CMU 15-281 Spring 2024: Machine Learning Lecture Notes

Nihar B. Shah and Tuomas Sandholm

Machine learning is a part of AI where machines learn primarily via examples (as opposed to primarily learning via human-specified rules) in a manner that can generalize to previously unseen situations.

1 Some examples

We will consider what is perhaps the most canonical form of machine learning – supervised binary classification. Let us look at some examples of it and then we'll parse through what this means:

- Given an email, an email provider would like to mark it as spam or not spam.
- Any company that has users conducting financial transactions would develop methods to identify fraudulent (vs. non-fraudulent) transactions.
- Camera companies want to design algorithms such that given an image, it can figure out whether it contains a human face or not. (If there is a human face in the view of the lens then the camera should automatically shout "cheese." This would look silly if there was no person in front of the camera.)
- Given an image of a set of cells from the body, researchers are trying to design methods that can tell whether the cells are cancerous.
- Companies' hiring departments are trying to design methods which, given a candidate's CV, would output whether this candidate should be interviewed or not.

In each of these examples, the output takes two possible values (e.g., spam vs. not spam). Hence the term "binary." The objective is to classify the input into one of these two values ("classes"), and hence the term "classification."

2 General recipe

In the description below, we will simplify certain aspects of machine learning to enable presentation in the class.

For simplicity, we will assume that the input (the email or the image etc.) can be written as a vector of real numbers (say, of length d). This can be done, for instance in the case of grayscale images, by taking the pixels and putting them in one long vector. The two possible classes are denoted as -1 and 1.

The goal then is to figure out a function, which we will denote as $f : \mathbb{R}^d \to \{-1, 1\}$. This function will take the email/image/... as input and then output its predicted class. The key question is: how do we figure out what this function should be?

To design this function, we assume we have access to (lots of) prior data. For instance in the case of spam detection, we have access to lots of emails that were previously marked by humans as spam or not spam. This prior data is used to *supervise* these methods, and hence the term "supervised" in supervised binary classification. This data is called "training data" and the process of using this training data to come up with a function f is called "training." Let us formalize this in terms of some notation. Suppose we have access to n such emails. Recall that we are converting all inputs to a vector of length d. Let us denote these n emails as $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$. For each email x_i, we also have an associated "label" $y_i \in \{-1, 1\}$ where 1 may represent

spam and -1 may represent not spam. Hence the training data comprises $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The labels may not be perfect (may have some errors) but are still assumed to be reasonably accurate.

Now suppose we have access to this training data. How do we come up with a function $f : \mathbb{R}^d \to \{-1, 1\}$ that can take as input a *new* email you just received and output whether the email is spam or not? The general recipe is as follows:

- Assume that f takes a certain form.
- Among all possible functions f that take this form, choose the f which makes the least amount of mistakes on the *training* data.
- Output this f.

3 Linear classifiers

Let us make this concrete via one specific form for function f. Let us suppose that f takes a linear combination of the coordinate values of its input vector. Then if the result is positive, it outputs 1 otherwise it outputs -1. More formally, any such function f is associated with a vector $w \in \mathbb{R}^d$ and a value $b \in \mathbb{R}$. Then $f(x) = sign(w^T x + b)$.[1] Such a function is called a "linear classifier." Why? Suppose $d = 2$. Consider some $w \in \mathbb{R}^2$ and $b \in \mathbb{R}$, say $w = [1, 0.5]$ and $b = 0$. Now consider a graph where you have the first coordinate of $x \in \mathbb{R}^2$ on the x axis and the second coordinate on the y axis. Can you plot f on this graph?[2]

For a general dimension d, any such function f operates similarly. The values w specify what is called a hyperplane (which is simply a generalization of a line). If the input x falls on one side of the hyperplane, it will output 1 and on the other side it will output -1.

The next natural question you should ask is – suppose we have chosen the form of f. Then how do we decide what exactly this function f should be. For instance, suppose we have decided that we want a linear classifier, then what should be the values of (w, b)?

As mentioned earlier, to answer this question we will rely on the training data. We need to design an algorithm such that given the training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, it finds the values of (w, b) such that the number of training data points $i \in \{1, 2, ..., n\}$ such that $sign(w^T x_i + b) = y_i$ is as large as possible.

How do you find such (w, b)? There are various algorithms. Two of the most famous ones are as follows.

