

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **宋凯**

**Student ID 201530612729**

**E-mail 554252593@qq.com**

**Tutor**   **吴庆耀**

**Date submitted** **2017.12 .2**

1. **Topic:**

Linear Regression, Linear Classification and Gradient Descent

1. **Time:**

Dec 2nd

1. **Reporter:**

Kai Song

1. **Purposes:**

Further understand of linear regression , Classification and gradient descent.

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

Using **housing** and **australian** Dataset in **Libsvm** to analysis with linear regression and linear classification(SVM)

**6. Experimental steps:**

1.Define gradient decent

2.Define loss function

3.Using reverse gradient an learning rate update theta(with loss function)

4. Cycle many times to get a better loss result

**7. Code:**

**1.linear regression:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import time

from sklearn.datasets import load\_svmlight\_file

from sklearn.cross\_validation import train\_test\_split

# Draw

def Draw(iterations, Loss\_train, Loss\_Validation):

plt.plot(np.arange(0,100,1), Loss\_train[0:100], label='Loss Train ')

plt.plot(np.arange(0,100,1), Loss\_Validation[0:100], label='Loss Validation')

plt.xlabel('iterations')

plt.ylabel('loss')

plt.title('Loss')

plt.legend()

plt.show()

# read data

def get\_data():

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

return data

def grad(theta, X\_train , y\_train):

grad = np.dot(X\_train.transpose(), np.dot(X\_train, theta)) - np.dot( X\_train.transpose(),y\_train)

return grad

if \_\_name\_\_ == '\_\_main\_\_':

X = get\_data()[0]

y = get\_data()[1]

X = X.toarray()#转换数组

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

train\_row = X\_train.shape[0]

test\_row = X\_test.shape[0]

col = X\_train.shape[1]

y\_train = y\_train.reshape(train\_row, 1)

y\_test = y\_test.reshape(test\_row, 1)

theta = np.random.random(size = (col, 1))#theta初始化

alpha = 0.0001#学习率

iterations = 1000 # 循环总次数

epsilon = 0.000001 # 收敛精度

count = 0 # 迭代的次数

theta1 = np.zeros((col, 1)) # 上次theta的值，初始为0向量

finish = 0 # 完成标志位

lossTrain = []

lossValidation = []

while count < iterations:

count += 1

# 所有训练数据的期望更新一次theta

theta = theta - alpha \* (grad(theta, X\_train , y\_train) ) #负梯度更新theta

if (np.linalg.norm(theta - theta1) < epsilon):

finish = 1

break

else:

theta1 = theta

LossTrain = (1/2) \* theta.transpose().dot(theta) + (1/2) \* (y\_train - X\_train.dot(theta)).transpose().dot((y\_train - X\_train.dot(theta)))

#训练集loss

LossValidation = (1/2) \* theta.transpose().dot(theta) + (1/2) \* (y\_test-X\_test.dot(theta)).transpose().dot((y\_test - X\_test.dot(theta)))

#测试集loss

lossTrain.append(LossTrain[0] / train\_row)

lossValidation.append(LossValidation[0] / test\_row)

print('iterations '.format(count), count, '\n', ' LossTrain: ',LossTrain,'\n' ' LossValidation: ',LossValidation)

Draw(count, lossTrain, lossValidation)

**2.linear classification:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import time

from sklearn.datasets import load\_svmlight\_file

from sklearn.cross\_validation import train\_test\_split

# Draw

def Draw(loops, train\_loss, validation\_loss):

#the loss

plt.plot(np.arange(0,loops-1,1), train\_loss[0:loops-1], label='Loss Train ')

plt.plot(np.arange(0,loops-1,1), validation\_loss[0:loops-1], label='Loss Validation ')

plt.xlabel('iteration')

plt.ylabel('loss')

plt.title('Loss')

plt.legend()

plt.show()

# read data

def get\_data():

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

return data

# 梯度

def grad(X, Y, theta, b):

grad = theta

for i in range(X.shape[0]):

if Y[i] \* (theta.transpose().dot(X[i]) + b) < 1:

grad = grad - (1 \* X[i] \* Y[i]).reshape(X.shape[1], 1)

else:

grad = grad - (0 \* X[i] \* Y[i]).reshape(X.shape[1], 1)

return grad

# Loss函数

def Loss(X, Y, theta, b):

lossFunction = (1/2) \* theta.transpose().dot(theta)

for i in range(X.shape[0]):

Tensor = Y[i] \* (theta.transpose().dot(X[i]) + b)

if Tensor < 1:

lossFunction = lossFunction + 1 - Tensor

return lossFunction

if \_\_name\_\_ == '\_\_main\_\_':

X = get\_data()[0]

y = get\_data()[1]

X = X.toarray()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

column = X\_train.shape[1]

train\_row = X\_train.shape[0]

test\_row = X\_test.shape[0]

y\_train = y\_train.reshape(train\_row, 1)

y\_test = y\_test.reshape(test\_row, 1)

theta = np.random.random(size = (column, 1))

alpha = 0.0001#学习率

b = 2

iterations = 1000 # 循环总次数

epsilon = 0.0001 # 收敛精度

count = 0 # iteration的次数

error = np.zeros((column, 1)) # 上次theta的值，初始为0向量

finish = 0 # 完成标志位

# 初始化

lossTrain = []

lossValidation = []

while count < iterations:

count += 1

theta = theta - alpha \* grad(X\_train, y\_train, theta, b)

if(np.linalg.norm(theta - error) < epsilon):

finish = 1

break

else:

error = theta

Loss\_Train = Loss(X\_train, y\_train, theta, b)

