Revisiting supervised and unsupervised learning (after experience with some models)

Herman Kamper

After experience with a few different supervised and unsupervised machine learning models, it is useful to revisit our views of supervised and unsupervised learning, as well as machine learning in general.

Machine learning

I started by stating that most machine learning methods are ultimately ways to set some numbers in a function that we don't want to set by hand. We normally learn these numbers from data.[1]

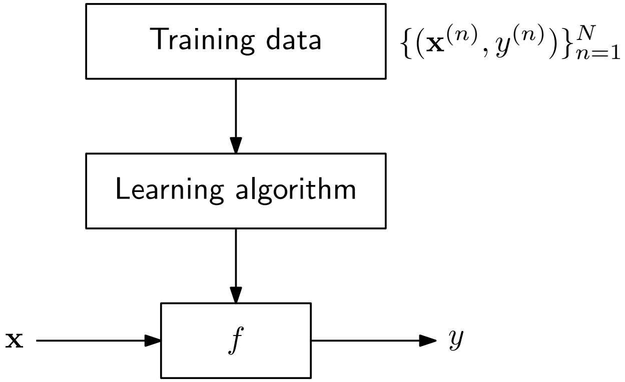
I sometimes think of this function as a function in a computer (i.e. declared with `def` in Python) and other times as a mathematical function with some numbers that we don't know—both views can be useful.

Formally, the function is normally called a *model* and the numbers are called the model's *parameters*.

[1] There are exceptions to using an "offline" dataset. When machine learning is used in control system applications, we might be updating parameters on the fly directly on data points as they are coming in. In reinforcement, we learn the parameters through interaction with the world.

Supervised learning

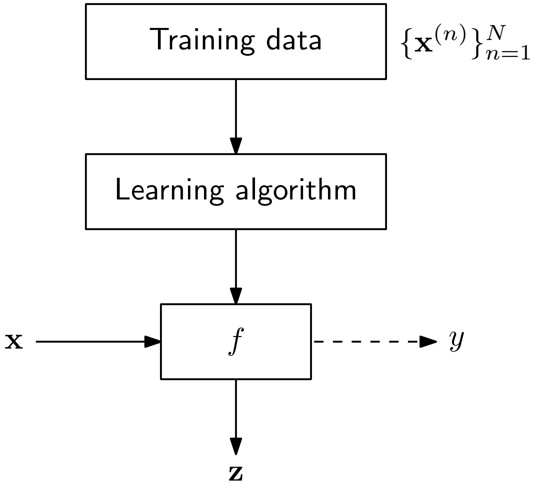
In supervised learning, we learn a model f from observed input features $\mathbf{x}$ with target outputs y.



Inside the model f, there are parameters that we need to fit from previously observed data (the training data). In general, we can denote these parameters together as $\boldsymbol{\theta}$. For instance in linear regression, $\boldsymbol{\theta}$ would be the weight vector $\mathbf{w}$. In a decision tree, $\boldsymbol{\theta}$ would specify the questions being asked at each node in the tree. After training, we can make predictions for inputs $\mathbf{x}$ that we haven't seen during training, $f(\mathbf{x}; \boldsymbol{\theta})$. If everything goes well, the model should generalise to these new inputs.

Unsupervised learning

In unsupervised learning, we learn a model f only from observed input features $\mathbf{x}$ without any corresponding labels.



Again the unsupervised model f has parameters $\boldsymbol{\theta}$ that we need to fit. For instance in PCA, the parameters $\boldsymbol{\theta}$ would be the projection vectors in $\mathbf{W}$. In K-means clustering, $\boldsymbol{\theta}$ would be the cluster centroids.

Sometimes we do have proxy output labels y that we use during training, but these are obtained directly from the data, which is why I indicate it with a dashed line in the figure. For instance in PCA, we can think of the target output as $\mathbf{x}$, i.e. the model is trying to reconstruct its own input.

At test time, we are normally interested in some latent (hidden) variables $\mathbf{z}$ that occur somewhere inside the model f. For instance in PCA, the $\mathbf{z}$'s are the projections of the inputs onto the lower-dimensional space. In K-means clustering, you can think of the $\mathbf{z}$'s as the cluster IDs to which feature vectors are assigned. In most cases, we then use these latent variables in a downstream task, e.g. to visualise data or to compress a dataset.

A machine learning model

What is a model?

A model is some simplified representation of something in the real world. For instance in classification, we might want to capture how the class label y relates to some some input feature vector $\mathbf{x}$: There is a true underlying process governing this mapping from $\mathbf{x}$ to y, but we don't know it. So we want will "model" it.[2]

To get a machine learning model, we need the following ingredients:

1. A model structure: How we go from inputs to outputs. Includes defining the model parameters, but values aren't yet assigned.
2. Data: The stuff we give to the learning algorithm.
3. A learning algorithm: The combination of a loss function and an optimisation procedure.
4. A model instance: Obtained by passing the data through the learning algorithm. Here we have specific values for the model parameters.

Let's unpack some of these.

Model structure

Other people might call this the "model class" or "model architecture" or even just "the model". How do you map the inputs to the outputs? In linear regression you take the dot product between the input $\mathbf{x}$ and a weight vector $\mathbf{w}$, giving the output $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$. In a decision tree, you iteratively ask questions about the input until you reach a leaf node, which then specifies the output. Sometimes you go through latent variables within the structure: In PCA, for instance,

[2] Formally, we are modelling the conditional expectation $\mathbb{E}[y|\mathbf{x}]$. Why is this what we are trying to do in regression settings?

the model produces a reconstruction of the input by going through the intermediate projection $\mathbf{z}$. Importantly, the model structure does not specify the actual values that the parameters take on: it just defines the form. So when for linear regression we write $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$, we don't specify the actual values that $\mathbf{w}$ takes on.

Data

In supervised learning we normally have $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$ while in unsupervised learning we normally have $\{\mathbf{x}^{(n)}\}_{n=1}^N$.

Learning algorithm

The learning algorithm takes the data and the model structure and outputs specific values for the parameters. So in linear regression, the learning algorithm takes the data and then gives us $\hat{\mathbf{w}}$, specific estimates of the weights. To do this, the learning algorithm requires two things:

1. A loss function. Examples include:

    - Squared loss
    - Mean absolute error
    - Negative log likelihood
    - Misclassification error
    - Sum of squared distances to centroids

2. An optimisation procedure. Examples include:

    - Closed-form solutions, e.g. in linear regression.
    - Gradient descent. There are actually several variants.
    - Greedy search, e.g. in decision trees.
    - Expectation maximisation. This is similar to the procedure in K-means clustering where you alternate between assigning items to clusters (the expected cluster given the current model) and updating the cluster centroids (maximising the objective given the current assignments).

In some cases it is possible to use different optimisation procedures for different loss functions. For instance, instead of using the closed-form solution in linear regression, you can optimise the weight vector $\mathbf{w}$ using gradient descent.[3] In a regression tree, you could decide to optimise the mean absolute difference instead of a squared loss by just changing a few lines of code. It is not always possible to swap out the components of the learning algorithm; for instance, closed-form optimisation is only possible for a handful of (simple) models.

Model instance

Other people might call this the "estimated model". The learning algorithm takes the model structure and data and outputs specific parameter values. The result is a function that we can actually use, with known parameter values. For instance in linear regression, we can now apply $f(\mathbf{x}; \hat{\mathbf{w}}) = \hat{\mathbf{w}}^\top \mathbf{x}$ to unseen data, because we have the specific parameter numbers $\hat{\mathbf{w}}$.

[3] This won't necessarily give the lowest training error, but could regularise the model.