South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Hao Zeng

Supervisor:
Qingyao Wu

Student ID：
201530613559

Grade:
Undergraduate

December 13, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—

I. Introduction

A. Motivation of Experiment

1. Compare and understand the difference between gradient descent and stochastic gradient descent.
2. Compare and understand the differences and relationships between Logistic regression and linear classification.
3. Further understand the principles of SVM and practice on larger data.

II. Methods and Theory

$$g(z) = \frac{1}{1+e^{-z}}$$

SGD:

$$\mathbf{g}_t \leftarrow \nabla J_i(\boldsymbol{\theta}_{t-1})$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$$

NAG:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \gamma \mathbf{v}_{t-1})$$
$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

RMSProp:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

AdaDelta:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\Delta\boldsymbol{\theta}_t \leftarrow -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t$$
$$\Delta_t \leftarrow \gamma \Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t$$

Adam:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t + \epsilon}}$$

III. Experiment

A. Dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

B. Environment for Experiment

python3, at least including following python package: sklearn, numpy, jupyter, matplotlib It is recommended to install anaconda3 directly, which has built-in python package above.

C. Experiment Step

The experimental code and drawing are completed on jupyter.

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize logistic regression model parameters, you can consider initalizing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
4. Calculate gradient toward loss function from **partial samples**.
5. **Update model parameters using different optimized methods(NAG, RMSProp, AdaDelta and Adam).**
6. Select the appropriate threshold, mark the sample whose predict scores **greater than the threshold as positive, on the contrary as negative**. Predict under validation set and get the different optimized method loss LNAG , LRMSProp, LSdaDelta and . LAdam
7. Repeate step 4 to 6 for several times, and **drawing graph of , , and with the number of iterations**.

Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.
3. Select the loss function and calculate its derivation, find more detail in PPT.
4. Calculate gradient toward loss function from **partial samples**.
5. **Update model parameters using different optimized methods(NAG, RMSProp, AdaDelta and Adam).**
6. Select the appropriate threshold, mark the sample whose predict scores **greater than the threshold as positive, on the contrary as negative**. Predict under validation set and get the different optimized method loss LNAG, LRMSProp, LSdaDelta and LAdam.
7. Repeate step 4 to 6 for several times, and **drawing graph of , , and with the number of iterations**.

D. Result

- Logistic Regression and Stochastic Gradient Descent

1. Source code:

```
import numpy as np
from numpy import *
from sklearn.externals.joblib import Memory
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import matplotlib.pylab as plt
from matplotlib import *
import random
```

