Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 59

Hello everyone, this is lecture four of week twelve. Recall that in the previous three lectures, we learned about the supervised learning problem. We had the simplifying assumption of a linear relation between the feature vector and the label data, and we used the squared error. In the last class, we implemented the OLS algorithm, the ordinary least squares algorithm. If you recall, the code was more about importing the data and processing the data rather than anything to do with the optimization algorithms we learned in the past eleven weeks. That is expected because we know the closed-form solution. But for ridge regression, we do not know the closed-form solution. We will implement that in this lecture.

Given that we have already imported and processed all the data in the previous code, it just remains to implement the ridge regression algorithm. Recall that in ridge regression, we have a constrained optimization version as well as an unconstrained optimization version. We will start with the constrained optimization version.

The objective function is $\sum_{i=1}^{n} ||y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_{ij}||^2$.

We will define f(x) and g(x) as we do for any constrained optimization problem.

We have not named these codes yet. For the ordinary least squares, we will name it OLS. For this, we will call it "ridge regression constrained version". I wrote it as code by mistake; let me correct that to text. All right. Now, let us define f(x). This is going to be something very similar to the OLS objective. But instead of the test data, the minimization happens on the training data. I will change the data reference to "train". Instead of beta, which is how I wrote the theory on the screen, we will write the code in terms of x. We will write this as x itself. I can write the data as x_train; that is not a problem. I have just written down the objective function. Fine.

We will write down the constraint as well.

The constraint is $\beta_1^2 + \beta_2^2 + ... + \beta_p^2 \leq s$.

Since we are writing the coefficients as x, this is going to be `x[1:].dot(x[1:]) - s`.

We have not defined s yet; we will do so shortly. We also have to define the gradient, $\nabla g(x)$. The element x[0], which is the intercept, has a derivative of zero. For the rest, the derivative is just 2x. We will insert a zero at the beginning of the derivative array for x[0].

So this is just 2 * x[1:], and then you insert a zero at the start. We will write this as an array; that will be helpful. In this array, we are inserting a zero at the beginning. That is all for $\nabla g(x)$.

Note that for constrained optimization, we have two algorithms: one is the augmented Lagrangian method and the other is the quadratic penalty method. Here we will start with the augmented Lagrangian method. We have the code in front of us. Let us pick it from there. This code is for equality constraints, but what we have is an inequality constraint. We will just use this code as is. We will copy it down and use it straight away. Let us check if things are fine.

The Lagrangian is just $f(x) + \gamma \times [g(x) + \mu/(2\gamma)]^2 - \mu^2/(4\gamma)$.

Then you have the gradient. We can remove this part since it is not required here. Yes, this is $\nabla f(x)$. So, $\nabla f(x)$ is... I think we need to write x_train with a column of ones in front. I think we have used a matrix H for that already. The other way of writing this is $H\beta$. Maybe I will do it that way because it is easier to write down the derivative and the Hessian. I am just changing this as $H \times x$; I think that should be fine. If you are asking what I am doing here,

I have written $(y - H\beta)^T(y - H\beta)$ to be precise. That is actually $||y - H\beta||^2$.

That is why you have `np.sum` and squared. The reason I wrote this is that when you differentiate this with respect to x, you can write the answer very easily.

So it is $-2 \times H^T(y\_train - Hx)$.

Now here we need to write the Hessian part. The Hessian is $2 \times H^TH$.

This part is... I do not know how many of you recall this: $2\gamma \times \nabla g \nabla g^T$, and that works when $g(x) + \mu/(2\gamma) > 0$.

You also need something for g(x) times the Hessian of g(x). Maybe I will find that page. It should be somewhere here. We only have one inequality constraint. All the H terms can be separated. You should have $\nabla^2 f(x)$. $\nabla^2 f(x)$ is what you have here, $2 \times H^TH$.

For the 2γ term, you have this: $\max(0, g(x) + \mu/(2\gamma))$ times $\nabla g(x) \nabla g(x)^T$.

