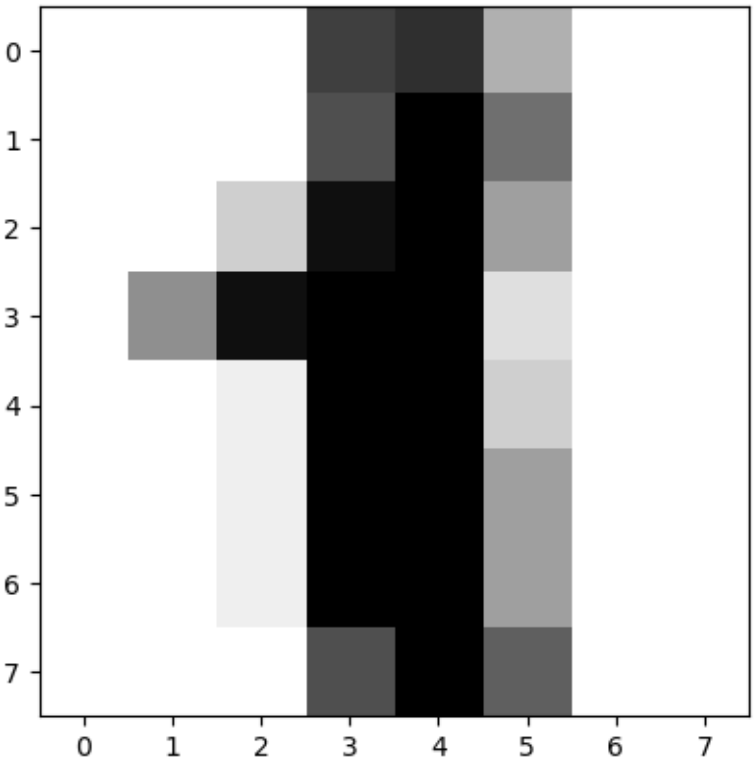
MNIST SVC

September 13, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

[2]:
```
data = datasets.load_digits()
```

[3]:
```
plt.imshow(data.images[1], cmap = plt.cm.gray_r);
```



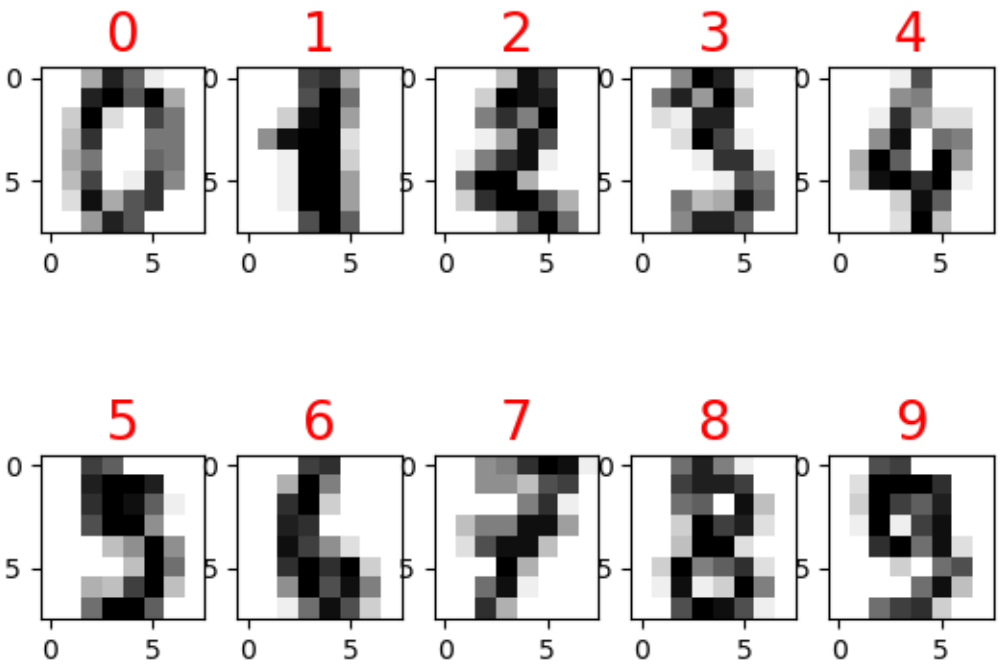
[4]:
```
data.target[1]
```

[4]:
```
1
```

[5]:
```
image_and_targets = list(zip(data.images, data.target))
image_and_targets[1]
```

