Pattern Recognition And Machine Learning - EPFL - Fall 2015
Emtiyaz Khan, Timur Bagautdinov, Carlos Becker, Ilija Bogunovic & Ksenia Konyushkova

2. Linear Regression and Gradient Descent

2.1 Goals

The goal of this exercise is to
- Implement grid search and gradient descent.
- Learn to debug your implementation.
- Learn to visualize results.
- Understand advantages and disadvantages of these algorithms.
- Study the effect of outliers using MSE and MAE cost functions.

2.2 Data and sample code

We will use the same dataset that we used in Exercise 1. Please download the file **`height_weight_gender.mat`** from the course website.

We have provided sample code that already contains useful snippets of code required for this exercise. Please download `gridSearch.m` and `gradientDescent.m` from the course website.

2.3 Normalizing the data

For gradient descent (and many other algorithms), it is always a good idea to preprocess your data. You should normalize input variables so that they have zero mean and unit variance. Why is this useful? This is a very important question you should think about.

In `gridSearch.m`, we have provided the code to normalize the features. Please look at it and make sure you understand it.

2.4 Computing the cost function

In this exercise, we will focus on simple linear regression which takes the following form,

$$y_n \approx f(x_{n1}) = \beta_0 + \beta_1 x_{n1} \tag{2.1}$$

We will use height as the input variable x_{n1} and weight as the output variable y_n. The coefficients β_0 and β_1 are also called *model parameters*. We will use a mean-square-error (MSE) function defined as follows,

$$\mathcal{L}(\beta_0, \beta_1) = \frac{1}{2N} \sum_{n=1}^{N} [y_n - \beta_0 - \beta_1 x_{n1}]^2 \,, \tag{2.2}$$

Our goal is to find β_0^* and β_1^* that minimize this *cost*.

Let us learn to compute this cost function in Matlab.

Exercise 2.1 Let us say that we have data for 3 people only:

- height = 1.7m, weight = 80kg
- height = 1.7m, weight = 79kg
- height = 1.7m, weight = 60kg

What is the value of the cost function when β_0 and β_1 both equal to 0? What is its value when $\beta_0 = 1$ and $\beta_1 = 2$? ■

The original weight/height data contains data for 10000 people. One way to compute the cost function is to use a `for` loop. However, Matlab allows us to write these operations in a more readable format using matrix multiplications. To be able to debug your code efficiently, this is highly recommended. Matlab (and many other languages) optimize matrix operations, which makes the code run efficiently (remember Matlab is an abbreviation for 'matrix laboratory').

Let us learn how to *vectorize* operations in Matlab. We will store all the (y_n, x_{n1}) pairs in a vector and a matrix as shown below.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\mathbf{X}} = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{N1} \end{bmatrix} \tag{2.3}$$

Let us first understand the data format.

Exercise 2.2 To understand this data format, answer the following questions,

- What does each *column* of $\widetilde{\mathbf{X}}$ represent?
- What does each *row* of $\widetilde{\mathbf{X}}$ represent?
- Why do we have 1's in $\widetilde{\mathbf{X}}$?
- If we have heights and weights of 3 people, what would be the size of $\mathbf{y}$ and $\mathbf{X}$? What would $\widetilde{X}_{32}$ represent?
- In `gridSearch.m`, we have already provided code to form `y` and `X`. Have a look at it and make sure you understand how they are constructed.
- Check if their sizes make sense (use functions `size()` and `length()`). ■

Now we will compute MSE. Define $\boldsymbol{\beta} = [\beta_0,\ \beta_1]^T$. The MSE can also be rewritten as

$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2N}\mathbf{e}^T\mathbf{e}, \text{ where } \mathbf{e} = \mathbf{y} - \widetilde{\mathbf{X}}\boldsymbol{\beta} \tag{2.4}$$

This was explained in the class. Make sure you understand why this is true.

Exercise 2.3 Let `beta = [1; 2]`. You can compute MSE as follows:

```
e = y - tX*beta %compute error
L = e'*e/(2*N)%compute MSE
```

If you get an error, most likely the sizes of the matrices and vectors are not correct.

Compute the MSE for Exercise 2.1 and check that you get the same answer. This way you are sure what you wrote is correct. Try a few more values of $\boldsymbol{\beta}$ to debug your code. ■

It is convenient to write these computations inside functions since we will have to do them many times. If you don't know how to write functions in Matlab, please read the section on functions in the Matlab guide that was given in the first exercise session.

