

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **黎浩坤**

**Student ID 201530611883**

**E-mail 524020165@qq.com**

**Tutor**   **谭明奎**

**Date submitted** **2017.12.8**

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

**2. Time:** 2017.12.2

**3. Reporter:**黎浩坤

**4. Purposes:**

Further understand of linear regression and gradient descent. Conduct some experiments under small scale dataset. Realize the process of optimization and adjusting parameters.

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

Linear Regression uses [Housing](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html#housing) in [LIBSVM Data](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/), 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](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#australian) in [LIBSVM](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html) 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.
2. Decide dataset into training set and validation set.
3. Initialize linear model parameters randomly.
4. Choose loss function and derivation.
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 +η\*D*.
8. Get the loss *Ltrain* under the training set and *Lvalidation* by validating under validation set.
9. Repeat step 5 to 8 for several times.
10. Drawing graph of *Ltrain* as well as *Lvalidation*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 randomly.
4. Choose loss function and derivation.
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 +η\*D*.
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 *Ltrain* under the training set and *Lvalidation* by validating under validation set.
9. Repeat step 5 to 8 for several times.
10. Drawing graph of *Ltrain* as well as *Lvalidation*with the number of iterations.

**7. Code:**

***Linear Regression Experiment:***

%matplotlib inline

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

import numpy as np

mem = Memory("./mycache")

@mem.cache

def get\_data():

data = load\_svmlight\_file("C:\\Users\\Administrator\\Desktop\\机器学习\\lab\\housing\_scale.txt")

return data[0],data[1]

data\_x,data\_y = get\_data() #Step1:Load the experiment data

from sklearn.model\_selection import train\_test\_split

x\_train,x\_valid,y\_train,y\_valid = train\_test\_split(data\_x,data\_y,test\_size=0.33,random\_state=42)

#Step2:divide dataset into training set and validation set

y\_train = np.mat(y\_train).T #shape y\_train to m\_train\*1

y\_valid = np.mat(y\_valid).T #shape y\_valid to m\_valid\*1

w = np.mat(np.random.random([14,1]))\*2-1 #Step3:Initialize linear model parameters randomly range(-1,1).

#Step4:Choose loss function and derivation.

$$ L\_D（w） = \frac{1}{2m}(y -Xw)^T (y -Xw) $$

$$ \frac{\partial L\_D（w）}{\partial w}=\frac{1}{m}(-X^Ty+X^T Xw) $$

from scipy import sparse

import matplotlib.pyplot as plt

m\_train = x\_train.shape[0] #number of training example

m\_valid = x\_valid.shape[0] #number of validation example

x\_1\_train = np.ones((m\_train,1),dtype='float64') #creat [1,1,……,1]^T

X\_train = sparse.hstack((x\_train,x\_1\_train)) #merge x\_train and [1,1,……,1]^T

x\_1\_valid = np.ones((x\_valid.shape[0],1),dtype='float64') #creat [1,1,……,1]^T

X\_valid = sparse.hstack((x\_valid,x\_1\_valid)) #merge x\_valid and [1,1,……,1]^T

Loss\_train = [] #Loss\_train-axis

Loss\_valid = [] #Loss\_valid-axis

x = [] #x-axis:the number of iterations

for n in range(0,200):#Step9:Repeat step5 to step8 for 100 times.

G = (1/m\_train)\*((-X\_train.T \* y\_train) + (X\_train.T \* X\_train \* w))

#Step5:Calculate gradient G toward loss function from all samples.

D = -G #Step6:Denote the opposite direction of gradient G as D.

learn\_rate = 0.05 #Learning rate

w = w + learn\_rate\*D #Step7:Update model parameters.

x.append(n) #Update x-axis

Loss\_train.append( ( 1/(2\*m\_train) \* (y\_train-X\_train\*w).T \* (y\_train-X\_train\*w) )[0,0])

Loss\_valid.append( ( 1/(2\*m\_valid) \* (y\_valid-X\_valid\*w).T \* (y\_valid-X\_valid\*w) )[0,0])

#Step8:Get the Loss under training set and validation set.

