

An Example for Model-Based Machine Learning

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](#) as well as stats about GDP per capita from the [IMF's website](#). 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
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

Let's plot the data for a few random countries ([Figure 1-17](#)).

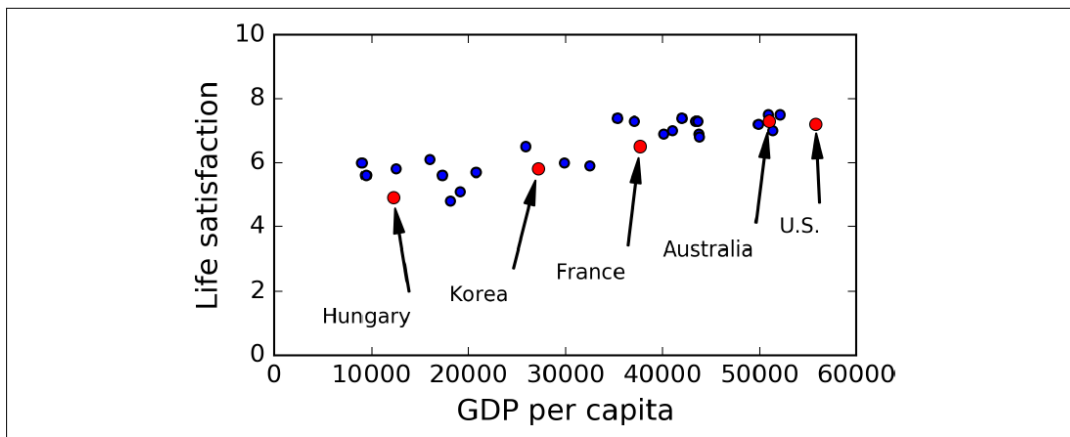


Figure 1-17. Do you see a trend here?

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

$$life_satisfaction = \theta_0 + \theta_1 \times GDP_per_capita$$

This model has two *model parameters*, θ_0 and θ_1 .⁵ By tweaking these parameters, you can make your model represent any linear function, as shown in Figure 1-18.

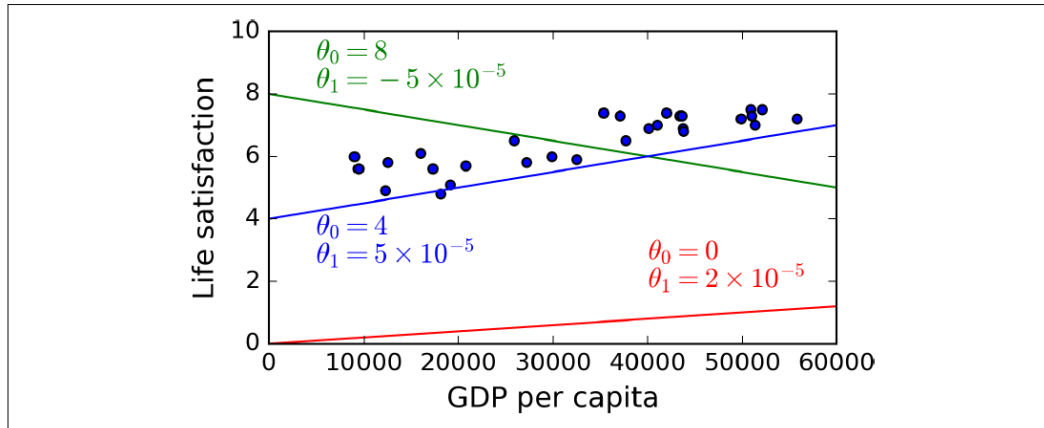


Figure 1-18. A few possible linear models

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 $\theta_0 = 4.85$ and $\theta_1 = 4.91 \times 10^{-5}$.

Now the model fits the training data as closely as possible (for a linear model), as you can see in Figure 1-19.

