Chapter 6

Logistic Regression

In the last chapter, we introduced the Naive Bayes model. The key idea in this approach is that we decompose the prediction probabilities based on Bayes rules:

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \tag{6.1}$$

This allowed us to learn a prediction model by separately estimating the feature likelihoods (i.e., $P(\mathbf{x}|y)$) and the class likelihoods (i..e, $P(y)$).

This general idea of decomposing the prediction probability via Bayes Rule is known as the *generative* approach to supervised classification models. This generative approach is usually contrasted with the *discriminative* approach, where we directly estimate the $P(y \mid \mathbf{x})$, rather than relying on a Bayes-rule based decomposition.

6.1 Logistic Regression Model

In a discriminative model, we directly parameterize the conditional probability $P(y \mid \mathbf{x})$. There are many possible ways to specify this parameterization; many of which we will cover in this course. In this section, we will start with the most fundamental approach, where we parameterize $P(y \mid \mathbf{x})$ based on a linear model.

A first approach

How might we parameterize $P(y \mid \mathbf{x})$ as a linear function? A simple idea would just be to define that

$$P(y \mid \mathbf{x}) = \mathbf{w}^\top \mathbf{x}. \tag{6.2}$$

However, the issue with such an approach is that it does not generate proper probabilities, because the range of an unconstrained linear function is the entire real line $\mathbb{R}$.

The logistic function

In order to get proper probabilities out of a linear function, we rely on what is called the *logistic* function (also known as the *sigmoid*). The logistic function is defined as follows:

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{6.3}$$

and it is almost always denoted by a σ. The key property of the logistic function is that it maps the real line to probabilities. In other words, the domain of the logistic is $z \in \mathbb{R}$, but the range of the logistic is well-defined probability values $\sigma(z) \in [0, 1]$. A visual plot of the logistic function is shown in Figure 6.1. From this plot, we can see that the logistic function has an S-like curve that asymptotes at 0 and 1.

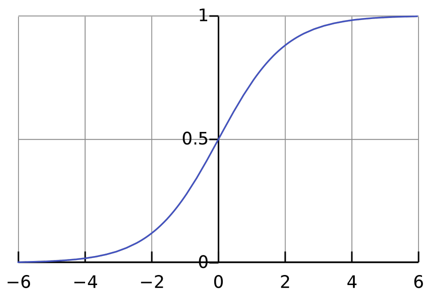

Figure 6.1: Illustration of the logistic function

Properties of the logistic function The logistic function has some useful properties that we will make use of throughout the course:

- It can be re-written as follows:

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z} \tag{6.4}$$

- Its derivative has a simple form:

$$\frac{\partial \sigma(z)}{\partial z} = (1-\sigma(z))\sigma(z). \tag{6.5}$$

- It has the following symmetry property:

$$1 - \sigma(z) = \sigma(-z) \tag{6.6}$$

We leave it to the reader to verify these facts algebraically. We will use them throughout the course.

Combining the logistic function and a linear model

We can use the logistic function to specify our linear classification model:

$$\begin{aligned} P(y \mid \mathbf{x}) &= \sigma(\mathbf{w}^\top \mathbf{x}) && (6.7) \\ &= \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}. && (6.8) \end{aligned}$$

Based on the properties of the logistic, we know that this model is guaranteed to output proper probabilities. Moreover, we can also show that this model defines a linear decision boundary, as we intended. As with Naive Bayes, we use the log-odds ratio to specify the decision boundary:[1]

$$\begin{aligned} \log\left(\frac{P(y=1 \mid \mathbf{x})}{P(y=0 \mid \mathbf{x})}\right) &= \log\left(\frac{\sigma(\mathbf{w}^\top \mathbf{x})}{1 - \sigma(\mathbf{w}^\top \mathbf{x})}\right) && (6.9) \\ &= \log\left(\frac{\frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}}{1 - \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}}\right) && (6.10) \\ &= \log\left(\frac{\frac{e^{\mathbf{w}^\top \mathbf{x}}}{1+e^{\mathbf{w}^\top \mathbf{x}}}}{\frac{1}{1+e^{\mathbf{w}^\top \mathbf{x}}}}\right) && (6.11) \\ &= \log\left(e^{\mathbf{w}^\top \mathbf{x}}\right) && (6.12) \\ &= \mathbf{w}^\top \mathbf{x}. && (6.13) \end{aligned}$$

