ML: Let's Look at Data

Slide Set 2

"Hands-On Machine Learning with Scikit-Learn, Keras, & TensorFlow", Geron

For example, suppose you want to know if money makes people happy, so you download the Better Life Index data from the OECD's website and World Bank stats about gross domestic product (GDP) per capita. Then you join the tables and sort by GDP per capita. Table 1-1 shows an excerpt of what you get.

Table 1-1. Does money make people happier?

Country	GDP per capita (USD)	Life satisfaction
Turkey	28,384	5.5
Hungary	31,008	5.6
France	42,026	6.5
United States	60,236	6.9
New Zealand	42,404	7.3
Australia	48,698	7.3
Denmark	55,938	7.6

Let's plot the data for these countries (Figure 1-18).

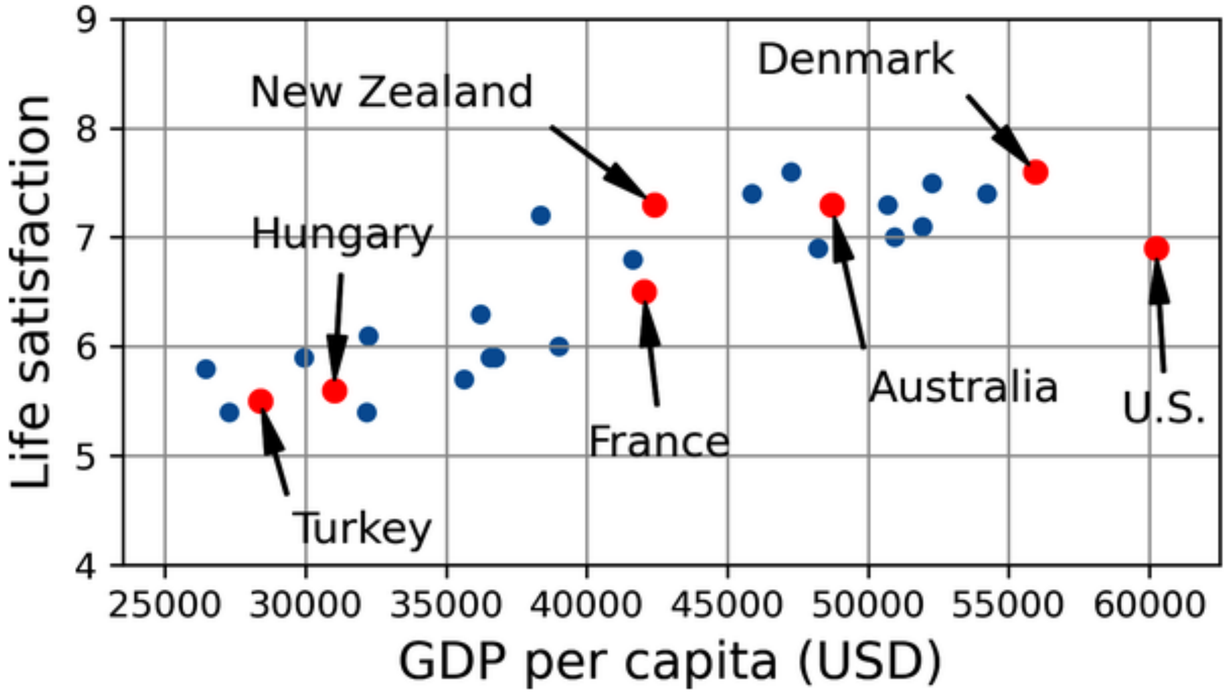

Figure 1-18. Do you see a trend here?

There does seem to be a trend here! Although the data is *noisy* (i.e., partly random), it looks like life satisfaction goes up more or less linearly as the country's GDP per capita increases. So you decide to model life satisfaction as a linear function of GDP per capita. This step is called *model selection*: you selected a *linear model* of life satisfaction with just one attribute, GDP per capita (Equation 1-1).

Equation 1-1. A simple linear model

$$\text{life\_satisfaction} = \theta_0 + \theta_1 \times \text{GDP\_per\_capita}$$

This model has two *model parameters*, θ_0 and θ_1.[4] By tweaking these parameters, you can make your model represent any linear function, as shown in Figure 1-19.