plt.figure(figsize=(12,12),dpi=120)

f1 = plt.subplot(211)

plt.plot(x, Loss\_train,color= 'red')

plt.plot(x, Loss\_valid,color = 'blue')

f1.set\_xlabel("Iteration time")

f1.set\_title("Loss\_train and Loss\_valid")

f2 = plt.subplot(212)

plt.plot(x, Loss\_valid)

f2.set\_xlabel("Iteration time")

f2.set\_title("Loss\_valid")

#Step10:Drawing graph of Loss\_train and Loss\_valid with the number of iterations.

print("Loss of Validation set after the last interation: "+str(Loss\_valid[199]))

#the Loss\_valid after the last interation

print("Mean squared error = "+str( ((y\_valid-X\_valid\*w).T \* (y\_valid-X\_valid\*w)/m\_valid)[0,0] ))

*Linear Classification Experiment:*

%matplotlib inline

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

import numpy as np

mem = Memory("./mycache")

@mem.cache

def get\_data():

data = load\_svmlight\_file("C:\\Users\\Administrator\\Desktop\\机器学习\\lab1\\australian\_scale.txt")

return data[0],data[1]

data\_x,data\_y = get\_data() #Step1:Load the experiment data

from sklearn.model\_selection import train\_test\_split

x\_train,x\_valid,y\_train,y\_valid = train\_test\_split(data\_x,data\_y,test\_size=0.33,random\_state=42)

#Step2:divide dataset into training set and validation set

y\_train = np.mat(y\_train).T #shape y\_train to m\_train\*1

y\_valid = np.mat(y\_valid).T #shape y\_valid to m\_valid\*1

w = np.mat(np.random.random([x\_train.shape[1],1]))\*2-1

b = np.random.random()\*2-1

#Step3:Initialize linear model parameters randomly range(-1,1)

#Step4:Choose loss function and derivation.

import matplotlib.pyplot as plt

m\_train = x\_train.shape[0] #number of training example

m\_valid = x\_valid.shape[0] #number of validation example

Loss\_train = [] #Loss\_train-axis

Loss\_valid = [] #Loss\_valid-axis

valid\_right\_rate = [] #valid\_right\_rate-axis

x = [] #x-axis:the number of iterations

for n in range(0,100):#Step9:Repeat step5 to step8 for 100 times.

G\_w = np.zeros([x\_train.shape[1],1]) #Initialize G\_w

G\_b = 0 #Initialize G\_b

C =1000000

x.append(n+1) #Update x-axis

for n in range(0,m\_train):

result = 1 - y\_train[n]\*(w.T\*x\_train[n].T+b)

if(result>=0):

G\_w = G\_w + (w-C\*x\_train[n].T\*y\_train[n])

G\_b = G\_b + (-C\*y\_train[n])

elif(result<0):

G\_w = G\_w + w

G\_b = G\_b

#Step5:Calculate gradient G toward loss function from all samples.

D\_w = -G\_w

D\_b = -G\_b

#Step6:Denote the opposite direction of gradient G as D.

learn\_rate = 0.000000001 #Learning rate

w = w + learn\_rate\*D\_w

b = b + learn\_rate\*D\_b

#Step7:Update model parameters.

threshold = 0 #Threshold

y\_fore = []

for n in range(0,m\_valid):

result = x\_valid[n]\*w+b

if(result>=threshold):

y\_fore.append(1)

elif(result<threshold):

y\_fore.append(-1)

#Calculate the forecast y

r = 0

for n in range(0,m\_valid):

if(y\_fore[n]==y\_valid[n]):

r = r+1

else:

r = r

valid\_right\_rate.append(r/m\_valid)

#Calculate the right rate of fore\_y

Loss\_train\_sum = 0

for n in range(0,m\_train):

result = 1 - y\_train[n]\*(w.T\*x\_train[n].T+b)

if(result>=0):

Loss\_train\_sum = Loss\_train\_sum + result

elif(result<0):

Loss\_train\_sum = Loss\_train\_sum

Loss\_train.append((0.5\*(w.T\*w) + C\*Loss\_train\_sum)[0,0])

