Pixel Similarity & Co

How to categorize images without Neural Networks

Dirk, 2021-06-01

Pixel Similarity 📌 & Co

How to categorize images without Neural Networks

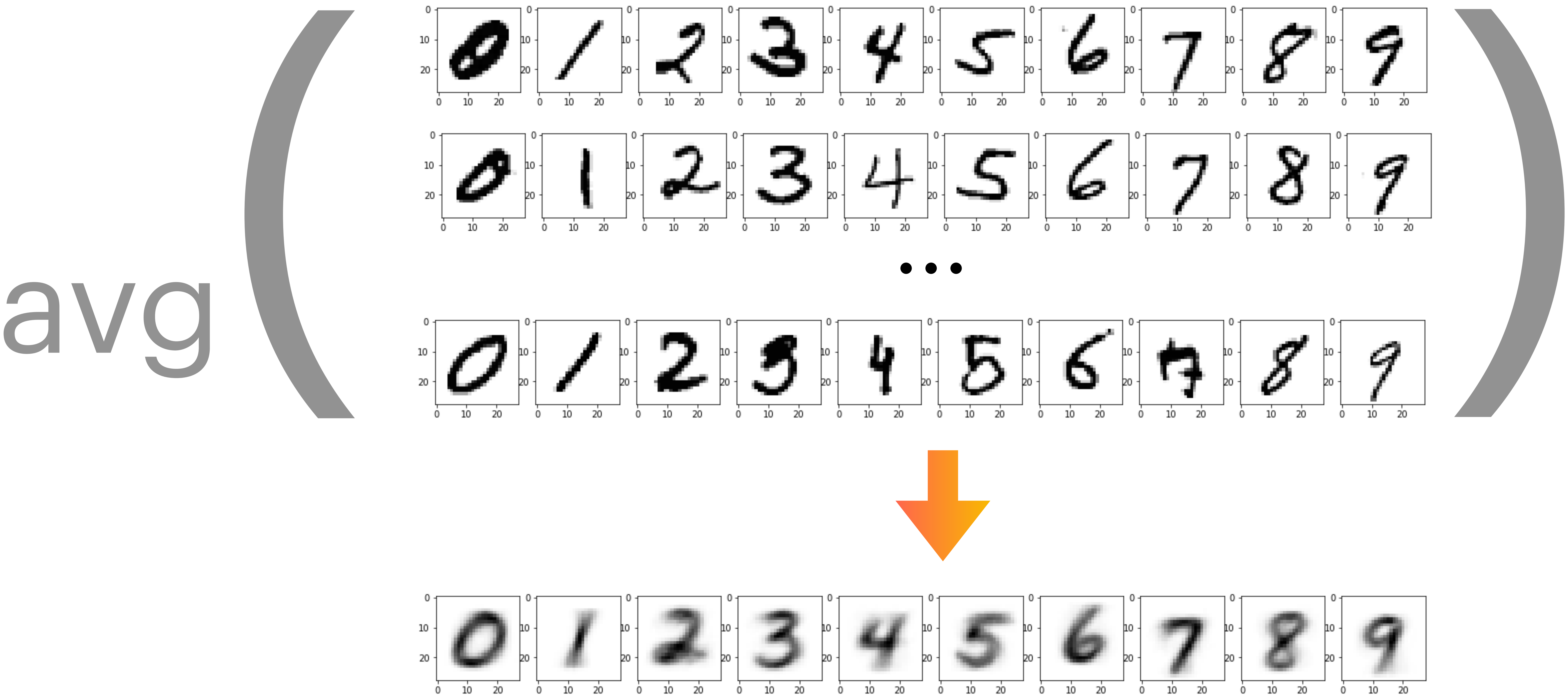
1

Overview of this basic approach:

- *What is the reasoning and steps?*
- *Describe what the distance means*

Dirk, 2021-06-01

"Ideal Images" as Target



X

"Ideal Images" as Target

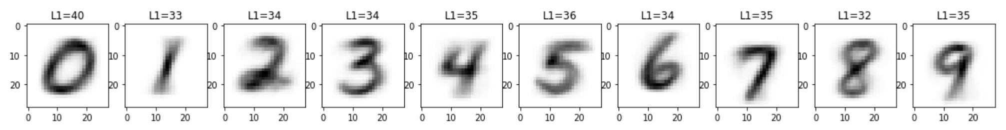
"ideal" 2 as tensor

avg ( )

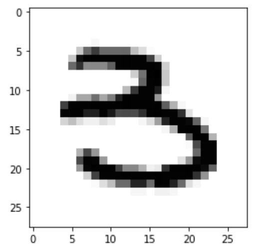
```
tensor([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  1,  1,  2,  4,  7, 12, 14, 16, 16, 15, 12,  8,  6,  4,  2,  1,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  2,  5, 11, 18, 30, 39, 46, 50, 49, 47, 38, 31, 23, 13,  6,  3,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  1,  7, 15, 25, 35, 44, 56, 61, 63, 65, 65, 60, 48, 40, 29, 14,  6,  1,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  4, 12, 21, 30, 39, 47, 53, 56, 58, 64, 66, 64, 57, 47, 38, 21,  7,  2,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  2,  5, 14, 20, 28, 35, 39, 45, 47, 52, 52, 52, 53, 54, 52, 40, 27,  8,  3,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  4,  7, 10, 15, 23, 25, 28, 29, 34, 39, 40, 41, 47, 47, 52, 42, 28,  9,  3,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  3,  5,  6,  9, 13, 13, 15, 18, 22, 21, 22, 31, 44, 50, 54, 42, 25,  9,  2,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  1,  4,  4,  4,  7,  8,  9, 11, 12, 11, 17, 32, 48, 55, 56, 41, 20,  7,  1,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  1,  1,  1,  3,  4,  5,  6,  6, 10, 20, 37, 55, 61, 54, 35, 16,  5,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  2,  3,  7, 14, 28, 48, 61, 61, 49, 29, 13,  3,  0,  1,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  1,  2,  3,  5,  9, 14, 25, 42, 55, 64, 55, 41, 22,  9,  3,  1,  1,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  1,  4,  7, 12, 15, 19, 22, 30, 45, 56, 60, 58, 48, 31, 15,  5,  2,  2,  1,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  2,  9, 16, 26, 35, 40, 46, 52, 62, 69, 70, 62, 53, 41, 25, 12,  6,  4,  4,  2,  1,  0,  0],
        [ 0,  0,  0,  0,  3, 12, 21, 34, 43, 50, 58, 66, 73, 79, 78, 71, 63, 55, 41, 27, 19, 14, 12, 10,  4,  2,  0,  0],
        [ 0,  0,  0,  1,  9, 22, 36, 45, 50, 57, 64, 71, 75, 77, 76, 69, 63, 59, 46, 34, 25, 22, 19, 13,  6,  2,  0,  0],
        [ 0,  0,  0,  4, 16, 31, 42, 48, 52, 55, 64, 68, 69, 71, 67, 66, 63, 55, 49, 43, 39, 36, 25, 18, 10,  2,  0,  0],
        [ 0,  0,  0,  8, 20, 34, 46, 52, 60, 67, 71, 70, 66, 60, 58, 56, 54, 46, 44, 46, 47, 42, 34, 21,  8,  3,  0,  0],
        [ 0,  0,  0, 10, 19, 35, 49, 60, 71, 75, 77, 71, 62, 53, 49, 45, 40, 38, 36, 35, 37, 32, 25, 16,  8,  3,  0,  0],
        [ 0,  0,  1,  6, 17, 30, 44, 56, 64, 65, 61, 56, 49, 44, 36, 31, 32, 33, 32, 28, 24, 18, 14,  7,  2,  1,  0,  0],
        [ 0,  0,  0,  2, 10, 17, 25, 33, 39, 42, 41, 34, 30, 25, 20, 18, 21, 23, 19, 12,  9,  7,  5,  3,  1,  0,  0,  0],
        [ 0,  0,  0,  0,  1,  5,  8, 12, 13, 13, 13, 11,  8,  7,  6,  6,  7,  7,  5,  3,  2,  3,  3,  1,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  1,  2,  2,  3,  3,  2,  2,  2,  2,  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,  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]], dtype=torch.int8)
```