[5]:
```
(array([[ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.],
        [ 0.,  0.,  0., 11., 16.,  9.,  0.,  0.],
        [ 0.,  0.,  3., 15., 16.,  6.,  0.,  0.],
        [ 0.,  7., 15., 16., 16.,  2.,  0.,  0.],
        [ 0.,  0.,  1., 16., 16.,  3.,  0.,  0.],
        [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
        [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
        [ 0.,  0.,  0., 11., 16., 10.,  0.,  0.]]),
 1)
```

[6]:
```
for i, (image, target) in enumerate(image_and_targets[: 10]):
    plt.subplot(2, 5, i+1)
    plt.imshow(image, cmap = plt.cm.gray_r)
    plt.title(target, fontsize=20, c='red')
```



[7]:
```
np.shape(data.images)
```

[7]:
```
(1797, 8, 8)
```

[8]:
```
len(data.images)
```

[8]:
```
1797
```

[9]:
```
flatshape = data.images.reshape(len(data.images), -1)
flatshape
```

[9]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

[10]:
```
np.shape(flatshape)
```

[10]:
```
(1797, 64)
```

[11]:
```
y = data.target
len(y)
```

[11]:
```
1797
```

[12]:
```
from sklearn.svm import SVC
SVC
```

[12]:
```
sklearn.svm._classes.SVC
```

[13]:
```
svm_classifier = SVC(random_state = 3432, C=0.5)
```

[14]:
```
svm_classifier.fit(flatshape, y)
```

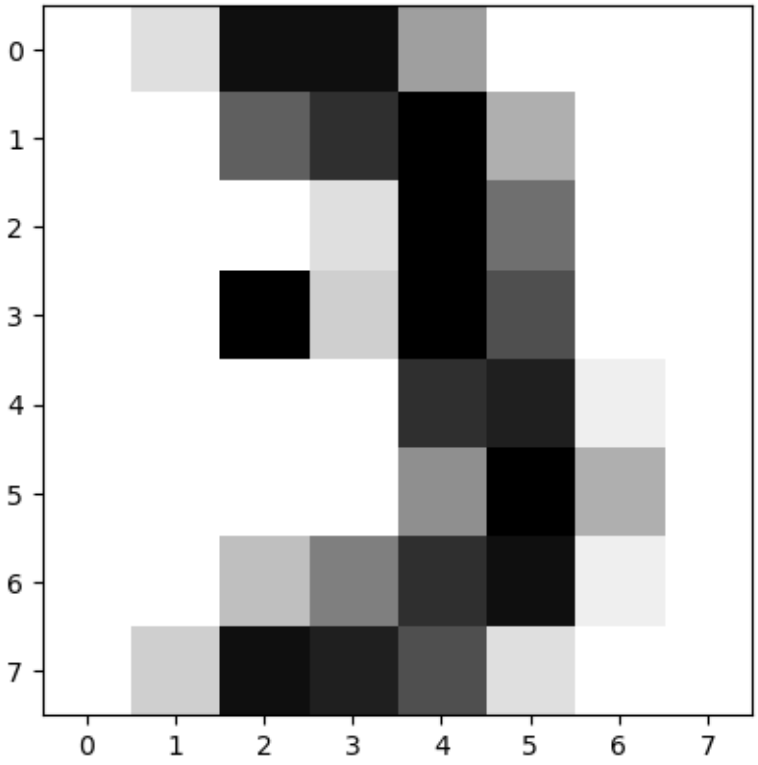
[14]:
```
SVC(C=0.5, random_state=3432)
```

[15]:
```
u = ([0, 2, 15, 15, 6, 0, 0, 0],
            [0, 0, 10, 13, 16, 5, 0, 0],
            [0, 0, 0, 2, 16, 9, 0, 0],
            [0, 0, 16, 3, 16, 11, 0, 0],
            [0, 0, 0, 0, 13, 14, 1, 0],
            [0, 0, 0, 0, 7, 16, 5, 0],
            [0, 0, 4, 8, 13, 15, 1, 0],
            [0, 3, 15, 14, 11, 2, 0, 0])
u = np.array(u)
u
```

