Introduction to Machine Learning

Irvin Avalos

1 Linear Regression

Suppose we are given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_{2,\ldots,\mathbf{x}_N}\}$ where each $\mathbf{x}_n \in \mathbb{R}^D$ has a corresponding target $t_n \in \mathbb{R}$. The goal is to predict one or more unknown continuous targets t when given a new vector $\mathbf{x}$ of input variables. To accomplish this we say that the prediction for t is given by the function $y(\mathbf{x}, \mathbf{w})$ where $\mathbf{w} \in \mathbb{R}^{D+1}$ is a vector of parameters that can be learned from the training data.

1.1 Simple Linear Regression

The simplest linear regression model is,

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D$$

where the predictions are linear in both weight space and feature space. As such, the predicted values, y, form a $D+1$ dimensional hyperplane.

1.1.1 Basis Functions

However, linear regression does not only have to be used to model data with linear behavior in the feature space. That is, nonlinear data in feature space can be modeled using linear regression via the use of basis functions:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}).$$

Our predictions are then only linear in parameter space, no longer in feature space as we are taking a linear combination of weighted parameters corresponding to the results from inputting data through a fixed nonlinear function, $\phi(\cdot)$.

1.2 Matrix-Vector Notation

Going back to the simple linear regression model, we can rewrite it by extending $\mathbf{x} \in \mathbb{R}^D$ to include $x_0 = 1$ such that the predictions become

$$y(\mathbf{x}, \mathbf{w}) = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = \sum_{d=0}^{D} w_d x_d = \mathbf{w}^T \mathbf{x} = \mathbf{x}^T \mathbf{w}.$$

Assuming our dataset consists of N samples we can write the predictions as

$$\begin{cases} y(\mathbf{x}_1, \mathbf{w}) & = & w_0 + w_1 x_{11} + w_2 x_{12} + \ldots + w_D x_{1D} & = & \mathbf{x}_1^T \mathbf{w} \\ y(\mathbf{x}_2, \mathbf{w}) & = & w_0 + w_1 x_{21} + w_2 x_{22} + \ldots + w_D x_{2D} & = & \mathbf{x}_2^T \mathbf{w} \\ \quad\vdots & & & & \\ y(\mathbf{x}_N, \mathbf{w}) & = & w_0 + w_1 x_{N1} + w_2 x_{N2} + \ldots + w_D x_{ND} & = & \mathbf{x}_N^T \mathbf{w} \end{cases}$$

where $\mathbf{x}_i = [1, x_{i1}, x_{i2}, \ldots, x_{iN}]^T$ for $i \in \{1, 2, \ldots, N\}$. From this we can now introduce the design matrix

$$\mathbf{X} = \begin{bmatrix} \leftarrow & \mathbf{x}_1 & \rightarrow \\ \leftarrow & \mathbf{x}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_N & \rightarrow \end{bmatrix} \in \mathbb{R}^{N \times (D+1)}$$

which allows us to once again rewrite the predictions as

$$\mathbf{Y}(\mathbf{X}, \mathbf{w}) = \mathbf{X}\mathbf{w}, \quad \mathbf{Y} \in \mathbb{R}^N$$

1.3 Error Function

An important question that you should be asking yourself at this point is: *How do we exactly compute the weights* $\mathbf{w}$*?*. As it turns out, this is a very simple task as we accomplish this by minimizing the *error function*

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}(t_n - y(x_n, w))^2$$

which measures the *misfit* between the targets and predictions. Note that $E(w)$ is a quadratic function, then $\frac{\partial}{\partial w}E(w)$ is a linear function and solving $\frac{\partial}{\partial w}E(w) = 0$ yields only one solution, thus simplifying the minimization problem.

