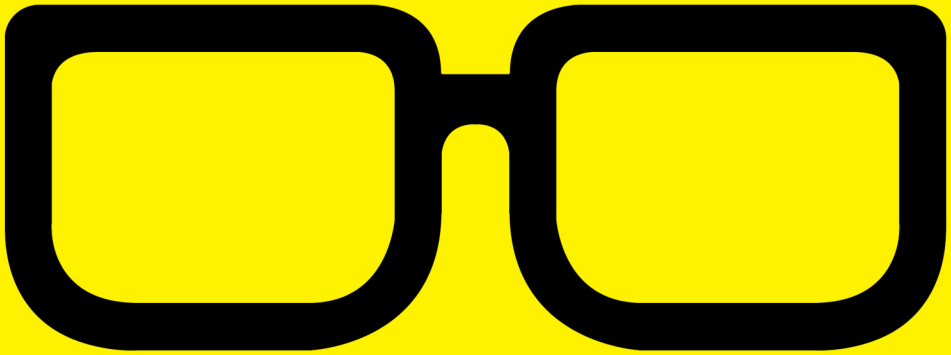
SPONSORED BY


GEEK GUIDE



# Machine Learning with Python

LINUX™ JOURNAL

# Table of Contents



Reuven M. Lerner offers training in Python, Git and PostgreSQL to companies around the world. He blogs at blog.lerner.co.il, tweets at @reuvenmlerner and curates DailyTechVideo.com. Reuven lives in Modi'in, Israel, with his wife and three children.

**GEEK GUIDES:**
Mission-critical information for the most technical people on the planet.

***Copyright Statement***
© 2016 *Linux Journal*. All rights reserved.

This site/publication contains materials that have been created, developed or commissioned by, and published with the permission of, *Linux Journal* (the "Materials"), and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of *Linux Journal* or its Web site sponsors. In no event shall *Linux Journal* or its sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

No part of the Materials (including but not limited to the text, images, audio and/or video) may be copied, reproduced, republished, uploaded, posted, transmitted or distributed in any way, in whole or in part, except as permitted under Sections 107 & 108 of the 1976 United States Copyright Act, without the express written consent of the publisher. One copy may be downloaded for your personal, noncommercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

*Linux Journal* and the *Linux Journal* logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners. If you have any questions about these terms, or if you would like information about licensing materials from *Linux Journal*, please contact us via e-mail at info@linuxjournal.com.

## About the Sponsor

### Intel® Software and Services Group

The Intel® Software and Services Group (SSG) employs thousands of software-focused professionals, and measured by engineering staff size, SSG would be among the world's top 10 software companies if it were an independent organization.

Recognizing that software is tightly coupled with, and a vital element of, all Intel® platforms and processors, SSG is a worldwide provider of software products and services, design resources, technical expertise and consulting. SSG primarily works with software companies such as Adobe, Microsoft and Oracle, and directly with CIOs of major corporations such as DreamWorks and Reuters Financial, as well as with individual software developers.

Through SSG's comprehensive enabling efforts, the software community can take maximum advantage of Intel® processor technologies across the computing spectrum from the Intel® Atom™ processor in small form factor mobile computing to Intel® Core™ processor and Intel® Xeon® processor families in computers, servers and entire IT infrastructures. SSG works with developers to enhance innovation and gain the best possible performance, uptime and efficiency. In addition, SSG is an integral part of the microprocessor design process, ensuring software requirements are comprehended in the development of future architectures and silicon designs.

# Machine Learning with Python

REUVEN M. LERNER

I first heard the term "machine learning" a few years ago, and to be honest, I basically ignored it that time. I knew that it was a powerful technique, and I knew that it was in vogue, but I didn't know what it really was—what problems it was designed to solve, how it solved them and how it related to the other sorts of issues I was working on in my professional (consulting) life and in my graduate-school research.

But in the past few years, machine learning has become a topic that most will avoid at their professional peril. Despite the scary-sounding name, the ideas behind machine learning aren't that difficult to understand. Moreover, a great deal of open-source software makes it possible for anyone to use machine learning in their own work or

Human minds basically are pattern-matching machines and excel at finding commonalities among different types of inputs; getting a computer to perform such categorization tasks is more than just an impressive trick.

research. I don't think it's an overstatement to say that machine learning already is having a huge impact on the computer industry and on our day-to-day lives.

