Lecture 2: Regression, regularization and the kernel trick

Bingqing Cheng

January 2020

Outline

Last lecture we covered the fundamentals of machine learning models. In this lecture, we move on to an important type of supervised machine learning task that was mentioned, i.e. building *regression* models. Regression involves building a model that tries to predict the continuous value of a particular label for a new data point. We start from a simple task, fitting data with linear models. Formally, this goes under the name of linear regression. We then move on to more sophisticated methods that fit the data with polynomials of different order, which are called polynomial regression.

After that, we will learn about regularization. We will be concerned with two classes of regularizers L2-regularization which is often called *Ridge-Regression* and L1-regularization which goes under the name *LASSO*.

Lastly, we will learn about the idea behind the kernel trick and compare different kernel functions.

1 Linear regression

1.1 Linear model

Consider a simple linear regression model

$$y = w_0 + \sum_{j=1,2,..,d} w_j x^{(j)} + \epsilon \tag{1}$$

where y is the dependent variable, and $\mathbf{x} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(d)} \end{bmatrix}$ is the d-dimensional independent or input variable. The terms w_i are the parameters of the model. The parameter w_0 is termed as intercept term or the bias term.

It is convenient to have a dummy input variable $x^0 \equiv 1$, and one can write

$$y = \sum_{j=0,1,..,d} w_j x^{(j)} + \epsilon. \tag{2}$$

The parameters $\mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_d \end{bmatrix}^T$ are usually called regression coefficients. The linear model in Eqn. (1) can thus be re-written in a more compact matrix form

$$y = \mathbf{x}\mathbf{w} + \epsilon. \tag{3}$$

This can be easily extended to the case where N pairs of observations $\{(y_i, \mathbf{x}_i)\}$ are considered, i.e.

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon, \tag{4}$$

where $\mathbf{y} \equiv (y_1, \cdots, y_N)^T$ and the design matrix $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_N)^T$.

The noise term ϵ accounts for the failure of data to lie on the straight line and represents the difference between the true underlying value and observed value of y. This noise can come from many different origins, for example, the stochastic error during the measurements, the hidden variables that the model does not include, the functional form of the model is not complete, etc. To make the regression tractable, one often assumes that ϵ is an independent and identically distributed random variable with mean zero and constant variance σ^2. In many cases, ϵ is assumed to be normally distributed $N(0, \sigma)$.

1.2 Fitting a linear model

If all the model coefficients ($\mathbf{w}$, σ) are known, the model is completely described and one can make a prediction of the oberservable y. However, in practice, we have exactly the reverse: The parameters $\mathbf{w}$ are generally unknown and the noise σ is difficult to measure in practice. In order to estimate the values of these parameters, N pairs of observations $\{(y_i, \mathbf{x}_i)\}$ are collected and are used to determine these unknown parameters. Various methods of estimation can be used to determine the estimates of the parameters. Among them, the method of *least squares* is a popular method of estimation.

Least squares For ordinary least square regression (no regularization), we minimize the *data dependent error* $E_D(\mathbf{w})$, which takes the form of the square loss cost function in this case:

$$E_D(\mathbf{w}) = \frac{1}{2}||\mathbf{X}\mathbf{w} - \mathbf{y}||^2, \tag{5}$$

or equivalently, in component form,

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w} \cdot \mathbf{X}^{(i)})^2. \tag{6}$$

This means we are trying to find

$$\min_{\mathbf{w}\in\mathcal{R}^d} ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2 = \min_{\mathbf{w}\in\mathcal{R}^d} (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}), \tag{7}$$

or equivalently, in component form,

$$\min_{\mathbf{w}\in\mathcal{R}^d} \sum_{i=1}^{n}(y_i - \mathbf{w} \cdot \mathbf{X}^{(i)})^2. \tag{8}$$

If $\text{rank}(\mathbf{X}) = d$, namely, the feature predictors $\mathbf{X}_{:,1}, \cdots \mathbf{X}_{:,d}$ are linearly independent, then there exists unique solution to this problem:

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y} \tag{9}$$

Is not, one can use pseudo inverse and write

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{10}$$

1.3 Regularization

In the least square fit, it is clear that the uniqueness of the solution is only guaranteed when $\text{rank}(\mathbf{X}) \geq d$. But even so we still may not want to use least squares if d is moderately close to n, because of the tendency to overfit. One way to deal with this is to *regularize*.

We will be concerned with two classes of regularizers: *L2-regularization* which is often called *Ridge-Regression*, and *L1-regularization* which goes under the name *LASSO* (and is closely related to *Compressed Sensing*.

