SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Machine Learning

Lecture notes integrated with the book "Machine Learning", Tom Mitchell

Author
Simone Bianco

November 10, 2024

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Machine Learning* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/Exyss/university-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

Sufficient knowledge of calculus, probability and algorithm design

Licence:

1

Introduction on Machine Learning

1.1 What is Machine Learning?

While many common tasks can be easily solved by computers through an algorithm, some are hard to formalize as a series of steps to be executed in a deterministic way. As an analogy, consider how language is made of syntax and semantics. Syntax can easily be formalized as a sequence of sub-structures that make up a phrase. If a sentence is slightly malformed, the machine can have an hard time trying to reconstruct the correct syntax, but in most cases this can be achieved. For semantics, instead, we have a whole different problem: some words could have more meanings, giving sentences different interpretations depending on the context of the conversation. This task is clearly harder for a machine. Sometimes, not even humans are capable of solving it!

In the past, these type of tasks where solved through *expert systems*, that being any system programmed by an human expert to solve a specific task. Expert systems can be viewed as a sequence of if-else conditions: if the task requires x then do y, and so on. Not all tasks can be solved through expert systems. In particular, some tasks need different solutions for many cases, making these primitive systems useless due to all cases being impossible to program.

To solve this type of complex and variable tasks, we use **machine learning**, which slowly teaches the machine how to solve the problem in the best way possible. The idea here is to program computers in a way that improves a specific *performance criterion* through *example data* and *past experiences*.

Machine learning uses **data mining** — the act of producing knowledge from known data — to increase the experience of the machine in solving the designed problem. In general, machine learning comes in handy when one of the following conditions holds:

- There is no human expertise on the task
- Human experts are unable to explain their expertise
- The solution needs to adapt to particular cases

The field of machine learning had an exponential growth in recent years due to the growing flood of online data — the so called *big data phenomenon* — and the increase of computational power to process such data through advanced algorithms based on theoretical results. First, we give a formal definition of a *learning problem*.

Definition 1: Learning problem

A **learning problem** is the improvement over a task T with respect to a performance measure P based on experience E.

For example, suppose that we want to program a machine that learns how to play checkers. We define the learning problem as:

- The task T is to play checkers
- The performance measure P is the percentage of games won in a tournament
- The experience E is the opportunity to play against self

But how can we improve such performance measure? What *exactly* should the machine learn? These questions reduce the learning problem to finding a valid mathematical representation of T, P and E. The training process can be described by four phases:

1. The human expert suggests what is an optimal move for each configuration of the board
2. The human expert evaluates each configuration, ranking them by optimality
3. The computer plays against an human an automatically detects with configurations lead to a win, a loss or a draw
4. The computer plays against itself to improve performance

Formally, this whole process can be expressed as a simple mathematical function called **target function**. In particular, we want to choose a target function that represents the learning problem in the best way possible and that can be computed by a machine.

For instance, consider the function $V : \text{Board} \to \mathbb{R}$, defined as follows:

- If b is a final board state and it is a win then $V(b) = 100$
- If b is a final board state and it is a loss then $V(b) = -100$
- If b is a final board state and it is a draw then $V(b) = 0$
- If b is not a final board state then $V(b) = V(b^*)$, where b^* is the best final board state that can be achieved starting from the board b playing the optimal moves

This function perfectly models our learning problem. However, it cannot be computed by any program since we haven't defined what an optimal set of moves is. We need a new definition that encodes this concept of optimal strategy for a checkers game.

For example, we can re-define V as follows:

$$V(b) = w_0 + w_1 \cdot \text{bp}(b) + w_2 \cdot \text{rp}(b) + w_3 \cdot \text{bk}(b) + w_4 \cdot \text{rk}(b) + w_5 \cdot \text{bt}(b) + w_6 \cdot \text{rt}(b)$$

where:

- $\text{bp}(b)$ is the number of black pieces
- $\text{rp}(b)$ is the number of red pieces
- $\text{bk}(b)$ is the number of black kings
- $\text{rk}(b)$ is the number of red kings
- $\text{bt}(b)$ is the number of red pieces threatened by black pieces
- $\text{rt}(b)$ is the number of black pieces threatened by red pieces

With this formulation, we have reduced the concept of leaning checkers to estimating the best possible values of the coefficients $w_1, \ldots, w_6$, which care called **weights**, that maximize the value of $V(b)$ for any board state b. This estimation process is referred to as *learning the function* V.

Definition 2: Learned function

Given a target function $f : X \to Y$ with weights $w_1, \ldots, w_k$, we define the **learned function** $\hat{f} : X \to Y$ as the current approximation of f computed by a learning algorithm.