- Minimization w.r.t. w_0:

$$\begin{aligned}
\frac{\partial E(w)}{\partial w_0} &= \frac{\partial}{\partial w_0}\frac{1}{2}\sum_{n=1}^{N}(t_n - w_0 - \sum_{d=1}^{D} w_d x_{nd})^2 \\
&= \frac{1}{2}\sum_{n=1}^{N} -2(t_n - w_0 - \sum_{d=1}^{D} w_d x_{nd}) \\
&= \sum_{n=1}^{N} -t_n + \sum_{n=1}^{N} w_0 + \sum_{n=1}^{N}\sum_{d=1}^{D} w_d x_{nd} \\
w_0^\star &= \frac{1}{N}\sum_{n=1}^{N} t_n - \sum_{d=1}^{D} w_d \cdot \frac{1}{N}\sum_{n=1}^{N} x_{nd}
\end{aligned}$$

  $\therefore$ the *bias parameter* measures the difference between the average of the targets and the weighted sum of the average of the inputs.

- Minimization w.r.t. w_k:

$$\frac{\partial E(w)}{\partial w_k} = \frac{\partial}{\partial w_k}\frac{1}{2}\sum_{n=1}^{N}(t_n - w_0 - \sum_{d=1}^{D} w_d x_{nd})^2$$

2 Support Vector Machines

Before we explore the technicalities behind the **maximal margin classifier** we must introduce and define what a **hyperplane** is.

Definition. In D dimensional space, a **hyperplane** is an affine subspace of dimension $D-1$. Geometrically, it divides the D dimensional space into two half-spaces. A hyperplane can be written as

$$b + w_1x_1 + w_2x_2 + \ldots + w_Dx_D = \mathbf{w}^T\mathbf{x} + b = 0,$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, $\mathbf{w} = (w_1, w_2, \ldots, w_D)^T$, and $b \in \mathbb{R}$. From this, we can say that:

- $\mathbf{x}$ lies to the left of the hyperplane if
$$\mathbf{w}^T\mathbf{x} + b < 0.$$
- $\mathbf{x}$ lies to the right of the hyperplane if
$$\mathbf{w}^T\mathbf{x} + b > 0.$$
- Otherwise $\mathbf{x}$ is said to lie on the hyperplane when
$$\mathbf{w}^T\mathbf{x} + b = 0.$$

Using this new knowledge suppose that we are given the dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ where $\mathbf{x}_i \in \mathbb{R}^D$. Every data point will fall into one of two *classes*, i.e., we will assign to each $\mathbf{x}_i$ a term $y_i \in \{-1, 1\}$. Finally, we will include a test observation $\mathbf{x}^\star = (x_1^\star, x_2^\star, \ldots, x_D^\star)^T$. Our goal now is to create a linear classifier that will correctly classify this test observation using its features. To do this we will use the idea of a *separating hyperplane*, i.e., we will create a hyperplane such that each point falls into one of the two sides from this hyperplane. Therefore, we can assign a label to each observation using the function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ such that

$$f(\mathbf{x}_i) = \begin{cases} \mathbf{w}^T\mathbf{x}_i + b > 0 & \text{if } y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b < 0 & \text{if } y_i = -1 \end{cases}.$$

- Note that to correctly classify every point $i = 1, 2, \ldots, N$ we require,
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0.$$
- Additionally, the magnitude $|f(\mathbf{x})|$ indicates how far a point lies from the *decision boundary* (i.e., hyperplane). Thus, points with values close to zero lie near this boundary.

A central challenge is that when data is linearly separable, there exist infinitely many hyperplanes that satisfy our linear classifier's constraints. This motivates the need to find the **maximal margin classifier**. Conceptually, this classifier depends on the *margin*, defined as the smallest perpendicular distance from the separating hyperplane to any point. More formally, to obtain the maximal margin hyperplane we solve

$$\begin{aligned} \min_{b, \mathbf{w}, M} \quad & M \\ \text{s.t.} \quad & \sum_{i=1}^{D} w_i^2 = 1, \\ & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq M \ \forall \ i = 1, 2, \ldots, N \end{aligned}$$

where M denotes the margin. The constraint $\sum_{i=1}^{D} w_i^2 = 1$ normalizes the weight vector such that M represents the geometric margin, and the optimization looks for the hyperplane that maximizes this value.