

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **常沛炜**

**Student ID 201530611128**

**E-mail 326999206@qq.com**

**Tutor**   **MingkuiTan**

**Date submitted** **2017. 12 . 08**

1. **Topic:**

**Linear Regression, Linear Classification and Gradient Descent**

**2. Time: 2017.12.02**

**3. Reporter: 常沛炜**

**4. Purposes:**

**Further understand of linear regression and gradient descent.**

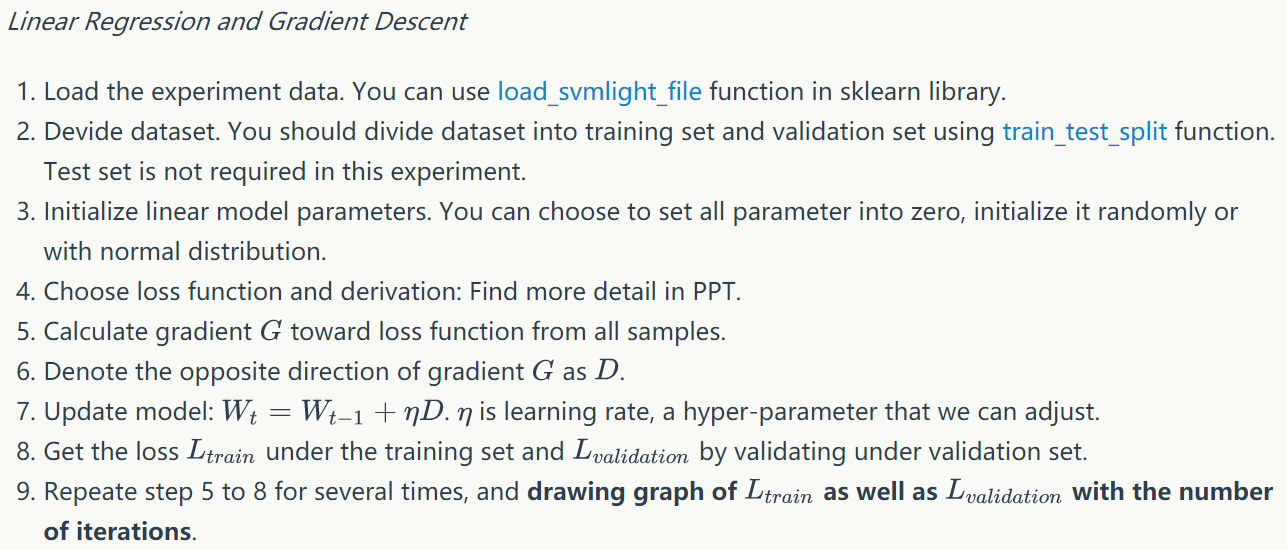
**Conduct some experiments under small scale dataset.**

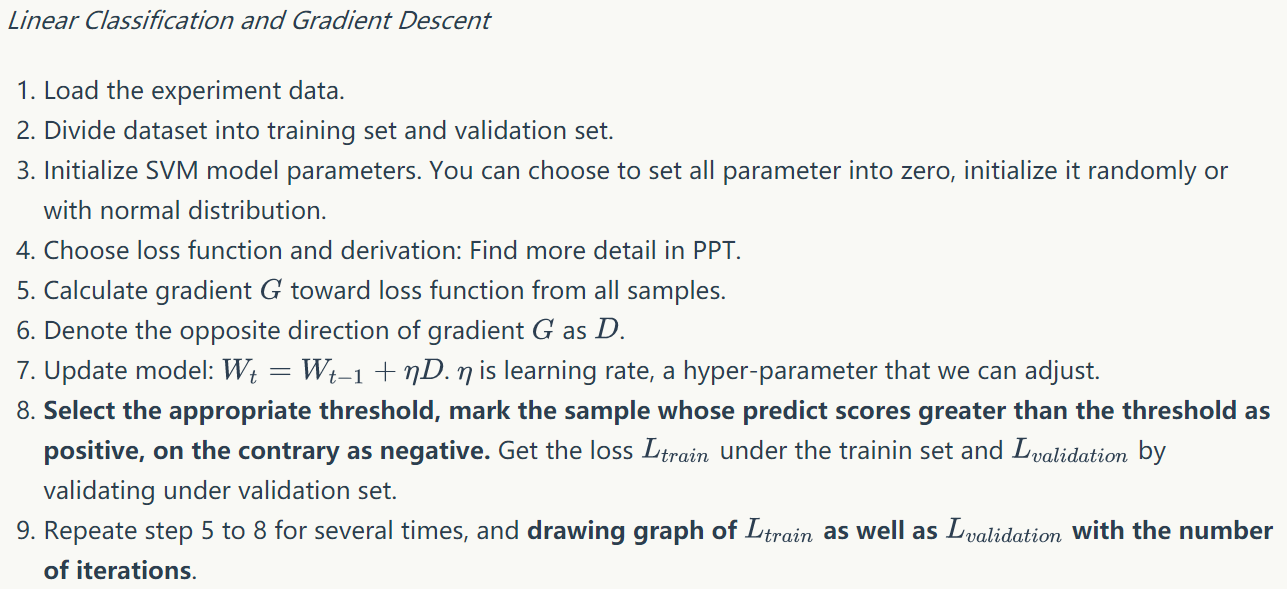
**Realize the process of optimization and adjusting parameters.**