In all cases, the idea is to formulate a *loss function* by taking the sum of the *data dependent error* $E_D(\mathbf{w})$ and a *regularization penalty*.

Ridge Regression In *Ridge Regression*, the regularization penalty is taken to be the L2-norm of the parameters

$$E_{ridge}(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = \frac{\lambda}{2}\sum_{j=1}^{d} w_j w_j. \tag{11}$$

Thus, the model is fit by minimizing the sum of the data dependent error $E_D(\mathbf{w})$ and the regularization term $E_{ridge}(\mathbf{w})$:

$$\mathbf{w}_{ridge}(\lambda) = \underset{\mathbf{w}\in\mathcal{R}^d}{\text{argmin}} \frac{1}{2}||\mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \frac{\lambda}{2}||\mathbf{w}||^2. \tag{12}$$

Notice that the parameter λ controls how much we weigh the fit and regularization term.

To solve $\mathbf{w}$, one way is to take the derivative of Eqn. (11) with respect to $\mathbf{w}$ and set it to zero, and get

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{y}, \tag{13}$$

where $\mathbf{I}$ is the d-rank identity matrix.

LASSO We will also be interested in the case where the penalty is the L1-norm of the parameters (sum of absolute values of parameters). This is called LASSO.

$$E_{LASSO}(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}|| = \frac{\lambda}{2}\sum_{j=1}^{d} |w_j|.$$

In this case,

$$\mathbf{w}_{LASSO}(\lambda) = \underset{\mathbf{w}\in\mathcal{R}^d}{\operatorname{argmin}} \frac{1}{2}||\mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \frac{\lambda}{2}||\mathbf{w}||.$$

LASSO tends to give sparse solutions.

2 More advanced regression methods

2.1 A brief recap on linear regression

In linear regression, the dependent variable y is assumed to be related to the input by $\mathbf{x} = \begin{bmatrix} x^0 & x^1 & \cdots & x^d \end{bmatrix}$:

$$y = \mathbf{x}\mathbf{w} + \epsilon, \tag{14}$$

where regression coefficients $\mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_d \end{bmatrix}^T$, and ϵ is the noise term. In a simple one-dimensional case, we have, in component form

$$y = w_0 + w_1 x + \epsilon. \tag{15}$$

2.2 Polynomial regression

In many cases, the linear model introduced in Eqn. (15) is not ideal. One can easily think of a number of examples that embrace a high degree of non-linear behavior, such as the production of chemical synthesis in terms of temperature, reaction rate as a function of concentrations of reactants, and monthly precipitation as a year goes by.

As a simple extension, a simple linear regression can be transformed by constructing polynomial features from the coefficients. Instead of writing the naive form of Eqn. (15), one can write the features in polynomials, so that the model looks like this (if we expand to the 4th order of x):

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + \epsilon. \tag{16}$$

An important observation is that this is still a linear model: to see this, imagine creating a new set of features

$$\phi(x) = \begin{bmatrix} \phi^{(0)} & \phi^{(1)} & \phi^{(2)} & \phi^{(3)} & \phi^{(4)} \end{bmatrix} = \begin{bmatrix} 1 & x^1 & x^2 & x^3 & x^4 \end{bmatrix}, \tag{17}$$

and now we can re-caste Eqn. (16) into the familiar linear form

$$y = w_0\phi^{(0)} + w_1\phi^{(1)} + w_2\phi^{(2)} + w_3\phi^{(3)} + w_4\phi^{(4)} + \epsilon, \tag{18}$$

or in a compact matrix notation

$$y = \phi(\mathbf{x})\mathbf{w} + \epsilon, \tag{19}$$

where $\phi(\mathbf{x})\mathbf{w}$ can be considered as a *model*, and ϵ is the part that the model cannot describe.

We see that the resulting polynomial regression is in the same class of linear models we considered above (i.e. the model is linear in ϕ) and can be solved by the same techniques. By considering linear fits within a higher-dimensional space built with these basis functions, the model has the flexibility to fit a much broader range of data.

2.3 Kernel regression

2.3.1 Dual representation

In polynomial regression, the features $\phi(\mathbf{x})$ are polynomial functions of different orders. Instead of using polynomials, we can take $\mathbf{x}$, and transform it into any fancy feature space $\phi(\mathbf{x})$, and still write the model as

$$f(\mathbf{x}) = \phi(\mathbf{x})\mathbf{w}, \tag{20}$$

and predict $\mathbf{w}$ by minimizing a loss function such as (if one uses the L2 ridge regression)

$$\underset{\mathbf{w}\in\mathcal{R}^d}{\operatorname{argmin}} \frac{1}{2}||\mathbf{\Phi w} - \mathbf{y}||^2 + \frac{\lambda}{2}||\mathbf{w}||^2, \tag{21}$$

where $\mathbf{\Phi} = (\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_N))^T$.

