COMP540 STATISTICAL MACHINE LEARNING HW01

Yiqing Lu, Chuanwenjie Wei

January 23, 2017

1. Background refresher

1)

Since the test sample is random, the results of the probability generated by *sampler.py* are always different. The probability that a sample from this distribution lies within the unit circle centered at (0.1, 0.2) lies in [0.17, 0.19].

2)

suppose $X \sim Poi\{x|\lambda\}$, $Y \sim Poi\{y|\mu\}$

$$\begin{aligned} P\{Z=z\} &= P\{X+Y=z\} = -\sum_{x=0}^{z} P\{X=x\}P\{Y=z-x\} \\ &= \sum_{i=0}^{z} e^{-\lambda}\frac{\lambda^x}{x!}e^{-\mu}\frac{\mu^{z-x}}{(z-x)!} \\ &= e^{-(\lambda+\mu)}\sum_{i=0}^{z} e^{-\lambda}\frac{\lambda^x\mu^{z-x}C_z^x}{z!} \\ &= \frac{e^{-(\lambda+\mu)}}{z!}(\lambda+\mu)^z \end{aligned}$$

So z=x+y is also a Poisson random variable.

3)

According to total probability

$$\begin{aligned} P\{X_1=x_1\} &= \int P\{X_1=x_1|X_0=x_0\}P(X_0=x_0)dx_0 \\ &= \int \alpha_0 e^{-\frac{(x_0-\mu_0)^2}{2\delta_0^2}}\alpha_1 e^{-\frac{(x_1-x_0)^2}{2\delta^2}}dx_0 \end{aligned}$$

$$= \quad \alpha_0\alpha_1 \int exp[(-\frac{1}{2\delta_0^2} - \frac{1}{2\delta^2})x_0^2 + (\frac{\mu_0}{\delta_0^2} + \frac{x_1}{\delta^2})x_0 + (-\frac{\mu_0^2}{2\delta_0^2} - \frac{x_1^2}{2\delta^2})]dx_0$$

$$= \quad \alpha_0\alpha_1 exp(-\frac{(x_1-\mu_0)^2}{2(\delta_0^2+\delta^2)}) \int exp[-\frac{\delta_0^2+\delta^2}{2\delta_0^2\delta^2}(x_0 - \frac{(\mu_0\delta^2 + x_1\delta_0^2)}{(\delta_0^2+\delta^2)})^2 dx_0$$

And because it should be a Gaussian form, suppose the normalization constant for this is α', the expression could reduce to

$$P\{X_1 = x_1\} \quad = \quad \frac{\alpha_0\alpha_1}{\alpha'} exp(-\frac{(x_1-\mu_0)^2}{2(\delta_0^2+\delta^2)})$$

So it's a Gaussian distribution with mean $\mu_1 = \mu_0$and variance$\delta_1^2 = \delta^2 + \delta_0^2$

Since P is a probability distribution and is already normalized, the normalization coefficient $\frac{\alpha_0\alpha_1}{\alpha'}$is just required for a Gaussian of the given variance. For example, $\alpha = \frac{1}{\sqrt{2\pi(\delta^2+\delta_0^2)}}$.

4)

$$|A - \lambda I| \quad = \quad \left| \begin{array}{cc} 13-\lambda & 5 \\ 2 & 4-\lambda \end{array} \right| = (13-\lambda)(4-\lambda) - 10 = 0$$

$$\lambda^2 - 17\lambda + 42 = 0$$

$$\lambda_1 = 3, \quad \lambda_2 = 14$$

$$\lambda_1 = 3 \Longrightarrow \quad \left\{ \begin{array}{c} 10x_1 + 5x_2 = 0 \\ 2x_1 + x_2 = 0 \end{array} \right. \quad \Longrightarrow \quad eigenvector = \left(\begin{array}{cc} \frac{1}{\sqrt{5}}, & -\frac{2}{\sqrt{5}} \end{array} \right)$$

$$\lambda_1 = 14 \Longrightarrow \quad \left\{ \begin{array}{c} -x_1 + 5x_2 = 0 \\ 2x_1 - 10x_2 = 0 \end{array} \right. \quad \Longrightarrow \quad eigenvector = \left(\begin{array}{cc} \frac{5}{\sqrt{26}}, & \frac{1}{\sqrt{26}} \end{array} \right)$$

