

Project-1 of “introduction to Statistical Learning and Machine Learning”

Yanwei Fu

September 25, 2018

1 Linear Regression and Nonlinear Bases

In class we discuss fitting a linear regression model by minimizing the squared error. This classic model is the simplest version of many of the more complicated models. However, it typically performs very poorly in practice. One of the reasons it performs poorly is that it assumes that the target y_i is a linear function of the features x_i , if with *an intercept of zero*. This drawback can be addressed by adding a bias variable and using nonlinear bases (although nonlinear bases may increase to overfitting).

In this question, you will start with a data set where least squares performs poorly. You will then explore how adding a bias variable and using nonlinear (polynomial) bases can drastically improve the performance. You will also explore how the complexity of a basis affects both the training error and the test error. In the final part of the question, it will be up to you to design a basis with better performance than polynomial bases.

You can use either Matlab, Python, or R codes. However, the sample codes are provided by Matlab.

1.1 Adding a Bias Variable

Download codes from the course webpage, and start Matlab in a directory containing the extracted files. If you run the script `example_basis.m`, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Report the training error.
4. Report the test error (on a dataset not used for training).
5. Draw a figure showing the training data and what the linear model looks like.

Unfortunately, this is an awful model of the data. The average squared training error on the data set is over 28000 (as is the test error), and the figure produced by the demo confirms that the predictions are usually nowhere near the training data:



The y-intercept of this data is clearly not zero (it looks like it's closer to 200), so we should expect to improve performance by adding a bias variable, so that our model is

$$y_i = w^T x_i + \beta$$

instead of

$$y_i = w^T x_i$$

Write a new function, *leastSquaresBias*, that has the same *input/model/predict* format as the *leastSquares* function, but that includes a bias variable w_0 .

In the report, you should explain the critical sections of your new function, the updated plot, and the updated training/test error.


1.2 Polynomial Basis

Adding a bias variable improves the prediction substantially, but the model is still problematic because the target seems to be is a *non-linear* function of

the input. Write a new function, `leastSquaresBasis(x,y,deg)`, that takes a data vector x (i.e., assuming we only have one feature) and the polynomial order deg . The function should perform a least squares fit based on a matrix X_{poly} where each of its rows contains the values $(x_i)^j$ for $j = 0$ up to deg . E.g., `leastSquaresBasis(x,y,3)` should form the matrix

$$X_{poly} = \begin{bmatrix} 1 & x_1 & (x_1)^2 & (x_1)^3 \\ 1 & x_2 & (x_2)^2 & (x_2)^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & (x_n)^2 & (x_n)^3 \end{bmatrix}$$

and fit a least squares model based on it.

In the report, you should explain the critical sections `leastSquaresBasis(x,y,3)` function of and report the training and test error for $deg = 0$ through $deg = 10$. Explain the effect of deg on the training error and on the test error. 

2 Regularization


Regularization is a core idea in machine learning: regularization can significantly reduce overfitting when we try very complicated models. In this question, you'll implement a simple L2-regularized least squares model and see the dramatic performance improvement it can bring on test data.

2.1 Preprocessing – Data standardization

The goal of standardization is to make the d input attributes of the data comparable. If we were predicting the price or real estate in Shanghai using as input attributes the number of rooms and the price of the property 3 years ago ($d = 2$), then we would run into trouble because the number of rooms is typically in the range 1 to 4, while the price 3 years ago is probably in the range 500,000 to 5,000,000. We need to place these two attributes on the same scale so that applying the same regularizer to their weights makes sense.

To achieve this, we standardize the input data by ensuring it has zero mean and unit variance. We can do this by computing

$$x_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j}$$

where $\bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$ is empirical mean of the j 'th attribute, and $\sigma_j^2 = \frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2$ is the empirical variance. Here, $i = 1, \dots, n$ denotes the index over the data and $j = 1, \dots, d$ is the index over input attributes. 

The subtraction of the mean is not necessary for datasets that are already known to be zero mean, such as acoustic time series. Likewise, for gray-scale images we know that each attribute (pixel) has the same range of possible values, so we can skip the variance normalization. If the attributes do not have the same range of values, then the variance rescaling can prove to be very useful.

For example if the first attribute is the number of people who entered a building, the second attribute the temperature of the building, and the third attribute the number of rooms in the building, then these three numbers are likely not to be in the same scale. Variance rescaling places them on a reasonable comparative scale.

The parameters \bar{x}_j and σ_j^2 should be considered part of the model. So at test time, we should center with respect to \bar{x}_j and scale with respect to σ_j , rather than centering and scaling the test set with respect to the mean and standard deviation of the test set. To see why, imagine the test set consisted of a single example; then scaling by the test set standard deviation, instead of the value derived from the training set, would clearly be wrong, since the standard deviation of a single data point is not defined. All this will be made more clear in the following section.

2.2 Load and standardize the data

Download the prostate cancer dataset from the course website. In this prostate cancer study 9 variables – including *age*, *log weight*, *log cancer volume*, etc. – were measured for 97 patients. [We will now construct a model to predict the 9th variable a linear combination of the other 8.](#) A description of this dataset appears in the textbook of Hastie *et al*:

“The data for this example come from a study by Stamey et al. (1989) that examined the correlation between the level of prostate specific antigen (PSA) and a number of clinical measures, in 97 men who were about to receive a radical prostatectomy. The goal is to predict the log of PSA (lpsa) from a number of measurements including log cancer volume (lcavol), log prostate weight lweight, age, log of benign prostatic hyperplasia amount lbph, seminal vesicle invasion svi, log of capsular penetration lcp, Gleason score gleason, and percent of Gleason scores 4 or 5 pgg45. ”

1. First, load the data and split it into a response vector (y) and a matrix of attributes (X);
2. **Second, randomly shuffle the orders of patients in the table** and then choose the first 50 patients as the training data. The remaining patients will be the test data.
3. Set both variables (i.e. the training set of X and y) to have zero mean and standardize the input variables to have unit variance.

