Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 58

Hello everyone, this is lecture three of week twelve. Recall that in the past two lectures, we learned the basics of supervised learning. We have a labeled dataset given to us, and we assume that the label y is related to the feature vector x by y = f(x). We minimize a given loss function for a given test vector. The task is to estimate $f(x_0)$ for a given test vector in such a way that it minimizes the given loss function.

Then we went to a special case when f is assumed to be linear, which is a simplifying assumption, and the loss function is given to be the squared loss function. In that case, we had the OLS, the ordinary least squares example. We discussed the OLS method and gave the closed-form solution. A similar method was also proposed, which is called ridge regression. In ridge regression, the objective is only slightly different. In OLS, we just minimize the sum of squared loss for each of the data points. In ridge regression, instead of just minimizing this, we are minimizing it with respect to some constraint on the parameters $\beta_0, \beta_1, ..., \beta_p$. The constraint is that the sum of squares of $\beta_1, \beta_2, ..., \beta_p$ should be less than or equal to some quantity s. That is the constrained optimization way of writing the ridge regression problem. There is also an unconstrained optimization way of writing the ridge regression problem.

We discussed both of these, and you can see them on the screen. We have two boxes. The box on the top refers to the constrained optimization version of the ridge regression problem, and the box on the bottom refers to the unconstrained optimization version of the ridge regression problem. This was one reason why I chose this example, this method ridge regression to be implemented, because I could implement both constrained optimization algorithms and unconstrained optimization algorithms that we discussed throughout this course.

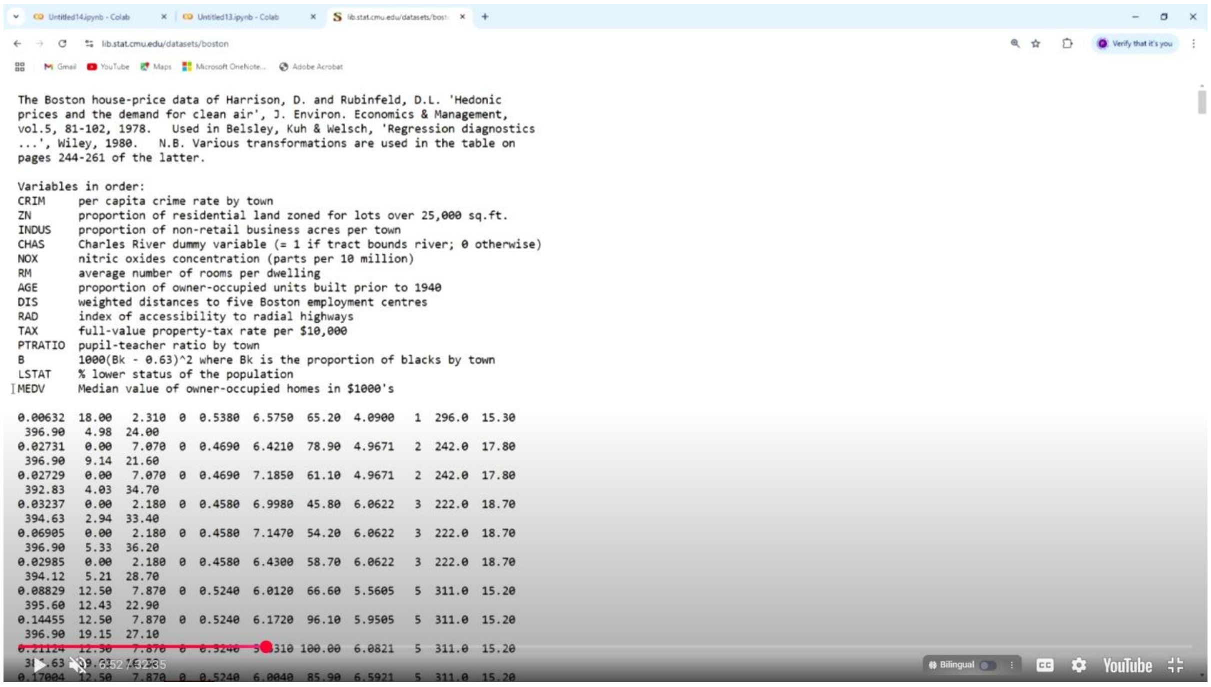
One of the questions that is still pending to be answered is how do we choose these parameters s and μ. Suppose we are given the label data (x_1, y_1), (x_2, y_2) that we discussed a few classes back.

