MNIST Random Forest & Adaboost Classification

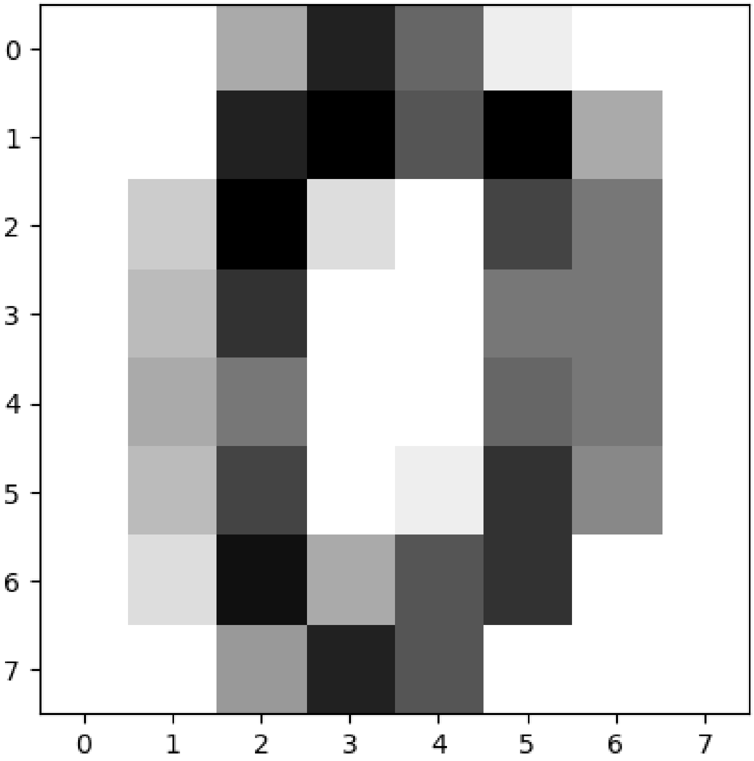
September 13, 2025

[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

[2]:

```
data = datasets.load_digits()
plt.imshow(data.images[0], cmap=plt.cm.gray_r);
```



[3]:

```
x = data.images.reshape(len(data.images), -1)
x
```

```
[3]: 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.]])
```

```
[4]: y = data.target
     y
```

```
[4]: array([0, 1, 2, ..., 8, 9, 8])
```

```
[5]: from sklearn.ensemble import RandomForestClassifier
```

```
[6]: rf_clf = RandomForestClassifier(n_estimators=1000, bootstrap=True,␣
      ↪oob_score=True)
     rf_clf.fit(x[:1000], y[:1000])
```

```
[6]: RandomForestClassifier(n_estimators=1000, oob_score=True)
```

This is a machine learning model from sklearn.ensemble used for classification tasks.

It builds an ensemble of decision trees and makes predictions based on the majority vote (for classification) of those trees.

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

```
[9]: p = rf_clf.predict(x[1000:])
     e = y[1000:]
     print(metrics.classification_report(e, p))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        79
           1       0.96      0.88      0.92        80
           2       1.00      0.90      0.95        77
           3       0.91      0.85      0.88        79
           4       0.98      0.95      0.96        83
           5       0.88      0.98      0.92        82
           6       0.98      1.00      0.99        80
           7       0.93      0.97      0.95        80
           8       0.88      0.91      0.90        76
           9       0.87      0.93      0.90        81

    accuracy                           0.93       797
   macro avg       0.94      0.93      0.93       797
weighted avg       0.94      0.93      0.93       797
```

[10]:

```
print(metrics.confusion_matrix(e,p))
```

```
[[78  0  0  0  1  0  0  0  0  0]
 [ 0 70  0  2  0  1  0  0  0  7]
 [ 1  1 69  4  0  0  0  0  0  2]
 [ 0  0  0 67  0  3  0  3  6  0]
 [ 0  0  0  0 79  1  0  2  0  1]
 [ 0  0  0  0  0 80  2  0  0  0]
 [ 0  0  0  0  0  0 80  0  0  0]
 [ 0  0  0  0  0  1  0 78  1  0]
 [ 0  2  0  0  1  2  0  1 69  1]
 [ 0  0  0  1  0  3  0  0  2 75]]
```

0.0.1 Now lets go to Adaboost classificaation

[11]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
```

[12]:

```
ab_clf = AdaBoostClassifier(base_estimator = SVC(probability=True,␣
 ↪kernel='rbf'), n_estimators = 100)
```

This line defines an AdaBoost classifier that uses a Support Vector Machine (SVM) with an RBF kernel as its base estimator.

0.0.2 Important Note:

SVMs are strong learners, not weak learners. Boosting typically works better with weak base models (like decision stumps). Using SVMs in AdaBoost:

Can be slow, especially with many estimators.

May overfit if not tuned properly.

If you're doing this for experimentation or research, it's okay. But for practical use, it's often better to use simpler models like DecisionTreeClassifier(max_depth=1) as the base estimator.

[19]:

```
ab_clf.fit(x[:1000], y[:1000]);
```

```
C:\Users\SPINO SHOP\anaconda3\Lib\site-packages\sklearn\ensemble\_base.py:156:
FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and
will be removed in 1.4.
  warnings.warn(
```

[20]:

```
p = ab_clf.predict(x[1000:])
e = y[1000:]
print(metrics.classification_report(e,p))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        79
           1       0.90      0.91      0.91        80
           2       0.97      0.87      0.92        77
           3       0.77      0.92      0.84        79
           4       0.91      0.96      0.94        83
           5       0.96      0.95      0.96        82
           6       0.99      0.93      0.95        80
           7       0.97      0.89      0.93        80
           8       0.89      0.87      0.88        76
           9       0.85      0.88      0.86        81

    accuracy                           0.92       797
   macro avg       0.92      0.92      0.92       797
weighted avg       0.92      0.92      0.92       797
```

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

```
[[77  0  0  0  1  0  1  0  0  0]
 [ 0 73  0  1  0  0  0  0  1  5]
 [ 0  0 67 10  0  0  0  0  0  0]
 [ 0  0  0 73  0  1  0  1  4  0]
 [ 0  0  0  0 80  0  0  0  0  3]
 [ 0  2  1  0  0 78  0  0  1  0]
 [ 0  5  0  0  0  0 74  0  1  0]
 [ 0  0  1  0  7  0  0 71  0  1]
 [ 0  1  0  3  0  1  0  1 66  4]
 [ 0  0  0  8  0  1  0  0  1 71]]
```