

**The Experiment Report of**

***Deep Learning***

**College: Software Engineering**

**Subject: Deep Learning**

**E-mail: nabeel10561@gmail.com**

**Tutor: Ming Kui tan**

**Date-submitted:2017.12.08**

**1. Topic:**Linear Regression, Linear Classification and gradient descent

**2. Time:** Friday 08,2017

**3. Reporter:**Muhammad Nabeel

**4. Purposes:**

1. Further understanding of linear regression and gradient decent
2. Conduct some gradient under small scale data-set
3. Realize the process of optimization and Adjusting Parameters

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

1. Housing\_scaled Data (Experiment one)
2. Australian\_scaled Data (Experiment two)

**6. Experimental steps:**

**Experiment: 01**

1. Load the experiment data. You can use [load\_svmlight\_file](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_svmlight_file.html) function in sklearn library.
2. Divide dataset. You should divide 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) function. Test set is not required in this experiment.
3. Initialize linear model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
4. Choose loss function and derivation.
5. Calculate gradient  toward loss function from all samples.
6. Denote the opposite direction of gradient G as D.
7. Update model:where define the 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  our loss function as well as  with the number of iterations**.

**Experiment: 02**

1. Load the experiment data. You can use [load\_svmlight\_file](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_svmlight_file.html) function in sklearn library.
2. Divide dataset. You should divide 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) function. Test set is not required in this experiment.
3. Initialize linear model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
4. Choose loss function and derivation.
5. Calculate gradient  toward loss function from all samples.
6. Denote the opposite direction of gradient G as D.
7. Update model:where define the 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  our loss function as well as  with the number of iterations**.

(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.):**

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

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

**11. Experimental results and curve**

Experiment code and result ：01

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

import numpy as np

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

import matplotlib.pyplot as plt

alpha = 0.001

iter = 110

accuracy = 0.001

m = 506

m\_train = 203

m\_test = 203

features=13

theta=[0,0,0,0,0, 0,0,0,0,0, 0,0,0,0]

iter\_num = [1]\*iter;

loss\_train = [1]\*iter;

loss\_test = [1]\*iter;

def get\_data():

data = load\_svmlight\_file(r'''C:\Users\HP\Desktop\housing\_scale.txt''',n\_features=13)

return data[0], data[1]

X, y = get\_data()

X = X.toarray()

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.5, random\_state=43)

print(X\_train)

print(y\_train)

def hypothesis(x):

result = theta[0]

for i in range (0,features):

result = result + theta[i+1] \* x[i]

return result

def loss(m,X,y):

sum=0

for i in range(0,m):

sum = sum + ( hypothesis(X[i]) - y[i] ) \*\*2

sum = sum / (2\*m)

return sum

def derivative(j,m,X,y):

sum=0

if(j==0):

for i in range(0,m):

sum = sum + ( hypothesis(X[i]) - y[i] )

else:

for i in range(0,m):

sum = sum + ( hypothesis(X[i]) - y[i] ) \* X[i][j-1]

sum = sum / m

return sum

def train():

for i in range(0,iter):

for j in range(0,features+1):

theta[j] = theta[j] - alpha \* derivative(j,m\_train,X\_train,y\_train)

iter\_num[i] = i;

loss\_train[i] = loss(m\_train,X\_train,y\_train);

loss\_test[i] = loss(m\_test,X\_test,y\_test)

def information():

print("loss on train:", loss\_train)

print("loss on test",loss\_test)

train()

information()

fig, ax = plt.subplots()

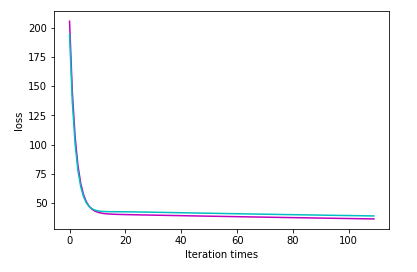
ax.plot(iter\_num, loss\_train,color = 'm', label='loss of train')

ax.plot(iter\_num, loss\_test, color = 'c', label='loss of test')

ax.set\_xlabel('Iteration times')

ax.set\_ylabel('loss')

plt.show()



Experiment code and result ： 02

# coding: utf-8

import numpy

# coding: utf-8

import numpy

from numpy import random

import matplotlib.pyplot as plt

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

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

# Load the experiment data

mem = Memory("./mycache")

