comp540 Homework1

Wanyi Ye (wy13), Xiaoye Steven Sun (xs6)

January 19th 2018

0 Background refresher

1 Locally Weighted Liner Regression

See attached pages.

2 Properties of the linear regression estimator

See attached pages.

3 Regression

3.1 Liner Regression

3.1.1 Liner Regression with one variable

A1. Computing the cost function. see `linear_regressor.py`

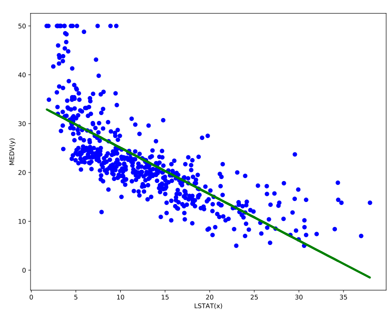
A2. Implementing Gradient descent. see `linear_regressor.py` and Figure 1

A3. Predicting on unseen data. see `linear_regressor.py` and `ex1.ipynb`
For lower status percentage = 5, we predict a median home value of **29.8034494122**
For lower status percentage = 50, we predict a median home value of **-12.9482128898**

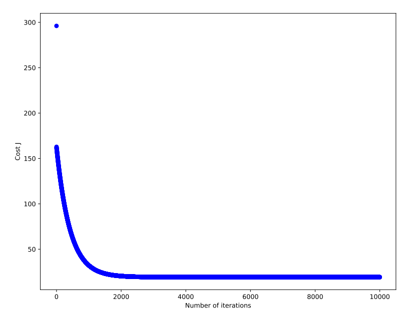
3.1.2 Liner Regression with multi variables

B1. Feature normalization. see `feature_normalize()` in `utils.py`

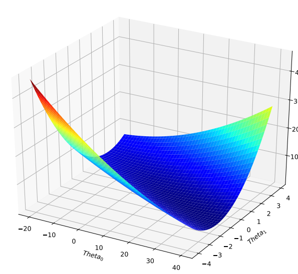
B2. Loss function and gradient descent. see `linear_regressor_multi.py`



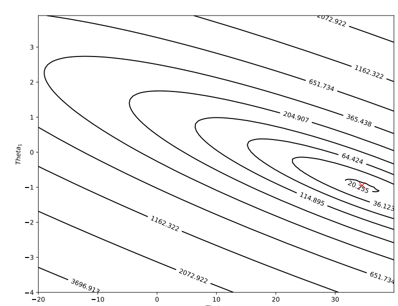
(a) Fitting line



(b) J



(c) Surface



(d) Contour

Figure 1: Gradient descent

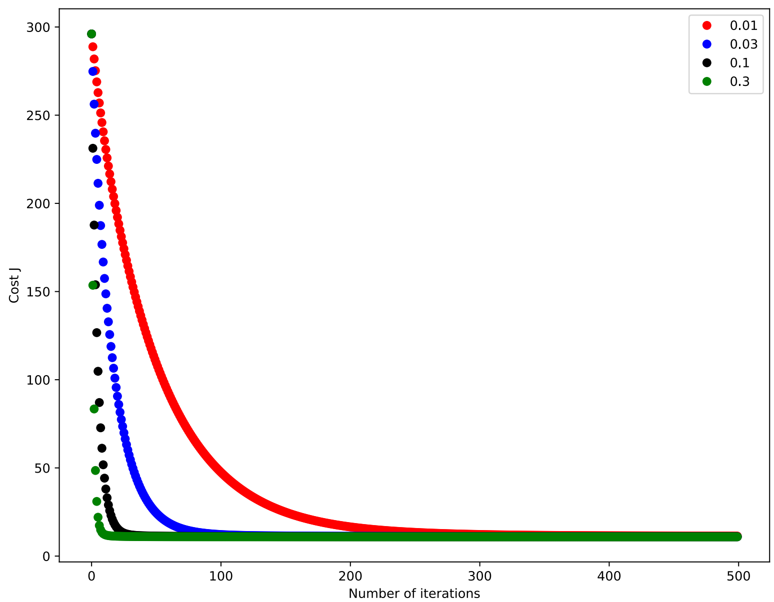


Figure 2: J as a function of the number of iterations for different learning rates

B3. Making predictions on unseen data. see `ex1_multi.ipynb`. For average home in Boston suburbs, we predict a median home value of **22.5328063241**.

