Machine Learning Homework 1 Report

R04921039 彭俊人

1. (1%) Linear regression function by Gradient Descent.

Linear regression： $h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$, Cost Function：$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$.

In order to minimize the cost for a best linear regression function, we update θ by

Gradient descent $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ until convergence, where $\frac{\partial}{\partial \theta_j} J(\theta)$ can be

derived as:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2}\left(h_\theta(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2}\left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j}\left(h_\theta(x) - y\right) \\
&= \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j}\left(\sum_{i=0}^{n} \theta_i x_i - y\right) \\
&= \left(h_\theta(x) - y\right) x_j
\end{aligned}$$

Therefore, updating θ becomes: $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$.

Here is my code:

```
num_points, num_features = train_X.shape
hypothesis = train_X.dot(theta)
loss = hypothesis - train_Y
gradient = np.dot(train_X.T, loss)/num_points
theta = theta - learning_rate * gradient
train_cost = loss.T.dot(loss) / (2.*num_points)
```

$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$

$\frac{\partial}{\partial \theta_j} J(\theta) = \left(h_\theta(x) - y\right) x_j$

$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$.

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

I normalized gradient and cost by dividing J(θ) with number of points for better efficiency in later calculation.

2. (1%) Describe your method.

1. **Read data:** Use Python package **csv.reader** to read data from **train.csv**
2. **Rearrange data as a 18 x 5760 array:**

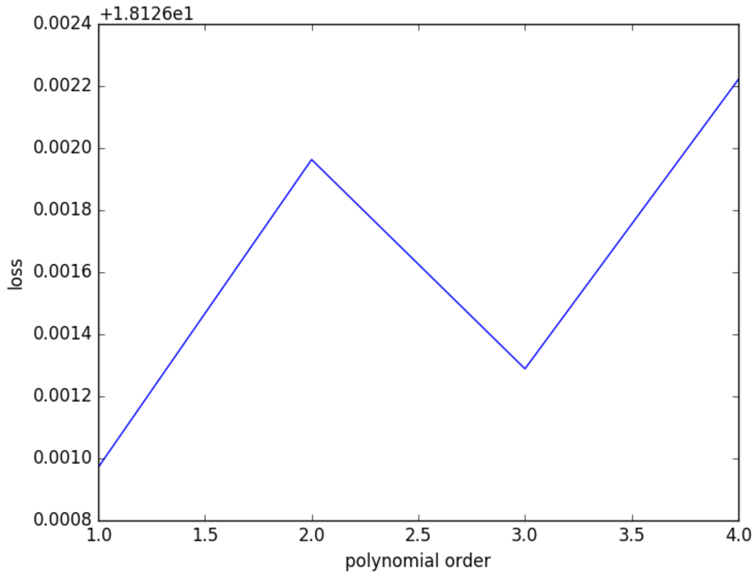
   The array has 18 columns of features and 5760 hours of data. I also replace all "NR" with 0.0.

```
Array shape: (18, 5761)
[['AMB_TEMP' '14' '14' ..., '13' '13' '13']
 ['CH4' '1.8' '1.8' ..., '1.8' '1.8' '1.8']
 ['CO' '0.51' '0.41' ..., '0.51' '0.57' '0.56']
 ...,
 ['WIND_DIREC' '35' '79' ..., '118' '100' '105']
 ['WIND_SPEED' '1.4' '1.8' ..., '1.5' '2' '2']
 ['WS_HR' '0.5' '0.9' ..., '1.6' '1.8' '2']]
```

3. **Use a sliding window to generate 5761 training data X and value Y**:
   X is a 162 x 5761 2D array that contains 5761 data point, each with 162 features (18 features x 9 hours), and Y is a 5761 array that contains value of the 10^{th} hour PM 2.5 data that we want to predict.
4. **Randomly shuffle X and Y**:
   Shuffle training data and concatenate a bias vector with value 1 to X.
5. **Choose features**:
   Use **Sequential forward selection (SFS)** after running a 10 fold cross validation for loss calculation. I found that using features [0,10, 9, 13, 12, 4] has the least cost.

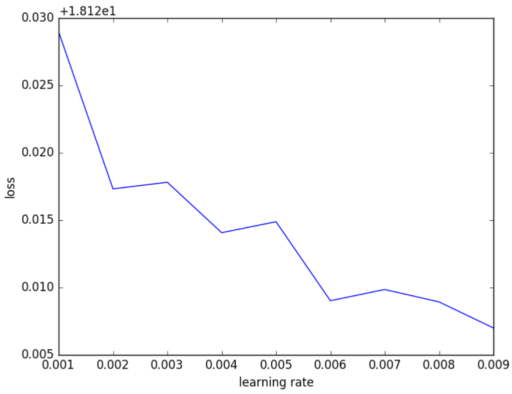
```
[peng@master hw1]$ ./kaggle_best.sh
10 fold cross-validation cost: 18.3519456786 [0, 10, 9, 13, 12, 4, 2, 8, 3, 17]
10 fold cross-validation cost: 18.3388413108 [0, 10, 9, 13, 12, 4, 2, 8, 3]
10 fold cross-validation cost: 18.3042340402 [0, 10, 9, 13, 12, 4, 2, 8]
10 fold cross-validation cost: 18.2908336858 [0, 10, 9, 13, 12, 4, 2]
10 fold cross-validation cost: 18.2680102684 [0, 10, 9, 13, 12, 4]
10 fold cross-validation cost: 18.3319695576 [0, 10, 9, 13, 12]
10 fold cross-validation cost: 18.3835986844 [0, 10, 9, 13]
10 fold cross-validation cost: 18.5676088832 [0, 10, 9]
10 fold cross-validation cost: 22.6799547017 [0, 10]
10 fold cross-validation cost: 143.453703553 [0]
```