@mem.cache

def get\_data():

data = load\_svmlight\_file(r'''C:\Users\HP\Desktop\seconddata.txt''',n\_features=14)

return data

def svm(W, xtrain, ytrain, xtest, ytest, reg):

gW = numpy.zeros(W.shape)

num\_classes = W.shape[1]

train\_loss = 0

scores\_train = xtrain.dot(W)

num\_train = xtrain.shape[0]

scores\_train\_correct = scores\_train[numpy.arange(num\_train), ytrain]

scores\_train\_correct = numpy.reshape(scores\_train\_correct, (num\_train, 1))

margins\_train = scores\_train - scores\_train\_correct + 1.0

margins\_train[numpy.arange(num\_train), ytrain] = 0.0

margins\_train[margins\_train <= 0] = 0.0

train\_loss += numpy.sum(margins\_train) / num\_train

train\_loss += 0.5 \* reg \* numpy.sum(W \* W)

margins\_train[margins\_train > 0] = 1.0

row\_sum = numpy.sum(margins\_train, axis=1)

margins\_train[numpy.arange(num\_train), ytrain] = -row\_sum

gW += numpy.dot(xtrain.T, margins\_train)/num\_train + reg \* W

test\_loss = 0

scores\_test = xtest.dot(W)

num\_test = xtest.shape[0]

scores\_test\_correct = scores\_test[numpy.arange(num\_test), ytest]

scores\_test\_correct = numpy.reshape(scores\_test\_correct, (num\_test, 1))

margins\_test = scores\_test - scores\_test\_correct + 1.0

margins\_test[numpy.arange(num\_test), ytest] = 0.0

margins\_test[margins\_test <= 0] = 0.0

test\_loss += numpy.sum(margins\_test) / num\_test

test\_loss += 0.5 \* reg \* numpy.sum(W \* W)

return train\_loss, test\_loss, gW

data = get\_data()

X=data[0].toarray()

Y=data[1]

Y=Y.reshape(len(Y),order='C')

Y=Y.astype(numpy.int)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=42)

N,D=x\_train.shape

C=len(list(set(y\_train)))

W = random.random(size=(D, C))

maxIterations=400

th = 0

eta = 0.001

L\_train=[];

L\_test=[];

for t in range(maxIterations):

y\_train\_pred = numpy.dot(x\_train,W)

y\_train\_pred[y\_train\_pred> th] = 1

y\_train\_pred[y\_train\_pred<=th] = 0

y\_test\_pred = numpy.dot(x\_test,W)

y\_test\_pred[y\_test\_pred> th] = 1

y\_test\_pred[y\_test\_pred<=th] = 0

train\_loss, test\_loss, grad\_W= svm(W, x\_train, y\_train, x\_test, y\_test, reg= 0.1)

L\_train.append (train\_loss)

L\_test.append (test\_loss)

W -= eta \* grad\_W

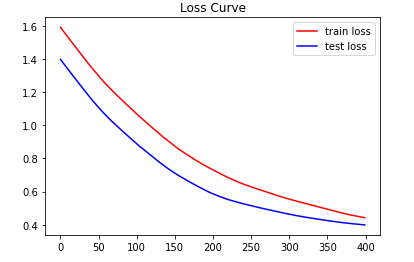
plt.plot(L\_train,'r',label='train loss')

plt.plot(L\_test,'b',label='test loss')

plt.title('Loss Curve') # give plot a title

plt.legend()

plt.show()



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

## Assessment Results (based on selected validation):

## Predicted Results (Best Results):

## Loss curve:

1. **Results analysis:**

For first experiment the estimated loss function as almost near the real value, which indicates its a better gradient descent and at the values:

alpha = 0.001

iter = 110

accuracy = 0.001.

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

Regression and classification are both related to prediction, where regression predicts a value from a continuous set, whereas classification predicts the 'belonging' to the class.For example, the price of a house depending on the 'size' (in some unit) and say 'location' of the house, can be some 'numerical value' (which can be continuous): this relates to regression.

Similarly, the prediction of price can be in words, viz., 'very costly', 'costly', 'affordable', 'cheap', and 'very cheap': this relates to classification.

**Regression:** given a set of data, find the best relationship that represents the set of data.

**Classification:** given a known relationship, identify the class that the data belongs to.

We can see that regression and classification start from opposing ends: to find a pattern or to find the pattern that it belongs to.

**14.Summary:**

Linear **regression** is a prediction when a variable (y) is dependent on a second variable (x) based on the**regression** equation of a given set of data. ... The stronger the relationship between the two sets of variables, the more likely your prediction will be accurate.

**Regression problems** are those where you are trying to predict or explain one thing (dependent variable) using what you know about other things (independent variables). That covers anything that can be expressed as numbers, probabilities, or true/false answers. Linear regression is intended more specifically for continuous numbers, and was first proposed by Adrien-Marie Legendre in 1805.

**Classification Problems,** **Classification** is a central topic in machine learning that has to do with teaching machines how to group together data by particular criteria. **Classification** is the process where computers group data together based on predetermined characteristics — this is called supervised learning.

**Classification problems** try to determine group membership by deriving probabilities. The first technique ever used was linear discriminant analysis (LDA), proposed by Sir R.A. Fisher in 1936—he used to to classify irises. I do not understand it fully, but believe that it used linear regression to derive probabilities for each group, and then used a Mahalanobis distance measure to assign to the closest group. One does not here much about LDA any more, I think because is has been supplanted by logistic regression.