

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**  AXEL MAKONDA MBUTA

**Student ID 201722800013**

**E-mail axel.makonda@yahoo.fr**

**Tutor**  Mr MING KUI TAN

**Date submitted** **2017.12 . 08**

1. **Topic:**

LINEAR REGRESSION , LINEAR CLASSIFICATION AND GRADIENT DESCENT

**2. Time:** N/A

**3. Reporter:** AXEL MAKONDA MBUTA

**4. Purposes:**

1. Further understand of linear regression, linear classification and gradient descent.
2. Conduct some experiments under small scale dataset.
3. Realize the process of optimization and adjusting parameters.
4. **Data sets and data analysis:** For linear regression experiment we used the scaled edition of Housing dataset in [LIBSVM Data](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/), including 506 samples and each sample has 13 features. And for linear classification experiment , we used the scaled edition of the Australian dataset in LIBSVM Data,  including 690 samples and each sample has 14 features.
5. **Experimental steps:**

**A. Linear regression**

1. We load the experiment data. Using  [load\_svmlight\_file](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_svmlight_file.html" \t "_blank) function in sklearn library.
2. We devide dataset into training set and validation set using [train\_test\_split](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \t "_blank) function. Test set was not required in this experiment.
3. We randomly initialize linear model parameters using Torch.
4. We choose loss function and derivation:
5. We compute predicted y and Performs a matrix multiplication of the matrices mat1 and mat2.
6. We compute and print loss
7. We back-prop to compute gradients of w1 and w2 with respect to loss
8. We update weights using gradient descent
9. We print the curve to see the results.

**B. linear classification**

1. We load the experiment data.
2. We divide dataset into training set and validation set.
3. We define SVM function and initialize SVM model parameters, by choosing to set some parameters into zero and others randomly.
4. Choose loss function and derivation
5. We update the model and adjust the high-parameters
6. **We select the appropriate threshold and get L-train and L-validation set**
7. **We draw the graph**
8. **We plot the curve**
9. **Code:**
   1. **linear regression**

# LINEAR REGRESSION AND GRADIENT DESCENT by AXEL

#Checking the python version

import sys

print ('python: {}'.format(sys.version))

#Loading the libraries

import io

import os.path

import torch

import matplotlib.pyplot as plt

import numpy as np

import scipy.sparse as sp

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

from sklearn.model\_selection import train\_test\_split

#Loading the experiment data using load\_svmlight\_file function

#defining the function that get the data

def get\_data():

data = load\_svmlight\_file("https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/housing\_scale",n\_features=13)

return data

[0], data[1]

x,Y = get\_data()

x = data[0].toarray()

Y = data[1]

#Dividing the dataset

x\_train, x\_validation, Y\_train, Y\_validation = train\_test\_split(x, Y, test\_size=0.5, random\_state=43)

# S is batch size; D\_in is input dimension;D\_out is output dimension.

S,D\_in=x\_train.shape

D\_out=1

# We create random input and output data

Dtype = torch.DoubleTensor

# We randomly initialize weights

W = torch.randn(D\_in, D\_out).type(Dtype)

b = 0

x\_train=torch.from\_numpy(x\_train)

Y\_train=torch.from\_numpy(Y\_train)

x\_validation=torch.from\_numpy(x\_validation)

Y\_validation=torch.from\_numpy(Y\_validation)

#We choose loss function and derivation

def linear\_regression(x, Y, weight, bias):

Y\_prec = (weight\*x)+ bias

print (Y\_prec)

return Y\_prec

def loss\_function(x, Y, weight, bias, Iterations=500):

N = Float(len(Y))

total\_error = 0.0

for i in range(Iterations):

total\_error += (X[i] - (weight\*y[i] + bias))\*\*2

return total\_error/N

def linear\_regression(X, y, m=0, b=0, epochs=100, learning\_rate=0.0001):

N = float(len(y))

for i in range(epochs):

y\_prec = (m \* X) + b

loss = sum([data\*\*2 for data in (y-y\_prec)]) / N

loss = ((y\_prec - y) \*\* 2)

print ("loss:", loss)

return loss

eta = 1e-6

L\_train=[];

L\_validation=[];

for t in range(400000):

# Compute predicted y

and Performs a matrix multiplication of the matrices mat1 and mat2.

Y\_train\_pred = torch.add(x\_train.mm(W),b)

Y\_validation\_pred =torch.add( x\_validation.mm(W),b)

# Compute and print loss

L\_train.append ( (Y\_train\_pred - Y\_train).pow(2).sum())

L\_validation.append ( (Y\_validation\_pred - Y\_validation).pow(2).sum())

# Backprop to compute gradients of w1 and w2 with respect to loss

grad\_Y\_train\_pred = 2.0 \* (Y\_train\_pred - Y\_train)

grad\_W = 1/N \* x\_train.t().mm(grad\_Y\_train\_pred)

grad\_b = 1/N \* grad\_Y\_train\_pred.sum()

# We update weights using gradient descent

W -= eta \* grad\_w

b -= eta \* grad\_b

plt.plot(L\_train,'r',label='Ltrain')

plt.plot(L\_validation,'b',label='Lvalidation')

plt.legend()

plt.show()