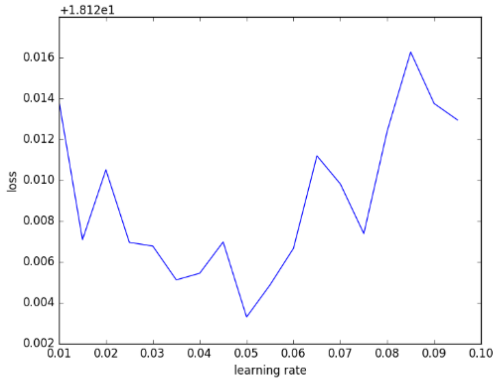
6. **Add polynomial terms in X**:
   Sweep with 10-fold cross validation to find the best order. With polynomial terms, X should be [[1, x1, x2, x1^2, x2^2, x1^3, x2^3, …],…]. I did an experiment with 10 times average cost of a 10-fold cross validation starting from order 1 to order 4 with the following graph as result. Apparently, order 1 has the least cost, i.e. X = [[1, x1, … xn], …]



7. **Find best learning rate:**

   At first, I use static learning rate, and use 10-fold cross validation to sweep each learning rate, and come up with following results.





   So I first stick with learning rate 0.05 and 0.009.

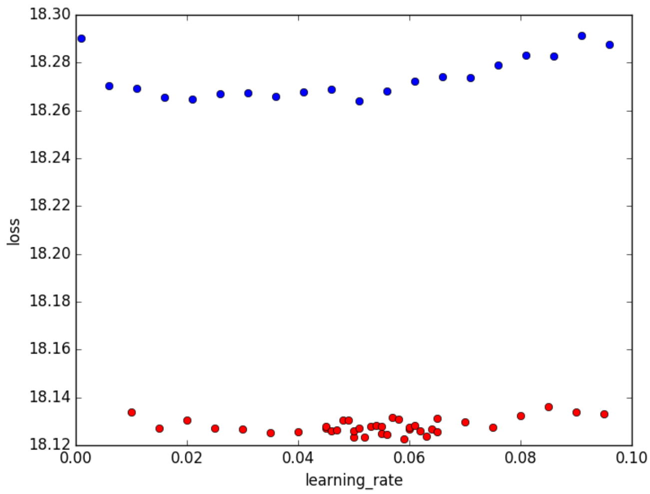
8. **Run gradient descent until converge:**

   With 1/10 of training data as validation data, I do gradient descent until the validation cost rebound. I also shrink learning rate whenever validation cost rebound, these adjustments gave me better results.

```
Iteration 14000 | Train Cost: 0.3460534728e-2 | Validation Cost: 2.4697676259e-2 | count: 4
Iteration 15000 | Train Cost: 0.3460107600e-2 | Validation Cost: 2.4697755061e-2 | count: 5
Adjust learning_rate to 0.0001000000, theta row back to iteration 14000
Iteration 16000 | Train Cost: 0.3460490776e-2 | Validation Cost: 2.4697680583e-2 | count: 0
Iteration 17000 | Train Cost: 0.3460447407e-2 | Validation Cost: 2.4697685712e-2 | count: 1
Adjust learning_rate to 0.0000100000, theta row back to iteration 16000
Iteration 18000 | Train Cost: 0.3460486391e-2 | Validation Cost: 2.4697681060e-2 | count: 0
Iteration 19000 | Train Cost: 0.3460482048e-2 | Validation Cost: 2.4697681545e-2 | count: 1
Adjust learning_rate to 0.0000010000, theta row back to iteration 18000
Iteration 20000 | Train Cost: 0.3460485953e-2 | Validation Cost: 2.4697681108e-2 | count: 0
```

9. **Choose other features:**

   Later, I found that choosing features [0,6,8,9,10], where 0 is bias, has better performance. Following is the comparison between original features [0,10,9,13,12,4] (blue dots) and new features [0,6,8,9,10] (red dots).



3. **(1%) Discussion on regularization.**

   I use 10-fold cross validation to tune each parameters, then I regulate each training epoch by using 1/10 of training data as validation data. I stop gradient descent once validation cost rebounds.

4. **(1%) Discussion on learning rate.** See Part 2.7, 2.8

5. **(1%) Other discussion and detail.** See Part 2.5, 2.6, 2.8