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

**Use the data sets ‘australian\_scale.txt’ for classification and ‘housing\_scale.txt’ for regression in svm-light. Use the function load\_svmlight\_file() in sklearn to get the X and y.**

**6. Experimental steps:**





**7. Code:**

**Regression:**

from sklearn.datasets import load\_svmlight\_file  
from sklearn.model\_selection import train\_test\_split  
import scipy.sparse  
import numpy as np  
import matplotlib.pyplot as plt  
  
# load the data file  
data = load\_svmlight\_file('housing\_scale.txt')  
  
# split the data into training set and alidation set  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(data[0], data[1], test\_size=0.25, random\_state=42) # Y = MX + b  
X\_train = np.asarray(scipy.sparse.csr\_matrix(X\_train).todense())  
Y\_train = np.asarray(scipy.sparse.csr\_matrix(Y\_train).todense())  
X\_test = np.asarray(scipy.sparse.csr\_matrix(X\_test).todense())  
Y\_test = np.asarray(scipy.sparse.csr\_matrix(Y\_test).todense())  
  
row = X\_train.shape[0]  
column = X\_train.shape[1]  
testrow = X\_test.shape[0]  
testcolumn = X\_test.shape[1]  
  
# Initialize the parameter  
X = np.hstack((X\_train, np.ones((row, 1)))) # Let the last column in X to be 1  
learning\_rate = 0.0005  
W = np.zeros((column + 1, 1)) # Merge the W and b  
gradient\_rounds = 1000 # rounds for training  
  
# pyplot initial  
plt.figure(1)  
plt.subplot(111)  
xplot = []  
yplot = []  
yplotV = []  
for i in range(gradient\_rounds):  
 gradient = -np.dot(X.T, (np.dot(X, W) - Y\_train.T)) # Compute the gradient  
 W = W + np.reshape(learning\_rate \* gradient, (column + 1, 1)) # Update the W  
 xplot.append(i)  
 loss = 0  
 for each in range(row):  
 loss += pow(Y\_train.T[each] - np.dot(X[each], W), 2) # Compute the loss in train set  
 yplot.append(0.5 \* loss / row)  
 loss = 0  
 for each in range(testrow):  
 eachx = X\_test[each]  
 x = np.hstack((eachx, 1))  
 loss += pow((Y\_test.T[each] - np.dot(x, W)), 2) # Compute the loss in validation set  
 yplotV.append(0.5 \* loss / testrow)  
plt.title('During the training')  
plt.xlabel('Rounds')  
plt.ylabel('Loss')  
plt.plot(np.array(xplot), np.array(yplot),color="blue", linewidth=1.0, linestyle="-", label="Train Set")  
plt.plot(np.array(xplot), np.array(yplotV),color="red", linewidth=1.0, linestyle="-", label="Validation Set")  
  
plt.legend(loc='upper right')  
  
plt.show()

**Classification:**

import numpy as np  
from sklearn.datasets import load\_svmlight\_file  
from sklearn.model\_selection import train\_test\_split  
import scipy.sparse  
import matplotlib.pyplot as plt  
import warnings  
warnings.simplefilter(action = "ignore", category = RuntimeWarning)  
  
# load the data file  
data = load\_svmlight\_file("australian\_scale.txt")  
  
# split the data into training set and validation set  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(data[0], data[1], test\_size=0.33, random\_state=42)  
X\_train = np.asarray(scipy.sparse.csr\_matrix(X\_train).todense())  
Y\_train = np.asarray(scipy.sparse.csr\_matrix(Y\_train).todense())  
X\_test = np.asarray(scipy.sparse.csr\_matrix(X\_test).todense())  
Y\_test = np.asarray(scipy.sparse.csr\_matrix(Y\_test).todense())  
  
row = X\_train.shape[0]  
column = X\_train.shape[1]  
testrow = X\_test.shape[0]  
testcolumn = X\_test.shape[1]  
  
