Solution to Assignment 1

Shoeb Mohammed and Zhuo Chen

January 25, 2016

1 Locally weighted linear regression

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{m}w^{(i)}(\theta^T x^{(i)}-y^{(i)})^2 \tag{1}$$

1.1

Matrix X and vectors θ , y are

$$X=\begin{bmatrix}{x^{(i)}}^T\\ \vdots \\ {x^{(m)}}^T\end{bmatrix},\theta=\begin{bmatrix}\theta_1\\ \vdots \\ \theta_d\end{bmatrix},y=\begin{bmatrix}y^{(i)}\\ \vdots \\ y^{(m)}\end{bmatrix} \tag{2}$$

Let W be the $m\times m$ diagnoal matrix

$$W=\frac{1}{2}diag(w^{(1)},\cdots,w^{(m)}) \tag{3}$$

Using (2) and (3), equation (1) can be written as

$$J(\theta)=(X\theta-y)^T W(X\theta-y) \tag{4}$$

1.2

The normal equations for un-weighted linear regression are

$$X^T X\theta=X^T y \tag{5}$$

Equation 4 can be re-written as below. Note that weights are greater than zero (negative weights don't make sense).

$$\begin{aligned}J(\theta)&=(X\theta-y)^T\sqrt{W}\sqrt{W}(X\theta-y)\\ &=(\sqrt{W}X\theta-\sqrt{W}y)^T(\sqrt{W}X\theta-\sqrt{W}y)\\ &=(X'\theta-y')^T(X'\theta-y') \text{ where } X'=\sqrt{W}X \text{ and } y'=\sqrt{W}y\end{aligned} \tag{6}$$

Now, equation 6 is similar to unweighted $J(\theta)$. Thus, using equation 5, θ in closed form is

$$\theta=\Big[(\sqrt{W}X)^T(\sqrt{W}X)\Big]^{-1}\Big(\sqrt{W}X\Big)^T\sqrt{W}y \tag{7}$$

1.3

Locally weighted linear regression is a non-parametric model. To estimate y, given x

- first calculate the weights $w^{(i)}$. This gives the matrix W as defined in equation 3
- Start with random guess for θ
- In the i$^{\text{th}}$ iteration of the algorithm, the θ is updated using the relation

$$\theta(i)\leftarrow\theta(i-1)-\alpha\Big(\sqrt{W}X\Big)^T\Big(\sqrt{W}X\theta(i-1)-\sqrt{W}y\Big) \tag{8}$$

- The above step is repeated until θ converges

2 Properties of the linear regression estimator

2.1

Vectors y,θ,ϵ and matrix X are related as

$$y=X\theta+\epsilon \tag{9}$$

ϵ is i.i.d $N(0,\sigma^2)$.

Thus, for the optimal value θ^* and fixed X

$$\begin{aligned}E[y]&=E[X\theta^*]+E[\epsilon]\\ &=E[X\theta^*]+0\\ &=X\theta^*\end{aligned} \tag{10}$$

Given the normal equations

$$\begin{aligned}\theta&=(X^T X)^{-1}X^T y\\ \implies E[\theta]&=(X^T X)^{-1}X^T E[y]\\ &=(X^T X)^{-1}X^T X\theta^*\\ &=\theta^*\end{aligned} \tag{11}$$

2.2

$$\begin{aligned} Var(\theta) &= E\big[(\theta - E(\theta))(\theta - E(\theta))^T\big] \\ &= E\big[(\theta - \theta^*)(\theta - \theta^*)^T\big] \\ &= E[\theta\theta^T] - \theta^*\theta^{*T} \end{aligned} \tag{12}$$

Use the normal equations (linear regression estimator) to get

$$\begin{aligned} E[\theta\theta^T] &= E\Big[(X^TX)^{-1}X^Tyy^T\big((X^TX)^{-1}\big)^T\Big] \\ &= (X^TX)^{-1}X^TE[yy^T]\big((X^TX)^{-1}\big)^T \end{aligned} \tag{13}$$

Use equation 9 to get

$$\begin{aligned} E[yy^T] &= E[(X\theta + \epsilon)(X\theta + \epsilon) \\ &= E[X\theta^*\theta^{*T}X^T + X\theta^*\epsilon^T + \epsilon\theta^{*T}X^T + \epsilon\epsilon^T] \\ &= X\theta^*\theta^{*T}X^T + E[\epsilon\epsilon^T] \\ &= X\theta^*\theta^{*T}X^T + diag(\sigma^2) \\ &= X\theta^*\theta^{*T}X^T + \sigma^2 I \end{aligned} \tag{14}$$

Equations 13 and 14 imply

$$\begin{aligned} E[\theta\theta^T] &= (X^TX)^{-1}X^T\Big[X\theta^*\theta^{*T}X^T + \sigma^2 I\Big]X\big((X^TX)^{-1}\big)^T \\ &= \Big[\theta^*\theta^{*T}X^T + (X^TX)^{-1}X^T\sigma^2 I\Big]X\big((X^TX)^{-1}\big)^T \\ &= \theta^*\theta^{*T} + (X^TX)^{-1}X^T\sigma^2 IX\big((X^TX)^{-1}\big)^T \\ &= \theta^*\theta^{*T} + \sigma^2(X^TX)^{-1}X^TX\big((X^TX)^{-1}\big)^T \\ &= \theta^*\theta^{*T} + \sigma^2(X^TX)^{-1} \end{aligned} \tag{15}$$