In this ebook, I introduce the basic ideas behind machine learning and show how you can use Python to apply machine learning ideas to a number of different problems. I hope by the time you finish reading this guide, you'll not only understand what machine learning aims to do, but also how to apply it to your own work and research.

## What Is Machine Learning?

Before doing anything else, let's define the terms: "machine learning" sounds somewhat ominous, leading to a Matrix-like world in which the machines have taken over. But machine learning, at least as our current world sees it, is a mechanism by which computers can put inputs into categories.

Wait, that's it? No, but that's a very good starting point for thinking about machine learning.

Human minds basically are pattern-matching machines

and excel at finding commonalities among different types of inputs; getting a computer to perform such categorization tasks is more than just an impressive trick. It means that computers can look through a large number of inputs and try to categorize those inputs.

And, of course, if there's something that computers do better than people, it's look through large quantities of data.

A related use of machine learning is to predict outputs based on inputs with some degree of certainty. So if I present you with an input value—a child's age, for example—then you can predict that child's height. Will your prediction be exact? No, but that's okay; machine learning uses statistical reasoning. Thus, you're looking for likely outcomes, not definite outcomes.

Because this is something that statisticians have been doing for years, there definitely are people who ask how machine learning is different from just statistics. One possible answer is that regression, one of the cornerstones of statistics, is just one type of model used in machine learning.

For example, let's say you're a credit-card company and you're trying to determine whether a purchase is legitimate or fraudulent. Too many false positives, and your customers will be angry. Too many false negatives, and you'll soon be out of business. Machine learning makes it possible to analyze someone's purchase history and determine whether a purchase is likely to be good or bad.

Another common and famous example is that of identifying e-mail spam. It used to be that spam was not only obnoxious, but also easy to identify. Today, spammers use a variety of techniques to make their e-mail

look legitimate. Machine learning allows a computer to accumulate information over time, getting an increasingly clear picture of what is considered a legitimate message.

And of course, if you've bought anything on-line in the last decade, you've likely been told that "people who bought this product also bought...", followed by a long list of things that, when you think about it, actually are of interest to you. This sort of categorization also can be attacked using machine learning. As more information is fed into the system, it can make increasingly accurate predictions of what someone is likely to want to buy (or already has bought from another store).

As you can see, the number and types of problems that can be solved using machine learning is large and varied. Consider going back to when Claude Shannon and others first proposed that people could encode boolean logic in electrical circuits. Would you have imagined that today we would be holding powerful computers (mobile phones) in our pockets, sharing videos and e-mail messages effortlessly and globally? In the same way, we're only at the start of a revolution in machine learning, and it remains to be seen just how far this will go.

There are, of course, some ways machine learning has been used with, well, interesting results. Those results don't mean the technology is necessarily wrong, but rather that statistical models provide likelihoods, not certainties. Uncertainty, matched with a large population, can create some awkward situations.

One famous, early example involved TiVo, a digital video recorder that chose what to watch based on your viewing

Moreover, we're seeing how our own organizations can use machine learning to sell more products, understand customer needs and even improve medical outcomes.

patterns. A *Wall Street Journal* article from 2002 was titled "My TiVo Thinks I'm Gay", and described someone trying to convince his TiVo that his choice in television programs was other than what the box's algorithms had determined.

Another famous case involved Target, which sent "so you're expecting" coupons to a customer based on her purchasing patterns, which told the machine learning algorithms that she was pregnant and would appreciate receiving such discounts. What Target's computer didn't realize was that the customer in question was a teenage girl who hadn't told her parents about the pregnancy. The parents, first irate at Target for making such seemingly unfair inferences, later directed their anger at their daughter.

The social, ethical and business ramifications of machine learning have yet to be determined. And yet, we're starting to see how large companies and organizations are using machine learning to sell more, keep people healthy and make everyone more productive.

Moreover, we're seeing how our own organizations can use machine learning to sell more products, understand customer needs and even improve medical outcomes.

Machine learning is a new application of statistical modeling. For many people, the term "statistical modeling" might not mean much, despite its demonstrated depth and power through many decades. But modeling is an important field, allowing people to describe and understand the world, or certain features of it. Statistical modeling lets you use previously collected data to make reasonable inferences and predictions about future data.

For example, the United States is now in the middle of an election season. Before every debate, primary and caucus, numerous polls make predictions about who will win—and for the most part, they can predict things accurately. But in some cases, the pollsters say they don't have enough information from previous years' elections to make a reasonable prediction. Or, they may make predictions despite a lack of earlier data and end up with egg on their faces.

