Talking About Learning

What Do We Mean When We Talk About Learning?

João Tiago Ascensão, Data Science @ Feedzai (soon) & LDSA, Ex-Farfetch, Ex-Uniplaces

Table of contents

Problem Statement

$D = \{(x_i, y_i)\}_{i=1}^{m}$ is the *data*, a collection of labeled observations (or *instances*).

$x_i = (x_i^{(1)}, \ldots, x_i^{(n)}) \in \mathbb{R}^m$ is a *representation*, as a vector of *features*. x_i^j is the *measurement* of the j-th feature for i-th observation i. Instances $x_i \sim \mathcal{D}$.

The *target* y_i results from a constant, unknown function $y_i = c(x_i), c \in C$.

We aim to infer c, the function that maps inputs x_i to outputs y_i, from the data:

$$\hat{y}_i = h(x_i) \approx c(x_i), h \in H, H \subseteq C$$

Problem Statement

$X \in \mathbb{R}^{m \times n}$ contains the observations in our dataset.

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \dots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \dots & x_2^{(n)} \\ \dots & \dots & \dots & \dots & \dots \\ x_m^{(1)} & x_m^{(2)} & x_m^{(3)} & \dots & x_m^{(n)} \end{bmatrix}$$

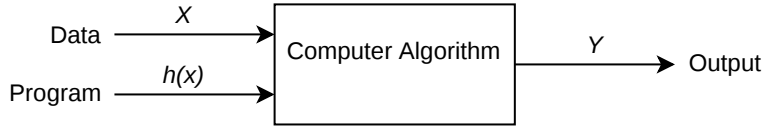
Problem Statement

$Y \in \mathbb{R}^{m \times 1}$ contains the observed values for the target, for each observation.

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{bmatrix}$$

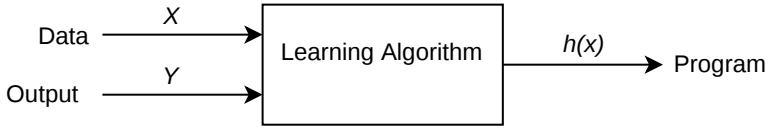
Traditional Programming

The algorithm applies the *hard-coded* hypothesis, as programmed



Machine Learning

The algorithm *learns* the hypothesis, as if programming itself.



Components of a Learning Algorithm

A learning algoritms aims to select the best available hypothesis $h \in H$.

The final hypothesis, more precisely, $h_\Theta(x)$, is a combination of:

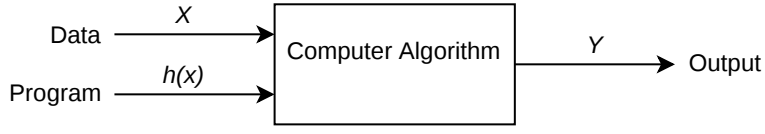
- A general hypothesis (i.e., H), embedded in the learning algorithm
- A finite set of parameter values, Θ, that adapt the general hypothesis to the data.

Take linear regression: $h_\Theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$.

It follows that if $H \subseteq C$, **there's no guarantee that $c \in H$**. *No magic applies.*

Traditional Programming

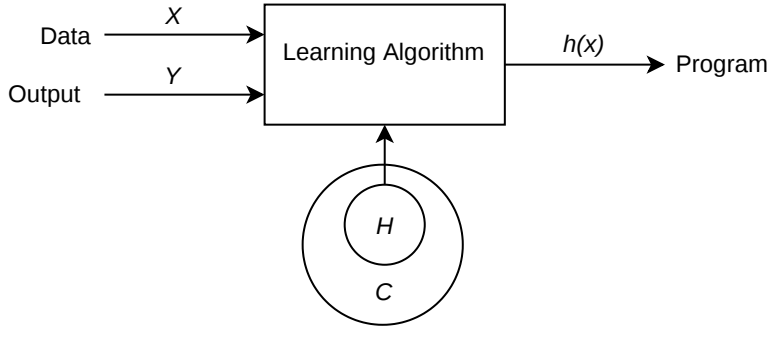
The programmer explores the hypothesis space.



Machine Learning

The algorithm explores a hypothesis space, pre-defined by the programmer.

The hypothesis is tailored to the data using *parameters*.



Components of a Learning Algorithm

An hypothesis h is said to be *consistent* with the data if it *fits* the data.

The **loss function**, or error, is the a penalty for failing to predict a single example i.

An example is the squared error loss: $L(h_\Theta(x_i), y_i) = (h_\Theta(x) - y_i)^2$.

In theory, we can evaluate all possibilities $h_\Theta \in H$ against any $(x_i, y_i) \in D$.

Components of a Learning Algorithm

The **cost function** computes the error over the entire data set.

Take the average loss, $J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} L(h_\Theta(x_i), y_i)$, for example.

Thus, we can define an optimization criterion for a learning algorithm: $\min_\Theta J(\Theta)$.

An hypothesis is perfectly consistent if $J(\Theta) = 0$. (Not necessarily a good thing.)

Components of a Learning Algorithm

Algorithms that can approximate any target concept are *universal approximators*.

This is different from *completeness*, i.e., searching the entire space of C, or H.

(Hence, optimization. Guessing and/or random search won't take us far.)

