South China University of Technology

The Experiment Report of Deep Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Ana Madeleyn Oporto Guzmán

Supervisor:
Mingkui Tan

Student ID：201722800070

Grade: 2017
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract

Logistic regression is named for the function used at the core of the method, the logistic function. Logistic regression uses an equation as the representation, very much like linear regression. Input values (**X**) are combined linearly using weights or coefficient values to predict an output value (**y**).

Gradient Descent is the process of minimizing a function by following the gradients of the cost function. This involves knowing the form of the cost as well as the derivative so that from a given point you know the gradient and can move in that direction, e.g. downhill towards the minimum value.

A **linear classifier** achieves this by making a classification decision based on the value of a linear combination of the characteristics; so in this experiment we try to:

1. Compare and understand the difference between gradient descent and stochastic gradient descent.
2. Compare and understand the differences and relationships between Logistic regression and linear classification.
3. Further understand the principles of SVM and practice on larger data.

KEY WORDS: Logistic regression, linear classification and stochastic gradient descent.

I. INTRODUCTION

Logistic regression was developed by statistician David Cox in 1958. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the odds of a given outcome by a specific factor. In this experiment we use some sophisticated ensemble algorithm on the data set (eg: **stochastic gradient descent).** Gradient Descent is the process of minimizing a function by following the gradients of the cost function.

This involves knowing the form of the cost as well as the derivative so that from a given point you know the gradient and can move in that direction, e.g. downhill towards the minimum value. In machine learning, we can use a technique that evaluates and updates the coefficients every iteration called stochastic gradient descent to minimize the error of a model on our training data. The way this optimization algorithm works is that each training instance is shown to the model one at a time. The model makes a prediction for a training instance, the error is calculated and the model is updated in order to reduce the error for the next prediction.

This procedure can be used to find the set of coefficients in a model that result in the smallest error for the model on the training data.

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.

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. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.

- For the first part of the experiment the dataset is a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. The data analysis is ***Logistic Regression and Stochastic Gradient Descent***

- For the second part the dataset is a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. The data analysis is ***Linear Classification and Stochastic Gradient Descent***

II. METHODS AND THEORY

Here, we will like to introduce some related works based on the methods and theory of the logistic methods for regression and linear methods for classification.

One drawback with the SVM is that the method does not explicitly output a probability or likelihood

of the labels, instead the output is a real value the magnitude of which should be monotonic with respect to the probability P.

This issue can be addressed by using a loss function based upon logistic or binary regression. The main idea behind logistic regression is that we are trying to model the log likelihood ratio by the function

f(x).

Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Sometimes logistic regressions are difficult to interpret; the Intellect us Statistics tool easily allows you to conduct the analysis, then in plain English interprets the output.

Binary logistic regression major assumptions:

1. The dependent variable should be dichotomous in nature (e.g., presence vs. absent).
2. There should be no outliers in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
3. There should be no high correlations (multi collinearity) among the predictors. This can be assessed by a correlation matrix among the predictors. Tabachnick and Fidell (2013) suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met.

At the center of the logistic regression analysis is the task estimating the log odds of an event. Mathematically, logistic regression estimates a multiple linear regression function defined as: logit (p) for i = 1…n .

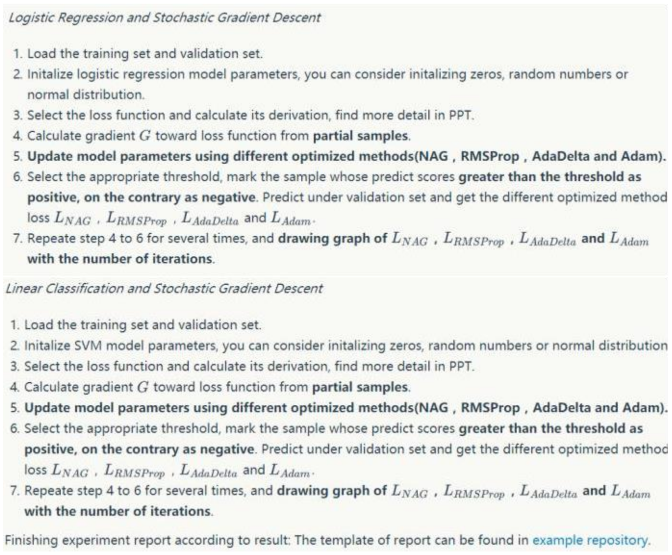
Overfitting : When selecting the model for the logistic regression analysis, another important consideration is the model fit. Adding independent variables to a logistic regression model will always increase the amount of variance explained in the log odds (typically expressed as R^2). However, adding more and more variables to the model can result in overfitting, which reduces the generalizability of the model beyond the data on which the model is fit.

