华南理工大学

South China University of Technology

The Experiment Report of

Machine Learning

College Software College

Subject Software Engineering

Members Zheng Li

Student ID 201530612064

E-mail li416570803@qq.com

Tutor Tan

Date submitted 2017. 12 . 14

1. Topic: Comparison of SVM and logistic regression for solving classification problems

2. Time: 2017.12.15

3. Reporter:Zheng Li

4. Purposes:

1. Understand the differences between gradient descent and stochastic gradient descent

2. Understand the difference between logistic regression and linear classification

3. Further understand the principles of SVM and its practice on larger data

5. Data sets and data analysis:

LIBSVM Data a9a data sets ,including 32561/16281(testing) samples. each sample has 123 features

6. Experimental steps:

1 read the training set and testing set
2 model parameters initialize ,using all zeros initialization
3 choosing Loss function and gets its gradient
4 partly getting the gradient of the samples
5 using different optimization ways to update model parameters
6 choosing proper threshold value(阈值), to classify the type of samples, and getting Loss value Lnag, Lrmsprop, Ladadelta, Ladam of different optimization methods in testing set
7 repeating step 4-6 several times, drawing the tendency chart of Lnag, Lrmsprop, Ladadelta, Ladam

7. Code:

(Fill in the contents of 8-11 respectively for logistic regression and linear classification)

Code is on the bottom

8. The initialization method of model parameters:

All zeros

9. The selected loss function and its derivatives:

Logistic loss :

```
def  loss_fun(X,y,W,lam):
    part1 = (lam/2)*W.dot(W)
    part2 = 0
    for i in np.arange(X.shape[0]):
        part2
    +=math.log(1+math.exp(-y[i]*( W.T.dot(X[i]) )))
    part2 /= X.shape[0]
    loss = part1 + part2
    return loss
```

linear classification loss : hinge loss

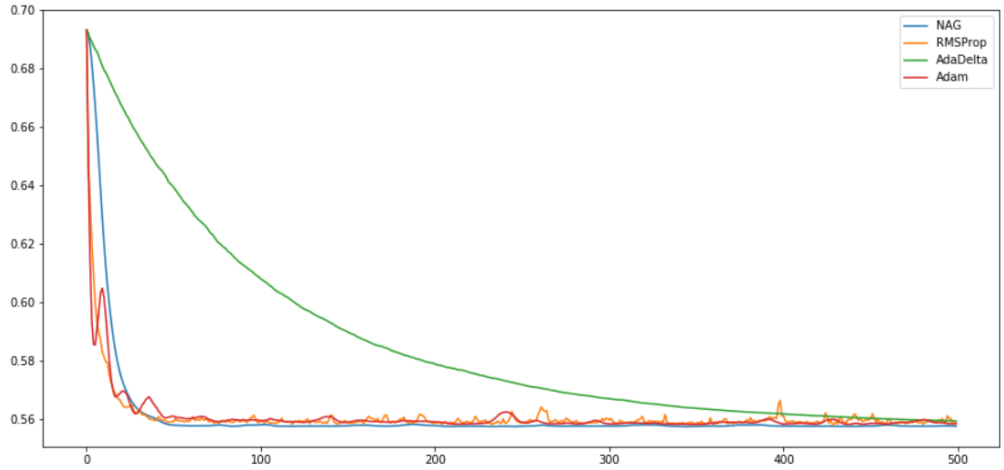
10. Experimental results and curve:(Fill in this content for various methods of gradient descent respectively)

Hyper-parameter selection:

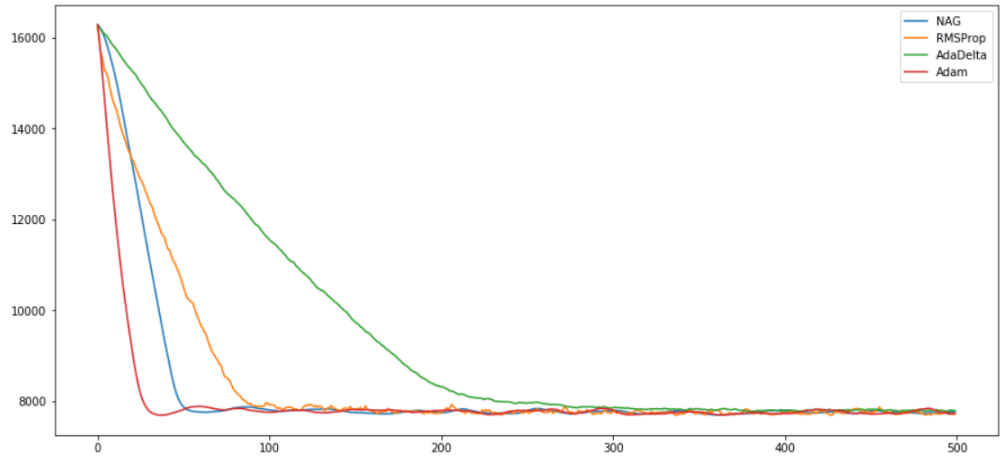
Predicted Results (Best Results):

Loss curve:

Logistic regression



Linear classification:



11. Results analysis:

Both two methods are not pretty good performing in this data set.

In linear classification, its precision rate is always the same value, about 0.77, though its loss value tends to smaller and smaller.

In logistic regression, it seems better than linear classification, which its precision rate can change with the loss value. And its precision rate can up to 0.84.

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

Similarities : As mentions above ,both two methods can classify binary classification problem, but result are not enough pretty.

Differences: logistic regression performs better than linear classification in the experiment data set.

13. Summary:

In this case, logistic regression is better than linear classification.

Code:

Linear classification:

```
X_train,y_train = load_svmlight_file('a9a.txt')
X_test,y_test = load_svmlight_file('a9a.t')
X_train = X_train.toarray()
X_test = X_test.toarray()
apnd = np.zeros(X_test.shape[0]).reshape(X_test.shape[0],1)
```

```
X_test    = np.concatenate((X_test,apnd),axis = 1)

#可以向矩阵加 1 表示 bias

apnd1 = np.ones(X_test.shape[0]).reshape(X_test.shape[0],1)
X_test = np.concatenate((X_test,apnd1),axis = 1)
apnd1 = np.ones(X_train.shape[0]).reshape(X_train.shape[0],1)
X_train = np.concatenate((X_train,apnd1),axis = 1)
#print(X_train)
#print(X_train.shape)
def formulate(W,Xi):
    g = W.T.dot(Xi)
    return g

def loss_fun(X,y,W):
    sum = 0
    for i in np.arange(len(y)):
        cost = max(0,1-y[i]*formulate(W,X[i]))
        sum+= cost
    return sum

def linear_clas_pred(X,W):
    prediction =np.zeros(X.shape[0])
    for i in np.arange(len(prediction)):
        prediction[i] = formulate(W,X[i])

    return prediction


def Gradient(X,y,W,C):
    all = range(X.shape[0])
    sample =random.sample(all,50)

    gred = np.zeros(W.shape)
    part =np.zeros(W.shape)
    for i in sample:

        if 1 - y[i]*formulate(W,X[i]) >=0:
            part += -y[i]*X[i]

    gred = W + (C/len(sample))*part
    return gred

def comparsion(X,y,W):
    predict = linear_clas_pred(X,W)
```

```
    prediction = np.zeros(predict.shape)
    for j in np.arange(len(predict)):
        if predict[j] >= 0 :
            prediction[j] = 1
        elif predict[j] < 0:
            prediction[j] = -1
    wrong = 0
    for j in np.arange(len(prediction)):
        if prediction[j] != y[j]:
            wrong += 1
    #print wrong
    return float(wrong)/len(prediction),loss_fun(X,y,W)

def NAG(X_train,y_train,X_test,y_test,eta,C,iterations):
    W = np.zeros(X_train.shape[1])
    V = np.zeros(W.shape)
    gamma = 0.9
    loss_tend = np.zeros(iterations)
    for i in np.arange(iterations):
        wr,loss = comparsion(X_train,y_train,W)
        twr,tlos = comparsion(X_test,y_test,W)
        #print('wrong rate: '+ str(wr))
        #print('loss :' + str(loss))
        #print('test wrong rate: '+ str(twr))
        #print('test loss :' + str(tlos))
        gred = Gradient(X_train,y_train,W,C)

        V = gamma*V + eta*gred
        W = W - V
        loss_tend[i] = tlos
    return loss_tend

def RMSProp(X_train,y_train,X_test,y_test,eta,C,iterations):
    W = np.zeros(X_train.shape[1])
    G = np.zeros(W.shape)
    gamma = 0.9
    el = 10**-8
    loss_tend = np.zeros(iterations)
    for i in np.arange(iterations):
        wr,loss = comparsion(X_train,y_train,W)
        twr,tlos = comparsion(X_test,y_test,W)
        #print('wrong rate: '+ str(wr))
        #print('loss :' + str(loss))
        #print('test wrong rate: '+ str(twr))
```

```
            #print('test loss :' + str(tlos))
            gred = Gradient(X_train,y_train,W,C)
            G = gamma*G + (1-gamma)*gred**2
            part =np.zeros(G.shape)
            for j in np.arange(len(part)):
                part[j] = eta/math.sqrt( G[j]+el )
            W = W - part*gred
            loss_tend[i] = tlos
        return loss_tend

def AdaDelta(X_train,y_train,X_test,y_test,C,iterations):
        W = np.zeros(X_train.shape[1])
        G = np.zeros(W.shape)
        Dt = np.zeros(W.shape)
        el = 10**-8
        gamma =0.95

        loss_tend = np.zeros(iterations)

        for i in np.arange(iterations):
            wr,loss = comparsion(X_train,y_train,W)
            twr,tlos = comparsion(X_test,y_test,W)
            #print('wrong rate: '+ str(wr))
            #print('loss :' + str(loss))
            #print('test wrong rate: '+ str(twr))
            #print('test loss :' + str(tlos))

            gred = Gradient(X_train,y_train,W,C)
            G = gamma*G + (1-gamma)*gred**2
            part = np.zeros(W.shape)
            for j in np.arange(len(part)):
                part[j] = math.sqrt(Dt[j] + el)/math.sqrt(G[j] + el)
            Dw = -1*part*gred
            W = W + Dw
            Dt = gamma*Dt + (1-gamma)*(Dw**2)
            loss_tend[i] = tlos
        return loss_tend

def Adam(X_train,y_train,X_test,y_test,eta,C,iterations):
        W = np.zeros(X_train.shape[1])
        G = np.zeros(W.shape)
        M = np.zeros(W.shape)
        B = 0.9
        el = 10**-8
```

```
    gamma = 0.999

    loss_tend = np.zeros(iterations)

    for i in np.arange(iterations):
        wr,loss = comparsion(X_train,y_train,W)
        twr,tlos = comparsion(X_test,y_test,W)
        #print('wrong rate: '+ str(wr))
        #print('loss :' + str(loss))
        #print('test wrong rate: '+ str(twr))
        #print('test loss :' + str(tlos))

        gred = Gradient(X_train,y_train,W,C)
        M = B*M + (1-B)*gred
        G = gamma*G + (1-gamma)*gred**2
        a = eta*math.exp( 1- gamma**i)/math.exp(1-B**i)
        part = np.zeros(W.shape)
        for j in np.arange(len(part)):
            part[j] = M[j]/math.sqrt(G[j] + el)
        W = W - a*part
        loss_tend[i] = tlos
    return loss_tend

def linear_classify_model(X_train,y_train,X_test,y_test,eta,C,iterations =100):
    NAG_loss = NAG(X_train,y_train,X_test,y_test,eta,C,iterations)
    RMSProp_loss = RMSProp(X_train,y_train,X_test,y_test,eta,C,iterations)
    AdaDelta_loss = AdaDelta(X_train,y_train,X_test,y_test,C,iterations)
    Adam_loss = Adam(X_train,y_train,X_test,y_test,eta,C,iterations)
    fig = plt.figure(figsize = (15,7))
    ax = fig.add_subplot(111)
    plt.plot(NAG_loss,label = 'NAG')
    plt.plot(RMSProp_loss,label = 'RMSProp')
    plt.plot(AdaDelta_loss,label = 'AdaDelta')
    plt.plot(Adam_loss,label = 'Adam')
    plt.legend(loc = 0)
    plt.show()

linear_classify_model(X_train,y_train,X_test,y_test,0.001,0.7,500)
```