"L1 Norm" Algo

ideal_digits



img

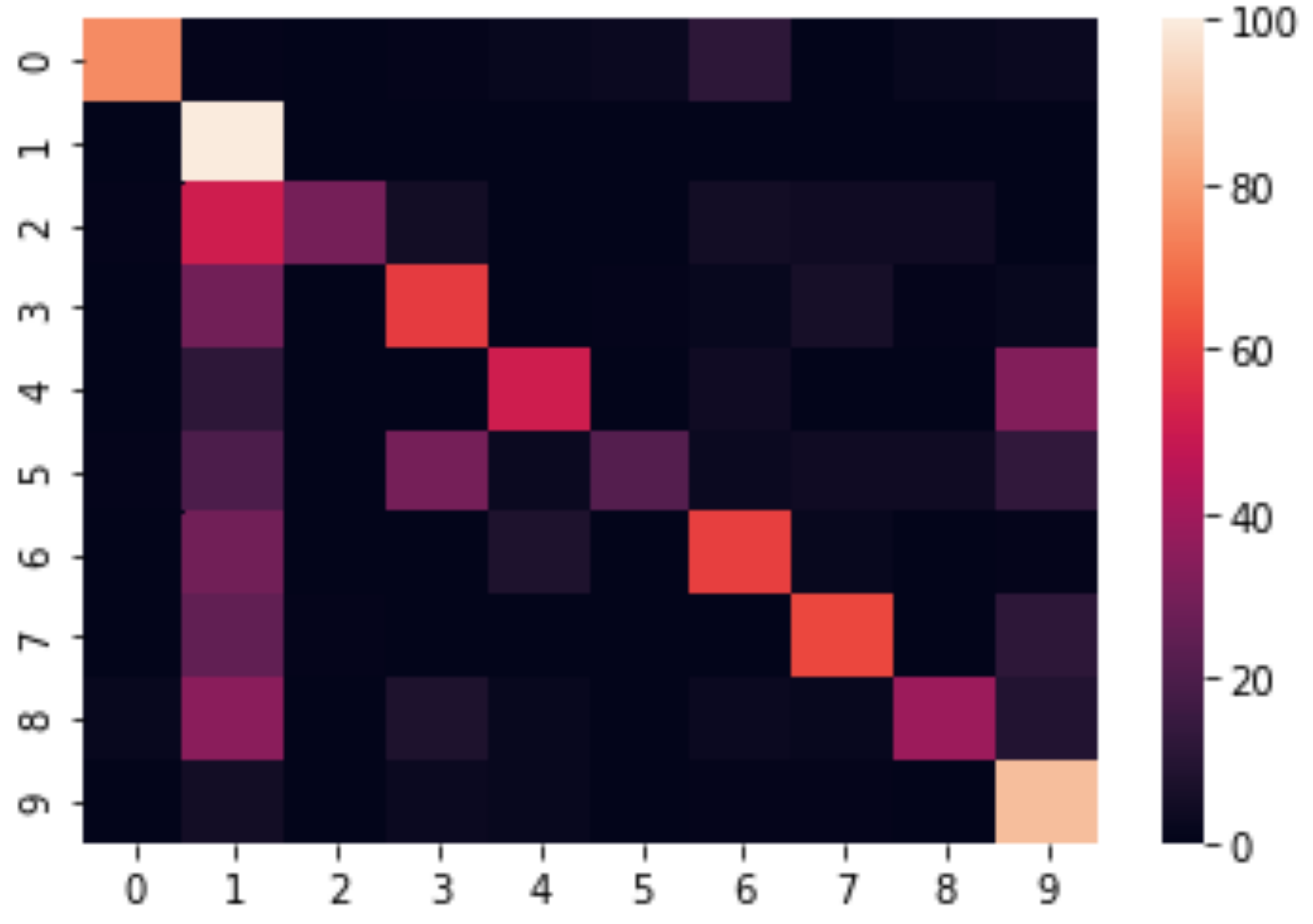


```python
# Example: L1 for img
l1=[(img - digit).abs().mean() for digit in ideal_digits]
l1
```

```
[tensor(44.2404),
 tensor(44.0219),
 tensor(46.1260),
 tensor(41.8001),
 tensor(46.5060),
 tensor(42.7080),
 tensor(44.3475),
 tensor(45.5552),
 tensor(45.4974),
 tensor(45.0498)]
```

```python
# index of smallest element is digit number
np.argmin(l1)
```

```
3
```




```
Classification report for classifier:
              precision    recall  f1-score   support

           0       0.95      0.76      0.84       100
           1       0.33      1.00      0.49       100
           2       0.97      0.30      0.46       100
           3       0.56      0.59      0.57       100
           4       0.75      0.51      0.61       100
           5       0.85      0.22      0.35       100
           6       0.67      0.60      0.63       100
           7       0.77      0.62      0.69       100
           8       0.78      0.39      0.52       100
           9       0.55      0.88      0.67       100

    accuracy                           0.59      1000
   macro avg       0.72      0.59      0.58      1000
weighted avg       0.72      0.59      0.58      1000
```

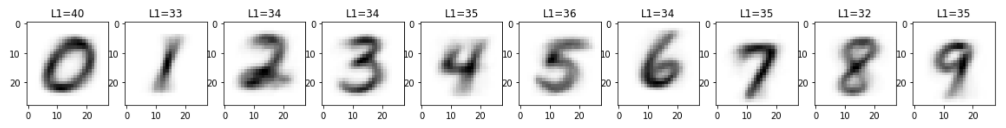
```python
confusion_matrix(true, pred)
```

```
array([[ 76,   1,   0,   1,   2,   3,  12,   0,   2,   3],
       [  0, 100,   0,   0,   0,   0,   0,   0,   0,   0],
       [  1,  51,  30,   5,   0,   0,   5,   4,   4,   0],
       [  0,  29,   0,  59,   0,   1,   2,   6,   1,   2],
       [  0,  12,   0,   0,  51,   0,   4,   0,   0,  33],
       [  1,  20,   0,  30,   3,  22,   3,   4,   4,  13],
       [  0,  29,   0,   0,   8,   0,  60,   2,   0,   1],
       [  0,  25,   1,   0,   0,   0,   0,  62,   0,  12],
       [  2,  35,   0,   8,   2,   0,   3,   2,  39,   9],
       [  0,   5,   0,   3,   2,   0,   1,   1,   0,  88]])
```

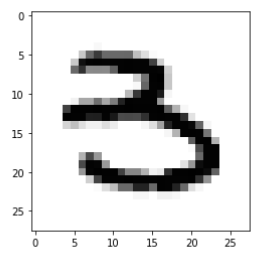
🧑‍🏫 The `L1-Norm` Algorithm brings solid results with an overall accuracy-value of 59%.