In terms of a prediction function, we end up with

$$f_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} > \gamma \\ 0 & \text{if } \mathbf{w}^\top \mathbf{x} < \gamma, \end{cases} \qquad (6.14)$$

where as usual $\gamma = 0$ is a natural decision boundary (i.e., corresponding to a 50/50 odds ratio).[2] Note that as with Naive Bayes we end up with a linear decision boundary that resembles the perceptron from Chapter 3. The key difference is that we derive and optimize this decision boundary based on a likelihood model.

6.2 Training a Logistic Regression Model

Logistic regression is a probabilistic model, so we can optimize it using the maximum likelihood approach.

[1]In fact, an alternative way to derive the logistic regression model is by assuming that the log-odds decision boundary is a linear function and deriving the logistic prediction function as a consequence.

[2]As with the perceptron and Naive Bayes, we consider predictions undefined for points lying directly on the decision boundary, an event with 0 probability of occurrence.

Likelihood for the logistic model

The log-likelihood of a training dataset $\mathcal{D}_{\text{trn}}$ can be written as

$$\log \mathcal{L}(\mathcal{D}_{\text{trn}}, \mathbf{w}) = \log \left(\prod_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} \sigma(\mathbf{w}^\top \mathbf{x}_i)^{y_i} (1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))^{1-y_i} \right) \tag{6.15}$$

$$= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} y_i \log(\sigma(\mathbf{w}^\top \mathbf{x}_i)) + (1 - y_i) \log(\sigma(-\mathbf{w}^\top \mathbf{x}_i)). \tag{6.16}$$

In other words, the likelihood is analogous to the Bernoulli likelihood, but we have replaced the θ parameter of the Bernoulli with the probability output $\sigma(\mathbf{w}^\top \mathbf{x}_i)$ from the linear logistic model.

Maximizing the likelihood

As usual, we aim to maximize the likelihood by taking the derivative of the likelihood and finding the parameters that set the expression to zero. In this case, since we aim to optimize the parameter vector $\mathbf{w}$, we will compute the *gradient* of the likelihood with respect to this full vector, rather than treating the individual components separately:

$$\nabla_{\mathbf{w}} \log \mathcal{L}(\mathcal{D}_{\text{trn}}, \mathbf{w}) = \nabla_{\mathbf{w}} \left(\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} y_i \log(\sigma(\mathbf{w}^\top \mathbf{x}_i)) + (1 - y_i) \log(\sigma(-\mathbf{w}^\top \mathbf{x}_i)) \right)$$

$$= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} y_i \mathbf{x}_i \frac{\sigma(\mathbf{w}^\top \mathbf{x}_i)(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))}{\sigma(\mathbf{w}^\top \mathbf{x}_i)} - (1 - y_i) \mathbf{x}_i \frac{\sigma(-\mathbf{w}^\top \mathbf{x}_i)(1 - \sigma(-\mathbf{w}^\top \mathbf{x}_i))}{\sigma(-\mathbf{w}^\top \mathbf{x}_i)}$$

$$= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} y_i \mathbf{x}_i (1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)) - (1 - y_i) \mathbf{x}_i \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

$$= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} \mathbf{x}_i \left(y_i - y_i \sigma(\mathbf{w}^\top \mathbf{x}_i) - \sigma(\mathbf{w}^\top \mathbf{x}_i) + y_i \sigma(\mathbf{w}^\top \mathbf{x}_i) \right)$$

$$= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} \mathbf{x}_i \left(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i) \right) \tag{6.17}$$

Unfortunately, however, it is not feasible to set this expression to zero and solve for $\mathbf{w}$, i.e., there is no closed form solution for the maximum.