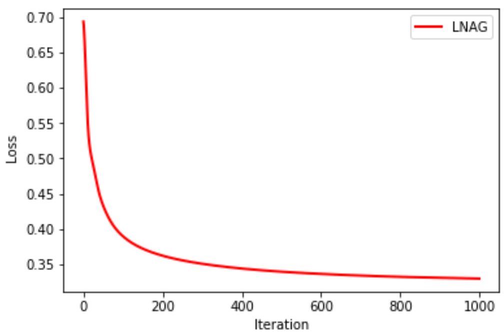
```
    train =
load_svmlight_file("a9a.txt"
    validation =
load_svmlight_file('a9a.t',
n_features=123)

    X_train = train[0].toarray()
    y_train =
train[1].reshape(X_train.shape[0],
1)

    X_validation =
validation[0].toarray()
    y_validation =
validation[1].reshape(X_validation.
shape[0], 1)

    X_train =
concatenate((ones((X_train.shape[0],
1), dtype='float'), X_train),
axis=1)
X_validation =
concatenate((ones((X_validation.sha
pe[0], 1), dtype='float'),
X_validation), axis=1)
    Y_train=y_train
    for i in range(Y_train.size):
        if Y_train[i]==-1:
            Y_train[i]=0
    Y_validation=y_validation
    for i in
range(Y_validation.size):
        if y_validation[i]==-1:
        Y_validation[i]=0
    def sigmoid(x):
    return 1/(1 + exp(-x))
    ef loss(x, y, w):
        return
-(dot(y.T,log(sigmoid(x.dot(w)))) +
(1 - y).T.dot(log(1 -
sigmoid(x.dot(w)))))[0]
def
NAG(xt,yt,xv,yv,w,eta,times,batch_size,gam
ma):
    v = zeros((xt.shape[1], 1))
    LNAG = []
    for i in range(times):
        sample_index =
random.sample(list(arange(0,yt.size)),batc
h_size)
        x = xt[sample_index, :]
        y = yt[sample_index, :]
        LNAG.append(loss(xv, yv, w) /
yv.size)
        g = x.T.dot(sigmoid(np.dot(x, w -
gamma * v)) - y) / y.size
        v = gamma * v +eta*g
        w = w - v
    return LNAG
        w=zeros((X_train.shape[1], 1))#
    参数全零初始化
        eta=0.01  #学习率
        times=1000  #学习次数
    L_NAG=NAG(X_train,Y_train,X_validat
    ion,Y_validation,w,eta,times,5000,0
    .9)
        def RMSProp(xt, yt, xv,
    yv,w,eta,times,batch_size,gamma,eps
    ilon):
            G = zeros((xt.shape[1], 1))
            LRMSProp = []
            for i in range(times):
                sample_index =
    random.sample(list(np.arange(0,
    yt.size)), batch_size)
                x = xt[sample_index, :]
                y = yt[sample_index, :]
                LRMSProp.append(loss(xv,
    yv, w) / yv.size)
                g =
    x.T.dot(sigmoid(dot(x, w)) - y) /
    y.size
                G = gamma * G + (1 - gamma)
    * g * g
                w = w - eta/sqrt(G +
    epsilon) * g
        return LRMSProp
        w=zeros((X_train.shape[1], 1))
```

```
    alpha=0.01  #学习率
    times=1000  #学习次数
    L_RMSProp=RMSProp(X_train,Y_train,X_validation,Y_validation,w,alpha,times,5000,0.9,1e-8)
    def AdaDelta(xt,yt,xv,yv,w,times,batch_size,gamma,epsilon):
        delta = zeros((xt.shape[1], 1))
        G = zeros((xt.shape[1], 1))
        LAdaDelta = []
        for i in range(times):
            sample_index = random.sample(list(arange(0, yt.size)), batch_size)
            x = xt[sample_index, :]
            y = yt[sample_index, :]
            LAdaDelta.append(loss(xv, yv,w) / yv.size)
            g = x.T.dot(sigmoid(np.dot(x,w)) - y) / y.size
            G = gamma * G + (1 - gamma) * g * g
            delta_w = -sqrt(delta + epsilon)/sqrt(G + epsilon) * g
            w = w+ delta_w
            delta = gamma * delta + (1 - gamma) * delta_w * delta_w
        return  LAdaDelta
    w=zeros((X_train.shape[1], 1))
    times=1000  #学习次数
    L_AdaDelta=AdaDelta(X_train,Y_train,X_validation,Y_validation,w,times,5000,0.95,1e-8)
    def Adam(xt,yt,xv,yv,w,eta,times,batch_size,beta,gamma,epsilon):
        G = zeros((xt.shape[1],1))
        moments= zeros((xt.shape[1],1))
        LAdam = []
        for i in range(times):
            sample_index = random.sample(list(arange(0, yt.size)), batch_size)
            x = xt[sample_index, :]
            y = yt[sample_index, :]
            LAdam.append(loss(xv, yv,w) / yv.size)
            g = x.T.dot(sigmoid(np.dot(x, w)) - y) / y.size
            moments = beta * moments + (1.0 - beta) * g
            G = gamma * G + (1.0 - gamma) * g * g
            alpha = eta *sqrt(1.0 - gamma**(i + 1)) / (1.0 - beta**(i + 1))
            w = w - alpha * moments / sqrt(G + epsilon)
        return LAdam
    w=zeros((X_train.shape[1], 1))#参数全零初始化
    eta=0.01  #学习率
    times=1000  #学习次数
    L_Adam=Adam(X_train,Y_train,X_validation,Y_validation,w,eta,times,5000,0.9,0.95,1e-8)
    count=arange(0,1000)
    x = count
    y1 = L_NAG
    y2=L_RMSProp
    y3=L_AdaDelta
    y4=L_Adam
    plt.plot(x,y1,"r-",linewidth=2, label='LNAG')
    plt.plot(x,y2,"y-",linewidth=2, label='LRMSProp')
    plt.plot(x,y3,"b-",linewidth=2, label='LAdaDelta')
    plt.plot(x,y4,"g-",linewidth=2, label='LAdam')
    plt . ylabel (" loss"  )
    plt . xlabel ("Iteration"  )
    plt.legend()
```

```
plt.show()
```