"L2 Norm" Algo

ideal_digits



img

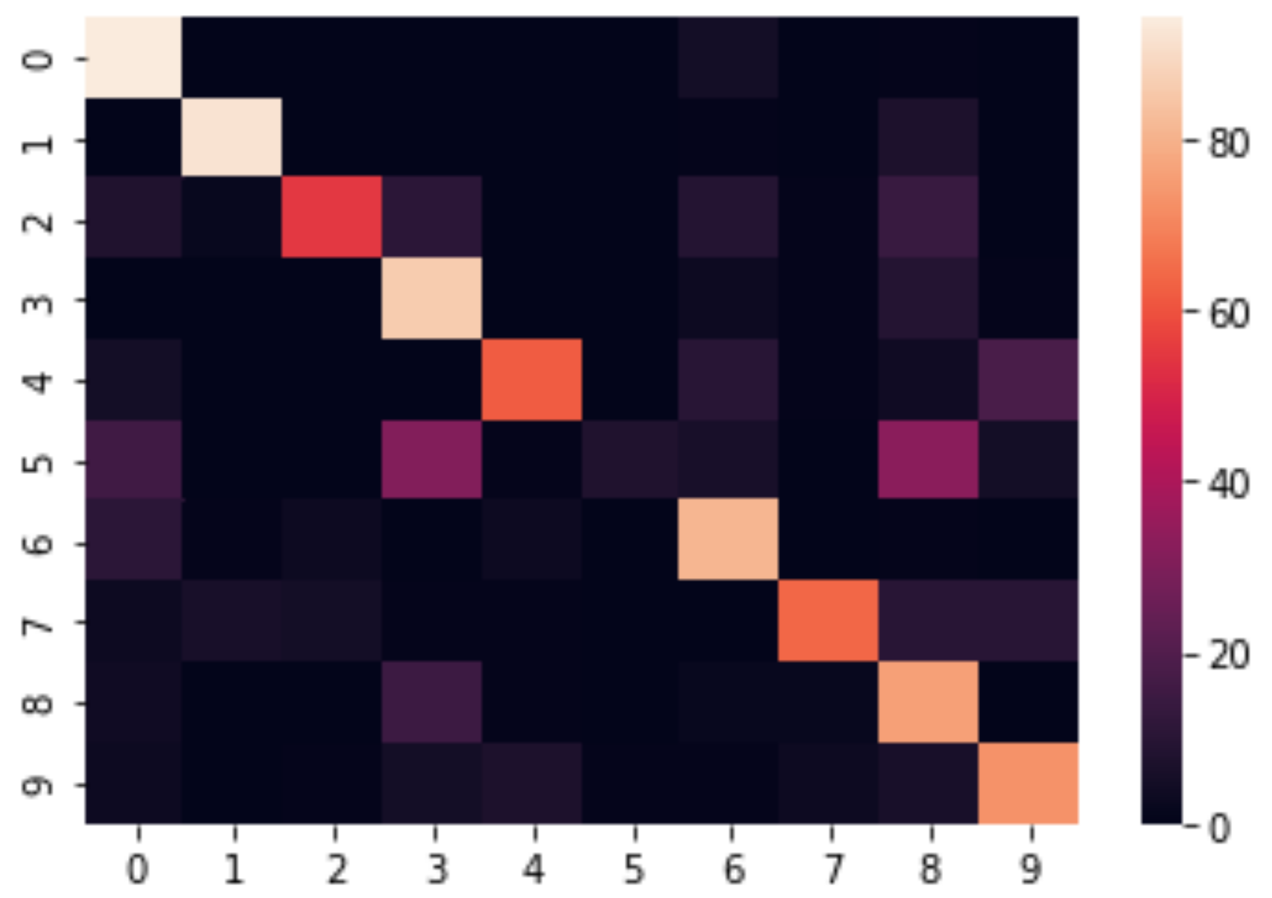


```python
# Example: L2 for img
l2=[((img - digit)**2).mean().sqrt() for digit in ideal_digits]
l2
```

```
[tensor(83.5524),
 tensor(91.7463),
 tensor(88.1979),
 tensor(83.2467),
 tensor(90.5193),
 tensor(85.2805),
 tensor(86.3737),
 tensor(90.3234),
 tensor(87.1980),
 tensor(89.5686)]
```

```python
# index of smallest element is digit number
np.argmin(l2)
```

```
3
```




```
Classification report for classifier:
              precision    recall  f1-score   support

           0       0.65      0.94      0.77       100
           1       0.91      0.92      0.92       100
           2       0.86      0.55      0.67       100
           3       0.58      0.86      0.69       100
           4       0.83      0.62      0.71       100
           5       0.89      0.08      0.15       100
           6       0.69      0.81      0.74       100
           7       0.89      0.64      0.74       100
           8       0.47      0.76      0.58       100
           9       0.68      0.73      0.71       100

    accuracy                           0.69      1000
   macro avg       0.74      0.69      0.67      1000
weighted avg       0.74      0.69      0.67      1000
```

```python
confusion_matrix(true, pred)
```

```
array([[94,  0,  0,  0,  0,  0,  5,  0,  1,  0],
       [ 0, 92,  0,  0,  0,  0,  1,  0,  7,  0],
       [ 8,  2, 55, 11,  0,  0,  9,  1, 14,  0],
       [ 0,  0,  0, 86,  0,  0,  3,  1,  9,  1],
       [ 5,  0,  0,  0, 62,  0, 10,  1,  4, 18],
       [16,  0,  0, 31,  1,  8,  6,  0, 33,  5],
       [11,  1,  3,  0,  3,  0, 81,  0,  1,  0],
       [ 3,  6,  5,  1,  1,  0,  0, 64, 10, 10],
       [ 4,  0,  0, 15,  1,  0,  2,  2, 76,  0],
       [ 3,  0,  1,  5,  7,  1,  1,  3,  6, 73]])
```

🧑‍🏫 The `L2-Norm` Algorithm brings even better results with an overall accuracy-value of 69%.

X

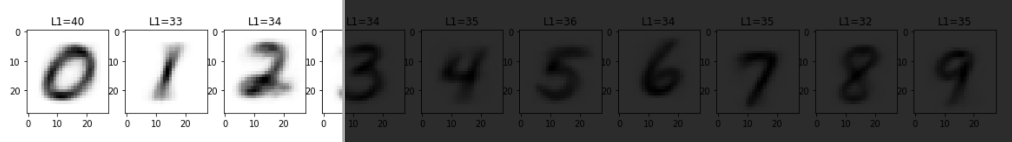
"L2 Norm" Algo

2

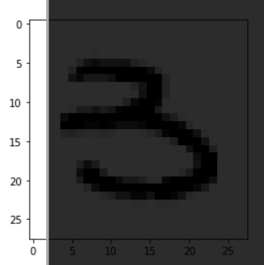
"Remember to step away from this book and jot down some ideas before you move on! "

[1] Chapter: I.3

ideal_digits



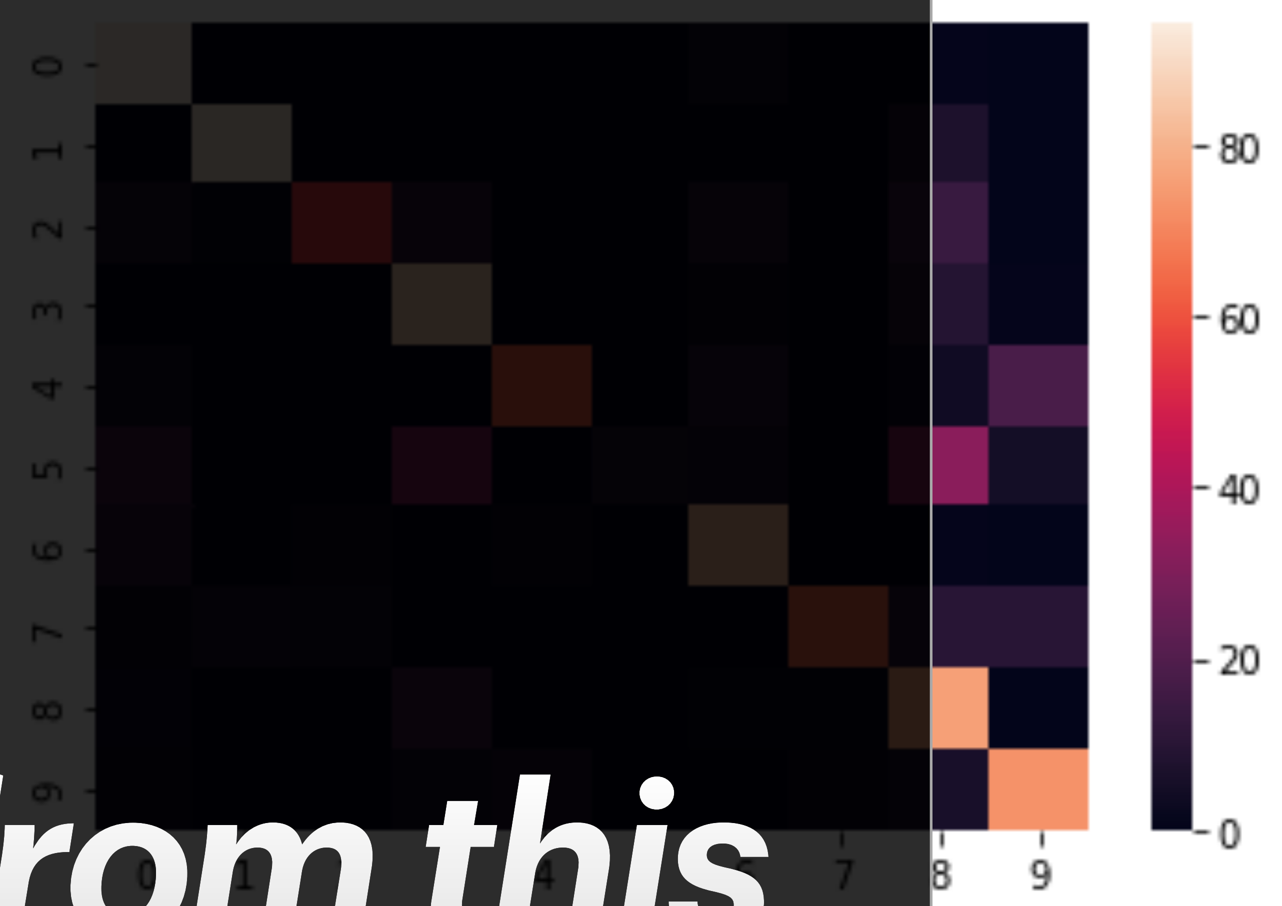
img



```
# Example: L2 for img
l2=[((img - digit)**2).mean().sqrt() for digit in ideal_digits]
l2
```

```
[tensor(83.5524),
 tensor(91.7463),
 tensor(88.1979),
 tensor(83.2467),
 tensor(90.5193),
 tensor(85.2805),
 tensor(86.3737),
 tensor(90.3234),
 tensor(87.1980),
 tensor(89.5686)]
```

```
# index of smallest element is digit number
np.argmin(l2)
```



