CS5487 Programming Report 1

Regression

Zhiyuan Yang
Department of Computer Science
City University of Hong Kong

Implement and test some of the regression methods: least-squares, regularized LS, L1-regularized LS, robust regression, bayesian regression. You can see some of the derivation on my blog.

Part 1 Polynomial function

(a) Implementation of the regression algorithms: see the code.

(b) Make a plot of the estimated function using polyx as inputs: see Figure 1
For BR, also plot the standard deviation around the mean: see Figure 2
Mean-squared error between the learned and the true function outputs: see Table 1

(c) Reduce the amount of training data and plot the estimated functions: see Figure 3
Make a plot of error versus training size: see Figure 4
For each size of subset, I run 20 trials of different random subsets and take the average error, and plot only last trail. Comment: From the Figure 4, I find that 100% data don't fit best, 75% fit best, when the data less than 70%, all algorithm perform worse. In the end, RLS and BR have better regression performances when the dataset is small. RLS is more robust and BR is tend to overfit.

(d) Add some outliers output values: see Figure 5 From figure 5, we can see that RR have the best performance to resist outliers. LS and RLS are most sensitive because they are square formed, so outliers with large values will amplify the error function a lot.

(e) estimate a higher-order polynomial(10th order): see Figure 6
the error: see Table 2
From the figure, I can see that LS and LLS perform better than others.

Code description

All of my code is general, can be changed by any parameters. After I have finished the separate code programming, I think whether I could combine them into a whole, and I only need to run once a time and I can get all 5 figures. So I redesigned some function to rebuild my code. And here are my code structure.

```
polynomial_function.m $ tree
├── main()
│   └── %import data
│   └── %control training data
│   └──for subset = 1:5 % each task
│   └─────for iter = 1:iteration % 20 trails
│   └──────────%regression algorithms
│   └── myerrorplot()
│   └── myplot()
│   └── myls()
```

```
    └── myrls()
    └── mylasso()
    └── myrr()
    └── mybr()
    └── mymse()
    └── mytrans()
```

Tips: If I set all function into one main function, variables can be shared globally