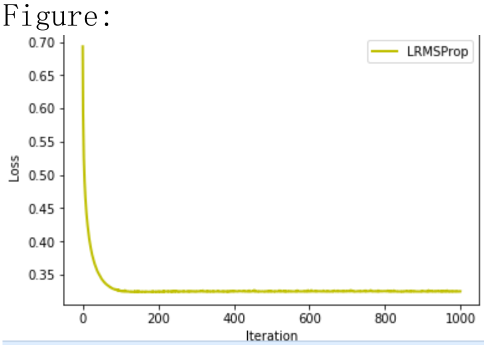
2. Parameter:
   Learning_rate=0.01Iteration=1000
   Gamma=0.9
   Batch_size=5000
   Eplison=1e-8

3. Model of NAG
   No eplison
   Figure:

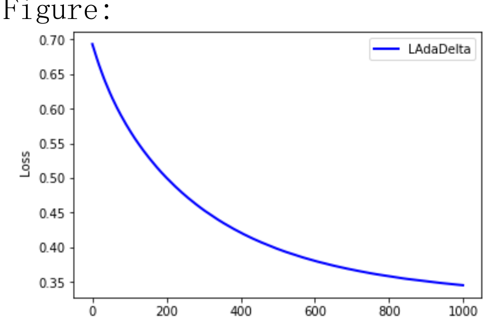


o

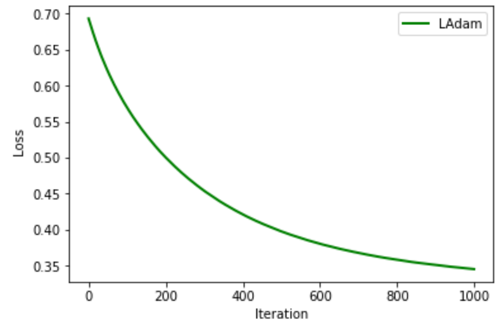
Model of RMSProp:
Learning_rate=0.01
Figure:



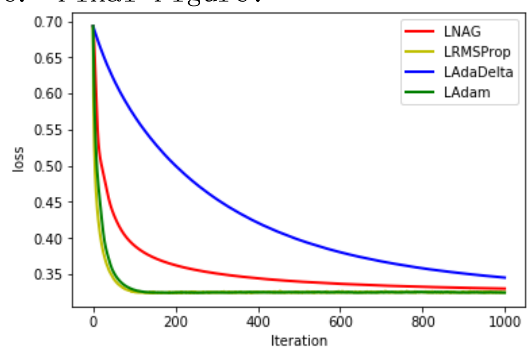
4. Model of AdaDelta
   No learning_rate
   Gamma=0.95
   Figure:



5. Model of Adam:
   Beta=0.9
   Gamma=0.95
   Figure:



6. Final Figure:



- Linear Classification and Stochastic Gradient Descent

1. Source code:

```
import numpy as np

from numpy import *

from sklearn.externals.joblib import Memory

from sklearn.datasets import load_svmlight_file

from sklearn.model_selection import train_test_split

import matplotlib.pylab as plt
```

```
from matplotlib import *

import random

X_train,y_train =load_svmlight_file('a9a.txt',n_features=123)

X_train = X_train.toarray()

X_validation,y_validation = load_svmlight_file('a9a.t',n_features=123)

X_validation= X_validation.toarray()

def loss(X,y,W,b,C):

    data_loss = 0.5*np.sum( W **2 ) + C *np.sum(np.maximum(0, 1-y*(np.dot(X,W)+b) ))

return data_loss/y.size

def gradient(X,y,w,b,C):

    margin = 1 - y * (np.dot(X,w)+b )

    y_tmp = -y

    y_tmp[margin<0] = 0

    dw =  w + C / y.size * np.dot(X.T,y_tmp)

    db = C / y.size * np.sum(y_tmp)

return dw, db

def NAG(xt,yt,w,b,eta,times,xv,yv,C=1,batch_size=1000):

    LNAG = []

    yy = 0.9

    vw = np.zeros(w.shape)

    vb = 0

    for i in range(times):

        random = list(set(np.random.randint(0,yt.size,size=batch_size)))

        dx = xt[random]

        dy = yt[random]

        w_gt,b_gt = gradient(dx,dy,w-yy*vw ,b-yy*vb ,C)

        vw = yy*vw + eta * w_gt

        w = w - vw

        vb = yy*vb + eta * b_gt

        b = b - vb

        LNAG.append( loss(xv,yv,w,b,C) )

return LNAG

eta=0.001

times=700

w = np.zeros(X_train.shape[1]).T

b=0

C = 50

batch_size = 3000

L_NAG = NAG(X_train,y_train,w,b,eta,times,X_validation,y_validation,C=C,batch_size=batch_size)
```

```
    plt.plot(np.arange(num_rounds),L_NAG,"r-",linewidth=2,label='LNAG')

    plt.legend(loc=1)

    plt.xlabel('Irelation')

    plt.ylabel('loss')

plt.show()

    def RMSProp(xt,yt,w,b,alpha,num_rounds,xv,yv,C,batch_size):

      LRMSProp = []

      yy = 0.9

      w_Gt = np.zeros(w.shape)

      b_Gt = 0

      e = 1e-9

      gama = 0.001

      for i in range(num_rounds):

        random = list(set(np.random.randint(0,X_train.shape[0],size=batch_size)))

        dx = X_train[random]

        dy = y_train[random]

        w_gt,b_gt = gradient(dx,dy,w,b,C=C)

        w_Gt = yy * w_Gt + (1-yy) * ( w_gt**2)

        w = w - gama / np.sqrt(w_Gt+e) * w_gt

        b_Gt = yy * b_Gt + (1-yy) * ( b_gt**2)

        b = b - gama / np.sqrt(b_Gt+e) * b_gt

 LRMSProp.append( loss(X_test,y_test,w,b,C=C) )

 return LRMSProp

    eta=0.001

    times=700

    w = np.zeros(X_train.shape[1]).T

    b=0

    C = 50

    batch_size = 3000

L_RMSProp= RMSProp(X_train,y_train,w,b,eta,times,X_validation,y_validation,C=C,batch_size=batch_size)

    plt.plot(np.arange(num_rounds),L_RMSProp,"y-",linewidth=2,label='LRMSProp')

    plt.legend(loc=1)

    plt.xlabel('Irelation')

    plt.ylabel('loss')

plt.show()

    def AdaDelta(xt,yt,w,b,eta,num_rounds,xv,yv,C,batch_size):

      train_loss_history = []
```

```
    test_loss_history = []

    yy = 0.95

    w_Gt = np.zeros(w.shape)

    b_Gt = 0

    e = 1e-6

    wt = np.zeros(w.shape)

    bt = 0

    for i in range(times):

      random =
list(set(np.random.randint(0,X_train.shape[0],size=batch_size)))

      dx = X_train[random]

      dy = y_train[random]

      w_gt,b_gt =
gradient(dx,dy,w,b,C)

      w_Gt = yy * w_Gt + (1-yy) *
( w_gt**2)

      wdw = - ( np.sqrt(wt+e) /
np.sqrt(w_Gt+e) ) * w_gt

      w = w + wdw

      wt = yy * wt + (1-yy) * ( wdw**2)

      b_Gt = yy * b_Gt + (1-yy) *
( b_gt**2)

      bdw = - ( np.sqrt(bt+e) /
np.sqrt(b_Gt+e) ) * b_gt

      b = b + bdw

      bt = yy * bt + (1-yy) * ( bdw**2)

 test_loss_history.append( loss(X_test,
y_test,w,b,C) )

 return test_loss_history

    eta=0.001

    times=700

    w = np.zeros(X_train.shape[1]).T

    b=0

    C = 50

    batch_size = 3000

L_AdaDelta =
AdaDelta(X_train,y_train,w,b,eta,times,X_validation,y_validation,C=C,batch_size=batch_size)

    plt.plot(np.arange(num_rounds),L_AdaDelta,"b-",linewidth=2,label='LAdaDelta')

    plt.legend(loc=1)

    plt.xlabel('Irelation')

    plt.ylabel('loss')

plt.show()

    def
Adam(xt,yt,w,b,eta,times,xv,yv,C,batch_size):

      LAdam = []

      beta1 = 0.9

      yy = 0.999

      gama = 1e-3
```

```
e = 1e-8
wm = np.zeros(w.shape)
w_Gt = np.zeros(w.shape)
bm = 0
b_Gt = 0
for i in range(times):
  random = list(set(np.random.randint(0,yt.size,size=batch_size)))
  dx = xt[random]
  dy = yt[random]
  w_gt,b_gt = gradient(dx,dy,w,b,C=C)
  wm = beta1 * wm + (1-beta1) * w_gt
  w_Gt = yy * w_Gt + (1-yy) * ( w_gt**2)
  alp = gama * np.sqrt(1-yy**(i+1) ) / (1-beta1**(i+1) )
  w = w - alp * wm / np.sqrt(w_Gt + e)
  bm = beta1 * bm + (1-beta1) * b_gt
  b_Gt = yy * b_Gt + (1-yy) * ( b_gt**2)
  alp = eta * np.sqrt(1-yy**(i+1) ) / (1-beta1**(i+1) )
  b = b - alp * bm / np.sqrt(b_Gt + e)
LAdam.append( loss(xv,yv,w,b,C) )
return LAdam
eta=0.001
times=700
w = np.zeros(X_train.shape[1]).T
b=0
C = 50
batch_size = 3000
L_Adam = Adam(X_train,y_train,w,b,eta,times,X_validation,y_validation,C,batch_size)
plt.plot(np.arange(num_rounds),L_Adam,"g-",linewidth=2,label='LAdam')
plt.legend(loc=1)
plt.xlabel('Irelation')
plt.ylabel('loss')
plt.show()
plt.plot(np.arange(num_rounds),L_NAG,"r-",linewidth=2,label='LNAG')
plt.plot(np.arange(num_rounds),L_RMSProp,"y-",linewidth=2,label='LRMSProp')
plt.plot(np.arange(num_rounds),L_AdaDelta,"b-",linewidth=2,label='LAdaDelta')
plt.plot(np.arange(num_rounds),L_Adam,"g-",linewidth=2,label='LAdam')
```

```
    plt.legend(loc=1)

    plt.xlabel('Irelation')

    plt.ylabel('loss')

plt.show()
```

