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:
Kai Song

Supervisor:
Qingyao Wu

Student ID：
201530612729

Grade:
Undergraduate

December 9, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—As wo all know , Logistic regression is a simple classification algorithm for learning to make such decisions. This experiment we utilize logistic regression and linear classification respectively for binary classification issue, and update model parameters in different opti-mization methods (NAG, RMSProp, AdaDelta and Adam) to accelerate the loss function to get a satisfied result.

I. Introduction

In logistic regression we use a different hypothesis class to try to predict the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class. Constantly, we use no more than four different optimiza-tion methods (NAG, RMSProp, AdaDelta and Adam), which contribute to accelerate the convergence speed of gradient descent.

II. Methods and Theory

A. logistic regression

The logistic regression is called as a linear classifier because it produces a decision boundary which is linear in nature. So, the classification makes by logistic regression is linear classification only.Specifically, we try to learn a function of the form:

$$P(y=1|x) = h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)} \equiv \sigma(\theta^\top x),$$
$$P(y=0|x) = 1 - P(y=1|x) = 1 - h_\theta(x).$$

$$\sigma(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}}$$

The functionσ(z)≡1+exp⁡(−z) is often called the "sigmoid" or "logistic" function – it is an S-shaped function that "squashes" the value of θ⊤x into the range [0,1] so that we may interpret hθ(x) as a probability. Our goal is to search for a value of θ so that the probability P(y=1|x)=hθ(x) is large when xx belongs to the "1" class and small when xx belongs to the "0" class (so that P(y=0|x) is large). For a set of training examples with binary labels {(x(i),y(i)):i=1,…,m} the following cost function measures how well a given hθ does this:

$$J(\theta) = -\sum_i \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) \right).$$

To minimize J(θ) we can use the same tools as for linear regression. We need to provide a function that computes J(θ) and ∇J(θ) for any requested choice of θ. The derivative of J(θ) as given above with respect to θj is:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} (h_\theta(x^{(i)}) - y^{(i)}).$$

Written in its vector form, the entire gradient can be expressed as:

$$\nabla_\theta J(\theta) = \sum_i x^{(i)} (h_\theta(x^{(i)}) - y^{(i)})$$

This is essentially the same as the gradient for linear regression except that now hθ(x)=σ(θ⊤x).

B. Stochastic Gradient Descent

The standard gradient descent algorithm updates the parameters θ of the objective J(θ)J(θ) as:

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$$

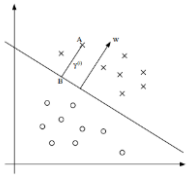
where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. Stochastic Gradient Descent (SGD) simply does away with the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by:

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

with a pair (x(i),y(i)) from the training set.

C. Support Vector Machine

Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.



$$f(\vec{w}, b) = \left[\frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(w \cdot x_i - b)) \right] + \lambda \|w\|^2.$$

In light of the above discussion, we see that the SVM technique is equivalent to empirical risk minimization with

Tikhonov regularization, where in this case the loss function is the hinge loss.

$$\ell(y, z) = \max(0, 1 - yz).$$

From this perspective, SVM is closely related to other fundamental classification algorithms such as regularized least-squares and logistic regression. The difference between the three lies in the choice of loss function: regularized least-squares amounts to empirical risk minimization with the square-loss, logistic regression employs the log-loss:

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz}).$$

SVMs belong to a family of generalized linear classifiers and can be interpreted as an extension of the perceptron. They can also be considered a special case of Tikhonov regularization. A special property is that they simultaneously minimize the empirical classification error and maximize the geometric margin; hence they are also known as maximum margin classifiers.

D. Nesterov accelerated gradient

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.

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

We would also like to adapt our updates to each individual parameter to perform larger or smaller updates depending on their importance.

E. Adadelta

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.

Instead of inefficiently storing W previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average E[g2]tE[g2]t at time step t then depends (as a fraction γ similarly to the Momentum term) only on the previous average and the current gradient

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

With Adadelta, we do not even need to set a default learning rate, as it has been eliminated from the update rule.

F. RMSprop

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class.

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 to the first update vector of Adadelta that we derived above:

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

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

G. Adam

Adaptive Moment Estimation (Adam) 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, Adam also keeps an exponentially decaying average of past gradients mt, similar to momentum:

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

beta_1 is 0.9（may need decent），gamma is 0.999，eta is 0.001,sometime may also need decent .

III. Experiments

A. Dataset

The data set used in this experiment is a9a, provided by LIBSVM Data. In this dataset, continuous features are discretized into quantiles, and each quantile is represented by a binary feature. In all, this data set contains 32,561 records for training and 16,281 records for testing and each sample has 123/123 (testing) features.

