

CS5487 Programming Assignment 1

Ji Yang

2019 October

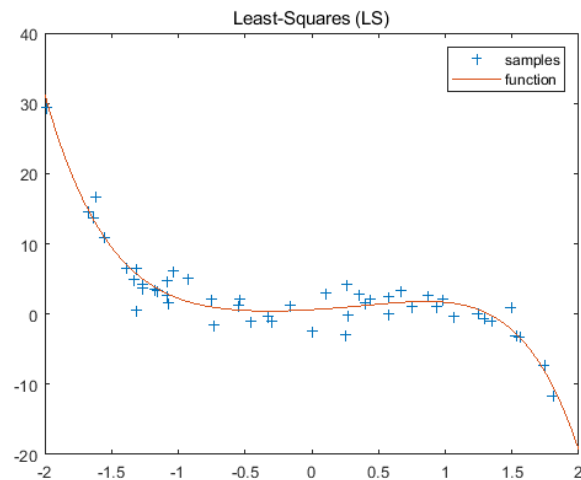
Polynomial Function

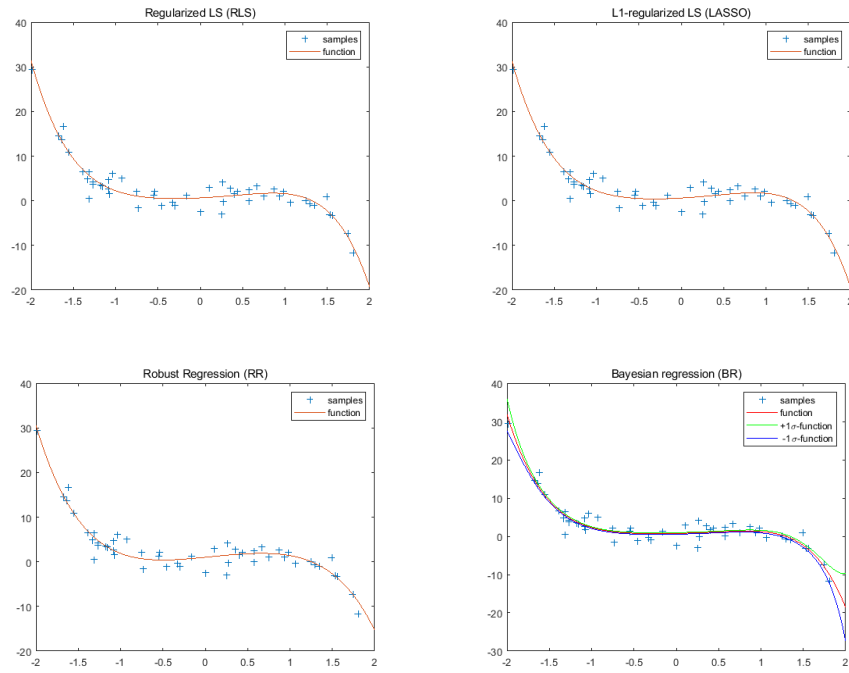
(a) Implement 5 regression algorithms

Source code can be found at <https://github.com/yangji12138/machine-learning/tree/master/Programming%201> or the Codes Appendix.

(b) Using Sample Data to estimate 5-th order poly function

	Least-Squares (LS)	Regularized LS (RLS) $\lambda = 0.48$	L1-Regularized LS (LASSO) $\lambda = 0$
MSE	0.4086	0.4076	0.4086
	Robust Regression (RR)	Bayesian Regression (BR)	
MSE	0.7680	0.4592	





(c) Subset of Training Data

	Subset Ratio	Least-Squares (LS)	Regularized LS (RLS) $\lambda = 0.48$	L1-Regularized LS (LASSO) $\lambda = 0.5$
MSE	10%	28.312	82.205	27.046
	25%	10.793	3.475	1.138
	50%	12.857	0.284	0.95
	75%	0.332	0.941	0.563
		Robust Regression (RR)	Bayesian Regression (BR)	
MSE	10%	13069.7	31.689	
	25%	1.011	13.292	
	50%	3.95	0.307	
	75%	0.257	0.73	