Now, what if we write $\mathbf{w}$ using the basis of features $\phi(\mathbf{x})$? This can be done because $\mathbf{w}$ can be solved as a matrix product

$$\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{\Phi y}. \tag{22}$$

For benefits that will become obvious in a moment, we do not explicitly solve for $\mathbf{w}$ but instead write down

$$\mathbf{w} = -\frac{1}{\lambda}\sum_i \alpha_i\phi(\mathbf{x}_i), \tag{23}$$

where α is the matrix that contains all the coefficients for expanding $\mathbf{w}$ in the basis of $\phi(\mathbf{x})$. Before diving into the details of that, we can already notice something very interesting happening here: $\mathbf{w}$ is given by a sum of the input vectors x_i, weighted by α_i/λ. If we are inclined, we can avoid explicitly computing $\mathbf{w}$, and predict a new point $\mathbf{x}$ directly from the data as

$$f(\mathbf{x}) = \phi(\mathbf{x})\mathbf{w} = -\frac{1}{\lambda}\sum_i \alpha_i\phi(\mathbf{x}_i)\phi(\mathbf{x}). \tag{24}$$

let $k(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i)\phi(\mathbf{x})$ be the *kernel function*. For now, just think of this as a change of notation. Using this, we can again write the ridge regression predictions as

$$f(\mathbf{x}) = \phi(\mathbf{x})\mathbf{w} = -\frac{1}{\lambda}\sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}). \tag{25}$$

Now things are really get interesting: we do not need to evaluate the basis functions $\phi(\mathbf{x})$ explicitly, and all we really need is the inner product of $\mathbf{x}$ with

each of the training elements $\mathbf{x}_i$. In the same way, we can write the data loss function as

$$E_D(\alpha) = \frac{1}{2}\sum_{i=1}^{n}(y_i + \frac{1}{\lambda}\sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}))^2, \tag{26}$$

and the regression penalty as

$$E_{ridge}(\alpha) = \frac{1}{2\lambda}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{27}$$

which, again, only rely on the inner product. One can then find α by minimizing the combined loss function $E_D(\alpha) + E_{ridge}(\alpha)$. After some manipulations, one can show that the solution is

$$\alpha = -\left(\mathbf{K} + \lambda\mathbf{I}\right)^{-1}\lambda\mathbf{y}, \tag{28}$$

where $\mathbf{K}$ is the *kernel matrix* with each component $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$.

More importantly, whenever a new data point $\mathbf{x}$ comes in, the prediction can be expressed as

$$f(\mathbf{x}) = \mathbf{k}\left(\mathbf{K} + \lambda\mathbf{I}\right)^{-1}\mathbf{y}, \tag{29}$$

where $\mathbf{k} = (k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_n, \mathbf{x}))$ is the vector of inner products between the data and the new point $\mathbf{x}$.

2.3.2 Kernel trick

So we can work completely with inner products, rather than the vectors themselves. So what? One way of looking at things is that we can implicitly use basis expansions, and just compute the kernel functions $k(\mathbf{x}_i, \mathbf{x}_j)$. This is called the *kernel trick*: getting around the computational expense in computing large basis expansions by directly computing kernel functions. Notice, however, that the kernel trick changes nothing compared with the regression in the feature space: we get exactly the same predictions as if we computed the basis expansion explicitly, and used traditional linear methods. We just compute the predictions in a different way.

In fact, one does not even need to have basis functions ϕ in mind when using the kernel trick. As long as the the kernel functions $k(\mathbf{x}_i, \mathbf{x}_j)$ fulfill certain rules (most importantly ensuring that the kernel matrix is positive semi-definite), one can just use it in the kernel regression discussed above.

2.3.3 Examples of kernel functions

Some commonly used kernel functions are:

Linear kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i\mathbf{x}_j$$

Polynomials of degree m

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \mathbf{x}_j)^m$$

Polynomials of up to degree m

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \mathbf{x}_j + 1)^m$$

Gaussian/Radial kernels

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right)$$

Sigmoid

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\zeta \mathbf{x}_i \mathbf{x}_j + \mu)$$