Supervised learning:

Training data: {(x1, g1),(x2, g2), ...,(xN , gN)} The feature vector X = (X1,X2, ...,Xp), where each variable Xj is quantitative. The response variable G is categorical. G ∊ G = {1, 2, ...,K} Form a predictor G(x) to predict G based on X. Email spam: G has only two values, say 1 denoting a useful email and 2 denoting a junk email. X is a 57-dimensional vector, each element being the relative frequency of a word or a punctuation mark.

G(x) divides the input space (feature vector space) into a collection of regions, each labeled by one class.

III. EXPERIMENT



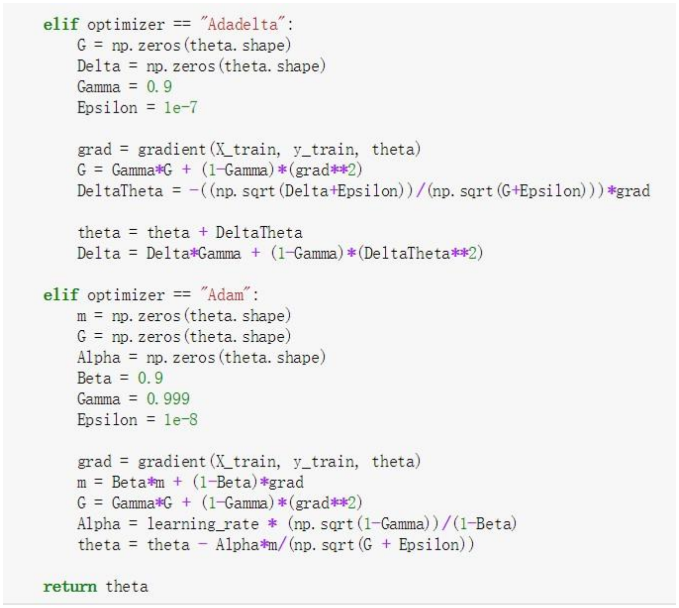
Logistic Regression and Stochastic Gradient Descent

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**.

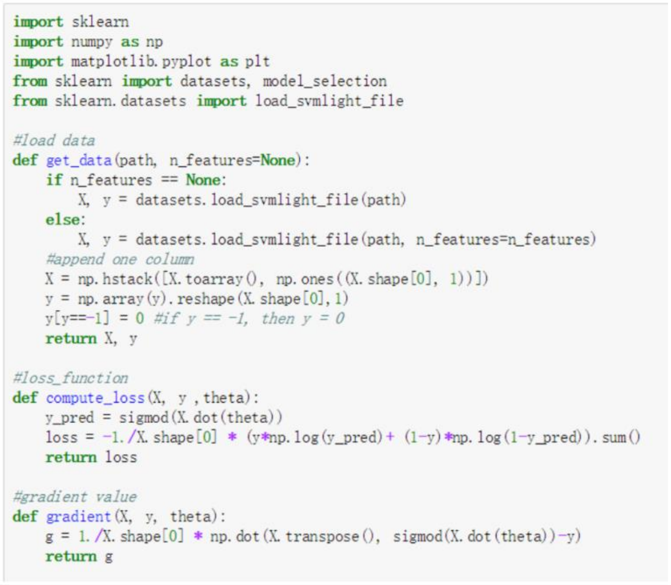
Linear Classification and Stochastic Gradient Descent

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**.