[15]:
```
array([[ 0,  2, 15, 15,  6,  0,  0,  0],
       [ 0,  0, 10, 13, 16,  5,  0,  0],
       [ 0,  0,  0,  2, 16,  9,  0,  0],
       [ 0,  0, 16,  3, 16, 11,  0,  0],
       [ 0,  0,  0,  0, 13, 14,  1,  0],
       [ 0,  0,  0,  0,  7, 16,  5,  0],
```

```
       [ 0,  0,  4,  8, 13, 15,  1,  0],
       [ 0,  3, 15, 14, 11,  2,  0,  0]])
```

[16]:
```
plt.subplot()
plt.imshow(u, cmap = plt.cm.gray_r);
```



[17]:
```
u = u.reshape(-1)
u
```

[17]:
```
array([ 0,  2, 15, 15,  6,  0,  0,  0,  0,  0, 10, 13, 16,  5,  0,  0,  0,
        0,  0,  2, 16,  9,  0,  0,  0,  0, 16,  3, 16, 11,  0,  0,  0,  0,
        0,  0, 13, 14,  1,  0,  0,  0,  0,  0,  7, 16,  5,  0,  0,  0,  4,
        8, 13, 15,  1,  0,  0,  3, 15, 14, 11,  2,  0,  0])
```

[18]:
```
svm_classifier.predict([u])
```

[18]:
```
array([3])
```

[19]:
```
svm_new = SVC(random_state = 3432, C=10)
svm_new.fit(flatshape[:1001], y[:1001])
```

[19]:
```
SVC(C=10, random_state=3432)
```

[20]:
```
svm_new.predict(flatshape[1001 :])
```

[20]:
```
array([4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4,
       4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5,
       6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9,
       8, 9, 8, 4, 1, 7, 7, 3, 5, 1, 0, 0, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7,
       3, 3, 4, 6, 6, 6, 9, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3,
       2, 1, 5, 4, 6, 3, 1, 7, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9,
       6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 3, 8, 4, 9, 0, 8,
       9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
       0, 8, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 5, 4,
       1, 7, 7, 3, 5, 1, 0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3,
       4, 6, 6, 6, 4, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3, 2, 1,
       7, 4, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9, 6, 8,
       7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8, 9, 8,
       0, 9, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1,
       2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4, 1, 7,
       7, 3, 5, 1, 0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3, 4, 6,
       6, 6, 4, 9, 1, 5, 0, 9, 6, 2, 8, 3, 0, 0, 1, 7, 6, 3, 2, 1, 7, 4,
       6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5,
       4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8, 0, 1, 2, 3,
       4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5,
       6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4, 1, 7, 7, 3, 5, 1,
       0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3, 4, 6, 6, 6, 4, 9,
       1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3, 2, 1, 7, 4, 6, 3, 1, 3,
       9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2,
       8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5,
       1, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
       4, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4, 1, 7, 7, 3, 5, 1, 0, 0, 0, 2,
       7, 8, 2, 0, 1, 2, 6, 8, 3, 7, 7, 3, 4, 6, 6, 6, 9, 9, 1, 5, 0, 9,
       5, 2, 8, 0, 1, 7, 6, 3, 2, 1, 7, 9, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4,
       3, 1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 2, 5, 7, 3, 5, 9,
       4, 5, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 8, 4, 5,
       6, 7, 8, 9, 0, 1, 2, 5, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9,
       8, 9, 8, 4, 1, 7, 7, 3, 5, 1, 0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 8,
       8, 7, 5, 8, 4, 6, 6, 6, 4, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7,
       6, 3, 2, 1, 7, 4, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 5, 1, 4, 0, 5, 3,
       6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9,
       0, 8, 9, 8])
```

