Machine Learning

Machine Learning

Machine Learning is the science (and art) of programming computers so they can learn from data.

Types of Machine Learning Systems

There are so many different types of Machine Learning systems that it is useful to classify them in broad categories, based on the following criteria:

- Whether or not they are trained with human supervision (supervised, unsupervised, semi supervised, and Reinforcement Learning)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model, much like scientists do (instance-based versus model-based learning)

Supervised/Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of **supervision they get during training.** There are four major categories: supervised learning, unsupervised learning, semi supervised learning, and Reinforcement Learning.

Supervised learning

In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels.

- **Classification** A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.
- **Regression** Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.) called 1 predictors. This sort of task is called regression. To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).

Note that some regression algorithms can be used for classification as well, and vice versa.

Here are some of the most important supervised learning algorithms

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests

Unsupervised learning

In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.

- Clustering
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis (HCA)
- Anomaly detection and novelty detection
  - One-class
  - SVM Isolation Forest
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally Linear Embedding (LLE)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
  - Apriori
  - Eclat

Clustering

It tries to detect groups of similar visitors. At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help. For example, it might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends. If you use a hierarchical clustering algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.

Visualization and dimensionality reduction

Visualization algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted. These algorithms try to preserve as much structure as they can (e.g., trying to keep separate clusters in the input space from overlapping in the visualization) so that you can understand how the data is organized and perhaps identify unsuspected patterns.

Dimensionality reduction is to simplify the data without losing too much information. One way to do this is to merge several correlated features into one. For example, a car's mileage may be strongly correlated with its age, so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear. This is called feature extraction.

It is often a good idea to try to reduce the dimension of your training data using a dimensionality reduction algorithm before you feed it to another Machine Learning algorithm (such as a supervised learning algorithm). It will run much faster, the data will take up less disk and memory space, and in some cases it may also perform better.

Anomaly detection and novelty detection

Anomaly Detection For example detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is shown mostly normal instances during training, so it learns to recognize them; then, when it sees a new instance, it can tell whether it looks like a normal one or whether it is likely an anomaly

Novelty detection it aims to detect new instances that look different from all instances in the training set. This requires having a very "clean" training set, devoid of any instance that you would like the algorithm to detect. For example, if you have thousands of pictures of dogs, and 1% of these pictures represent Chihuahuas, then a novelty detection algorithm should not treat new pictures of Chihuahuas as novelties. On the other hand, anomaly detection algorithms may consider these dogs as so rare and so different from other dogs that they would likely classify them as anomalies (no offense to Chihuahuas).

Association rule learning

Association rule learning, in which the goal is to dig into large amounts of data and discover interesting relations between attributes. For example, suppose you own a supermarket. Running an association rule on your sales logs may reveal that people who purchase barbecue sauce and potato chips also tend to buy steak. Thus, you may want to place these items close to one another.

Semi supervised learning

Since labeling data is usually time-consuming and costly, you will often have plenty of unlabeled instances, and few labeled instances. Some algorithms can deal with data that's partially labeled. This is called semisupervised learning.

Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the unsupervised part of the algorithm (clustering). Now all the system needs are for you to tell it who these people are. Just add one label per person and it is able to name everyone in every photo, which is useful for searching photos.

Reinforcement Learning

Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards). It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

Batch and Online Learning

Another criterion used to classify Machine Learning systems is whether or not the system can learn **incrementally from a stream of incoming data.**

Batch learning

In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.

If you want a batch learning system to know about new data (such as a new type of spam), you need to train a new version of the system from scratch on the full dataset (not just the new data, but also the old data), then stop the old system and replace it with the new one.

Online learning

In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or in small groups called mini batches. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.

One important parameter of online learning systems is how fast they should adapt to changing data: this is called the **learning rate**. If you set a high learning rate, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data (you don't want a spam filter to flag only the latest kinds of spam it was shown). Conversely, if you set a low learning rate, the system will have more inertia; that is, it will learn more slowly, but it will also be less sensitive to noise in the new data or to sequences of nonrepresentative data points (outliers).