This data τ is given; we cannot just go ahead and solve the problem like we did in OLS because we need the parameter s if we are solving the constrained version, or μ if we are solving the unconstrained version. That is a question which needs to be answered. In practice, what we do is try different values of s and choose the value for which the testing error is the minimum. That is exactly what we do. We will see that when we write the code for ridge regression.

Another important question is, where is the data? What data are we going to use to solve this ridge regression problem? The dataset that we are going to use is called the Boston housing data from 1970. It is a housing dataset. This is publicly available. Let me show the data to you.

This code was from last week. This data is actually available on Carnegie Mellon University's website: lib.stat.cmu.edu/datasets/Boston. This is called the Boston house price data. Let me explain what this data actually means. This is a labeled dataset. The first thirteen variables you have here are CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT. These thirteen parameters are the independent variables, the feature vector. The last one, MEDV, the median value of the owner-occupied homes in thousands of dollars, is the dependent variable. Given these variables, CRIM, ZN, ..., LSTAT, we need to find the median value of owner-occupied homes. The training data is given to us. There are many data points here. Each data point has fourteen values. The first thirteen are the independent variables, and the fourteenth is the dependent variable.

(Refer Slide Time 7:00)



```
The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

Variables in order:
CRIM     per capita crime rate by town
ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS    proportion of non-retail business acres per town
CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX      nitric oxides concentration (parts per 10 million)
RM       average number of rooms per dwelling
AGE      proportion of owner-occupied units built prior to 1940
DIS      weighted distances to five Boston employment centres
RAD      index of accessibility to radial highways
TAX      full-value property-tax rate per $10,000
PTRATIO  pupil-teacher ratio by town
B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT    % lower status of the population
MEDV     Median value of owner-occupied homes in $1000's

0.00632  18.00   2.310  0  0.5380  6.5750  65.20  4.0900   1  296.0  15.30
 396.90   4.98  24.00
0.02731   0.00   7.070  0  0.4690  6.4210  78.90  4.9671   2  242.0  17.80
 396.90   9.14  21.60
0.02729   0.00   7.070  0  0.4690  7.1850  61.10  4.9671   2  242.0  17.80
 392.83   4.03  34.70
0.03237   0.00   2.180  0  0.4580  6.9980  45.80  6.0622   3  222.0  18.70
 394.63   2.94  33.40
0.06905   0.00   2.180  0  0.4580  7.1470  54.20  6.0622   3  222.0  18.70
 396.90   5.33  36.20
0.02985   0.00   2.180  0  0.4580  6.4300  58.70  6.0622   3  222.0  18.70
 394.12   5.21  28.70
0.08829  12.50   7.870  0  0.5240  6.0120  66.60  5.5605   5  311.0  15.20
 395.60  12.43  22.90
0.14455  12.50   7.870  0  0.5240  6.1720  96.10  5.9505   5  311.0  15.20
 396.90  19.15  27.10
0.21124  12.50   7.870  0  0.5240  5.[illegible]310 100.00  6.0821   5  311.0  15.20
 3[illegible].63  [illegible]
0.17004  12.50   7.870  0  0.5240  6.0040  85.90  6.5921   5  311.0  15.20
```

For example, this first value refers to CRIM, this value refers to ZN, this value refers to INDUS, and so on, up to the thirteenth, and the fourteenth is MEDV. So, when the independent variables are 0.00632, 18, 2.31, ..., then the dependent variable turns out to be 24. To explain this more clearly, let's check what each of these variables means. CRIM means per capita crime rate by town. This is the house price data of Boston city. In Boston city, there are different towns, different areas. In each of those areas, the data points vary. There are places where the crime rate is high and places where it is low. In one of the places, it is 0.00632; in another, it is 0.02731, and so on. The second variable, ZN, refers to the proportion of residential land zoned for lots over 25,000 square feet. The third, INDUS, refers to the proportion of non-retail business acres per town. In each of these places, you have a certain percentage which is residential area and a certain percentage which is industrial area. In the first data point, 18% is the proportion of residential land, and 2.31% is the proportion of industrial land. In the next one, there is 0% residential land and 7.07% industrial land, and so on.