Note that in the training step, we will learn the bias term (i.e. intercept term β_0) separately, i.e. not regularize bias term. Recall that the purpose of regularization is to get rid of unwanted input attributes. Mathematically, what we are saying is that the bias term will be computed separately as follows,

$$\beta_0 = \bar{y} - \bar{x}^T \hat{\beta}$$



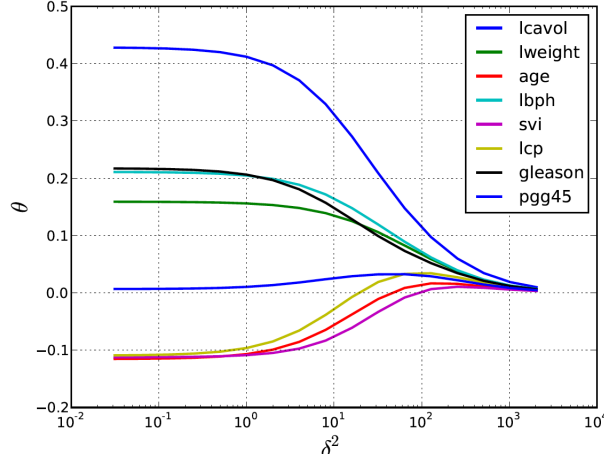


Figure 1: Regularization path for ridge regression.

where \bar{y} is the mean of the elements of the training data vector y and x^T is the vector of 8 means for the input attributes. Note that in this case the 8-dimensional parameter vector $\hat{\beta}$ includes all the parameters other than the bias term that have been learned with ridge regression. That is, we first learn $\hat{\beta}$ using standardized data and then proceed to learn β_0 .

When we encounter a new input x^* in the test set, we need to standardize it before making a prediction. The actual prediction should be:

$$\hat{y} = \bar{y} + \sum_{j=1}^8 \frac{x_j^* - \bar{x}_j}{\sigma_j} \hat{\theta}_j$$

where x_j and σ_j are the mean and standard deviation of the j -th attribute obtained from the training data. One reason for standardizing the inputs is that we want them to be comparable.

Had we had an input much bigger than the other, we would have wanted to apply a different regularizer to it. By standardizing the inputs first, we only need a single scalar regularization coefficient σ^2 .

2.3 Ridge regression

We will now construct a model using ridge regression to predict the 9th variable as a linear combination of the other 8.

The ridge method is a regularized version of least squares, with objective function:

$$\min_{\theta \in \mathbb{R}^d} \|y - X\theta\|_2^2 + \delta^2 \|\theta\|_2^2$$

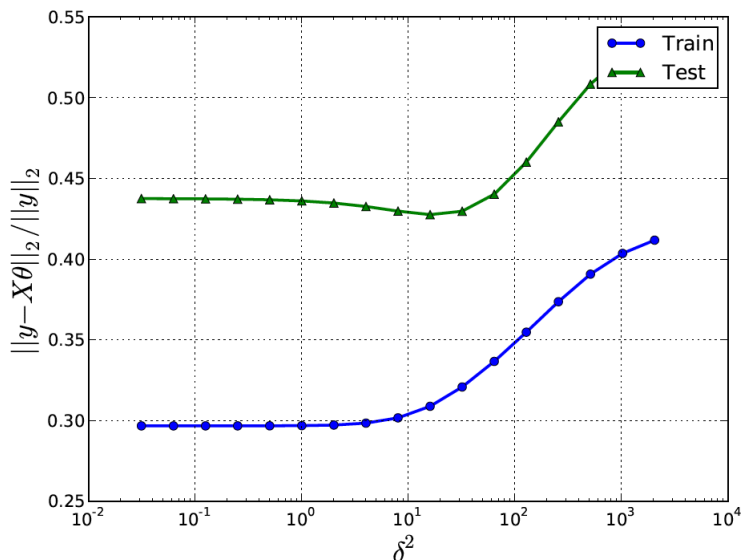


Figure 2: Relative error of the ridge estimator against regularization parameter δ^2 .

Here δ is a scalar, the input matrix $X \in \mathbb{R}^{n \times d}$ and the output vector $y \in \mathbb{R}^n$ and the parameter vector $\theta \in \mathbb{R}^d$.

1. Write code for ridge regression starting from the following skeleton:

```
function ridge(X, y, d2)
    ???
    return theta
```

Compute the ridge regression solutions for a range of regularizers (δ^2). Plot the values of each θ in the y-axis against δ^2 in the x-axis. This set of plotted values is known as a regularization path. Your plot should look like Figure 1. [Hand in your version of this plot, along with the code you used to generate it.](#)

2. For each computed value of θ , compute the train and test error. Remember, you will have to standardize your test data with the same means and standard deviations as above (\bar{X} , Xstd , \bar{y}) before you can make a prediction and compute your test error.
3. Choose a value of δ^2 using cross-validation. What is this value? Show all your intermediate cross-validation steps and the criterion you used to choose δ^2 . Plot the train and test errors as a function of δ^2 . Your plot

should look like Figure 2. Hand in your version of this plot, along with the code used to generate it.