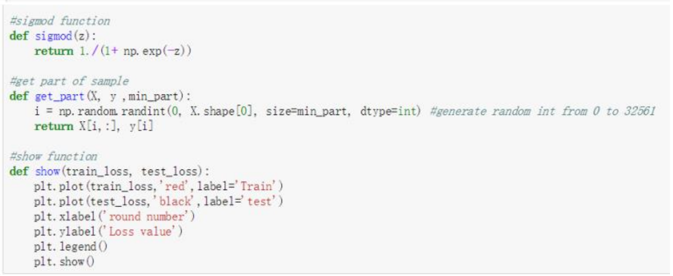
Finishing experiment report according to result: The template of report can be found in example repository.



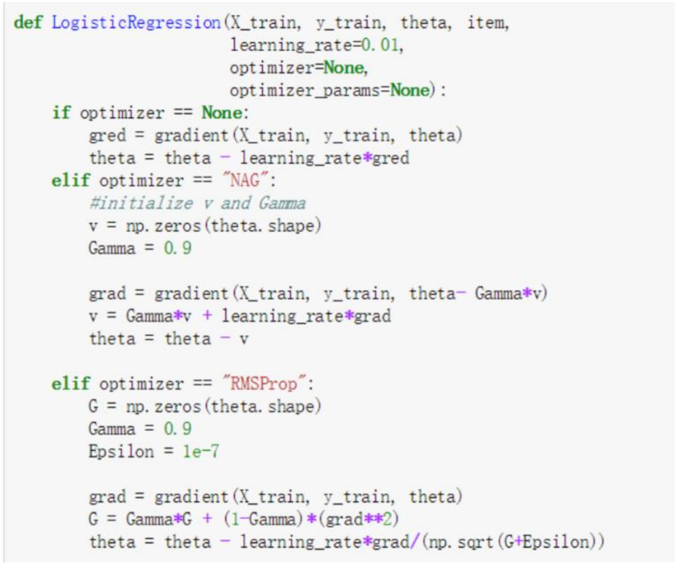
```
    elif optimizer == "Adadelta":
        G = np.zeros(theta.shape)
        Delta = np.zeros(theta.shape)
        Gamma = 0.9
        Epsilon = 1e-7

        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
```



```
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
```



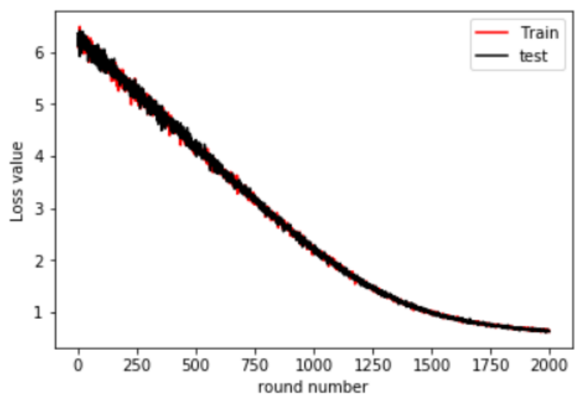
```
#sigmod function
def sigmod(z):
    return 1./(1+ np.exp(-z))

#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()
```



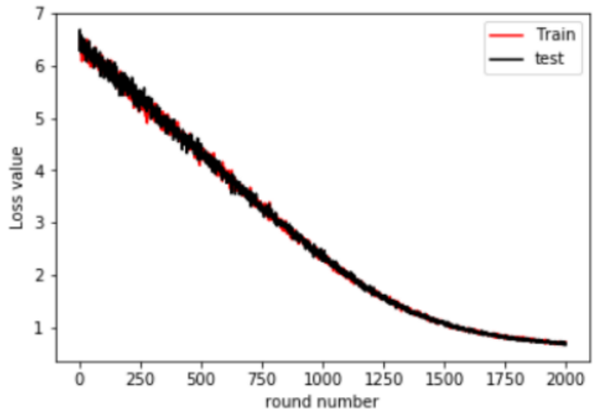
```
def LogisticRegression(X_train, y_train, theta, item,
                       learning_rate=0.01,
                       optimizer=None,
                       optimizer_params=None):
    if optimizer == None:
        gred = gradient(X_train, y_train, theta)
        theta = theta - learning_rate*gred
    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))
```