You can go through each of these variables. What it means is in an area where the crime rate was 0.00632, the proportion of residential land is 18%, the proportion of business acres is 2.3%, and so on, the median value of the owner-occupied homes turns out to be $24,000. This is in thousands of dollars, so 24 means $24,000. Similarly, in an area where the per capita crime rate was 0.02731, the proportion of residential land is 0%, the industrial land is 7.07%, and so on, the median value was $21,600. You have 506 data points this way. That is the data we have taken.

What are we going to do with ridge regression? Very simply, suppose you are given a new data point of this sort. We assume that the median value is related linearly to all the other variables. Given a new test vector, assume we are picking another region where you have some other crime rate, some other proportion of residential land, some other proportion of industrial land, and so on. If we give that, can we actually predict the median value? How close are we to the prediction? That is exactly what you are trying to achieve with ridge regression.

You can see that it is a machine learning problem because you are giving so many data points, and from these data points, the machine learns that if this is the crime rate, this is the proportion of residential land, this is the proportion of industrial land, and so on, the house price at that region is going to be so much. That is the kind of learning that the machine does. That is why we call this a machine learning problem or, more precisely, a supervised learning problem.

This is the data we are actually going to use. If you are asking where we would get the new test points, we have 506 data points given to us on the screen. We are going to split this into roughly 80 percent and 20 percent. Eighty percent will be the training data with which we will train the machine, and the remaining 20 percent will be the testing data. We have 506 data points in all. Eighty percent of 506 is about 405, but we will take 400 data points for training and 106 for testing. If you have learned machine learning, you would actually pick these 400 randomly. That requires a little bit of work. For the sake of simplicity, what I am doing is picking the first 400 points as the training dataset and the remaining 106 as the testing dataset.

Taking this dataset, we are just going to solve the ridge regression problem and check how good our predictions are going to be. Let us write down the code for that. We will start by first importing the data. I am going to do something you may not be familiar with because file operations were not something I taught in week two when we were introducing Python. That is more of a special application which was almost not needed for this course. I will just write the code and explain what each step does, but if you want to learn this further, you have tutorials to learn it.

What we are now going to do is import this data. For that, I need a library called pandas. Like NumPy is one library, pandas is another library which is very useful for file operations like reading from a file, writing to a file, and so on. Here we are going to read from a file. The URL is the data URL we just had: lib.stat.cmu.edu/datasets/Boston. We are going to read the data from this URL. We will use `read_csv`, a command available in pandas. But there are certain catches. We want to tell the computer that each of these data points are separated by spaces, and sometimes there are multiple spaces or newlines. How do we tell the computer that when you see spaces, assume that the first data is over and the next data is coming? We use the `sep` command, which means separator. The separation between two data points is a space. We say `s+` because it could be multiple spaces or it could be a newline. That is why it is `s+`.

There is also another issue: we do not want all the initial text. The data only starts from a certain row. If you just ask it to pick, it will include the headers like "Boston" as data points. We need to skip a certain number of rows. Let's count the number of rows to skip: 1, 2, 3, 4, 5, 6, 7, up to 22. We will skip 22 rows. There is a command for that: `skiprows=22`. Usually, it takes the first line as a header, but here we are saying there is no header. So, don't take any header; just start with the data from there.

Let's print the first ten lines of the data to see if it has picked the data correctly. The first data point should have 0.00632, 18, 2.31, and so on, but you should have had 14 data points. It is showing only 11 from index 0 to 10. In the next line, there is a value at index 3, and then NaN for the remaining places. This is a problem. What it has done is taken one data point and split it into two different lines. The reason is each data point is bent into two different lines. We need to correct this. This is called data processing. In machine learning, the initial part involves a lot of data processing before you actually go to the code.

