

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**  陈星宇

**Student ID 201530611289**

**E-mail 549278303@qq.com**

**Tutor**   **谭明奎**

**Date submitted** **2017. 12 .4**

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

**2. Time: 2017.12.2**

**3. Reporter:陈星宇**

**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:**

（1）Linear Regression uses Housing data set in LIBSVM Data, including 506 samples and each sample has 13 features.

（2）Linear classification uses Australian data set in LIBSVM Data, including 690 samples and each sample has 14 features. The positive class is labeled with 1 and -1 for the opposite .

**6. Experimental steps:**

**(a) Linear Regression and Gradient Descent**

(1) Load the experiment data by using load\_svmlight\_file function in sklearn library.

(2) Dividing dataset. Divide dataset into training set and validation set using train\_test\_split function. In this experiment we use 80% data as training set and 20% as validation set .

(3) Initialize linear model parameters.

(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 +ηD . η 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) Repeat step 5 to 8 for several times, and drawing graph of loss of train as well as loss of validation with the number of iterations.

**(b) Linear Classification and Gradient Descent**

(1) Load the experiment data by using load\_svmlight\_file function in sklearn library.

(2) Dividing dataset. Divide dataset into training set and validation set using train\_test\_split function. In this experiment we use 80% data as training set and 20% as validation set .

(3) Initialize SVM model parameters.

(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 +ηD . η 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) Repeat step 5 to 8 for several times, and drawing graph of loss of train as well as loss of validation with the number of iterations.

**7. Code:**

**(a) Linear Regression**

# -\*- coding: utf-8 -\*-

from sklearn import datasets

from sklearn import model\_selection

import numpy as np

import matplotlib.pyplot as plt

def loss(X,y,w):

m = y.shape[0]

return 0.5\*(((y-X\*w).T\*(y-X\*w))).sum()/m

def gradient(X,y,w):

return X.T\*(X\*w-y)

def gradientDecent(X,y,w,alpha,num\_rounds,val\_x,val\_y):

m = y.shape[0]

train\_loss\_history = []

val\_loss\_history = []

print("origin train loss:%f"%loss(X,y,w))

train\_loss\_history.append(loss(X,y,w))

print("origin validation loss:%f"%loss(val\_x,val\_y,w))

val\_loss\_history.append(loss(val\_x,val\_y,w))

print("")

for i in range(num\_rounds):

w = w - gradient(X,y,w)\*alpha/m

train\_loss\_history.append(loss(X,y,w))

val\_loss\_history.append(loss(val\_x,val\_y,w))

return w,train\_loss\_history,val\_loss\_history

def train(X,y,val\_x,val\_y):

m = X.shape[1]

init\_w = np.matrix(np.zeros(m)).T

print("begin to train")

alpha=0.1

num\_rounds=10

print("learning rate alpha:%f"%alpha)

print("number of rounds:%d"%num\_rounds)

print("")

w,train\_loss\_history,val\_loss\_history = gradientDecent(X,y,init\_w,alpha,num\_rounds,val\_x,val\_y)

plt.plot(np.arange(num\_rounds+1),train\_loss\_history,label='train loss')

plt.plot(np.arange(num\_rounds+1),val\_loss\_history,label='validation loss')

plt.legend(loc=1)

plt.xlabel('number\_of\_rounds')

plt.ylabel('loss')

return w,train\_loss\_history,val\_loss\_history

def getData():

X,y = datasets.load\_svmlight\_file('./housing\_scale',n\_features=13)

X = np.matrix(X.toarray())

ones = np.matrix(np.ones((X.shape[0],1)))

X = np.concatenate((ones,X),axis=1)

y = np.matrix(y).T

train\_x,test\_x,train\_y,test\_y = model\_selection.train\_test\_split(X,y,test\_size=0.2)

return train\_x,test\_x,train\_y,test\_y

train\_x,test\_x,train\_y,test\_y = getData()

w,train\_loss,val\_loss = train(train\_x,train\_y,test\_x,test\_y)

print("final train loss:%f"%train\_loss.pop())

print("final validation loss:%f"%val\_loss.pop())

**(b) Linear Classification**

# -\*- coding: utf-8 -\*-

from sklearn import datasets

from sklearn import model\_selection

import numpy as np

import matplotlib.pyplot as plt

def loss(X,y,w,b,C):

m = y.shape[0]

hinge = sum(list(map(lambda x:max(0,x[0]),(1-y.A\*(X\*w+b).A))))

w\_2 = sum(w.A\*\*2)[0]

return (0.5\*w\_2+C\*hinge)/m

def gradient(X,y,w,C,b):

m = y.shape[0]

dw = np.zeros((X.shape[1],1))

db = 0

db = 0

indicator = 1-y.A\*((X\*w+b).A)

for i in range(m):

if indicator[i]>=0:

dw += w - C\*(y[i]\*X[i]).T

db += -C\*y[i]

else:

dw += w

return [dw,db]

def checkGradient(X,y,w,C,b):

