CHAPTER 5

Logistic regression

1 Machine learning as optimization

The perceptron algorithm was originally written down directly via cleverness and intuition, and later analyzed theoretically. Another approach to designing machine learning algorithms is to frame them as optimization problems, and then use standard optimization algorithms and implementations to actually find the hypothesis. Taking this approach will allow us to take advantage of a wealth of mathematical and algorithmic technique for understanding and solving optimization problems, which will allow us to move to hypothesis classes that are substantially more complex than linear separators.

We begin by writing down an *objective function* $J(\Theta)$, where Θ stands for *all* the parameters in our model. Note that we will sometimes write $J(\theta, \theta_0)$ because when studying linear classifiers, we have used these two names for parts of our whole collection of parameters, so $\Theta = (\theta, \theta_0)$. We also often write $J(\Theta; \mathcal{D})$ to make clear the dependence on the data $\mathcal{D}$. The objective function describes how we feel about possible hypotheses Θ: we will generally look for values for parameters Θ that minimize the objective function:

$$\Theta^* = \arg\min_{\Theta} J(\Theta) \quad .$$

You can think about Θ^* here as "the theta that minimizes J".

A very common form for an ML objective is

$$J(\Theta) = \left(\frac{1}{n} \sum_{i=1}^{n} \underbrace{\mathcal{L}(h(x^{(i)}; \Theta), y^{(i)})}_{\text{loss}} \right) + \underbrace{\lambda}_{\text{constant}} \underbrace{R(\Theta)}_{\text{regularizer}} \quad . \tag{5.1}$$

The *loss* tells us how unhappy we are about the prediction $h(x^{(i)}; \Theta)$ that Θ makes for $(x^{(i)}, y^{(i)})$. A common example is the 0-1 loss, introduced in chapter 1:

$$L_{01}(h(x; \Theta), y) = \begin{cases} 0 & \text{if } y = h(x; \Theta) \\ 1 & \text{otherwise} \end{cases} \quad ,$$

which gives a value of 0 for a correct prediction, and a 1 for an incorrect prediction. In the case of linear separators, this becomes:

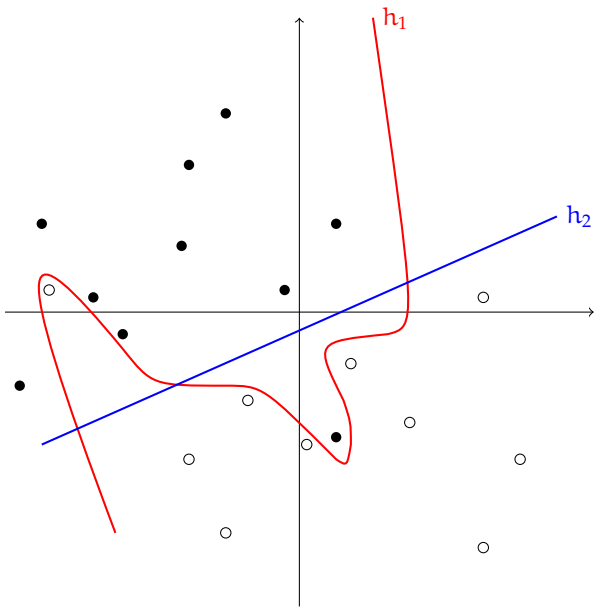
$$L_{01}(h(x; \theta, \theta_0), y) = \begin{cases} 0 & \text{if } y(\theta^T x + \theta_0) > 0 \\ 1 & \text{otherwise} \end{cases} \quad .$$

2 Regularization

If all we cared about was finding a hypothesis with small loss on the training data, we would have no need for regularization, and could simply omit the second term in the objective. But remember that our ultimate goal is to *perform well on input values that we haven't trained on!* It may seem that this is an impossible task, but humans and machine-learning methods do this successfully all the time. What allows *generalization* to new input values is a belief that there is an underlying regularity that governs both the training and testing data. We have already discussed one way to describe an assumption about such a regularity, which is by choosing a limited class of possible hypotheses. Another way to do this is to provide smoother guidance, saying that, within a hypothesis class, we prefer some hypotheses to others. The regularizer articulates this preference and the constant λ says how much we are willing to trade off loss on the training data versus preference over hypotheses.

This trade-off is illustrated in the figure below. Hypothesis h_1 has 0 training loss, but is very complicated. Hypothesis h_2 mis-classifies two points, but is very simple. In absence of other beliefs about the solution, it is often better to prefer that the solution be "simpler," and so we might prefer h_2 over h_1, expecting it to perform better on future examples drawn from this same distribution. Another nice way of thinking about regularization is that we would like to prevent our hypothesis from being too dependent on the particular training data that we were given: we would like for it to be the case that if the training data were changed slightly, the hypothesis would not change by much.

To establish some vocabulary, we say that h_1 is *overfit* to the training data.



A common strategy for specifying a *regularizer* is to use the form

$$R(\Theta) = \left\|\Theta - \Theta_{prior}\right\|^2$$

when we have some idea in advance that θ ought to be near some value Θ_{prior}. In the absence of such knowledge a default is to *regularize toward zero*:

$$R(\Theta) = \|\Theta\|^2 \quad .$$

Learn about Bayesian methods in machine learning to see the theory behind this and cool results!

