

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **黄品超**

**Student ID 201530611760**

**E-mail 201530611760@email.scut.edu.cn**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017.12.7**

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

**2. Time: 2017/12/8**

**3. Reporter: 黄品超**

**4. Purposes:**

## First practical experiment over linear regression as well as linear classification using method of gradient descent this semester. Further more conduct some experiments under small scale data set.

## I propose that, through this experiment , I will have an good realization on the process of optimization and adjusting parameters without going deep into the theory, as well as a further understanding of linear regression and gradient descent.

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

## The follow totally come from the experiment hints.

## 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.

1. **Experimental steps:**

## The following content totally comes from the experiment hints.

## The experimental code and drawing are completed on jupyter.

# TITILE: Linear Regression and Gradient Descent

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

# 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.

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

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

# Calculate gradient toward loss function from all samples.

# Denote the opposite direction of gradient as .

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

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

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

## **TITILE: Linear Classification and Gradient Descent**

## Load the experiment data.

## Divide dataset into training set and validation set.

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

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

## Calculate gradient toward loss function from all samples.

## Denote the opposite direction of gradient as .

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

## 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.

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

1. **Code:**

**Regression：**

def model(w,x):

result = np.dot(x,w)

return result

def gradiant(x,y,w):

grad=np.dot((x.transpose((1,0))),y)-np.dot(np.dot((x.transpose((1,0))),x),w)

return grad

def refresh(w\_old,eta,G1):

w = w\_old + np.dot(eta,G1)

return w

def linear\_regression(w,x,y,LOSS,compute\_times):

eta = 0.0002

loss = 100

times = 0

size = x.shape[0]

print(size)

while times <3000:

y1 = model(w,x)

loss = square\_loss(y1,y,size)

LOSS.append(loss)

G1 = gradiant(x,y,w)

w = refresh(w,eta,G1)

times = times + 1

compute\_times.append(times)

if times == 1:

print('first loss is: ', loss)

print('loss is: ', loss)

return w

**Classification:**

def model(w,x):

result = np.dot(x,w)

return result

def hinge\_loss(y1,y,size):

one\_array = np.ones((size,1))

minus = one\_array - y\*y1

zero = np.zeros((size,1))

out = np.maximum(minus, zero)

result = out.sum()/(2\*size)

return result

def refresh(w\_old,eta,G1):

w = w\_old - np.dot(eta,G1)

return w

def compute\_gw(y1,y,x):

gw = zeros((1,15))

for i in range(0, len(y1)):

loss = 1 - y[i]\*y1[i]

if loss >=0:

gw = gw +(-y[i]\*x[i])

result = np.array(gw)

return result

def linear\_classification(x,y,w,RIGHT,compute\_times,loss\_pic):

eta = 0.0009

loss = 100

times = 0

size = x.shape[0]

right = 0

while times <4000:

y1 = model(w,x)

right = right\_percent(y,y1)

RIGHT.append(right)

loss1 = hinge\_loss(y1,y,size)

loss\_pic.append(loss1)

gw = compute\_gw(y1,y,x)

Dw = w + 0.9\*(gw.transpose(1,0))

w = refresh(w,eta,Dw)

times = times + 1

compute\_times.append(times)

if times == 1:

print('first right percent is: ', right)

print('right percent is: ', right)

return w

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

In terms of validation method, for simplicity, here choose the hold-out.

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

For both two assignments, I chose the Chi-squared distribution. For it turns out to be the best method, when I test it among the other three, zeros, randomly or with normal distribution.

Here’s the code

def ini\_para(feature\_num):

prng = random.seed(1)

# 全 0

#w = zeros((1,feature\_num), dtype = float)

# 随机

#w1 = np.random.random([1,feature\_num])

# 卡方分布

w2 = np.random.chisquare(1,size=(1,feature\_num))

# 正态分布

#w3 = np.random.randn(1,feature\_num)

# 打印参数，测试用

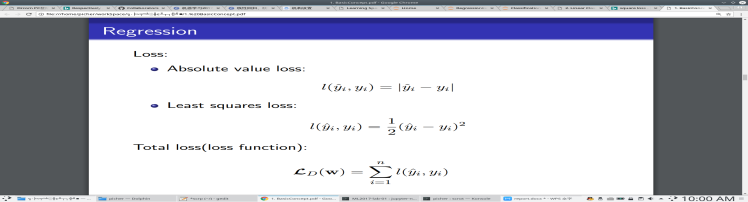
#print (w)