5)

a) if

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad , \quad B = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\Longrightarrow \quad AB \neq BA \Longrightarrow \quad (A+B)^2 \neq A^2 + 2AB + B^2$$

b) if

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad , \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\Longrightarrow AB = 0, \quad A \neq 0, \quad B \neq 0$$

6)

$$uu^T = I \quad \Longrightarrow \quad u^T u = I$$

$$A^T A \quad = \quad = (I - 2uu^T)^T (I - 2uu^T) = I - 2(uu^T)^T - 2uu^T + 4(uu^T)^T (uu^T) = I$$

7)

a)

$$f^{''}(x) \quad = \quad 6x$$

Because $x \geq 0 \Longrightarrow f^{''}(x) \geq 0$, $f(x) = x^3$is convex.

b)

$$f^{''}(x_1, x_2) \quad = \quad 0$$

$f(x_1, x_2) = max(x_1, x_2)$ are convex on S.

c)

Because f and g are convex on S, then $f^{''}(x) \geq 0$, $g^{''}(x) \geq 0$

$$f^{''}(x) + g^{''}(x) \quad \geq \quad 0$$

$f(x) + g(x)$ are convex on S.

d)

Because f and g are convex and non-negative on S, then $f^{''}(x) \geq 0$, $g^{''}(x) \geq 0$, $f(x) \geq 0$, $g(x) \geq 0$

Since f and g have their minimum within S at the same point, suppose the common minimum number isx^* if $x < x^*$,then $f(x) \leq 0$, $g(x) \leq 0$, $\Rightarrow f^{'}g^{'}(x) \geq 0$
if $x > x^*$,then $f(x) \geq 0$, $g(x) \geq 0$, $\Rightarrow f^{'}g^{'}(x) \geq 0$

$$\Rightarrow fg^{''}(x) \quad = \quad (fg^{'} + f^{'}g)^{'}(x) = (f^{''}g(x) + fg^{''}(x) + 2f^{'}g^{'}(x)) \geq 0$$

$fg(x)$ are convex on S.

8)

To use the method of Lagrange multipliers, we can rewrite the function with a constraint $\sum_{i=1}^{K} p_i - 1 = 0$

$$\Lambda \quad = \quad -\sum_{i=1}^{K} p_i log\,(p_i) + \lambda \left(\sum_{i=1}^{K} p_i - 1 \right)$$

Let

$$\frac{\partial \Lambda}{\partial p_i} \quad = \quad 0$$

We can get

$$log\,(p_i) \quad = \quad \lambda - 1$$

So all the p_i has the same value. Since$\sum_{i=1}^{K} p_i = 1$, $p_i = \frac{1}{K}$

The ponit $p^* = p_i$ is unique, and we also know that $H\,(p^*) = logK \geq 0$, so p^* is a maximum.

2. Locally weighted linear regression

1)

Let

$$Z \quad = \quad X\theta - y$$

where

$$z_i \quad = \quad \theta^T x^{(i)} - y^{(i)}$$

And let

$$W_{ii} = \frac{1}{2}w^{(i)},\ W_{ij} = 0\ for\ any\ i \neq j$$

Then we can derive

$$\begin{aligned}(X\theta - y)^T W (X\theta - y) &= Z^T W Z \\ &= \frac{1}{2}\sum_{i=1}^{m} w^{(i)} z_i^2 \\ &= \frac{1}{2}\sum_{i=1}^{m} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2 \\ &= J(\theta)\end{aligned}$$

2)

Since

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

So

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial\left(\theta^T X^T W X\theta - 2y^T W y - 2y^T W X\theta\right)}{\partial\theta} \\ &= 2\left(X^T W X\theta - X^T W y\right)\end{aligned}$$

In order to minimize $J(\theta)$, we set $\frac{\partial J(\theta)}{\partial\theta} = 0$, so we can get

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

Therefore,

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

3)

We use R to write down the algorithm for calculating parameters by batch gradient descent for locally weighted linear regression.