3 A new hypothesis class: linear logistic classifiers

For classification, it is natural to make predictions in $\{+1, -1\}$ and use the $0-1$ loss function. However, even for simple linear classifiers, it is very difficult to find values for θ, θ_0 that minimize simple training error

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\text{sign}(\theta^T x^{(i)} + \theta_0), y^{(i)}) \ .$$

This problem is NP-hard, which probably implies that solving the most difficult instances of this problem would require computation time *exponential* in the number of training examples, n.

The "probably" here is not because we're too lazy to look it up, but actually because of a fundamental unsolved problem in computer-science theory, known as "P vs NP."

What makes this a difficult optimization problem is its lack of "smoothness":

- There can be two hypotheses, (θ, θ_0) and (θ', θ_0'), where one is closer in parameter space to the optimal parameter values (θ^*, θ_0^*), but they make the same number of misclassifications so they have the same J value.
- All predictions are categorical: the classifier can't express a degree of certainty about whether a particular input x should have an associated value y.

For these reasons, if we are considering a hypothesis θ, θ_0 that makes five incorrect predictions, it is difficult to see how we might change θ, θ_0 so that it will perform better, which makes it difficult to design an algorithm that searches through the space of hypotheses for a good one.

For these reasons, we are going to investigate a new hypothesis class: *linear logistic classifiers*. These hypotheses are still parameterized by a d-dimensional vector θ and a scalar θ_0, but instead of making predictions in $\{+1, -1\}$, they generate real-valued outputs in the interval $(0, 1)$. A linear logistic classifier has the form

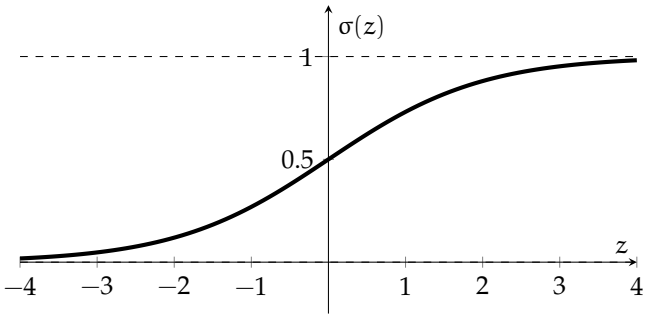
$$h(x; \theta, \theta_0) = \sigma(\theta^T x + \theta_0) \ .$$

This looks familiar! What's new?

The *logistic* function, also known as the *sigmoid* function, is defined as

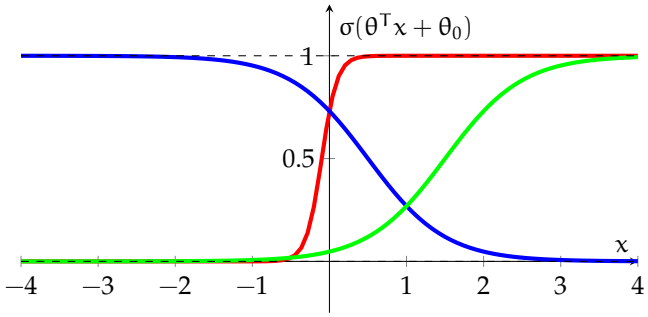
$$\sigma(z) = \frac{1}{1 + e^{-z}} \ ,$$

and plotted below, as a function of its input z. Its output can be interpreted as a probability, because for any value of z the output is in $(0, 1)$.



Study Question: Convince yourself the output of σ is always in the interval $(0, 1)$. Why can't it equal 0 or equal 1? For what value of z does $\sigma(z) = 0.5$?

What does a linear logistic classifier (LLC) look like? Let's consider the simple case where $d = 1$, so our input points simply lie along the x axis. The plot below shows LLCs for three different parameter settings: $\sigma(10x + 1)$, $\sigma(-2x + 1)$, and $\sigma(2x - 3)$.



Study Question: Which plot is which? What governs the steepness of the curve? What governs the x value where the output is equal to 0.5?

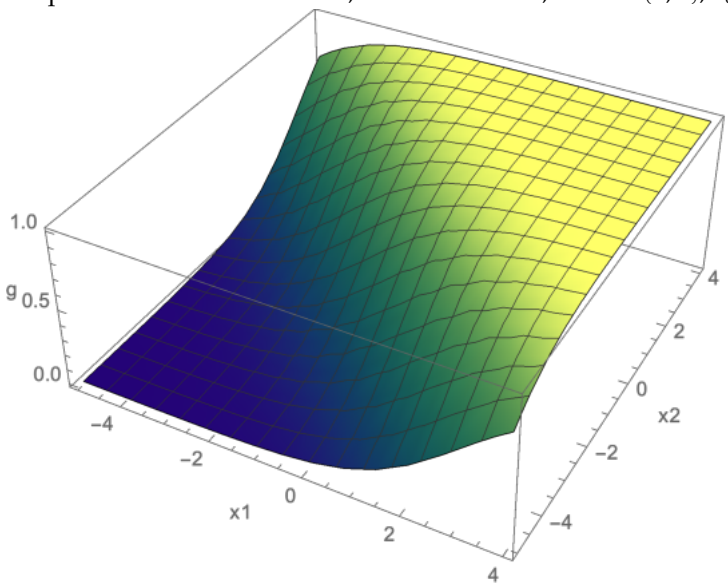
But wait! Remember that the definition of a classifier from chapter 2 is that it's a mapping from $\mathbb{R}^d \to \{-1, +1\}$ or to some other discrete set. So, then, it seems like an LLC is actually not a classifier!

