Chapter 9

Linear Regression

The optimization and loss function view of machine learning allows us to easily generalize to different kinds of target outputs. So far in this course we have been focused on the supervised classification task, where the target outputs are categorical. Using the loss function perspective, we can easily generalize and design a model for real-valued targets, i.e., the *regression* task. In this chapter, we introduce the standard loss function used for regression and discuss how it can be combined with a basic linear model.

Motivating example

As a motivating example, consider a (synthetic) medical example, where we aim to predict how long (e.g., the number of days) it will take for a cancer patient to go into remission after receiving treatment. In a simple univariate model, we might use the size of a tumour (e.g., the diameter in millimeters) as a feature for the prediction. Here, our input and output values are both continuous, making regression a natural fit for this task.[1] A visual illustration of what this data might look like is shown in Figure 9.1.

9.1 Loss Function for Regression

One might imagine that a natural loss function for linear regression would be the mean absolute error (MAE)

$$L(y, \hat{y}) = |y - \hat{y}|. \tag{9.1}$$

The MAE is natural because it simply measures the absolute value of the difference between the prediction $\hat{y}$ and the ground truth y. This is sensible for regression because this loss is well-defined for the entire range of real values. It also fits with the intuition that a regression loss should measure the distance between the prediction and the target.

[1] We assume that fractional values for days are permissible.

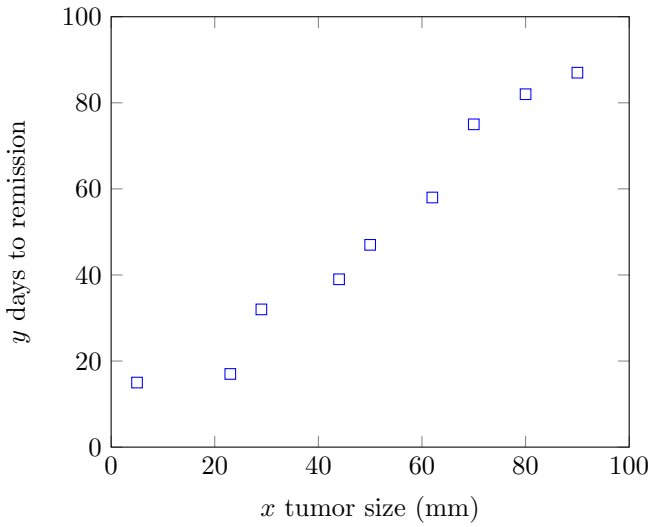

Figure 9.1: Illustration of a simple univariate regression dataset, with the single feature (i.e., tumor size) on the x-axis and the target (i.e., time to remission) on the y-axis.

However, the issue with the MAE loss is that it is not smooth and easily differentiable, due to the use of the absolute value. As an alternative, the standard loss function used in regression is the mean-squared error (MSE):

$$L(y, \hat{y}) = \|y - \hat{y}\|^2 \tag{9.2}$$
$$= (y - \hat{y})^2. \tag{9.3}$$

Like the MAE loss, the MSE has a natural interpretation. In this case we are taking the (squared) Euclidean distance between the prediction and the target, rather than absolute distance. The Euclidean distance is a natural measure of distance in Euclidean space, and taking the square is simply necessary to ensure that this function is twice differentiable. Compared to the MAE, the main difference of the MSE is that its quadratic nature assigns relatively higher penalties as the magnitude of the errors increase. In other words, the MSE tends to penalize larger errors more aggressively, compared to the MAE.

9.2 A Linear Regression Model

We can combine the MSE with a linear model to obtain one of the most classic and essential machine learning approaches: *linear regression*. Similar to the previous linear models covered in this course, we define the prediction function as

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}, \tag{9.4}$$

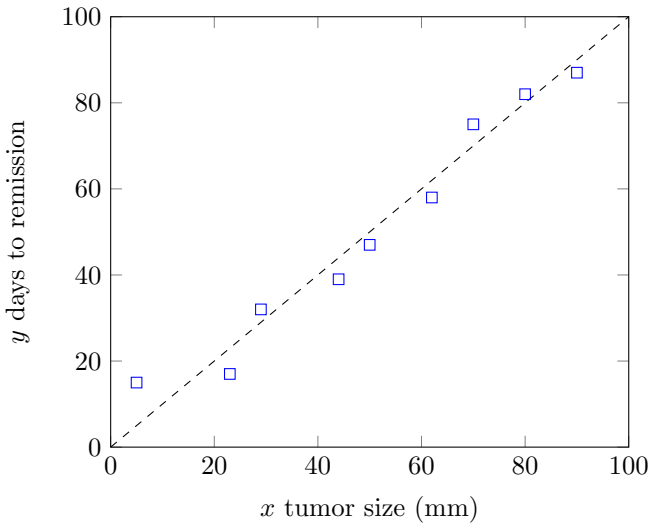

Figure 9.2: Illustration of the best fit regression line for the (artificial) medical prediction dataset.

where $\mathbf{w} \in \mathbb{R}^m$ is a vector of learnable parameters and $\mathbf{x} \in \mathbb{R}^m$ is a vector of input features. Note that as with previous linear models, we must assume that one of the input features is a constant value. Otherwise, we must add an explicit bias/intercept term b to the equation, making it

$$f(\mathbf{x}) = b + \mathbf{w}^\top \mathbf{x}. \tag{9.5}$$

As usual, we omit the explicit intercept term without loss of generality and for notational convenience.