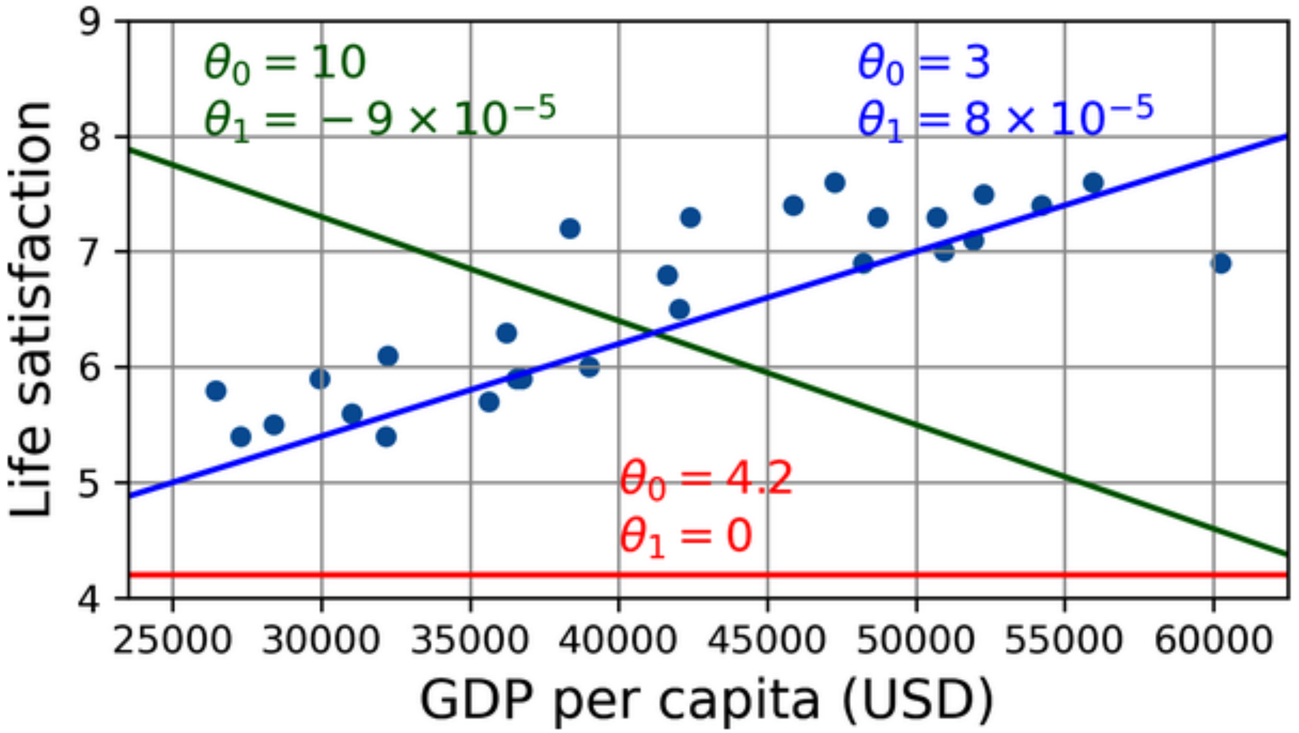

Figure 1-19. A few possible linear models

Before you can use your model, you need to define the parameter values θ_0 and θ_1. How can you know which values will make your model perform best? To answer this question, you need to specify a performance measure. You can either define a *utility function* (or *fitness function*) that measures how *good* your model is, or you can define a *cost function* that measures how *bad* it is. For linear regression problems, people typically use a cost function that measures the distance between the linear model's predictions and the training examples; the objective is to minimize this distance.

This is where the linear regression algorithm comes in: you feed it your training examples, and it finds the parameters that make the linear model fit best to your data. This is called *training* the model. In our case, the algorithm finds that the optimal parameter values are $\theta_0 = 3.75$ and $\theta_1 = 6.78 \times 10^{-5}$.

WARNING

Confusingly, the word "model" can refer to a *type of model* (e.g., linear regression), to a *fully specified model architecture* (e.g., linear regression with one input and one output), or to the *final trained model* ready to be used for predictions (e.g., linear regression with one input and one output, using $\theta_0 = 3.75$ and $\theta_1 = 6.78 \times 10^{-5}$). Model selection consists in choosing the type of model and fully specifying its architecture. Training a model means running an algorithm to find the model parameters that will make it best fit the training data, and hopefully make good predictions on new data.

Now the model fits the training data as closely as possible (for a linear model), as you can see in Figure 1-20.

