CS5487 Programming Assignment 1

Ji Yang

2019 October

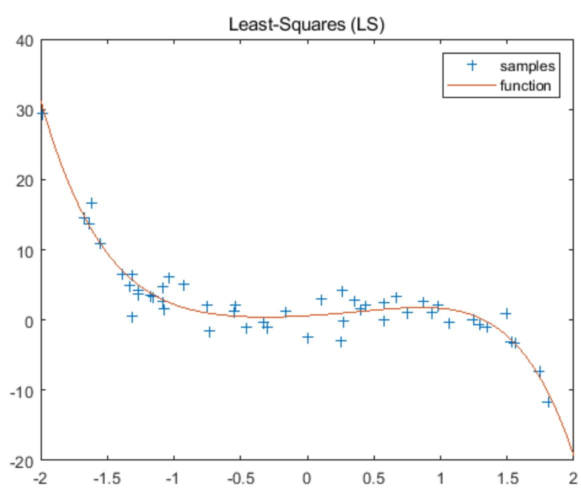
1 Polynomial Function

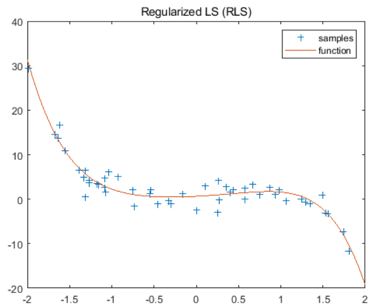
1 (a) Implement 5 regression algorithms

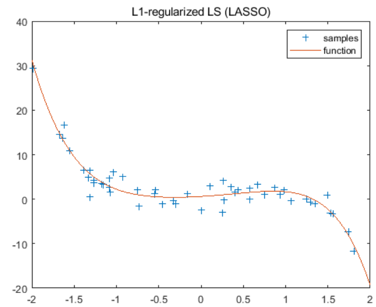
Source code can be found at `https://github.com/yangji12138/machine-learning/tree/master/Programming%201` or the Codes Appendix.

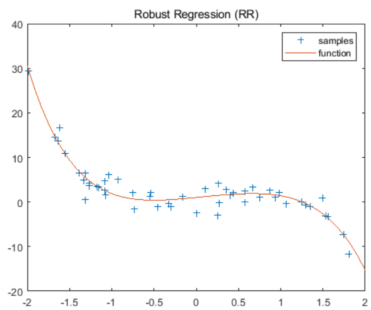
1 (b) Using Sample Data to estimate 5-th order poly function

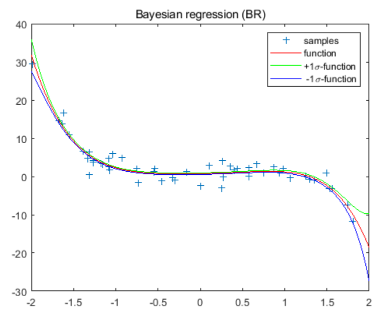
	Least-Sqaures (LS)	Regularized LS (RLS) $\lambda = 0.48$	L1-Regularized LS (LASSO) $\lambda = 0$
MSE	0.4086	0.4076	0.4086
	Robust Regression (RR)	Bayesian Regression (BR)	
MSE	0.7680	0.4592	











1 (c) Subset of Training Data

	Subset Ratio	Least-Sqaures (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	

