Introduction to Machine Learning

Kyunghyun Cho

Courant Institute of Mathematical Sciences and
Center for Data Science,
New York University

December 21, 2016

Abstract

This is a lecture note for the course CSCI-UA.0473-001 (Intro to Machine Learning) at the Department of Computer Science, Courant Institute of Mathematical Sciences at New York University.

The following text books are recommended in addition to this lecture note:

- "Pattern Recognition and Machine Learning" by Chris Bishop [1]
- "Machine Learning: a probabilistic perspective" by Kevin Murphy [2]

For practical exercises on using the techniques in this lecture note, I recommend the following book:

- "Introduction to Machine Learning with Python" by Andreas Müller and Sarah Guido

Notations

Throughout this lecture note, I will use the following notational conventions:

- A bold-faced lower-case alphabet is used for a vector: $\mathbf{x}$
- A bold-faced upper-case alphabet is used for a matrix: $\mathbf{W}$
- A lower-case alphabet is often used for a scalar: x, η
- A lower-case alphabet is also used for denoting a function
-

Chapter 1

Supervised Learning

1.1 Classification

1.1.1 Problem Setup

In supervised learning, our goal is to build or find a machine M that takes as input a multi-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ and outputs a response vector $\mathbf{y} \in \mathbb{R}^{d'}$. That is,

$$M : \mathbb{R}^d \to \mathbb{R}^{d'}.$$

Of course this cannot be done out of blue, and we first assume that there exists a reference design M^* of such a machine. We then refine our goal as to build or find a machine M that imitates the reference machine M^* as closely as possible. In other words, we want to make sure that for any given $\mathbf{x}$, the outputs of M and M^* coincide, i.e.,

$$M(\mathbf{x}) = M^*(\mathbf{x}), \text{ for all } \mathbf{x} \in \mathbb{R}^d. \tag{1.1}$$

This is still not enough for us to find M, because there are infinitely many possible M's through which we must search. We must hence decide on our *hypothesis set* H of potential machines. This is an important decision, as it directly influences the difficulty in finding such a machine. When your hypothesis set is not constructed well, there may not be a machine that satisfies the criterion above.

We can state the same thing in a slightly different way. First, let us assume a function D that takes as input the outputs of M and M^* and returns how much they differ from each other, i.e.,

$$D : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}_+,$$

where $\mathbb{R}_+$ is a set of non-negative real numbers. As usual in our everyday life, the smaller the output of D the more similar the outputs of M and M^*. An example of such a function would be

$$D(a,b) = \begin{cases} 0, & \text{if } a = b \\ 1, & \text{otherwise} \end{cases}.$$

It is certainly possible to tailor this divergence function, or a *per-example cost* function, for a specific target task. For instance, consider an intrusion detection system M which takes as input a video frame of a store front and returns a binary indicator, instead of a real number, whether there is a thief in front of the store (0: no and 1: yes). When there is no thief ($M^*(\mathbf{x}) = 0$), it does not cost you anything when M agrees with M^*, but you must pay \$10 for security dispatch if M predicted 1. When there is a thief in front of your store ($M^*(\mathbf{x}) = 1$), you will lose \$100 if the alarm fails to detect the thief ($M(\mathbf{x}) = 0$) but will not lose any if the alarm went off. In this case, we may define the per-example cost function as

$$D(a,b) = \begin{cases} 0, & \text{if } a = b \\ -10, & \text{if } a = 0 \text{ and } b = 1 \\ -100, & \text{if } a = 1 \text{ and } b = 0 \end{cases}.$$

Note that this divergence is asymmetric.

Given a divergence function D, we can now state the supervised learning problem as finding a machine M, with in a given hypothesis set H, that minimizes its divergence from the reference machine M^* for any given input. That is,

$$\arg\min_{M \in H} \int_{\mathbb{R}^d} D(M^*(\mathbf{x}), M(\mathbf{x})) \mathrm{d}\mathbf{x}. \tag{1.2}$$

You may have noticed that these two conditions in Eqs. (1.1)–(1.2) are not equivalent. If a machine M satisfies the first condition, the second conditions is naturally satisfied. The other way around however does not necessarily hold. Even then, we prefer the second condition as our ultimate goal to satisfy in machine learning. This is because we often cannot guarantee that M^* is included in the hypothesis set H. The first condition simple becomes impossible to satisfy when $M^* \notin H$, but the second condition gets us a machine M that is *close* enough to the reference machine M^*. We prefer to have a suboptimal solution rather than having no solution.