Equations 12 and 15 imply

$$Var(\theta) = \sigma^2(X^TX)^{-1} \tag{16}$$

3 Problem 3: Part 1: Implementing linear regression

Table 1: Files modified and plots generated for Problem3: Part1

Problem	function implemented	Files edited	Output and Plots
3.1.A1	`LinearReg_SquaredLoss.loss`	`linear_regressor.py`	Fig 1
3.1.A2	`LinearRegressor.train`	`linear_regressor.py` and `ex1.py`	Fig 2,Fig 3 and Fig 4
3.1.A3	`LinearRegressor.predict`	`linear_regressor.py` and `ex1.py`	
3.1.B1	`feature_normalize`	`utils.py`	
3.1.B2	`LinearRegressor.train` `LinearReg_SquaredLoss.loss`	`linear_regressor_multi.py`	Fig 5
3.1.B3		`ex1_multi.py`	
3.1.B4	`LinearRegressor_Multi.normal_equation`	`linear_regressor_multi.py` and `ex1_multi.py`	
3.1.B5		`ex1_multi.py`	Fig 6

3.1 Problem 3.1.A

- Implemented the `LinearReg_SquaredLoss.loss` and `LinearRegressor.train` methods in file `linear_regressor.py`.
- Add code in `ex1.py` to predict median home values at LSTAT = 5% and 50%.
- θ predicted: [34.55363411,−0.95003694]
- Median home values predicted:
  LSTAT = 5%: 298034.494122
  LSTAT = 50%: -129482.128898

3.2 Problem 3.1.B

- Implemented the `feature_normalize` method in file `utils.py` to normalize the data.
- Implemented the `LinearRegressor.train` and `LinearReg_SquaredLoss.loss` methods in file `linear_regressor_multi.py`.
- Add code in `ex1_multi.py` to predict median home values at average home.
- Add code in `ex1_multi.py` to explore convergence of gradient descent.
- For average home in Boston suburbs, we predict a median home value of 225328.063241 using gradient descent.
- For average home in Boston suburbs, we predict a median home value of 225328.06324113 using normal equation.

- When learning rate ¿=1, the loss function could not convergent. So $alpha = 0.3$ is a good learning rate and 10 iteration is enough for that.

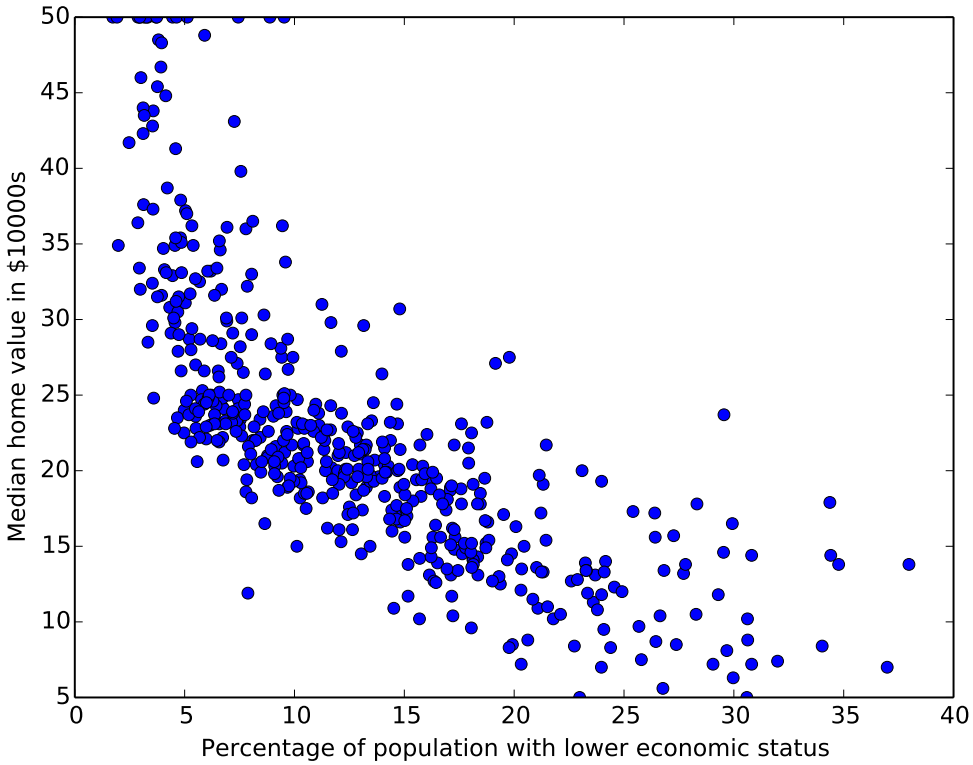


Figure 1: The training data for linear regression.

4 Problem 3: Part 2: Implementing regularized linear regression

Table 2: Files modified and plots generated for Problem3: Part2

Problem	function implemented	Files edited	Output and Plots
3.2.A1	`RegularizedLinearReg_SquaredLoss.loss`	`reg_linear_regressor_multi.py`	Fig 7 and Fig 8
3.2.A2	`RegularizedLinearReg_SquaredLoss.grad_loss`	`reg_linear_regressor_multi.py`	Fig 7 and Fig 8
3.2.A3	`learning_curve` and `feature_normalize`	`utils.py`	Fig 9, Fig 10 and Fig 1
3.2.A4	-	`ex2.py` added code for λ=1,10,100	Fig 12
3.2.A5	`validation_curve`	`utils.py`	Fig 13
3.2.A6	-	`ex2.py` added code for λ=1 from prob3.2.A5	`Error=3.0987791808`
3.2.A7	`averaged_learning_curve`	`utils.py`	Fig 14

4.1 Problem 3.2.A1 and Problem 3.2.A2:

- Implemented the `RegularizedLinearReg_SquaredLoss.loss` and `RegularizedLinearReg_SquaredLoss.grad_loss` methods in file `reg_linear_regressor_multi.py` per requirements of the problem.

The updated methods use regularized regression to calculate the loss and gradient. The output from `ex2.py` is shown in Fig 7 and Fig 8.