It's a parametric method.

```
X=data.frmae(rep(1,nrow(x)),x)
m=length(x); W=matrix(0,m,m)
regr.x = seq(from min(x),to max(x),by=.1);regr.y=rep(0,length(y))

for (k in 1:length(regr.x)){
    for (l in 1:m){
    W[l,l]=exp(-(1/2)*((regr.x[k]-x[l])**2)/(tau**2))
}
    theta = solve(t(X)%*%W%*%X)%*%t(X)%*%W%*%y

   regr.y[k] = theta[,2]*regr.x[k] + theta[,1]
}
```

3. Properties of the linear regression estimator

1)

Let

$$X = (x_1, x_2 \dots x_m)^T,\ Y = (y_1, y_2 \dots y_m)^T$$

So we can rewrite the regression model

$$Y = X\theta^* + \varepsilon$$

Since θ is the least squares estimator, So $\theta = \left(X^T X\right)^{-1} X^T Y$

$$E(\theta) = \left(X^T X\right)^{-1} X^T E(Y)$$

Since

$$E(Y) = X\theta^*$$

So

$$E(\theta) = \theta^*$$

2)

As mentioned above, $\theta = \left(X^T X\right)^{-1} X^T Y$

So

$$Var(\theta) = \left(X^T X\right)^{-1} X^T Cov(Y) \left(\left(X^T X\right)^{-1} X^T\right)^T$$

Since $Cov(Y) = \sigma^2$,

So

$$Var(\theta) = \sigma^2 \left(X^T X\right)^{-1} X^T X \left(\left(X^T X\right)^{-1}\right)^T$$

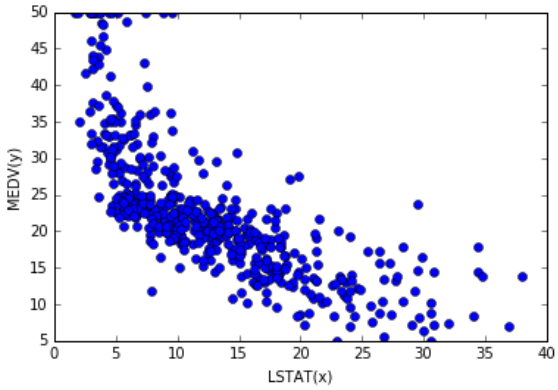
$$= \sigma^2 \left(X^T X\right)^{-1}$$

4. Implementing linear regression and regularized linear regression

4.1 Implementing linear regression

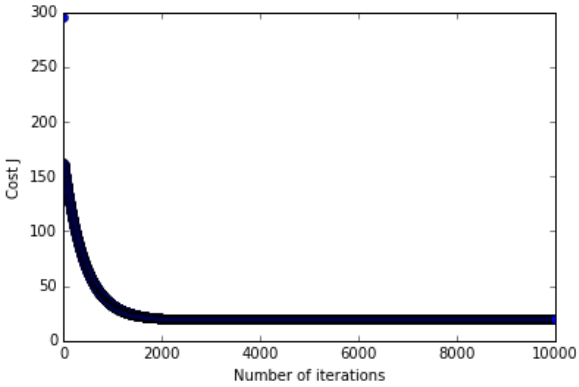
4.1.1 Linear regression with one variable

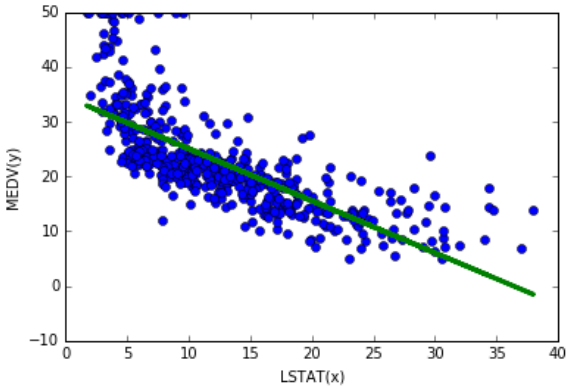
- **Plotting The Data**



Before starting on any task, it is often useful to understand the data by visualizing it. For this dataset, we can see from the plot above that there exists a negative relationship between the percentage of population of lower economic status and the median home value.