[21]:
```
y[1001:]
```

[21]:
```
array([4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4,
       4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5,
       6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9,
       8, 9, 8, 4, 1, 7, 7, 3, 5, 1, 0, 0, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7,
       3, 3, 4, 6, 6, 6, 4, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3,
```

```
2, 1, 7, 4, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9,
6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8,
9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4,
1, 7, 7, 3, 5, 1, 0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3,
4, 6, 6, 6, 4, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3, 2, 1,
7, 4, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9, 6, 1,
7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8, 9, 8,
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4, 1, 7,
7, 3, 5, 1, 0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3, 4, 6,
6, 6, 4, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3, 2, 1, 7, 4,
6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5,
4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8, 0, 1, 2, 3,
4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5,
6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4, 1, 7, 7, 3, 5, 1,
0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3, 4, 6, 6, 6, 4, 9,
1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7, 6, 3, 2, 1, 7, 4, 6, 3, 1, 3,
9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2,
8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5,
6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
0, 9, 5, 5, 6, 5, 0, 9, 8, 9, 8, 4, 1, 7, 7, 3, 5, 1, 0, 0, 2, 2,
7, 8, 2, 0, 1, 2, 6, 3, 3, 7, 3, 3, 4, 6, 6, 6, 4, 9, 1, 5, 0, 9,
5, 2, 8, 0, 1, 7, 6, 3, 2, 1, 7, 4, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4,
3, 1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 2, 5, 7, 9, 5, 4,
4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5,
6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 9, 5, 5, 6, 5, 0, 9,
8, 9, 8, 4, 1, 7, 7, 3, 5, 1, 0, 0, 2, 2, 7, 8, 2, 0, 1, 2, 6, 3,
3, 7, 3, 3, 4, 6, 6, 6, 4, 9, 1, 5, 0, 9, 5, 2, 8, 2, 0, 0, 1, 7,
6, 3, 2, 1, 7, 4, 6, 3, 1, 3, 9, 1, 7, 6, 8, 4, 3, 1, 4, 0, 5, 3,
6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5, 4, 8, 8, 4, 9,
0, 8, 9, 8])
```

[22]:
```
from sklearn import metrics
```

[23]:
```
p = svm_new.predict(flatshape[1001 :])
e = y[1001 :]
print(metrics.classification_report(e,p))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        79
           1       0.99      0.96      0.97        79
           2       1.00      0.97      0.99        77
           3       0.96      0.87      0.91        79
           4       0.99      0.95      0.97        83
           5       0.93      0.99      0.96        82
           6       0.99      0.99      0.99        80
```

```
              7       0.96      0.99      0.98        80
              8       0.91      0.97      0.94        76
              9       0.94      0.96      0.95        81

    accuracy                            0.96       796
   macro avg       0.97      0.96      0.96       796
weighted avg       0.97      0.96      0.96       796
```

As above accuracy is 96%.

In a confusion matrix, precision is the proportion of correctly predicted positive cases out of all instances predicted as positive, while recall is the proportion of correctly predicted positive cases out of all actual positive cases. Essentially, precision measures the accuracy of positive predictions, and recall measures the model's ability to find all relevant instances.

The F1-score is a metric that combines precision and recall into a single value, providing a balanced measure of a classification model's performance, especially useful for imbalanced datasets. t's calculated as 2 * (precision * recall) / (precision + recall).

support is the how many sample we have from each data.

macro avg is the usual mean of each cols. but weighted avg equals "sum of (each data (precision or recall or f1_score) * support of each data) / sum of support (796)".

[24]:
```
print(metrics.confusion_matrix(e,p))
```

```
[[78  0  0  0  1  0  0  0  0  0]
 [ 0 76  0  0  0  0  0  0  2  1]
 [ 1  0 75  1  0  0  0  0  0  0]
 [ 0  0  0 69  0  3  0  2  5  0]
 [ 0  0  0  0 79  0  0  0  0  4]
 [ 0  0  0  0  0 81  1  0  0  0]
 [ 0  1  0  0  0  0 79  0  0  0]
 [ 0  0  0  0  0  1  0 79  0  0]
 [ 0  0  0  1  0  1  0  0 74  0]
 [ 0  0  0  1  0  1  0  1  0 78]]
```

As above, the rows are expected and the cols are predicted. for example from the first row, it means that the expected values is 78 and our machine predict them sucessfully. and 1 one of them was 0 but our machine wrongly predicted number 4. and so on for all rows. So it means that the recall 0 is high (99%)

If we look at as cols. we see 75 in third col. it means that all the prediction of 2 was true. beacuse of that the precision of 2 is 100%.

it means that the rows are recall and the cols are precision.

0.0.1 So for better result there are some ways:

0.0.2 1. Increase datas: instead of 1797 data we need more than 10000 datas.

0.0.3 2. Increase dimensions: Instead of 8*8 matrix, we need more dimension matrix (more features).

0.0.4 3.Change algorithm

0.0.5 4. Hyperparameter tuning