

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **林智威**

**Student ID 201530612316**

**E-mail sezhiweilin@mail.scut.edu.cn**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017. 12 . 8**

**1. Topic:** Linear Regression, Linear Classification and Gradient Descent

**2. Time:** 2017.12.2

**3. Reporter:** Zhiwei Lin

**4. Purposes:**

1)Further understand of linear regression and gradient descent.

2)Conduct some experiments under small scale dataset.

3)Realize the process of optimization and adjusting parameters.

**5. Data sets and data analysis:**

Linear Regression uses Housing in LIBSVM Data, including 506 samples and each sample has 13 features. You are expected to download scaled edition. After downloading, you are supposed to divide it into training set, validation set.

Linear classification uses australian in LIBSVM Data, including 690 samples and each sample has 14 features. You are expected to download scaled edition. After downloading, you are supposed to divide it into training set, validation set.

**6. Experimental steps:**

The experimental code and drawing are completed on jupyter.

**Linear Regression and Gradient Descent**

1.Load the experiment data. You can use load\_svmlight\_file function in sklearn library.

2.Devide dataset. You should divide dataset into training set and validation set using train\_test\_split function. Test set is not required in this experiment.

2.Initialize linear model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.

4.Choose loss function and derivation: Find more detail in PPT.

5.Calculate gradient toward loss function from all samples.

6.Denote the opposite direction of gradient as .

7.Update model: . is learning rate, a hyper-parameter that we can adjust.

8.Get the loss under the training set and by validating under validation set.

9.Repeate step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

**Linear Classification and Gradient Descent**

1.Load the experiment data.

2.Divide dataset into training set and validation set.

3.Initialize SVM model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.

4.Choose loss function and derivation: Find more detail in PPT.

5.Calculate gradient ***G*** toward loss function from all samples.

6.Denote the opposite direction of gradient ***G*** as .***D***

7.Update model: .***Wt = Wt-1 + αD α*** is learning rate, a hyper-parameter that we can adjust.

8.Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Get the loss under the trainin set and by validating under validation set.

9.Repeate step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

**7. Code:**

**#calculate w & gradient**

w = w - 0.0007 \*(Lambda \* w + X\_train.T \* ( X\_train\*w - y\_train ));# 0.0007是学习速率

**#calculate loss**

loss = 1/2\*Lambda\*(w.T\*w) + 1/2\*(y\_train - X\_train \* w).T \* (y\_train - X\_train \* w)#loss函数

**#calculate gradient**

def gradient(x,y,w):

M = x.shape[0]

N = y.shape[0]

m = np.zeros((M,1))

\_y =y

for i in range(M):

if(1-y[i].T\*(x[i]\*w)<0):

\_y[i] = 0

g = w + 0.5\*x.T.dot(\_y)

return g

**#calculate loss**

def cal\_loss(x,y,w):

l=0

M = x.shape[0]

\_y = y

for i in range(M):

if(1 - y[i].T\*(x[i]\*w)<0):

\_y[i]=0

l+=\_y[i]

l/=M

l+=(w.T\*w)/2

return l

**#Calculate accuracy**

def cal\_right\_rate(x,y,w):

m = y.shape[0]

right\_num = 0

right = np.zeros(m)

right = x\*w

for i in range(m):

if right[i][0] > 0:

right[i][0]=1

if right[i][0] < 0:

right[i][0]=-1

for i in range(m):

if (right[i][0] == y[i][0]):

right\_num+=1

return right\_num/m

(8.1-12.1 respectively for linear regression and 8.2-12.2 for linear classification)

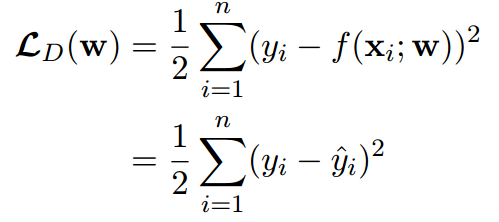
**8.1 Selection of validation (hold-out, cross-validation, k-folds cross-validation, etc.):**

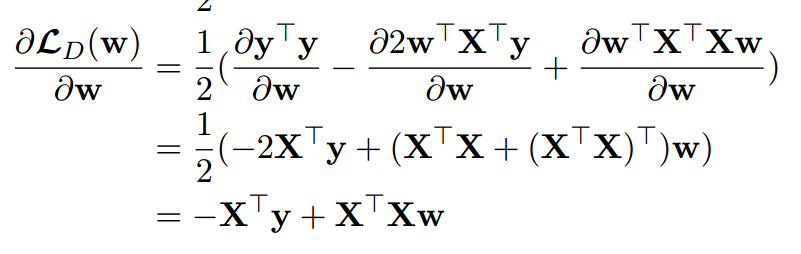
cross-validation, I divided dataset into training set and validation set using train\_test\_split function.