4.1.1.1 Computing the cost function



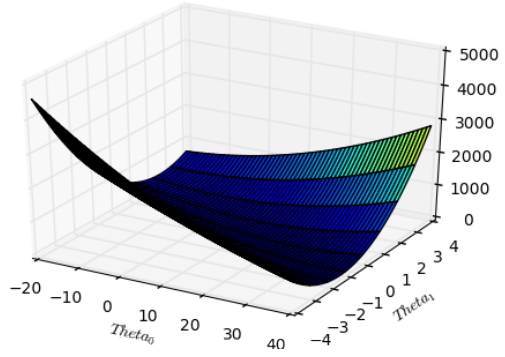


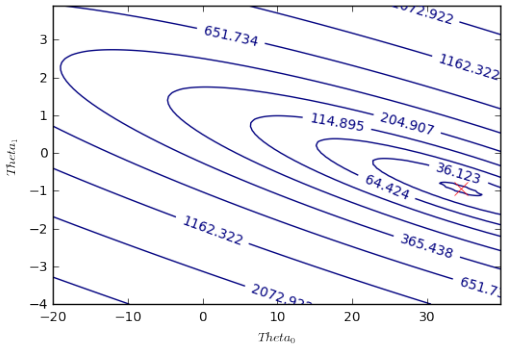
Then, We will use batch gradient descent which performs the following update on each iteration to fit the parameter to the housing data.

We can get the optimal parameter $(34.55363411, -0.95003694)$ when the cost $J(\theta)$—computed by the least square method—converges to its lowest, which is shown in the first plot. And the green line in the second plot is to check the median home value with this optimal theta.

4.1.1.2 Implementing gradient descent

- **Visualizing** $J(\theta)$





To understand the cost function better, we can plot the cost over a 2-dimensional grid of theta0 and theta1 values.

For a linear regression with a squared error, its loss function is just like a bowl, and we can get the optimal parameters when the value of the error—the value of cost function—reaches the bottom which is 36.123 as shown in the second plot.

- **Comparing with sklearn's linear regression model**

The coefficients computed by sklearn is $(34.5538408794, -0.950049353758)$ respectively, which are almost the same as computed by gradient descent—$(34.55363411, -0.95003694)$

4.1.1.3 Predicting on unseen data

When we use the final values for θ to make predictions on median home values for census tracts, we can predict a median home value of 31.5990192289 where the percentage of the population of lower economic status is 5% while a median home value of 27.9698781179 for lower status percentage50

- **Assessing model quality**

Though we have used all the given data to estimate the model, we still need to explore train/test splits and crossvalidation approaches to check its performance on the mean squared error and the R^2.

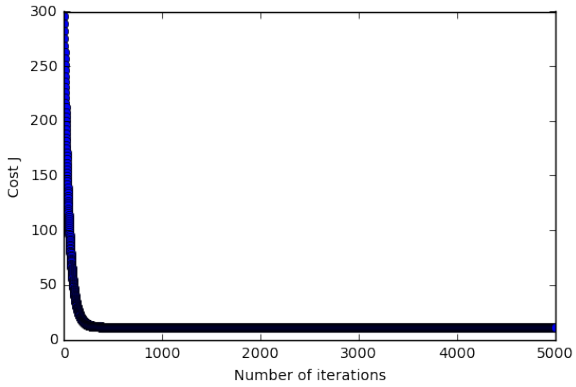
Finally, the fice fold cross_validation MSE equals to 42.61890333 while five fold cross_validation r_squared equals 0.297106799977. The MSE is somewhat small but it's still not an ideal model since it can explain only 29.7% of all the information from data.