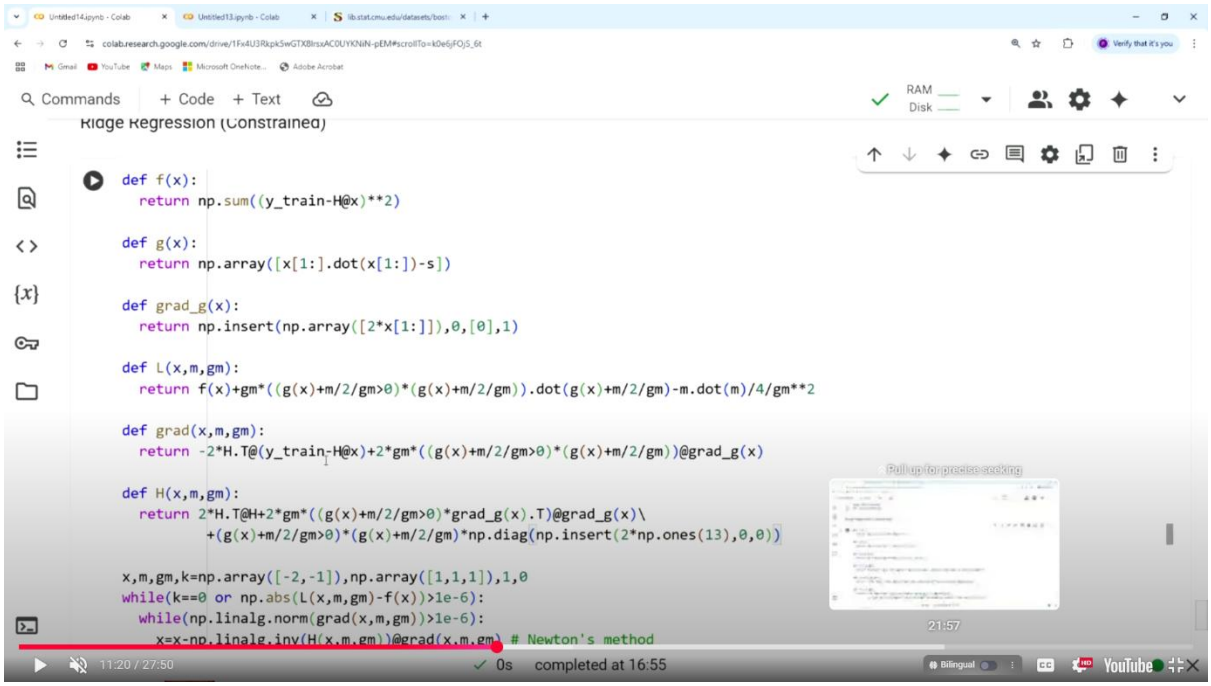
That is one part. The other part is the Hessian of g. I have to write that. Since you need $g(x) + \mu/(2\gamma) \geq 0$, I will include this here as well. You will have this part here as well, and you will also have this again.

This is $\max(0, g(x) + \mu/(2\gamma))$; nothing fancy.

And what is the Hessian of g? It is a diagonal matrix with the value 2 everywhere, but only for the coefficients, not the intercept. The first value, for the intercept, is 0. So it is a diagonal matrix with 0 and then thirteen 2's. It is a 14×14 matrix where the first diagonal element is 0 and the rest are 2. I will write that here. It is `np.diag([0] + [2]*13)`. It would have been nice if we could write it as fourteen elements easily, but now you will have to insert a zero. So with this, I am inserting a zero. Maybe if it is not nicely readable, I will write it this way. This is possibly a little more readable.

So what I have written is $\max(0, g(x) + \mu/(2\gamma))$, and this part is 0, 2, 2, 2, 2, 2... thirteen times. You are just forming a diagonal matrix with that. All right.

(Refer Slide Time 11:20)



Now you should also change the initial value. x is a 14-dimensional vector. I will just pick it as twenty ones. Sorry, fourteen ones. It is actually [20, 20, 20,...] fourteen times. I am just picking some large value. You only have one constraint. Let us set the initial multiplier to 0, and m can be 1, no problem. γ is 1, no problem, and k is fixed to be 0. All right. The rest of the code has no problem. We will disable the prints because there could be a lot of iterations. The rest of the code is the same.

We still have one more thing to accomplish, which is the value of s. This actually needs some work, but since I have done the work offline, I will just give a particular list which contains the right answer roughly. I will write down the list of s to be the following: 0.1, 0.2, 0.5, 1, 2, 5, 10, and 256. We are going to run this code for all values of s in this list. We will choose the one that has the minimum testing error. Fine. Since I have to run for all of these, I will run it with a loop: `for i in range(len(s_list))`, and `s = s_list[i]`. I have to adjust the indentation. After you finish this, we will print the value of s and the testing error because that is finally what we need. Instead of beta, it is the x we have found at the end.

All right, so this is basically the code. If you want me to spend a minute running over it again: I have defined the objective function to be minimized,

the constraint is $\hat{\beta}_1^2 + ... + \hat{\beta}_p^2 \leq s$, and the gradient ∇g.

I have defined the augmented Lagrangian function, its gradient, and its Hessian. Since we do not know what the value of s is, I am giving a list of s values, and we are going to check for each of these values.