	precision	recall	f1-score	support
0	0.65	0.94	0.77	100
1	0.91	0.92	0.92	100
2	0.86	[illegible]	[illegible]	[illegible]
3	0.58	[illegible]	[illegible]	[illegible]
4	0.83	0.62	0.71	100
5	0.89	0.08	0.15	100
6	0.69	0.81	0.74	100
7	0.89	0.64	0.74	100
8	0.47	0.76	0.58	100
9	0.68	0.73	0.71	100
accuracy			0.69	1000
macro avg	0.74	0.69	0.67	1000
weighted avg	0.74	0.69	0.67	1000

```
confusion_matrix(true, pred)
```

```
[[94,  0,  0,  0,  0,  0,  5,  0,  1,  0],
 [ 0, 92,  0,  0,  0,  0,  1,  0,  7,  0],
 [ 8,  2, 55, 11,  0,  0,  9,  1, 14,  0],
 [ 0,  0,  0, 86,  0,  0,  3,  1,  9,  1],
 [ 5,  0,  0,  0, 62,  0, 10,  1,  4, 18],
 [16,  0,  0, 31,  1,  8,  6,  0, 33,  5],
 [11,  1,  3,  0,  3,  0, 81,  0,  1,  0],
 [ 3,  6,  5,  1,  1,  0,  0, 64, 10, 10],
 [ 4,  0,  0, 15,  1,  0,  2,  2, 76,  0],
 [ 3,  0,  1,  5,  7,  1,  1,  3,  6, 73]])
```

👨‍🏫 The `L2-Norm` Algorithm brings even better results with an overall accuracy-value of 69%.

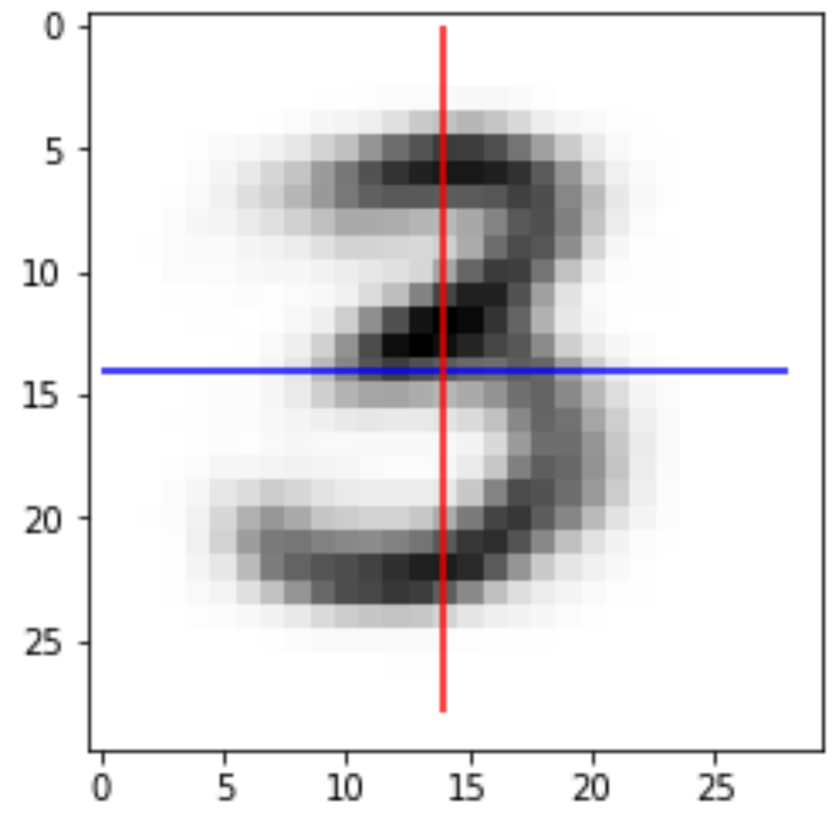
Find a Value which distincts every image class

