Machine Learning Algorithms from a Statistical Viewpoint

Germán Hernández, Fabio Gonzalez and Juan Carlos Galeano
Department of Computer Systems Engineering
National University of Colombia-Bogotá

Abstract
A unifying statistical framework for some automated (machine) learning algorithms is presented.

1 Introduction

In recent years we have witnessed the application of a variety of automated (machine) learning algorithms, particularly processing data sets whose sizes and complexities are beyond the ability of humans to handle. Such algorithms have appeared in diverse fields such as: computer vision (face and character recognition), text and information retrieval (web mining e.g. google, spam detection), data mining (discovering patterns in large data bases), bioinformatics (DNA and protein sequence analysis, genomics), speech recognition, time series prediction (weather, finance), game playing (reinforcement, web agents), etc. This collection of data sets is growing (e.g. the genome project) and the data sets are generally available online.

The focus of this paper is to give a unifying statistical framework for some automated (machine) learning algorithms. These include disparate algorithms such as regression, neural networks and decision trees. Some other learning algorithms such as principal components analysis (PAC), Bayesian networks, hidden Markov models are not included in this overview.

Section 2 describes what is considered as learning and learning algorithms. Section 3 reviews the function-fitting and classification learning tasks and algorithms. Section 4 presents a formal setting for these learning algorithms.
2 Learning

Here learning will be considered as the process of gaining experience, knowledge or skill through training or schooling, in order to perform a task. In general we have two types of learning: supervised learning when we have a teacher or critic that provides feedback to the learner about responses to the training examples, and unsupervised when we have only training examples but no feedback about the answers of the learner, see figure 1. Some examples of tasks where learning algorithms have been successful are autonomous driving, speech recognition, handwritten character recognition and systems which learn user’s interests (e.g. newsreaders).

Learning algorithms try to emulate the learning capabilities of humans; a computer program or algorithm is said to learn if it improves its performance at some task through experience [7].

2.1 Elements of a model of learning

There are some elements of any model of learning needs to specify, these are:

1. **Learner**: “Who” does the learning? In our case, we will consider a computer program, and it is often required that an algorithm performs a task efficiently (in polynomial time), and using a finite amount of memory.

2. **Domain**: “What” is being learned? In automated learning we have:
   
   – **Concept learning (Negative-positive classification)**: when the learner is trying to learn an unknown concept (such as what is a chair and what is not a chair). In concept learning, the learner is trying to come up with a “rule” to separate positive examples from negative examples, i.e., the learner’s goal is to infer how an unknown target function
classifies examples as positive or negative from a given domain. As a concrete example consider the task of handwritten character recognition. A learning algorithm is given a set of examples where each contains a handwritten character as specified by a set of attributes (e.g., the height of the letter) along with the name (label) for the intended character. This set of examples is often called the training data. The goal of the learner is to efficiently construct a rule often referred to as hypothesis or classifier, that can be used to take some previously unseen character and with high accuracy determine the proper label. One possible task of the learner would be to classify each character as numeral or non-numeral. This can be generalized to multiple classification.

– **Function learning**: when the learner is trying to learn an unknown function (a classical example is learning a physical law by curve fitting to data[3]).

– **Technique/behavior learning**: when the learner is trying to learn an unknown technique (such as a how to play a game in order to win).

3. **Information Source**: “From what” is the learner learning? How is the learner informed about the domain? This can happen in many ways:

   a. **Examples**: the learner is given examples. These examples can be chosen in a variety of ways:
      
      • these can be chosen at random from some known or unknown distribution,
      • these can be chosen arbitrarily,
      • these can be chosen maliciously by some adversary who wants to know the worst these behavior of a learning algorithm,
      • these can be carefully chosen by a helpful teacher who wants to facilitate the learning process,
      • these can be generated for some random process (e.g., nature).

   b. **Queries**: the learner may get information about the domain by asking questions to a teacher.

   c. **Experimentation**: the learner may get information about the domain by actively experimenting with it.

4. **Information Representation**: How are features of the examples described?

5. **Prior Knowledge**: What does the learner know about the domain initially?