Because machine learning is based on statistical modeling, it makes certain assumptions. First and foremost, it tries to find correlations among data, but doesn't claim to find causality. This is a well known statement, and one that every elementary statistics class attempts to teach—and yet, human instincts drive us toward seeing causality even when that's far from demonstrated.

Machine learning also works, as I wrote earlier, only when there is some data with which to consider. Is someone a likely terrorist risk? Is this envelope meant to be delivered to Main Street or to Maine? Are you really a good potential customer for a new romance novel? All of those questions can be solved, to some degree, with machine learning—assuming that the system has sufficient inputs. Amazon's

first customer wasn't recommended any books, because that functionality didn't exist. But even if the software had existed, it wouldn't have been possible to get a reasonable recommendation, because there weren't yet any purchases.

And of course, machine learning is only as good as the input data. If your input data has a limited number of factors, or those factors aren't enough to distinguish between elements of your data set, machine learning won't be able to do much for you. If your data set contains a great deal of noise, or outliers, then machine learning might not help.

## Supervised vs. Unsupervised Learning

I've already described machine learning as a way of having a computer put input data into categories. At the same time, I indicated that the "learning" part of machine learning comes from the fact that the system's model improves with time, as it gets more (good) input data. However, there's still the question of how the computer is supposed to know how to categorize things.

For example, let's take a set of people. Say you have a bunch of information about them, including gender, age, height, weight and nationality. In most cases, you won't want to use all of those factors. The type of categories into which you want to sort data will drive the factors you use. Thus, if you want to categorize by driving ability, you'll use different factors from expected adult height, which is different from the number of languages you can expect the person to know.

There are two basic approaches to categorization, and each has its uses. In supervised learning, you take an initial data set and categorize each element. These initial

Unsupervised learning can be used to find correlations that people ordinarily wouldn't expect. It can be used to find potential customers or for the ubiquitous "people who bought X also bought Y" recommendation systems.

assignments are the "supervised" part of the learning. You can think of it as analogous to teaching a young child the letter A. You show many different forms of A, until the child is able to recognize a variety of shapes and forms of A.

Supervised learning is a good choice when you have some initial samples and want to categorize additional samples. For example, some spam filtering systems used to ask you to feed in good e-mail messages, so they would have a sense of what was considered non-spam; this was a form of supervised learning.

In unsupervised learning, by contrast, the computer is asked to divide the data set into a number of groups without a training set. You then can know that the data is divided into several different groups, based on the factor or factors you have identified.

Unsupervised learning can be used to find correlations that people ordinarily wouldn't expect. It can be used to find potential customers or for the ubiquitous "people who bought X also bought Y" recommendation systems. In unsupervised learning, the idea is that the model is able

to crunch through enough distinct and useful data inputs that it can categorize the data. People need to describe the categorization that takes place, but the computer can try to maximize the clustering of the data points.

## Models: the Core of Machine Learning

Machine learning, as I've already indicated, is a special case of statistical modeling. In a statistical model, you assume that members of your population have different values, and that you can define a function describing a line separating those values into different groups.

For example, assume that your input data contains height and age information about two groups of children. Each child is either between the ages of 2–5 or between the ages of 15–18. You can imagine plotting the ages and heights of those children.

With this population, it's probably fair to say that given someone's age, you can predict height. In such a scenario, you would say that age is the independent variable, and height is the dependent variable. In mathematical terms, you could say:

```
height = f(age)
```

This function will not predict everyone's height perfectly accurately, but it will be fairly close. As a statistical model, it'll tell the likely height given someone's age, within a certain margin of error. If you have children, you likely took them as babies for a check-up. At that check-up, your baby's height and weight were compared against such a plot to

ensure that he or she was on a reasonable growth path.

You can do something else with this data set as well. You can define a function that, given a new data point, can categorize it into either the younger (2–5) group or the older (15–18) group.

But of course, populations generally aren't divided into such clearly distinct categories. If your inputs consisted of children throughout the age range of 2–18, you still would be able to make some predictions about their height based on age. But the line between the younger group and the older group would be much harder to determine.

Even with this expanded group, you can say that there is a correlation—that age plays a role in determining height. You can say that the older the children are, the more likely they are to be taller. But, there will be some children who even at age 12 are taller than others at age 18. With such real-world data, categorizing children into "younger" and "older" groups based on height becomes more difficult, with a large number of errors.