4.2 Problem 3.2.A3

- Implemented the `learning_curve` function in file `utils.py` per the requirements of the problem. The output from `ex2.py` is shown in Fig 9
- Implemented the `feature_normalize` function in file `utils.py`. The output from `ex2.py` is shown in Fig 10 and Fig 11

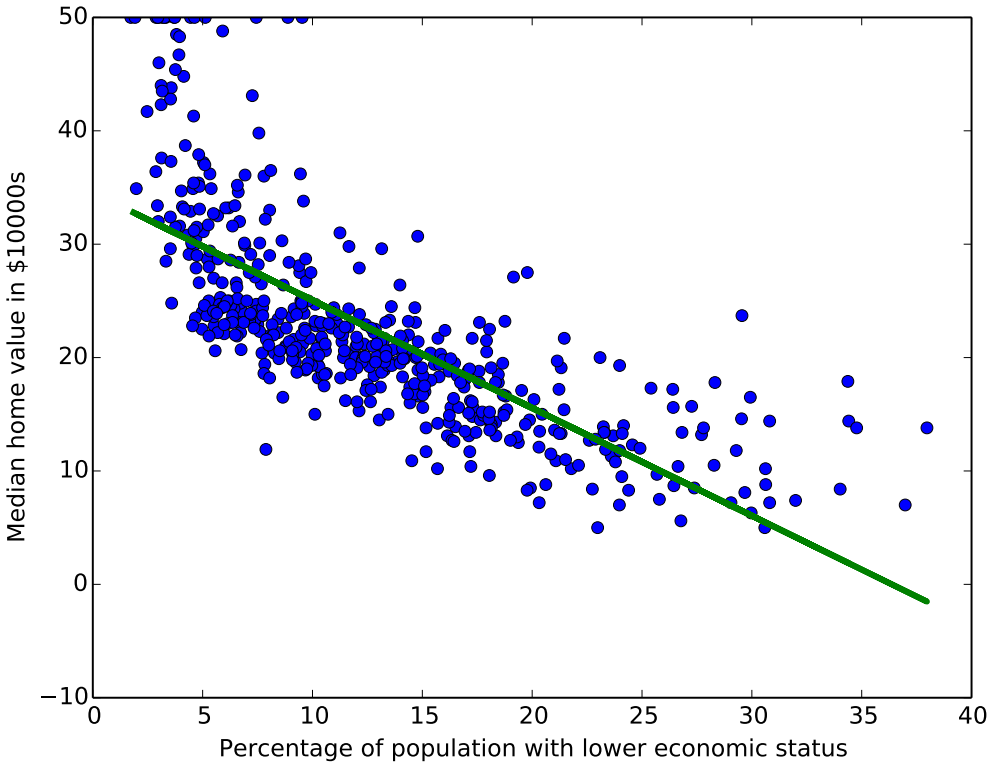


Figure 2: Fitting a linear model to the data in Fig 1

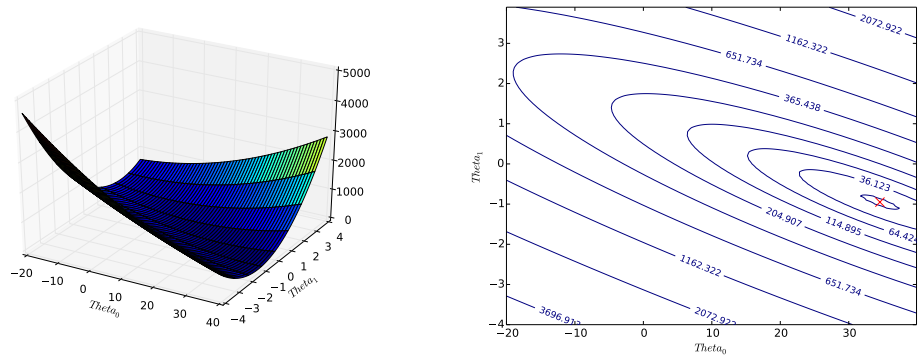


Figure 3: Surface and contour plot of cost function

4.3 Problem 3.2.A4

- The results for $\lambda = 1.0$ are shown in Fig 12a and Fig 12b.
- The results for $\lambda = 10.0$ are shown in Fig 12c and Fig 12d.
- The results for $\lambda = 100.0$ are shown in Fig 12e and Fig 12f.
  We observe that low value of the regularization parameter λ is strongly biased while a large value for λ has high variance.

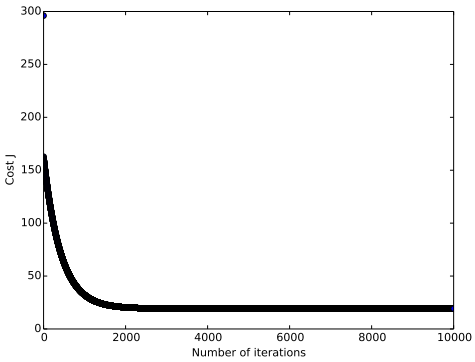


Figure 4: Convergence of gradient descent for linear regression with single variables

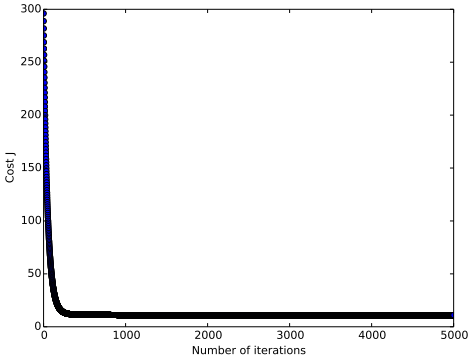


Figure 5: Convergence of gradient descent for linear regression with multi variables

