

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **Haixu Liu**

**Student ID 201530612347**

**E-mail 1059350952@qq.com**

**Tutor**   **Mingkui Tan**

**Date submitted 2017. 12 . 8**

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

**2. Time: 2017. 12. 2**

**3. Reporter: Haixu Liu**

**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 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:**

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.

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

Calculate gradient G toward loss function from all samples.

Denote the opposite direction of gradient G as D.

Update model: . Wt = Wt-1 + a \* D. ‘a’ is learning rate, a hyper-parameter that we can adjust.

Get the loss ‘L train’ under the training set and ‘L validation’ by validating under validation set.

Repeate step 5 to 8 for several times, and drawing graph of ‘L train’ as well as ‘L validation’ with the number of iterations.

**7. Code:**

1. Linear regression:

data = load\_svmlight\_file("housing\_scale.txt")

data\_train, data\_test, value\_train, value\_test = train\_test\_split(data[0], data[1], test\_size=0.33)

learning\_rate = 0.0005

initial\_w = random.random(size=(14, 1))

num\_iter = 1000

bias\_train = np.mat(np.ones(shape=[data\_train.shape[0], 1]))

bias\_train = csr\_matrix(bias\_train).todense()

bias\_test = np.mat(np.ones(shape=[data\_test.shape[0], 1]))

bias\_test = csr\_matrix(bias\_test).todense()

x\_mat\_train = csr\_matrix(data\_train).todense()

x\_mat\_train = np.hstack((x\_mat\_train, bias\_train))

y\_train = np.mat(value\_train).T

y\_mat\_train = csr\_matrix(y\_train).todense()

x\_mat\_test = csr\_matrix(data\_test).todense()

x\_mat\_test = np.hstack((x\_mat\_test, bias\_test))

y\_test = np.mat(value\_test).T

y\_mat\_test = csr\_matrix(y\_test).todense()

initial\_w = csr\_matrix(initial\_w).todense()

gradient = - np.dot(x\_mat\_train.T, y\_mat\_train) + np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), initial\_w)

d = - gradient

w = initial\_w

loss\_list\_train = []

loss\_list\_test = []

for i in range(num\_iter):

w = w + learning\_rate \* d

d = np.dot(x\_mat\_train.T, y\_mat\_train) - np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), w)

# d\_test = np.dot(x\_mat\_test.T, y\_mat\_test) - np.dot(np.dot(x\_mat\_test.T, x\_mat\_test), w)

loss\_train = np.dot((y\_mat\_train - np.dot(x\_mat\_train, w)).T, (y\_mat\_train - np.dot(x\_mat\_train, w))) / 2 / 339

loss\_test = np.dot((y\_mat\_test - np.dot(x\_mat\_test, w)).T, (y\_mat\_test - np.dot(x\_mat\_test, w))) / 2 / 167

loss\_list\_train.append(loss\_train.tolist()[0][0])

loss\_list\_test.append(loss\_test.tolist()[0][0])

2. Linear classification:

data = load\_svmlight\_file("australian\_scale.txt")

data\_train, data\_test, value\_train, value\_test = train\_test\_split(data[0], data[1], test\_size=0.33)

learning\_rate = 0.0005

initial\_w = random.random(size=(15, 1))

num\_iter = 1000

bias\_train = np.mat(np.ones(shape=[data\_train.shape[0], 1]))

bias\_train = csr\_matrix(bias\_train).todense()

bias\_test = np.mat(np.ones(shape=[data\_test.shape[0], 1]))

bias\_test = csr\_matrix(bias\_test).todense()

x\_mat\_train = csr\_matrix(data\_train).todense()

x\_mat\_train = np.hstack((x\_mat\_train, bias\_train))

y\_train = np.mat(value\_train).T

y\_mat\_train = csr\_matrix(y\_train).todense()

x\_mat\_test = csr\_matrix(data\_test).todense()

x\_mat\_test = np.hstack((x\_mat\_test, bias\_test))

y\_test = np.mat(value\_test).T

y\_mat\_test = csr\_matrix(y\_test).todense()

initial\_w = csr\_matrix(initial\_w).todense()

# gradient = - np.dot(x\_mat\_train.T, y\_mat\_train) + np.dot(np.dot(x\_mat\_train.T, x\_mat\_train), initial\_w)

# d = - gradient

# c = random.random(size=(1, 15))

gradient\_list = []

for i in range(x\_mat\_train.shape[0]):

a = np.dot(initial\_w.T, x\_mat\_train[0].T)

b = np.dot(y\_mat\_train[0], a)

if b[0] < 1:

gw\_train = - np.dot(y\_mat\_train[0], x\_mat\_train[0])

else:

gw\_train = 0

gradient\_list.append(gw\_train)

c = choice(gradient\_list)

gradient = initial\_w + 0.9 \* c

w = initial\_w

d = - gradient

loss\_list\_train = []

loss\_list\_test = []

for j in range(num\_iter):