4.1.2 Linear regression with multiple variables

4.1.2.1 Feature normalization

When features differ by orders of magnitude, feature scaling becomes important to make gradient descent converge quickly. So we sometimes need to normalize these features.

4.1.2.2 Loss function and gradient descent



Just use the same batch gradient descent method in the former problem, we can get the optimal parameters

$[2.25328063e+01\ -9.13925619e-01\ 1.06949712e+00\ 1.07531669e-01\ 6.87258582e-01\ -2.05340341e+00\ 2.67719690e+001.55788957e-02\ -3.10668099e+00\ 2.56946272e+00\ -1.97453430e+00\ -2.05873147e+00\ 8.55982884e-01\ -3.74517559e+00]$ when the value of loss function converges to its lowest.

4.1.2.3 Making predictions on unseen data

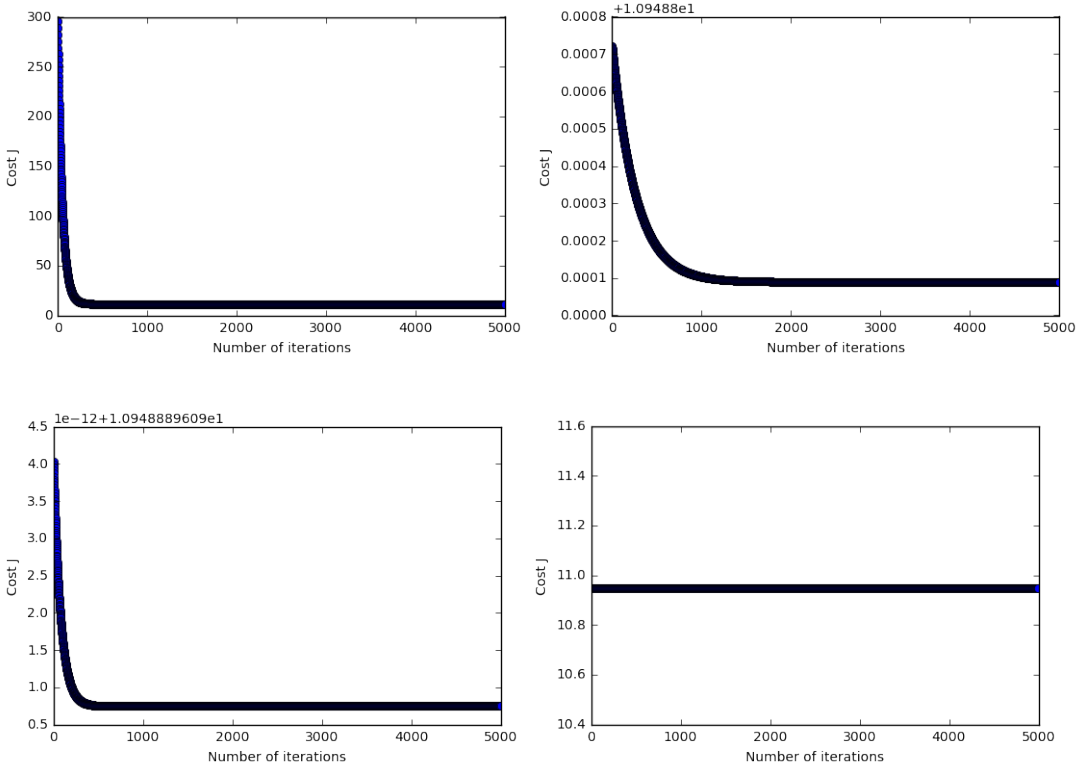
Based on the theta derived above, we can predict a median home value of 22532.8063241 for average home in Boston suburbs.

4.1.2.4 Normal equations

We can also use the closed form solution for theta which does not require any feature scaling to derive the optimal parameters.