* 1. **linear classification**

1. # coding: utf-8
2. # LINEAR CLASSIFICATION AND GRADIENT DESCENT by AXEL
3. #Checking the python version
4. import sys
5. print ('python: {}'.format(sys.version))
6. #Loading the libraries
7. import numpy
8. from numpy import random
9. import matplotlib.pyplot as plt
10. from sklearn.externals.joblib import Memory
11. from sklearn.datasets import load\_svmlight\_file
12. from sklearn.model\_selection import train\_test\_split
13. # Loading the experiment dataset
14. #defining thefunction that get the data
15. def get\_data():
16. data = load\_svmlight\_file("https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/australian\_scale")
17. return data
18. #Dividing the data into training and validating set
19. X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, Y, test\_size=0.4, random\_state=43)
20. N,D=x\_train.shape
21. C=len(list(set(y\_train)))
22. #defining the SVM function and,Initializing SVM model parameters
23. def svm(w, Xtrain, Ytrain, Xtest, Ytest, Reg):
24. gW = numpy.zeros(w.shape)
25. num\_classes = w.shape[1]
26. scores\_train = Xtrain.dot(w)
27. train\_loss = 0
28. num\_train = Xtrain.shape[0]
29. margins\_train = scores\_train - scores\_train\_correct + 1.0
30. margins\_train[numpy.arange(num\_train), Ytrain] = 0.0
31. margins\_train[margins\_train <= 0] = 0.0
32. margins\_train[margins\_train > 0] = 1.0
33. Scores\_train\_correct = scores\_train[numpy.arange(num\_train), Ytrain]
34. Scores\_train\_correct = numpy.reshape(scores\_train\_correct, (num\_train, 1))
36. train\_loss += numpy.sum(margins\_train) / num\_train
37. train\_loss += 0.5 \* Reg \* numpy.sum(w \* w)
39. row\_sum = numpy.sum(margins\_train, axis=1)
40. margins\_train[numpy.arange(num\_train), Ytrain] = -row\_sum
41. gW += numpy.dot(Xtrain.T, margins\_train)/num\_train + Reg \* w
43. test\_loss = 0
44. Scores\_test = Xtest.dot(w)
45. num\_test = Xtest.shape[0]
46. Scores\_test\_correct = Scores\_test[numpy.arange(num\_test), Ytest]
47. Scores\_test\_correct = numpy.reshape(Scores\_test\_correct, (num\_test, 1))
48. margins\_test = Scores\_test - Scores\_test\_correct + 1.0
49. margins\_test[numpy.arange(num\_test), Ytest] = 0.0
50. margins\_test[margins\_test <= 0] = 0.0
51. test\_loss += numpy.sum(margins\_test) / num\_test
52. test\_loss += 0.5 \* reg \* numpy.sum(w \* w)
53. return train\_loss, test\_loss, gW
54. data = get\_data()
55. X=data[0].toarray()
56. Y=data[1]
57. Y=Y.reshape(len(Y),order='C')
58. Y=Y.astype(numpy.int)
59. w = random.random(size=(D, C))
60. maxIters=500
61. th = 0
62. eta = 0.001
63. L\_train=[];
64. L\_test=[];
66. for t in range(maxIters):
67. Y\_train\_pred = numpy.dot(x\_train,W)
68. Y\_train\_pred[y\_train\_pred> th] = 1
69. Y\_train\_pred[y\_train\_pred<=th] = 0
71. Y\_test\_pred = numpy.dot(x\_test,W)
72. Y\_test\_pred[y\_test\_pred> th] = 1
73. Y\_test\_pred[y\_test\_pred<=th] = 0
75. train\_loss, test\_loss, grad\_W= svm(w, X\_train, Y\_train, X\_test, Y\_test, Reg= 0.1)
77. L\_train.append (train\_loss)
78. L\_test.append (test\_loss)
79. w -= eta \* grad\_W
80. plt.plot(L\_train,'r',label='train loss')
81. plt.plot(L\_test,'b',label='test loss')
82. # We give the plot a title
83. plt.title('Loss Curve')
84. plt.legend()
85. plt.show()

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

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

**-**For the both linear regression and classification we used cross-validation and separate the dataset into two parts

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

**-**in linear regression we randomly initialize the model parameters

-in linear classification , we choose to set some model parameters to zero and others were initialized randomly.

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

For a classification , a fundamental loss function is the 0-1-loss:

L(y,t) = {01if y = totherwiseL(y,t) = {0if y = t1 otherwise

For regression, the squared loss is :

L(y,t)=(y−t)2L(y,t)=(y−t)2

**:**

And  as its derivatives

**11. Experimental results and curve:**

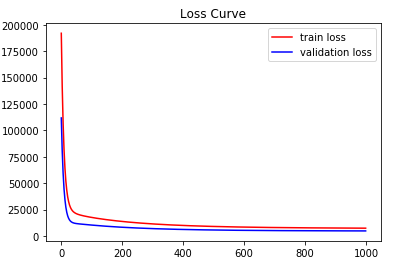
Hyper-parameter selection (η, epoch, etc.):

## Assessment Results (based on selected validation):

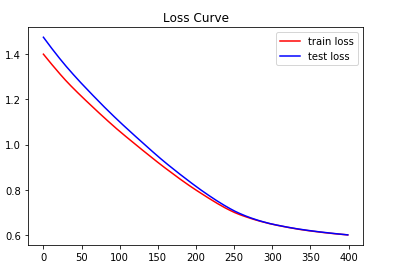
## Predicted Results (Best Results):

## Loss curve:

**Linear Regression:**



**Linear classification**



**12. Results analysis:**

The bigger is the iteration number ,the better is the result that means that in this experiments we should also care about the number of the iterations to get a better result.

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

-The validation procedures are commonly applied in regression and classification methods.

- Linear classification uses SVM function

Regression is continuous, the classification is discrete, which is the difference. The similarity is that all are supervised.

**14. Summary:**

we did our experiments in linear regression and linear classification using the scaled edition of housing and australian datasets

-we started by downloading the dataset and dividing it into training and validation set.

-we defined the models and initialized them.

-we choosed the loss functions

-we updated the models and plotted the curves

And in the end we analyzed the results