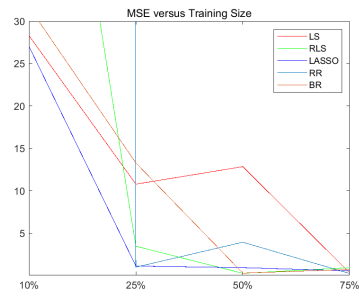


Figure 2: MSE versus training set

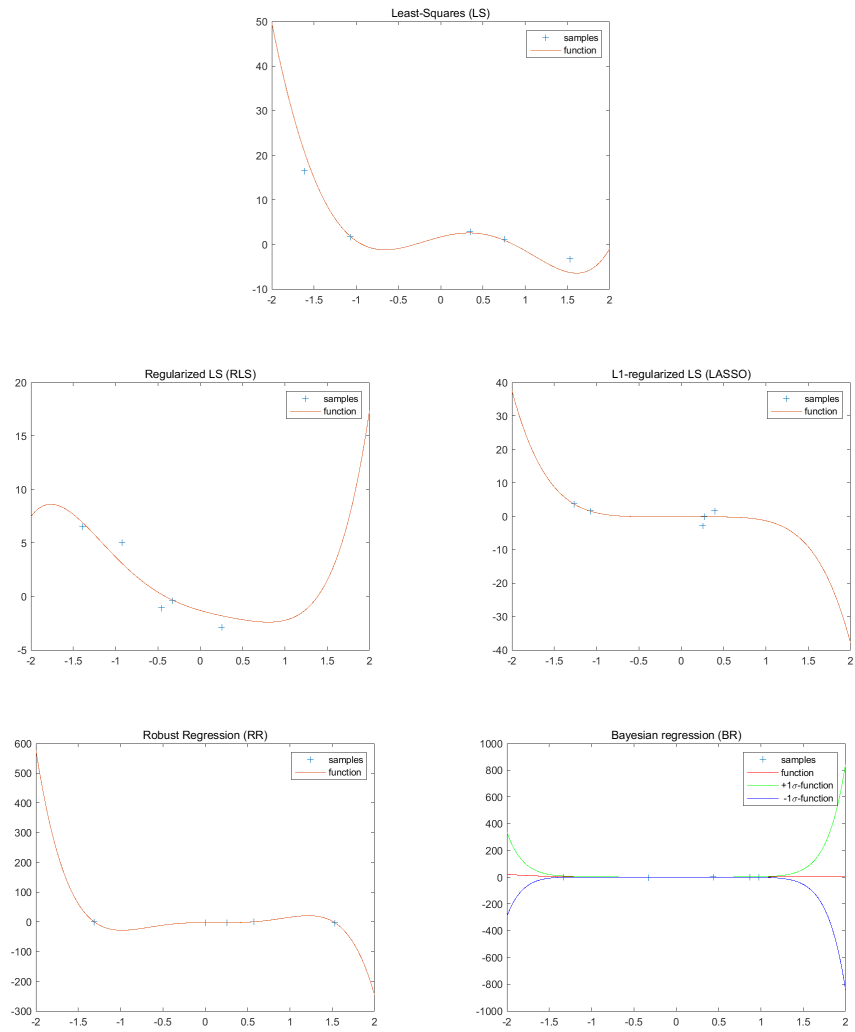


Figure 3: 10% Random Sample

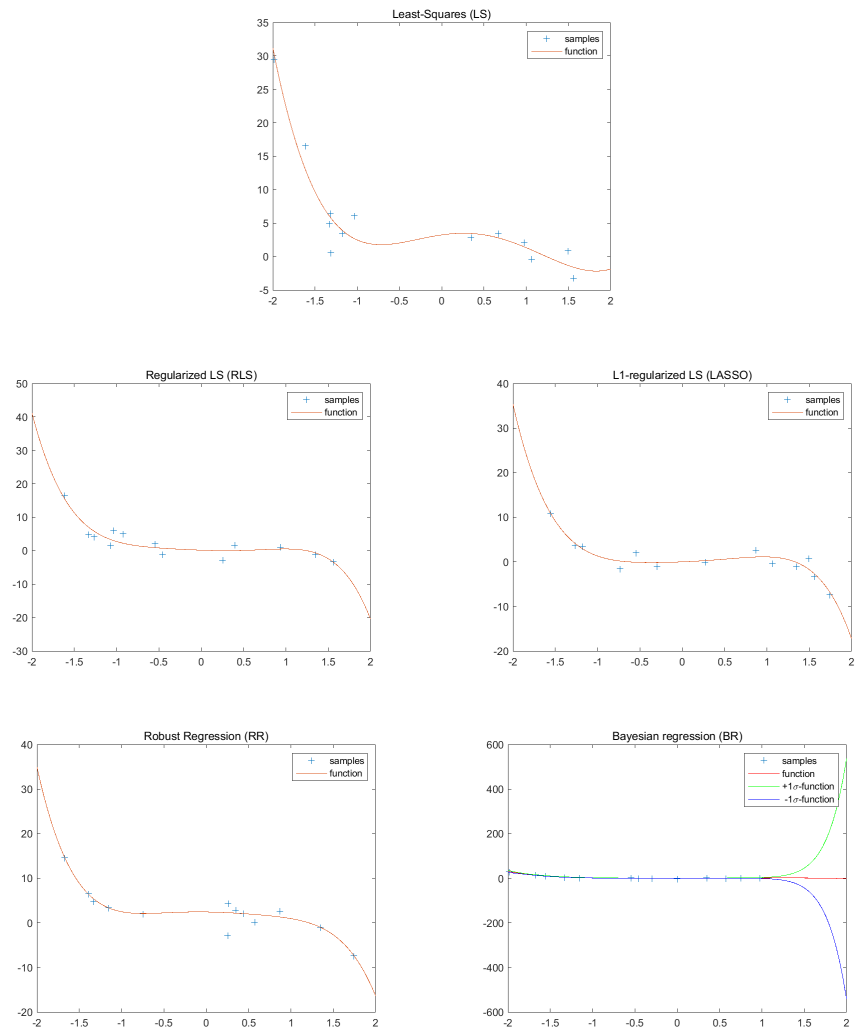


Figure 4: 25% Random Sample

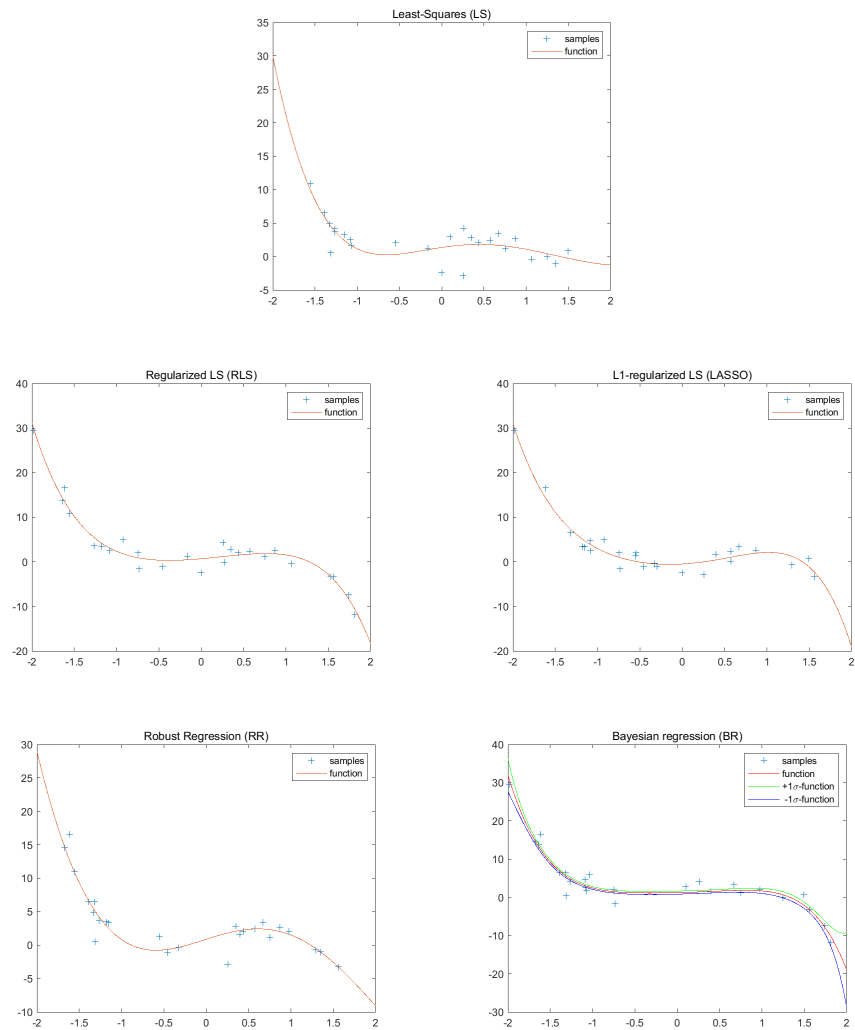


Figure 5: 50% Random Sample

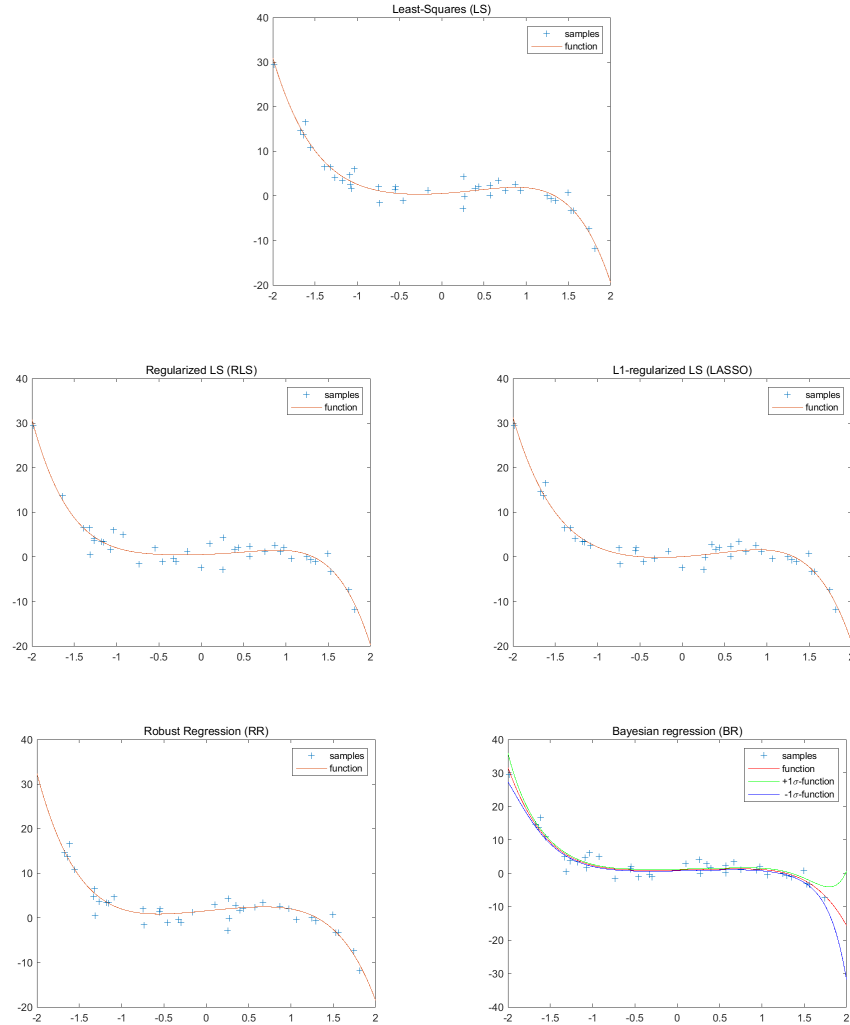


Figure 6: 75% Random Sample

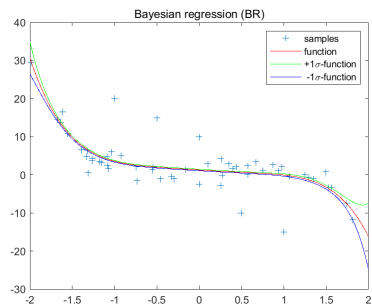