Gradient ascent

As there is no closed-form solution that maximizes the likelihood in this case, we must opt for a numerical solution. Luckily, there is a simple numerical procedure that can be used to obtain the maximum of a function. In particular, we can use the *gradient ascent* approach, where we repeatedly shift our parameters in the direction of the gradient using the following update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha \nabla_{\mathbf{w}} \log \mathcal{L}(\mathcal{D}_{\text{trn}}, \mathbf{w}^{(k)}). \tag{6.18}$$

Plugging in the log-likelihood of logistic regression gives us the following

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{trn}}} \mathbf{x}_i \left(y_i - \sigma((\mathbf{w}^{(k)})^\top \mathbf{x}_i) \right) \tag{6.19}$$

The idea is that we continually take steps in the direction specified by the gradient, which corresponds to the direction of maximal increase for the function. Once we reach a stopping point where the gradient is zero, we will be at the maximum of function. The α hyperparameter controls the *step size* or *learning rate* for the gradient ascent, i.e., how large the update is at each step.

We will discuss the theory of gradient ascent (and descent) in more detail in the next chapter. In the context of logistic regression, it suffices to say that gradient ascent is guaranteed to find the maximum in finite iterations, as long as we choose a reasonable step size.

Connections to the perceptron algorithm

The gradient ascent update equation for logistic regression (Equation 6.19) is remarkably similar to the update equation for the perceptron algorithm. Whereas the perceptron always shifted by a factor of $\pm\mathbf{x}_i$, the logistic function scales the update by the term $\left(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)\right)$ (as well as the step size). The term $\left(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)\right)$ essentially corresponds to the magnitude of the error in the prediction. For example, if the true label $y_i = 1$ is positive and the prediction $\sigma(\mathbf{w}^\top \mathbf{x}_i)$ is small (i.e., near zero), then there will be a large correction in the positive direction, whereas if the true label $y_i = 0$ is negative and $\sigma(\mathbf{w}^\top \mathbf{x}_i)$ is large (i.e., near one), then there will be a large correction in the negative direction. Thus, one way of interpreting logistic regression is that it is a soft, probabilistic version of a perceptron. Whereas the perceptron predicts only positive or negative, the logistic model makes probabilistic predictions and scales its update factors according to the magnitude of its probabilistic prediction error.

6.3 The Softmax and Multiclass Models

The logistic regression model is inherently a binary classification approach. However, this model can be adapted to work with multiclass problems using a generalization of the logistic function, called the *softmax*. The softmax function allows one to define probabilities over a discrete set of outcomes. Formally, a k-dimensional softmax maps a k-dimensional real vector $\mathbf{z} \in \mathbb{R}$ to a probability value on $k - 1$-dimensional unit simplex Δ^{k-1}. Or, in more intuitive terms, it maps a vector of k real values to a normalized probability distribution over k discrete outcomes. The softmax is defined as

$$\text{softmax}(\mathbf{z})[i] = \frac{e^{\mathbf{z}[i]}}{\sum_{j=1}^{k} e^{\mathbf{z}[j]}}. \tag{6.20}$$

One can verify that the two-dimensional softmax recovers the logistic function as a special case.

To define a linear, multiclass classification model, one can combine the softmax with a linear function to define the prediction probability for each class, given some input features:

$$P(y = k|\mathbf{z}) = \text{softmax}(\mathbf{W}^\top \mathbf{x})[k], \tag{6.21}$$

where now the model parameters are defined by a matrix $\mathbf{W} \in \mathbb{R}^{k \times m}$, rather than a single vector. An analogous maximum likelihood and gradient ascent approach can be used to train such a multi-class model.

6.4 Pros and Cons of Discriminative Learning

Both generative (e.g., Naive Bayes) and discriminative (e.g., logistic regression) models can be useful for classification tasks. However, it is worth considering their pros and cons. The key difference between these approaches is that generative models decompose the $P(y|\mathbf{x})$ using Bayes rule, whereas discriminative approaches directly model the conditional probability. As a consequence, discriminative models tend to have fewer parameters and be faster to train. On the other hand, generative models can leverage stronger assumptions about the structure of feature and class likelihoods, which often makes these models perform better on smaller datasets. In practice, discriminative models are more popular than generative ones—e.g., most neural networks are discriminative—but generative models are still useful in tasks that involve small datasets or where prior knowledge can be encoded into the probability distributions.