The formulation in Eq. (1.2) is however not satisfactory. Why? Because not every point $\mathbf{x}$ in the input space $\mathbb{R}^d$ is born equal. Let us consider the previous example of a video-based intrusion detection system again. Because the camera will be installed in a fixed location pointing toward the store front, video frames will generally look similar to each other, and will only form a very small subset of all possible video frames, unless some exotic event happens. In this case, we would only care whether our alarm M works well for those frames showing the store front and people entering or leaving the store. Whether the divergence between the reference machine and my alarm is small for a video frame showing the earth from the outer space would not matter at all.

And, here comes probability. We will denote by $p_X(\mathbf{x})$ the probability (density) assigned to the input $\mathbf{x}$ under the distribution X, and assume that this probability reflects how likely the input $\mathbf{x}$ is. We want to emphasize the impact of the divergence D on likely inputs (high $p_X(\mathbf{x})$) while ignoring the impact on unlikely inputs (low $p_X(\mathbf{x})$). In other words, we weight each per-example cost with the probability of the corresponding example. Then the problem of supervised learning becomes

$$\arg\min_{M \in H} \int_{\mathbb{R}^d} p_X(\mathbf{x}) D(M^*(\mathbf{x}), M(\mathbf{x})) \mathrm{d}\mathbf{x} = \arg\min_{M \in H} \mathbb{E}_{\mathbf{x} \sim X} \left[D(M^*(\mathbf{x}), M(\mathbf{x})) \right]. \tag{1.3}$$

Are we done yet? No, we still need to consider one more hidden cost in order to make the description of supervised learning more complete. This hidden cost comes from the operational cost of each machine M in the hypothesis set H. It is reasonable to think that some machines are cheaper or more desirable to use than some others are. Let us denote this cost of a machine by $C(M)$, where $C : H \to \mathbb{R}_+$. Our goal is now slightly more complicated in that we want to find a machine that minimizes both the cost in Eq. (1.3) and its operational cost. So, at the end, we get

$$\arg\min_{M \in H} \underbrace{\mathbb{E}_{\mathbf{x} \sim X} \left[D(M^*(\mathbf{x}), M(\mathbf{x})) \right] + \lambda C(M)}_{\text{Expected Cost}}, \tag{1.4}$$

where $\lambda \in \mathbb{R}_+$ is a coefficient that trades off the importance between the expected divergence (between M^* and M) and the operational cost of M.

In summary, supervised learning is a problem of finding a machine M such that has bot the low expectation of the divergence between the outputs of M^* and M over the input distribution and the low operational cost.

In Reality It is unfortunately impossible to solve the minimization problem in Eq. (1.4) in reality. There are so many reasons behind this, but the most important reason is the input distribution p_X or lack thereof. We can decide on a divergence function D ourselves based on our goal. We can decide ourselves a hypothesis set H ourselves based on our requirements and constraints. All good, but p_X is not controllable in general, as it reflects how the world is, and the world does not care about our own requirements nor constraints.

Let's take the previous example of video-based intrusion system. Our reference machine M^* is a security expert who looks at a video frame (and a person within it) and determines whether that person is an intruder. We may decide to search over any arbitrary set of neural networks to minimize the expected loss. We have however absolutely no idea what the precise probability $p(\mathbf{x})$ of any video frame. Instead, we only observe $\mathbf{x}$'s which was randomly sampled from the input distribution by the surrounding environment. We have no access to the input distribution itself, but what comes out of it.

We only get to observe a *finite* number of such samples $\mathbf{x}$'s, with which we must approximate the expected cost in Eq. (1.4). This approximation method, that is approximation based on a finite set of samples from a probability distribution, is called a *Monte Carlo method*. Let us assume that we have observed N such samples: $\left\{\mathbf{x}^1, \ldots, \mathbf{x}^N\right\}$. Then we can approximate the expected cost by

$$\underbrace{\mathbb{E}_{\mathbf{x} \sim X} \left[D(M^*(\mathbf{x}), M(\mathbf{x})) \right] + \lambda C(M)}_{\text{Expected Cost}} = \underbrace{\frac{1}{N} \sum_{n=1}^{N} D(M^*(\mathbf{x}^n), M(\mathbf{x}^n)) + \lambda C(M)}_{\text{Empirical Cost}} + \varepsilon, \tag{1.5}$$

where ε is an approximation error. We will call this cost, computed using a finite set of input vectors, an *empirical cost*.