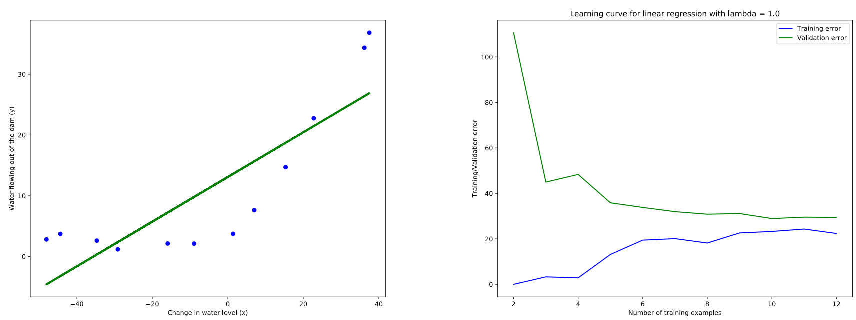
B4. Normal equations. see `linear_regressor_multi.py` and `ex1_multi.ipynb`. For average home in Boston suburbs, we predict a median home value of **22.5328063241**. Yes. The result matches the one in B3.

B5. Exploring convergence of gradient descent. The learning rates we tried are 0.01, 0.03, 0.1, 0.3. We found that J converges the fastest when the learning rate is 0.3 (see Figure. 2). It takes about 30 iterations to converge. When the learning rate is larger than 0.3, J blows up. When the learning rate is small, it takes more iterations to converge. For example, when the learning rate is 0.01, it converges at about 300 iterations.

3.2 Regularized Liner Regression

A1. Regularized liner regression cost function. see `reg_linear_regressor_multi.py`.

A2. Gradient of the regularized liner regression cost function. see `reg_linear_regressor_multi.py`



(a) Fitting line

(b) Learning curve

Figure 3: λ=0

and Figure 3a.

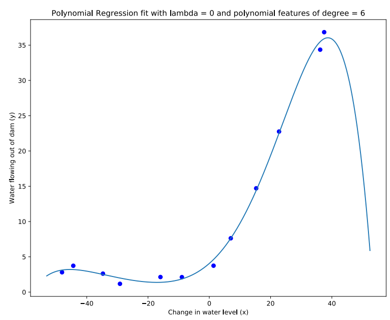
A3. Learning curves. see `utils.py`, Figure 3b and 4.

A4. Adjusting the regularization parameter. Figure 5, 6, 7 show the fit line and the learning curve with λ=1, 10, 100 respectively. When λ=1, it shows the lowest validation error among all four λ values (including λ=0). This means that when λ=1, it achieves a good trade-off between bias and variance. When λ is larger than 10, the regularization term is too large so that the variance is too low and the bias is to large. When λ is less than 1, the resulting model tends to over-fitting the training data.

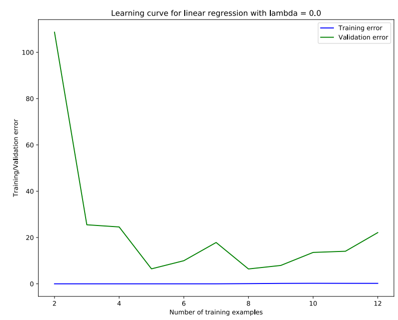
A5. Selecting λ using a validation set. see `utils.py` and Figure 8. As we can see from Figure 8, the λ that achieves the minimum validation error is 3.

A6. Computing test set error. See `ex2.ipynb`. Results: testing error: 4.39762337668 @ reg= 3.0

A7. Plotting learning curves with randomly selected examples. See `utils.py` and Figure 9.