By definition, the learned function $\hat{f}$ will never be equal to the target function f: the target function's weights are always unknown by definition. The idea here is to approximate f by repeatedly applying small changes to the weights $\hat{w}_1, \ldots, \hat{w}_k$ of $\hat{f}$ in order to estimate the weights of f. To learn a function f, we need a *dataset*. A dataset is a set of instances that can be used by a learning algorithm to improve the performance of the learned function.

Definition 3: Dataset

Let $f : X \to Y$ be a target function and let $f_{\text{train}}(x)$ be the training value obtained by x in the training data. Given a set of n training inputs $X_D = \{x_1, \ldots, x_n\}$, the **dataset** of the learning problem is the set of samples defined as:

$$D = \{(x_i, f_{\text{train}}(x_i)) \mid x_i \in X_D\}$$

After training, the learned function $\hat{f}$ will have learned the values of the inputs in the dataset, returning a value as close as possible to the one in the dataset (in some cases the returned value is exactly the same). However, we are interested in estimating the *other* possible inputs, i.e. those that aren't in the dataset.

In summary, a machine learning problem is the task of learning a target function $f : X \to Y$ through a dataset D for a set X_D of n inputs. To learn a function f means computing an approximating function $\hat{f}$ that returns values as close as possible to f, especially for values outside of the dataset D, implying that $\forall x \in X - - - X_D$ it should holds that $f(x) \approx \hat{f}(x)$. In order for the learned function to be *good*, the set of training inputs X_D must be very very small compared to the set of total inputs, meaning that $|X_D| \lll |X|$.

There are distinct types of machine learning problems based on:

- The type of dataset used:
    1. **Supervised learning**: problems where the model learns patterns from labeled data. Here, the dataset corresponds to $D = \{(x_y, y_i) \mid i \in X_D\}$, where y_i is the sample of the function value for x_i
    2. **Unsupervised learning**: problems where the model learns patterns from unlabeled data. Here, the dataset corresponds to $D = \{x_y \mid i \in X_D\}$
    3. **Reinforcement learning**: problems in which an agent learns to make decisions by interacting with an environment and receiving rewards or penalties based on its actions.
- The type of function to be learned:
    1. **Concept learning**: the input set is $X = A_1 \times \ldots \times A_n$, where A_i is a finite set, and the output set is $Y = \{0, 1\}$.
    2. **Discrete Classification**: the input set is $X = A_1 \times \ldots \times A_n$, where A_i is a finite set, and the output set is $Y = \{c_1, \ldots, c_k\}$.
    3. **Discrete Regression**: the input set is $X = A_1 \times \ldots \times A_n$, where A_i is a finite set, and the output set is $Y = \mathbb{R}^m$.
    4. **Continuous Classification**: the input set is $X = \mathbb{R}^n$ and the output set is $Y = \{c_1, \ldots, c_k\}$.
    5. **Continuous Regression**: the input set is $X = \mathbb{R}^n$ and the output set is $Y = \mathbb{R}^m$.

Classification problems are based on the classification of inputs into predetermined categories, while regression problems involve the approximation of functions defined over $\mathbb{R}$. Reinforcement learning, instead, is used for dynamic systems with unknown or partially known evolution model, usually robotic tasks and game playing.

1.2 Hypotheses, Consistency and Noise

After discussing the basic notation and terminology, we are ready to deepen our understanding on how to learn a problem. Given a target function f, we consider a set of functions H called **hypothesis space**. Each hypothesis is a possible approximation of f, where $h(x)$ is the estimation of h over x for $f(x)$. The learning task is to find the best approximation $h^* \in H$ of the function f using the dataset D.

For now, we won't care about *how* the hypothesis space is represented (there are many ways to do so). By definition, the value $h(x)$ can be computed for each $x \in X$, but we can check whether $h(x) = f(x)$ or not only for instances $x \in X_D$. This means that some hypothesis may have some values that are *inconsistent* with the dataset itself, making them useless.

Definition 4: Consistent Hypothesis

An hypothesis h is said to be **consistent** with a dataset D of a target function f if and only if $h(x) = f(x)$ for all $x \in X_D$

This reduces our interest in finding the best hypothesis h^* that maximizes a performance P and that is consistent with the dataset, i.e. the function with the best approximation of f for each $x \notin X_D$ and that returns the exact value of $f(x')$ for each $x' \in X_D$.

For these reasons, we will base our results on the **inductive learning hypothesis**: any hypothesis that approximates the target function well over a sufficiently large dataset will also approximate the target function well over other unobserved examples. This concept is implicit by restricting our interest to the *version space*.

