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:
Zanjie Fang

Supervisor:
Mingkui Tan

Student ID：
201530611456

Grade:
Undergraduate

December 14, 2017

Logistic Regression, Linear Classification and Gradient Descent

Abstract — This experiment aims to compare and understand the difference between various stochastic gradient descent methods for solving classification problems. In addition, compare and understand the differences and relationships between Logistic regression and linear classification.

I. Introduction

In this experiment, I compare SVM and Logistic regression for solving classification problems, and update the model parameter using four different optimized methods(NAG，RMSProp，AdaDelta and Adam). The stochastic gradient descent methods adopt minibatch stochastic gradient descent and the initialization of the BATCH size is 100. In theory, using four stochastic gradient descent methods to update the model parameter to optimize the classification problems may result in the same resultsm, but their different speeds on convergence. And luckily, the result of this experiment meet this assumption. The adam method performs better than other methods.

II. Methods and Theory

A. The chosen methods

For both Logistic regression and linear classification, I use the following four stochastic gradient descent methods to update the model parameter. More details are as following:

1. NAG

The core idea of NAG (Nesterov accelerated gradient) is to use Momentum to predict the next gradient, rather than using the current Θ.

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

2. RMAProp

RMSProp is to solve the problem of AdaGrad's learning rate tends to 0.

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

3. AdaDelta

AdaDelta uses $\sqrt{\Delta_{t-1} + \epsilon}$ to estimate the learning rate.

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

4. Adam

Adam is the adaptive estimates of lower-order moments adaptively adjusts the 1st order (mean) and 2nd order moments (variance).

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

B. The related loss functions

1. The loss function of the Logistic regression:

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i \cdot \mathbf{w}^\top \mathbf{x}_i}) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

2. The loss function of the linear classification:

$$\frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

C. The derivation process

1. The gradient of the loss function of the Logistic regression:

$$\lambda \mathbf{w} + \frac{1}{n}\sum_{i=1}^{n} \frac{y_i \mathbf{x}_i}{1 + e^{y_i \cdot \mathbf{w}^\top \mathbf{x}_i}}$$

2. The gradient of the loss function of the Linear classification:

$$\frac{\partial f(\mathbf{w}, b)}{\mathbf{w}} = \mathbf{w} + C\sum_{i=1}^{N} g_{\mathbf{w}}(\mathbf{x}_i)$$
$$\frac{\partial f(\mathbf{w}, b)}{b} = C\sum_{i=1}^{N} g_b(\mathbf{x}_i)$$

III. Experiment

A. Dataset

This experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. The label of each data is 1 or -1.

B. Implementation

Load datasets:

```
import os
os.chdir('D://dataset')
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file

# 加载训练集
X_train, y_train = load_svmlight_file('a9a_train.txt')
X_train = X_train.toarray()
X_train = np.c_[X_train,np.ones((X_train.shape[0], 1))]# bias
y_train = y_train.reshape(y_train.shape[0], 1)

# 加载测试集
X_test, y_test = load_svmlight_file('a9a_test.txt')
X_test = X_test.toarray()
X_test = np.c_[X_test,np.zeros((X_test.shape[0], 1))]# 最后一维缺失，补上
X_test = np.c_[X_test,np.ones((X_test.shape[0], 1))]# bias
y_test = y_test.reshape(y_test.shape[0], 1)
```

1. The codes of the loss function and the gradient of the Logistic regression:

```
# 计算梯度
def calDirection(X_train, y_train,W):
    temp = np.zeros((X_train.shape[1], 1))
    randIndex = int(np.random.uniform(0,len(X_train) - BATCH))
    for index in range(BATCH):
        Yi = y_train[randIndex+index][0]
        Xi = X_train[randIndex+index].reshape(1, X_train.shape[1])
        temp += Yi/(1 + np.exp(Yi*np.dot(W.T, Xi.T)[0][0]))*Xi.T
    direction = LAMBDA*W -  temp / BATCH
    return direction

# 计算loss
def calLoss(W, X_t, y_t):
    temp = 0
    for index in range(len(X_t)):
        Yi = y_t[index][0]
        Xi = X_t[index].reshape(1, X_t.shape[1])
        temp += np.log(1 + np.exp(- Yi*np.dot(W.T, Xi.T)[0][0]))
    Loss = LAMBDA/2 * np.dot(W.T, W) + temp/X_t.shape[0]
    return Loss
```