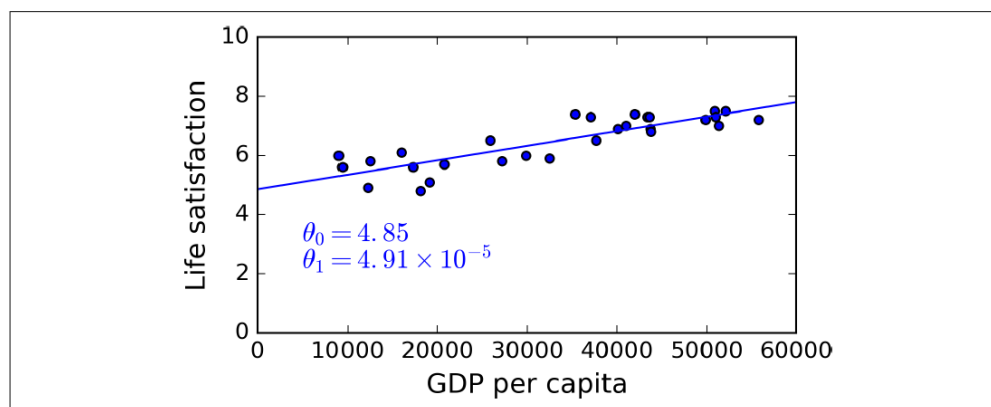


Figure 1-19. The linear model that fits the training data best

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 \$22,587, and then apply your model and find that life satisfaction is likely to be somewhere around $4.85 + 22,587 \times 4.91 \times 10^{-5} = 5.96$.

To whet your appetite, [Example 1-1](#) shows the Python code that loads the data, prepares it,⁶ creates a scatterplot for visualization, and then trains a linear model and makes a prediction.⁷

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 \$22,587, and then apply your model and find that life satisfaction is likely to be somewhere around $4.85 + 22,587 \times 4.91 \times 10^{-5} = 5.96$.

To whet your appetite, [Example 1-1](#) shows the Python code that loads the data, prepares it,⁶ creates a scatterplot for visualization, and then trains a linear model and makes a prediction.⁷

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

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

# Load the data
oecd_bli = pd.read_csv("oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv("gdp_per_capita.csv",thousands=',',delimiter='\t',
                             encoding='latin1', na_values="n/a")

# Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

# Visualize the data
country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
plt.show()

# Select a linear model
lin_reg_model = sklearn.linear_model.LinearRegression()

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

# Make a prediction for Cyprus
X_new = [[22587]] # Cyprus' GDP per capita
print(lin_reg_model.predict(X_new)) # outputs [[ 5.96242338]]
```



If you had used an instance-based learning algorithm instead, you would have found that Slovenia has the closest GDP per capita to that of Cyprus (\$20,732), and since the OECD data tells us that Slovenians' life satisfaction is 5.7, you would have predicted a life satisfaction of 5.7 for Cyprus. If you zoom out a bit and look at the two next closest countries, you will find Portugal and Spain with life satisfactions of 5.1 and 6.5, respectively. Averaging these three values, you get 5.77, which is pretty close to your model-based prediction. This simple algorithm is called *k-Nearest Neighbors* regression (in this example, $k = 3$).

Replacing the Linear Regression model with k-Nearest Neighbors regression in the previous code is as simple as replacing this line:

```
clf = sklearn.linear_model.LinearRegression()
```

with this one:

```
clf = sklearn.neighbors.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.

This is what a typical Machine Learning project looks like. In [Chapter 2](#) you will experience this first-hand by going through an end-to-end project.

We have covered a lot of ground so far: you now know what Machine Learning is really about, why it is useful, what some of the most common categories of ML systems are, and what a typical project workflow looks like. Now let's look at what can go wrong in learning and prevent you from making accurate predictions.

*This example comes from Chapter 1 of the book, titled "Hands-On Machine Learning with Scikit-Learn & TensorFlow", by Aurelien Geron, 2017 O'Reilly.