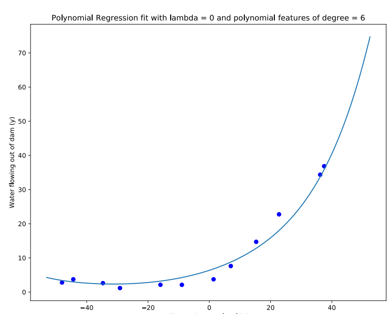


(a) Fitting line

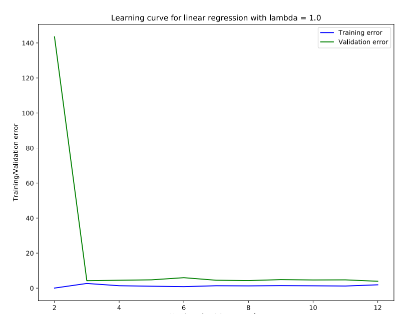


(b) Learning curve

Figure 4: λ=0, polynomial

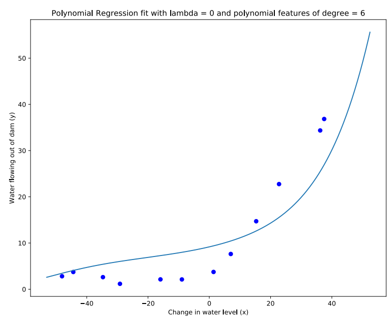


(a) Fitting line

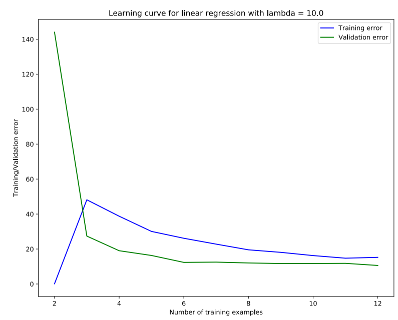


(b) Learning curve

Figure 5: λ=1

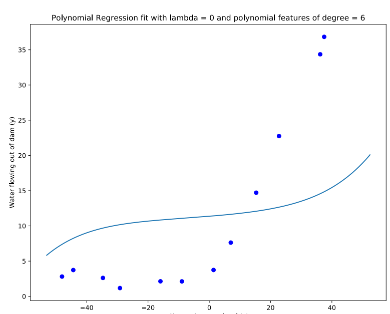


(a) Fitting line

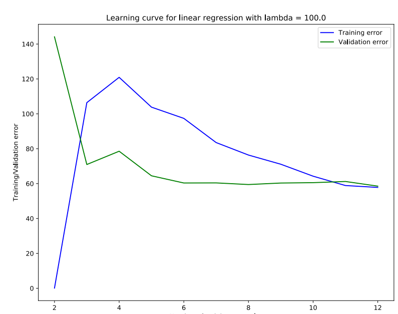


(b) Learning curve

Figure 6: λ=10



(a) Fitting line



(b) Learning curve

Figure 7: λ=100

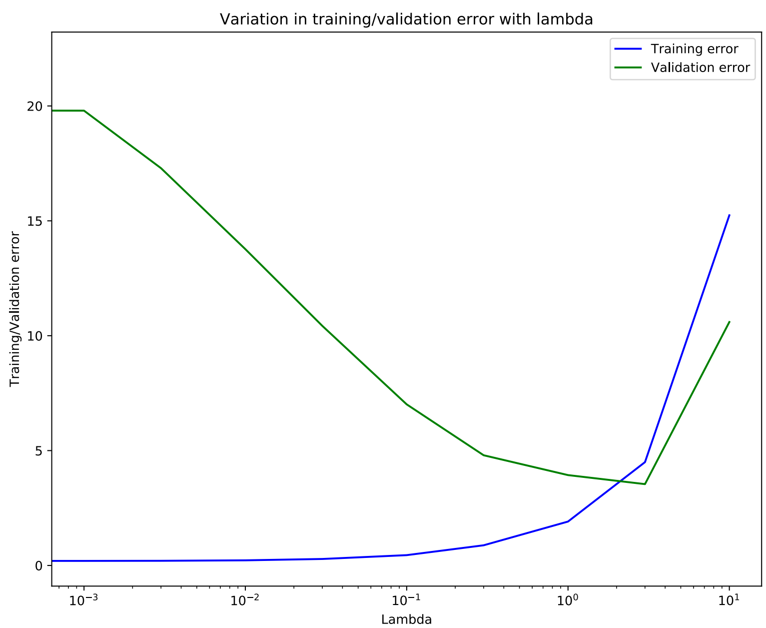


Figure 8: Training and testing error with different λ

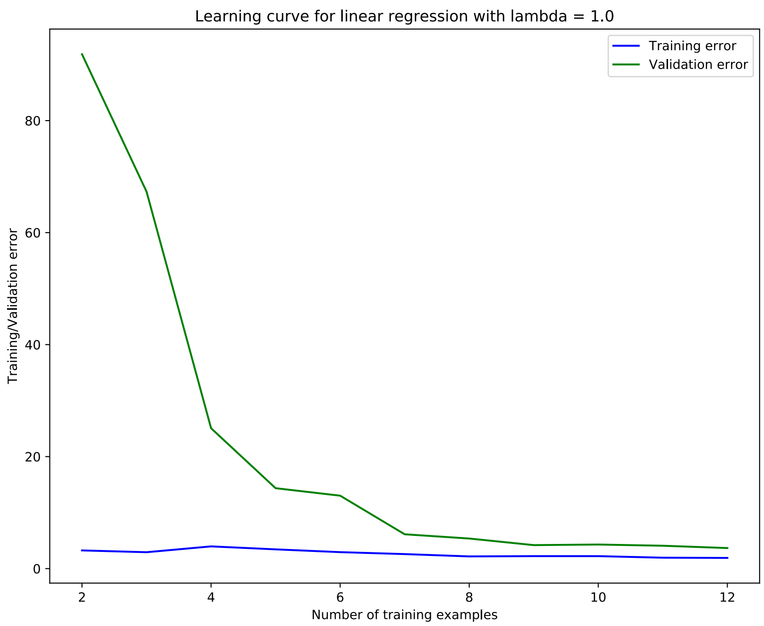


Figure 9: Averaged learning curve λ=1

COMP540. HW1

1.1

$X_{m\times d}$. m is number of samples.