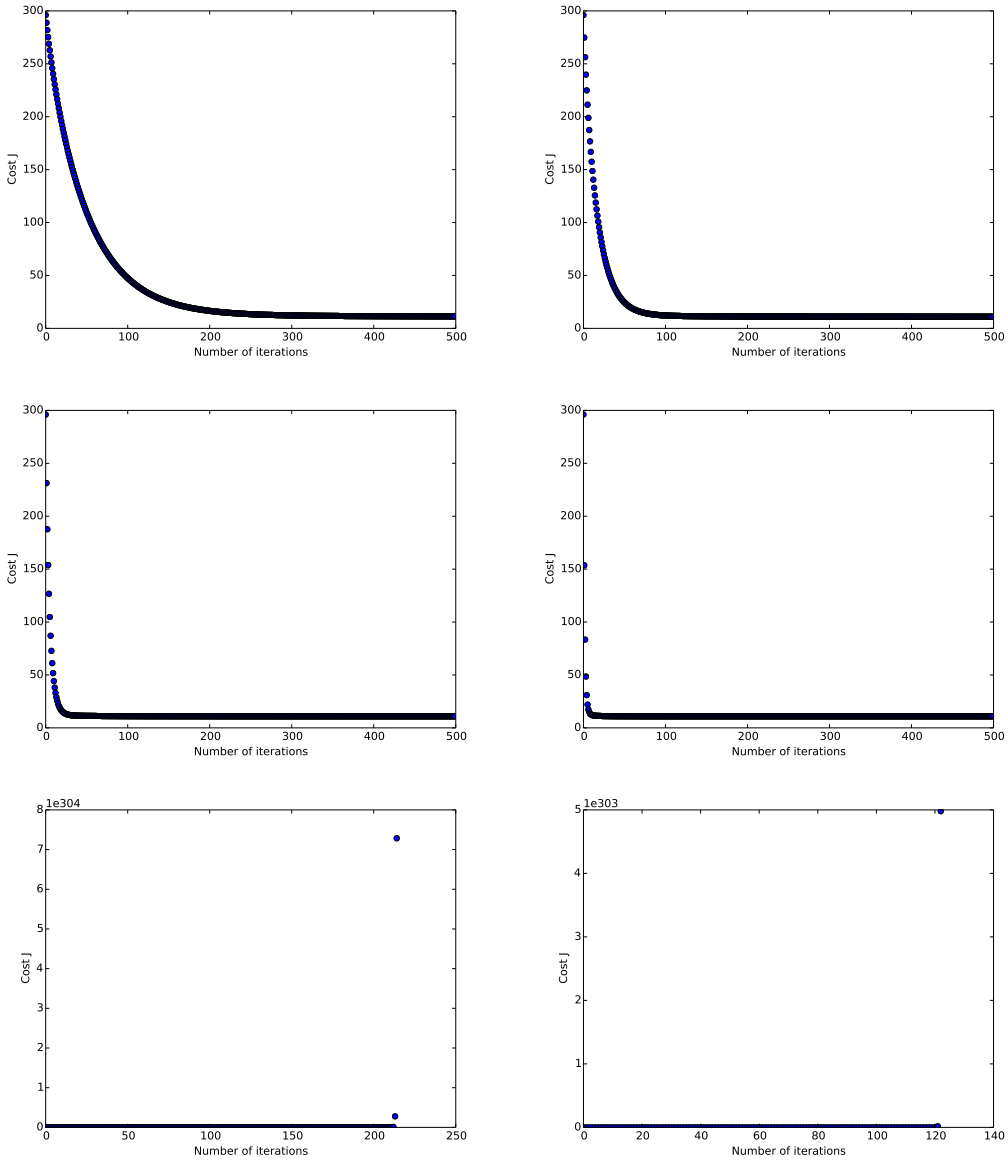


Figure 6: Convergence of gradient descent for linear regression with multi variables at different learning rate: $alpha$=0.01,0.03,0.1,0.3,1,3

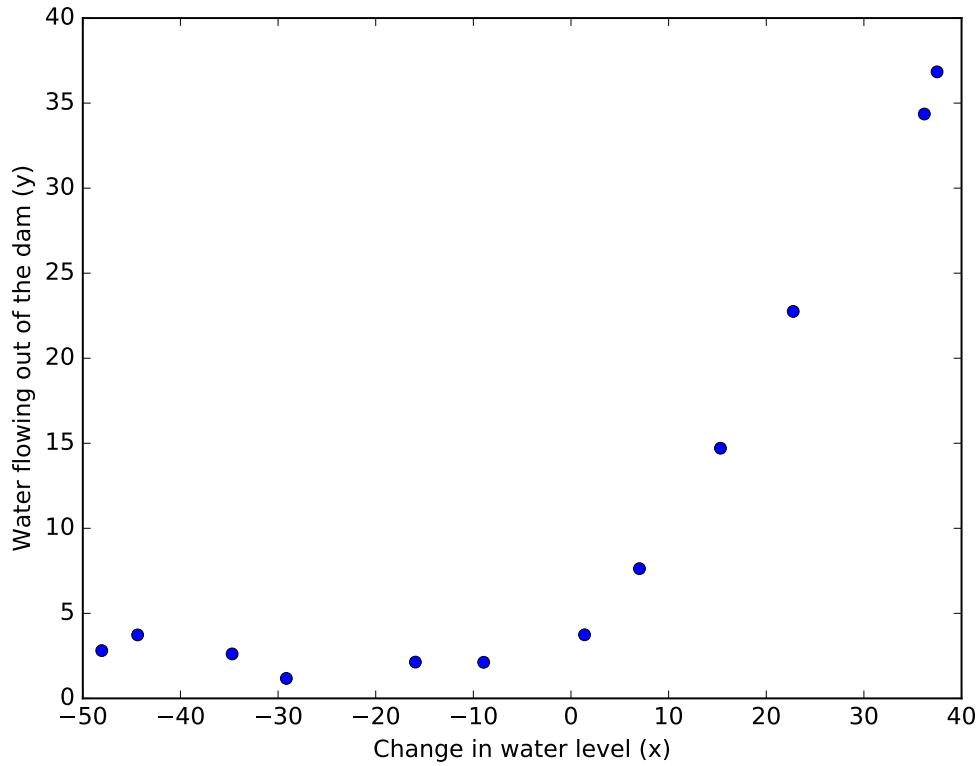


Figure 7: The training data for regularized linear regression.