Instance-Based Versus Model-Based Learning

One more way to categorize Machine Learning systems is by how they generalize. Most Machine Learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to make good predictions for (generalize to) examples it has never seen before. Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.

Instance-based learning

The system learns the examples by heart, then generalizes to new cases by using a similarity measure to compare them to the learned examples (or a subset of them).

Possibly the most trivial form of learning is simply to learn by heart. If you were to create a spam filter this way, it would just flag all emails that are identical to emails that have already been flagged by users—not the worst solution, but certainly not the best. Instead of just flagging emails that are identical to known spam emails, your spam filter could be programmed to also flag emails that are very similar to known spam emails. This requires a measure of similarity between two emails. A (very basic) similarity measure between two emails could be to count the number of words they have in common. The system would flag an email as spam if it has many words in common with a known spam email.

Model-based learning

Another way to generalize from a set of examples is to build a model of these examples and then use that model to make predictions. This is called model-based learning.

Main Challenges of Machine Learning

In short, since your main task is to select a learning algorithm and train it on some data, the two things that can go wrong are "bad algorithm" and "bad data."

Bad Data

- Insufficient Quantity of Training Data
- Nonrepresentative Training Data
- Poor-Quality Data
- Irrelevant Features

Bad Algorithm

- Overfitting the Training Data
- Underfitting the Training Data

Overfitting the Training Data

Machines can over generalizing is something if we are not careful. In Machine Learning this is called overfitting: it means that the model performs well on the training data, but it does not generalize well.

Overfitting happens when the model is too complex relative to the amount and noisiness of the training data. Here are possible solutions:

- Gather more training data.
- Simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data, or by constraining the model.
- Reduce the noise in the training data (e.g., fix data errors and remove outliers)

Regularisation Constraining a model to make it simpler and reduce the risk of overfitting is called regularization.

Hyperparameter The amount of regularization to apply during learning can be controlled by a hyperparameter. A hyperparameter is a parameter of a learning algorithm (not of the model). As such, it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training. If you set the regularization hyperparameter to a very large value, you will get an almost flat model (a slope close to zero); the learning algorithm will almost certainly not overfit the training data, but it will be less likely to find a good solution. Tuning hyperparameters is an important part of building a Machine Learning system.

Underfitting the Training Data

As you might guess, underfitting is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data. For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be **inaccurate, even on the training examples**.

Here are the main options for fixing this problem:

- Select a more powerful model, with more parameters.
- Feed better features to the learning algorithm (feature engineering).
- Reduce the constraints on the model (e.g., reduce the regularization hyperparameter).

Sampling Bias

It is crucial to use a training set that is representative of the cases you want to generalize to. This is often harder than it sounds: if the sample is too small, you will have *sampling noise* (i.e., nonrepresentative data as a result of chance), but even very large samples can be nonrepresentative if the sampling method is flawed. This is called *sampling bias*.

Feature Engineering

A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called feature engineering, involves the following steps:

- **Feature selection** selecting the most useful features to train on among existing features
- **Feature extraction** combining existing features to produce a more useful one —as we saw earlier, dimensionality reduction algorithms can help
- Creating new features by gathering new data

Testing and Validating

The only way to know how well a model will generalize to new cases is to actually try it out on new cases. One way to do that is to put your model in production and monitor how well it performs. This works well, but if your model is horribly bad, your users will complain—not the best idea.

A better option is to split your data into two sets: the training set and the test set. As these names imply, you train your model using the training set, and you test it using the test set. The error rate on new cases is called the generalization error (or out-of-sample error), and by evaluating your model on the test set, you get an estimate of this error. This value tells you how well your model will perform on instances it has never seen before.

If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

It is common to use 80% of the data for training and hold out 20% for testing. However, this depends on the size of the dataset: if it contains 10 million instances, then holding out 1% means your test set will contain 100,000 instances, probably more than enough to get a good estimate of the generalization error.

Cross-validation

Each model is evaluated once per validation set after it is trained on the rest of the data. By averaging out all the evaluations of a model, you get a much more accurate measure of its performance. There is a drawback, however: the training time is multiplied by the number of validation sets.