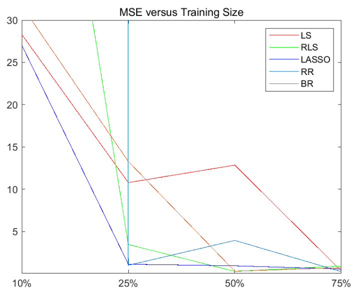


Figure 2: MSE versus training set

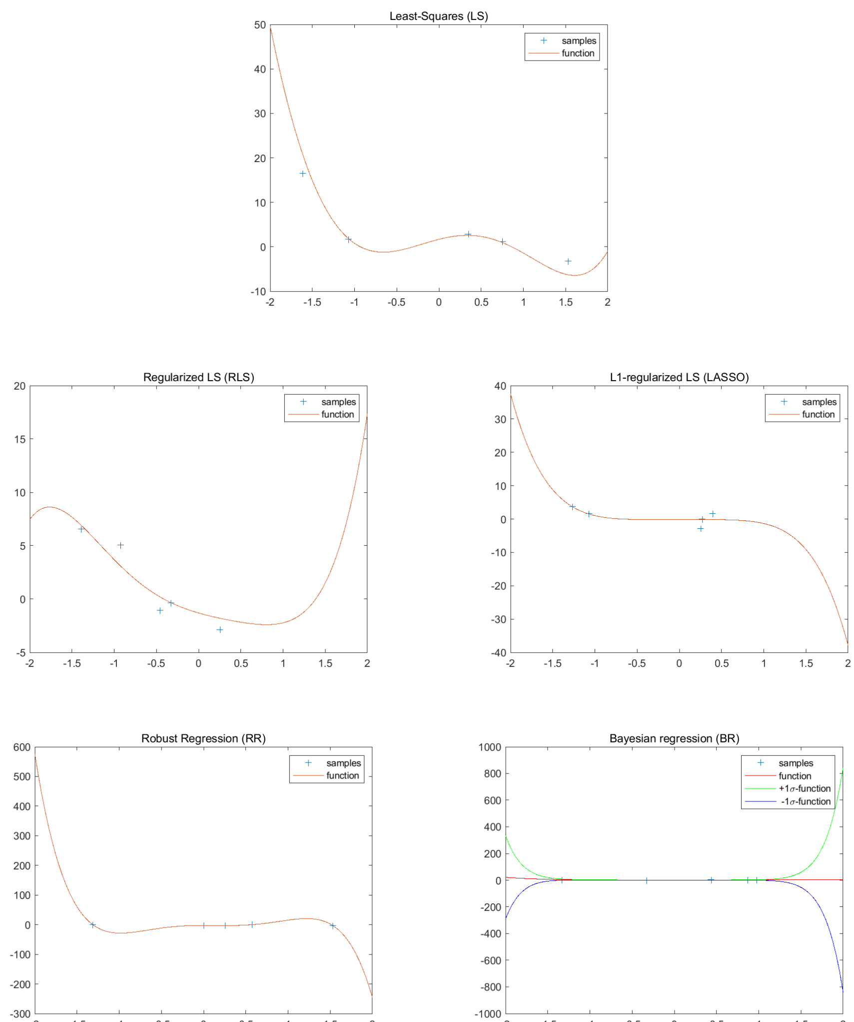


Figure 3: 10% Random Sample

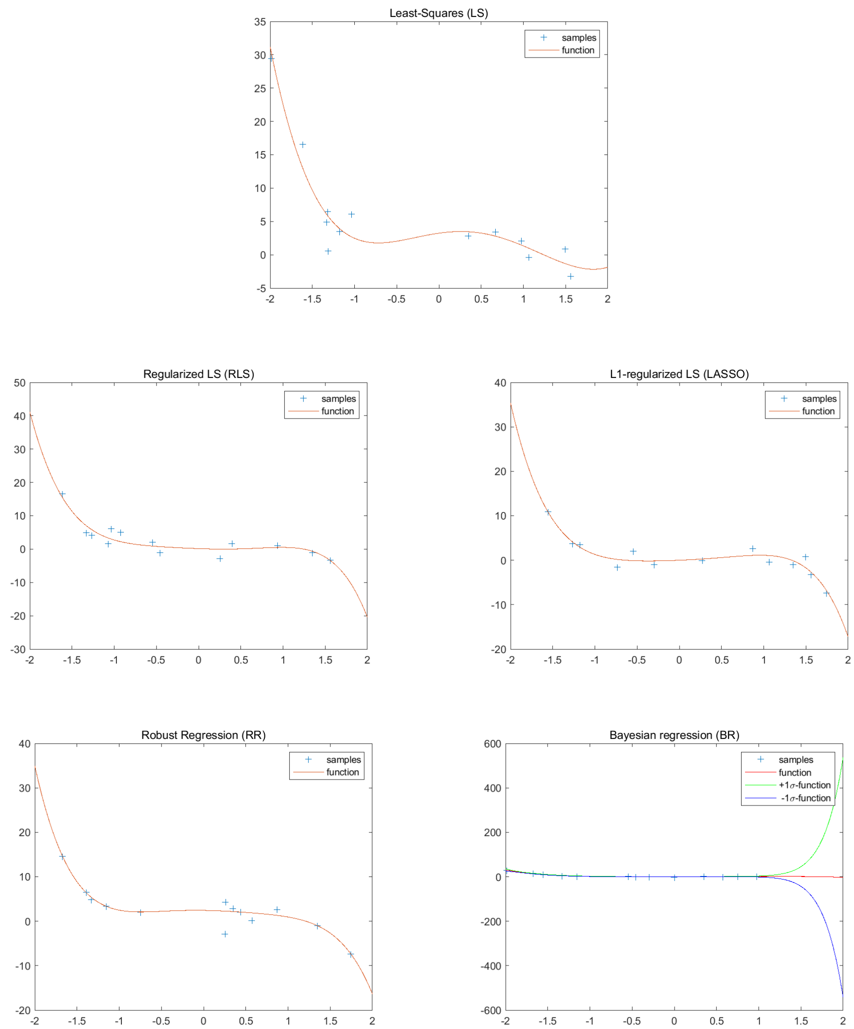


Figure 4: 25% Random Sample