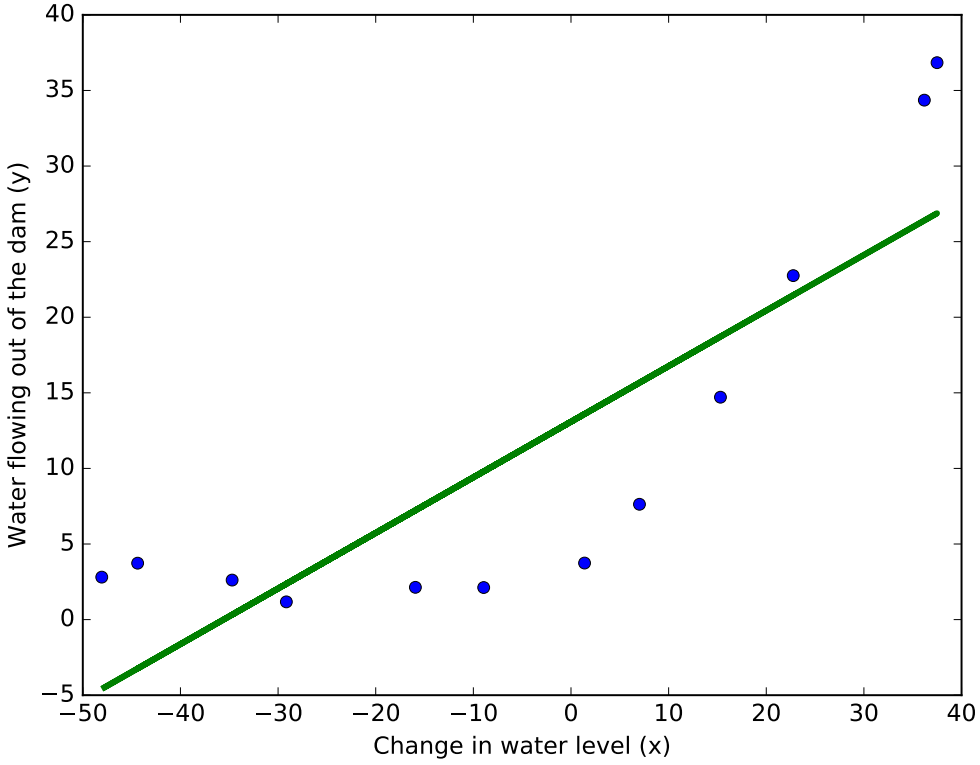


Figure 8: The best fit line for the training data.

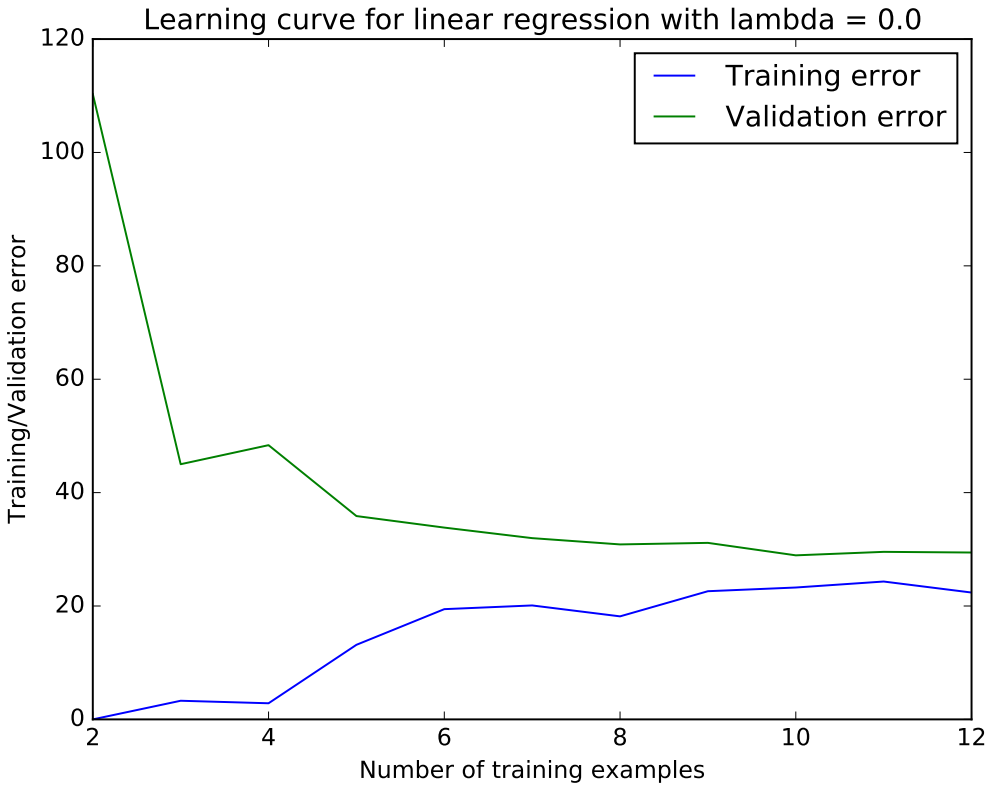


Figure 9: Learning curves.