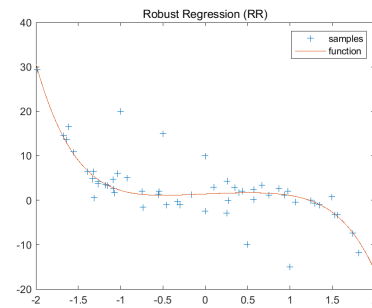
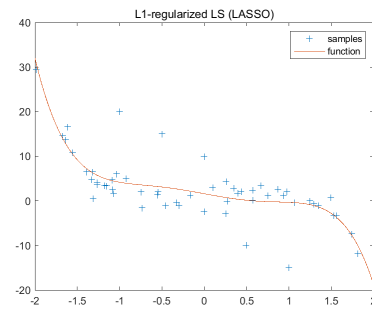
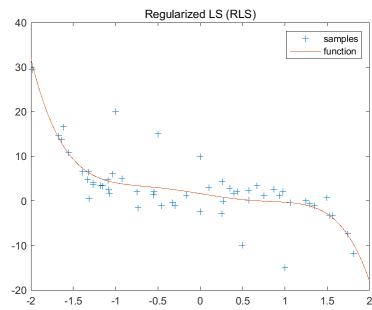
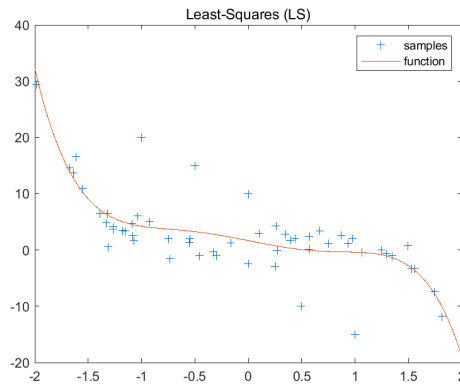
Conclusion

Observe the experiment results, we can find that:

- (i) RLS, LASSO and Bayesian Regression tends to be more robust
- (ii) RL and Roust Regression tends to overfit

(d) Adding Outlier Output Values