And the median home value for average home in Boston suburbs is supposed to be 22532.8063241, with theta equals $[3.64911033e{+}01\ -1.07170557e{-}01\ 4.63952195e{-}02\ 2.08602395e-02\ 2.68856140e+00\ -1.77957587e+01\ 3.80475246e+00\ 7.51061703e-04\ -1.47575880e{+}00\ 3.05655038e{-}01\ -1.23293463e{-}02\ -9.53463555e{-}01\ 9.39251272e{-}03\ -5.25466633e-01]$. We can see the results computed by these two methods are almost the same.

4.1.2.5 Exploring convergence of gradient descent

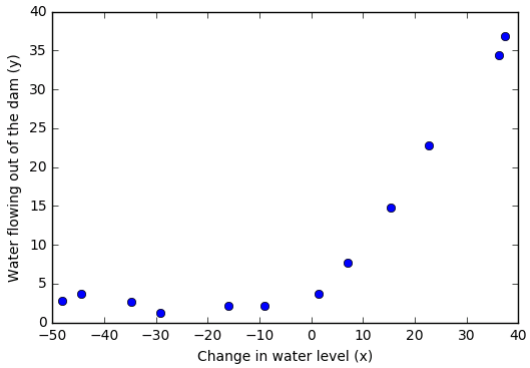


In this part of the exercise, we will get to try out different learning rates for the dataset.

From the graph above, we can see that when learning rate equals 0.01, cost converges most quickly and need only about 200 iterations. And the second fastest one is when learning rate, iterations equals 0.1, 300~400 iterations, respectively. When learning rate is too large, the gradient descent for every step is also large and it's hard for the cost to reach the lowest, just like the fourth graph.

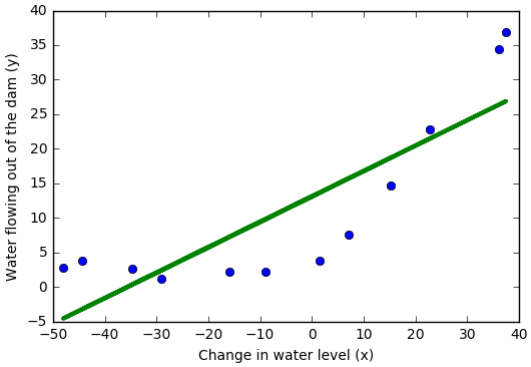
4.2 Implementing regularized linear regression

4.2.1 Regularized linear regression: an example



From the figure, we can get a trainning set data and see the relationship between change in water level(x) and water flowing out of the dam(y).

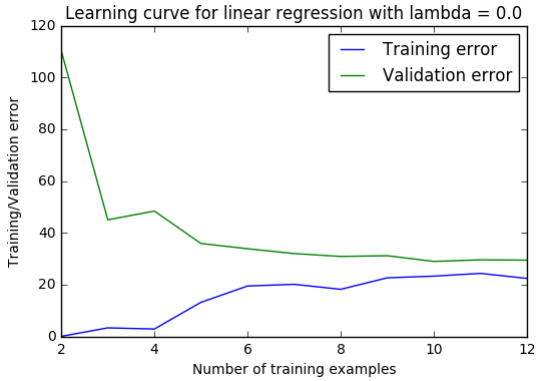
4.2.2 Regularized Linear Regression cost function and gradient



Then we use regularized linear regression to calculate the cost $J(\theta)$, and use batch gradient descent to calculate θ in each step.

The green line in the picture shows the linear estimation with the least cost $J(\theta) = 22.373906$ and $\theta = [13.08790353\ 0.36777923]$ in this case.

