华南理工大学

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Ruolin liang

Supervisor:
Qingyao Wu

Student ID：201721046029

Grade:
Graduate

December 9, 2017

Logistic regression, linear classification and stochastic gradient descent

Abstract—Gradient descent optimization algorithms, while increasingly popular, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by. TIn the course of this overview, we look at different variants of gradient descent, introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent. Logistic regression mainly solves the problem of dichotomy or multi-classification, which is to use the regression idea to set up a logistic function, which makes the output 0 or 1 classification strategy.

I. INTRODUCTION

Logistic regression is also called logistic regression analysis, which is a generalized linear regression analysis model, which is often used for data mining, automatic disease diagnosis and economic forecasting. A classification of logistic regression is very widely applied machine learning algorithms, it will be the data fitting to a logit function (or called logistic function), so that they can complete to estimate the probability of events. Ogistic regression is a generalized linear model, so it has many similarities with multiple linear regression analysis. Their model forms are basically the same, have w 'x + b, w and b is for parameters, the difference between the dependent variable, multiple linear regression directly w' x + b as dependent variable, namely 'x + y = w b, while by logistic regression function will L w' x + b corresponding to a hidden states p, p = L (w 'x + b), and then according to the size of 1 - p p and determine the value of the dependent variable. If L is a logistic function, then logistic regression, if L is a polynomial function is polynomial regression. The dependent variables of logistic regression can be dichotomies or multivariate, but the second classification is more commonly used and easier to interpret, and multiple classes can be processed using the softmax method. The most commonly used is the logistic regression of dichotomiesThere are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function. Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update. Batch gradient descent. Stochastic gradient descent. Mini-batch gradient descent.

Nesterov accelerated gradient (NAG) [14] is a way to give our momentum term this kind of prescience.

We know that we will use our momentum term γv_{t-1} to move the parameters θ. Computing $\theta - \gamma v_{t-1}$

thus gives us an approximation of the next position of the parameters (the gradient is missing for the

full update), a rough idea where our parameters are going to be. We can now effectively look ahead

by calculating the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of our parameters:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

. Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data. Dean et al. [6] have found that Adagrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google, which – among other things – learned to recognize cats in Youtube videos10. Moreover,Pennington et al. [16] used Adagrad to train GloVe word embeddings, as infrequent words require much larger updates than frequent ones.

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.

RMSprop is an unpublished, adaptive learning rate method proposed RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates. RMSprop in fact is identical. Adaptive Moment Estimation (Adam) [10] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients vt like Adadelta and RMSprop.

II. METHODS AND THEORY

Logistic Regression and Stochastic Gradient Descent

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

III. EXPERIMENT

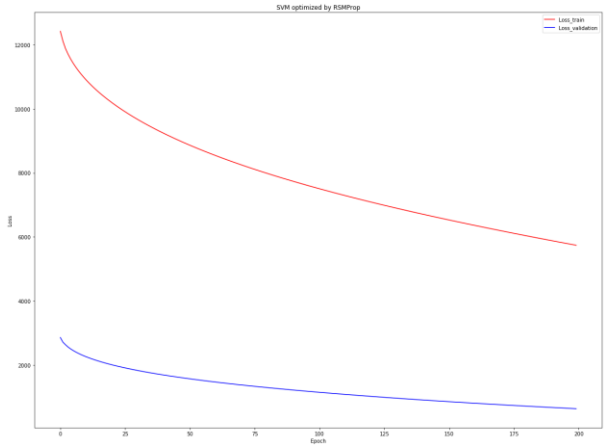
Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

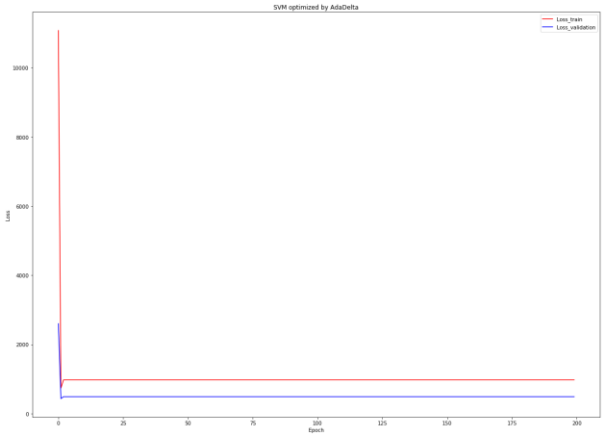
Environment for experiment is python3, at least including following python package: sklearn，numpy，jupyter，matplotlib

