Splitting Data

Bookmark this page

Supervised and Unsupervised Learning

So far, you've only covered unsupervised machine learning algorithms that run on unlabeled data. Even in the few example cases where we labeled the data for visualization purposes, those labels were removed before feeding the data into PCA, Isomap, and K-Means, and only added back in at the end when the data was graphed. You've taken a look at two types of dimensionality reduction methods that aim to extract a simplified version of the nature of your dataset. And you've also experimented with a clustering algorithm that ranks similarity and attempts to minimize within-cluster differences between the centroid and samples. This is a good time to document these algorithms, along with their use cases in your **course map**. You should take note of the strengths of weaknesses of the algorithm, as well as a sample use cases for when you should use each one.

Every single machine learning class, or *estimator* as SciKit-Learn call them, implements the .fit() method as you've seen. This will continue to hold true for the supervised one's as well. The unsupervised estimators also allowed you to make use of the following methods:

- `.transform()` : without changing the number of samples, alters the value of each existing feature by changing its units
- `.predict()` : only with clustering, you could predict the label of the specified sample

For the rest of this course, you're going to focus on supervised learning algorithms. The main difference between supervised and unsupervised learning is that with supervised learning, you actually guide the machine's *hand* at choosing the right answers. By showing the computer examples of what you want it to do, instead of just asking it to tell you something interesting about your data, the computer's responsibility shifts to deriving a set of rules that when applied to raw data, has a decent chance of choosing the answers you've trained it to. What you'll now see is that supervised learning estimators implement a slightly different set of distinct methods:

- `.predict()` : After training your machine learning model, you can predict the labels of new and never seen samples

- `.predict_proba()` : For some estimators, you can further see what the probability of the new sample belonging to each label is
- `.score()`: The ability to score how well your model fit the training data

Overfitting & Scoring Results

While training your machine with supervised learning, it's important to track how well its performing. Consider the spam email identification example we presented in The Possibilities section from The Big Picture module: Imagine you had 100 spam emails and 100 regular emails and you trained an estimator to differentiate between the two. And it did so *perfectly*. What comes next? With such an exact estimator that 100% reliably labels emails, you'd probably want to integrate it into your email client. Do you see a problem here?

Every email that you quizzed your estimator, it's actually seen before during its training! You were telling it, this email is spam, this email isn't spam, etc. What you've essentially done is create a basic model that simply regurgitates the label *you've already given* for any specific email. That's not machine learning. In fact, that's no more remarkable than opening up a text file, saving something inside of it, and then being amazed when the *exact* same text is found within the text file upon reopening. This is called **overfitting**. Your goal with machine learning is to create a generalizable algorithm that can be applied to data it *hasn't seen* yet and still do the *task* you've trained it to do. In our case, properly classify the spam status of an email.

To make this possible, of course you'll still need to fit your estimator; but when it comes to *testing* it, that part will have to be done with data the estimator has never seen before. In other words, all of your training **transformation and modeling** needs to done using just your training data, without ever seeing your testing data. This will be your way of validating the true accuracy of your model. This is doable by splitting your training data into two portions. One part will actually be used for the training as usual, but the other part of the data is retained and used during testing only. How much data should you hold back? If you hold back too much data then your algorithm's performance is going to suffer, since you didn't train it well. If you don't hold back enough you won't have a statistically sufficient number of 'quiz' questions to gauge your machine learning model with.

English ▼

POWERED BY OPENedX

Download on the App Store

ANDROID APP ON Google play