1. "Average Ink" Value

👨‍💻 sum of all pixels

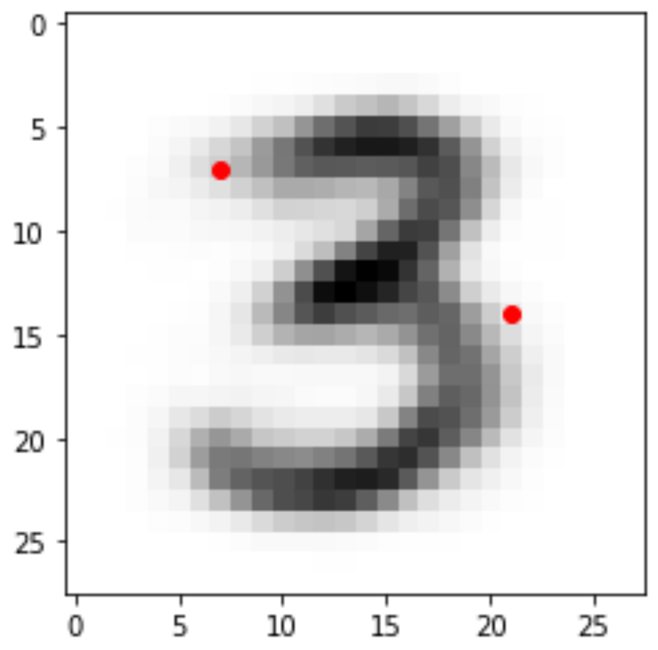
2. "Middle Line" Ratio

👨‍💻 sum of blue line / sum of red line

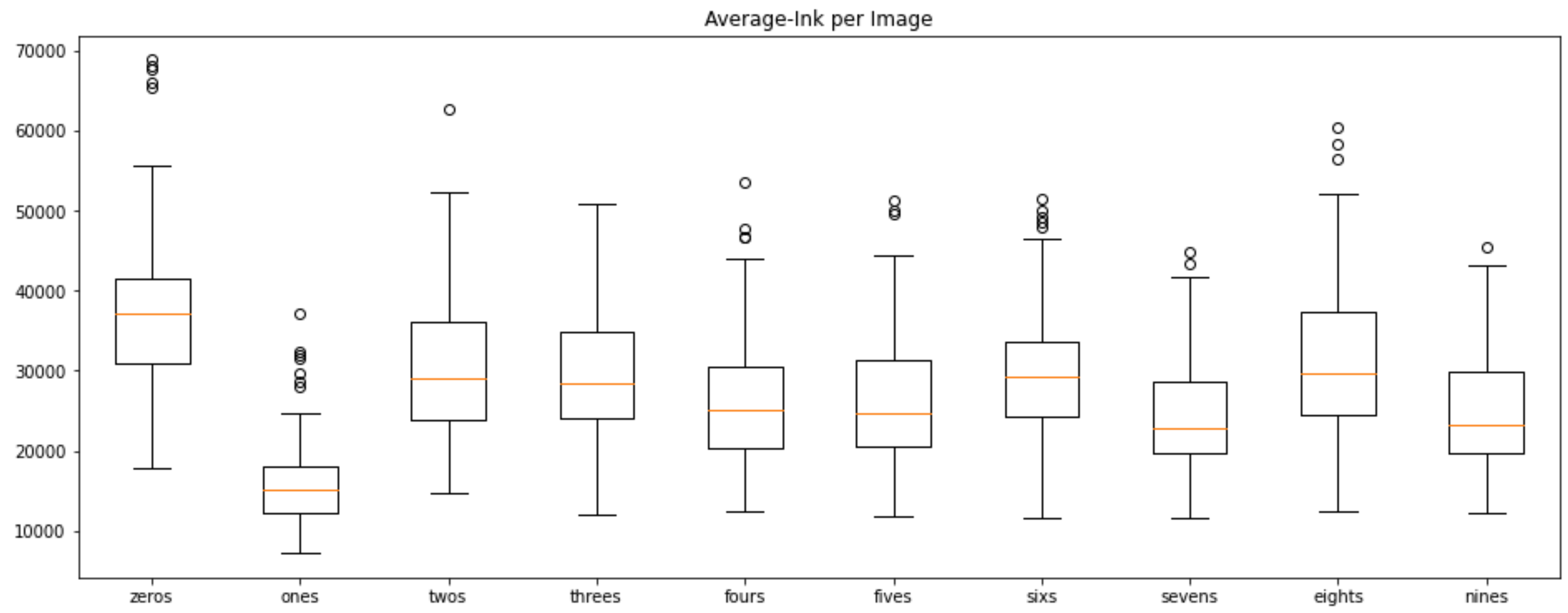


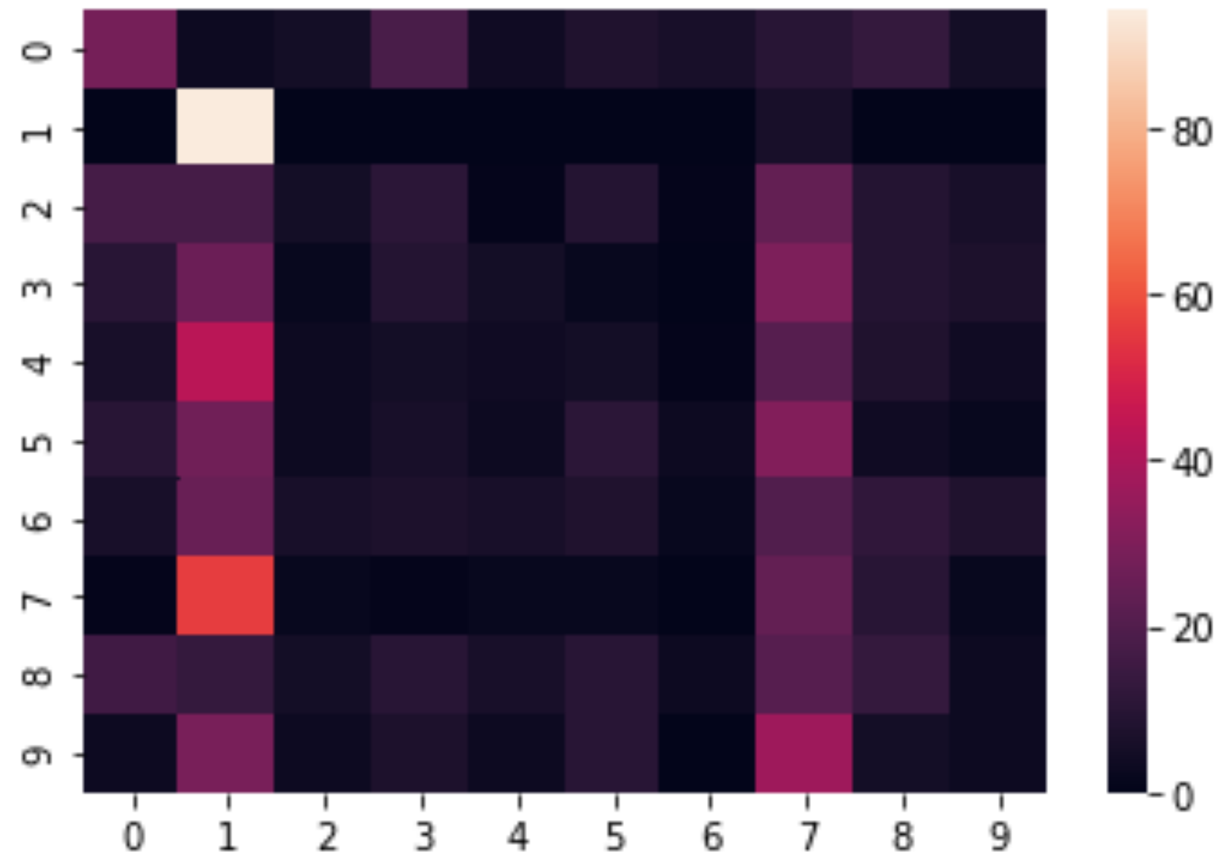
3. "Random Points" Value

👨‍💻 sum of pixel value of red points



"Average Ink" Algo





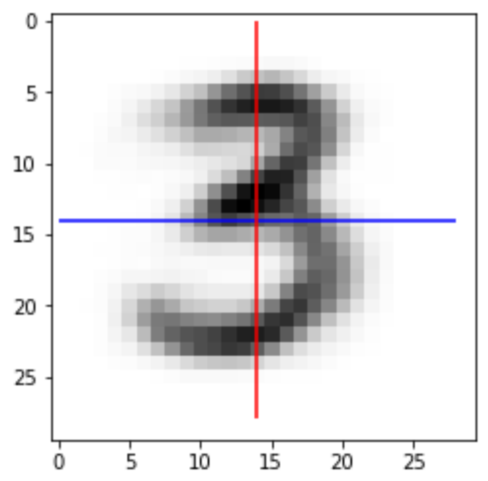

```
Classification report for classifier:
              precision    recall  f1-score   support

           0       0.29      0.28      0.28       100
           1       0.28      0.94      0.43       100
           2       0.15      0.05      0.07       100
           3       0.12      0.09      0.10       100
           4       0.12      0.04      0.06       100
           5       0.17      0.11      0.13       100
           6       0.12      0.02      0.03       100
           7       0.11      0.24      0.15       100
           8       0.16      0.13      0.14       100
           9       0.07      0.03      0.04       100

    accuracy                           0.19      1000
   macro avg       0.16      0.19      0.15      1000
weighted avg       0.16      0.19      0.15      1000
```

