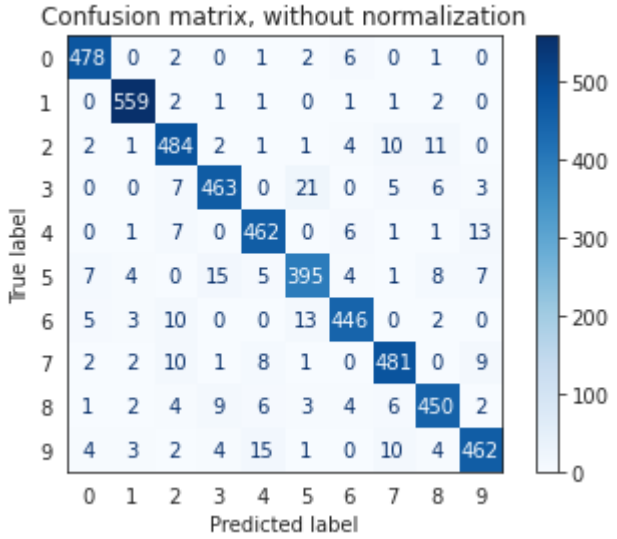
```
best alpha: 1.000000
alpha_min: 0.000001     alpha_max: 1000.000000
best_hidden_layer_sizes (20, 20)
acc: 0.935
```



Test MLPC Performance

We check the test accuracy and confusion matrix.

```
Accuarcy=  0.936
------------------------
Confusion matrix, without normalization
[[478   0   2   0   1   2   6   0   1   0]
 [  0 559   2   1   1   0   1   1   2   0]
 [  2   1 484   2   1   1   4  10  11   0]
 [  0   0   7 463   0  21   0   5   6   3]
 [  0   1   7   0 462   0   6   1   1  13]
 [  7   4   0  15   5 395   4   1   8   7]
 [  5   3  10   0   0  13 446   0   2   0]
 [  2   2  10   1   8   1   0 481   0   9]
 [  1   2   4   9   6   3   4   6 450   2]
 [  4   3   2   4  15   1   0  10   4 462]]
```



Conclusions on the MLP classifier

Discuss your results here

Regarding the confusion matrix:

° The true label 1 has 559 true positives on MLP model, while on the KNN there were 565 true positives.

° The worst predcted true label is still the number 5. However and in constrast with the KNN model, we can see that the that in this case we have 395 true labels, while in KNN we had 407. Meaning that in this case there were more mismatches for the labe 5.

° This means that this calssificer is slightly worst than the KNN classifier by analyzing the minimum and maximun values. This explains why the accuracy is lower: 93.6 % in constrast with 93.98 for the first model

Predicting probabilities

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

Out[714]:

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.5	0.0	0.5	0.0	0.0	0.0	0.0

Conclusions on the probability pairplots

° From the diagonal we can see the the target values in comparaison with the predicted class probability represented as a number between 0 and 1. From the class 0 until the class number 9. From the class 0, we can see that the accuracy is very high, almost reaching 100%, so the pairplot allow us to visualize the data points or samples for each row of the 5000 datapoints of our classifier, and the datapoints to the right, sidewise on the right side of each plot of the diagonal, reprensent the incorrenctly classified datapoints.

° The worst predicted class can be visualized on the labels 5 and 8, because if we inspect the diagonal plot, we can see on the righ side small datapoints that don't belong to the target class.

° As annotation, we can see that all the [i,j] positions where i != j, that is, pairplots different from the diagonal, show a data points position distribution between the different classes. This means that the triangle shape is the result of 5000 datapoints over 10 classes probabilities and its clustering represents the dense probability ranges.