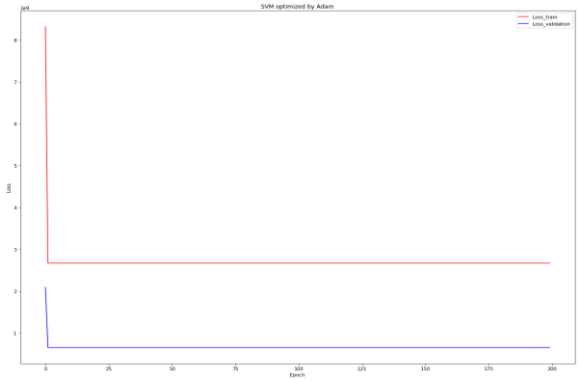
The linear classification’code is here:

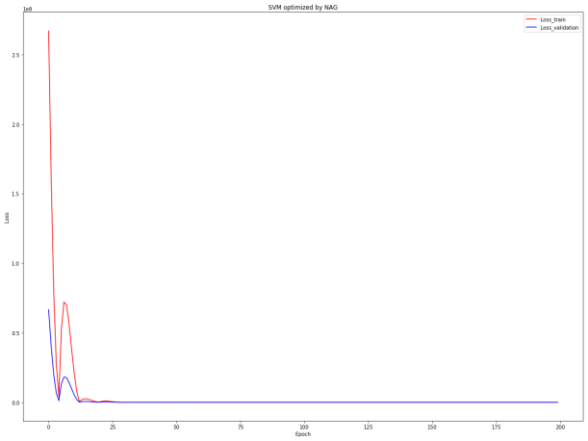
```
from sklearn.datasets import load_svmlight_file
import numpy as np
import matplotlib.pyplot as plt

def read_date(fileName):
    data=load_svmlight_file(fileName)
    return data[0],data[1]

path_train="/home/lrl/Downloads/a9a"
path_test="/home/lrl/Downloads/a9a.t"
x_train,y_train=load_svmlight_file(path_train)
x_train=x_train.toarray()
x_test,y_test=load_svmlight_file(path_test)
x_test=x_test.toarray()

column=np.zeros((x_test.shape[0]))
x_test=np.column_stack((x_test,column))

column_train=np.ones((x_train.shape[0]))
column_test=np.ones((x_test.shape[0]))
x_train=np.column_stack((x_train,column_train))
m,n=np.shape(x_train)
y_train=y_train.reshape((m,1))
x_test=np.column_stack((x_test,column_test))
m_train,n_train=np.shape(x_test)
y_test=y_test.reshape((m_train,1))
print x_train.shape,y_train.shape
print x_test.shape,y_test.shape

def SVMgradient(x,y,w,eta,c):

    h=1-y*np.dot(x,w)
    #print h.shape
    tmp=np.where(h>0,y,0)

    w=w-eta*(w-c*np.dot(x.T,tmp))
    #print w.shape
    y_predict=np.where(np.dot(x,w)>0,1,-1)

loss=np.sum(w*w)+c*np.sum(np.maximum(1-y*np.dot(x,w),
0))

    return loss,w


def draw_plot(Loss_train, Loss_test, name):
    plt.figure(figsize=(20,15))
    plt.plot(Loss_train,color="r",label="Loss_train")
    plt.plot(Loss_test,color="b",label="Loss_validation")
    plt.legend()
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.title("Logistic regression optimized by "+ name)
    plt.show()

def NAG(x,y):

    gamma=0.7
    m,n=np.shape(x)
    w=np.random.random((n,1))
    v=np.array(np.zeros(w.shape))
    epoch=200
    costs=[]
    for epoch in range(epoch):

        eta=0.23
        Loss,grad=SVMgradient(x,y,w-gamma*v,eta,gamma)
        v=gamma*v+eta*grad
        w=w-v
        costs.append(Loss)
    return costs,w
Loss_train1,grad=NAG(x_train,y_train)
Loss_test1,grad=NAG(x_test,y_test)
draw=draw_plot(Loss_train1, Loss_test1, 'NAG')

def Adam(x,y):
    costs=[]
    c=1.2
    m,n=np.shape(x)
    w=np.random.random((n,1))
    v=np.array(np.zeros(w.shape))
    beta1=0.8
```