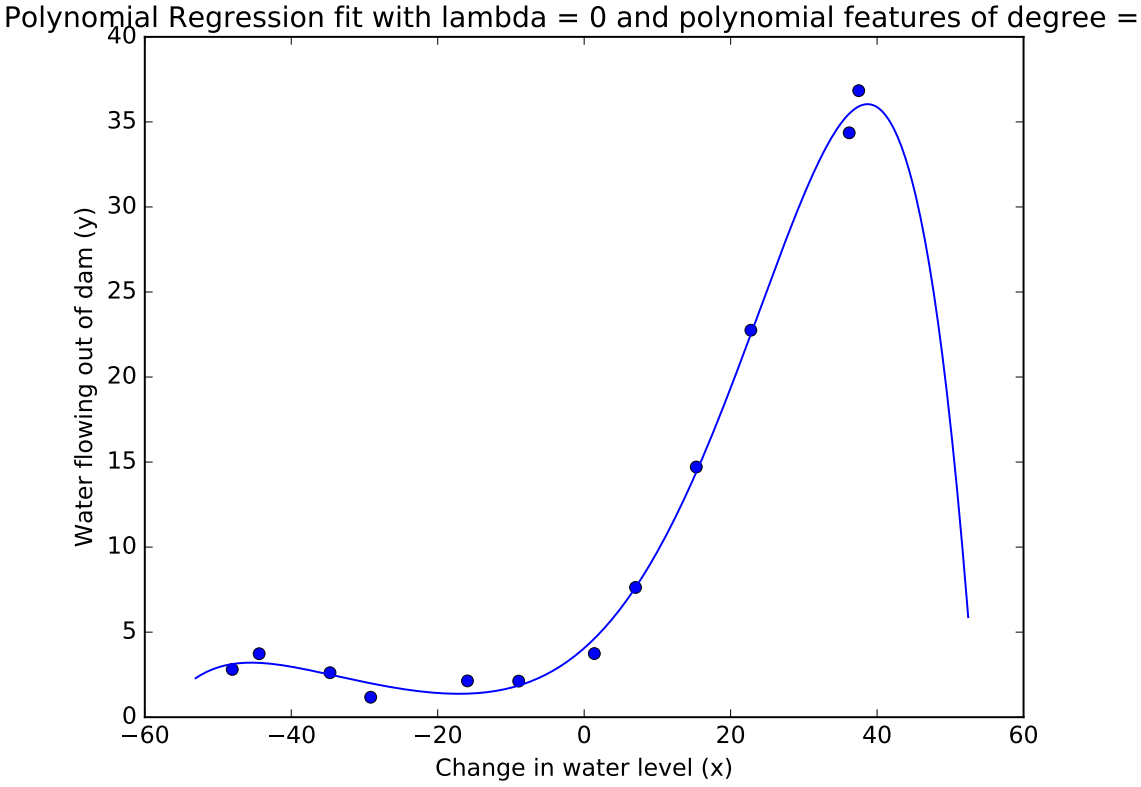


Figure 10: Polynomial fit for lambda = 0 with a p=8 order model.

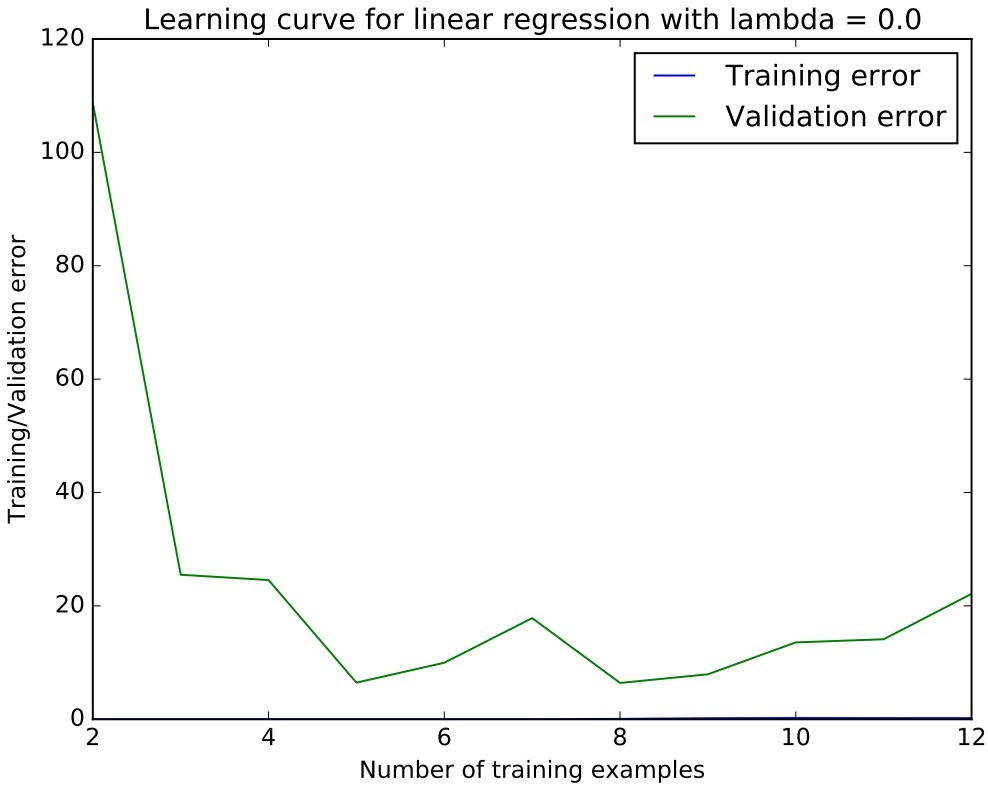
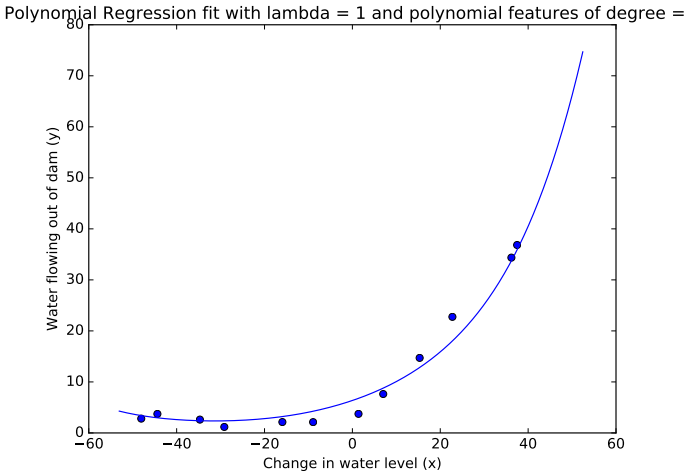
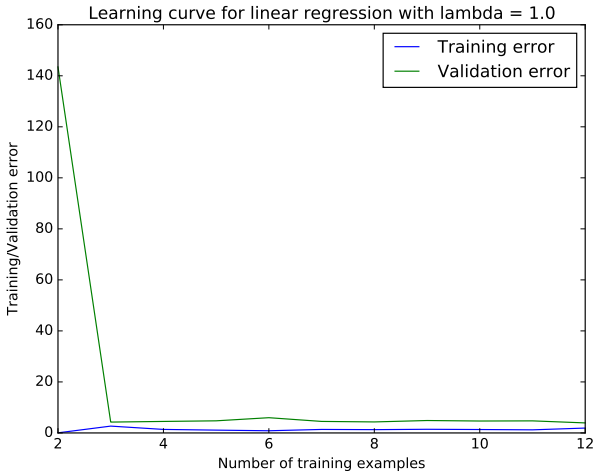