Figure 1: 5 regression algorithms

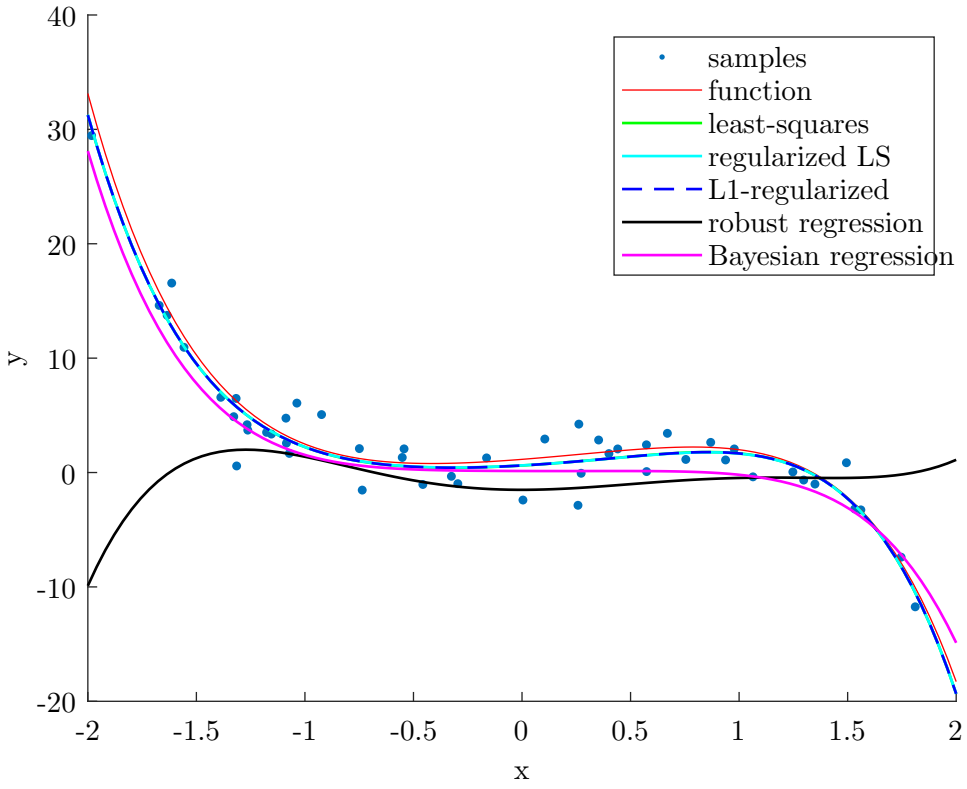

Table 1: normal mean-squared error

regression algorithms	average mean-squared error
least-squares	0.40864388
regularized LS	0.40823652
L1-regularized LS	0.40864377
robust regression	105.56778300
bayesian regression	3.90461227