# w，d的迭代更新

w = w + learning\_rate \* d

for k in range(x\_mat\_train.shape[0]):

a = np.dot(w.T, x\_mat\_train[0].T)

b = np.dot(y\_mat\_train[0], a)

if b[0] < 1:

gw\_train = - np.dot(y\_mat\_train[0], x\_mat\_train[0])

else:

gw\_train = 0

c = choice(gradient\_list)

gradient = w + 0.9 \* c

# d\_test = np.dot(x\_mat\_test.T, y\_mat\_test) - np.dot(np.dot(x\_mat\_test.T, x\_mat\_test), w)

loss\_train = np.dot((y\_mat\_train - np.dot(x\_mat\_train, w)).T, (y\_mat\_train - np.dot(x\_mat\_train, w))) / 2 / 339

loss\_test = np.dot((y\_mat\_test - np.dot(x\_mat\_test, w)).T, (y\_mat\_test - np.dot(x\_mat\_test, w))) / 2 / 167

loss\_list\_train.append(loss\_train.tolist()[0][0])

loss\_list\_test.append(loss\_test.tolist()[0][0])

negative\_class = []

positive\_class = []

for i in range(x\_mat\_train.shape[0]):

hyperplane = np.dot(w.T, x\_mat\_train[i].T).tolist()

if hyperplane[0][0] > 0:

positive\_class.append(hyperplane[0][0])

else:

negative\_class.append(hyperplane[0][0])

print("positive class is: ", positive\_class)

print("/n/n/n/n")

print("negative class is: ", negative\_class)

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

Cross-validation

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

Learning rate: learning rate = 0.0005

w: initial\_w = random.random(size=(15,1))

iteration times: num\_iter = 1000

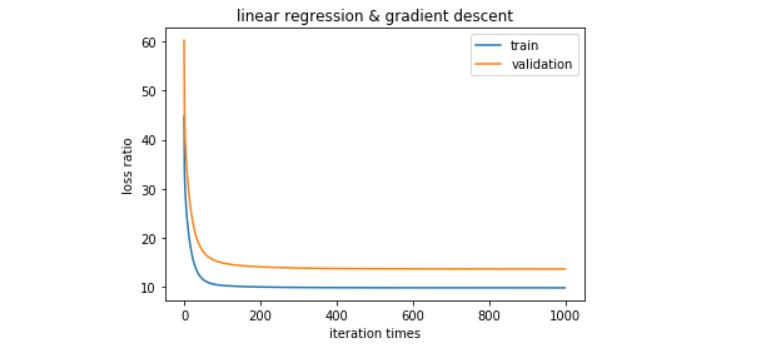
**10. The selected loss function and its derivatives:**

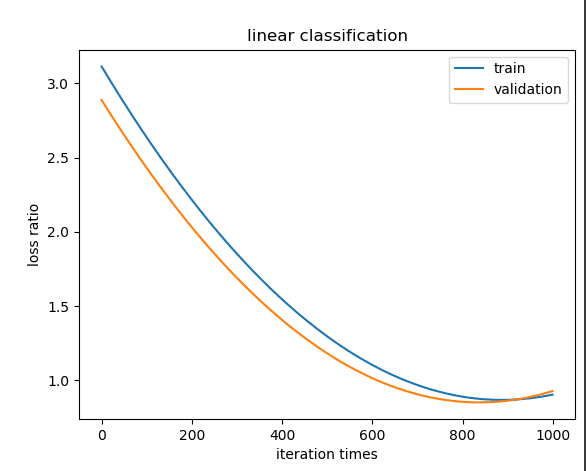
Loss: 1/2 \* 1/n \* (y - Xw)^T \* (y - Xw)

Hinge Loss: max(0; 1 - yi(w^T \* xi + b))

**11. Experimental results and curve:**

## Loss curve:





**12. Results analysis:**

In experiment1, we can understand that when we have enough training times, loss ration will drop to a certain stable figure, the training set and validation set have little difference, the result is related to the initialization.

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

We need to use support vector machine in the second experiment, linear classification, and the loss functions are different, but they both need to use gradient descent.

**14. Summary:**

The problem should be that the loss function has some mistakes during the two experiments, and I have misunderstand the two theory of linear regression and support vector machine, which may lead to the mistake of the graph, especially experiment2.