2. The codes of the loss function and the gradient of the Linear classification:

```
# 计算HingeLoss，返回hingeLoss
def calHingeLoss(Yi, Xi, W):
    result = 1 - (Yi*np.dot(W.T, Xi.T))[0][0]
    if result > 0:
        hingeLoss = result
    else:
        hingeLoss = 0
    return hingeLoss

# 计算梯度,返回梯度
def calDirection(X_train, y_train,W):
    temp = np.zeros((X_train.shape[1], 1))
    randIndex = int(np.random.uniform(0,len(X_train) - BATCH))
    for index in range(BATCH):
        Yi = y_train[randIndex+index][0]
        Xi = X_train[randIndex+index].reshape(1, X_train.shape[1])
        hingeLoss = calHingeLoss(Yi, Xi, W)
        if hingeLoss > 0:
            temp = temp + Yi * Xi.T
        else:
            temp = temp+np.zeros((X_train.shape[1], 1))
    direction = W - c * temp/BATCH
    return direction

# 计算loss，返回Loss
def calLoss(W, X_t, y_t):
    temp = 0
    for index in range(len(X_t)):
        Yi = y_t[index][0]
        Xi = X_t[index].reshape(1, X_t.shape[1])
        temp += calHingeLoss(Yi, Xi, W)
    temp = c * temp
    Loss = 1/2 * np.dot(W.T, W) + temp/X_t.shape[0]
    return Loss
```

3. The codes of the implementation of the four stochastic gradient descent methods:

(1). NAG

```
# 用NAG更新参数
def updateNAG(W, V, direction):
    learning_rate = 0.01
    GARMA = 0.95
    direction = calDirection(X_train, y_train, W - GARMA*V)
    #direction = direction - GARMA*V
    V = GARMA * V + learning_rate *direction
    W = W - V# 更新参数
    return W, V, direction

def NAG_loss():
    # 记录验证集的loss随迭代次数的值
    lossValidation = []
    #lossValidationClassification = []
    direction = np.zeros((X_train.shape[1], 1))
    W = np.zeros((X_train.shape[1], 1))
    V = np.zeros((X_train.shape[1], 1))

    for i in range(epoch):
        W, V, direction = updateNAG(W, V, direction)
        # 计算测试集的loss
        Ltest = calLoss(W, X_test,y_test)
        lossValidation.append(Ltest[0][0])# 将测试集的loss加入列表
    return lossValidation
```

(2). RMSProp

```
# 用RMSProp更新参数
def updateRMSProp(W, G):
    GARMA = 0.95
    EPSEN = 1e-8
    learning_rate = 0.01
    direction = calDirection(X_train, y_train, W)
    G = GARMA * G + (1 - GARMA)*np.dot(direction.T, direction)
    temp = (learning_rate / np.sqrt(G + EPSEN ))*direction
    W = W - temp# 更新参数
    return W, G

def RMSProp_loss():
    # 记录验证集的loss随迭代次数的值
    lossValidation = []
    #lossValidationClassification = []
    W = np.zeros((X_train.shape[1], 1))
    G = np.zeros((X_train.shape[1], 1))

    for i in range(epoch):
        W, G = updateRMSProp(W, G)
        # 计算测试集的loss
        Ltest = calLoss(W, X_test,y_test)
        lossValidation.append(Ltest[0][0])# 将测试集的loss加入列表
    return lossValidation
```