	Least-Squares (LS)	Regularized LS (RLS) $\lambda = 0.48$	L1-Regularized LS (LASSO) $\lambda = 1$
MSE	2.746	2.352	2.551
	Robust Regression (RR)	Bayesian Regression (BR)	
MSE	0.933	1.661	

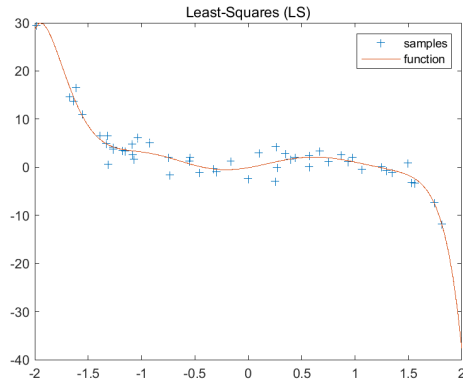


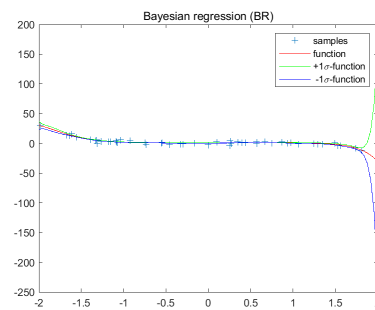
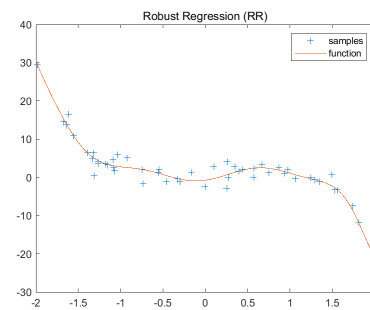
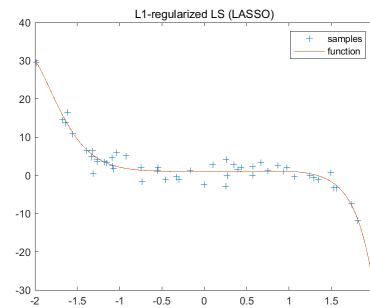
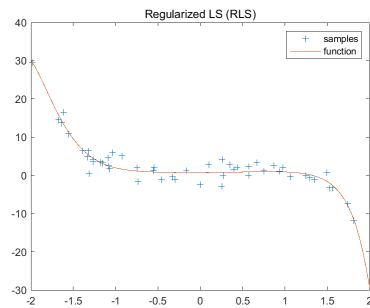
Conclusion

- (i) Here, I add 5 more values, that are obviously outliers.
- (ii) Robust Regression and Bayesian Regression are robust to the presence of outliers, while LS, RLS and LASSO are more sensitive
- (iii) Robust Regression is designed to limit the effects of outliers. Bayesian Regression contains prior knowledge which also limits the effects of data-driven prediction.