Loss\_Validation = Loss(X\_test, y\_test, theta, b)

lossTrain.append(Loss\_Train[0] / train\_row)

lossValidation.append(Loss\_Validation[0] / test\_row)

print('iterations '.format(count), count, '\n','Loss\_train: ',Loss\_Train / train\_row, '\n ''Loss\_Validation: ',Loss\_Validation / test\_row)

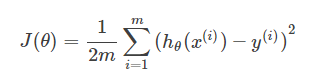
Draw(count, lossTrain, lossValidation)

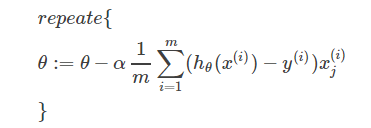
**8.** **The initialization method of model parameters:**

Initializerandomly

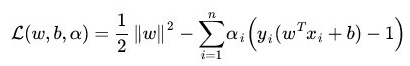
**9.** **The selected loss function and its** **derivatives:**

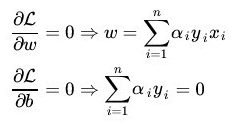
1.Linear regression:





2.SVM:





**10****.** **Experimental results and curve:**

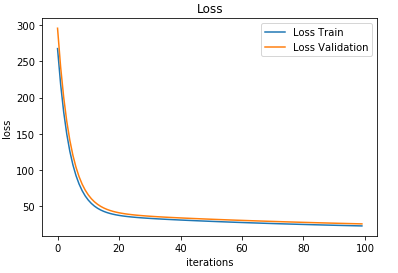
1.Linear regression:

## Hyper-parameter selection: α=10-3~10-6 iteration = 1000

## Predicted Results (Best Results):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LearningRate | 0.001 | 0.0001 | 0.00001 | 0.000001 |
| 1000iterations  LossTrain | 3872.76639757 | 3927.14056827 | 7120.97017544 | 19953.45240817 |
| 1000iterations  LossValidation | 2944.07378044 | 2905.2307854 | 5270.26519367 | 14938.71611779 |

## Loss curve:



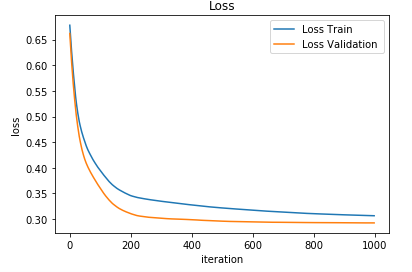
2.linear classcification: **α=10-3~10-5 C=1 teration=1000**

## Hyper-parameter selection:

## Predicted Results (Best Results):

|  |  |  |  |
| --- | --- | --- | --- |
| LearningRate | 0.001 | 0.0001 | 0.00001 |
| 1000iterations  LossTrain | 0.31485125 | 0.30740952 | 0.37643028 |
| 1000iterations  LossValidation | 0.2972224 | 0.28842528 | 0.31509189 |

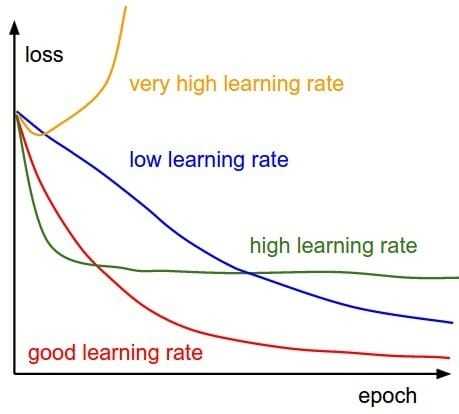
## Loss curve:



**11. Results analysis:**

Large learning rate may lead you to step too far, directly over the minimum mean we want; too small a learning rate will cause you to step too small, you may go a long way, in fact, Your goal is still a great distance.

Learning rate adjustment is the key to our gradient descent algorithm.



The choice of learning rate strategy in the network training process is constantly changing, in the beginning, the parameters are relatively random, so we should choose a relatively large learning rate, so that loss decreased faster; when training for some time, Parameter update should have a smaller amplitude, so the learning rate will generally do decay, attenuation is also very many ways, such as to a certain number of steps multiplied by the learning rate of 0.1, there are exponential decay.

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

**Similarities**:

1. all use gradient decent to find a better parameter
2. must use derivatives to get the parameters
3. one is for predict and the other is for classify

**Differences:**

1. Regression and classification problems different only in different ranges of their output. In the classification problem, the output is only allowed to take two values; in the regression problem, the output can take any real number
2. the output of regression is continuous while the classification’s is discrete.

**13. Summary**

This experiment shows the basic supervise learning machine of regression and classification, and also the importance of loss function and gradient decent , we could use linear regression to find a best parameter of function to predict the result of other feature .

Meanwhile , we could use SVM to classify the different features and figure out a Hyperplane. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-[probabilistic](https://en.wikipedia.org/wiki/Probabilistic_classification) [binary](https://en.wikipedia.org/wiki/Binary_classifier) [linear classifier](https://en.wikipedia.org/wiki/Linear_classifier).