Definition 5: Version space

The **version space** of a target function f with respect to the hypothesis space H and the dataset H, written as $\mathrm{VS}_{H,D}$, is the subset of H that contains all the hypothesis that are consistent with D.

$$\mathrm{VS}_{H,D} = \{h \in H \mid \forall x \in X_D \;\; h(x) = f(x)\}$$

Through this definition, the best hypothesis h^* corresponds to:

$$h^* \in \underset{h \in V_{H,D}}{\arg\max}\, P(h, D)$$

The simplest way to compute the version space is through bruteforce: we enumerate all the hypothesis space and test the consistency of each hypothesis, discarding the invalid ones. This algorithm is clearly *infeasible* since enumerating all the different hypothesis would require an immense amount of time.

Now, let's take a step back. Consider a *concept learning* problem. The idea is pretty straightforward: we want to teach a concept to a machine. Here, the concept is described as a target function $c : X \to \{0, 1\}$, where X is called **instance space** and the dataset corresponds to $D = \{(x_i, c(x_i)) \mid i \in X_D\}$. By definition, any hypothesis inside the hypothesis space is actually **associated** with a particular set of instances, that being all instances that are classified as positive by such hypothesis. In fact, we have a mapping ϕ between the hypothesis space H and the power set $\mathcal{P}(X)$.

$$\phi_H : H \to \mathcal{P}(x) : h \mapsto \{x \in X \mid h(x) = 1\}$$

In general, however, this mapping is not surjective. Suppose now that, for a given concept problem, there is an hypothesis space H' for which $\phi_{H'}$ is effectively surjective, implying that H' can represent any subset of X. We notice that:

- For all instances $x \in X - X_D$, there are two hypothesis $h_1, h_2 \in \mathrm{VS}_{H,D}$ such that $h_1(x) = 0$ and $h_2(x) = 1$. This means that the hypothesis space H' is *perfect* for the dataset, but it cannot be used to predict the value of any instance that is not sampled.
- There is an instance $x \in X - X_D$ for which there are two hypothesis $h_1, h_2 \in \mathrm{VS}_{H,D}$ such that $h_1(x) = 0$ and $h_2(x) = 1$. This means that, even thought the space H is not *perfect* for the dataset, the space still cannot predict the value for some instance that is not sampled.

In other words, if the hypothesis space is too **powerful** and the search is complete, then the system won't be able to classify new instances, meaning that we have no generalization power. These deductions imply that an algorithm that outputs a version space is always unusable because it cannot totally or partially predict instances that aren't in D. Hence, the more information the hypothesis space encapsulates about the values in X_D, the harder it becomes to generalize and predict values for samples outside XD. In other words, a more expressive hypothesis space can **overfit** to the data, making it more difficult to make accurate predictions on unsampled data. We will return to the concept of overfitting in the next sections. If the output of the algorithm is a single hypothesis, instead, the prediction for any instance $x \in X - X_D$ will either be 0 or 1, with no possible conflicts. Now, we look at another fundamental problem in defining in concept learning.

The process of reducing the representation power in favor of **generalization power** — as in reducing the hypothesis space from H' to H — is called **language bias**. This bias imposes a *restriction* on the language, used to represent the hypothesis. Moreover, the process of *selecting* one particular hypothesis among the set of possible ones — i.e., choosing $h^* \in H$ — is called **search bias**. Note that, in some contexts, it is also possible to choose a hypothesis h^* within H' directly.

In machine learning, the concept of learning bias is crucial for improving a model's ability to generalize. A good learning bias helps guide the learning algorithm towards patterns in the data that are useful for predicting unseen samples, increasing the system's generalization power. This bias allows the model to make accurate predictions on new data that wasn't part of the training set. Without such a bias, a system would simply **memorize** the dataset, failing to predict values for samples outside the training set, rendering it

ineffective in real-world applications. Systems lacking generalization capabilities would be of little use, as they wouldn't be able to provide meaningful predictions beyond the data they were trained on.

In real-world applications, datasets often contain **noise**, which refers to irrelevant or erroneous information that can distort the true underlying patterns in the data. Noise can come from a variety of sources, including measurement errors, data entry mistakes, incomplete data or random fluctuations in the system being studied. This noise complicates the *learning process*, as machine learning models may *struggle* to distinguish between true *signal* and *noise*.

A noisy data-point in a dataset D for a function f can be formulated as a pair $(x_i, y_i) \in D$, where $y_i \neq f(x_i)$. This means that there may be *no consistent hypothesis* with noisy data, i.e. $\text{VS}_{H,D} = \varnothing$. In these scenarios, **statistical methods** must be employed to implement robust algorithms, in order to reduce the noise in the data.