The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Ruiyong Li

Supervisor:
Mingkui Tan or Qingyao Wu

Student ID：201530612033

Grade: 2015
Undergraduate or Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—

- 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.

I. Introduction

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

II. Methods and Theory

Logistic Regression and Stochastic Gradient Descent
Linear Classification and Stochastic Gradient Descent
NAG, RMSProp, AdaDelta and Adam

III. Experiment

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 G 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 L_{NAG} , $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.
7. Repeate step 4 to 6 for several times, and **drawing graph of L_{NAG} , $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} 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 G 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 L_{NAG} , $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.
7. Repeate step 4 to 6 for several times, and **drawing graph of L_{NAG} , $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} with the number of iterations**.

Finishing experiment report according to result: The template of report can be found in example repository.

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

#load data
def get_data(path, n_features=None):
    if n_features == None:
        X, y = datasets.load_svmlight_file(path)
    else:
        X, y = datasets.load_svmlight_file(path, n_features=n_features)
    #append one column
    X = np.hstack([X.toarray(), np.ones((X.shape[0], 1))])
    y = np.array(y).reshape(X.shape[0],1)
    y[y==-1] = 0 #if y == -1, then y = 0
    return X, y

#loss_function
def compute_loss(X, y ,theta):
    y_pred = sigmod(X.dot(theta))
    loss = -1./X.shape[0] * (y*np.log(y_pred)+ (1-y)*np.log(1-y_pred)).sum()
    return loss

#gradient value
def gradient(X, y, theta):
    g = 1./X.shape[0] * np.dot(X.transpose(), sigmod(X.dot(theta))-y)
    return g
```

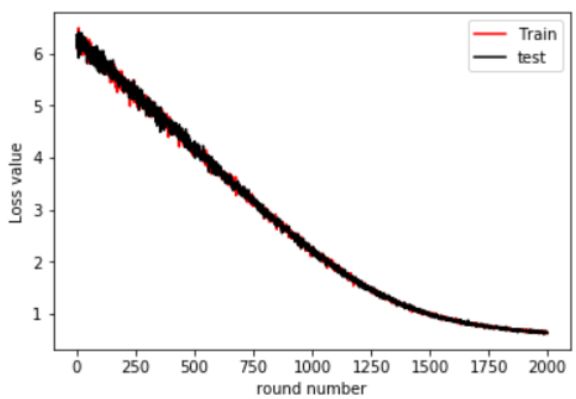
```
#sigmod function
def sigmod(z):
    return 1./(1+ np.exp(-z))

#get part of sample
def get_part(X, y ,min_part):
    i = np.random.randint(0, X.shape[0], size=min_part, dtype=int) #generate random int from 0 to 32561
    return X[i,:], y[i]

#show function
def show(train_loss, test_loss):
    plt.plot(train_loss,'red',label='Train')
    plt.plot(test_loss,'black',label='test')
    plt.xlabel('round number')
    plt.ylabel('Loss value')
    plt.legend()
    plt.show()
```

```
def LogisticRegression(X_train, y_train, theta, item,
                       learning_rate=0.01,
                       optimizer=None,
                       optimizer_params=None):
    if optimizer == None:
        gred = gradient(X_train, y_train, theta)
        theta = theta - learning_rate*gred
    elif optimizer == "NAG":
        #initialize v and Gamma
        v = np.zeros(theta.shape)
        Gamma = 0.9

        grad = gradient(X_train, y_train, theta- Gamma*v)
        v = Gamma*v + learning_rate*grad
        theta = theta - v

    elif optimizer == "RMSProp":
        G = np.zeros(theta.shape)
        Gamma = 0.9
        Epsilon = 1e-7

        grad = gradient(X_train, y_train, theta)
        G = Gamma*G + (1-Gamma)*(grad**2)
        theta = theta - learning_rate*grad/(np.sqrt(G+Epsilon))
```

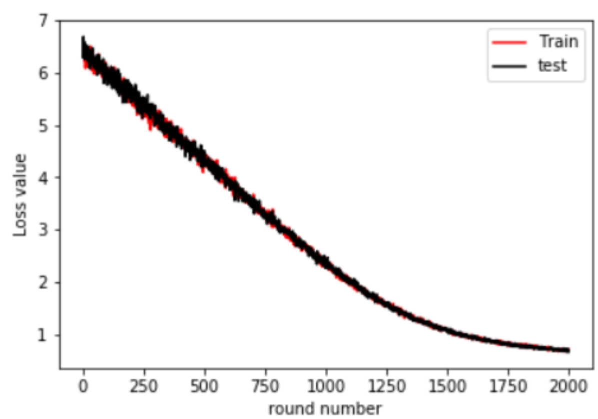
```
elif optimizer == "Adadelta":
    G = np.zeros(theta.shape)
    Delta = np.zeros(theta.shape)
    Gamma = 0.9
    Epsilon = 1e-7

    grad = gradient(X_train, y_train, theta)
    G = Gamma*G + (1-Gamma)*(grad**2)
    DeltaTheta = -((np.sqrt(Delta+Epsilon))/(np.sqrt(G+Epsilon)))*grad

    theta = theta + DeltaTheta
    Delta = Delta*Gamma + (1-Gamma)*(DeltaTheta**2)

elif optimizer == "Adam":
    m = np.zeros(theta.shape)
    G = np.zeros(theta.shape)
    Alpha = np.zeros(theta.shape)
    Beta = 0.9
    Gamma = 0.999
    Epsilon = 1e-8

    grad = gradient(X_train, y_train, theta)
    m = Beta*m + (1-Beta)*grad
    G = Gamma*G + (1-Gamma)*(grad**2)
    Alpha = learning_rate * (np.sqrt(1-Gamma))/(1-Beta)
    theta = theta - Alpha*m/(np.sqrt(G + Epsilon))

return theta
```

IV. CONCLUSION

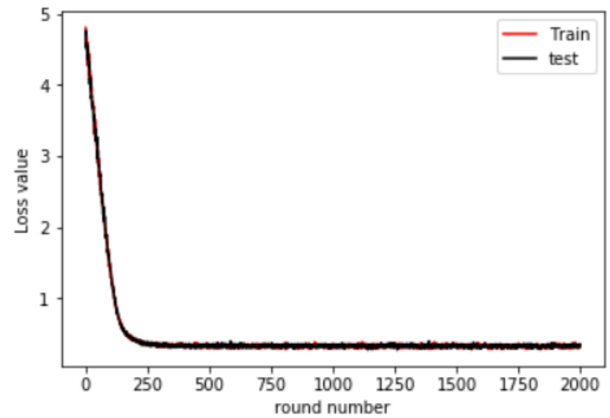
without optimizer



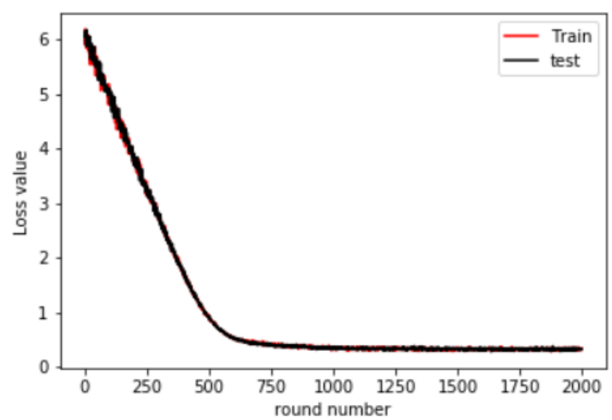
NAG



RMSProp



Adadelta



Adam