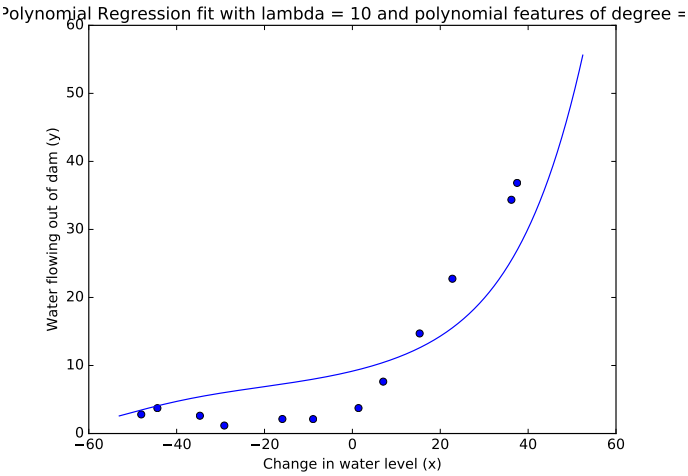
Figure 11: Learning curve for lambda = 0.



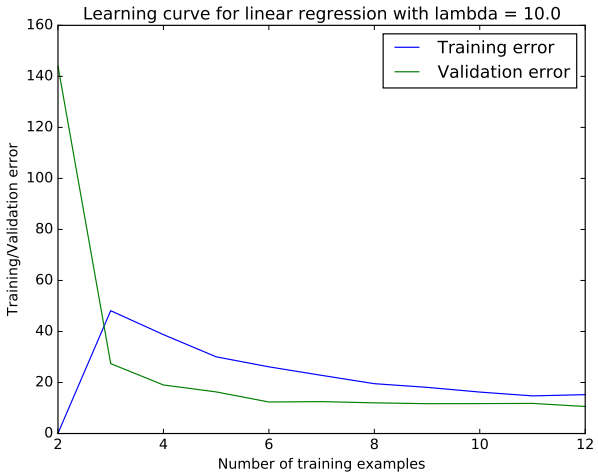
(a) Polynomial fit for lambda = 1 with a p=8 order model.



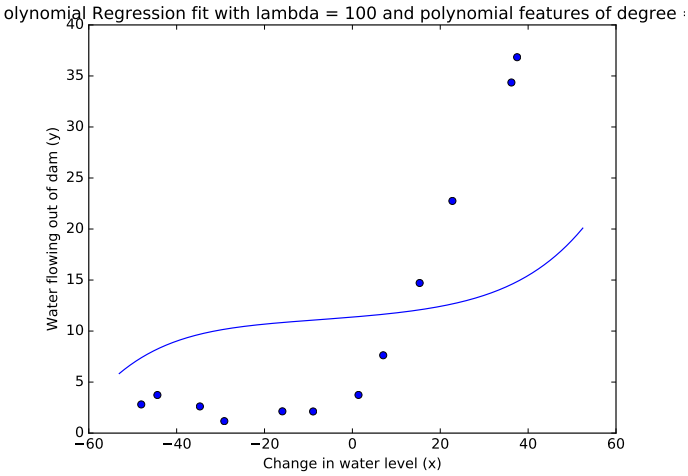
(b) Learning curve for lambda = 1.



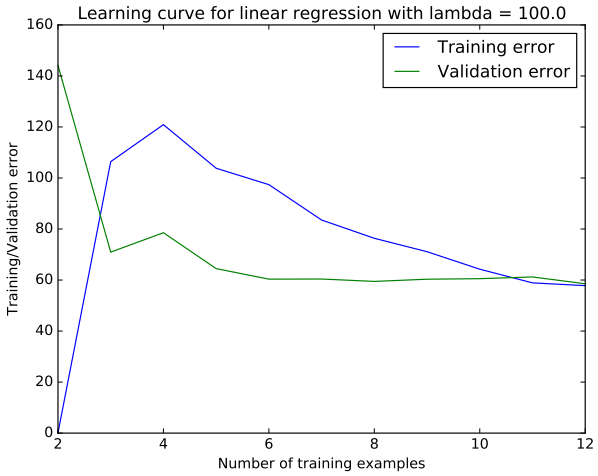
(c) Polynomial fit for lambda = 10 with a p=8 order model.



(d) Learning curve for lambda = 10.



(e) Polynomial fit for lambda = 100 with a p=8 order model.



(f) Learning curve for lambda = 100.

Figure 12: Polynomial fits and learning curves for different lambda

4.4 Problem 3.2.A5

• Completed the `validation_curve` function in the file `utils.py`. The results from `ex2.py` are shown in Fig 13.
Due to randomness in the training and validation splits of the dataset, the cross validation error can sometimes be lower than the training error. We observe that value of $\lambda = 1$ is a good choice for the regularization parameter.

