Chapter 7

Empirical Risk Minimization

The first two parts of this course introduced the notions of decision boundaries and likelihoods, and we saw how both these concepts could be used to define classification models. Implicit in the previous discussions has been the concept of *optimization*, i.e., the idea of learning or refining a model's parameters according to some criterion. In Chapter 3 we saw how the perceptron algorithm iteratively optimized a linear decision boundary by correcting mistakes. In Chapters 5 and 6 we saw how machine learning models can be learned by optimizing the maximum likelihood criterion.

In this next part of the course, we will focus on the optimization perspective of machine learning. As with prior chapters, this content is not distinct from the concepts previously introduced. Rather, we will aim to provide more concrete details and justification for ideas that were only implicit or discussed in passing throughout the previous chapters. In this chapter, we will start by introducing the theoretical framework of *empirical risk minimization.*

7.1 Loss Functions and Minimizing Risk

From an optimization perspective, the goal of machine learning is to find some set of parameters Θ that maximizes our predictive performance. Put in other way, our goal is to find an optimal set of parameters that minimizes the number of mistakes we make.

We can formalize this idea as follows. Assume that we have probability distribution $P_{\mathcal{D}}$ over datapoints $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^m$ are the input features and $y_i \in \mathbb{R}$ is the target for point i. Note that assuming real-valued features and targets leaves open the possibility of having binary, integer, or categorical values as a special case. Assume that we are also given some non-negative loss function $L(y, \hat{y})$, which measures the error or mismatch between the prediction $\hat{y}$ and the true label y. For instance, in the case of binary classification, a natural

loss function would be the 0-1 loss function, which gives 0 error for correct predictions and an error of 1 for incorrect predictions. Lastly, we assume that we have some prediction model f_Θ that generates predictions based on parameters Θ and that this model belongs to a particular family $\mathcal{F}$. For example, we might consider the $\mathcal{F}$ to be the family of linear models.

Under this setup, our goal is to solve the following optimization problem

$$\arg \min_{f_\Theta \in \mathcal{F}} \mathbb{E}_{(\mathbf{x},y)\sim P_\mathcal{D}}[L(y, f_\Theta(\mathbf{x}))] \tag{7.1}$$

$$= \arg \min_{f_\Theta \in \mathcal{F}} \int_{\mathbb{R}^m \times \mathbb{R}} [L(y, f_\Theta(\mathbf{x}))dP_\mathcal{D}(\mathbf{x}, y). \tag{7.2}$$

The intuition is that we want to find the best model $f_\Theta \in \mathcal{F}$, which minimizes the expected value of our error over the full dataset distribution $P_\mathcal{D}$.

7.2 Minimizing Empirical Risk

Of course, in practice we will not have access to the true underlying data distribution $P_\mathcal{D}$. Instead, we have access to finite training and testing datasets. In this case, we can assume that our training set $\mathcal{D}_{\text{trn}}$ corresponds to a set of independently and identically distributed samples from the distribution $P_\mathcal{D}$. Using these samples, we can then estimate the performance of a particular model (i.e., a particular set of parameters) using a quantity called the empirical risk

$$R(\Theta) = \frac{1}{|\mathcal{D}_{\text{trn}}|} \sum_{(\mathbf{x}_i,y_i)\in\mathcal{D}_{\text{trn}}} L(y_i, f_\Theta(\mathbf{x}_i)). \tag{7.3}$$

We then say that a function f^*_Θ minimizes the *empirical risk* on the training dataset if and only if

$$f^*_\Theta = \arg \min_{f_\Theta \in \mathcal{F}} R(\Theta). \tag{7.4}$$

Note that we use notation here to emphasize that the empirical risk is a function of the model parameters Θ; i.e., when we attempt to minimize $R(\Theta)$, we do so by optimizing over the parameter set.

The i.i.d. assumption In the above discussion, we assumed that the training datapoints are *independent and identically distributed (i.i.d)*. This is a critical assumption, which ensures that the empirical risk is a consistent estimator of the true underlying risk. We must assume that all the training examples are *independent*, which means that our training points cannot depend on one another. We also must assume that the points are identically distributed, which means that they are all drawn from the same distribution (i..e, $P_\mathcal{D}$). From a statistical perspective, these assumptions ensure that the empirical average performance converges to the true performance over the full underlying distribution as our number of training examples increases.