6. **Performance Criteria**: We have several possible criteria for evaluating a learning algorithm:

   a. **Accuracy**

   b. **Speed of learning**
In this lecture we will consider two tasks that are solved by supervised learning algorithms: function-fitting and classification.

3 Function-fitting and classification learning tasks

3.1 Function-fitting

Assume that we want to “learn” a function [9]

\[ f : \mathbb{R}^p \rightarrow \mathbb{R}^k \]

that underlies the relationship between some inputs and outputs and we have at hand a training set of observations or examples

\[ T = \{(x_i, y_i) \mid i = 1, 2, ..., n\}. \]

If the measurements generating this set were exact, then \( f(x_i) \) should be equal to \( y_i \), but in general, we expect this values to be affected by noise.

In some contexts the \( x_i \) rather than be chosen to do the training, are being generated by a probability distribution on \( \mathbb{R}^p \). Thus, one might take as starting point, instead of the unknown \( f \), a probability measure on \( \mathbb{R}^k \) for each \( x \in \mathbb{R}^p \), i.e., we have the conditional probability measures \( \mu(y|x) \) on \( \mathbb{R}^k \), for each \( x \in \mathbb{R}^p \).
\(\mathbb{R}^p\) and this case the \(y_i\) observed is a sample of the probability measure on \(\mathbb{R}^k\) for the given \(x_i\), (see figure 2). In many cases the starting point is only a single probability measure \(\mu\) on \(\mathbb{R}^p \times \mathbb{R}^k\) from which the pairs \((x_i, y_i)\) are randomly drawn. In section 4 we will relate this probability with the unknown function \(f\).

We have many learning algorithms to approximate the unknown function next we review two of them: regression and neural networks.

### 3.1.1 Regression

In regression we assume that the unknown function \(f\), has a specific form and that the space of all this functions can be parameterized by \(N\) real numbers, i.e., we have one candidate function \(f_\omega\) for every vector \(\omega \in \mathbb{R}^N\). Usually we look for the best fit estimate by the least squares method

\[
f^*(x) = \arg \min_{f_\omega} \sum_{i=1}^{n} (f_\omega(x_i) - y_i)^2
\]

In linear regression we consider that the functions of the form

\[
f(x) = x^T \beta
\]

with \(\beta\) the matrix of \(N = p \times k\) of parameters. In a more general context we can consider functions of the form

\[
f_\omega(x) = \sum_{r=1}^{R} \phi_r(x) \omega_k;
\]

where the functions \(\phi_r\) form a linear basis, by example polynomial or trigonometric expansions.

### 3.1.2 Multilayer Neural Networks

The use multilayer neural networks to approximate functions is equivalent to do a regression in which the function is seen as superposition of sigmoid functions \[2\]

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

this function has the nice property of \(\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))\) that allows to derive gradient decent rules to train.

An element that applies the sigmoid function \(\sigma\) to a linear combinations of its inputs is a called neuron, it is called this way because it has some analogy with the neurons of the nervous system. An array of at least three layers of this neuron is called a multilayer neural network.
\[ \sum_{i=0}^{n} w_i x_i = 0 \]

\[ o = \sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \]

Figure 3: Neuron [7]

Figure 4: Multilayer (3-layer) neural networks topology [7]
This neural network defines a function of the form
\[ f_\omega(x) = \sigma(\omega_{o1}h(x)), \sigma(\omega_{o2}h(x)), \ldots, \sigma(\omega_{ok}h(x)) \text{ and} \]
\[ h(x) = (\sigma(\omega_{h1}x), \sigma(\omega_{h2}x), \ldots, \sigma(\omega_{hl}x)) \]
where \( \omega_{oj} \) are the weights of the \( j \)th output neuron, \( \omega_{hj} \) are the weights of the \( j \)th hidden neuron and \( h \) are the outputs of the hidden neurons. In this case we have \( N = (p \times l) + (l \times k) \) parameters. There is a gradient decent method of training for this type of networks [4] and a proof [1][2] that with only three layers this networks can approximate, to any degree of precision, any measurable function.

