Lecture Notes 1: Machine Learning Basics

Instructor: Ashok Cutkosky

Machine learning is all about identifying some pattern in data and applying those patterns to new data. Formally, let us suppose that you have bunch of datapoints or observations $\mathbf{z}_1, \dots, \mathbf{z}_N$. For example, each $\mathbf{z}_i$ could be a tuple $(\mathbf{x}_i, \mathbf{y}_i)$ where x_i is some object like a picture or the state of a board game or information about a medical patient, and $\mathbf{y}_i$ is some label like an object in the picture, the best move in the board game, or the diagnosis of the patient. The typical machine learning workflow consists of two main choices:

1. Choose some kind of *model* to explain the data. In supervised learning in which $\mathbf{z} = (\mathbf{x}, \mathbf{y})$, typically we pick some function f and use the model $\mathbf{y} \approx f(\mathbf{x}, \mathbf{w})$ where $\mathbf{w}$ is the *parameter* of the model. We will let W be the set of acceptable values for $\mathbf{w}$.

2. Fit the model to the data. That is, using our dataset $\mathbf{z}_1, ;\mathbf{z}_N$, choose some value $\hat{\mathbf{w}}$. This is often called *training* the model.

For most of this course, will consider the case that $W = \mathbb{R}^d$ for some d so that we are attempting to minimize a real-valued function of a real vector.

For example, in a binary classification task if $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \{-1, 1\}$, we might choose the simple linear classifier model

$$\mathbf{y} \approx \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle)$$

After completing step 2, we will need some way to measure how good our choice of $\hat{\mathbf{w}}$ is. This is accomplished through a *loss function* $\mathcal{L}(\mathbf{w})$. Typically, the loss function takes the form of some kind of expectation:

$$\mathcal{L}(\mathbf{w}) = \mathop{\mathbb{E}}_{\mathbf{z}}[\ell(\mathbf{w}, \mathbf{z})]$$

where the expectation is over some unknown distribution of observations $\mathbf{z} \sim P_{\mathbf{z}}$. In this course, we will consider solely losses of this form. For example, in the binary classification task, we might write:

$$\ell(\mathbf{w}, \mathbf{z}) = \mathbb{1}[\mathbf{y} \neq \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle)]$$

This loss is 1 if we misclassify the example $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ and zero otherwise. The full loss $\mathcal{L}(\mathbf{w}) = \mathbb{E}[\ell(\mathbf{w}, \mathbf{z})]$ is simply the probability of misclassifying a random example. In general, when $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ and we have some model $y \approx f(\mathbf{w}, \mathbf{z})$, it is common for the loss to take the form:

$$\ell(\mathbf{w}, \mathbf{z}) = l(f(\mathbf{w}, \mathbf{z}), \mathbf{y})$$

However, in order to allow for greater generality we will not use this notion of l and f and consider just the composition ℓ.

With this idea, we can formally characterize the task of step 2: we are trying to find a $\hat{\mathbf{w}}$ that minimizes the loss $\mathcal{L}(\mathbf{w})$. Thus, step two is trying to identify:

$$\mathbf{w}_\star = \mathop{\text{argmin}}_{\mathbf{w} \in W} \mathcal{L}(\mathbf{w})$$

We can measure our ability to find the argmin by the *suboptimality gap*:

$$\mathcal{L}(\hat{\mathbf{w}}) - \mathcal{L}(\mathbf{w}_\star)$$

If we have a small suboptimality gap, then we have done a good job of training the model. Note that this does not mean that the model is actually good: it might be that $\mathcal{L}(\mathbf{w}_\star)$ is actually quite high because we made a poor choice of model in the first place.

This problem is difficult for two main reasons. The first of these is a *computational* barrier. It turns out that minimizing arbitrary functions $\mathcal{L}$ is computationally intractable (in fact, even the example just provided for linear classification is NP-hard to minimize). This issue is usually addressed by choosing some *surrogate* loss ℓ^S with $\mathcal{L}^S(w) = \mathbb{E}_z[\ell^S(w,z)]$. Then, we need to instead solve the problem:

$$\operatorname*{argmin}_{w\in W} \mathcal{L}^S(w)$$

The surrogate should be chosen such that:

1. $\mathcal{L}^S(w) \approx \mathcal{L}(w)$.
2. It is computationally feasible to minimize $\mathcal{L}^S$.

There is some natural tension here: surrogates that more accurately capture the original loss $\mathcal{L}$ may be harder to minimize.

The second reason the problem is difficult is that usually we do not even know how to evaluate the function $\mathcal{L}$!. In particular, we do not know the distribution over z in $\mathcal{L}(w) = \mathbb{E}_z[\ell(w,z)]$. This makes the problem of minimizing $\mathcal{L}(w)$ not just an optimization problem but also a *stochastic* optimization problem ("stochastic" is really just a fancy word for "involving randomness").