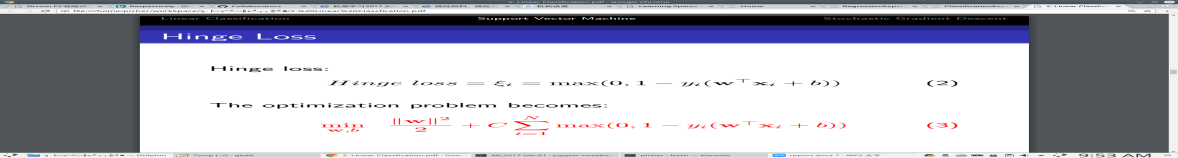
#print(w1.dtype)

return w2

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

For linear\_regression, I chose the square loss:

**** Respectively for linear\_classification, I chose the hinge loss:

****

**11. Experimental results and curve:**

For regression:

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

η = 0.0002

## Assessment Results (based on selected validation):

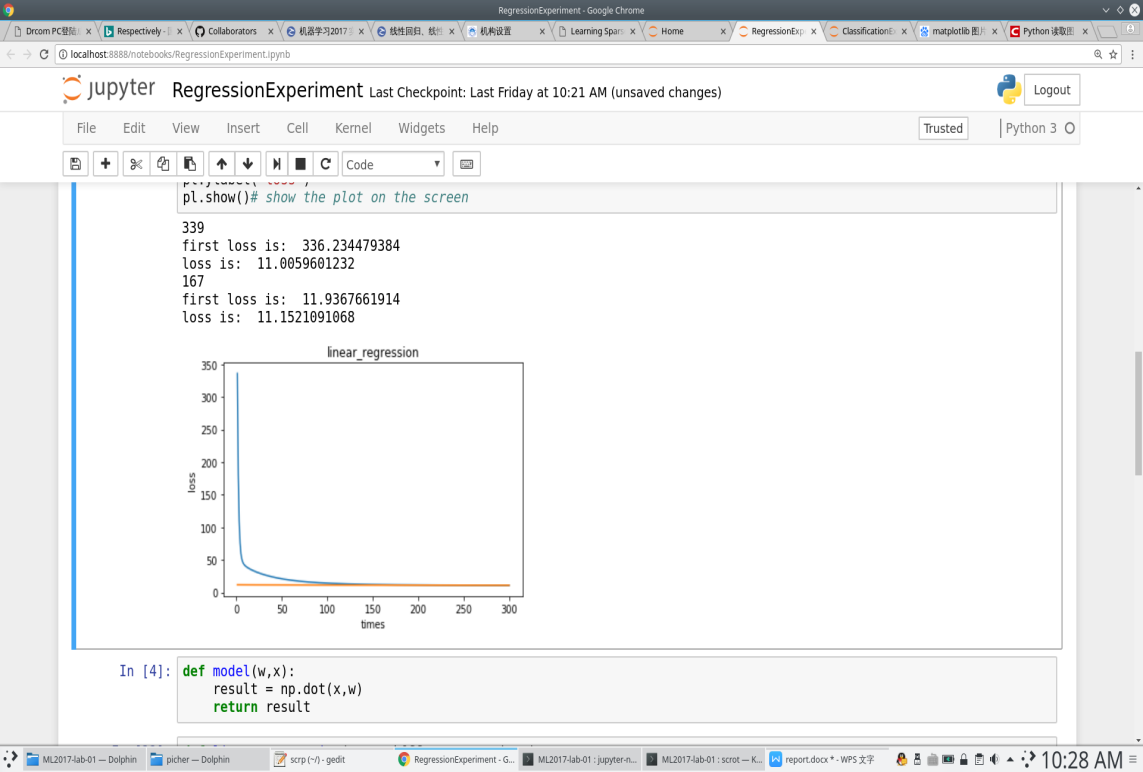
## Train and test data set should reach a similar result at the end of training.