Inference We have so far talked about what is a correct way to find a machine M for our purpose. We concluded that we want to find M by minimizing the empirical cost in Eq. (1.5). This is a good start, but let's discuss why we want to do this first. There may be many reasons, but often a major complication is the expense of running the reference machine M^* or the limited access to the reference machine M^*. Let us hence make it more realistic by assuming that we will have access to M^* only once at the very beginning together with a set of input examples. In other words, we are given

$$D_{\text{tra}} = \left\{ (\mathbf{x}^1, M^*(\mathbf{x}^1)), \ldots, (\mathbf{x}^N, M^*(\mathbf{x}^N)) \right\},$$

to which we refer as a *training set*. Once this set is available, we can find M that minimizes the empirical cost from Eq. (1.5) without ever having to query the reference machine M^*.

Now let us think of what we would do when there is a *new* input $\mathbf{x} \notin D_{\text{tra}}$. The most obvious thing is to use $\hat{M}$ that minimizes the empirical cost, i.e., $\hat{M}(\mathbf{x})$. Is there any other way? Another way is to use all the models in the hypothesis set, instead of using only one model. Obviously, not all models were born equal, and we cannot give all of them the same chance in making a decision. Preferably we give a higher weight to the machine that has a lower empirical cost, and also we want the weights to sum to 1 so that they reflect a properly normalized proportion. Thus, let us (arbitrarily) define, as an example, the weight of each model as:

$$\omega(M) = \frac{1}{Z} \exp\left(-J(M, D_{\text{tra}})\right),$$

where J corresponds to the empirical cost, and

$$Z = \sum_{M \in H} \exp\left(-J(M, D_{\text{tra}})\right)$$

is a normalization constant.

With all the models and their corresponding weights, I can now think of many strategies to *infer* what the output of M^* given the new input $\mathbf{x}$. Indeed, the first approach we just talked about corresponds to simply taking the output of the model that has the highest weight. Perhaps, I can take the weighted average of the outputs of all the machines:

$$\sum_{M \in H} \omega(M) M(\mathbf{x}),$$

which is equivalent to $\mathbb{E}\left[M(\mathbf{x})\right]$ under our arbitrary construction of the weights.[1] We can similarly check the variance of the prediction. Perhaps I want to inspect a set of outputs from the top-K machines according to the weights.

We will mainly focus on the former approach, which is often called *maximum a posteriori* (MAP) in this course. However, in a few lectures, we will also consider the latter approach in the framework of *Bayesian* modelling.

[1] *Is it really arbitrary, though?*

1.1.2 Perceptron

Let us examine how this concept of supervised learning is used in practice by considering a binary classification task. Binary classification is a task in which an input vector $\mathbf{x} \in \mathbb{R}^d$ is classified into one of two classes, negative (0) and positive (1).

1.1.3 Logistic Regression

1.1.4 Support Vector Machines

One Problem, Many Loss Functions

Nonlinear Feature Function and Kernel Trick

1.1.5 Decision Tree*

1.1.6 Ensemble Methods*

1.2 Regression

1.2.1 Linear Regression and Regularization

Linear Regression

Regularization and Prior Distributions

1.2.2 Gaussian Process Regression

Bayesian Approach to Machine Learning

Gaussian Process Regression

Chapter 2

Unsupervised Learning

2.1 Dimensionality Reduction

2.1.1 Problem Setup

2.1.2 Principal Component Anslysis

Maximum Variance Criterion

Minimum Reconstruction Criterion

Probabilistic Principal Component Anlysis

Expectation-Maximization Algorithm

PCA with Missing Values: Collaborative Filtering

2.1.3 Other Dimensionality Reduction Techniques

2.2 Clustering

2.2.1 Problem Setup

Clustering vs. Dimensionality Reduction

2.2.2 k-Means Clustering

2.2.3 Mixture of Gaussians

2.2.4 Other Clustering Methods

2.3 Time-Series Modelling

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.