B. Implementation

Step1:Initialization: The initialization of the parameters are set up in each optimiza-tion methods and the iteration of the training is 500.

Step2:Coding: The regression and classification are coding partly, while each contain same optimiza-tion methods (NAG, RMSProp, AdaDelta and Adam).

The kernel code of regression experiment will show in Fig.1 and the classification experiment will show in Fig.2 ,which contain the main method of loss function and gradient decent.

Step3:Result: We implementation the training methods and training in same iteration, which is 500 cycles. On the other hand , I list the each loss of the optimiza-tion methods (NAG, RMSProp, AdaDelta and Adam), and each are shown in Table I.

```
def gradient_sgd(w):
    random_num = random.randint(0,m)
    return (X_train[random_num].T *
(sigmoid(X_train[random_num] * w) -
y_train[random_num]))

def sigmoid(z):
    return 1/(1+exp(-z))

def loss(x,y,w):
    return  -( y*log(sigmoid(x * w))
+ (1-y)*log(1-sigmoid(x *
w)) ).sum()/ x.shape[0]

loss_train,loss_test = ([],[])
m, n = np.shape(X_train)
w = np.ones((n, 1))
        alpha = 0.01
```

Fig.1 Kernel code of Regression

```
def stochastic_gradient(w):
    index =  (1 - y_train * (X_train
* w) < 0)
    y = y_train.copy()
    y[index] = 0
    randomNum =
np.random.randint(0,X_train.shape[0
])
    Epgradient = -
((X_train)[randomNum].T *
y[randomNum]).reshape(123,1)
    gradient = w +  Epgradient
    return gradient

def loss(x,y,w):
    Eploss = 1 - y * x.dot(w)
    Eploss[Eploss < 0] = 0
    loss = 0.5 *
np.dot(w.transpose(), w ).sum() +
Eploss.sum()
    return loss / x.shape[0]

loss_train, loss_test = ([],[])
def gradientDescent(w):
    for i in range (0, iteration ):
        gradient =
stochastic_gradient(w)
        w -= alpha * gradient

loss_train.append(loss(X_train ,
y_train ,w))

loss_test.append(loss(X_
train , y_train ,w))
```

Fig.2 Kernel code of Classification

TABLE I
Loss of Regression for Optimization Methods

SGD	0.62020
NAG	0.45308
AdaDelta	0.46806
RMSProp	4.97527
Adam	0.79916

TABLE I I
Loss of Classification for Optimization Methods

GD	0.50795
NAG	0.48792
AdaDelta	0.50877
RMSProp	5.67612
Adam	0.50575

At the same time ,we get the graph of each optimiza-tion methods in both experiment ,which shows the magnificent difference among the methods. Shows in Fig.3 and Fig.4.

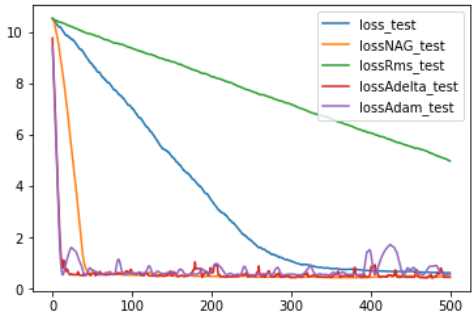


Fig.3 Loss of five different gradient methods in Regression experiment

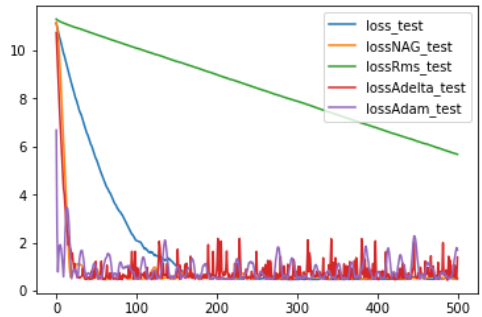


Fig.4 Loss of five different gradient methods in Classification experiment

IV. Conclusion

This experiment implement the Logistic Regression, Linear Classification and Stochastic Gradient Descent. This experiment help us compare and understand the difference between gradient descent and stochastic gradient descent, the differences and relationships between Logistic regression and linear classification, and further understand the principles of SVM and practice on larger data. Moreover, I've learned about almost six types of optimization methods for gradient descent, which I've never heard, in addition there are ever more optimiza-tion methods are waited to be found and study. It is a long way for us to study machine learning ,only in aspiration can we study further more and gain more.