Taking the linear model into account, the full empirical risk minimization for linear regression can be written as:

$$\arg\min_{\mathbf{w}\in\mathbb{R}^m} R(\mathbf{w}) = \arg\min_{\mathbf{w}\in\mathbb{R}^m} \frac{1}{|\mathcal{D}_{\text{trn}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{trn}}} (y - \mathbf{w}^\top \mathbf{x})^2 \tag{9.6}$$

$$= \arg\min_{\mathbf{w}\in\mathbb{R}^m} \frac{1}{n} \|\mathbf{X}_{\text{trn}}\mathbf{w} - \mathbf{Y}_{\text{trn}}\|^2. \tag{9.7}$$

Here, we use $\mathbf{X}_{\text{trn}} \in \mathbb{R}^{n\times m}$ to denote a matrix with $n = |\mathcal{D}_{\text{trn}}|$ rows, with each row corresponding to a vector of training features (i.e., $\mathbf{X}_{\text{trn}}[:, i] = \mathbf{x}_i$). Similarly, $\mathbf{Y}_{\text{trn}} \in \mathbb{R}^n$ is a vector with each entry corresponding to a different training target (i.e., $\mathbf{Y}_{\text{trn}}[i] = y_i$). In intuitive terms, we want to find the linear model that minimizes the average squared Euclidean distance between the prediction and the target.

9.3 Optimizing Linear Regression

One option to optimize a linear regression model is to use gradient descent. Using the vector notation of Equation 9.7, a full batch gradient descent approach would result in the following update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha^{(k)} \nabla_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}_{\text{trn}}\mathbf{w}^{(k)} - \mathbf{Y}_{\text{trn}}\|^2 \tag{9.8}$$

$$= \mathbf{w}^{(k)} - \alpha^{(k)} \nabla_{\mathbf{w}} \frac{1}{n} (\mathbf{X}_{\text{trn}}\mathbf{w}^{(k)} - \mathbf{Y}_{\text{trn}})^\top (\mathbf{X}_{\text{trn}}\mathbf{w}^{(k)} - \mathbf{Y}_{\text{trn}}) \tag{9.9}$$

$$= \mathbf{w}^{(k)} - \alpha^{(k)} \frac{2}{n} \mathbf{X}_{\text{trn}}^\top (\mathbf{X}_{\text{trn}}\mathbf{w}^{(k)} - \mathbf{Y}_{\text{trn}}) \tag{9.10}$$

$$= \mathbf{w}^{(k)} - \alpha^{(k)} \frac{2}{n} \left(\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}\mathbf{w}^{(k)} - \mathbf{X}_{\text{trn}}^\top \mathbf{Y}_{\text{trn}} \right). \tag{9.11}$$

However, linear regression is also one of the models where a closed-form solution is attainable for the minimizer. If we take the gradient, set to zero, and solve, we get the following

$$\nabla_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}_{\text{trn}}\mathbf{w} - \mathbf{Y}_{\text{trn}}\|^2 = 0 \tag{9.12}$$

$$\frac{2}{n} \left(\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}\mathbf{w} - \mathbf{X}_{\text{trn}}^\top \mathbf{Y}_{\text{trn}} \right) = 0 \tag{9.13}$$

$$\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}\mathbf{w} = \mathbf{X}_{\text{trn}}^\top \mathbf{Y}_{\text{trn}} \tag{9.14}$$

$$\mathbf{w} = (\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}})^{-1} \mathbf{X}_{\text{trn}} \mathbf{Y}_{\text{trn}}. \tag{9.15}$$

9.3.1 Gradient descent or closed form?

Both the gradient descent and closed-form approaches have pros and cons.

Computational complexity The computational complexity of computing the closed form solution is $\mathcal{O}(n^2 m + n^3)$, since we need to compute and invert the $n \times n$ matrix $\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}$. In contrast, a single step of gradient descent has time complexity $\mathcal{O}(n^2 m)$, since its cost is dominated by the matrix multiplication to compute $\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}$. Thus, gradient descent will have efficiency benefits when $m << n$ (i.e., the number of features is less than the number of training examples) and when the number of iterations K required to converge is much less than n. In practice, minibatch stochastic gradient descent can also be used, which has even cheaper iteration costs and can often converge equally fast as full batch gradient descent.