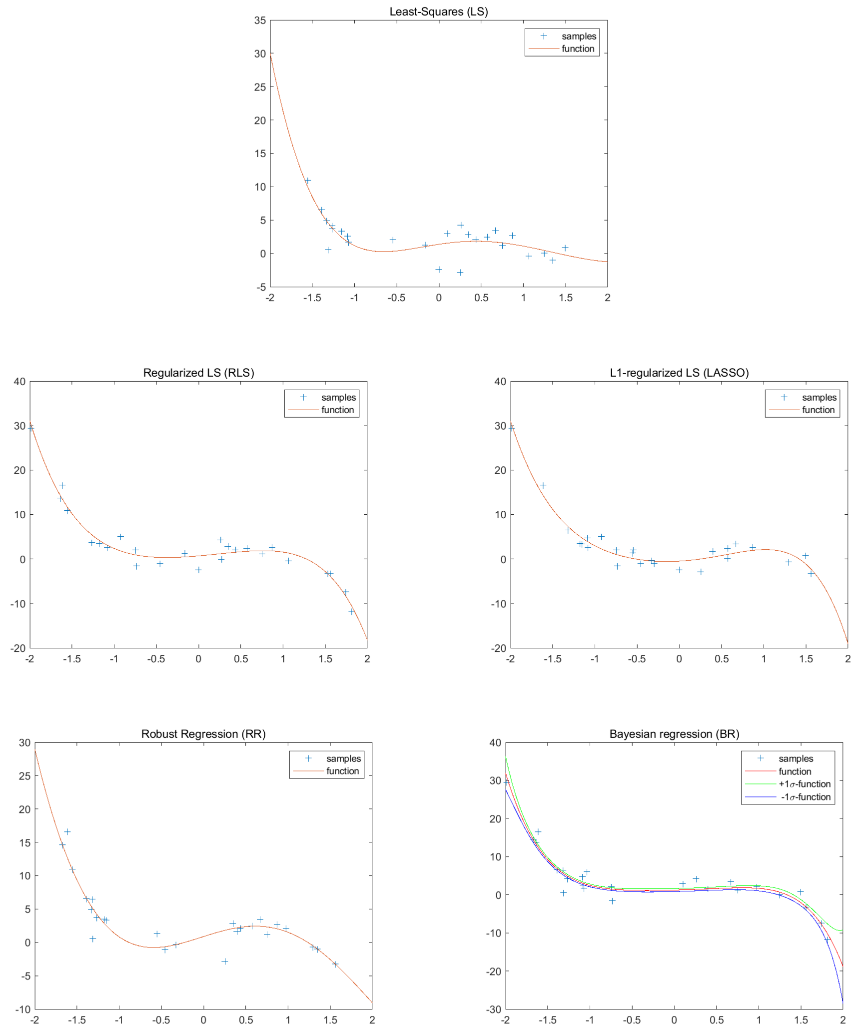


Figure 5: 50% Random Sample

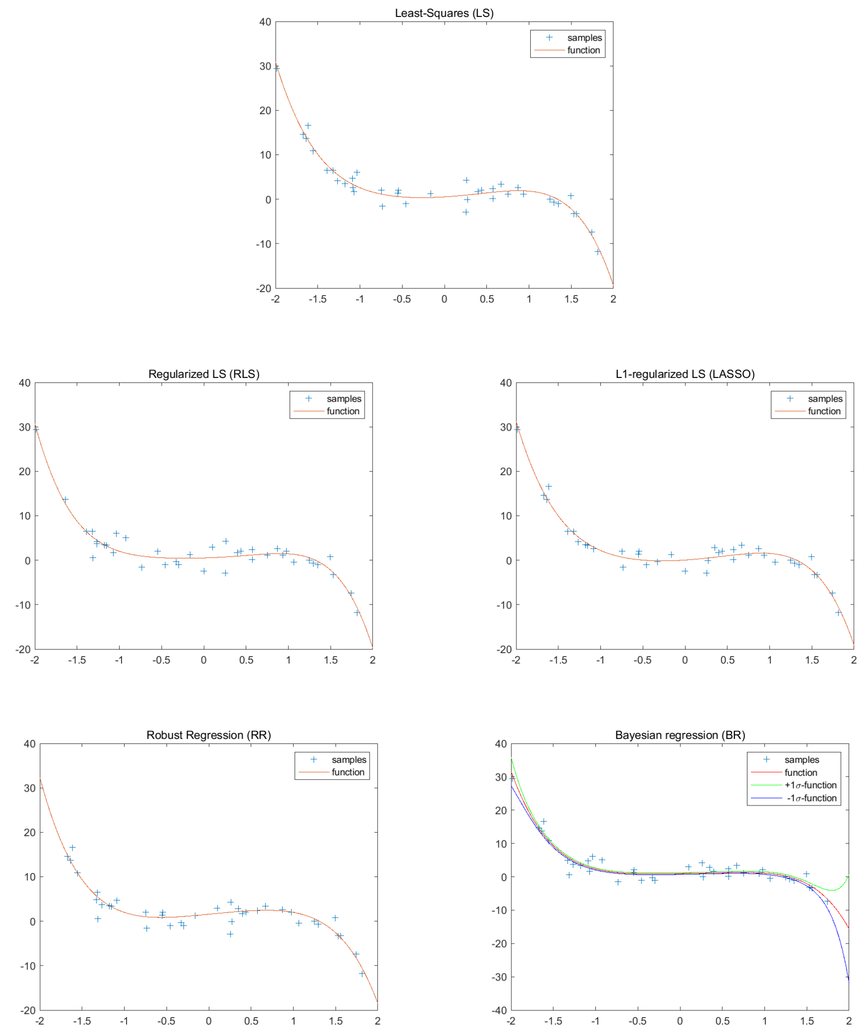


Figure 6: 75% Random Sample

Conclusion

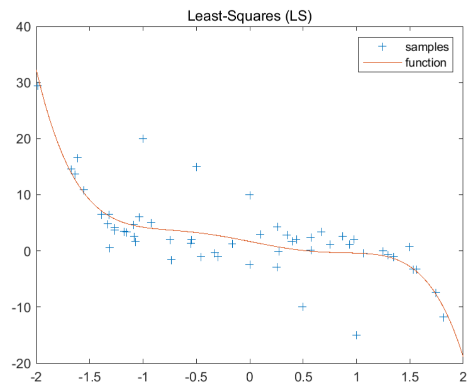
Observe the experiment results, we can find that:

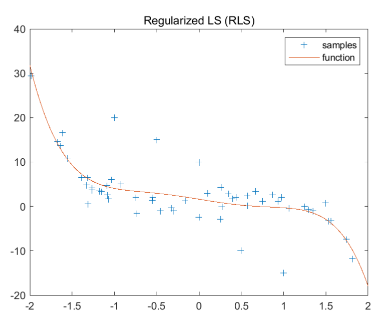
(i) RLS, LASSO and Bayesian Regression tends to be more robust

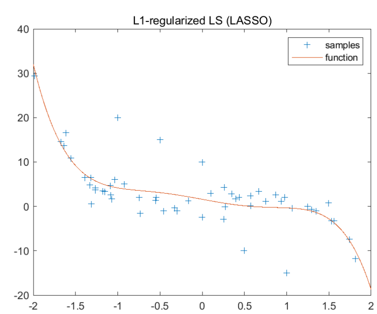
(ii) RL and Roust Regression tends to overfit