**9.1The initialization method of model parameters:**

all- zero initialation

**10.1 The selected loss function and its derivatives:**

****

****

**11.1 Experimental results and curve:**

## Hyper-parameter selection (η, epoch, etc.):

I finally choose 0.0007 as my η, the learning rate. It’s not a difficult job, just change it more times, then find the min loss function value.

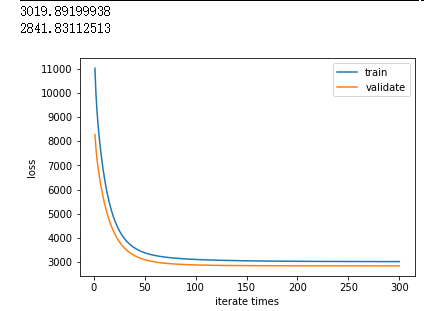
## Assessment Results (based on selected validation):

2841.831

## Predicted Results (Best Results):

0

## Loss curve:



**12.1 Results analysis:**

Actually, the loss of the validation set should be larger than the loss of the training set. Because the *w* has been optimized a lot in the training data, it will fit the training data well, but the validation set which *w* is not familiar with should have a larger loss.

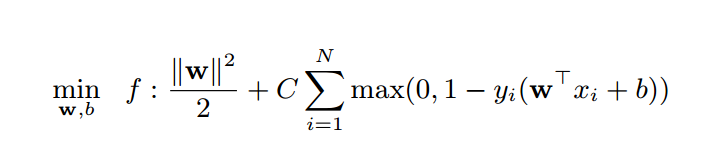
**8.2 Selection of validation (hold-out, cross-validation, k-folds cross-validation, etc.):**

cross-validation, I divided dataset into training set and validation set using train\_test\_split function.

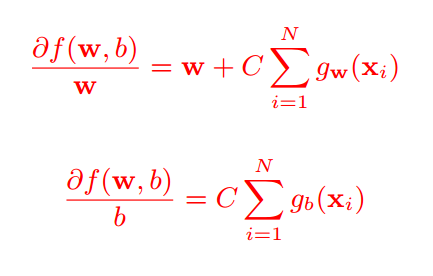
**9.2 The initialization method of model parameters:**

all- zero initialation

**10.2 The selected loss function and its derivatives:**

****

Derivative:

****

**11.2 Experimental results and curve:**

## Hyper-parameter selection (η, epoch, etc.):

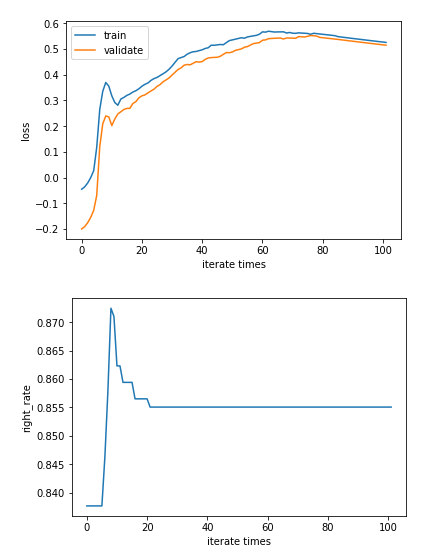
When I first set the parameter as 0.5, the accuracy is rising, and then descend to a very low accuracy as 35.80%. Consider that the parameter is too large, I change a lot of time make it stay at 0.0002. Finally it get a good scores. When I get to the 0.00015 it has a little change, but the max accuracy is lower than 0.0002.

## Assessment Results (based on selected validation):

87.25%

## Predicted Results (Best Results): accuracy = 100%

## Loss curve:



**12.2 Results analysis:**

The max accuracy is near 87.25%, and finally the accuracy is astringed to the 85.51%. The result is based on the whole data.

**13. Similarities and differences between linear regression and linear classification:**

Similarities: they used the same method called gradient descent to minimize the loss. Due to the minimize and maximize can be convert to each other, so the classification can change into a problem like regression using gradient descent.

Differences: ***w*** in the linear regression is going to fit the data, as 2-D function, trying to find a ***f*** to represent the discrete data. However, w in the linear classification is trying to divide the data into two part, and large the margin.

**14. Summary:**

In this experiment, I learned a lot about machine learning, especially the gradient descent. Gradient descent is a very valid method to minimize the loss function, so that we could get the best answer. Now I know the basic idea about machine learning. It’s quite interesting.

However, from the experiment, I find myself unfamiliar with python to solve the problem, it wasted a lot of my time. Hope to get improvement next time.