In machine learning, you aim to find a model that can produce an output based on an input—or more often, a large set of outputs based on a large set of inputs. Different models, using different techniques and algorithms, will come up with different measurements.

In the end, all of these models reduce your input data to numbers or groups of numbers on which the algorithm can operate. Thus, a spam filter isn't comparing words; it's comparing the result of a function that operates on words. However, that function might look at each individual word, combinations of two, three or four words, or even

the combination of all words in an e-mail message, before deciding whether a message is spam.

Creating and refining these models and adjusting the parameters used to invoke the model, as well as the way in which you process the input data, is a key part of machine learning. Moreover, it's important to have a way to check your model's accuracy. It might seem to do a good job of categorizing your input data, but is it really that powerful?

## Python and scikit-learn

You could, of course, invent all of this yourself. However, one of the reasons why Python has become such a popular language among data scientists is the extensive set of libraries available that have already done so for you.

The Python package for machine learning is known as scikit-learn. There are also other high-performance machine learning library alternatives, including H2O and Intel Data Analytics Acceleration Library, which can offer additional functionality and performance.

The sckikit-learn package is available for download from the Python Package Index (PyPI) and can be installed using the Python `pip` command-line utility. scikit-learn depends on a number of other Python libraries written and optimized for use in mathematics and science. The most fundamental of those is NumPy, which implements a "NumPy array" data type (not to be confused with Python's built-in "array" data type). NumPy arrays provide a thin layer of Python on top of a C implementation. Operations, thus, run quite quickly and consume far less memory than a standard Python

Note that although standard distributions of Python are not known for performance, there are vendor-supplied distributions like the free Intel Distribution for Python and Continuum Anaconda that can yield dramatic NumPy and SciPy performance improvements.

list would require.

On top of NumPy sits SciPy, implementing many hundreds (or perhaps thousands?) of functions useful to anyone doing scientific calculations. SciPy's functions use NumPy as their underlying data structure.

Note that although standard distributions of Python are not known for performance, there are vendor-supplied distributions like the free Intel Distribution for Python and Continuum Anaconda that can yield dramatic NumPy and SciPy performance improvements.

SciPy offers the possibility of plugins, known as scikits. scikit-learn is, thus, a plugin for SciPy that implements a variety of algorithms to assist in the creation of machine learning models. Moreover, the API to each of the implementations is the same, meaning that once you learn how to use scikit-learn with one type of model, using other types of models shouldn't be particularly difficult.

And indeed, one of the more important choices to make in machine learning is the type of model you want to use.

Sometimes, the choice is obvious. But in many cases, you can choose from different models, either implementing different algorithms or using the same algorithm with different parameters. In each case, you'll want to test your model to make sure you aren't fooling yourself into thinking the model works when it doesn't.

One common problem with models is that of "overfitting". An overfit is when a model does perfectly describe and classify the test data, but only because the model is custom-designed to that test data. The moment the model encounters real-world data, it demonstrates its brittleness. The model is so tracked to the training data, it's incapable of truly learning.

Imagine teaching a child to recognize the letter A, but only using serif typefaces and making it very clear that those serifs (the little lines that appear on some letters) are a crucial part of the identification process. That child won't be able to recognize an A in a sans-serif font. Sometimes, it's better for a model to be less specific in order to identify a greater number of potential inputs.

## An Example of Machine Learning

Ronald Fisher was a biologist who worked during the first part of the 20th century. However, he's best known as the father of modern statistics. Fisher was trying to understand genetics and the general behaviors of populations of biological species, and the tool that he employed—and then helped to extend—was that of statistics.

One of Fisher's initial data sets was iris flowers. As with many flowers, irises come in a variety of species. Fisher

calculated the lengths and widths of two parts of the iris flower (petals and sepals) and found that using those measurements, he could determine which species an iris belonged to with a fair degree of certainty. In 1936, he took these four measurements from 150 different flowers, 50 from each of three species (setosa, virginica and versicolor). He then categorized these flowers and showed that based on this statistical model, you could take a new, previously unseen iris and correctly categorize it by measuring the dimensions of its petals and sepals.

This data set is a classic in the sense that it's old. But it's also a classic, and standard, way to introduce people to the world of machine learning. That's because the data set is small, easily understood and demonstrates many aspects of machine learning.