Given an LLC, with an output value in $(0, 1)$, what should we do if we are forced to make a prediction in $\{+1, -1\}$? A default answer is to predict $+1$ if $\sigma(\theta^T x + \theta_0) > 0.5$ and -1 otherwise. The value 0.5 is sometimes called a *prediction threshold*.

In fact, for different problem settings, we might prefer to pick a different prediction threshold. The field of *decision theory* considers how to make this choice from the perspective of Bayesian reasoning. For example, if the consequences of predicting $+1$ when the answer should be -1 are much worse than the consequences of predicting -1 when the answer should be $+1$, then we might set the prediction threshold to be greater than 0.5.

Study Question: Using a prediction threshold of 0.5, for what values of x do each of the LLCs shown in the figure above predict $+1$?

When $d = 2$, then our inputs x lie in a two-dimensional space with axes x_1 and x_2, and the output of the LLC is a surface, as shown below, for $\theta = (1, 1), \theta_0 = 2$.



Study Question: Convince yourself that the set of points for which $\sigma(\theta^T x + \theta_0) = 0.5$, that is, the separator between positive and negative predictions with prediction threshold 0.5 is a line in (x_1, x_2) space. What particular line is it for the case in the figure above? How would the plot change for $\theta = (1, 1)$, but now with $\theta_0 = -2$? For $\theta = (-1, -1), \theta_0 = 2$?

4 Loss function for logistic classifiers

We have defined a class, LLC, of hypotheses whose outputs are in $(0, 1)$, but we have training data with y values in $\{+1, -1\}$. How can we define a loss function? Intuitively, we would like to have *low loss if we assign a low probability to the incorrect class.* We'll define a loss function, called *negative log-likelihood* (NLL), that does just this. In addition, it has the cool property that it extends nicely to the case where we would like to classify our inputs into more than two classes.

In order to simplify the description, we will assume that (or transform so that) the labels in the training data are $y \in \{0, 1\}$, enabling them to be interpreted as probabilities of being a member of the class of interest. We would like to pick the parameters of our classifier to maximize the probability assigned by the LCC to the correct y values, as specified in the training set. Letting guess $g^{(i)} = \sigma(\theta^T x^{(i)} + \theta_0)$, that probability is

Remember to be sure your y values have this form if you try to learn an LLC using NLL!

$$\prod_{i=1}^{n} \begin{cases} g^{(i)} & \text{if } y^{(i)} = 1 \\ 1 - g^{(i)} & \text{otherwise} \end{cases} ,$$

under the assumption that our predictions are independent. This can be cleverly rewritten, when $y^{(i)} \in \{0, 1\}$, as

$$\prod_{i=1}^{n} g^{(i)^{y^{(i)}}} (1 - g^{(i)})^{1 - y^{(i)}} .$$

Study Question: Be sure you can see why these two expressions are the same.

Now, because products are kind of hard to deal with, and because the log function is monotonic, the θ, θ_0 that maximize the log of this quantity will be the same as the θ, θ_0 that maximize the original, so we can try to maximize

$$\sum_{i=1}^{n} \left(y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)}) \right) .$$

We can turn the maximization problem above into a minimization problem by taking the negative of the above expression, and write in terms of minimizing a loss

$$\sum_{i=1}^{n} \mathcal{L}_{\text{nll}}(g^{(i)}, y^{(i)})$$

where $\mathcal{L}_{\text{nll}}$ is the *negative log-likelihood* loss function:

$$\mathcal{L}_{\text{nll}}(\text{guess}, \text{actual}) = -\left(\text{actual} \cdot \log(\text{guess}) + (1 - \text{actual}) \cdot \log(1 - \text{guess})\right) .$$

This loss function is also sometimes referred to as the *log loss* or *cross entropy*.

You can use any base for the logarithm and it won't make any real difference. If we ask you for numbers, use log base e.

5 Logistic classification as optimization

We can finally put all these pieces together and develop an objective function for optimizing regularized negative log-likelihood for a linear logistic classifier. In fact, this process is usually called "logistic regression," so we'll call our objective J_{lr}, and define it as

That's a lot of fancy words!

$$J_{lr}(\theta, \theta_0; \mathcal{D}) = \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{nll}(\sigma(\theta^T x^{(i)} + \theta_0), y^{(i)}) \right) + \lambda \|\theta\|^2 \quad .$$

Study Question: Consider the case of linearly separable data. What will the θ values that optimize this objective be like if $\lambda = 0$? What will they be like if λ is very big? Try to work out an example in one dimension with two data points.