We are spending a little bit of time doing the data processing. Now, the data is bent into two different rows. We will read all these data points: from index 0 to 10 on the first line and index 0 and 1 on the second line, all into x, and the dependent variable into y. That is how we want the data to be: x_1, y_1, x_2, y_2, x_3, y_3, ..., x_{506}, y_{506} because we have 506 data points.

For x, I am going to take `df.values` for the even rows (0, 2, 4, ...) which give the first 11 values, and then append the values from the odd rows (1, 3, 5, ...) which are the next two values. This will give a 506x13 matrix for x. For y, I just need the value from the odd rows at the second column (index 1). If you are not clear with this code, `df.values[::2]` means all rows from first to last in steps of 2. This will give a 506x11 matrix. We want to append the two values from the odd rows to get a 506x13 matrix. We do this by appending the columns from the odd rows. For y, we just pick the odd rows and the second column in each of them, which is the dependent variable.

Let's check if we have got them right by printing the first ten rows of x and y. The first row should have 0.00632, 18, 2.31, 0, ... up to 13 values. The second row should have 0.02731, 0, 7.07, 0, ... and so on. Similarly, y should have 24, 21.6, 34.7, and so on. Yes, we have got this correct.

What we have done until now is read the data from the StatLib website and stored the independent variables in x, which is a 506x13 matrix, and the dependent variable in y, which is 506 real numbers. Now we have to split this into training data and testing data. We will do that. The training data is the first 400 of x and y, and the testing data is the next 106 of x and y. So, `x_train` is `x[:400]`, `x_test` is `x[400:]`, `y_train` is `y[:400]`, and `y_test` is `y[400:]`. The training data and testing data are now split.

Now, the quick thing we can do is verify the answer of the OLS problem, the ordinary least squares problem. Recall from the last class that the solution of the ordinary least squares problem is given by $(H^TH)^{-1}H^TY$. What is H and what is Y? H is a matrix with a column of ones in the front and then the x matrix.

Note: $x_{n,p}=x_n^p$

The first row of x is $x_{1,1}, x_{1,2}, ..., x_{1,p}$, which is the p-dimensional data point x_1.

The second row is $x_{2,1}, x_{2,2}, ..., x_{2,p}$, which is x_2, and so on.

So, H is just the x_train matrix with a column of ones added in the first column. We have the x_train matrix, which is $x_{1,1}$ to $x_{n,p}$, and we add a column of ones in front. Similarly, Y is just y_train.

To create H, we insert a column of ones to x_train at the beginning. So, H is `np.insert((np.ones((400,1)), x_train))`. Now we have H, and Y is y_train.

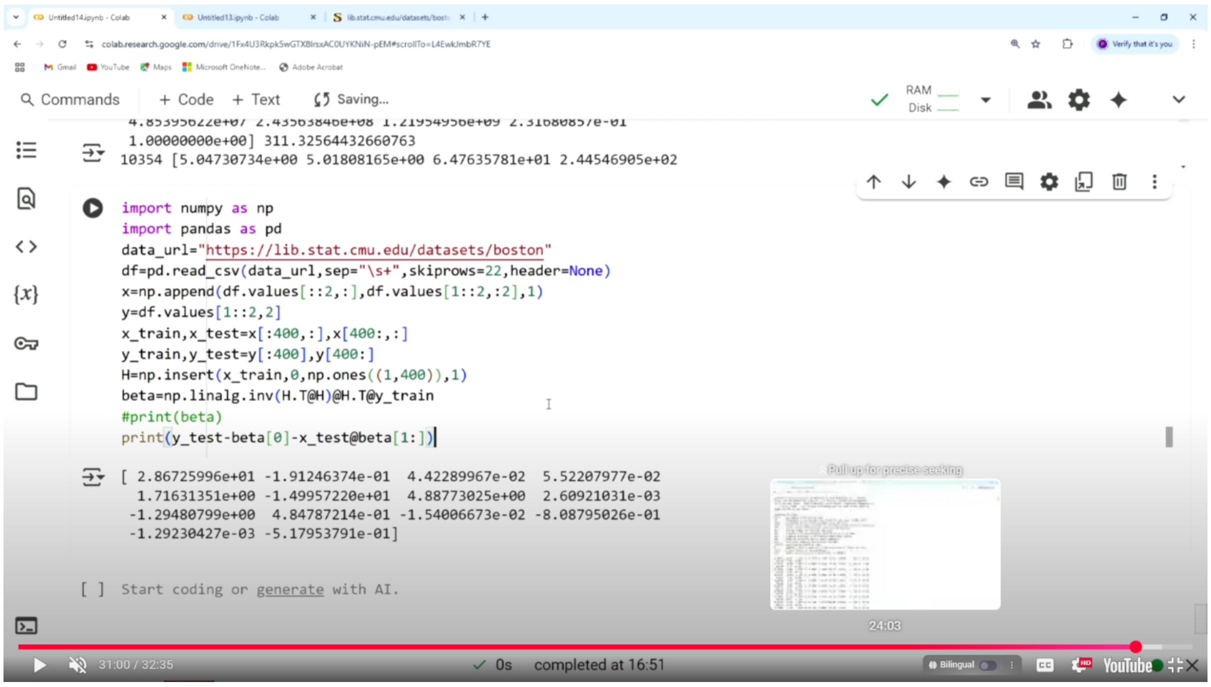
The solution $\hat{\beta}$ is $(H^TH)^{-1}H^TY$.

