Linear Regression

Simple linear regression models the relationship between a scalar dependent variable y and a d-dimensional independent variable $\mathbf{x} \in \mathbb{R}^d$ as a linear function $f : \mathbb{R}^d \to \mathbb{R}$. Formally, this relationship is parametrized by a vector of weights $\mathbf{w} = (w_0, ..., w_d)^T$ which define a line in $\mathbb{R}^d$:

$$f(\mathbf{x}) = w_0 + w_1 x_1 + ... + w_d x_d \qquad\qquad y(\mathbf{x}) = f(\mathbf{x}) + \varepsilon(\mathbf{x}). \tag{1}$$

Here $f(\mathbf{x})$ and $\varepsilon(\mathbf{x})$ are the prediction and error of the model at $\mathbf{x}$, respectively, and $y(\mathbf{x})$ is the target function, *i.e.* the true value of the system at $\mathbf{x}$. Typically, we assume an "augmented" $\mathbf{x}$, which allows us to define f in matrix notation as follows.

$$\mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \qquad\qquad \Longrightarrow \qquad\qquad f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} \tag{2}$$

As of now, the only condition we have placed on $\mathbf{w}$ is that it defines a line in $\mathbb{R}^d$. Linear regression relies on a dataset of observations $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, to optimize $\mathbf{w}$ in linear function space. By making the following definitions,

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \qquad\qquad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{nd} \end{pmatrix} \tag{3}$$

we can also define an error vector, which contains the error in the model at each observed data point.

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon(\mathbf{x}_1) \\ \vdots \\ \varepsilon(\mathbf{x}_n) \end{pmatrix} = \mathbf{y} - \mathbf{X}\mathbf{w} \tag{4}$$

Intuitively, we desire to minimize the magnitude of $\boldsymbol{\varepsilon}$ with respect to $\mathbf{w}$. However, in reality some elements of $\boldsymbol{\varepsilon}$ may differ in sign; minimizing the sum over all the elements runs the risk of building error cancellation into the model. Instead we use an *ordinary least-squares* (OLS) treatment, which minimizes the strictly-positive squared Euclidean norm of $\boldsymbol{\varepsilon}$. This is called the cost function, $C(\mathbf{w})$.

$$C(\mathbf{w}) = ||\boldsymbol{\varepsilon}||_2^2 = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 \qquad\qquad \hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}(C(\mathbf{w})) \tag{5}$$

When the columns of $\mathbf{X}$ are linearly independent, this minimization has the following unique solution.

$$\mathbf{X}^T\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^T\mathbf{y} \qquad\qquad \Longrightarrow \qquad\qquad \hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{y}. \tag{6}$$

Now we have a model for the relationship between $\mathbf{x}$ and y which can make a prediction at an unknown (augmented) point $\mathbf{x}^*$ as follows:

$$f(\mathbf{x}^*) = (\mathbf{x}^*)^T\hat{\mathbf{w}} \tag{7}$$

Ridge Regression

Linear regression relies on the assumption of linear independence in the columns of $\mathbf{X}$, but for real-world datasets this is rarely the case. When $\mathbf{X}$ contains exact linear dependencies, Equation 6 has singularities in the $(\mathbf{X}^T\mathbf{X})^{-1}$ term and thus no solution. When $\mathbf{X}$ has near-linear dependencies, the coefficients $\hat{\mathbf{w}}$ blow up in the OLS procedure, rendering model overly sensitive to noise. This problem is termed *multicollinearity* and can be addressed by ridge regression, which adds an $\mathcal{L}_2$ regularization to the OLS cost function to penalize large values in $\mathbf{w}$.

$$C(\mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda||\mathbf{w}||_2^2 \qquad \hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}(C(\mathbf{w})) \tag{8}$$

where λ is a regularization parameter. This can be thought of as introducing a constraint on the size of $\mathbf{w}$ into the OLS minimization. In this case, solving for $\hat{\mathbf{w}}$ yields the following solution.

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{9}$$

The regularization term stabilizes the inverse by bounding its smallest eigenvalues away from zero. Once $\hat{\mathbf{w}}$ optimized, the prediction at an unseen point $\mathbf{x}^*$ is again given by

$$f(\mathbf{x}^*) = \hat{\mathbf{w}}^T\mathbf{x}^*. \tag{10}$$

Kernel Ridge Regression

Ridge regression is one of the simplest algorithms that can be "kernelized" to achieve a nonlinear treatment. To use kernel methods, we define two key concepts: a nonlinear feature map $\phi(\mathbf{x})$ that maps $\mathbf{x}$ from the input space to an abstract feature space $\mathcal{F}$, and a kernel function κ that corresponds to the inner product in $\mathcal{F}$. Formally,

$$\phi : \mathbb{R}^d \to \mathcal{F} \qquad \kappa(\mathbf{x}, \mathbf{x}') = \langle\phi(\mathbf{x}), \phi(\mathbf{x}')\rangle. \tag{11}$$