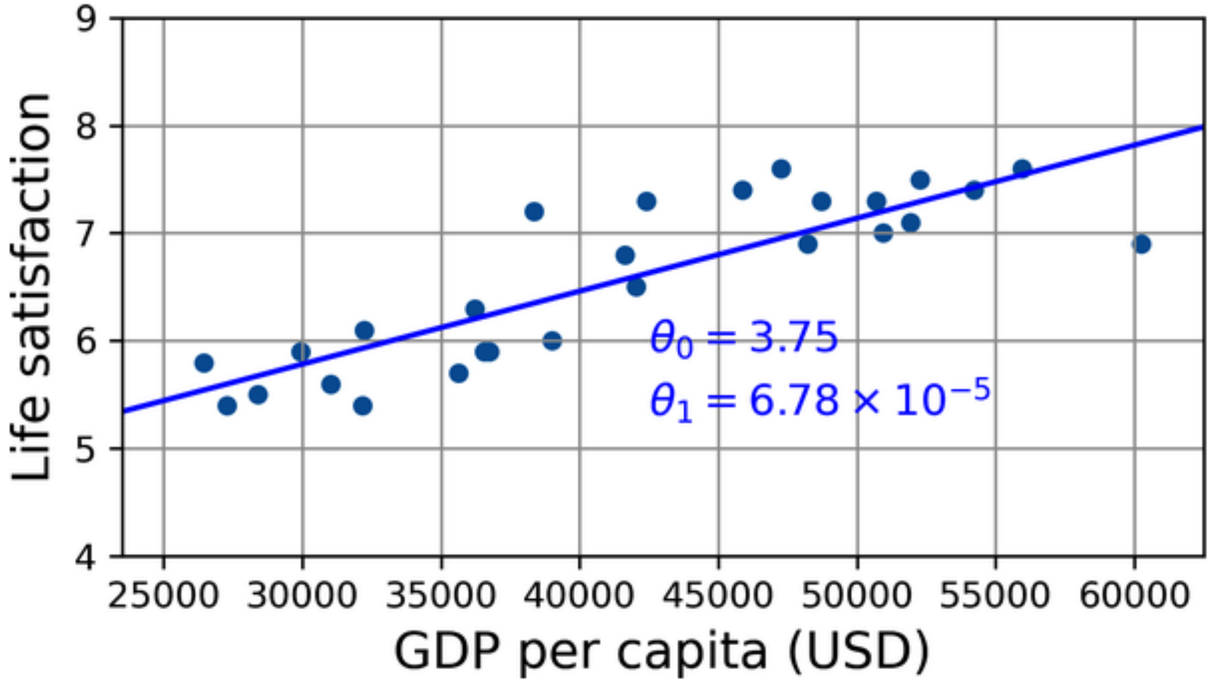

Figure 1-20. The linear model that fits the training data best

You are finally ready to run the model to make predictions. For example, say you want to know how happy Cypriots are, and the OECD data does not have the answer. Fortunately, you can use your model to make a good prediction: you look up Cyprus's GDP per capita, find $37,655, and then apply your model and find that life satisfaction is likely to be somewhere around $3.75 + 37{,}655 \times 6.78 \times 10^{-5} = 6.30$.

To whet your appetite, Example 1-1 shows the Python code that loads the data, separates the inputs `X` from the labels `y`, creates a scatterplot for visualization, and then trains a linear model and makes a prediction.[5]

Example 1-1. Training and running a linear model using Scikit-Learn

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

# Download and prepare the data
data_root = "https://github.com/ageron/data/raw/main/"
lifesat = pd.read_csv(data_root + "lifesat/lifesat.csv")
X = lifesat[["GDP per capita (USD)"]].values
y = lifesat[["Life satisfaction"]].values

# Visualize the data
lifesat.plot(kind='scatter', grid=True,
             x="GDP per capita (USD)", y="Life satisfaction")
plt.axis([23_500, 62_500, 4, 9])
plt.show()

# Select a linear model
model = LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[37_655.2]]  # Cyprus' GDP per capita in 2020
print(model.predict(X_new)) # output: [[6.30165767]]
```

Replacing the linear regression model with *k*-nearest neighbors regression in the previous code is as easy as replacing these lines:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

with these two:

```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3)
```

If all went well, your model will make good predictions. If not, you may need to use more attributes (employment rate, health, air pollution, etc.), get more or better-quality training data, or perhaps select a more powerful model (e.g., a polynomial regression model).

In summary:

- You studied the data.
- You selected a model.
- You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
- Finally, you applied the model to make predictions on new cases (this is called *inference*), hoping that this model will generalize well.

Main Challenges of Machine Learning

In short, since your main task is to select a model and train it on some data, the two things that can go wrong are "bad model" and "bad data". Let's start with examples of bad data.

Insufficient Quantity of Training Data

For a toddler to learn what an apple is, all it takes is for you to point to an apple and say "apple" (possibly repeating this procedure a few times). Now the child is able to recognize apples in all sorts of colors and shapes. Genius.

Machine learning is not quite there yet; it takes a lot of data for most machine learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

THE UNREASONABLE EFFECTIVENESS OF DATA

In a famous paper published in 2001, Microsoft researchers Michele Banko and Eric Brill showed that very different machine learning algorithms, including fairly simple ones, performed almost identically well on a complex problem of natural language disambiguation[6] once they were given enough data (as you can see in Figure 1-21).