Closed-form is preferable to complex optimization, but often not possible.

Components of a Learning Algorithm

Gradient descent minimizes an objective function $J(\Theta)$, parameterized by Θ.

It works by updating the parameters in the opposite direction of the gradient of the objective function $\Delta_\theta J(\Theta)$, with respect to the parameters.

The vanilla gradient descent (i.e., batch) runs in *epochs*, each epoch using *all data* to update each parameter.

Components of a Learning Algorithm

The learning rate, α, determine the size of the steps we take to reach a minimum:

$$\theta_{t+1} = \theta_t - \alpha \Delta_\theta J(\Theta)$$

The $\Delta_\theta J(\Theta)$ concerns to the partial derivative of $J(\Theta)$, with respect to θ.

Intuitively, we follow *downhill* on the surface of the objective function.

Generalization

Given all we've seen, having a learning algorithm and some data doesn't ensure success:

- The concept $c \in C$ must be approximable by our learning algorithm
- If we use the data set D to fine-tune our general hypothesis, the sample must be representative and contain good, non-sparse, consistently measured features.

And although both conditions are necessary, they are not sufficient for *generalization*.

Generalization

Generalization is the ability of the trained hypothesis $h_\Theta(x)$ to make good predictions on unseen data, assuming that incoming instances $x \sim \mathcal{D}$.

It requires balancing in-sample and out-of-sample consistency, in what is known as the bias-variance trade-off.

Generalization

Simplistic algorithms with limited sets of hypothesis H can't quite stretch and adapt to underlying patterns in the data, also known as *under-fitting*.

Highly-flexible learning algorithms are great at consistency but prone to *overfitting*, i.e., reacting to noise and corner-cases in the data with increasing complexity.

Generalization

In both cases, the learning algorithm will select a bad hypothesis: wildly inconsistent or consistently wrong.

Therefore, **the trade-off between bias (i.e., stupid) and variance (i.e., overthinking)**.

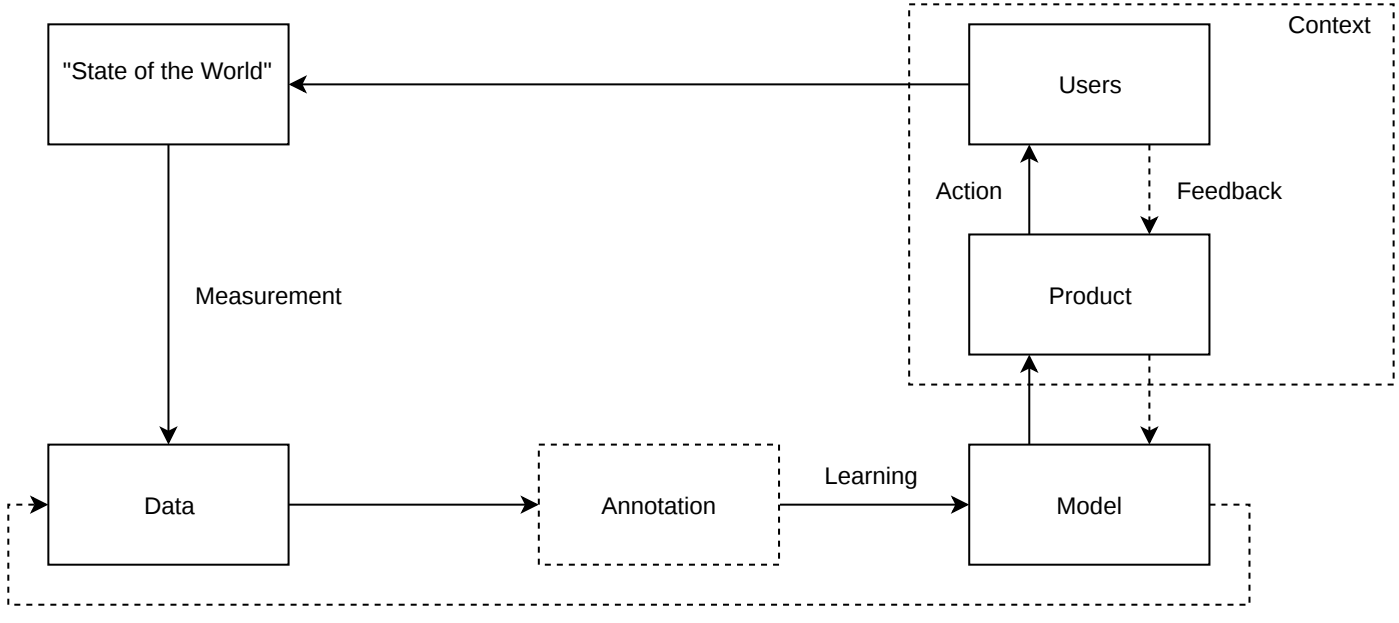
No such thing as a free-lunch

The no free lunch theorem proves that all learning algorithms are equivalent, when their performance is averaged across all possible problems.

So, on average, random search would be as good as fancy optimization.

It follows that some algorithms will perform better than other on specific problems, due to encoded prior knowledge, fit to the problem space at hand.

Real Life


"State of the World"
Users
Context
Action
Feedback
Measurement
Product
Data
Annotation
Learning
Model