IV. CONCLUSION

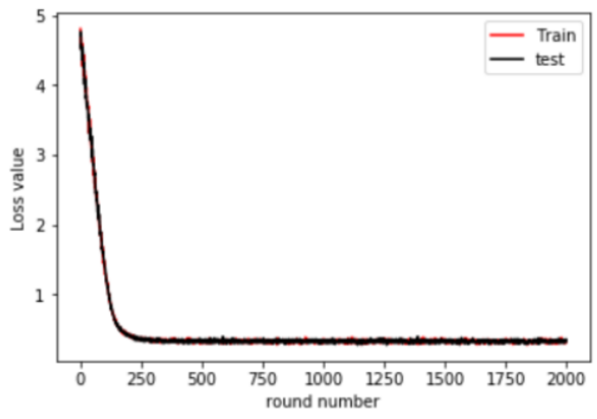
without optimizer



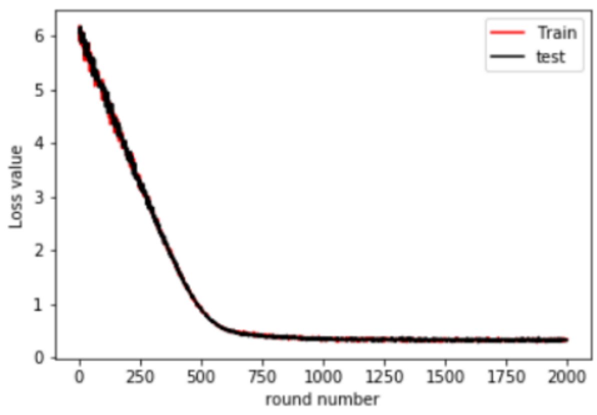
NAG



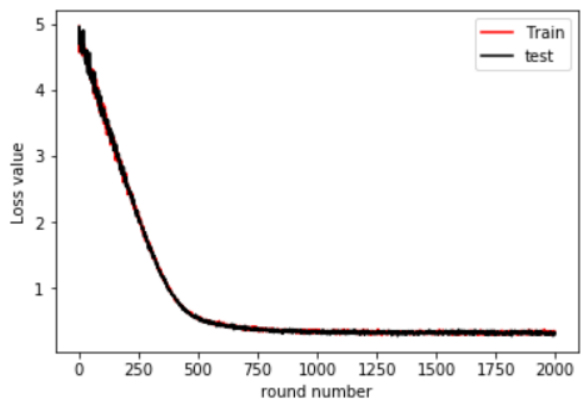
RMSProp



Adadelta



Adam



In the experiment we have seen the logistic regression and linear classification with stochastic gradient descent where we Compare and understand the difference between gradient descent and stochastic gradient descent and we understand the principles of SVM and practice on larger data.

By analyzing this results:

Logistic regression and stochastic gradient descent

As the algorithm sweeps through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm converges. If this is done, the data can be shuffled for each pass to prevent cycles. Typical implementations may use an adaptive learning rate so that the algorithm converges.

A compromise between computing the true gradient and the gradient at a single example is to compute the gradient against more than one training example (called a "mini-batch") at each step. This can perform significantly better than "true" stochastic gradient descent described, because the code can make use of vectorization libraries rather than computing each step separately. It may also result in smoother convergence, as the gradient computed at each step uses more training examples.

-Linear classification and stochastic gradient descent

We can see the advantages of Stochastic Gradient Descent:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

And the disadvantages of Stochastic Gradient Descent include:

-SGD requires a number of hyper-parameters such as the regularization parameter and the number of iterations.

-SGD is sensitive to feature scaling.

The SGD supports multi-class classification by combining multiple binary classifiers in a “one versus all” (OVA) scheme. For each of the K classes, a binary classifier is learned that discriminates between that and all other $K - 1$ classes.

And to conclude we would like to say The major advantage of SGD is its efficiency, which is basically linear in the number of training examples. If X is a matrix of size (n, p) training has a cost of $O(kn\bar{p})$, where k is the number of iterations (epochs) and $\bar{p}$ is the average number of non-zero attributes per sample.