As you might have guessed, this is a case of supervised learning: create a model, and feed it not only the data, but also how to classify that data.

Now, let's use NumPy and scikit-learn to experiment with machine learning and see what you can do with this data set. I'm going to assume that you already have installed Python on your computer and that it is a recent enough version to include the `pip` command-line utility. Execute this command on the command line to install (or upgrade) NumPy, SciPy and scikit-learn:

```
$ sudo pip install -U numpy scipy scikit-learn
```

Once those have been installed, you can use them. First, import NumPy using the `np` alias that is universally used

within the NumPy world:

```
import numpy as np
```

scikit-learn comes with a number of sample data sets, including the "iris" data set described above. You'll import the function that can be used to load the data set:

```
from sklearn.datasets import load_iris
```

You now can run the function and get the data set:

```
iris = load_iris()
```

If you look at the `iris` variable that you have defined, you'll see that it's an object of type "Bunch", meaning that it collects everything you'll need in order to work with the data. The `iris` variable, once defined, has several attributes you might want to explore: `DESCR` (a textual description of the data set), `target_names` (the categories into which you can sort flowers), `data` (a NumPy array containing the measurements for all 150 flowers) and `feature_names` (the names of the features stored in the `data` attribute).

Combined, this gives you enough information to create a model and start training it.

In order to create a model with scikit-learn, you're going to need to have the features (that is, input data) and response (output data for training purposes) in separate NumPy arrays. Because each piece of input data contains several features—in this case, several flower measurements—and yet is only in

a single category, it makes sense that the response will be a one-dimensional NumPy array, whose length is the same as the number of observations.

scikit-learn expects you to put your input data (that is, features) in a variable called X (yes, uppercase X), a matrix (that is, multi-dimensional NumPy array). The target (that is, the classification) will be in a variable called y, a vector.

You can create these from the `iris` data as follows:

```
X = iris.data
y = iris.target
```

You can use the `shape` attribute to double-check that they both have the same width, but different lengths:

```
>>> y.shape
    (150,)

>>> X.shape
    (150, 4)
```

Now you can start to train your model! But wait…what model are you going to use? Each model implements an algorithm—often a statistical function—along with one or more parameter values. Choosing the right model can make a huge difference in the types of correlations you'll find.

A common model is KNN, or "K nearest neighbors". KNN assumes that there's a way to determine the "distance" between two of your inputs. For example, if you have a flower with measurements [1,2,3,4] and another flower

with measurements [1,2,5,6], you would have to decide how "close" these are to one another. In the case of KNN, the default is to use the Euclidean distance between them.

Given a distance calculation, here's how KNN works:

- You pick a value for K.
- You then search for the K closest inputs (observations) to the one you want to categorize.
- You use the most popular response to assign it to K.

In other words, you're basically saying that if K=3, and a new input is close to two blues and one red, you'll say that the new observation should be categorized as blue.

Now, it's possible for KNN to come up with tie votes. It's also possible to play with the value of K. That's the difference between an algorithm and a model; the KNN algorithm can lead to many different models, each with its own distance calculation and value of K.

In scikit-learn, you always approach a model in the same way, with the same four steps. This is a big help when you want to approach new types of models; the scikit-learn API is highly standardized and makes it possible to concentrate on the models rather than how you work with them. The steps are:

1. Import the class representing your model.
2. Create an instance of this class.

3. Train the model with your data.

4. Predict the response for a new observation.

Let's go through all four of those steps now, given your "iris" input data and the KNN model.

*1) Import the class:* scikit-learn comes with a very large number of model classes. The KNN model is in sklearn.neighbors.KNeighborsClassifier. As you learn more about machine learning, you definitely should explore the other models it offers:

```
from sklearn.neighbors import KNeighborsClassifier
```

*2) Create an instance of this class:* this is just like creating an instance of any other Python class. Any arguments you pass to the class when creating an instance affect the model. In the case of KNN, you can pass any value you want. Let's try 3:

```
knn = KNeighborsClassifier(n_neighbors=3)
```

*3) Train the model with your data:* take the X and y values, set above, and train the model with the "fit" method:

```
knn.fit(X, y)
```

Notice that you don't get a value back. That's because the model is learning; you're feeding it information that it can use in the future.

