The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
CaiZhe

Supervisor:
Qingyao Wu

Student ID:
201721045701

Grade:
Graduate

December 13, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

[1] Abstract—

The principle of Logistic Regression, Linear Classification and Stochastic Gradient Descent is used in this experiment to train the model.

I. INTRODUCTION

The motivation of this Experiment is to Compare and understand the difference between gradient descent and stochastic gradient descent. Compare and understand the differences and relationships between Logistic regression and linear classification. Further understand the principles of SVM and practice on larger data.

II. METHODS AND THEORY

In statistics, logistic regression is a regression model where the dependent variable (DV) is categorical. Tthe output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick.

In the field of machine learning, the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics.

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration.

III. EXPERIMENT

A.DataSet

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

B.Implementation

1.Logistic Regression and Stochastic Gradient Descent

1)Load the training set and validation set.Initalize logistic regression model parameterswith zero.

2)Calculate gradient toward loss function from partial samples

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

NAG: $\omega_t = \omega_{t-1} - v_t$

RMSProp:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\omega_t = \omega_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

Adadelta:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\omega_t = \omega_{t-1} - \frac{\sqrt{E[\omega^2]_{t-1} + \varepsilon}}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

$$E[\omega^2]_t = \gamma E[\omega^2]_{t-1} + (1-\gamma) \omega_t^2$$

Adam:

$$m_t - \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t - \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{v_t + \varepsilon}} m_t$$

3)Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).

NAG:

```
def NAG(X_data, y_data, w, learn_rate, iter_times, X_test, y_test, batch_num):
    mome = 0.5
    momeIncrease = 20
    v = matrix(zeros(w.shape))
    Loss = []
    for i in range(iter_times):
        if i == momeIncrease:
            mome = 0.9
        X_randdata, y_randdata = randChoice(X_data,y_data,batch_num)
        f = multiply(y_randdata, X_randdata * (w - mome * v))
            grad = - X_randdata.T*(multiply(f < 1, y_randdata)) / X_randdata.shape[0]
        v = mome * v + learn_rate * grad
        w = w - v
        f2 = multiply(y_test, X_test * w)
        z = matrix(np.c_[array(1-f2), zeros(f2.shape)])
        L = (z.max(1).sum())/X_test.shape[0]
        Loss.append(L)
    return w, Loss
```

RMSProp:

```
def RMSProp(X_data, y_data, w, learn_rate, iter_times, X_test, y_test, batch_num):
    gamma = 0.9
    E = matrix(zeros(w.shape))
    epsilon = 1e-05
    Loss = []
    for i in range(iter_times):
        X_randdata, y_randdata = randChoice(X_data,y_data,batch_num)
        f = multiply(y_randdata, X_randdata * w)
            grad = - X_randdata.T*(multiply(f < 1, y_randdata)) / X_randdata.shape[0]
        E = gamma * E + (1 - gamma) * multiply(grad, grad)
        w = w - multiply(learn_rate / sqrt(E + epsilon), grad)
```

```
        f2 = multiply(y_test, X_test * w)
        z = matrix(np.c_[array(1-f2), zeros(f2.shape)])
        L = (z.max(1).sum())/X_test.shape[0]
        Loss.append(L)
    return w, Loss
```

Adadelta:

```
def Adadelta(X_data, y_data, w, learn_rate, iter_times, X_test, y_test, batch_num):
    gamma = 0.95
    E = matrix(zeros(w.shape))
    Ew = matrix(zeros(w.shape))
    epsilon = 1e-06
    Loss = []
    w = zeros(w.shape)
    for i in range(iter_times):
        X_randdata, y_randdata = randChoice(X_data,y_data,batch_num)
        grad = (X_randdata.T * (sigmoid(X_randdata * w) - y_randdata)) / X_randdata.shape[0]
        E = gamma * E + (1 - gamma) * multiply(grad, grad)
        delta_w = - multiply((sqrt(Ew + epsilon) / sqrt(E + epsilon)) ,grad)
        Ew = gamma * Ew + (1 - gamma) * multiply(delta_w, delta_w)
        w = w + delta_w
        h = sigmoid(X_test * w)
        f2 = ((sigmoid(X_test * w) >= 0.5) == y_test).sum()
        L = ((multiply(y_test, log(h)) + multiply(1 - y_test, log(1 - h))) / (-X_test.shape[0])).sum()
        Loss.append(L)
    return w, Loss
```

```
def Adam(X_data, y_data, w, learn_rate, iter_times, X_test, y_test, batch_num):
        beta1 = 0.9
        beta2 = 0.999
        m = matrix(zeros(w.shape))
        v = matrix(zeros(w.shape))
        epsilon = 1e-08
        Loss = []
        for i in range(iter_times):
            X_randdata, y_randdata = randChoice(X_data,y_data,batch_num)
            grad = (X_randdata.T * (sigmoid(X_randdata * w) - y_randdata)) / X_randdata.shape[0]
            m = beta1 * m + (1 - beta1) * grad
            v = beta2 * v + (1 - beta2) * multiply(grad, grad)
    #       m = m / (1 - beta1 ** (i + 1))
    #       v = v / (1 - beta2 ** (i + 1)) w = w - multiply(learn_rate / (sqrt(v) + epsilon), m)
            h = sigmoid(X_test * w)
            f2 = ((sigmoid(X_test * w) >= 0.5) == y_test).sum()
            L = ((multiply(y_test, log(h)) + multiply(1 - y_test, log(1 - h))) / (-X_test.shape[0])).sum()
            Loss.append(L)
        return w, Los
```