3.1 Perceptron Algorithm

This was invented by Rosenblatt in the 1950s. It assumes that there does exist some (w, b) such that it perfectly classifies the training data. The question it addressed was: how do we find this (w, b)?

The algorithm is simple to describe. It is an iterative algorithm:

1. Initialize $w = 0$ and $b = 0$.
2. If $sign(w^T x_i + b) = y_i$ for all $i \in \{1, 2, ..., n\}$ then return this w and b.
3. Pick any arbitrary $i \in \{1, 2, ..., n\}$ such that $sign(w^T x_i + b) \neq y_i$; let's call it i_0.
4. Update $w \leftarrow w + y_{i_0} x_{i_0}$ and $b \leftarrow b + y_{i_0}$.
5. Go to step 2.

If there truly exists some (w, b) that correctly classifies the entire training data, then the algorithm is guaranteed to stop in a finite time and output a (w, b) which correctly classifies all training data. (However, there doesn't exist any such (w, b) then this algorithm will keep running forever.)

[1] You can assume sign(0)=1.

[2] The term "linear" has occurred previously in this course. Recall linear constraints from linear programming, which also had the term "linear." Try to establish a connection between the two places we have seen "linear" so far in the course.

I encourage you to implement and try out this algorithm! Choose some (w, b) and some $x_1, \ldots, x_d$; generate $y_i = sign(w^T x_i + b)$ for every i; then run this algorithm on the training data you have generated. Does this output a (w, b) which correctly classifies all training data? Is it always the same (w, b) that you had initially chosen?

3.2 Support Vector Machines (SVMs)

While the perceptron tries to find some (w, b) which correctly classifies all training data, SVMs try to find the "best" (w, b) among all possible choices which can correctly classify all training data. The notion of best is in terms of how far is the separating line from the nearest training point. The algorithm extensively relies on optimization-based techniques.

4 First AI winter

After the invention of the perceptron algorithm, there was a huge excitement about AI. In fact, the New York Times carried an article which quoted the navy saying that that "it expects [the algorithms] will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." There was huge hype.

And then came a bust. People started reporting negative results. Here is a simple one. Suppose $d = 2$. Suppose you just want to learn an XOR function, that is, suppose you have $x_1 = [-1, -1], y_1 = -1$, $x_2 = [1, -1], y_2 = 1$, $x_3 = [-1, 1], y_3 = 1$ and $x_4 = [1, 1], y_4 = -1$. Then is there any (w, b) that correctly classifies all of the training data? No. The perceptron algorithm on this data will keep running forever. So if the perceptron cannot even learn such a simple function, how can it be expected to do far more complex things like reading, writing, etc.?!

Researchers then had also started working on early incarnations of deep learning (discussed below) but they did not have enough computation nor enough data to succeed.

5 Deep learning

In subsequent years, people said that the perceptron still seemed a bit useful, but the form of f that it assumed was very restrictive. So...why not put many such perceptrons together to create a more complex form of f which can capture the XOR function and much more?! And this is precisely what are called deep neural networks or deep learning.

Let us go into more detail. Here is how you construct a more complex form of f. The function f takes as input x. Now, instead of having just one perceptron, the input x is fed in parallel to many perceptrons. Each of them is called a "neuron."[3] Each of these perceptrons can have different values of (w, b). Let us suppose we used k_1 such neurons. We collect the outputs of these neurons to form a k_1-length vector. This vector is now fed as input to k_2 other neurons, which can have its own values of (w, b). The outputs are then collected as a vector, and this is again fed as input to k_3 other neurons. And so on. The resulting f can now capture XOR functions and other far more complex items.

This entire function f is called a neural network. Each time you have a set of neurons to which the same input is fed, this is called a "layer." The number of layers is called the "depth." The number of neurons in any layer is called the "width" of the layer. The width of the widest layer in a neural network is called the width of the neural network. The original perception we had studied earlier was a single layer neural network. If you have more than one layer, you call it deep learning or deep neural networks.

Now given training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, one finds the function f of this form (that is, one finds values of the parameters for *all* the perceptrons) such that it correctly classifies as much of the training data as possible. This is done through optimization algorithms like "graadient descent" and its variants.

[3] In practice, you use a slight generalization of a perceptron, where the sign is replaced by other functions (called "activation functions"). This distinction is beyond the scope of this lecture.