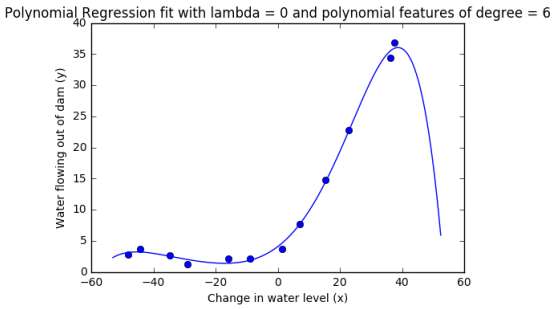
4.2.3 Learning curves

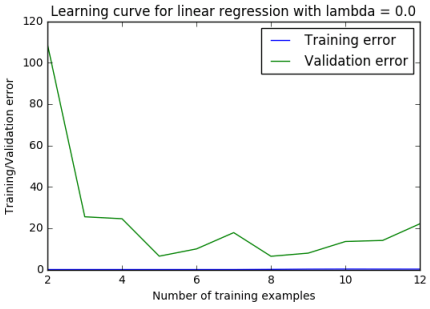


In this case, we calculate the training error and validation error using the calculated $\theta$when the regularization parameter $\lambda = 0$.

The green line in the picture is the validation error, and the blue one is the training error. In the picture, we can see that the validation error is very big initially, and gradually decrease to a stable value. The training error, on the contrary, is small at first and then ascends to a stable value. The training error and the validation error would be equal finally, and the value should be the variance of the noise. The training error is computed on the training subset instead of the entire training set, while the validation error is computed over the entire validation set. When computing the training error, there are only a few points to fit and calculate error and thus resulting in a small error. If we computer the validation error, however, there are only a few points to fit as well but we have to calculate over the whole validation set and thus resulting in a big error.

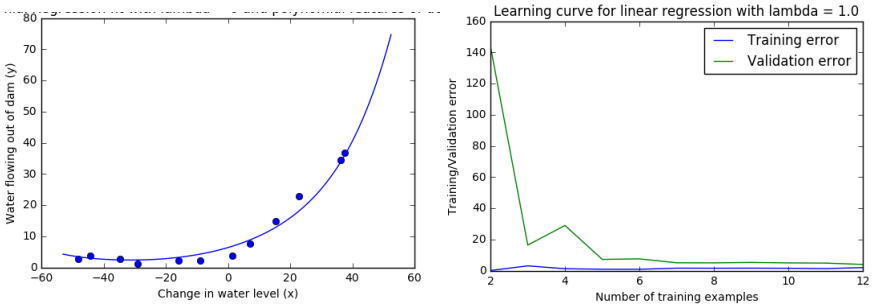
4.2.4 Learning curves for polynomial regression





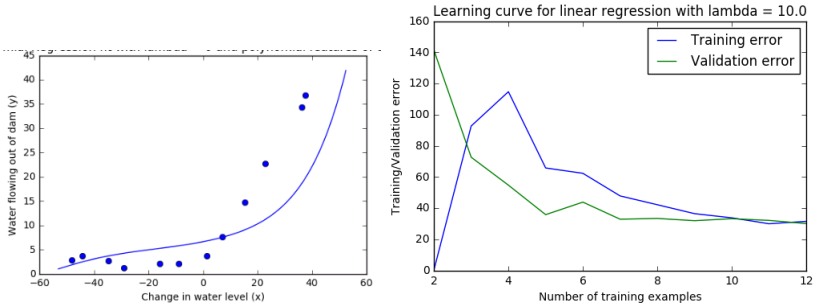
From the above two figures, we get the fitting plot and the learning curve for linear regression when the regularization parameter $\lambda = 0$ and the polynomial fetures of degree=6.

$\theta = [11.21758977\ 10.88676468\ 12.86207254\ 10.28506173\ -4.20098901\ -11.4141408\ -4.90366713]$ in this case.