Loss\_valid\_sum = 0

for n in range(0,m\_valid):

result = 1 - y\_valid[n]\*(w.T\*x\_valid[n].T+b)

if(result>=0):

Loss\_valid\_sum = Loss\_valid\_sum + result

elif(result<0):

Loss\_valid\_sum = Loss\_valid\_sum

Loss\_valid.append((0.5\*(w.T\*w) + C\*Loss\_valid\_sum)[0,0])

#Step8:Get the Loss under training set and validation set.

plt.figure(figsize=(12,12),dpi=80)

f2 = plt.subplot(212)

f2.set\_title("Accuracy")

f2.set\_xlabel("Iteration time")

plt.plot(x,valid\_right\_rate)

#Drawing graph of accuracy with the number of iterations

f1 = plt.subplot(211)

plt.plot(x, Loss\_train, color = 'red')

plt.plot(x, Loss\_valid, color = 'yellow')

f1.set\_title("Loss\_train and Loss\_valid")

f1.set\_xlabel("Iteration time")#Step10:Drawing graph of Loss\_train and Loss\_valid with the number of iterations.

print(valid\_right\_rate[99])

**8. Selection of validation:**

Linear Regression: hold-out

Linear Classification: hold-out

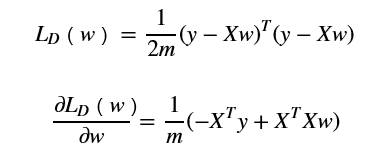
**9. The initialization method of model parameters:**

Linear Regression: randomly

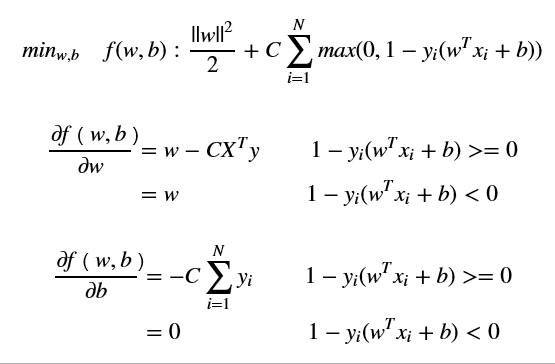
Linear Classification: randomly

**10. The selected loss function and its derivatives:**

Linear Regression:



Linear Classification:

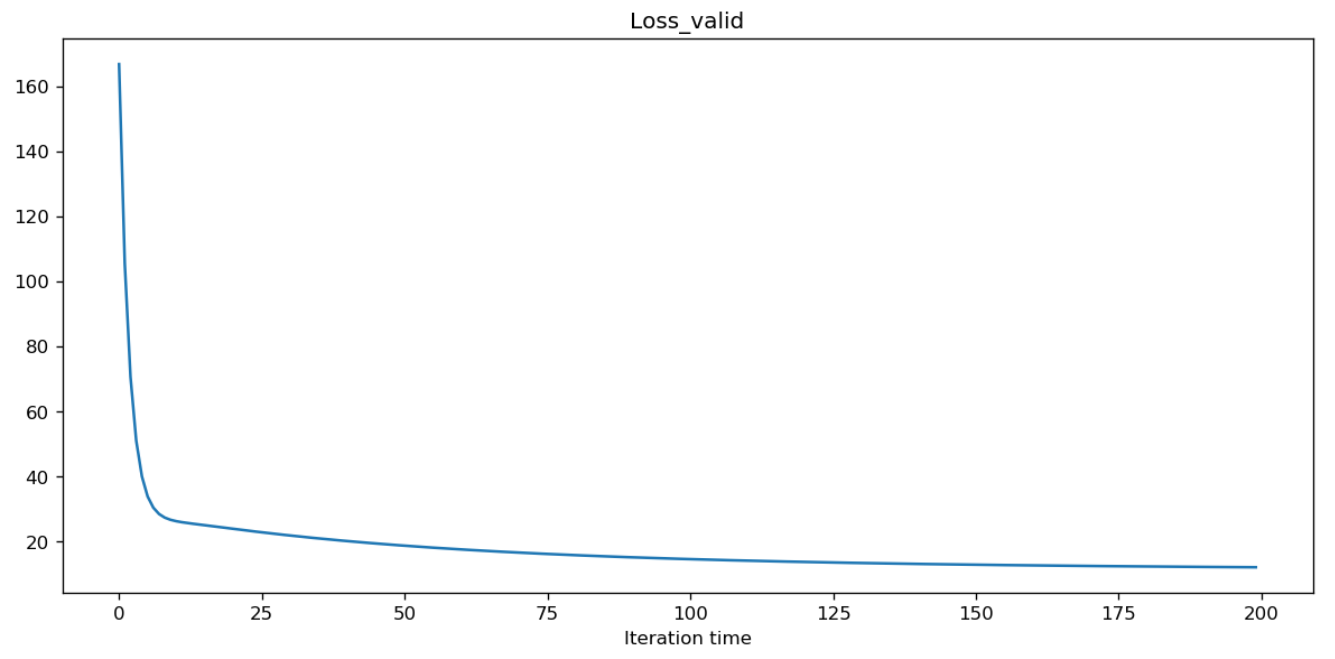


**11. Experimental results and curve:**

*Linear Regression:*

## Hyper-parameter selection: η = 0.05

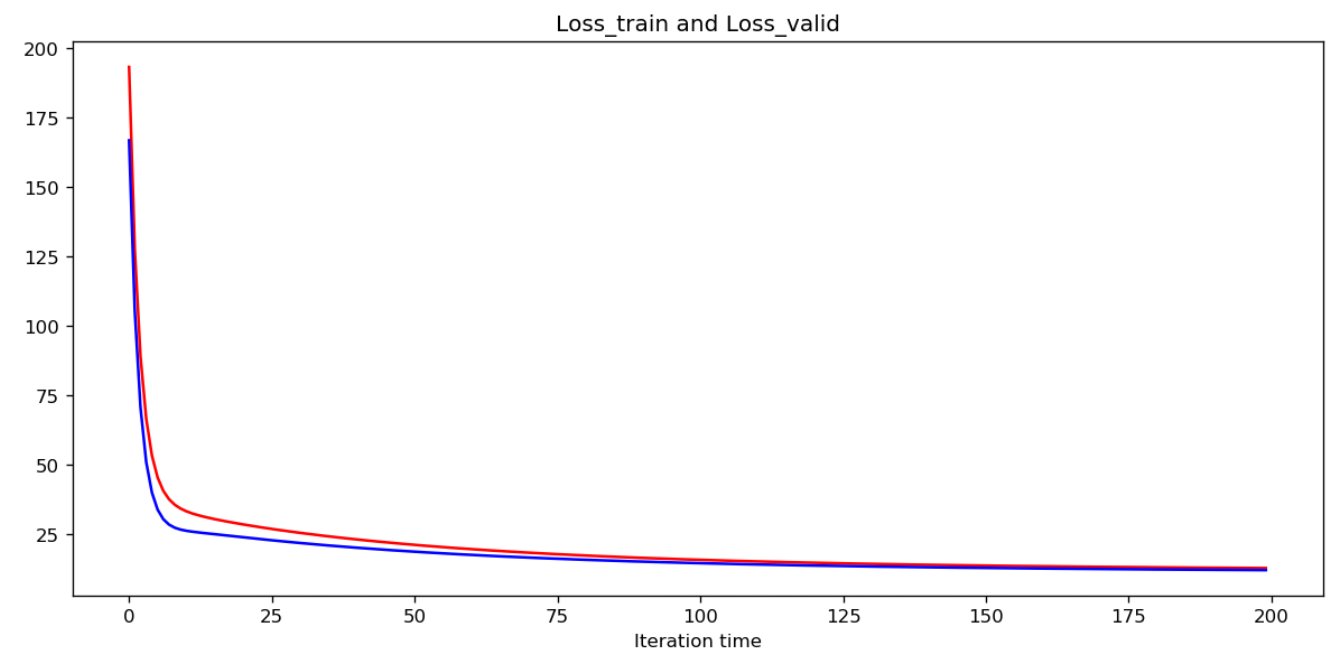
## Assessment Results:



## Predicted Results:



## Loss curve:

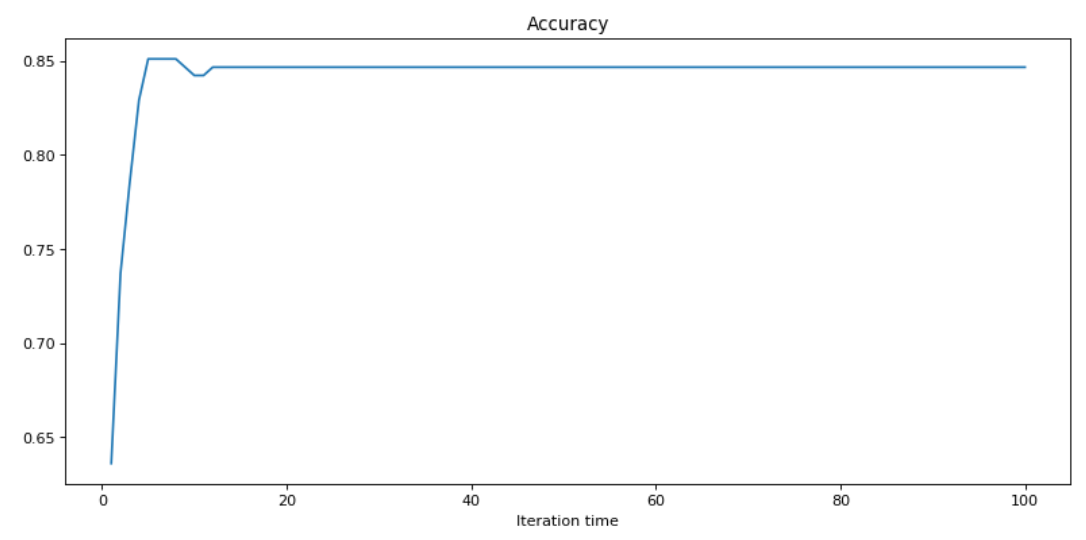


*Linear Classification:*

## Hyper-parameter selection: C =1000000

η = 0.000000001

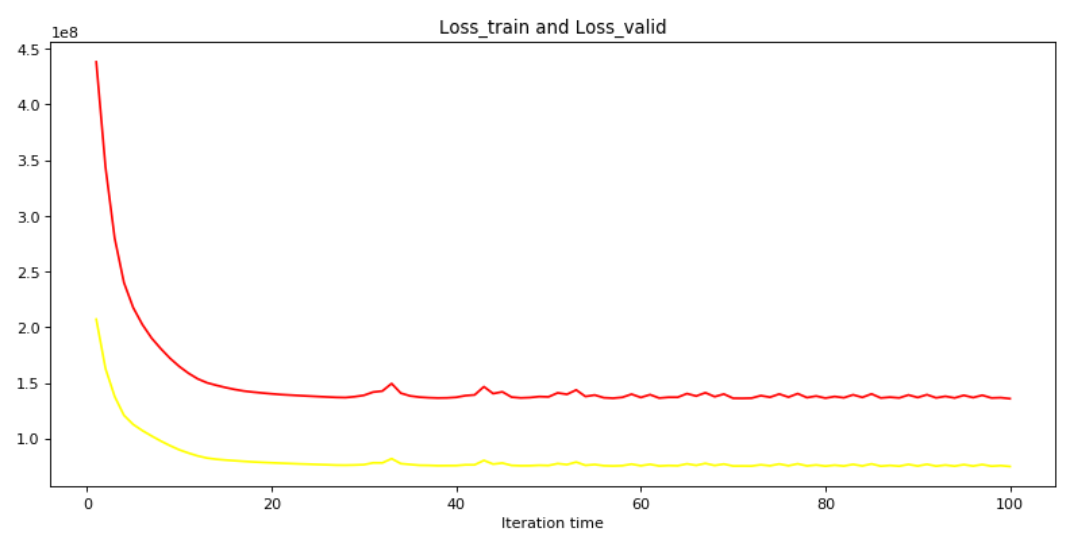
## Assessment Results:



## Predicted Results:

Best accuracy = 0.8464912280701754

## Loss curve:



**12. Results analysis:**

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

**14. Summary:**