Figure 2: standard deviation around the mean

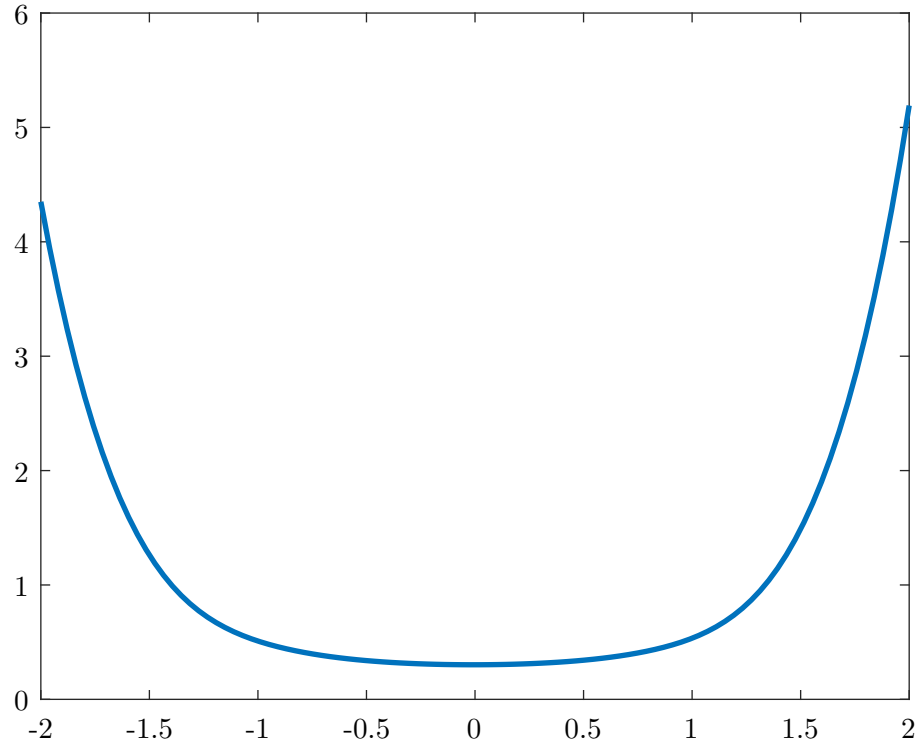

Figure 3: reduced data

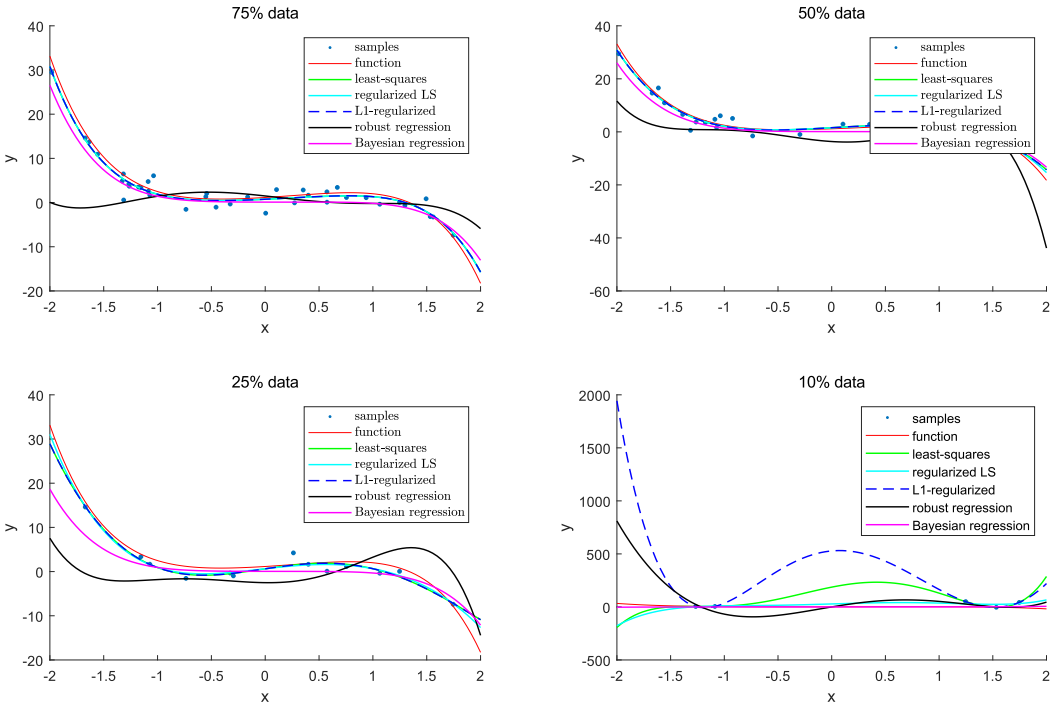

Figure 4: error versus training size

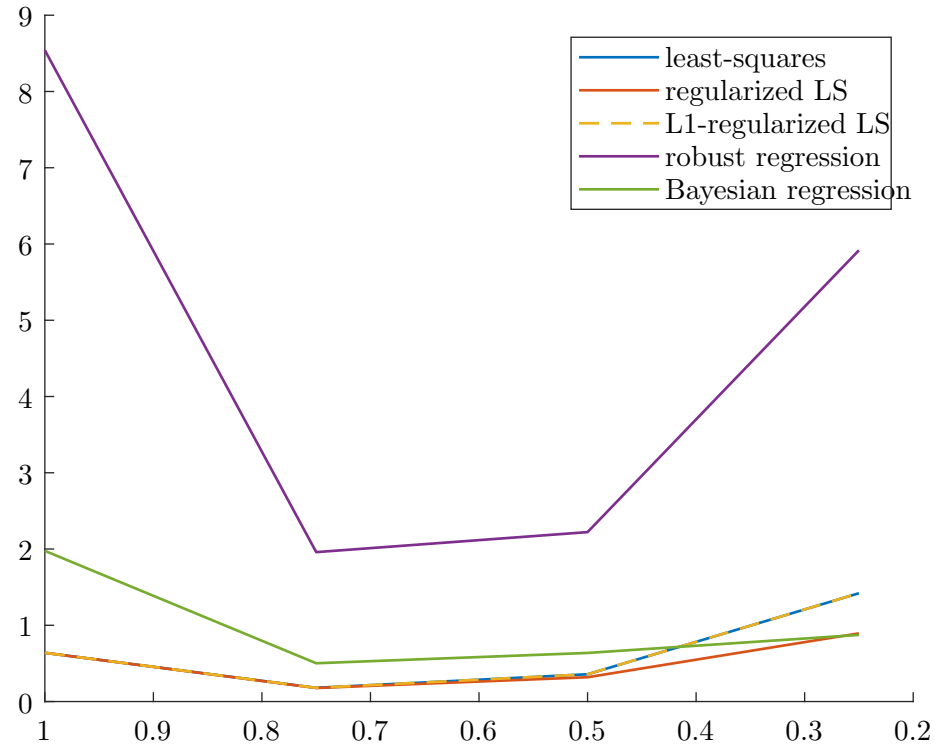

Figure 5: Add some outliers output values

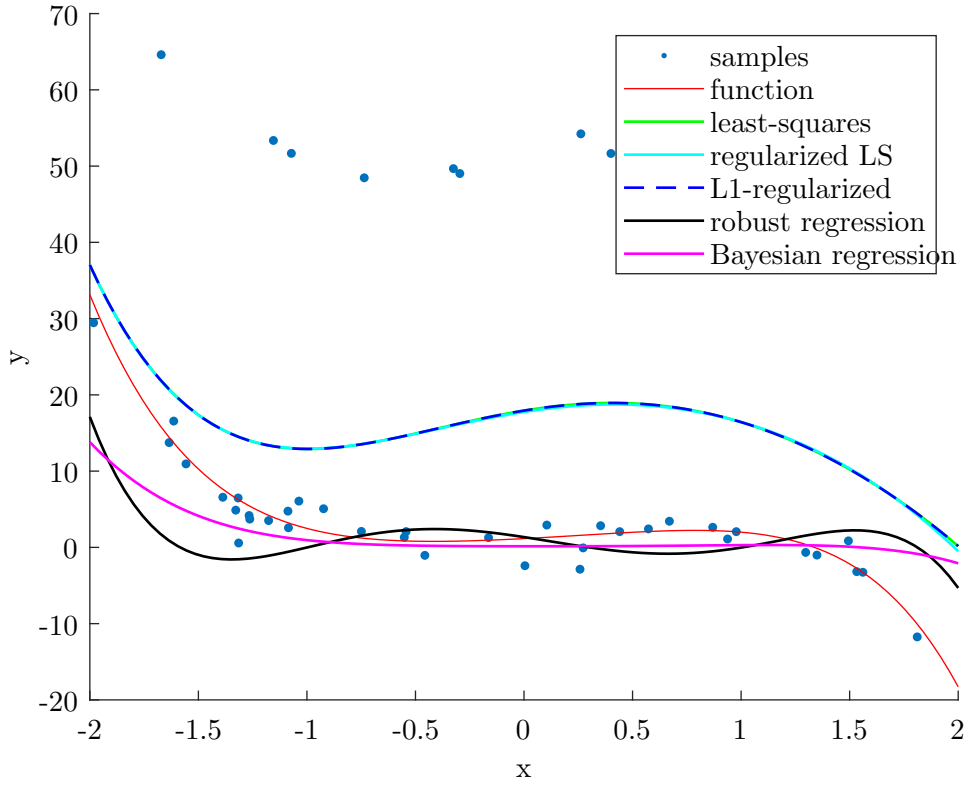

Figure 6: 10th-order polynomial