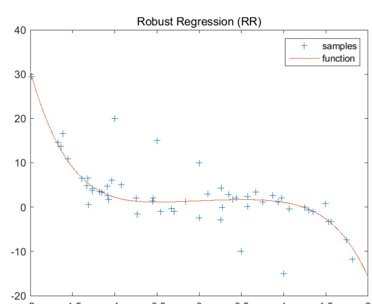
(d) Adding Outlier Output Values

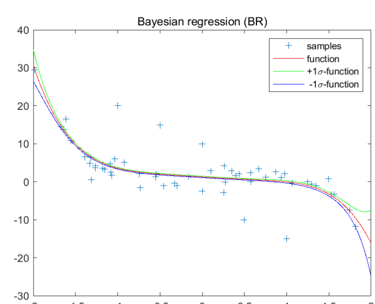
	Least-Sqaures (LS)	Regularized LS (RLS) $\lambda = 0.48$	L1-Regularized LS (LASSO) $\lambda = 1$
MSE	2.746	2.352	2.551
	Robust Regression (RR)	Bayesian Regression (BR)	
MSE	0.933	1.661	











Conclusion

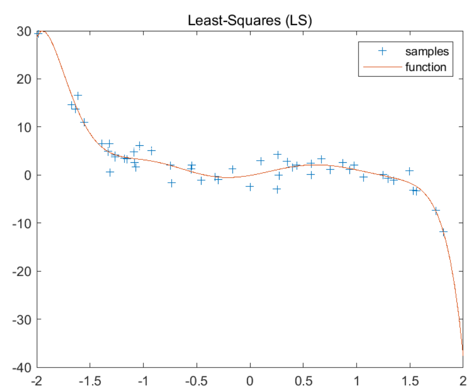
(i) Here, I add 5 more values, that are obviously outliers.

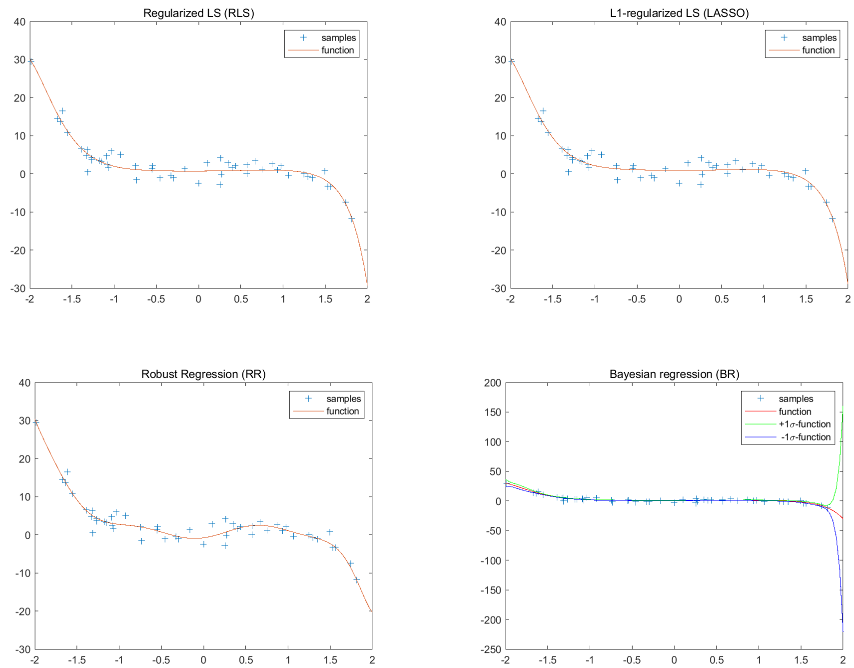
(ii) Robust Regression and Bayesian Regression are robust to the presence of outliers, while LS, RLS and LASSO are more sensitive

(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-Sqaures (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

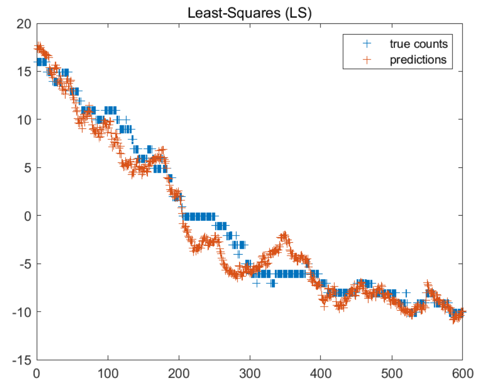
(i) Here, I test 10th order of polynomial function.