## Predicted Results (Best Results):

## Both test and train data set can reach a min loss of 10.5 around.

## Loss curve:

The test data set, whose loss line is red, uses the model parameters result based on training on train data set. So it does not drop sharply like the blue line.



For classification:

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

η = 0.0009

C = 0.9

## Assessment Results (based on selected validation):

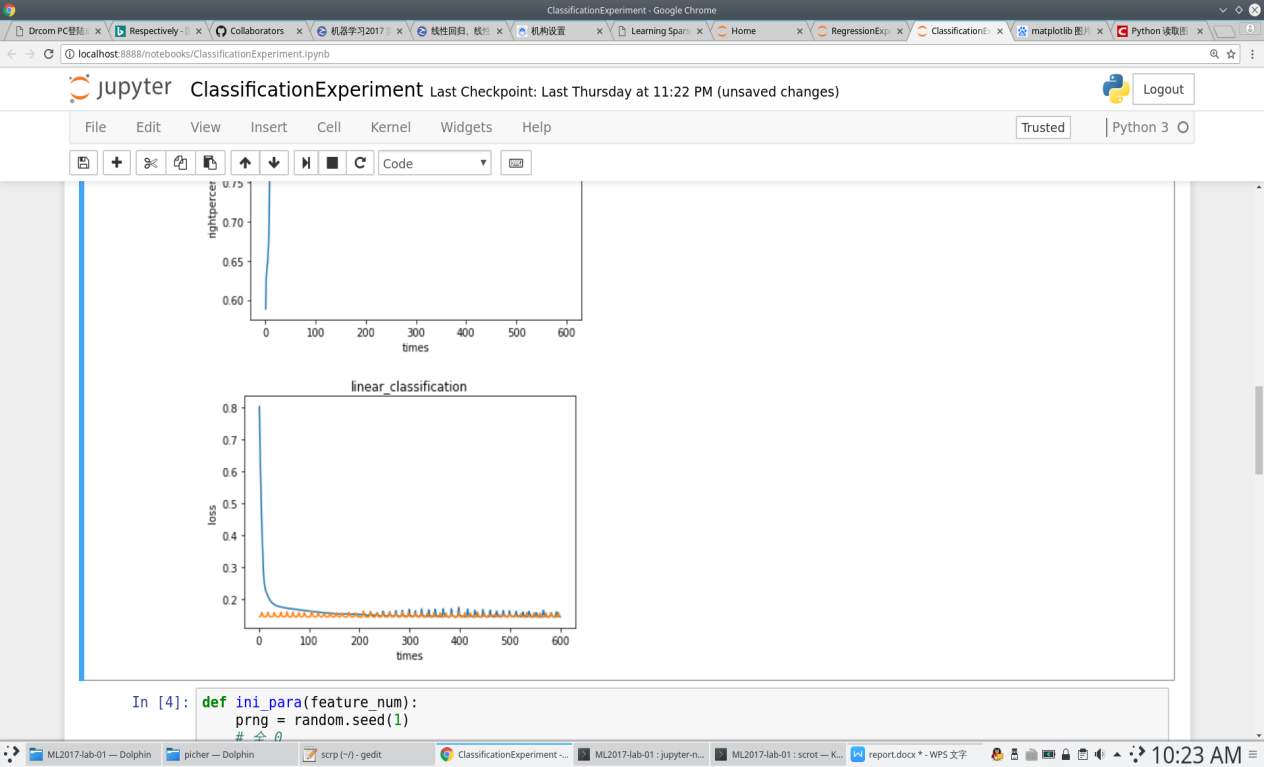
## Train and test data set should reach a similar result at the end of training, both on accuracy rate and loss.

## Predicted Results (Best Results):

Accuracy rate can reach up to 85.7142%, and it definitely converge at this rate. On the other hand, loss shakes periodically, min loss is 0.14.

## Loss curve:

The test data set, whose loss line is red, uses the model parameters result based on training on train data set. So it does not drop sharply like the blue line.



1. **Results analysis:**

Results for regression was in a surprise, what makes the result different

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

Firstly, they all share the same model function at first, secondly, they are similar in the loss function is different, it is

**14. Summary:**

In this report, I present the complete process of these two experiment. In particular go through the process of optimization and adjusting parameters, in which, for the first time,