2. Parameter:

eta=0.001

times=700

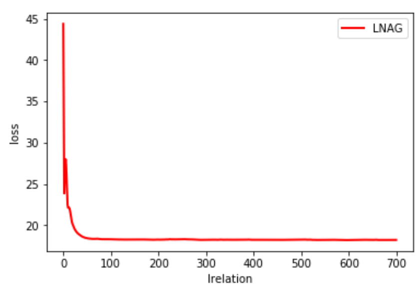
w = np.zeros(X_train.shape[1]).T

b=0

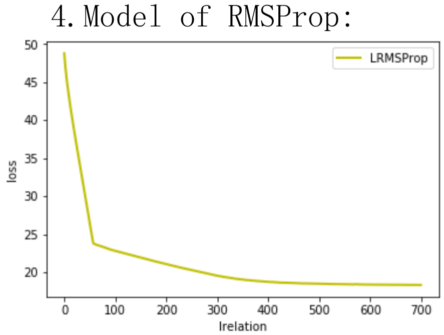
C = 50

batch_size = 3000

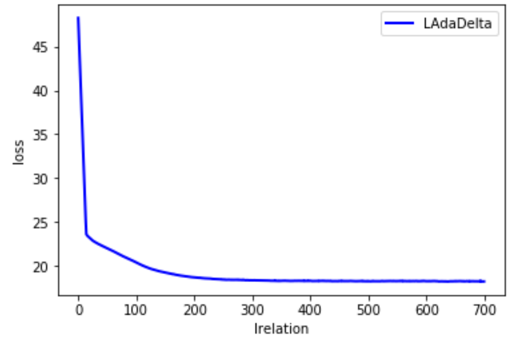
3. Model of NAG:



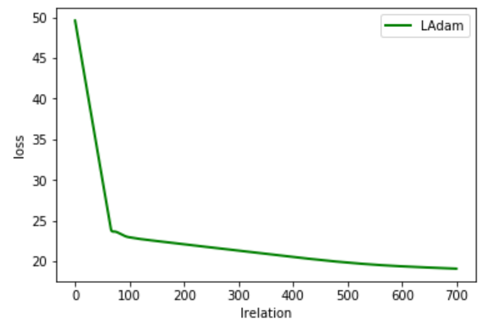
4. Model of RMSProp:



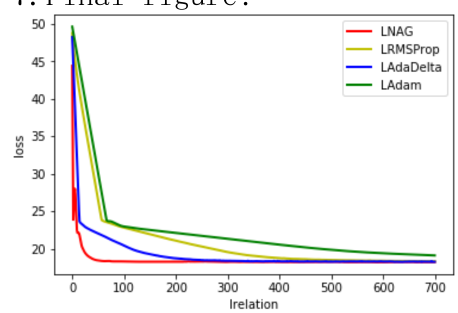
5. Model of AdamDelta:



6. Model of Adam:



7. Final figure:



IV. CONCLUSION

During this experiment , I have learned a lot , and it also exposes many problems. Although the experiment was completed on its own, the unskilled grasp of the knowledge point, although it took a lot of time, was still not complete. So I still refer to the students code.

In addition, I spent less time writing reports and I was not familiar with all English writing, so summaries were less frequently written and the whole report seemed chaotic. I will try my best to improve on the next experiment