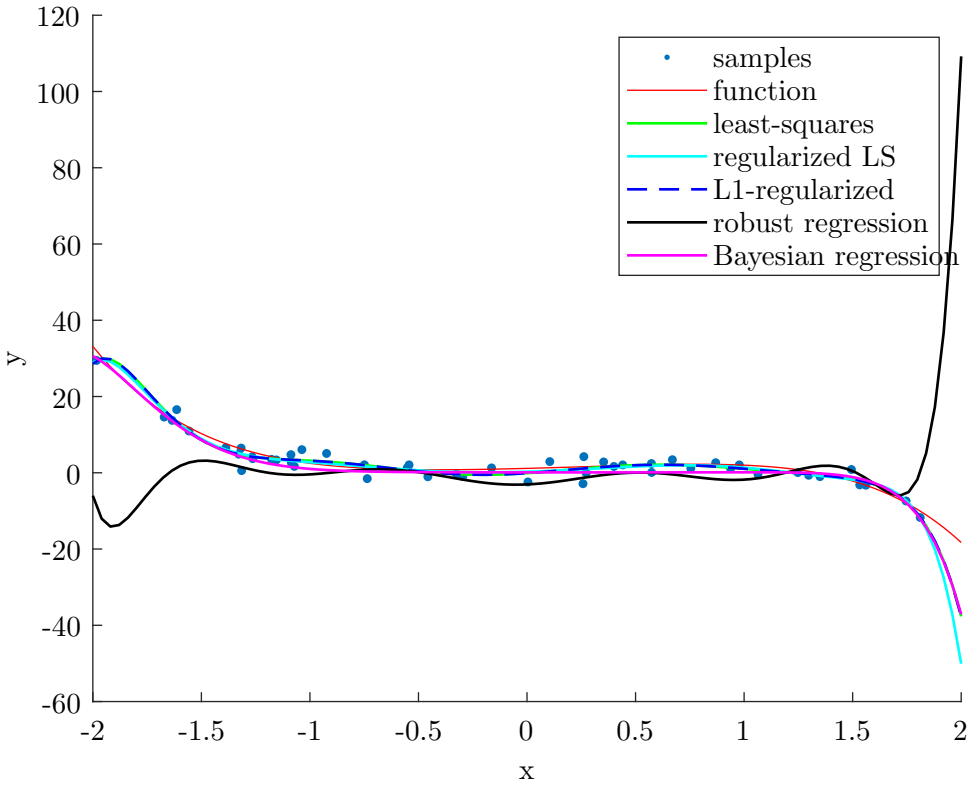

Table 2: 10th mean-squared error

regression algorithms	average mean-squared error
least-squares	7.98310659
regularized LS	17.82521110
L1-regularized LS	7.97191793
robust regression	382.81100900
bayesian regression	8.36130530

Part 2 Polynomial function

(a) Let's first look at using the features directly, see Figure 7 and the error see Table 3
I can see that "regularized LS" method works the best.

(b) Now try some other non-linear feature transformations. I have tried 2nd order polynomial, Sigmoid, sin(x), atan(x), x(1,:), Softsign...I also combine some of them together to see the performance. Above all, I find a best one: 'Phi = (exp(x) - exp(-x)) ./ (exp(x) + exp(-x));Phi = Phi(1,:)'. see Figure 8 see Table 4