### 3.2 Classification

Given some objects with \( k \) characteristics that can be codified as real numbers. Let's assume that there are \( p \) different classes of this objects \( \{C_1, C_2, \ldots, C_p\} \). We “learn” to classify this objects with a training set of classified examples
\[ T = \{(x_k, c_k) \mid i = 1, 2, \ldots, n\}. \]
This problem reduces to “learn” a function
\[ f : \mathbb{R}^p \rightarrow [0,1]^k \]
that assigns to an object in class \( C_i \), more exactly to its vector of characteristics, the vector \( e_i \). We have many learning algorithms to approximate this function, next we discuss two of them: decision trees and multilayer neural networks

#### 3.2.1 Decision Trees

Decision trees are considered when we have a classification problem that have:

- Instances describable by attribute–value
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possibly noisy training data

Some examples are equipment or medical diagnosis and credit risk analysis. In a decision tree we have that:

- Each internal node tests an attribute
- Each branch corresponds to attribute value
- Each leaf node assigns a classification

The top-down induction algorithm for decision trees is

Main loop:
1. A ← the “best decision attribute using information gain for next node

2. Assign A as decision attribute for node

3. For each value of A, create new descendant of node

4. Sort training examples to leaf nodes

5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

The information gain (entropy) measures how well a given attribute separates the training examples according to the target classification. Here are the classification function is of the form

\[
f(x) = \begin{cases} 
(1, 0, 0, 0, ..., 0) & \text{if } x^1 \in A_1^1, x^2 \in A_1^2, ..., x^p \in A_1^p \\
(0, 1, 0, 0, ..., 0) & \text{if } x^1 \in A_2^1, x^2 \in A_2^2, ..., x^p \in A_2^p \\
\vdots & \vdots \\
(0, 0, 0, ..., 0, 1) & \text{if } x^1 \in A_k^1, x^2 \in A_k^2, ..., x^p \in A_k^p 
\end{cases}
\]

where \(x = (x^1, x^2, x^3, ..., x^p)\) and \(A_{ij}\) is the set defined by the union of all the conditions on attribute \(j\) over branches of the tree that lead to class \(i\).

### 3.2.2 Multilayer Neural Networks Classifier

Here the application to approximate the classification function is the application of the multilayer neural network to approximate particular case of function. to illustrate it use se the example in figure 6.
Figure 6: Multilayer neural networks classifier of phonemes using the two first formants [7]

4 Formal Setting

Now we present a unifying formal setting for the algorithms presented based on [3]. Let us consider the following probability space

\[(\Omega = X \times Y, \beta_\Omega, \mu)\]

where

i) \(X\) is the space of inputs of our learning task, usually \(X = \mathbb{R}^p\)

ii) \(Y\) is the space of outputs of our learning task, usually \(Y = \mathbb{R}^k\)

iii) \(\beta_\Omega\) is the Borel \(\sigma\)-algebra on \(\Omega\),

vi) \(\mu\) is a probability measure on \(\Omega\).

Then let us assume that the training set is produced by sampling \(\Omega\) independently with probability \(\mu\) and obtaining

\[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n).\]

For every integrable function

\[g : X \times Y \rightarrow \mathbb{R}^k\]

a version of the Fubini’s theorem states that

\[\int_{\Omega} g(x, y) \, d\mu = \int_{X} \left( \int_{Y} g(x, y) \, d\mu(y|x) \right) \, d\mu_X\]
where $\mu_X(B) = \mu(\pi^{-1}(B))$. We define the regression function $f_\mu$ of $\mu$ by

$$f_\mu = \int_{\mathcal{Y}} y \, d\mu(y|x),$$

this is $f_\mu(x) = \mathbb{E}[y|x]$.

In each the learning algorithms presented before we define a particular family of integrable functions $\mathcal{F}$ and our goal is to “learn” (i.e., to find a good approximation of) $f_\mu$ on $\mathcal{F}$. Usually the metric is

$$\mathcal{E}(f, f_\mu) = \int_X [f - f_\mu]^2 \, d\mu_X.$$

Given that we have only a random sample of training examples from $\Omega$ we look for the best approximation in least squares error.

References