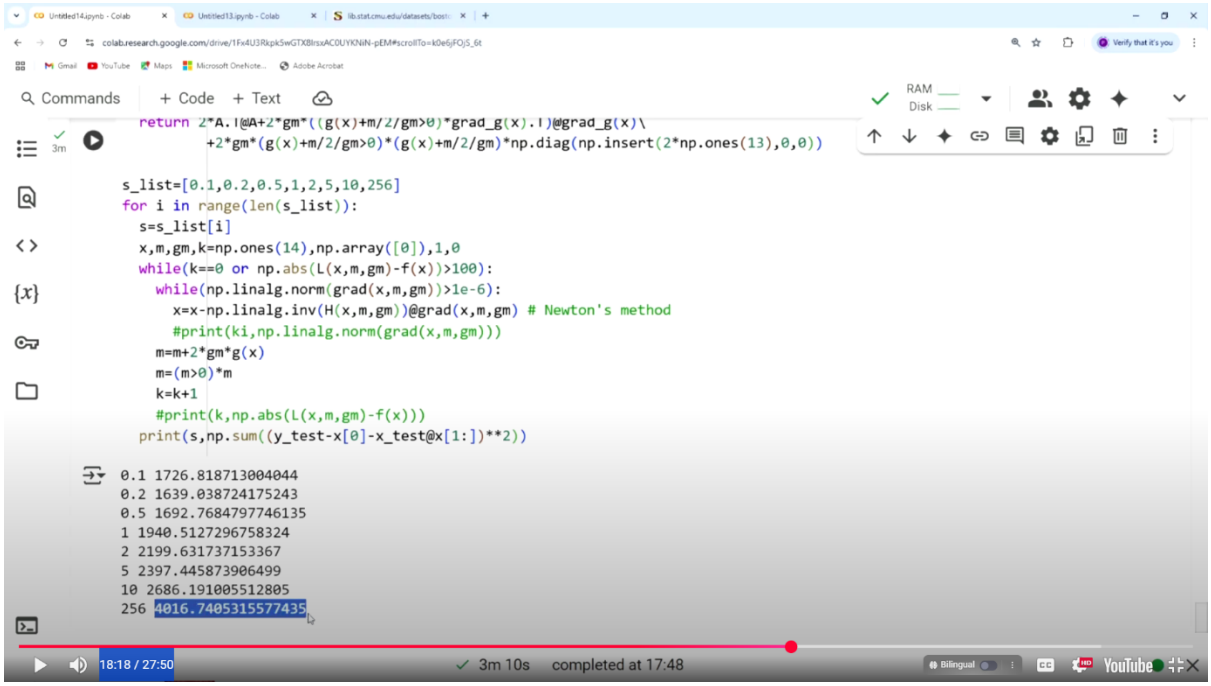
There is one more error. There is a small problem. The issue is I am using H for the matrix and also for the Hessian. I should change the name of the data matrix. Let me call it capital A instead. Since that is easier to change, I am doing that. I should first run this and get the same

answer, just changing the variable capital H to capital A. Wherever it is there, I have changed it everywhere; otherwise, it will give an error. This may take a few seconds to complete.

I have made a few changes. One of them is that we usually fix the tolerance as 1e-6. Since we are running on big data with 400 data points and also for many values of s, I increased the tolerance to 1e-4. You can give it as 1e-6; it will take more time, but the answers will not be very different.

You can see that when you give s = 0.1, the test error is 1726.8. When s = 0.2, we get 1639.04. Similarly, for 0.5 it is higher, and so on. The value 256 was given on purpose because you can see that we are getting the same answer as OLS: 4016.74.

(Refer Slide Time 18:18)



The reason is that if

s is actually greater than or equal to $\hat{\beta}_1^2$(OLS) + $\hat{\beta}_2^2$(OLS) + ... + $\hat{\beta}_p^2$(OLS),

then the condition $\hat{\beta}_1^2$ + ... + $\hat{\beta}_p^2 \leq$ s becomes redundant.

If you just put the OLS solution, the constraint is satisfied. That was the reason I gave a very high value of s, which is 256 in this case.

If you compute $\hat{\beta}_1$.dot($\hat{\beta}_1$) for the OLS coefficients, that sum is 254.58. I just wanted to give something higher than that, so I gave 256. You can see that you are getting back the OLS answer.

The most important thing to note is that when we used OLS, the testing error was 4616. When we use ridge regression, the test error drops drastically, from over 4000 to around 1600. The value that we are looking for is actually s = 0.2. You can see that the error starts at 1726,

decreases to 1639, and then increases further. You can check with more values as well. Since I have worked this out before, I am able to give you the right value of s in this case.

In fact, I included the μ value (the Lagrange multiplier) and got the μ value for each value of s. I will tell you why this is important later. Note that for each value of s, we got the corresponding μ value printed, and also the testing error. Nevertheless, the error is the least when s = 0.2.