7.2.1 Training and Generalization

It is important to note that minimizing the empirical risk on a particular training set $\mathcal{D}_{\text{trn}}$ does not necessarily mean that our model will do well on new samples from the distribution $P_{\mathcal{D}}$. This is the problem of generalization. For instance, if our training data consists of only a tiny number of examples, then we cannot reasonably expect to find model that will generalize to new examples well, especially if we have a large possible space of models (e.g., if $|\mathcal{F}| >> |\mathcal{D}_{\text{trn}}|$).

The issue is that our true goal is to minimize the error on some underlying data distribution $P_{\mathcal{D}}$, but we almost never have access to this full distribution. For example, in the case of classifying spam emails, the underlying data distribution would correspond to the distribution of all possible emails that might ever be sent!

In practice, we can estimate generalization performance by evaluating on a held-out test set $\mathcal{D}_{\text{tst}}$. However, it is important to note that this is just another proxy for the true performance of the model on the underlying data distribution. This issue of generalization is at the core of machine learning, and we will return to it in detail in Chapter 10.

7.3 Loss Functions in Practice

The empirical risk minimization perspective is powerful because we have the flexibility of defining different loss functions, depending on our constraints and the task at hand. In the case of binary classification, we might use the 0-1 loss mentioned above. In a regression task, we might use the absolute difference $|y - \hat{y}|$ between our prediction and the true value as a loss function.

Desiderata for loss functions

It is important to realize that not all loss functions are created equal. While some loss functions may appear intuitive and natural (e.g., the 0-1 loss for classification), in practice they can be difficult to work with. Generally, when designing loss functions, we need to balance two—often conflicting—desiderata:

1. **The loss should reflect the performance goals of the task.** For instance, if we are doing binary classification, then the 0-1 loss might be the most natural loss function, but this loss function would not be appropriate for a regression task with real-valued targets.

2. **We must be able to optimize the loss function efficiently.** In particular, we want to be able to solve the empirical risk minimization problem with a reasonable amount of computational expense (e.g., time and space complexity). This desire often conflicts with the first desire above. The 0-1 loss function, for example, requires exponential time complexity to optimize in many cases.

In the remainder of this course, we will see various functions—for both classification and regression—that attempt to balance these objectives.

Perceptron and the hinge-loss

For example, in the perceptron model of Chapter 3, we implicitly optimized a loss called the hinge-loss. Here, we must assume that the output $\hat{y}$ of the perceptron is the raw prediction $\hat{y} = \mathbf{w}^\top \mathbf{x}$ (i.e., before we apply the sign function). Moreover, as in Chapter 3, we assume that the negative class is denoted by $y = -1$ rather than $y = 0$ for mathematical convenience. With these assumptions, we can write the loss of the perceptron as

$$\begin{aligned} L(y, \hat{y}) &= \frac{1}{2}(\text{sign}(\hat{y}) - y)\hat{y} \\ &= \begin{cases} 0 & \text{if } \text{sign}(\hat{y}) = y \\ \hat{y} & \text{otherwise.} \end{cases} \end{aligned}$$

The basic idea behind the hinge-loss is that we achieve 0 loss whenever we get the right prediction. However, when we make the wrong prediction, we penalize the model proportionally to how strong the incorrect prediction was. Note that in the perceptron hinge-loss we compare the raw output of the linear model (i.e., a continuous value) with the binary target value. This is common in loss functions. We often compare a real-valued prediction (e.g., a probability) with a discrete valued target.

Likelihood-based losses and cross-entropy

In Chapters 5 and 6 we introduced the idea of learning machine learning models based on the maximum likelihood principle. We can, in fact, reinterpret this approach from the perspective of empirical risk minimization. In this view, the goal of those methods is to minimize a likelihood-based loss, usually defined as the *negative log-likelihood loss*. For example, in the case of logistic regression, the maximum likelihood objective corresponds to the following negative log-likelihood loss:

$$L_{\text{x-ent}}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}). \tag{7.5}$$

where $\hat{y}$ is the probability predicted by the model. This loss is actually a very popular loss function used in binary classification and is known as the cross-entropy loss.[1] Unlike the 0-1 loss, the cross-entropy loss is easy to optimize, as we will discuss in Chapter 8. The cross-entropy loss also has a natural probabilistic interpretation, where the error is the negative log-probability that our model assigns to the true label.

Likelihood-based loss functions—including the cross-entropy loss—are some of the most popular loss functions in machine learning. The general recipe is that you first define some probabilistic assumptions for your data (e.g., the Bernoulli Naive Bayes assumption), which allow you to define a model that assigns probabilities to datapoints. Then, you can define a loss based on the negative log-likelihood that the model assigns to your data.

[1] The cross-entropy name comes from connections to information theory, which we will discuss in Chapter 12.