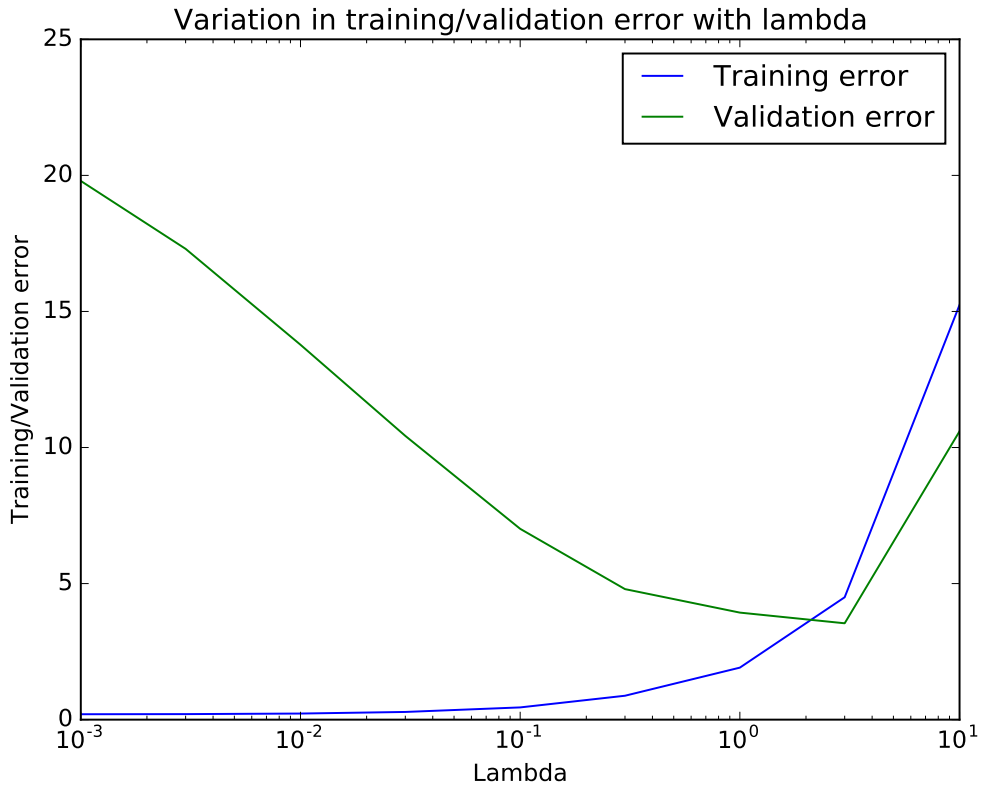


Figure 13: Variation in training/validation error with lambda.

4.5 Problem 3.2.A6

• Per the requirements, added code to `ex2.py`. This uses the best model parameters from section 4.4 and evaluates the error on test data. We picked a value of about $\lambda = 1.0$ for the best model. The output is

```
Theta at lambda = 1.0 (problem3.2.A6) is  [ 11.21759608   8.38067931   5.21899733   3.62613424   2.11030661
   1.95470373   0.78523708]
Error at lambda = 1.0 (problem3.2.A6) is  3.0987791808
```

4.6 Problem 3.2.A7

• Completed the `averaged_learning_curve` function in the file `utils.py` per the requirements of the problem.
The output from `ex2.py` is plotted in Fig 14

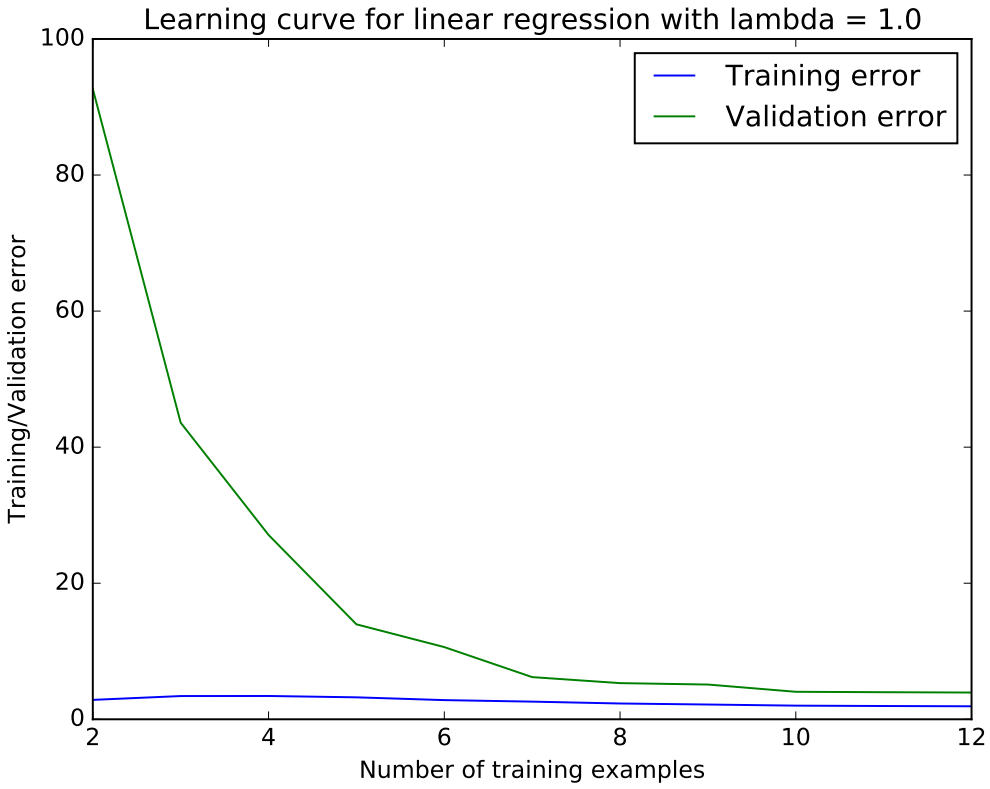


Figure 14: Averaged learning curves for lambda=1.