Stability and interpretability One benefit of the closed-form solution is that we do not need to rely on an iterative algorithm, which may lead to spurious or incorrect solutions (e.g., if the step size is incorrect). With the closed-form solution, we can use an off-the-shelf matrix inversion algorithm, which guarantees stability. In addition, we can detect *ill-posed* cases where $\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}$ is not

invertible. In these cases, the matrix $\mathbf{X}_{\text{trn}}^\top \mathbf{X}_{\text{trn}}$ is *singular*, which implies that there is no unique solution to the optimization problem. This can happen—for example—when $m >> n$, meaning we have too many features and not enough training examples. The benefit of the closed-form solution is that we will generally detect this situation (e.g., our matrix inversion algorithm will give an error), whereas gradient descent will end up producing a solution, which may be unreasonable.

Flexibility The gradient descent approach has the upper hand when it comes to flexibility. As we will see in later chapters, it is often advantageous to combine different loss terms into a single optimization. The gradient descent approach is able to accommodate such modifications gracefully, as long as the additional loss terms are smooth, differentiable, and preferably convex. On the other hand, there is no guarantee that we can find a closed-form solution when combining the MSE loss with other loss functions.

9.4 A Maximum-Likelihood Perspective

It is possible to derive the mean-squared error and the linear regression based on the notion of maximum likelihood. However, the derivation is far less intuitive, compared to the classification setting. In the linear regression setting, we assume the following probabilistic model

$$p(y|\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \epsilon, \tag{9.16}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is random Gaussian noise with some constant variance σ^2. Equation 9.17 assumes that the distribution of the target values corresponds to the additive combination of our deterministic prediction function $\mathbf{w}^\top \mathbf{x}$ as well as some Gaussian noise. Intuitively, we assume that our data is not a perfect linear function, but that it corresponds to a linear function with some noise added on top. Figure 9.3 illustrates this idea.

Another way of interpreting this assumption is that the data corresponds to a normal distribution with a conditional mean centered around $\mathbf{w}^\top \mathbf{x}$

$$p(y|\mathbf{x}) = \mathcal{N}(\mathbf{w}^\top \mathbf{x}, \sigma^2), \tag{9.17}$$

where $\mathcal{N}(\mathbf{w}^\top \mathbf{x}, \sigma^2)$ denotes a normal distribution with mean $\mathbf{w}^\top \mathbf{x}$ and some constant variance σ^2. Equation 9.17 again implies that our target values are normally distributed around the prediction line $\mathbf{w}^\top \mathbf{x}$.

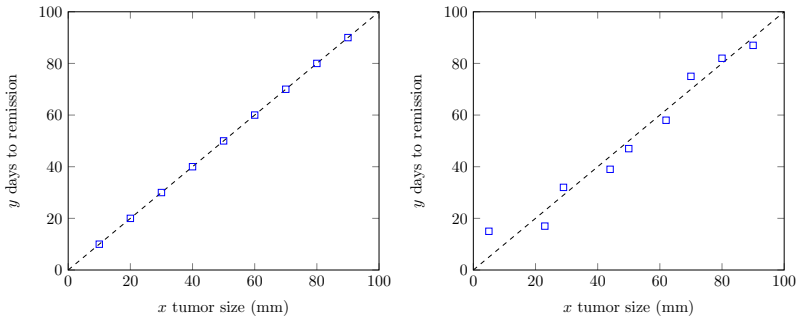

Figure 9.3: In the real world it is unlikely to encounter data that is perfectly predicted by a linear function (left figure). For example, we would not expect the size of a tumor to *perfectly* predict how many days before a patient went into remission. Instead, we often encounter data that can be approximated by a linear function, such as data that is roughly linear with additive noise (right figure).

Now, if we consider the log-likelihood of this model, we obtain

$$\log \mathcal{L}(\mathbf{w}, \mathcal{D}_{\text{trn}}) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{trn}}} \log\left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mathbf{w}^\top\mathbf{x})^2}{2\sigma^2}}\right) \tag{9.18}$$

$$= \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{trn}}} -\log(\sqrt{2\pi}\sigma) - \frac{(y-\mathbf{w}^\top\mathbf{x})^2}{2\sigma^2} \tag{9.19}$$

$$\propto -\sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{trn}}} (y-\mathbf{w}^\top\mathbf{x})^2, \tag{9.20}$$

where as usual we ignore the constant terms that do not depend on the parameter in the final expression. Thus, under this probabilistic model, we can recover that the maximizing the log-likelihood is equivalent to minimizing the MSE.