(3). AdaDelta

```
# 用AdaDelta更新参数
def updateAdaDelta(W, G, deltaT):
    deltaW = np.zeros((X_train.shape[1], 1))
    GARMA = 0.9999
    EPSEN = 1e-8
    direction = calDirection(X_train, y_train, W)
    G = GARMA * G + (1 - GARMA)*np.dot(direction.T, direction)
    deltaW = - (np.sqrt(deltaT + EPSEN ) / np.sqrt(G + EPSEN ))*direction
    W = W + deltaW# 更新参数
    deltaT = GARMA * deltaT + (1 - GARMA)*np.dot(deltaW.T, deltaW)
    return W, G, deltaT
def AdaDelta_loss():
    # 记录验证集的loss随迭代次数的值
    lossValidation = []
    #lossValidationClassification = []
    W = np.zeros((X_train.shape[1], 1))
    G = np.zeros((X_train.shape[1], 1))
    deltaT = np.zeros((X_train.shape[1], 1))

    for i in range(epoch):
        W, G, deltaT = updateAdaDelta(W, G, deltaT)
        # 计算测试集的loss
        Ltest = calLoss(W, X_test,y_test)
        lossValidation.append(Ltest[0][0])# 将测试集的loss加入列表
    return lossValidation
```

(4). Adam

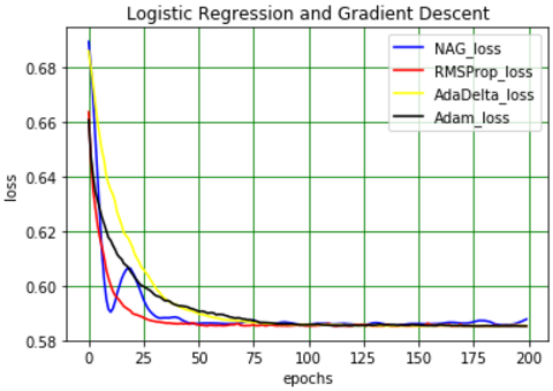
```
# 用Adam更新参数
def updateAdam(W, G, M, i):
    ALFA = 0
    GARMA = 0.999
    EPSEN = 1e-8
    BELTA = 0.9
    learning_rate = 0.05

    direction = calDirection(X_train, y_train, W)
    M = BELTA*M + (1 - BELTA)*direction
    G = GARMA * G + (1 - GARMA)*np.dot(direction.T, direction)
    ALFA = learning_rate*np.sqrt(1 - GARMA**(i+1) )/(1 - BELTA**(i+1))
    W = W - ALFA*M/np.sqrt(G + EPSEN)
    return W, G, M

def Adam_loss():
    W = np.zeros((X_train.shape[1], 1))
    G = np.zeros((X_train.shape[1], 1))
    M = np.zeros((X_train.shape[1], 1))
    # 记录验证集的loss随迭代次数的值
    lossValidation = []
    #lossValidationClassification = []
    for i in range(epoch):
        W, G, deltaT = updateAdam(W, G, M, i)
        # 计算测试集的loss
        Ltest = calLoss(W, X_test,y_test)
        lossValidation.append(Ltest[0][0])# 将测试集的loss加入列表
    #print(lossValidation)
    return lossValidation
```

4. The result of four stochastic gradient descent methods:
The unified initialization parameters:
epoch = 200, BATCH = 100, LAMBDA = 1.