# Initialize the parameter  
X = np.hstack((X\_train, np.ones((row, 1)))) # Let the last column in X to be 1  
learning\_rate = 0.000001  
W = np.zeros((column + 1, 1)) # Merge the W and b  
gradient\_rounds = 2000 # rounds for training  
xplot = []  
yplot = []  
yplotlos=[]  
yplotV = []  
yplotVlos=[]  
for t in range(gradient\_rounds):  
 sum = 0  
 for each in range(row):  
 it = np.reshape(X[each], (1, column + 1)) # The random row  
 if (Y\_train.T[each] \* (np.dot(W.T, it.T))[0]) < 1: # max(0, Y\_train.T[random\_num] \* (np.dot(W.T, it.T)))  
 # then update the W  
 sum += np.reshape(np.dot(Y\_train.T[each], it),(column + 1, 1))  
 W+=learning\_rate\*sum  
 # Computing the correct  
 xplot.append(t)  
 correct = 0  
 rate = 0  
 loss=0  
 # Compute the correct rate in train set  
 for i in range(row):  
 loss+=max(0,1-Y\_train.T[i]\*np.dot(W.T,X[i]))  
 if np.dot(X[i], W)[0] > 0:  
 judge = True  
 else:  
 judge = False  
 if judge == (Y\_train.T[i] > 0):  
 correct += 1 # Hit!  
 yplotlos.append(loss/row)  
 rate = correct / row  
 yplot.append(rate)  
 correct = 0  
 rate = 0  
 loss=0  
 # Compute the correct rate in validation set  
 for i in range(testrow):  
 eachx = X\_test[i]  
 x = np.hstack((eachx, 1))  
 loss+=max(0, 1 - Y\_test.T[i] \* np.dot(W.T, x))  
 if np.dot(x, W)[0] > 0:  
 judge = True  
 else:  
 judge = False  
 if judge == (Y\_test.T[i] > 0):  
 correct += 1 # Hit!  
 yplotVlos.append(loss/testrow)  
 rate = correct / testrow  
 yplotV.append(rate)  
  
plt.figure(1)  
plt.subplot(111)  
plt.title('During the training')  
plt.xlabel('Rounds')  
plt.ylabel('Hit Rate')  
plt.plot(np.array(xplot), np.array(yplot), color="blue", linewidth=1.0, linestyle="-", label="Train Hit rate")  
plt.plot(np.array(xplot), np.array(yplotV), color="red", linewidth=1.0, linestyle="-", label="Validation Hit rate")  
plt.legend(loc='best')  
plt.figure(2)  
plt.subplot(111)  
plt.title('During the training')  
plt.xlabel('Rounds')  
plt.ylabel('Loss')  
plt.plot(np.array(xplot), np.array(yplotlos), color="blue", linewidth=1.0, linestyle="-", label="Train Loss")  
plt.plot(np.array(xplot), np.array(yplotVlos), color="red", linewidth=1.0, linestyle="-", label="Validation Loss")  
plt.legend(loc='best')  
  
  
plt.show()

**I also complete the SGD:**

random\_num = np.random.random\_integers(row - 1) # The random number for Stochastic gradient descent  
it = np.reshape(X[random\_num], (1, column + 1)) # The random row  
if (Y\_train.T[random\_num] \* (np.dot(W.T, it.T))[0]) < 1: # max(0, Y\_train.T[random\_num] \* (np.dot(W.T, it.T)))  
 # then update the W  
 W = (1 - learning\_rate / 3) \* W + learning\_rate \* np.reshape(np.dot(Y\_train.T[random\_num], it), (column + 1, 1))  
else:  
 # Nearly make the W unchanged  
 W = (1 - learning\_rate / 3) \* W

**the answer plot is below.**

(Fill in the contents of 8-12 respectively for linear regression and linear classification)

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

**Both of regression and classification, I use hold-out method for validation. I use the train\_test\_split() function in scipy to split the data X,y into train set and validation set. I set the test\_size=0.33 which means The train set will occupy 2/3 of the data.**

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

**Both of regression and classification, I set all parameter into zero but learning rate and training rounds. The each row of W\_(which means the parameter after merging the origin W and b) is zero.**

**By the way. In the classification, because I compute the gradient of all the training set, so the learning rate is a bit small.**

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

**Regression: Squared loss:**

**Which means the sum of each square of the distance of the real label and the prediction label.**

**Its gradient:**

**Classification: Hinge loss:**

**If the is lower than 0, then the gradient is 0 which means the round W will not change.**

**Else, the gradient:**

**11. Experimental results and curve:**

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

Regression: η: 0.0005 epoch: 1000

Classification: η: 0.000001 epoch:2000

## Assessment Results (based on selected validation):

Regression: Just compute the value of loss function(Squared loss).

