Machine Learning

|Chapter 03 - Classification

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Classifier
- MNIST
- MINST Training & Testing set
- Training Binary Classifier
- Stochastic Gradient Descent

Intro

- The most **common supervised** learning tasks are regression and classification.
- Now we will turn our attention to **classification systems**.

MINST

- Set of **70,000** small images of digits handwritten by high school students and employees of the US Census Bureau
- Each image is labeled with the digit it represents
- It is often called the "**Hello World**" of Machine Learning
- Each image has **28×28** pixels (**784** features)
- Each feature simply represents one pixel's intensity, from 0 (white) to 255 (black)

MINST

- Scikit-Learn provides many helper functions to download popular datasets
- The following code fetches the MNIST dataset from OpenML.org

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', as_frame=False)
```

MINST

- The sklearn.datasets package contains mostly three types of functions:
  - fetch_* functions such as fetch_openml() to download real-life datasets,
  - load_* functions to load small toy datasets bundled with Scikit-Learn (so they don't need to be downloaded over the internet), and
  - make_* functions to generate fake datasets, useful for tests

MINST

- Generated datasets are usually returned as an (X, y) tuple containing the input data and the targets, both as NumPy arrays.
- Other datasets are returned as sklearn.utils.Bunch objects, which are dictionaries whose entries can also be accessed as attributes. They generally contain the following entries:

```
mnist.keys()
```

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

```
print(mnist.DESCR)
```

MINST

```
X, y = mnist.data, mnist.target
```

```
X
```

```
array([[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.]])
```

```
X.shape
```

```
(70000, 784)
```

```
y
```

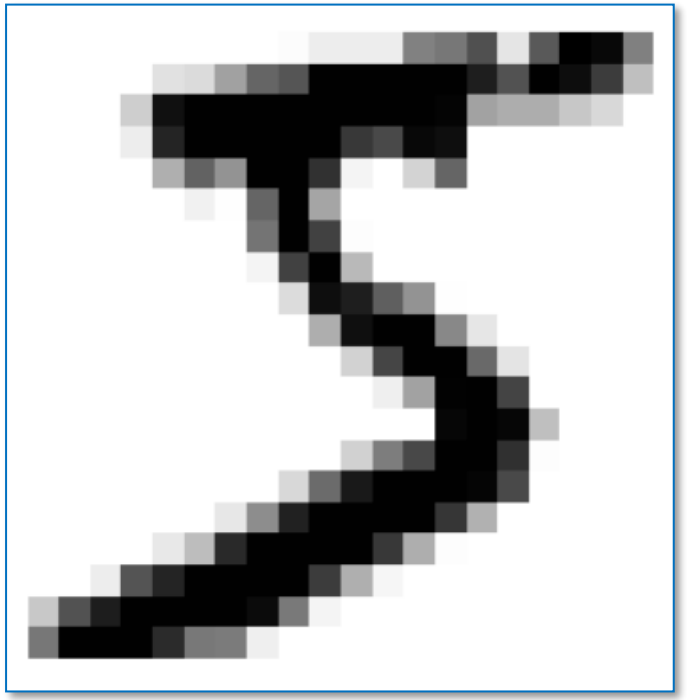
```
array(['5', '0', '4', ..., '4', '5', '6'], dtype=object)
```

MINST

```
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image = image_data.reshape(28, 28)
    plt.imshow(image, cmap="binary")
    plt.axis("off")

some_digit = X[0]
plot_digit(some_digit)
save_fig("some_digit_plot")  # extra code
plt.show()
```