Figure 7: plot counting people

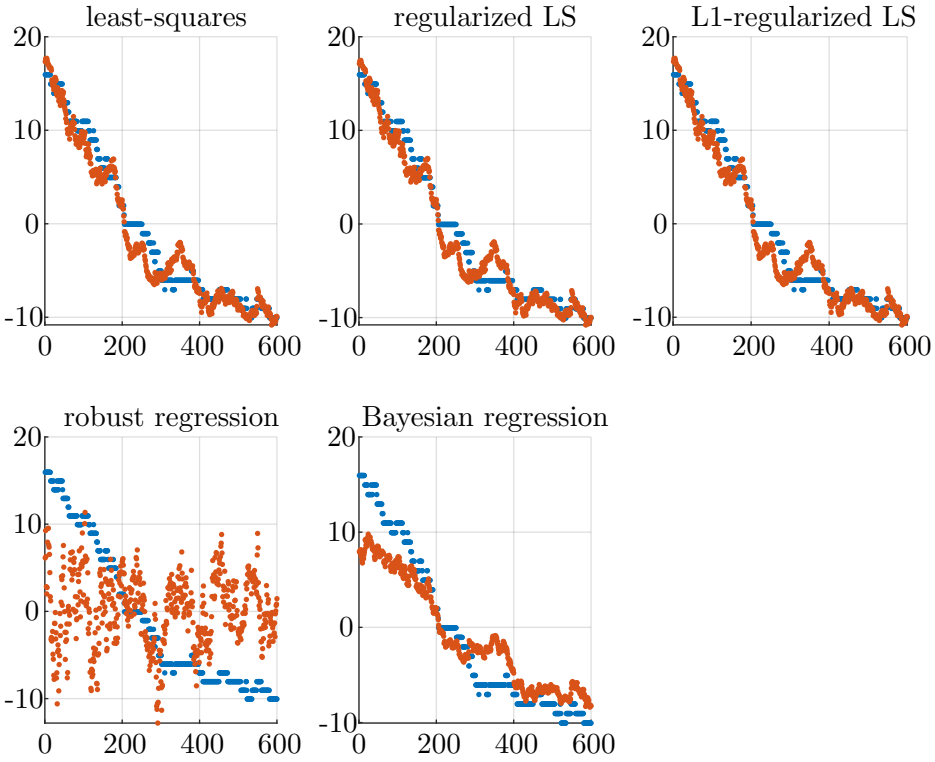

Table 3: mean-squared and mean-absolute error

regression algorithms	average mean-squared error	mean-absolute error
least-squares	3.102840	1.358400
regularized LS	2.918590	1.322900
L1-regularized LS	3.102840	1.358400
robust regression	82.720200	7.722400
bayesian regression	10.271000	2.648900

Figure 8: best plot counting people

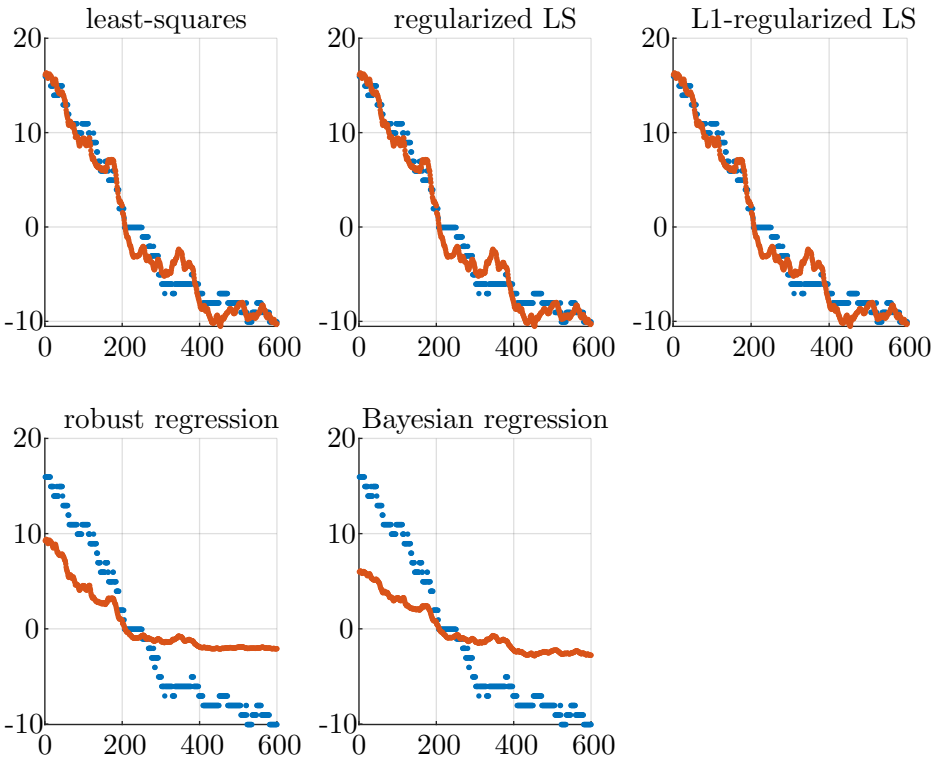

Table 4: best mean-squared and mean-absolute error

regression algorithms	average mean-squared error	mean-absolute error
least-squares	2.302690	1.226200
regularized LS	2.301690	1.225900
L1-regularized LS	2.302690	1.226200
robust regression	27.785500	4.797000
bayesian regression	32.310900	5.112700