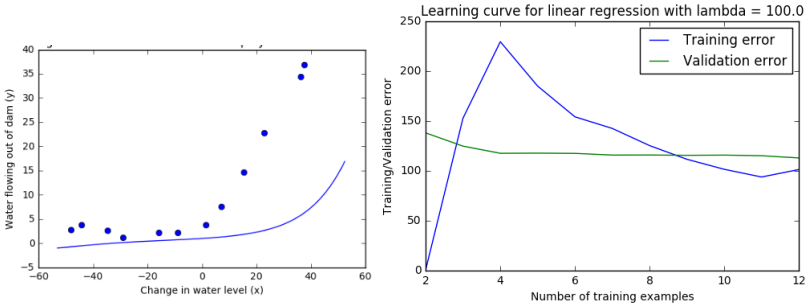
From the above two figures, we get the fitting plot and the learning curve for linear regression when the regularization parameter $\lambda = 1$ and the polynomial features of degree=6.

$\theta = [11.21729071\ 8.38095266\ 5.21767546\ 3.62016694\ 2.11114049\ 1.96213756\ 0.78456863]$ in this case.



From the above two figures, we get the fitting plot and the learning curve for linear regression when the regularization parameter $\lambda = 10$ and the polynomial features of degree=6.

$\theta = [7.99522768\ 3.16919938\ 1.66832884\ 2.29388986\ 1.07659853\ 1.68338151\ 0.69304378]$ in this case.

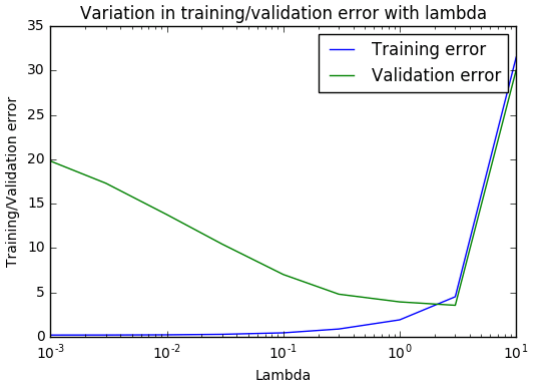


From the above two figures, we get the fitting plot and the learning curve for linear regression when the regularization parameter $\lambda = 100$ and the polynomial features of degree=6.

θ = [1.54082489 1.04663151 0.58991869 0.97159156 0.50478517 0.89924706 0.45130032] in this case.

We can see that when $\lambda = 1$, the model has the best validation error and thus it's the best model. It could both prevent overfitting and underfitting, and get a small training error and validation data as well. When $\lambda = 0$, the learning curve fits the points pretty well, and the training error is near 0, but it's overfitting and thus causing a big validation error. When $\lambda = 10$ and $\lambda = 100$, however, the training error and the validation error become larger when λ is bigger. And the two learning curves show that they're underfitting.

4.2.5 Selecting λusing a validation set



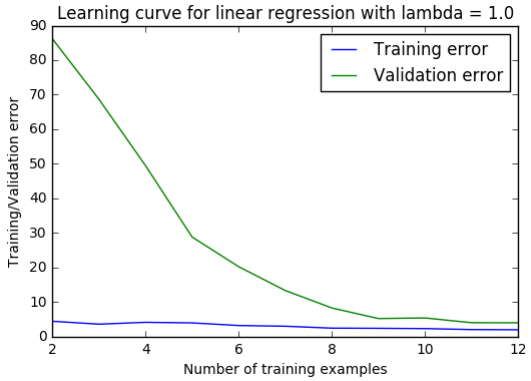
We try λ in the following range: {0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10}. The figure shows the relationship between λ and the two errors. We can see the training error becomes bigger with a larger λ, and the validation error reduces

at first, and then becomes larger. It gets the minimum value when $\lambda = 3$. The data set we choose to calculate the validation error is over the entire validation set which containing the set we have to use to calculate θ. A better validation error leads to a better performance in the test data set. So $\lambda = 3$ is the best choice of λ for the problem.

4.2.6 Calculating test error on the best model

We choose $\lambda = 3$ for the best model for the problem. The test error in this case is 4.40825183889.

4.2.7 Plotting learning curves with randomly selected examples



We choose the regularization parameter $\lambda = 1$. In this figure, we can see the relationship between the number of training examples and the two average error. The repeated multiple times we choose is 50. The average validation error gets smaller when the the number of training examples becomes larger, and the average training error is always near 0.