delta = 1e-6

dw = (loss(X,y,w+delta,b,C)-loss(X,y,w-delta,b,C))/(np.ones(w.shape)\*delta\*2)

db = (loss(X,y,w,b+delta,C)-loss(X,y,w,b-delta,C))/delta\*2

return dw,db

def gradientDecent(X,y,w,C,b,alpha,num\_rounds,val\_x,val\_y):

m = y.shape[0]

train\_loss\_history = []

val\_loss\_history = []

print("origin train loss:%f"%loss(X,y,w,b,C))

train\_loss\_history.append(loss(X,y,w,b,C))

print("origin validation loss:%f"%loss(val\_x,val\_y,w,b,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,b,C))

print("")

for i in range(num\_rounds):

new\_w = w - gradient(X,y,w,C,b)[0]\*alpha/m

new\_b = b - gradient(X,y,w,C,b)[1]\*alpha/m

w = new\_w

b = new\_b

train\_loss\_history.append(loss(X,y,w,b,C))

val\_loss\_history.append(loss(val\_x,val\_y,w,b,C))

return w,b,train\_loss\_history,val\_loss\_history

def predict(X,y,w,b):

pred = X\*w+b

pred\_y = list(map(lambda x:1 if x[0]>0 else -1,pred.A))

acc = (y.A1==pred\_y).sum()/len(y.A)

print("acc:%f"%acc)

def train(X,y,val\_x,val\_y):

m = X.shape[1]

init\_w = np.matrix(np.zeros(m)).T

print("begin to train")

C=1

b = 1

alpha = 0.1

num\_rounds=10

print("C:%f"%C)

print("learning rate:%f"%alpha)

print("number of rounds:%d"%num\_rounds)

print("")

#print("check gradient")

#print(gradient(X,y,init\_w,C,b))

#print(checkGradient(X,y,init\_w,C,b))

#print("")

w,b,train\_loss\_history,val\_loss\_history = gradientDecent(X,y,init\_w,C,b,alpha,num\_rounds,val\_x,val\_y)

plt.plot(np.arange(num\_rounds+1),train\_loss\_history,label='train loss')

plt.plot(np.arange(num\_rounds+1),val\_loss\_history,label='validation loss')

plt.legend(loc=1)

plt.xlabel('number\_of\_rounds')

plt.ylabel('loss')

return w,b,train\_loss\_history,val\_loss\_history

def getData():

X,y = datasets.load\_svmlight\_file('./australian\_scale',n\_features=14)

X = np.matrix(X.toarray())

y = np.matrix(y).T

train\_x,test\_x,train\_y,test\_y = model\_selection.train\_test\_split(X,y,test\_size=0.2)

return train\_x,test\_x,train\_y,test\_y

train\_x,test\_x,train\_y,test\_y = getData()

w,b,train\_loss,val\_loss = train(train\_x,train\_y,test\_x,test\_y)

print("final train loss:%f"%train\_loss.pop())

print("final validation loss:%f"%val\_loss.pop())

print("train:")

predict(train\_x,train\_y,w,b)

print("test:")

predict(test\_x,test\_y,w,b)

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

In this experiment we use 80% data as training set and 20% as validation set .

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

In this experiment we initial the model parameters to zeros

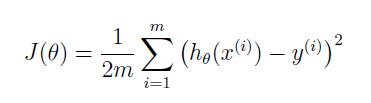
**10. The selected loss function and its derivatives:**

**(a) Linear Regression**

We define our hypothesis as

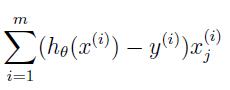
****

And we use Least Square Loss as our loss function which defined as

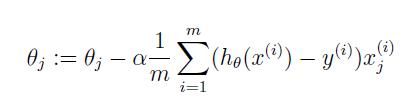
****

Where m denotes the number of training examples and yi denotes the label of training data. Our objective is to minimize the loss function .In gradient decent, we can reach the minimum of loss function by updating the parameter .The update rule is to subtract the negative decent of loss function from parameters.

The derivative of the loss function can be computed as



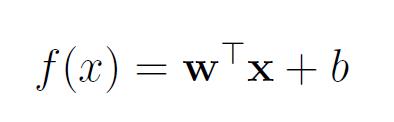
So in each iteration we update our parameter θ using the following equation:



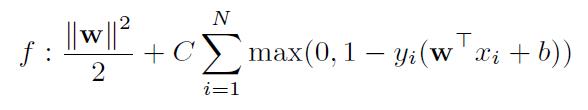
Where α denotes the learning rate and x(i)j denotes the jth feature of the ith training examples