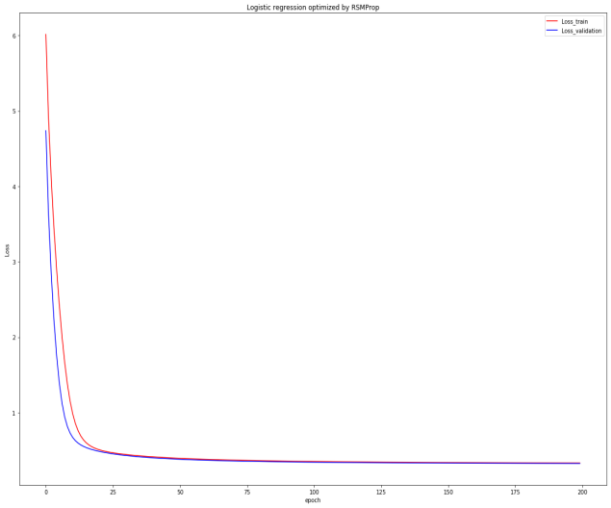
```
    beta2=0.989
    beta1_exp=1.0
    beta2_exp=1.0
    eta=1
    epsilon=1e-8
    epoch=200
    for epoch in range(epoch):
        cost,grad=SVMgradient(x,y,w,eta,c)
        m=beta1*m+(1.0-beta1)*grad
        v=beta2*v+(1.0-beta2)*np.square(grad)
        beta1_exp*=beta1
        beta2_exp*=beta2

w=w-eta*(m/(1.0-beta1_exp))/(np.sqrt(v/(1.0-beta2_exp))+epsilon)
        costs.append(cost)
    return costs,w
Loss_train1,grad=Adam(x_train,y_train)
Loss_test1,grad=Adam(x_test,y_test)
draw=draw_plot(Loss_train1, Loss_test1, 'Adam')


def AdaDelta(x,y):
    c=0.03
    gamma=1.0
    epsilon=1e-8
    m,n=np.shape(x)
    w=np.random.random((n,1))
    grad_expect=np.array(np.zeros(w.shape))
    delta_expect=np.array(np.zeros(w.shape))
    epoch=200
    costs=[]
    eta=0.01
    for epoch in range(epoch):
        cost,grad=SVMgradient(x,y,w,eta,c)

grad_expect=gamma*grad_expect+(1.0-gamma)*np.square(grad)

delta=-np.multiply(np.sqrt(delta_expect+epsilon)/np.sqrt(grad_expect+epsilon),grad)
        w=w+delta

delta_expect=gamma*delta_expect+(1.0-gamma)*np.square(delta)
        costs.append(cost)
    return costs,w
Loss_train1,grad=AdaDelta(x_train,y_train)
Loss_test1,grad=AdaDelta(x_test,y_test)
draw=draw_plot(Loss_train1, Loss_test1, 'AdaDelta')

def RSMProp(x,y):
    epoch=200
    m,n=np.shape(x)
    w=np.random.random((n,1))
    gamma=0.99
    costs=[]
    eta=0.01
    epsilon=1e-8
    grad_expect=np.array(np.zeros(w.shape))

    for epoch in range(epoch):
        cost,grad=SVMgradient(x,y,w,eta,c)

grad_expect=gamma*grad_expect+(1.0-gamma)*np.square(grad)
        w=w-eta*grad/np.sqrt(grad_expect+epsilon)

        costs.append(cost)
    return costs,w
Loss_train1,grad=RSMProp(x_train,y_train)
Loss_test1,grad=RSMProp(x_test,y_test)
draw=draw_plot(Loss_train1, Loss_test1, 'RSMProp')
```











Logistic regression is not available for numerical reasons

IV. CONCLUSION

Logistic regression is also called logistic regression analysis, which is a generalized linear regression analysis model, which is often used for data mining, automatic disease diagnosis and economic forecasting. A classification of logistic regression is very widely applied machine learning algorithms, it will be the data fitting to a logit function (or called logistic function), so that they can complete to estimate the probability of events. the course of this overview, we look at different variants of gradient descent, introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent. By experiment,we gain that compare and understand the difference between gradient descent and stochastic gradient descent,compare and understand the differences and relationships between Logistic regression and linear classificationand further understand the principles of SVM and practice on larger data. Change the effect of the experiment by adjusting the parameters