The classical way to deal with this is through *empirical risk minimization*. Here, we make use of a dataset $\mathbf{z}_1, \dots, \mathbf{z}_N$ to define:

$$\hat{\mathcal{L}}^S(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell^S(\mathbf{w}, \mathbf{z}_i)$$

We make a critical assumption: suppose that each $\mathbf{z}_i$ is drawn independently from the distribution $P_{\mathbf{z}}$. Intuitively, we should expect $\hat{\mathcal{L}}^S \approx \mathcal{L}^S$ since averages typically approach their mean. Then, instead of trying to minimize $\mathcal{L}^S$, which we do not know, we can minimize $\hat{\mathcal{L}}^S$. The function $\hat{\mathcal{L}}^S(\mathbf{w})$ is the empirical loss. This leads to the overall suboptimality gap:

$$\mathcal{L}(\hat{\mathbf{w}}) - \mathcal{L}(\mathbf{w}_\star) = \underbrace{\mathcal{L}(\hat{\mathbf{w}}) - \mathcal{L}^S(\hat{\mathbf{w}})}_{\text{surrogate quality}} + \underbrace{\mathcal{L}^S(\hat{\mathbf{w}}) - \hat{\mathcal{L}}^S(\hat{\mathbf{w}})}_{\text{empirical approximation}} + \underbrace{\hat{\mathcal{L}}^S(\hat{\mathbf{w}}) - \hat{\mathcal{L}}^S(\mathbf{w}_\star)}_{\text{suboptimality from algorithm}} + \mathcal{L}^S(\mathbf{w}_\star) + \mathcal{L}^S(\mathbf{w}_\star) - \mathcal{L}(\mathbf{w}_\star)$$

This idea is called empirical risk minimization because in statistics the loss function is often called the "risk", and so $\hat{\mathcal{L}}$ is the "empirical risk".

Empirical risk minimization comes with the important caveat that the point $\hat{\mathbf{w}}_\star = \operatorname{argmin} \hat{\mathcal{L}}^S(\mathbf{w})$ may not be very close to the actual minimizer $\mathbf{w}_\star$. As a result, we can end up with a situation in which

$$\hat{\mathcal{L}}^S(\hat{\mathbf{w}}) < \mathcal{L}^S(\mathbf{w}_\star)$$

This is known as *overfitting*. There is an entire field of statistical learning theory dedicated to understand which properties of ℓ^S lead to overfitting and which do not, and there are still many open problems in this area. A similar amount of work has gone into identifying conditions under which there are reasonable surrogate losses ℓ^S that are computationally tractable while also providing sufficiently good approximations to the true loss ℓ. However, for our purposes whenever we perform empirical risk minimization we will simply assume that the surrogate ℓ^S is such that the overfitting issue is not as important as actually solving the optimization problem. Also, we will generally assume that the surrogate ℓ^S is a good enough approximation to the loss we care about ℓ.

We will usually analyze exclusively the surrogate loss $\mathcal{L}^S(\mathbf{w}) = \mathbb{E}_{\mathbf{z}}[\ell^S(\mathbf{w}, \mathbf{z})]$. In order to keep notation simpler, we will drop the S and simply refer to this as the loss $\mathcal{L}$ and ℓ. If we ever actually need the true losses, it will be made clear on a case-by-case basis.

There is second way of dealing with the problem of unknown $\mathcal{L}^S$ that becomes viable when the size N of the dataset is extremely large, as may happen in some modern language modeling tasks for which data is scraped en masse off of the internet. In this case, N might be so large that it is computationally infeasible to look at any one datapoint $\mathbf{z}_i$ more than once. In this case we will be able to design algorithms that provably cannot overfit.

Assuming the overfitting problem is negligible, we are left with the managing the quality of the surrogate loss. Remember that we want to choose the surrogate loss ℓ^S such that $\mathcal{L}(\mathbf{w}) = \mathbb{E}_{\mathbf{z}}[\ell(\mathbf{w}, \mathbf{z})] \approx \mathbb{E}_{\mathbf{z}}[\ell^S(\mathbf{w}, \mathbf{z})] = \mathcal{L}^S(\mathbf{w})$ and also such that we are able to design algorithms that minimize $\mathcal{L}^S$ effectively. A common strategy to ensure that $\mathcal{L}^S$ is minimizable easily is to make $\ell(\mathbf{w}, \mathbf{z})$ a *convex* function of $\mathbf{w}$ for all $\mathbf{z}$. A convex function is one for which the *epigraph*:

$$\{(\mathbf{w}, k) \in \mathbb{R}^{d+1} : k \geq \ell(\mathbf{w}, \mathbf{z})\}$$

is a convex set. Algebraically, when ℓ is differentiable, this implies the inequality for all $\mathbf{w}, \mathbf{w}'$

$$\ell(\mathbf{w}', \mathbf{z}) \leq \ell(\mathbf{w}, \mathbf{z}) + \langle \nabla \ell(\mathbf{w}, \mathbf{z}), \mathbf{w}' - \mathbf{w} \rangle$$

In words, the value of the first-order Taylor approximation to ℓ at any point $\mathbf{w}$ is smaller than the true function value at all other points. This means that *local* information at a point $\mathbf{w}$ in terms of the gradient $\nabla \ell(\mathbf{w}, \mathbf{z})$ actually provides a *global* bound on the value of $\ell(\mathbf{w}, \mathbf{z})$ at every other point. This property has engendered a vast body of literature on algorithms for optimizing such losses.

Unfortunately, essentially all of the popular deep learning models used to obtain impressive results today do not appear to allow for surrogates ℓ^S that are both good approximations of the true loss ℓ and also convex. Thus, we will focus much of our study on what can be done without using convexity.