華南理工大學

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Zhishang Zhou

Supervisor:
Mingkui Tan and Qingyao Wu

Student ID:
201720144931

Grade:
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

***Abstract*—In this experiment report, I will show you how Logistic Regression, Linear Classification and Stochastic Gradient Descent works. Firstly, I will introduce how those things being proposed. Secondly, some details about those method and theory will being displayed. Thirdly, detailed experiments will be shown. Finally, I will give a brief conclusion.**

I. Introduction

In statistics, logistic regression, or logit regression, or logit model is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variablethat is, where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

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 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.

Generally, we iteratively solve those two problem by using Stochastic Gradient Descent method. Batch methods, such as limited memory BFGS, which use the full training set to compute the next update to parameters at each iteration tend to converge very well to local optima. They are also straight forward to get working provided a good off the shelf implementation (e.g. minFunc) because they have very few hyper-parameters to tune. However, often in practice computing the cost and gradient for the entire training set can be very slow and sometimes intractable on a single machine if the dataset is too big to fit in main memory. Another issue with batch optimization methods is that they dont give an easy way to incorporate new data in an online setting. Stochastic Gradient Descent (SGD) addresses both of these issues by following the negative gradient of the objective after seeing only a single or a few training examples. The use of SGD In the neural network setting is motivated by the high cost of running back propagation over the full training set. SGD can overcome this cost and still lead to fast convergence.

II. Methods and Theory

Logistic Regression: Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. 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.

In linear regression we tried to predict the value of $y^(i)$ for the ith example $x^{(i)}$ using a linear function $y = h_\theta(x) = \theta^T x$. This is clearly not a great solution for predicting binary-valued labels $y^{(i)} \in \{0, 1\}$. 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. Specifically, we will try to learn a function of the form:

$$P(y = 1|x) = h_{theta}(x) = \frac{1}{1 + exp(\theta^T x)} = \sigma(\theta^T x),$$

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - h_{theta}(x)$$

The function $\sigma(z) = \frac{1}{1+exp(-z)}$ is often called the sigmoid or logistic function it is an S-shaped function that squashes the value of $\theta^T x$ into the range $[0, 1]$ so that we may interpret $h_{theta}(x)$ as a probability. Our goal is to search for a value of θ so that the probability $P(y = 1|x) = h_{theta}(x)$ is large when x belongs to the 1 class and small when x 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 h_θ does this:

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

Note that only one of the two terms in the summation is non-zero for each training example (depending on whether the label $y^{(i)}$ is 0 or 1). When $y^{(i)} = 1$ minimizing the cost function means we need to make $h_\theta(x^{(i)})$ large, and when $y^{(i)} = 0$ we want to make $1 - h_\theta(x^{(i)})$ large as explained above.

We now have a cost function that measures how well a given hypothesis $h_\theta(x^{(i)})$ fits our training data. We can learn to classify our training data by minimizing $J(\theta)$ to find the best choice of θ. Once we have done so, we can classify a new test point as 1 or 0 by checking which of these two class labels is most probable: if $P(y = 1|x) > P(y = 0|x)$ then we label the example as a 1, and 0 otherwise. This is the same as checking whether $h_\theta(x^{(i)}) > 0.5$.

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

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

This is essentially the same as the gradient for linear regression except that now $h_theta(x) = \sigma(\theta^T x)$.

Linear Classification: Given training data (x_i, y_i) for $i = 1...n$, with $x_i \in R_m$ and $y_i \in 1, 1$, learn a classfier $f(x)$ such that $y_i f(x_i) > 0$ for a correct classification. And $f(x)$ often takes the form just like Linear Regression's model. Linear classification attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model. Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable causes the other (for example, higher SAT scores do not cause higher college grades), but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear classification line has an equation of the form $Y = a + bX$, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when $x = 0$).

Stochastic Gradient Descent: The standard gradient descent algorithm updates the parameters θ of the objective $J(\theta)$ 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.

Generally each parameter update in SGD is computed w.r.t a few training examples or a minibatch as opposed to a single example. The reason for this is twofold: first this reduces the variance in the parameter update and can lead to more stable convergence, second this allows the computation to take advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient. A typical minibatch size is 256, although the optimal size of the minibatch can vary for different applications and architectures.

In SGD the learning rate α is typically much smaller than a corresponding learning rate in batch gradient descent because there is much more variance in the update. Choosing the proper learning rate and schedule (i.e. changing the value of the learning rate as learning progresses) can be fairly difficult. One standard method that works well in practice is to use a small enough constant learning rate that gives stable convergence in the initial epoch (full pass through the training set) or two of training and then halve the value of the learning rate as convergence slows down. An even better approach is to evaluate a held out set after each epoch and anneal the learning rate when the change in objective between epochs is below a small threshold. This tends to give good convergence to a local optima. Another commonly used schedule is to anneal the learning rate at each iteration t as $\frac{a}{b+t}$ where a and b dictate the initial learning rate and when the annealing begins respectively. More sophisticated methods include using a backtracking line search to find the optimal update.