*4) Predict the response for a new observation:* the model has learned! Now you can ask the model that if you have a flower with certain measurements, into which category should it go? For this, you use the "predict" method, passing it a 1x4 NumPy array containing the same four features you used in all of your previous observations:

```
knn.predict([[3,5,4,2]])
```

You get back the following response:

```
array([1])
```

In other words, the model believes the most appropriate response would be category 1. Or, if you feed that back into `iris.target_names`:

```
iris.target_names[knn.predict([[3,5,4,2]])]
```

you get back:

```
array(['versicolor'],
    dtype='|S10')
```

So, the new flower would most likely be of type "versicolor".

## Validating

Now is when things start to become truly interesting—and complex.

You've trained the model, and it gives out some

You've trained the model, and it gives out some responses, but are those responses true? Are they reasonable? Would you be better off using a different model? The same algorithm with different parameters?

responses, but are those responses true? Are they reasonable? Would you be better off using a different model? The same algorithm with different parameters?

All of those are possible, and it's up to you to figure it out. But how?

Moreover, there's a big flaw in the example above: it trained a model and then decided that it was good enough. But maybe it wasn't. How can you feed data back into the model to see?

The last question is a fairly easy one to address. If you have new data for which you know the classification, then you can test your model against those. But, what if you don't? You can divide your data into parts—and use one part to train the model and the other part to test it.

This is done using a technique known as train-split-test. And sure enough, scikit-learn provides a way of doing this:

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
 ➥test_size=0.4)
```

Now you have two values for X (one for training and one for testing) and two values of Y (one for training and one for testing). By setting a test size of 0.4, you're saying you want the test to contain a random 40% of your values.

It's not uncommon to cross-validate across many different parts of your input data to ensure that each part of the input is tested against each part of the output.

## Comparing Models

No matter what though, you'll want to find out whether your model does a better job than other possibilities. Indeed, you'll likely want to try several different models and compare them against one another.

The sklearn.metrics module contains a large number of methods that can help you compare the results of running models. For example, after training several different models with the same input data, give those different models a bunch of new observations for which you *know* the answer. Get the predicted outcomes, and then compare those predictions against the known answers.

For example, you could create a KNN model with K=3 and another KNN model with K=5. You then can run the `predict` method on each of them:

```
knn3_predictions = knn3.predict(X)
knn5_predictions = knn5.predict(X)
```

Since you know the expected outcome for each of these, you then can compare how the predictions went with the expected outcome, using the `accuracy_score` method

within scikit-learn.metrics:

```
from sklearn import metrics
print(metrics.accuracy_score(actual_y, knn3_predictions))
print(metrics.accuracy_score(actual_y, knn5_predictions))
```

This accuracy can be done even if you use models that aren't using the same algorithm. For example, perhaps logistic regression might provide you with a better fit than KNN; you easily can fire that up, train it with the same inputs and see if it does a better (or worse) job than the KNN numbers 3 and 5.

By evaluating a number of possible model candidates, using various algorithms and parameters, you can find something that truly describes your input data and allows you to make predictions about new data. You could argue that by following this process, there is certainly some machine learning going on—but above and beyond that, there's human learning taking place as well, with people getting a better sense of what to measure and how to measure it.

## Conclusion

Machine learning is a large, complex and deep topic that is receiving a growing degree of attention—and deservedly so! Python's scikit-learn library makes it surprisingly easy to work with input data and explore that data using different types of models.

There is a growing business case for the use of machine learning to identify trends among customers, staff and

operations. By applying machine learning models to your input data, you can gain insights into what is (and isn't) working in your business and where to improve—finding correlations and similarities among data points that you might not have expected.

I believe that machine learning has the potential to influence our lives profoundly. The fact that with some basic statistics and programming knowledge, you can apply it to your own work, makes it all the more worthwhile to start to do so.■

# Resources

Many resources are available to help you learn about data science in general and machine learning in particular.

The documentation on the scikit-learn site (scikit-learn.org) is a good place to start, with some tutorials and lots of clearly written documentation.

If you want a higher-level, broader perspective, *Building Machine Learning Systems with Python*, written by Willi Richert and Luis Pedro Coelho and published by Packt Press, has a lot of good information, including hands-on tutorials and descriptions of the different types of models you can (and should) work with.

Another good resource is the site Machine Learning Mastery (machinelearningmastery.com) by Jason Brownlee. His ebooks are relatively inexpensive and are packed with useful information about machine learning. But, even his free materials are of high quality and can help you understand when and how to use machine learning.