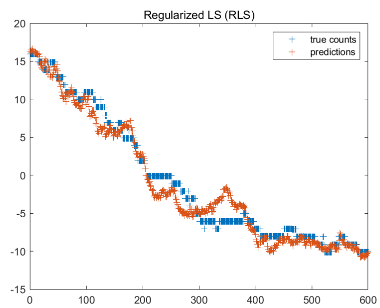
(ii) LS tends to overfit when learning a more complex model.

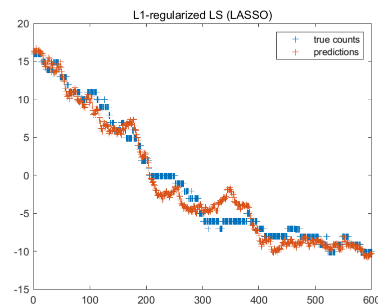
(iii) On the contrary, RLS and LASSO can avoid get overfitting.

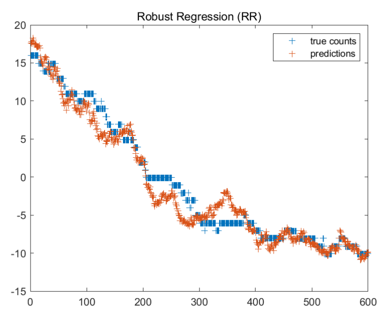
2 (a) Using Feature Directly

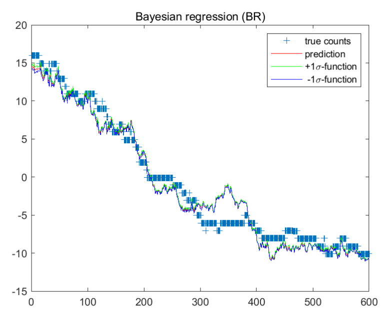
	Least-Sqaures (LS)	Regularized LS (RLS) $\lambda = 0.7$	L1-Regularized LS (LASSO) $\lambda = 4$
MSE	3.102	2.62	2.45
MAE	1.365	1.277	1.25
	Robust Regression (RR)	Bayesian Regression (BR)	
MSE	3.112	3.146	
MAE	1.365	1.448	











Conclusion

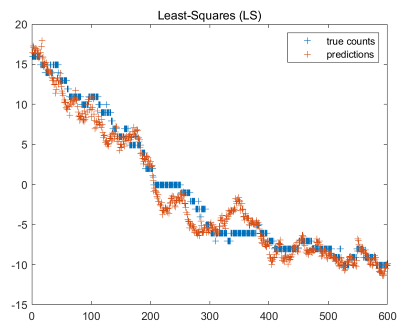
(i) LASSO works the best.

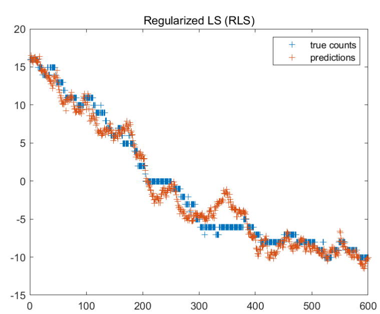
(ii) I find that all the methods have the similar results finally