$y_{m\times 1}$ d is the dimension

Define $W \triangleq diag(w_1, \dots w_m)$

Let $\underset{m\times 1}{A} = X\theta - y$ $\underset{m\times 1}{A} = \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix}$

So, we have $J(\theta) = A^T W A = A^T diag(w_1, \dots, w_m) A$

$$= a_1^2 w_1 + \dots + a_m^2 w_m = \sum_{i=1}^{m} w_i a_i^2$$

$$a_i = \sum_{j=1}^{d} X_{id} \cdot \theta_d - y_i$$

$$= \theta^T \cdot x^{(i)} - y_i$$

So. $J(\theta) = \sum_{i=1}^{m} w_i (\theta^T x^{(i)} - y^{(i)})^2$

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

Let $2w_i = w^{(i)}$. The $W = diag(\frac{w^{(1)}}{2}, \dots \frac{w^{(m)}}{2})$ #

COMP 540 HW1

1.2. $J(\theta) = (X\theta - y)^T W (X\theta - y)$

$$= (X\theta)^T W X\theta - (X\theta)^T W y - y^T W X\theta + y^T W y.$$

$$= \theta^T X^T W X\theta - \theta^T X^T W y - y^T W X\theta + y^T W y$$

$$\frac{\partial J(\theta)}{\partial \theta} = 2 \cdot \theta^T X^T W X - y^T W^T X - y^T W X$$

$$= 2\theta^T X^T W X - 2 y^T W X = 0$$

$$\Rightarrow \quad \theta^T X^T W X = y^T W X$$

$$X^T W^T X \theta = \cancel{y^T W X} \; X^T W^T y$$

$$\theta = (X^T W X)^{-1} X^T W y.$$

#

COMP 540 HW 1

1.3.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \begin{cases} \frac{1}{2} \sum_{i=1}^{m} w^{(i)} \cdot 2 \cdot \left(\sum_{k=1}^{d} (\theta_k x_k^{(i)}) - y^{(i)} \right) x_j^{(i)} & j \neq 0. \\ \frac{1}{2} \sum_{i=1}^{m} w^{(i)} \cdot 2 \cdot \left(\sum_{k=1}^{d} (\theta_k x_k^{(i)}) - y^{(i)} \right). & j = 0. \end{cases}$$

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{m} w^{(i)} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)} \qquad j \neq 0$$

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{m} w^{(i)} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \qquad j = 0$$

$$x_0^{(i)} = 1$$

input is $\vec{x}$

start with a random $\vec{\theta}$

while (iter < 10000)

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{m} w^{(i)} \left(\sum_{k=0}^{d} (\theta_k x_k^{(i)}) - y^{(i)} \right) x_j^{(i)} \qquad j \neq 0$$

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{m} w^{(i)} \left(\sum_{k=0}^{d} (\theta_k x_k^{(i)}) - y^{(i)} \right) \qquad j = 0$$

non-parametric, since the # of parametric depends on the # of input data.

COMP 540 HW1

2.1

$$y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{pmatrix} \quad X = \begin{bmatrix} - x^{(1)} - \\ - x^{(2)} - \\ \vdots \\ - x^{(i)} - \\ \vdots \\ - x^{(m)} - \end{bmatrix} \quad \epsilon = \begin{pmatrix} \epsilon^{(1)} \\ \epsilon^{(2)} \\ \vdots \\ \epsilon^{(i)} \\ \vdots \\ \epsilon^{(m)} \end{pmatrix}$$

$\because \theta = (X^T X)^{-1} X^T y.$ for Least Square Estimator

$y = X\theta^* + \epsilon$

$$\therefore E[\theta] = E[(X^T X)^{-1} X^T y] = E[(X^T X)^{-1} X^T (X\theta^* + \epsilon)]$$

$$= E[(X^T X)^{-1} X^T X \theta^*] + E[(X^T X)^{-1} X^T \epsilon]$$

$$= \theta^* + (X^T X)^{-1} X^T E[\epsilon] = \theta^*$$

#

COMP 540 HW1

2.2

$$\mathrm{Var}[\theta] = \mathrm{Var}[(X^TX)^{-1}X^T(X\theta^* + \epsilon)]$$

$$= \mathrm{Var}[(X^TX)^{-1}X^T\epsilon]$$

$$= (X^TX)^{-1}X^T \mathrm{Var}(\epsilon)((X^TX)^{-1}X^T)^T$$

$$= (X^TX)^{-1}X^T \sigma^2 I \quad X(X^TX)^{-1^T}$$

$$= \sigma^2(X^TX)^{-1}X^TX(X^TX)^{-1}$$

$$= (X^TX)^{-1}\sigma^2$$