logistic regression:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_svmlight_file
import math
import random
import time

X_train,y_train = load_svmlight_file('a9a.txt')
X_test,y_test = load_svmlight_file('a9a.t')
X_train = X_train.toarray()
X_test = X_test.toarray()
apnd = np.zeros(X_test.shape[0]).reshape(X_test.shape[0],1)
X_test    = np.concatenate((X_test,apnd),axis = 1)

def formulate(W,Xi):
    g = 1/(1+math.exp(-1*W.T.dot(Xi)))
    return g


def loss_fun(X,y,W,lam):
    part1 = (lam/2)*W.dot(W)
    part2 = 0
    for i in np.arange(X.shape[0]):
        part2 +=math.log(1+math.exp(-y[i]*( W.T.dot(X[i]) )))
    part2 /= X.shape[0]
    loss = part1 + part2
    return loss

def logistic_pred(train,W):
    prediction = np.zeros(train.shape[0])
    for i in np.arange(len(prediction)):
        pred = formulate(W,train[i])
        prediction[i] = pred
    return prediction




def Gradient(X,y,W,lam):
    all =range(X.shape[0])
    sample =random.sample(all,50)

    gred = np.zeros(W.shape)
    part1 = lam*W
```

```
    part2 = np.zeros(W.shape)
    for i in sample:
        part2 += y[i]*X[i]/(1+math.exp(y[i]*W.T.dot(X[i])))
    part2 /=len(sample)
    gred =part1 - part2
    return gred

def comparsion(X,y,W,lam):
    predict= logistic_pred(X,W)
    prediction = np.zeros(predict.shape)
    #print(len(predict))
    for j in np.arange(len(predict)):
        if predict[j] >= 0.5:
            prediction[j] = 1
        else:
            prediction[j] = -1
    wrong = 0
    for j in np.arange(len(prediction)):
        if prediction[j] != y[j]:
            wrong += 1
    #print('wrong rate: '+ str(float(wrong)/len(prediction)))
    #print('loss :' + str(loss_fun(X,y,W,lam)))

    return float(wrong)/len(prediction),loss_fun(X,y,W,lam)


def NAG(X_train,y_train,X_test,y_test,eta,lam,iterations):
    W = np.zeros(X_train.shape[1])
    V = np.zeros(W.shape)
    gamma = 0.9
    loss_tend = np.zeros(iterations)
    for i in np.arange(iterations):
        wr,loss = comparsion(X_train,y_train,W,lam)
        twr,tlos = comparsion(X_test,y_test,W,lam)
        #print('wrong rate: '+ str(wr))
        #print('loss :' + str(loss))
        #print('test wrong rate: '+ str(twr))
        #print('test loss :' + str(tlos))
        gred = Gradient(X_train,y_train,W,lam)

        V = gamma*V + eta*gred
        W = W - V
        loss_tend[i] = loss
    return loss_tend
```