We can print β.

Let's see how the values of beta look. There might be an error; it should be H^TY. Let's check that. This is the answer we have. $\hat{\beta}_0$ is 28.6, $\hat{\beta}_1$ is -0.19, $\hat{\beta}_2$ is 0.0442, $\hat{\beta}_3$ is 0.0552, and so on. What this means is that for a new data point x_0,

we find the predicted value $\hat{y}_0$ by $\hat{\beta}_0 + \hat{\beta}_1 x_{0,1} + \hat{\beta}_2 x_{0,2} + ... + \hat{\beta}_p x_{0,p}$.

(Refer Slide Time 31:00)



```
 4.85395622e+07 2.43563846e+08 1.21954956e+09 2.31680857e-01
 1.00000000e+00] 311.32564432660763
10354 [5.04730734e+00 5.01808165e+00 6.47635781e+01 2.44546905e+02
```

```
import numpy as np
import pandas as pd
data_url="https://lib.stat.cmu.edu/datasets/boston"
df=pd.read_csv(data_url,sep="\s+",skiprows=22,header=None)
x=np.append(df.values[::2,:],df.values[1::2,:2],1)
y=df.values[1::2,2]
x_train,x_test=x[:400,:],x[400:,:]
y_train,y_test=y[:400],y[400:]
H=np.insert(x_train,0,np.ones((1,400)),1)
beta=np.linalg.inv(H.T@H)@H.T@y_train
#print(beta)
print(y_test-beta[0]-x_test@beta[1:])
```

```
[ 2.86725996e+01 -1.91246374e-01  4.42289967e-02  5.52207977e-02
  1.71631351e+00 -1.49957220e+01  4.88773025e+00  2.60921031e-03
 -1.29480799e+00  4.84787214e-01 -1.54006673e-02 -8.08795026e-01
 -1.29230427e-03 -5.17953791e-01]
```

[ ] Start coding or generate with AI.

Note that we have got $\hat{\beta}_0$, $\hat{\beta}_1$, $\hat{\beta}_2$, ..., $\hat{\beta}_{13}$.

With this, we can find the predicted values for the test data. We have 106 test data points, x_{400} to x_{505}. y_test is the actual value.

The predicted value for a test point x_i is $\hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + ... + \hat{\beta}_{13} x_{i,13}$.

The difference between the actual value y_i and the predicted value should be as low as possible. If we print these differences for the 106 test points, we get values like -7.018, -12.58, -8.8, and some are very close like -0.02.

The linearity assumption is not a very great assumption, but the process is very simple. A better metric to look at is the squared norm of this error vector. We square each difference and find the sum. That sum is 4016. This is the error you get when you use the OLS method.

I wanted to start with OLS; we will move to what we do in ridge regression in the next lecture. Thank you.