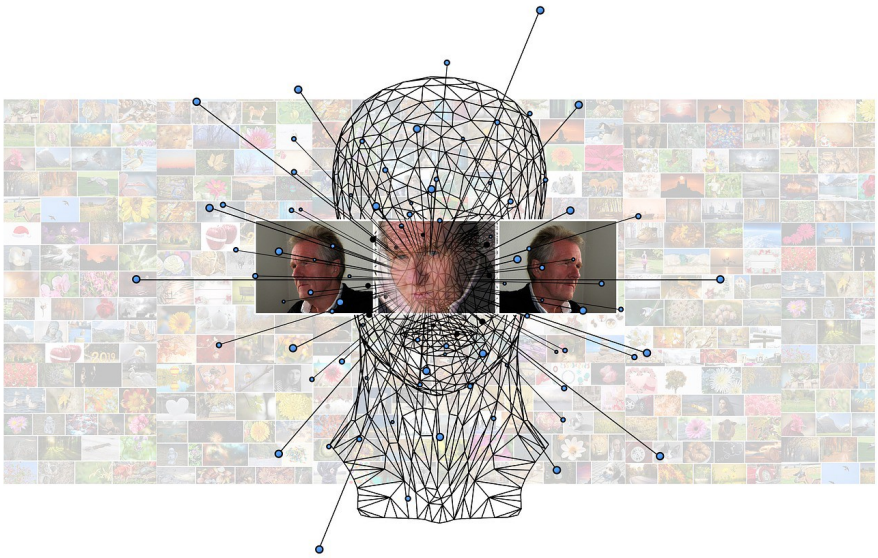
# When to use different machine learning algorithms: a simple guide

Roger Huang Follow
Feb 7 · 5 min read



If you've been at machine learning long enough, you know that there is a "no free lunch" principle—there's no one-size-fits-all algorithm that will help you solve every problem and tackle every dataset.

I work for Springboard—we've put a lot of research into machine learning training and resources. At Springboard, we offer the first online course with a machine learning job guarantee.

What helps a lot when confronted with a new problem is to have a primer for what algorithm might be the best fit for certain situations. Here, we talk about different problems and data types and discuss what might be the most effective algorithm to try for each one, along with a resource that can help you implement that particular model.

Remember: the proof is in the pudding: the best approach to your data is the model that empirically gives you the best results. This guide is meant to hone your first instincts and help you remember what models might be the most effective for each problem, and which would be impractical to use.

Let's start by talking about the variables we need to consider.

### Unsupervised learning vs supervised learning

**Unsupervised learning** is where you allow the machine learning algorithm to start learning and outputting a result without any explicit human processing of the data beforehand.

**Supervised learning** involves some labeling and processing of the training data beforehand in order to structure it for processing.

The kind of learning you can perform will matter a lot when you start working with different machine learning algorithms.

### Space and time considerations

There are **space and time considerations** for each machine learning algorithm. While in practice you'll likely work with optimized versions of each algorithm packaged in a framework, it is good to consider how the algorithms you choose can affect performance.

### The output

Third, and perhaps most important, is **the output that you want to get**. Are you trying to categorize data? Use it to predict future data points? What you're looking to get as a result and what you want to do to your data will largely determine the algorithmic approaches you should take.

## Some examples

### You're looking to build a simple predictive model with a well-structured dataset without too many complications.

Your best bet here is probably linear regression, something that can take a whole host of factors and then give you a predictive result with a simple error rate explanation and a simple explanation for which factors contribute to the prediction. It doesn't take much computational power to run a linear regression either.

**Resource**: Linear Regression—Detailed View

### You're looking to classify data that's already been labeled into two or more sharply distinct types of labels (e.g., trying to determine if children are likely

### male or female based on their weight and height) in a supervised setting.

The first instinct you should have when you see a situation like this is to apply the **logistic regression model**. After running the model, you'll see that it forces every data point into two different categories, allowing you to easily output which point belongs to which category. The logistic regression model can also be easily generalized to working with multiple target and result classes if that's what your problem demands.

**Resource**: Building a Logistic Regression

### You're looking to place unlabeled continuous data into different groups (e.g., putting customers with certain recorded traits and trying to discover categories/groups they can belong to).

The first natural fit for this problem is the K-Means clustering algorithm, which will group and cluster data by measuring the distance between each point. Then there are a variety of clustering algorithms, such as Density-Based Spatial Clustering of Applications with Noise and Mean-Shift algorithms.

**Resource**: The 5 Clustering Algorithms Data Scientists Need to Know

### You're looking to predict whether a string of characters or a grouping of traits falls into one category of data or another (supervised text classification)—e.g, whether a review is positive or negative.

Your best bet here is probably Naive Bayes, which is a simple but powerful model that can be used for text classification. With some text pre-processing and cleaning (being especially careful to remove filler stop words such as "and" that might add noise to your dataset), you can get a remarkable set of results with a very simple model.

Another decent bet is logistic regression, which is a simple model to grasp and explain, and less hard to pick apart than Naive Bayes (which will often assign probabilities word by word rather than holistically labeling a text snippet as being part of one group or another).

Moving on to something more powerful, a Linear Support Vector Machine algorithm will likely help improve your performance. If you

want to skip right ahead here, you can (though I suggest trying both models and comparing which one works best—Naive Bayes has an absurdly easy implementation on frameworks like scikit-learn and it isn't very computationally expensive so you can afford to test both).

Lastly, bag-of-words analysis could also work—consider doing an ensemble of different methods and testing all of these methods against one another, depending on the dataset in question.

**Resource**: Multi-Class Text Classification Model Comparison and Selection

## You're looking to do unstructured learning on large-scale image or video datasets (e.g., image classification).

The best algorithm to tackle going through different images is a convolutional neural network that is organized similarly to how animal visual cortexes are analyzed.

Measured by performance (reduced error rate) in the ImageNet competition, the SE-Resnet architecture comes out on top, though as the field is still developing, new advances come out almost every day.

You should be aware, however, that convolutional neural networks are dense and require a lot of computational power—so make sure that you have the hardware capability to run these models on large-scale datasets.

**Resource**: Review of Deep Learning Algorithms for Image Classification

## You're looking to classify result points that come out of a well-defined process (ex: number of hires from a pre-established interview process, wherein you know or can computationally infer the probabilities of each event).

The best option for this is probably a decision tree algorithm that will clearly explain what the split points are between classifying something into one group or another.

**Resource**: Decision Trees in Machine Learning

### You're looking to do time series analysis with well-defined, supervised data (e.g., predicting stock prices based on historical patterns in the stock market arranged on a chronological basis from the past to the present).

A recurrent neural network is set up to do sequence analysis by containing an in-stream internal memory of data it processes, allowing it to take into account the relationship between data and the time horizon and order it's deployed in.

**Resource**: Recurrent Neural Networks and LSTM

## Wrapping up

Take the recommendations and resources above, and apply them as a sort of first instinct for your modeling—it'll help you jump into any work you do just a little bit faster. If you're interested in being mentored by a machine learning expert in learning how to train your instincts further, check out Springboard's AI/Machine Learning Career Track.