We will run the same code with the quadratic penalty method. The function and the constraints are the same. The rest of the code will change. We have the code right here. These are for equality constraints, so we will adjust it for inequality constraints. Here is the code. We will just run this for s = 0.2 because we know that is the lowest. We just want to see whether the quadratic penalty method also gives us the answer.

We have the Q function: $f(x) + \gamma \times [g(x)]^2$.

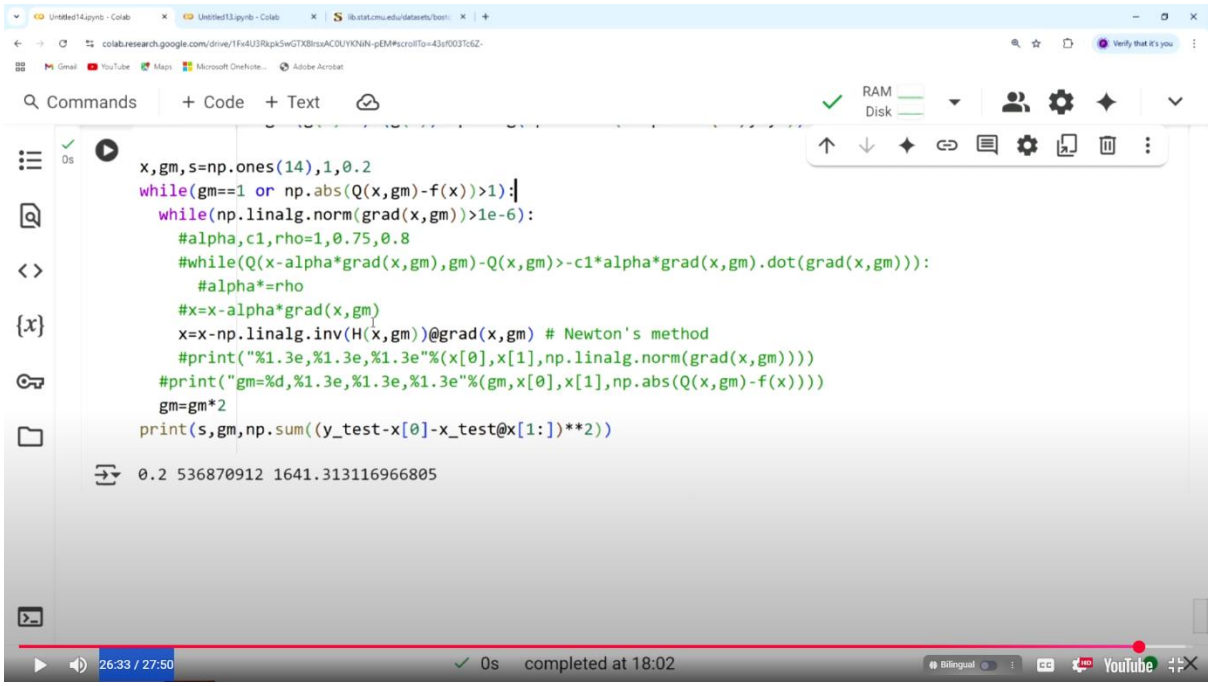
The gradient ∇f and Hessian $\nabla^2 f$ need to be modified as given here.

This is $-2A^T(y\_train - Ax)$. Here you will have $2 \times A^TA$.

You will also have the part corresponding to $\nabla g \nabla g^T$ and the part that corresponds to the Hessian of the constraint g. That will be similar to this. I will copy it and then modify it as required. This is just for g(x); you remove the μ terms. It is just for $g(x) \geq 0$, and the same Hessian matrix that we want to have. Here again, I will modify the initial γ to 1. x needs to be changed; x is just an array of fourteen ones. We will remove the prints. I will just copy the command from here. We will run the code. s needs to be set to 0.2. Here we will print γ instead of μ.

For s = 0.2, we have got around the same answer, 1641 instead of 1639. But one thing you need to note is that the value of γ is huge again. It is about 536 million. In the Indian system, that is like 53 crores. We usually get the right answer with a very high value of γ. Here, as you can see, it is 5.36×10^8, and that is where you get the right answer, even after modifying the tolerance to be 1 instead of 1e-6. Nevertheless, that is because the problem is just too huge. With some modifications, we are able to solve this constrained problem using both the augmented Lagrangian method and the quadratic penalty method.

(Refer Slide Time 26:40)



In the next lecture, which is going to be the final lecture of this course, I will also solve for the unconstrained version. In these four lectures, I have explained an application and solved a constrained optimization problem using both constrained optimization algorithms that we have learned for nonlinear programming. In the next lecture, I will work on the unconstrained version as well and also let you know about a few more commands for how to carry on with optimization if you are using Python to solve an optimization problem.

Thank you.