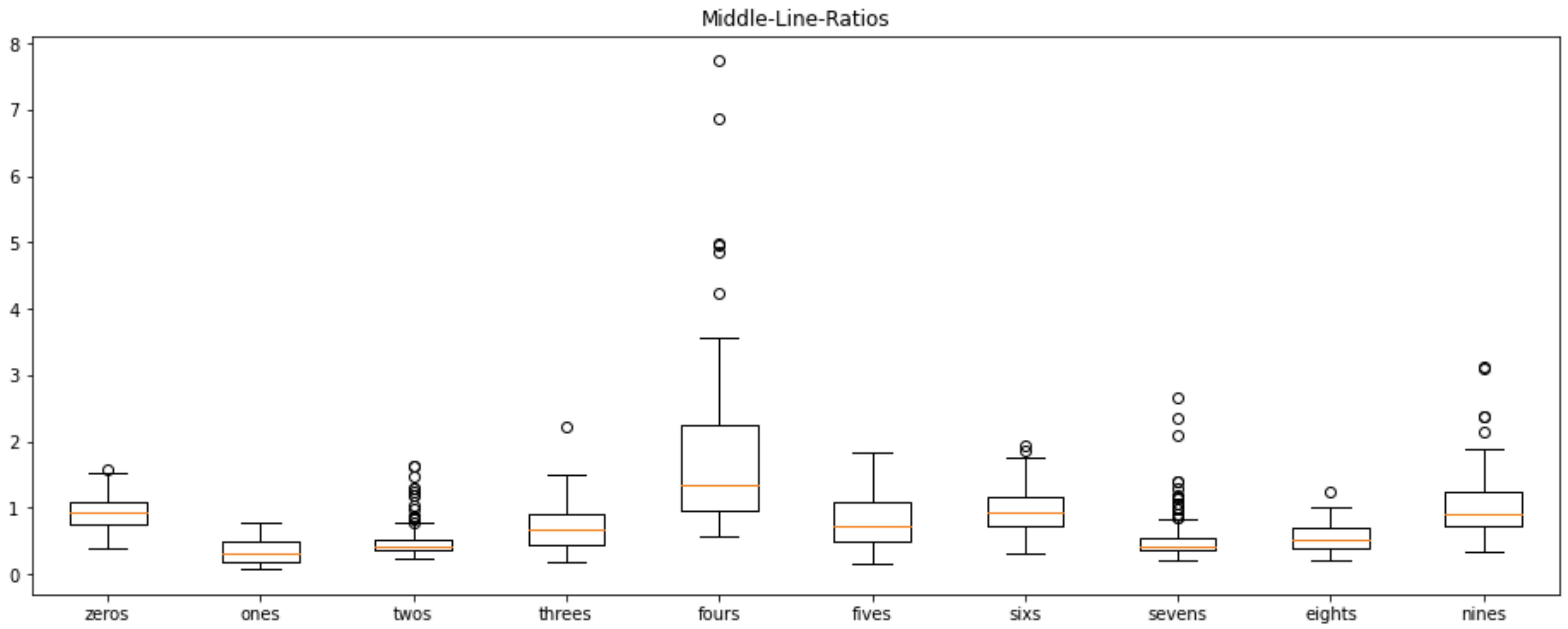
```
confusion_matrix(true, pred)
```

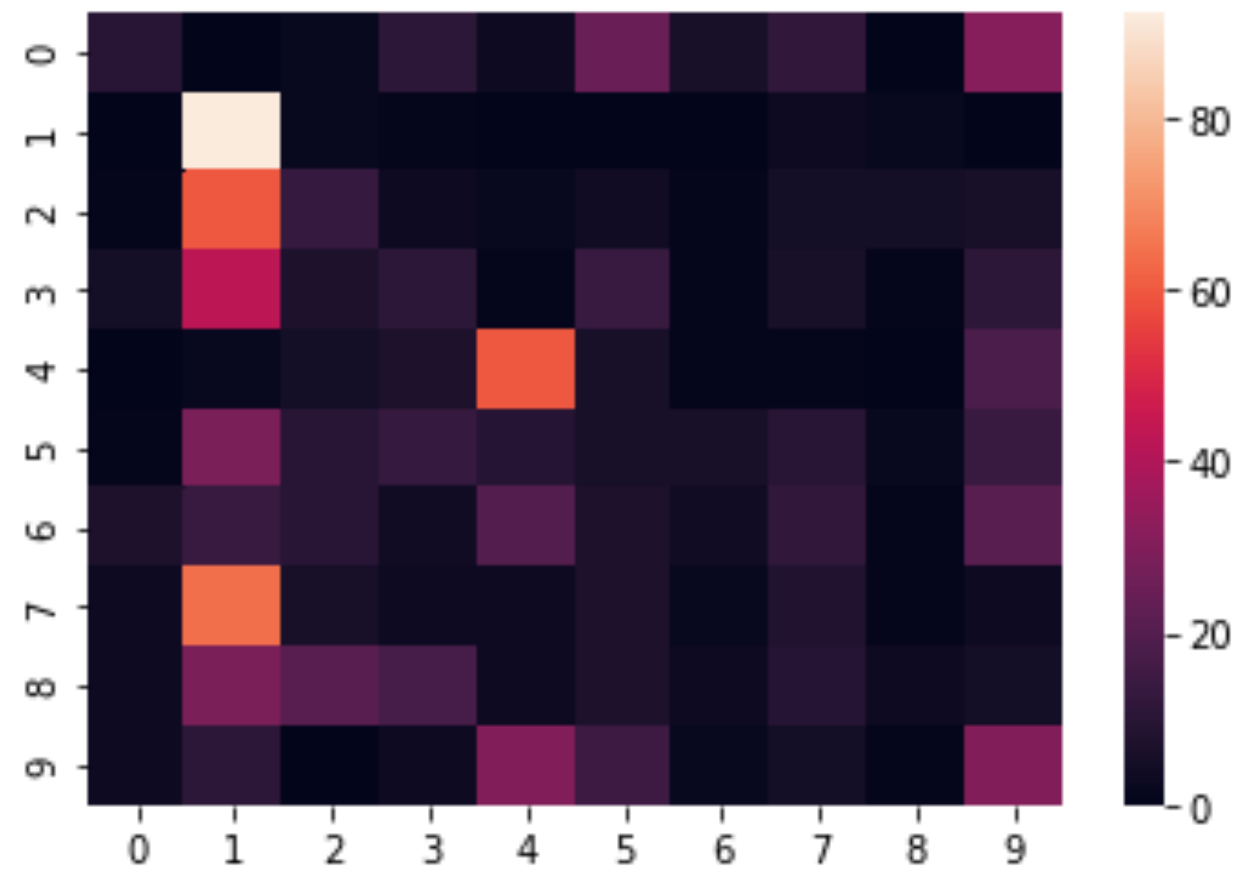
```
array([[28,  3,  5, 18,  4,  8,  6, 10, 13,  5],
       [ 0, 94,  0,  0,  0,  0,  0,  6,  0,  0],
       [17, 17,  5, 11,  1,  9,  1, 24,  9,  6],
       [10, 26,  2,  9,  5,  2,  0, 30,  9,  7],
       [ 6, 43,  3,  5,  4,  5,  1, 21,  8,  4],
       [10, 27,  3,  6,  3, 11,  3, 31,  4,  2],
       [ 6, 25,  6,  7,  6,  8,  2, 20, 12,  8],
       [ 1, 56,  2,  1,  2,  2,  0, 24, 10,  2],
       [16, 13,  5, 10,  6, 10,  3, 21, 13,  3],
       [ 3, 29,  3,  7,  3, 10,  0, 37,  5,  3]])
```

🧑‍🏫 The `Average-Ink` Algorithms results are disappointing: the average accuracy is only 19% and probably because of its very simply formula so low.

"Middle Line Ratio" Algo







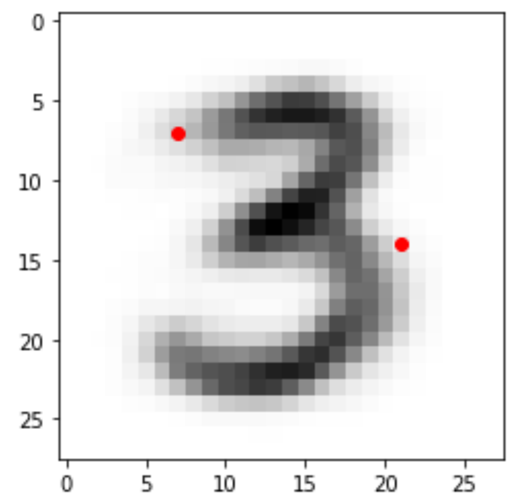

```
Classification report for classifier:
              precision    recall  f1-score   support

           0       0.30      0.10      0.15       100
           1       0.27      0.92      0.41       100
           2       0.17      0.13      0.15       100
           3       0.15      0.11      0.13       100
           4       0.46      0.60      0.52       100
           5       0.07      0.06      0.06       100
           6       0.15      0.04      0.06       100
           7       0.11      0.08      0.09       100
           8       0.19      0.03      0.05       100
           9       0.22      0.30      0.25       100

    accuracy                           0.24      1000
   macro avg       0.21      0.24      0.19      1000
weighted avg       0.21      0.24      0.19      1000
```