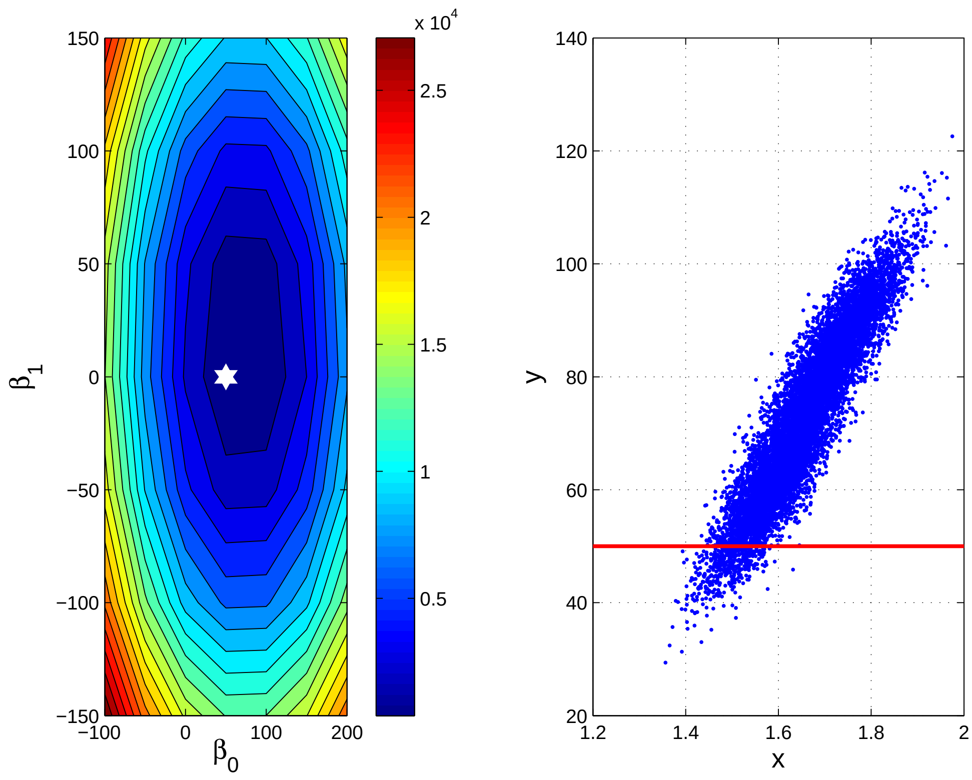
Exercise 2.4 Embed the code above in a Matlab function `L = computeCost(y, ... tX, beta)`. Test if the function yields the same output as Exercise 2.1.

2.5 Grid Search

Now we are ready to implement our first algorithm: grid search. Revise the lecture notes.

Exercise 2.5 Modify `gridSearch.m` to implement grid search. You will have to write `for` loops and compute the cost function for each setting of β_0 and β_1. Once you have all values of cost function in the variable `L`, the code finds the minimum (as discussed in the class). Make sure you understand what is happening in this code. The code should print the minimum value of the cost function along with β_0^* and β_1^*. It should also show a contour plot and the plot of the fit, as shown below.

Does this look like a good estimate? Why not? What is the problem? Why is the MSE plot not smooth?



As discussed in the class, grid search is very sensitive to the grid spacing. The following exercise will help you understand the issues.

Exercise 2.6 Repeat the above exercise by changing the grid spacing to 10 instead of 50. Compare the new fit to the old one.

Discuss with others.
- To obtain an accurate fit, do you need a coarse grid or a fine grid?
- Try different values of grid spacing. What do you observe?
- How does increasing number of values affect the computational cost? How fast

or slow does your code run?
Hint: you can use `tic` and `toc` to time execution in Matlab. ■

2.6 Gradient Descent

In the lecture, we derived the following expressions for the gradients,

$$\frac{\partial \mathcal{L}(\beta_0,\beta_1)}{\partial \beta_0} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \beta_0 - \beta_1 x_{n1}) = -\frac{1}{N}\sum_{n=1}^{N} e_n \tag{2.5}$$

$$\frac{\partial \mathcal{L}(\beta_0,\beta_1)}{\partial \beta_1} = \frac{1}{N}\sum_{i=1}^{N}(y_n - \beta_0 - \beta_1 x_{n1})x_{n1} = -\frac{1}{N}\sum_{n=1}^{N} e_n x_{n1} \tag{2.6}$$

Denoting the vector of these two gradients by $\mathbf{g}(\boldsymbol{\beta})$, we can write these operations in vector form as follows,

$$\mathbf{g}(\boldsymbol{\beta}) := \begin{bmatrix} \frac{\partial \mathcal{L}(\beta_0,\beta_1)}{\partial \beta_0} \\ \frac{\partial \mathcal{L}(\beta_0,\beta_1)}{\partial \beta_1} \end{bmatrix} = -\frac{1}{N}\begin{bmatrix} \sum_{n=1}^{N} e_n \\ \sum_{n=1}^{N} e_n x_{n1} \end{bmatrix} = -\frac{1}{N}\widetilde{\mathbf{X}}^T \mathbf{e} \tag{2.7}$$

Make sure you understand this.