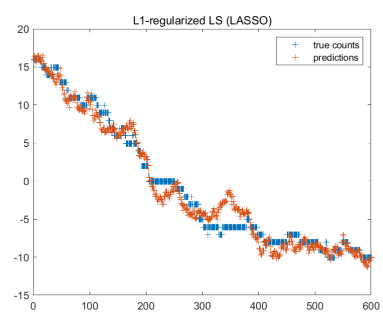
2 (b) 2nd Order Polynomial

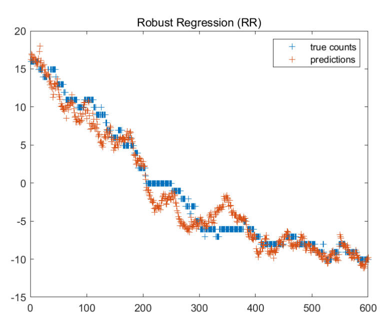
	Least-Sqaures (LS)	Regularized LS (RLS) $\lambda = 0.57$	L1-Regularized LS (LASSO) $\lambda = 3$
MSE	2.923594	2.425267	2.265683
MAE	1.326746	1.207617	1.177769

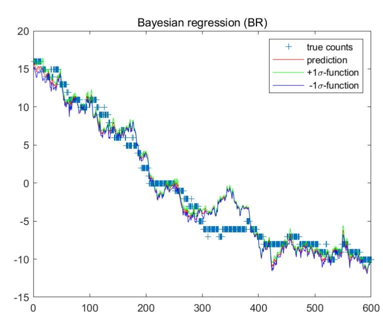
	Robust Regression (RR)	Bayesian Regression (BR)
MSE	2.898128	2.913107
MAE	1.307929	1.296899











Conclusion

(i) I find that this 2nd order polynomial feature transformation has a better performance!

Codes

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

Bayesian Regression

```
function [miu, sigma] = BR(x ,y ,n ,q, prior1, prior2)
% x is the input set;
% y is the output set;
% n is the set size
% q is the order of polynomial
b = zeros(n,1);

for i = 1:n
    b(i) = y(i);
    for j = 1:q+1
        A(j,i) = x(i)^(j-1);
    end
end

sigma = inv((1/prior1)*eye(q+1) + (1/prior2)*(A*A'));
miu = (1/prior2)*sigma*A*b;
end
```

LASSO

```
function [theta] = LASSO(x ,y ,n ,q)
% x is the input set;
% y is the output set;
% n is the set size
% q is the order of polynomial
% Here, A represents big phi; b is the outputs of samples
b = zeros(n,1);
for i = 1:n
    b(i) = y(i);
    for j = 1:q+1
        A(j,i) = x(i)^(j-1);
    end
end
% Since LASSO doesn;t have closed-form answer, we use equivalent quadratic programming
```

```
% to get a approximate solution
lamda = 0.5;
% Set quadprog parameter
H = [A*(A'), -A*(A'); -A*(A'), A*(A')];
f = lamda*ones(2*q+2,1) - [A*b;-A*b];
B = -eye(2*q+2);
c = zeros(2*q+2,1);
target = quadprog(H,f,B,c);
thetaM = target(1:q+1,:);
thetaN = target(q+2:end,:);
theta = thetaM - thetaN;
end
```

Linear Regression

```
function [theta] = LS(x ,y ,n ,q)
% x is the input set;
% y is the output set;
% n is the set size
% q is the order of polynomial
b = zeros(n,1);
for i = 1:n
    b(i) = y(i);
    for j = 1:q+1
        A(j,i) = x(i)^(j-1);
    end
end
theta = inv(A*(A'))*A*b;
end
```

RLS

```
function [theta] = RLS(x ,y ,n,q)
% x is the input set;
% y is the output set;
% n is the set size;
% q is the order of polynomial;
% Here we set lamda 1;
b = zeros(n,1);
lamda = 0.48;
for i = 1:n
    b(i) = y(i);
    for j = 1:q+1
        A(j,i) = x(i)^(j-1);
    end
end
theta = inv(A*(A') + lamda * eye(q+1))*A*b;
```

```
end
```

Robust Regression

```
function [theta] = robust(x ,y ,n ,q)
% x is the input set;
% y is the output set;
% n is the set size
% q is the order of polynomial
% Here, A represents big phi; b is the outputs of samples
b = zeros(n,1);
for i = 1:n
    b(i) = y(i);
    for j = 1:q+1
        A(j,i) = x(i)^(j-1);
    end
end

%Transform the program into a standard linear program
f = [zeros(q+1,1);ones(n,1)];
B = [-A',-eye(n);A',-eye(n)];
c = [-b;b];
target = linprog(f,B,c);
theta = target(1:(q+1),:);
end
```