**(b) Linear Classification**

We define our hypothesis as

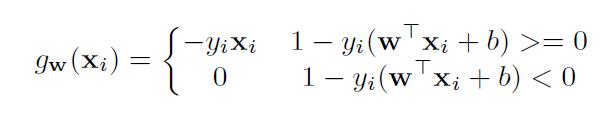
****

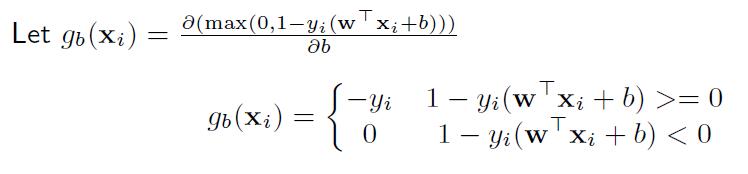
Where w is a parameter vector and b is another parameter set by us .And we define our loss function (using hinge loss to measure the loss of prediction and the grand truth) as

****

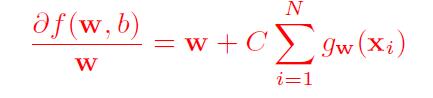
Where w is the parameters and ||w||2 denotes the sum of squared wi .N denotes the number of training examples .C is a parameter set by us. Our objective is to minimize the loss function .In gradient decent, we can reach the minimum of loss function by updating the parameter .The update rule is to subtract the negative decent of loss function from parameters.

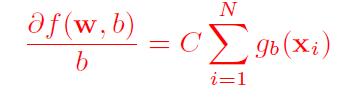
The derivative of the loss function can be computed as





And then we have





So in each iteration we update our parameter using the following equation :

w = w –η\*Gradient\_of\_w(f(w,b))/m

b = b -η\*Gradient\_of\_b(f(w,b))/m

Whereηdenotes the learning rate and m is the number of training examples.

**11. Experimental results and curve:**

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

1. **Linear Regression**

We chose learning rateη as 0.1 and number of rounds as 10

**(b) Linear Classification**

We chose C=1,b=1,learning rateη as 0.1 and number of rounds as 20

## Assessment Results (based on selected validation):

1. **Linear Regression**

We calculate the loss of training set and validation set and compare them.

**(b) Linear Classification**

We calculate the loss of training set and validation set and compare them. We also calculate the prediction accuracy of training set and validation set.

## Predicted Results (Best Results):

1. **Linear Regression**

The best result is :

Training loss: 28.338978

Validation loss: 38.042849

**(b) Linear Classification**

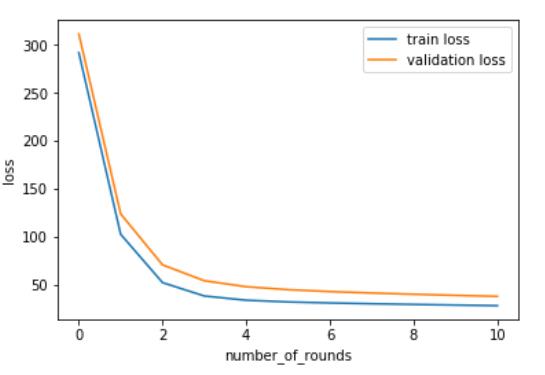
The best result is :

Training accuracy: 0.860507

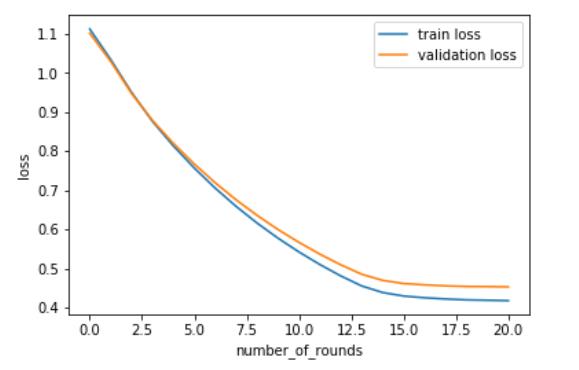
Validation accuracy: 0.847826

## Loss curve:

1. **Linear Regression**

****

**(b) Linear Classification**



**12. Results analysis:**

**(a) Linear Regression**

From the curve we can see that as the number of iteration increases, the loss of training set and validation set both decrease, and they will converge to a small value .As a result we can say that gradient decent works pretty good on linear regress .

**(b) Linear Classification**

From the curve we can see that as the number of iteration increases, the loss of training set and validation set both decrease (and the accuracy of training set and validation set increase, which are not plotted above), and they will converge to a small value .As a result we can say that gradient decent works pretty good on linear classification.

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

**Similarities:**

Linear regression and linear classification has the similar form of hypothesis, and the decision boundary are both a straight line (in the perspective of two dimension). The gradient decent step are also similar of the two model.

**Differences:**

Linear regression is a regression task, which means to predict a (continuous) value by fitting the training data using a straight line .Linear classification is a classification task, whose objective is to find a straight line to separate the different classes. The loss function of these two model are also different.

**14. Summary:**

In this experiment we implement a linear regression model and a linear classification model based on SVM. The objective of this experiment is to try and see how the gradient decent works on models and from the result we can see gradient decent is a normal way to reach the optimal of objective function. By calculating the derivative and plotting the curve, I get a better understanding of this two model and gradient decent.