To move from ridge regression to kernel ridge regression (KRR), we replace all instances of $\mathbf{x}$ with its corresponding feature vector $\phi(\mathbf{x})$. This means that KRR makes a prediction at $\mathbf{x}^*$ as follows.

$$f(\mathbf{x}^*) = [\phi(\mathbf{x}^*)]^T\hat{\mathbf{w}}. \tag{12}$$

Defining $\Phi_i \equiv [\phi(\mathbf{x}_i)]$, the design matrix $\mathbf{X}$ becomes

$$\mathbf{\Phi} = \begin{pmatrix} \Phi_1^T \\ \vdots \\ \Phi_n^T \end{pmatrix} \tag{13}$$

and we move from ridge regression to kernel ridge regression by replacing $\mathbf{X}$ in Equation 9 with $\mathbf{\Phi}$.

$$\hat{\mathbf{w}} = \left(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I}_n\right)^{-1}\mathbf{\Phi}^T\mathbf{y} \tag{14}$$

The key to kernel methods is the so-called "kernel trick": if we can rearrange these equations to only work with the inner products of feature vectors, then we can perform a nonlinear regression in $\mathcal{F}$ without explicitly defining the feature map ϕ (or the feature space itself). We'll start by using the Woodbury matrix identity[1] to rearrange Equation 14.

$$\hat{\mathbf{w}} = \mathbf{\Phi}^T\left(\mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{y} \tag{15}$$

Notice that the term $\mathbf{\Phi}\mathbf{\Phi}^T$ is a matrix of inner products in $\mathcal{F}$.

$$[\mathbf{\Phi}\mathbf{\Phi}^T]_{ij} = \Phi_i^T \Phi_j = \langle \Phi_i, \Phi_j \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{16}$$

Since this matrix plays an important role in kernel methods, we define $\mathbf{K} \equiv \mathbf{\Phi}\mathbf{\Phi}^T$ as the kernel matrix of inner products between the observed feature vectors. With this definition, the equation for $\hat{\mathbf{w}}$ becomes

$$\hat{\mathbf{w}} = \mathbf{\Phi}^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \tag{17}$$

Now only the $\mathbf{\Phi}^T$ term on the right side of the equality in Equation 17 prevents us from working exclusively with inner products. However, our end goal is not to solve for $\hat{\mathbf{w}}$ in itself, but to use it to make a prediction via Equation 12. So let's think about what happens when we plug Equation 17 into the KRR model (Equation 12), which makes a prediction at an unseen point $\mathbf{x}^*$.

$$f(\mathbf{x}^*) = [\phi(\mathbf{x}^*)]^T \mathbf{\Phi}^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \tag{18}$$

Notice that $[\phi(\mathbf{x}^*)]^T \mathbf{\Phi}^T$ is a row vector of inner products between the feature vector at $\mathbf{x}^*$ and the feature vectors of observations. We will call this $[\mathbf{k}(\mathbf{x}^*)]^T$, where

$$\mathbf{k}(\mathbf{x}^*) \equiv \begin{pmatrix} \langle \phi(\mathbf{x}^*), \Phi_1 \rangle \\ \vdots \\ \langle \phi(\mathbf{x}^*), \Phi_n \rangle \end{pmatrix} \tag{19}$$

We have now arrived at a form of Equation 12 which involves only inner products, *i.e.*, kernel functions.

$$f(\mathbf{x}^*) = [\mathbf{k}(\mathbf{x}^*)]^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \tag{20}$$

Perhaps a more intuitive way of thinking about these equations is to define $\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$:

$$f(\mathbf{x}^*) = \mathbf{k}^T \alpha = \sum_{i=1}^{n} k_i \alpha_i = \sum_{i=1}^{n} \alpha_i \kappa(\mathbf{x}^*, \mathbf{x}_i). \tag{21}$$

The penalty term λ is considered a *hyperparameter* of the algorithm, because neither defined nor determined in the equations given. Therefore the KRR algorithm can be said to depend parametrically on the chosen value of λ. In practice, the value of λ is inferred from the observations; cross-validation is one common way to do this and is discussed in the next section.

Cross-Validation

Leave-one-out cross-validation (CV) is a technique for optimizing hyperparameters in machine-learning. The dataset of observed points ($\mathbf{X}$) is partitioned into N_X bins of size one; thus each bin has one distinct sample. A validation set is formed by taking the first of these bins, while a training set is formed from the rest. The model is optimized on the training set, and then optimal hyperparameters are determined by minimizing the error on the singleton validation set. This procedure is repeated for each bin. At the end, there is a set of N_X optimal hyperparameters obtained in this manner. We take as our final hyperparameters the median of this set. This algorithm can be easily generalized to bins of arbitrary size, and is readily available in the SciKit-Learn software package for Python.

References

[1] W. W. Hager. Updating the inverse of a matrix. *SIAM Rev.*, 31:221.