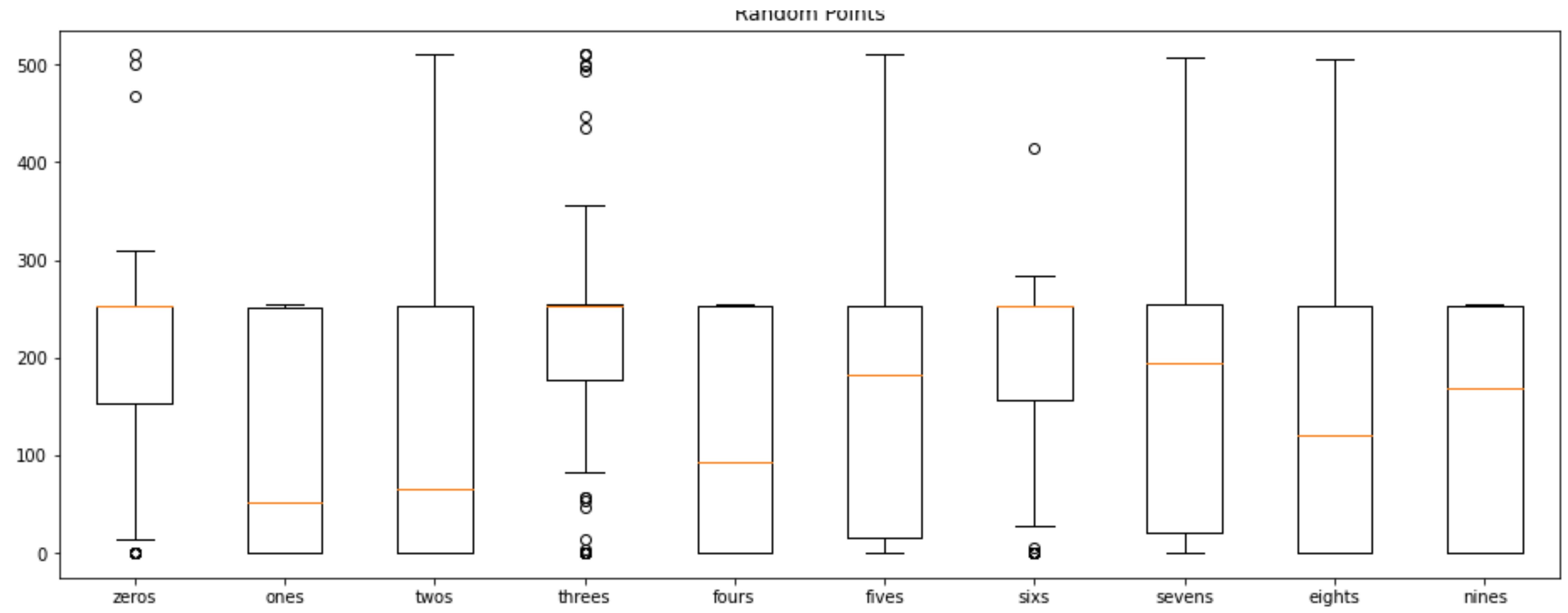
```
confusion_matrix(true, pred)
```

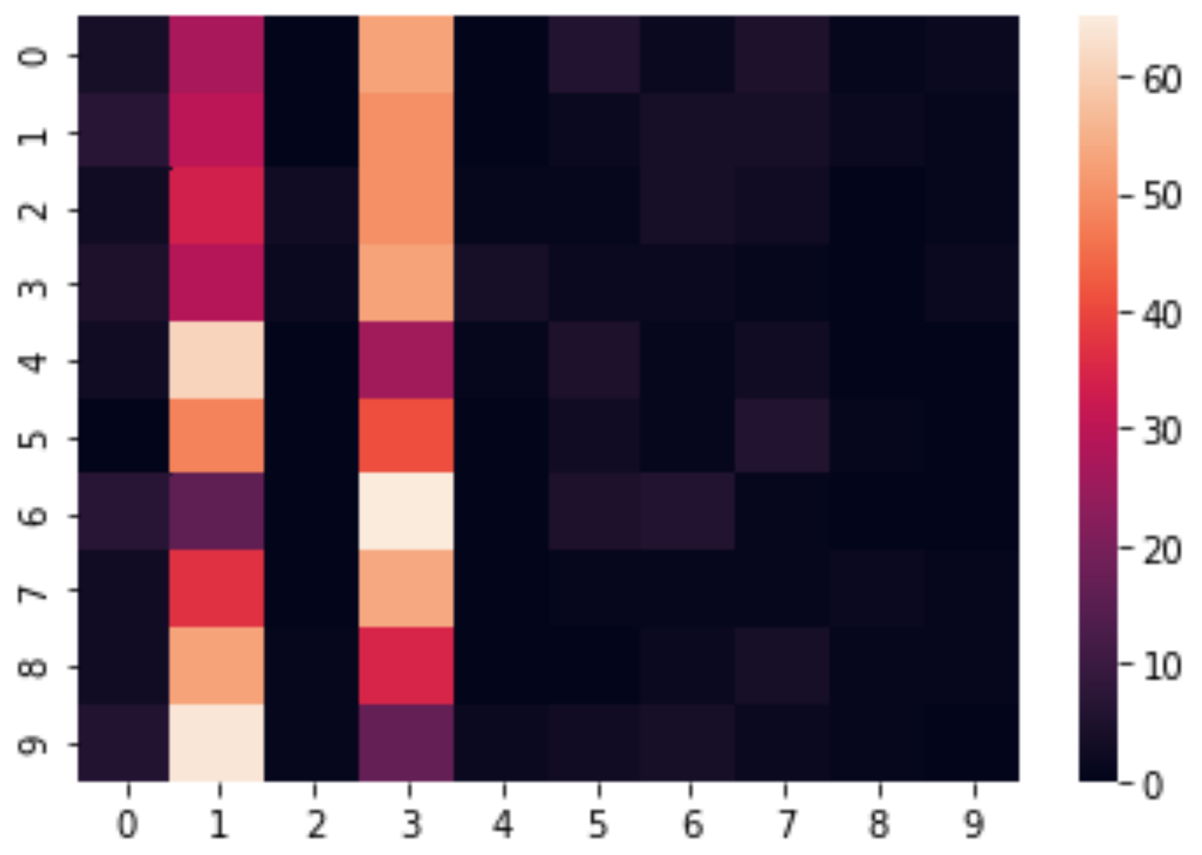
```
array([[10,  0,  2, 11,  3, 25,  6, 12,  0, 31],
       [ 0, 92,  2,  1,  0,  0,  0,  3,  2,  0],
       [ 1, 60, 13,  3,  2,  4,  1,  5,  5,  6],
       [ 5, 43,  7, 11,  1, 14,  1,  6,  1, 11],
       [ 0,  2,  5,  7, 60,  6,  1,  1,  0, 18],
       [ 1, 29, 10, 13,  9,  6,  6, 10,  2, 14],
       [ 7, 14, 10,  4, 20,  7,  4, 12,  1, 21],
       [ 3, 64,  6,  3,  3,  7,  2,  8,  1,  3],
       [ 3, 29, 21, 17,  3,  7,  3,  9,  3,  5],
       [ 3, 11,  0,  3, 30, 15,  2,  5,  1, 30]])
```

The `Middle-Line-Ratio` Algorithm is almost as disappointing as the Average-Ink Algorithm. The average accuracy is only 24%.

"Random Points" Algo








```
Classification report for classifier:
              precision    recall  f1-score   support

           0       0.10      0.04      0.06       100
           1       0.08      0.30      0.12       100
           2       0.43      0.03      0.06       100
           3       0.12      0.53      0.19       100
           4       0.12      0.01      0.02       100
           5       0.11      0.03      0.05       100
           6       0.22      0.06      0.09       100
           7       0.03      0.01      0.02       100
           8       0.12      0.01      0.02       100
           9       0.00      0.00      0.00       100

    accuracy                           0.10      1000
   macro avg       0.13      0.10      0.06      1000
weighted avg       0.13      0.10      0.06      1000
```