Now implement a function that computes the gradients.

Exercise 2.7 Implement a Matlab function `g = computeGradient(y,tX,beta)` using Eq. 2.7. Verify that the function returns the right values. First, manually compute the gradients for hand-picked values of $\mathbf{y}$, $\widetilde{\mathbf{X}}$, and β and compare them to the output of `computeGradient`. ■

Once you make sure that your gradient code is correct, get some intuition about the gradient values:

Exercise 2.8 Compute the gradients for
- $\beta_0 = 0$ and $\beta_1 = 0$
- $\beta_0 = 50$ and $\beta_1 = 10$

What do the values of these gradients tell us? For example, think about the norm of this vector. In which case are they bigger? What does that mean? *Hint:* Imagine a quadratic function and estimate its gradient near its minimum and far from it. ■

As we know from the lecture notes, the update rule for gradient descent at step k is

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial \boldsymbol{\beta}} \tag{2.8}$$

where $\alpha > 0$ is the step size. This can be implemented easily in one line:

```
beta = beta - alpha .* g
```

Exercise 2.9 Insert your gradient code in `gradientDescent.m`. Run the code and visualize the iterations. Also, look at the printed messages that show $\mathcal{L}$ and values of $\beta_0^{(k)}$ and $\beta_1^{(k)}$. Take a detailed look at these plots,

- Is the cost being minimized?
- Is the algorithm converging?

- What are the final values of β_1 and β_0 found?
- Is the algorithm converging at a good rate?

■

Now let's experiment with the value of step size and initialization parameters and see how it influences the result and convergence. In theory, gradient descent converges when the value of the step size is chosen appropriately.

Exercise 2.10 Try the following values of step size:

(a) 0.001
(b) 0.01
(c) 0.5
(d) 1
(e) 2
(f) 2.5

What do you observe? Did the procedure converge? ■

Exercise 2.11 Try different initializations with fixed step size $\alpha = 0.1$, for instance:

- $\beta_0 = 0$, $\beta_1 = 0$
- $\beta_0 = 100$, $\beta_1 = 10$
- $\beta_0 = -1000$, $\beta_1 = 1000$

How many iterations does it take to converge now? Why? ■

2.7 Effect of Outliers and MAE cost function

In the course we talked about *outliers*. Outliers might occur due to measurement errors. For example, in the weight/height data, a coding mistake could introduce points whose weight is measured in pounds rather than kilograms.

Such outlier points may have a strong influence on model parameters. For example, MSE (the one you implemented above) is known to be sensitive to outliers, as discussed in the class.

Exercise 2.12 Let's simulate the presence of two outliers, and their effect on linear regression under MSE cost function,

- Modify the code to keep only a few data examples,

```
% load data
load('height_weight_gender.mat');
height = height * 0.025;
weight = weight * 0.454;
% sub-sample, get only 1 for every 50 points
height = height(1:50:end);
weight = weight(1:50:end);
```

- Plot the data. You should get a cloud of points similar, but less dense, than what you saw before with the whole dataset.
- As before, find the values of β_0, β_1 to fit a linear model (using MSE cost function), and plot the resulting f together with the data points.

- Now we will add two outliers points simulating the mistake that we entered the weights in pounds instead of kilograms.

```
% simulate outliers
% weight in pounds instead of kilos
height(end+1) = 1.1;
weight(end+1) = 51.5 / 0.454;
height(end+1) = 1.2;
weight(end+1) = 55.3 / 0.454;
```

- Fit the model again to the augmented dataset with the outliers. Does it look like a good fit?

■

One way to deal with outliers is to use a *robust* cost function, such as the Mean Absolute Error (MAE), as discussed in the class.

Exercise 2.13 Create a new `computeCost()` function for the Mean Absolute Error cost function. You can use function `abs()` and `sum()`.

Unfortunately, you cannot use gradient descent (why? we discussed this in the class). Use the grid search script that you wrote to fit a model to the augmented data that contains the outliers. Plot the resulting model f together with the two curves obtained in the previous exercise.

- Is the fit using MAE *better* than the one using MSE?
- You can see in the contour plot that the cost function is not strictly convex. How would this affect the optimization? Think about it.
- Bonus question: try Huber loss using gradient descent.

■