(e) Higher Order Polynomial Function (10th Order)

	Least-Squares (LS)	Regularized LS (RLS) $\lambda = 10$	L1-Regularized LS (LASSO) $\lambda = 1.8$
MSE	7.983	2.877	2.637
	Robust Regression (RR)	Bayesian Regression (BR)	
MSE	1.289	3.043	





Conclusion

- (i) Here, I test 10th order of polynomial function.
- (ii) LS tends to overfit when learning a more complex model.
- (iii) On the contrary, RLS and LASSO can avoid get overfitting.

Codes

Source code can be found at <https://github.com/yangji12138/machine-learning/tree/master/Programming%201>.

Bayesian Regression

```

1 function [miu, sigma] = BR(x ,y ,n ,q, prior1 , prior2)
2 % x is the input set;
3 % y is the output set;
4 % n is the set size
5 % q is the order of polynomial
6 b = zeros(n,1);
7
8 for i = 1:n
9     b(i) = y(i);

```

```

10     for j = 1:q+1
11         A(j,i) = x(i)^(j-1);
12     end
13 end
14
15 sigma = inv((1/prior1)*eye(q+1) + (1/prior2)*(A*A'));
16 miu = (1/prior2)*sigma*A*b;
17 end

```

LASSO

```

1 function [theta] = LASSO(x ,y ,n ,q)
2 % x is the input set;
3 % y is the output set;
4 % n is the set size
5 % q is the order of polynomial
6 % Here, A represents big phi; b is the outputs of samples
7 b = zeros(n,1);
8 for i = 1:n
9     b(i) = y(i);
10    for j = 1:q+1
11        A(j,i) = x(i)^(j-1);
12    end
13 end
14 % Since LASSO doesn't have closed-form answer, we use
    equivalent quadratic programming
15 % to get a approximate solution
16 lamda = 0.5;
17 % Set quadprog parameter
18 H = [A*(A') , -A*(A') ; -A*(A') , A*(A') ];
19 f = lamda*ones(2*q+2,1) - [A*b;-A*b];
20 B = -eye(2*q+2);
21 c = zeros(2*q+2,1);
22 target = quadprog(H,f,B,c);
23 thetaM = target(1:q+1,:);
24 thetaN = target(q+2:end,:);
25 theta = thetaM - thetaN;
26 end

```

Linear Regression

```

1 function [theta] = LS(x ,y ,n ,q)
2 % x is the input set;
3 % y is the output set;
4 % n is the set size
5 % q is the order of polynomial
6 b = zeros(n,1);

```

```

7 for i = 1:n
8     b(i) = y(i);
9     for j = 1:q+1
10         A(j,i) = x(i)^(j-1);
11     end
12 end
13 theta = inv(A*(A'))*A*b;
14 end

```

RLS

```

1 function [theta] = RLS(x ,y ,n,q)
2 % x is the input set;
3 % y is the output set;
4 % n is the set size;
5 % q is the order of polynomial;
6 % Here we set lamda 1;
7 b = zeros(n,1);
8 lamda = 0.48;
9 for i = 1:n
10     b(i) = y(i);
11     for j = 1:q+1
12         A(j,i) = x(i)^(j-1);
13     end
14 end
15 theta = inv(A*(A') + lamda * eye(q+1))*A*b;
16 end

```

Robust Regression

```

1 function [theta] = robust(x ,y ,n ,q)
2 % x is the input set;
3 % y is the output set;
4 % n is the set size
5 % q is the order of polynomial
6 % Here, A represents big phi; b is the outputs of samples
7 b = zeros(n,1);
8 for i = 1:n
9     b(i) = y(i);
10     for j = 1:q+1
11         A(j,i) = x(i)^(j-1);
12     end
13 end
14
15 %Transform the program into a standard linear program
16 f = [ zeros(q+1,1); ones(n,1) ];
17 B = [-A', -eye(n); A', -eye(n) ];

```

```
18 c = [-b;b];  
19 target = linprog(f,B,c);  
20 theta = target(1:(q+1),:);  
21 end
```