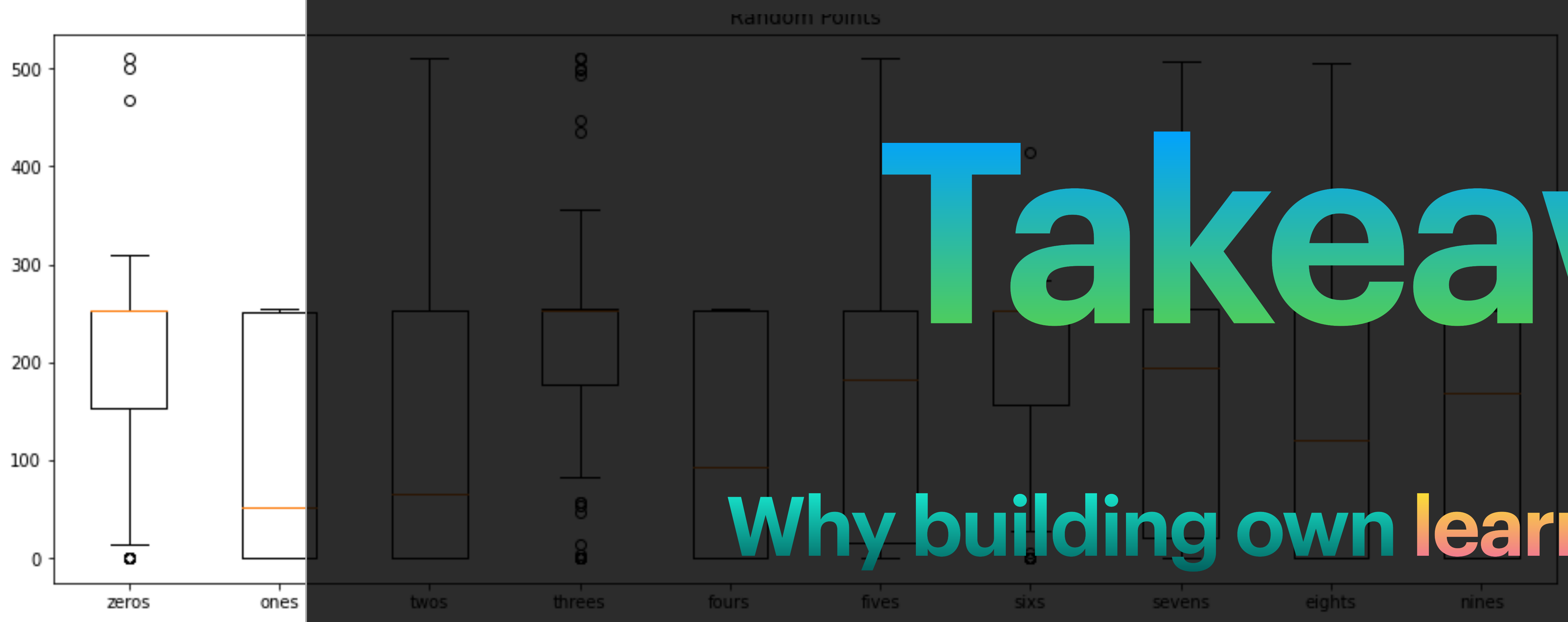
```
confusion_matrix(true, pred)
```

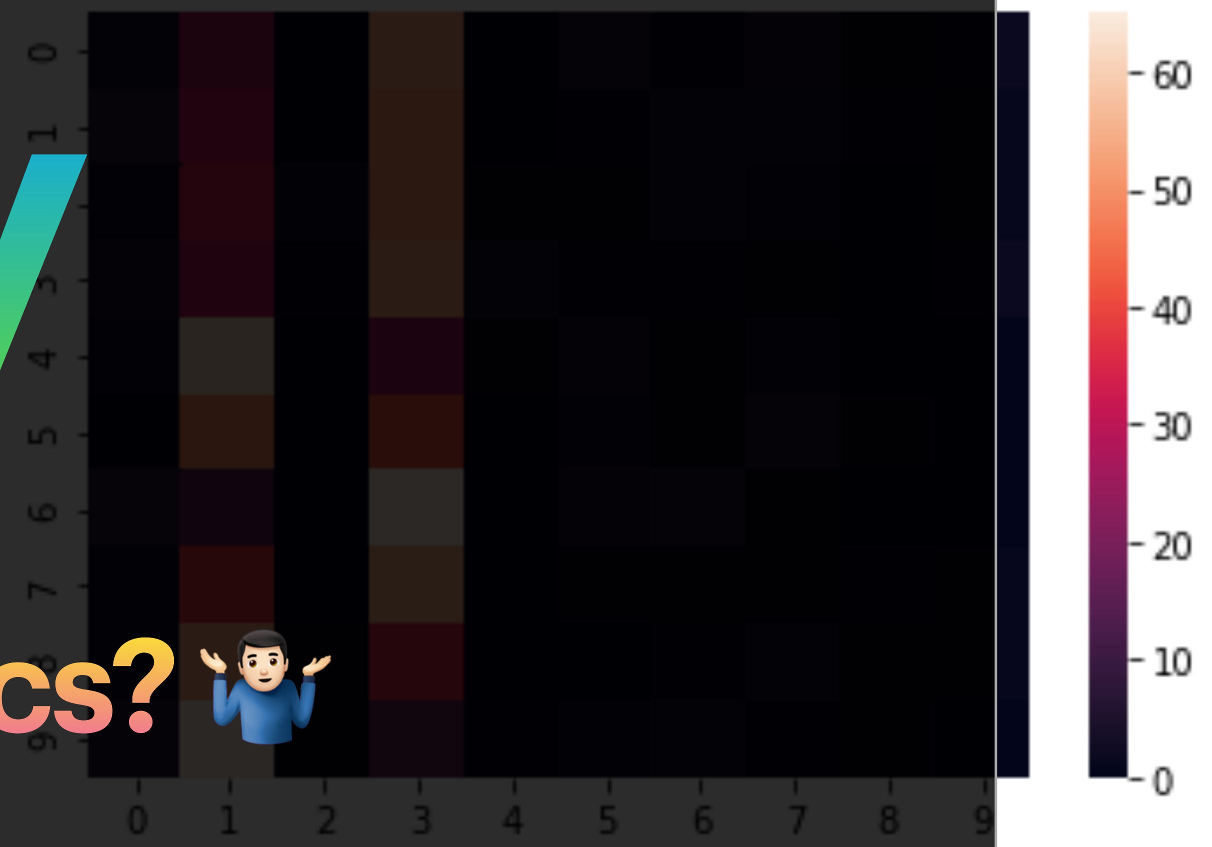
```
array([[ 4, 27,  0, 53,  0,  6,  2,  5,  1,  2],
       [ 7, 30,  0, 50,  0,  2,  4,  4,  2,  1],
       [ 3, 34,  3, 50,  1,  1,  4,  3,  0,  1],
       [ 5, 29,  2, 53,  4,  2,  2,  1,  0,  2],
       [ 3, 61,  0, 26,  1,  5,  1,  3,  0,  0],
       [ 0, 48,  0, 41,  0,  3,  1,  6,  1,  0],
       [ 7, 16,  0, 65,  0,  5,  6,  1,  0,  0],
       [ 3, 37,  0, 54,  0,  1,  1,  1,  2,  1],
       [ 3, 53,  1, 35,  0,  0,  2,  4,  1,  1],
       [ 6, 64,  1, 17,  2,  3,  4,  2,  1,  0]])
```

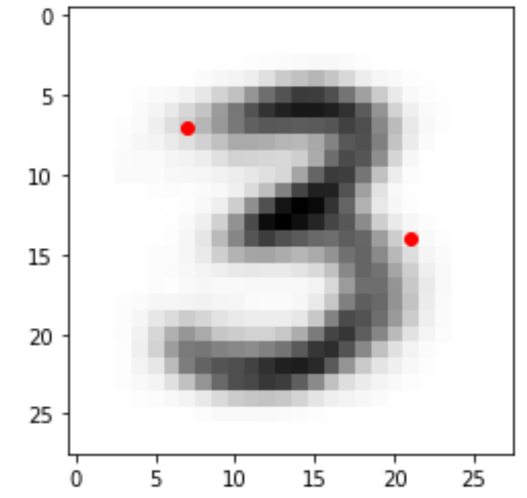
🧑‍🏫 The `Random-Points` Algorithm gives only 10% accuracy. But it heavily depends on where to place the points and how many. So it is improvable ...

X

"Random Points" Algo







Takeaway

Why building own learning metrics? 🤷🏻‍♂️

Not a substitue for NN, but might be an enhancement:

- can increase performance
- not enough classified sample images
- "extra boost" to stand out with your NN (e.g. Kaggle)
- Example: 🍄

```
confusion_matrix(true, pred)
```

```
Classification report for classifier:
              precision    recall  f1-score   support

weighted avg       0.13      0.10      0.06      1000
```

🧑‍🏫 The `Random-Points` Algorithm gives only 10% accuracy. But it heavily depends on where to place the points and how many. So it is improvable ...

References

1. Howards, J & Gugger, S. (2020): Deep Learning for Coders with fastai & PTorch. O'Reilly *or corresponding Jupyter Notebook on* Github
2. Code on my Github Repo