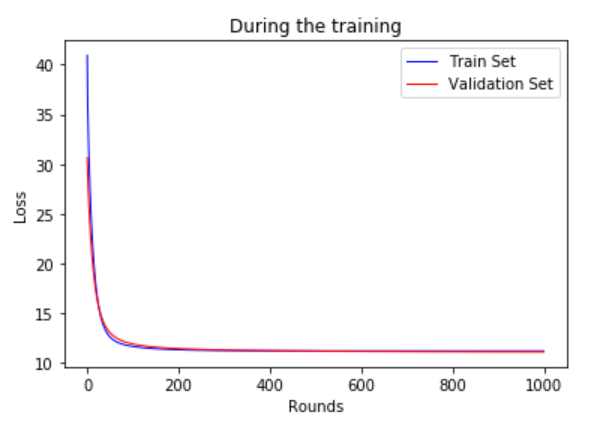
Classification: Use the hit rate(the rate of correct predict in all prediction)

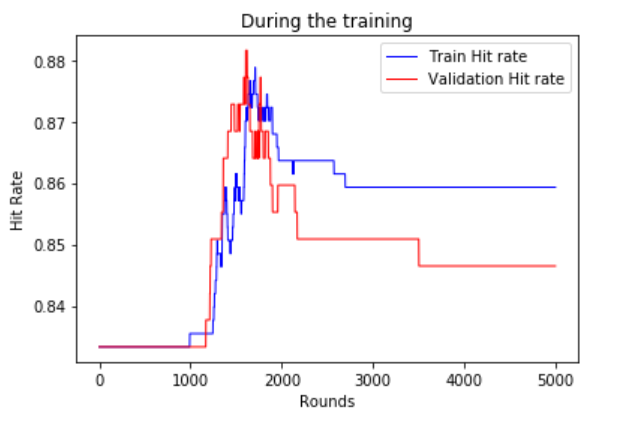
## Predicted Results (Best Results):

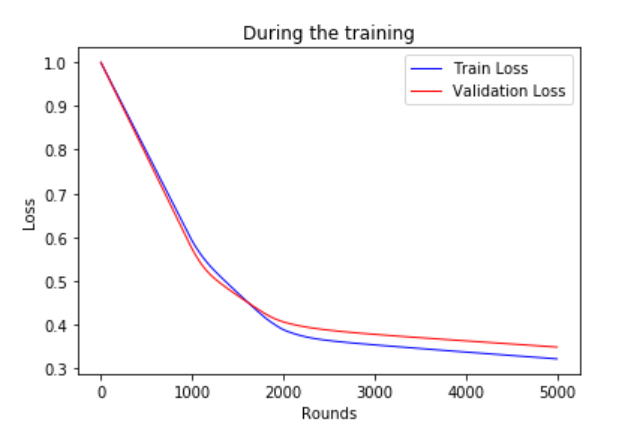
Regression: The squared loss(/rows) can be lowest about 12

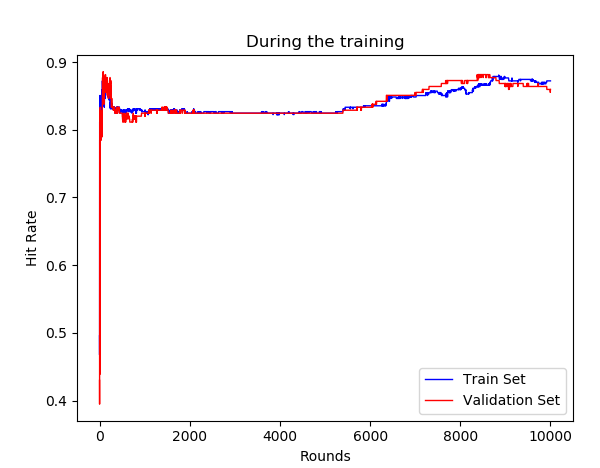
Classification: The hit rate can be up to 0.88

## Loss curve:

Regression: 

Classification: Hit rate: 

Loss: 

And the SGD version: 

**12. Results analysis:**

**The result of regression and classification is under my imagination. Both of them, the loss is decreasing during the training and the accuracy of models are increasing. Even neither of them reach a very high accuracy, that satisfy my thinking.**

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

**First to say is that the labels of linear regression are Continuity and discrete in linear classification. So in the SVM we just need to find a hyperplane to split the y=+1 and y=-1. However we should get the actual value in linear regression.**

**But by the gradient descent, the action in training step act similarly in them. Which to say that all we should do is compute the gradient and update the parameter based on learning rate. And in the coding process, I just got really obstacles while doing the regression but after finishing it, I can quickly solve the classification problem. It really makes sense.**

**14. Summary:**

**After the lab1, I understood the linear regression and linear classification deeper. And have a good feeling about the process of gradient descent, and accumulate more experience in training models and setting parameter, which will benefit me in the later experiment, learning and work.**

**I hope that I can learn more in the coming experiments.**