BinaryClassifier

March 10, 2021

Load the dataset from sklearn

[2]:
```
from sklearn.datasets import fetch_openml
mnist=fetch_openml('mnist_784',version=1)
mnist.keys()
```

[2]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names',
'target_names', 'DESCR', 'details', 'url'])
```

Let's learn more about the data...

[3]:
```
mnist.DESCR
```

[3]:
```
"**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges  \n**Source**:
[MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown  \n**Please
cite**:  \n\nThe MNIST database of handwritten digits with 784 features, raw
data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a
training set of the first 60,000 examples, and a test set of 10,000 examples
\n\nIt is a subset of a larger set available from NIST. The digits have been
size-normalized and centered in a fixed-size image. It is a good database for
people who want to try learning techniques and pattern recognition methods on
real-world data while spending minimal efforts on preprocessing and formatting.
The original black and white (bilevel) images from NIST were size normalized to
fit in a 20x20 pixel box while preserving their aspect ratio. The resulting
images contain grey levels as a result of the anti-aliasing technique used by
the normalization algorithm. the images were centered in a 28x28 image by
computing the center of mass of the pixels, and translating the image so as to
position this point at the center of the 28x28 field.  \n\nWith some
classification methods (particularly template-based methods, such as SVM and
K-nearest neighbors), the error rate improves when the digits are centered by
bounding box rather than center of mass. If you do this kind of pre-processing,
you should report it in your publications. The MNIST database was constructed
from NIST's NIST originally designated SD-3 as their training set and SD-1 as
their test set. However, SD-3 is much cleaner and easier to recognize than SD-1.
The reason for this can be found on the fact that SD-3 was collected among
Census Bureau employees, while SD-1 was collected among high-school students.
Drawing sensible conclusions from learning experiments requires that the result
be independent of the choice of training set and test among the complete set of
samples. Therefore it was necessary to build a new database by mixing NIST's
```

```
datasets.  \n\nThe MNIST training set is composed of 30,000 patterns from SD-3
and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from
SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained
examples from approximately 250 writers. We made sure that the sets of writers
of the training set and test set were disjoint. SD-1 contains 58,527 digit
images written by 500 different writers. In contrast to SD-3, where blocks of
data from each writer appeared in sequence, the data in SD-1 is scrambled.
Writer identities for SD-1 is available and we used this information to
unscramble the writers. We then split SD-1 in two: characters written by the
first 250 writers went into our new training set. The remaining 250 writers were
placed in our test set. Thus we had two sets with nearly 30,000 examples each.
The new training set was completed with enough examples from SD-3, starting at
pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new
test set was completed with SD-3 examples starting at pattern # 35,000 to make a
full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000
from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample
training set is available.\n\nDownloaded from openml.org."
```

The data consists of two important keys: data and target. Data - contains an array with one row per instance and one column per feature Target - contains an array with the labels (we're classifying information so labels are really important)

[4]:
```python
data, labels = mnist['data'], mnist['target']
```

Let's see what shapes we're dealing with…

[5]:
```python
data.shape
```

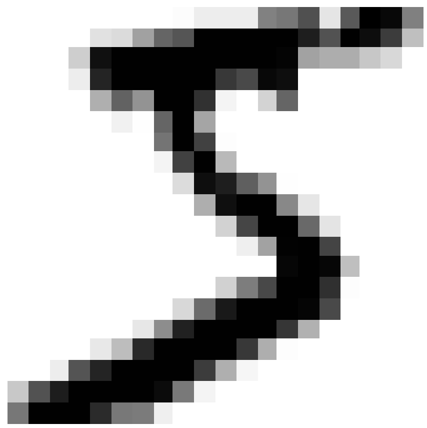
[5]:
```
(70000, 784)
```

[6]:
```python
labels.shape
```

[6]:
```
(70000,)
```

Let's look at one digit from the dataset. Each image has 784 features - 28x28. Let's grab and image's data, reshape it, and then display it.

[7]:
```python
import matplotlib as mpl
import matplotlib.pyplot as plt
digit = data[0]
digit_image = digit.reshape(28,28)
plt.imshow(digit_image, cmap='binary')
plt.axis('off')
plt.show()
```



```
[8]: labels[0]
```

```
[8]: '5'
```

The image kind of looks like a 5, and sure enough it is labeled as a 5. Because ML algorithms usually expect numbers, we should cast the labels as numbers.

```
[10]: import numpy as np
labels = labels.astype(np.uint8)
```

Before we start doing any manipulations, we should create our training set and test set. What does the description have to say about the dataset in regards to training/test?

```
[22]: data_train, data_test, labels_train,
labels_test = data[:60000], data[60000:], labels[:60000],
labels[60000:]
```

Break of the data based on fives and not fives

```
[14]: labels_train_5 = (labels_train == 5)
labels_test_5 = (labels_test == 5)
```

Back to notes...

Let's use a Stochastic Gradient Descent algorithm to train on identifying fives

```
[16]: from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state = 21)
```

Why random_state = 21?

[17]:
```
sgd_clf.fit(data_train, labels_train_5)
```

[17]:
```
SGDClassifier(random_state=21)
```

Model is trained...now, we can use it to predict the value of a digit

[18]:
```
sgd_clf.predict([digit])
```

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

This value was a five if you remember from before so we received what we expected. Back to notes...

[24]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, data_train, labels_train_5,
                cv = 3, scoring = 'accuracy')
```

[24]:
```
array([0.9649, 0.9674, 0.9583])
```

Wow! Well done!!! Let's call it a day!

Just kidding. We need to rethink this. Is Accuracy a valid metric for evaluating this model.

[26]:
```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, data, labels=None):
        return self
    def predict(self, data):
        return np.zeros((len(data), 1), dtype = bool)
```

[27]:
```
never_5_clf = Never5Classifier()
```

[28]:
```
cross_val_score(never_5_clf, data_train, labels_train_5,
               cv=3, scoring='accuracy')
```

[28]:
```
array([0.91125, 0.90855, 0.90915])
```

No, accuracy is not a good evaluator. What the above shows is that basically, only about 10% of the images are fives meaning that if you guessed an image was not a five, you would be right 90% of the time. It doesn't really give us a good idea of how accurate our model is. Let's evaluate a different way using a Confusion Matrix.

[30]:
```
from sklearn.model_selection import cross_val_predict
label_train_predict = cross_val_predict(sgd_clf, data_train,
                                       labels_train_5,
                                       cv = 3)
```

[32]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(labels_train_5, label_train_predict)
```

[32]:
```
array([[53875,   704],
       [ 1484,  3937]])
```

The above confusion matrix shows us how often the model was "confused." The first row is dealing with the negative class, the not a five class. The model accurately identified 53,875 not-5's as not a 5 (true negatives); however, it thought 704 images were fives, but they weren't (false positives - falsely identified as a positive result). It accurately identified 3,937 fives as fives (true positives), but it misidentified 1,484 fives as not fives (false negatives).

[ ]:
```python

```