```
def RMSProp(X_train,y_train,X_test,y_test,eta,lam,iterations):
    W = np.zeros(X_train.shape[1])
    G = np.zeros(W.shape)
    gamma = 0.9
    el = 10**-8
    loss_tend = np.zeros(iterations)
    for i in np.arange(iterations):
        wr,loss = comparsion(X_train,y_train,W,lam)
        twr,tlos = comparsion(X_test,y_test,W,lam)
        #print('wrong rate: '+ str(wr))
        #print('loss :' + str(loss))
        #print('test wrong rate: '+ str(twr))
        #print('test loss :' + str(tlos))
        gred = Gradient(X_train,y_train,W,lam)
        G = gamma*G + (1-gamma)*gred**2
        part =np.zeros(G.shape)
        for j in np.arange(len(part)):
            part[j] = eta/math.sqrt( G[j]+el )
        W = W - part*gred
        loss_tend[i] = loss
    return loss_tend

def AdaDelta(X_train,y_train,X_test,y_test,lam,iterations):
    W = np.zeros(X_train.shape[1])
    G = np.zeros(W.shape)
    Dt = np.zeros(W.shape)
    el = 10**-8
    gamma =0.95

    loss_tend = np.zeros(iterations)

    for i in np.arange(iterations):
        wr,loss = comparsion(X_train,y_train,W,lam)
        twr,tlos = comparsion(X_test,y_test,W,lam)
        #print('wrong rate: '+ str(wr))
        #print('loss :' + str(loss))
        #print('test wrong rate: '+ str(twr))
        #print('test loss :' + str(tlos))

        gred = Gradient(X_train,y_train,W,lam)
        G = gamma*G + (1-gamma)*gred**2
        part = np.zeros(W.shape)
        for j in np.arange(len(part)):
```

```
                part[j] = math.sqrt(Dt[j] + el)/math.sqrt(G[j] + el)
            Dw = -1*part*gred
            W = W + Dw
            Dt = gamma*Dt + (1-gamma)*(Dw**2)
            loss_tend[i] = loss
        return loss_tend


def Adam(X_train,y_train,X_test,y_test,eta,lam,iterations):
        W = np.zeros(X_train.shape[1])
        G = np.zeros(W.shape)
        M = np.zeros(W.shape)
        B = 0.9
        el = 10**-8
        gamma = 0.999

        loss_tend = np.zeros(iterations)

        for i in np.arange(iterations):
            wr,loss = comparsion(X_train,y_train,W,lam)
            twr,tlos = comparsion(X_test,y_test,W,lam)
            #print('wrong rate: '+ str(wr))
            #print('loss :' + str(loss))
            #print('test wrong rate: '+ str(twr))
            #print('test loss :' + str(tlos))

            gred = Gradient(X_train,y_train,W,lam)
            M = B*M + (1-B)*gred
            G = gamma*G + (1-gamma)*gred**2
            a = eta*math.exp( 1- gamma**i)/math.exp(1-B**i)
            part = np.zeros(W.shape)
            for j in np.arange(len(part)):
                part[j] = M[j]/math.sqrt(G[j] + el)
            W = W - a*part
            loss_tend[i] = loss
        return loss_tend



def logistic_model(X_train,y_train,X_test,y_test,eta,lam,iterations =100):
        NAG_loss = NAG(X_train,y_train,X_test,y_test,eta,lam,iterations)
        RMSProp_loss = RMSProp(X_train,y_train,X_test,y_test,eta,lam,iterations)
        AdaDelta_loss = AdaDelta(X_train,y_train,X_test,y_test,lam,iterations)
        Adam_loss = Adam(X_train,y_train,X_test,y_test,eta,lam,iterations)
```

```
        fig = plt.figure(figsize = (15,7))
        ax = fig.add_subplot(111)
        plt.plot(NAG_loss,label = 'NAG')
        plt.plot(RMSProp_loss,label = 'RMSProp')
        plt.plot(AdaDelta_loss,label = 'AdaDelta')
        plt.plot(Adam_loss,label = 'Adam')
        plt.legend(loc = 0)
        plt.show()
logistic_model(X_train,y_train,X_test,y_test,0.005,0.5,500)
```