(1) On the Logistic regression:



The parameters of four stochastic gradient descent methods:

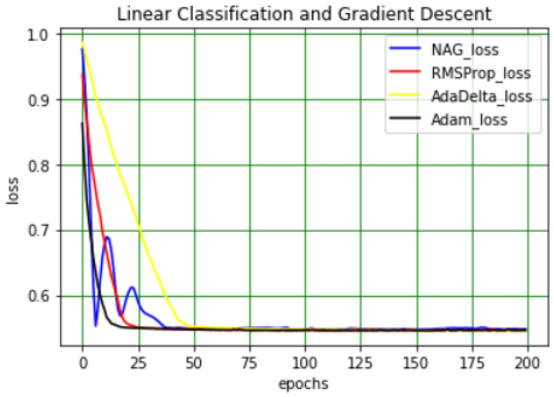
a. NAG:
learning_rate = 0.01, GARMA = 0.95,
W = np.zeros((X_train.shape[1], 1)),
V = np.zeros((X_train.shape[1], 1)).

b. RMSProp:
GARMA = 0.95, EPSEN = 1e-8, learning_rate = 0.01
W = np.zeros((X_train.shape[1], 1)),
G = np.zeros((X_train.shape[1], 1)).

c. AdaDelta:
GARMA = 0.9999, EPSEN = 1e-8,
deltaW = np.zeros((X_train.shape[1], 1)),
W = np.zeros((X_train.shape[1], 1)) ,
G = np.zeros((X_train.shape[1], 1)),
deltaT = np.zeros((X_train.shape[1], 1)).

d. Adam:
ALFA = 0, GARMA = 0.999, EPSEN = 1e-8
BELTA = 0.9, learning_rate = 0.05,
W = np.zeros((X_train.shape[1], 1)) ,
G = np.zeros((X_train.shape[1], 1)),
M = np.zeros((X_train.shape[1], 1)).

(2) On the Linear classification:



The parameters of four stochastic gradient descent methods:

a. NAG:
learning_rate = 0.01, GARMA = 0.95,
W = np.zeros((X_train.shape[1], 1)),
V = np.zeros((X_train.shape[1], 1)).

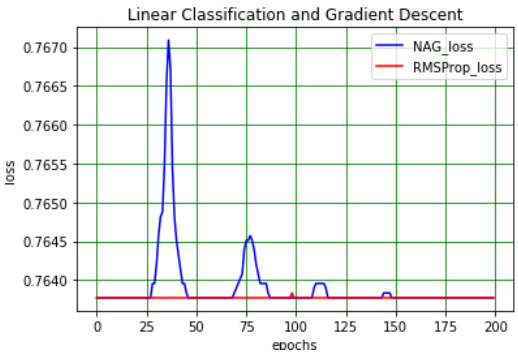
b. RMSProp:
GARMA = 0.95, EPSEN = 1e-8, learning_rate = 0.01
W = np.zeros((X_train.shape[1], 1)),
G = np.zeros((X_train.shape[1], 1)).

c. AdaDelta:
GARMA = 0.9999, EPSEN = 1e-8,
deltaW = np.zeros((X_train.shape[1], 1)),
W = np.zeros((X_train.shape[1], 1)) ,
G = np.zeros((X_train.shape[1], 1)),
deltaT = np.zeros((X_train.shape[1], 1)).

d. Adam:
ALFA = 0, GARMA = 0.999, EPSEN = 1e-8
BELTA = 0.9, learning_rate = 0.1,
W = np.zeros((X_train.shape[1], 1)) ,
G = np.zeros((X_train.shape[1], 1)),
M = np.zeros((X_train.shape[1], 1)).

IV. CONCLUSION

Through this experiment, I have a deeper understanding of SVM and Logistic regression for solving classification problems. Especially when updating the parameters, once the learning rate is too low, then the loss can not converge to a stable value. Furthermore, I do not know why the rightrate of the classification performs like this:



The code is as following:

```
def calRightRate(W, X_t, y_t):
    right = 0
    for index in range(X_t.shape[0]):
        Ypredict= np.dot(X_t[index], W)
        if Ypredict > 0:
            Ypredict = 1
        else:
            Ypredict = -1
        if Ypredict == y_t[index][0]:
            right += 1
    rightRate = right / float(y_t.shape[0])
    return rightRate
```

I will try to solve this problem next.