```
y[0]
```

```
'5'
```

MINST

```
plt.figure(figsize=(9, 9))
for idx, image_data in enumerate(X[:100]):
    plt.subplot(10, 10, idx + 1)
    plot_digit(image_data)
plt.subplots_adjust(wspace=0, hspace=0)
save_fig("more_digits_plot", tight_layout=False)
plt.show()
```

MINST Training & Testing set

- The MNIST dataset is actually already **split into** a training set (the first 60,000 images) and a test set (the last 10,000 images):
- **Shuffled**; this will guarantee that…
  - All cross-validation folds will be **similar** (you don't want one fold to be missing some digits).
  - Moreover, some learning algorithms are **sensitive** to the order of the training instances, and they perform poorly if they get many similar instances in a row. Shuffling the dataset ensures that this won't happen:

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Training Binary Classifier

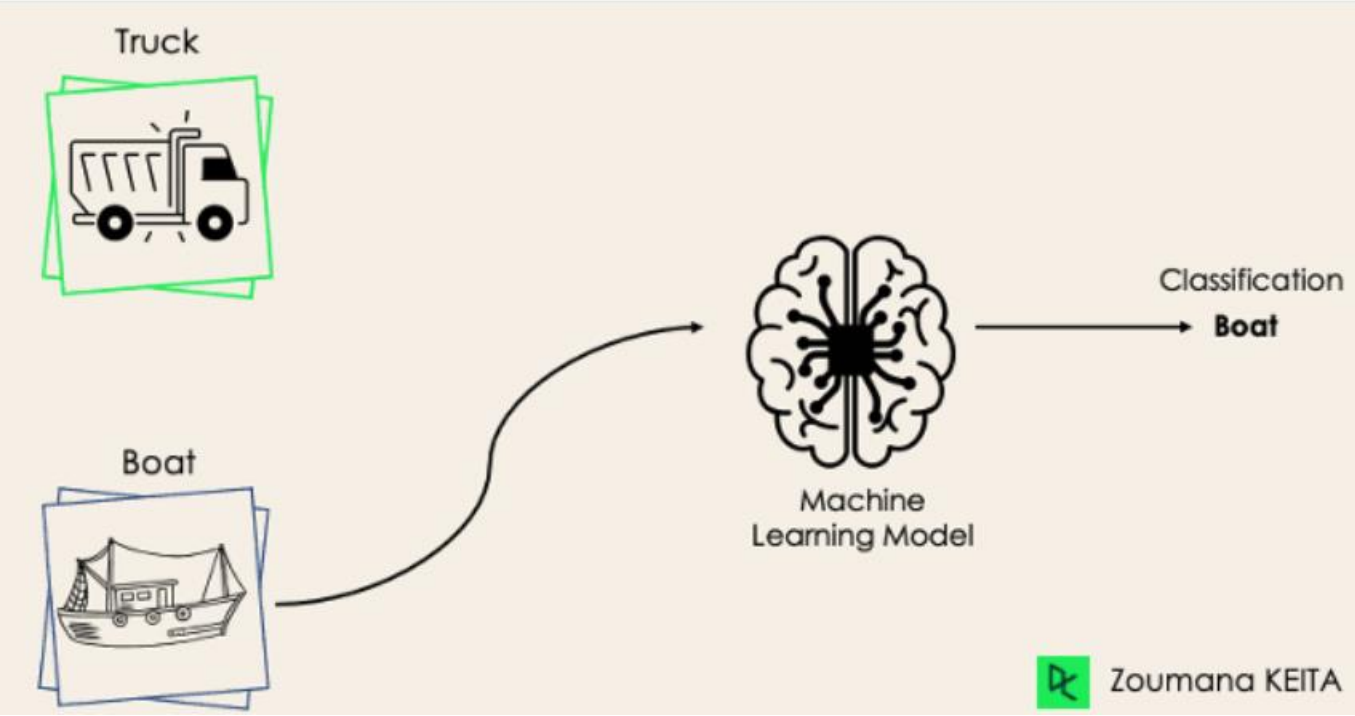
- Let's simplify the problem for now and only try to identify one digit — for example, the number 5.
- This "**5-detector**" will be an example of a binary classifier, capable of **distinguishing** between just two classes, **5 and not-5**.

```
y_train_5 = (y_train == '5')  # True for all 5s, False for all other digits
y_test_5 = (y_test == '5')
```

- Okay, now let's pick a classifier and train it. A good place to start is with a Stochastic Gradient Descent (SGD) classifier, using Scikit-Learn's SGDClassifier class

Binary Classifier

- Binary classifiers distinguish between two classes



Stochastic Gradient Descent

- The classifier is capable of handling very large datasets efficiently. This is in part because SGD deals with training instances independently, one at a time, which also makes SGD well suited for online learning, as you will see later

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

```
SGDClassifier
SGDClassifier(random_state=42)
```

Stochastic Gradient Descent

- The SGDClassifier relies on randomness during training (hence the name “stochastic”).
- If you want reproducible results, you should set the random_state parameter.

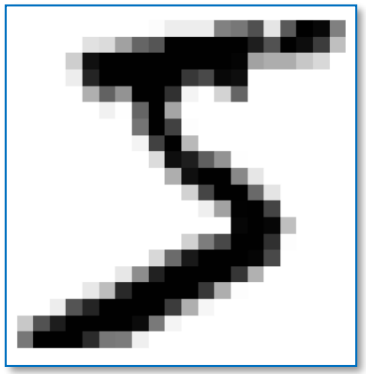
```
sgd_clf.predict([some_digit])
```

```
array([ True])
```

```
some_digit = X[0]
```

```
y[0]
```

```
'5'
```



- The classifier guesses that this image represents a 5 (True)

Next Lecture

- Performance Measures