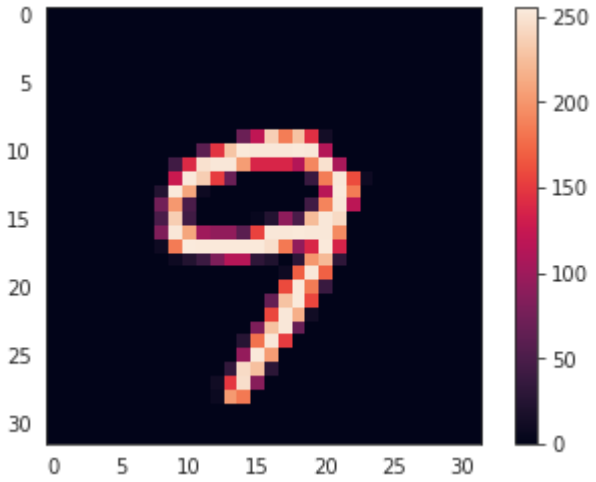
° When these clusters tend to approach to high probability values, then we can state that the label was correctly classified. Otherwise, the classifier should set their probability values near to zero and therefore the triangle shape should not be present, since this represents the missmatches and therefore higher error. We know each row should contain a single probability with value near to 1 or 100, since each number has a unique class.

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.

```
Predicted pixel: 9
```



```
target        318
prediction    318
dtype: int64
```

Out[721]:

	target	prediction
109	5	1
113	8	4
132	9	7
133	2	7
136	9	3
170	8	0
189	4	9
195	0	4
209	8	3
214	8	2


Pred: 9 Pred: 9 Pred: 3 Pred: 0 Pred: 3 Pred: 3 Pred: 8 Pred: 4 Pred: 9 Pred: 3
Pred: 3 Pred: 4 Pred: 7 Pred: 0 Pred: 3 Pred: 2 Pred: 6 Pred: 5 Pred: 1 Pred: 7
Pred: 0 Pred: 9 Pred: 9 Pred: 8 Pred: 1 Pred: 9 Pred: 1 Pred: 1 Pred: 2 Pred: 6
Pred: 3 Pred: 1 Pred: 0 Pred: 7 Pred: 1 Pred: 8 Pred: 6 Pred: 0 Pred: 4 Pred: 6
Pred: 1 Pred: 8 Pred: 3 Pred: 5 Pred: 3 Pred: 2 Pred: 7 Pred: 0 Pred: 2 Pred: 7
Pred: 4 Pred: 7 Pred: 0 Pred: 2 Pred: 9 Pred: 6 Pred: 5 Pred: 4 Pred: 3 Pred: 6
Pred: 8 Pred: 5 Pred: 8 Pred: 9 Pred: 4 Pred: 2 Pred: 7 Pred: 7 Pred: 4 Pred: 0
Pred: 5 Pred: 6 Pred: 2 Pred: 6 Pred: 6 Pred: 9 Pred: 5 Pred: 6 Pred: 5 Pred: 4
Pred: 1 Pred: 6 Pred: 5 Pred: 4 Pred: 8 Pred: 5 Pred: 4 Pred: 5 Pred: 2 Pred: 7
Pred: 4 Pred: 7 Pred: 6 Pred: 5 Pred: 8 Pred: 6 Pred: 1 Pred: 9 Pred: 9 Pred: 8

Remarks:

° From the above chart, we are able to evidence exactly where the classifier is making mistakes. In general, it is very effective for the classification of numbers on the range 0-9, but missclassifies the label 0 and sometimes fails when the numbers are too tilted, or extremly wrong written or in cases where its curves resamble the ones of other numbers. Similarly happened with the KNN classifier, which also showed a maximum classifications of labels '1', because it is the easiest one to predict, due to the easy-to follow topology of this array of pixels.

How to improve your results?

Recapitulate and discuss your results here and propose some ideas on how to improve them

° One possible way to improve our results can be, for the KNN model, to test with other possible K number of neighbors and re evaluate more candidates for the number of leafs, using alongside the Minkowski distance formula (p) parameters and storing themin a dictionary, we can keep testing posibilities with the GridSearchCV function, which optimizes by cross-validated grid-search over KNN model trained with the preprocessed dataset alongside with 10-fold cross validation. This showed that the best parameters were an euclidean distance (p) equal to 1, leafs = 1 and 3 neighbors, obtaining afinal score of 93.98%.

° Regarding the MLP, using a hidden number of layers of 20 each one, and an alpha parameter of 1.0, which helped to design a model with 93.54% score. A possible alternative to improve the model can be to implement different activation functions, such as 'tanh' and 'relu'. Different solver types, such as 'adam' or 'sdg'. And finally, testing with a wider learning rate range.