4)drawing graph of LNAG,LRMSProp,Ladadelta,LAdam and with the number of iterations.

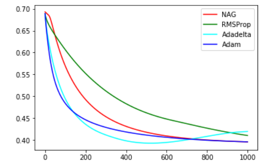
2.TextLinear Classification and Stochastic Gradient Descent

1)Load the training set and validation set.Initalize logistic regression model parameterswith zero.

2)Calculate gradient toward loss function from partial samples

```
def logistic(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=30)
    y_train[where(y_train[:] == -1)] = 0
    y_test[where(y_test[:] == -1)] = 0
    L_NAG, L_RMSProp, L_Adadelta, L_Adam = GradientDescent(X_train,y_train,0.001,1000, X_test,y_test,256)
    plt.figure('logistic regression')
    plt.plot(arange(1000),L_NAG, 'r', label='NAG')
    plt.plot(arange(1000),L_RMSProp, 'g', label='RMSProp')
    plt.plot(arange(1000),L_Adadelta, 'cyan', label='Adadelta')
    plt.plot(arange(1000),L_Adam, 'b', label='Adam')
    plt.legend(loc='best')
    plt.show()

X, y = load_svmlight_file("logistic_data.txt")
X = X.todense()
X = np.array(X)
logistic(X, y)
```



3)Update model parameters using different optimized methods(NAG， RMSProp， AdaDelta and Adam).

NAG:

```
def NAG(X_data, y_data, w, learn_rate, iter_times, X_test, y_test, batch_num):
    mome = 0.5
    momeIncrease = 20
    v = matrix(zeros(w.shape))
    Loss = []
    for i in range(iter_times):
        if i == momeIncrease:
            mome = 0.9
        X_randdata, y_randdata = randChoice(X_data,y_data,batch_num)
        f = multiply(y_randdata, X_randdata * (w - mome * v))
        grad = - X_randdata.T*(multiply(f < 1, y_randdata)) / X_randdata.shape[0]
v = mome * v + learn_rate * grad
        w = w - v
        f2 = multiply(y_test, X_test * w)
        z = matrix(np.c_[array(1-f2), zeros(f2.shape)])
        L = (z.max(1).sum())/X_test.shape[0]
        Loss.append(L)
    return w, Loss
```

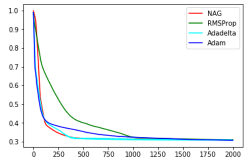
RMSProp:

```
def RMSProp(X_data, y_data, w, learn_rate, iter_times, X_test, y_test, batch_num):
    gamma = 0.9
    E = matrix(zeros(w.shape))
    epsilon = 1e-05
    Loss = []
    for i in range(iter_times):
        X_randdata, y_randdata = randChoice(X_data,y_data,batch_num)
        f = multiply(y_randdata, X_randdata * w)
        grad = - X_randdata.T*(multiply(f < 1, y_randdata)) / X_randdata.shape[0]
        E = gamma * E + (1 - gamma) * multiply(grad, grad)
        w = w - multiply(learn_rate / sqrt(E + epsilon), grad)
        f2 = multiply(y_test, X_test * w)
        z = matrix(np.c_[array(1-f2), zeros(f2.shape)])
        L = (z.max(1).sum())/X_test.shape[0]
        Loss.append(L)
    return w, Loss
```

4)drawing graph of LNAG,LRMSProp,Ladadelta,LAdam and with the number of iterations.

```python
def SVM(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
    L_NAG, L_RMSProp, L_Adadelta, L_Adam = GradientDescent(X_train,y_train,0.001,2000, X_test,y_test,256)
    plt.figure('logistic regression')
    plt.plot(arange(2000),L_NAG, 'r', label='NAG')
    plt.plot(arange(2000),L_RMSProp, 'g', label='RMSProp')
    plt.plot(arange(2000),L_Adadelta, 'cyan', label='Adadelta')
    plt.plot(arange(2000),L_Adam, 'b', label='Adam')
    plt.legend(loc='best')
    plt.show()

X, y = load_svmlight_file("logistic_data.txt")
X = X.todense()
X = np.array(X)
SVM(X, y)
```



IV. CONCLUSION

I further understand the principle of Logistic regression ,linear classification and stochastic gradient descent. At the same time we should if work hard if we want to leran something.