One final but important point regarding SGD is the order in which we present the data to the algorithm. If the data is given in some meaningful order, this can bias the gradient and lead to poor convergence. Generally a good method to avoid this is to randomly shuffle the data prior to each epoch of training. In this experiment we will implement SGD variants – NAG, RMSprop, AdaDelta, Adam.

III. Experiments

A. Dataset

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

B. Implementation

Logistic Regression and Stochastic Gradient Descent:

1. Load the experiment data. I use load_svmlight_file function in sklearn library.
2. Initialize linear model parameters.I use normal distribution to initialize my model parameters with $\mu = 0$ and $\sigma = 0.1$.
3. Choose loss function and derivation: I choose mean square loss function to demonstrate the model's behavior.
4. Calculate a batch gradient G toward loss function from all samples.
5. Using gradient G, implement different optimized methods(NAG, RMSProp, AdaDelta and Adam) to get update direction D.
6. Update model: $W_t = W_{t-1} + \eta' D$. η' is learning rate decided by different optimized methods(NAG, RMSProp, AdaDelta and Adam).
7. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Here I set the threshold as 0.5. Predict under validation set and get the different

optimized method loss L_{NAG},$L_{RMSprop}$,$L_{AdaDelta}$ and L_{ADAM}.

8. Repeate step 4 to 7 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 experiment data. I use load_svmlight_file function in sklearn library.
2. Initalize SVM model parameters.I use normal distribution to initialize my model parameters with $\mu = 0$ and $\sigma = 0.1$.
3. Choose loss function and derivation: I choose hinge loss function to demonstrate the model's behavior.
4. Calculate a batch gradient G toward loss function from all samples.
5. Using gradient G, implement different optimized methods(NAG, RMSProp, AdaDelta and Adam) to get update direction D.
6. Update model: $W_t = W_{t-1} + \eta' D$. η' is learning rate decided by different optimized methods(NAG, RMSProp, AdaDelta and Adam).
7. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Here I set the threshold as 0.5. Predict under validation set and get the different optimized method loss L_{NAG},$L_{RMSprop}$,$L_{AdaDelta}$ and L_{ADAM}.
8. Repeate step 4 to 7 for several times, and drawing graph of L_{NAG},$L_{RMSprop}$,$L_{AdaDelta}$ and L_{ADAM} with the number of iterations.

C. Results

We run the training loops for 1000 times and plot the loss value shown as below:

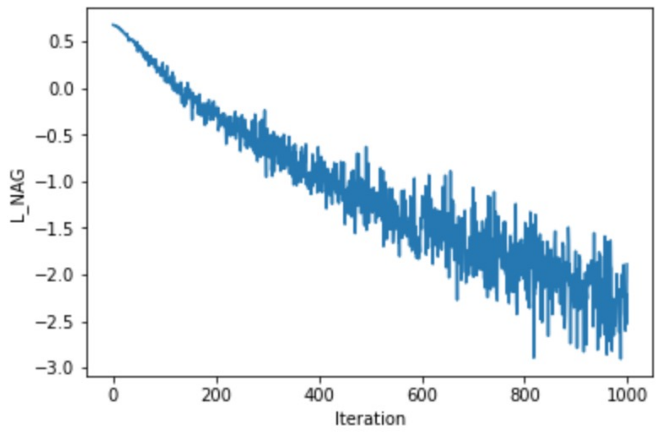

Fig. 1. Logistic Regression's L_{NAG}

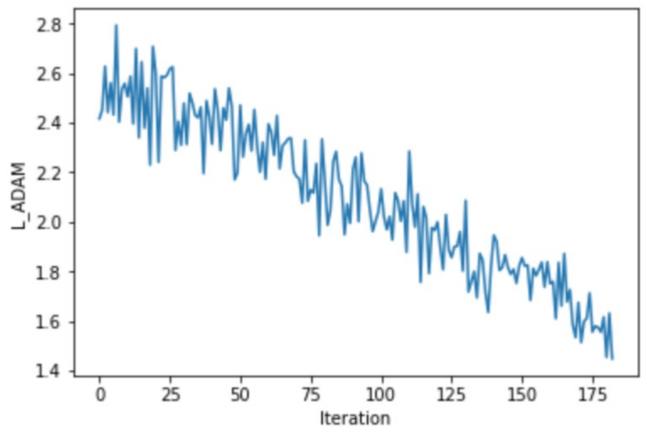

Fig. 2. Logistic Regression's L_{Adam}

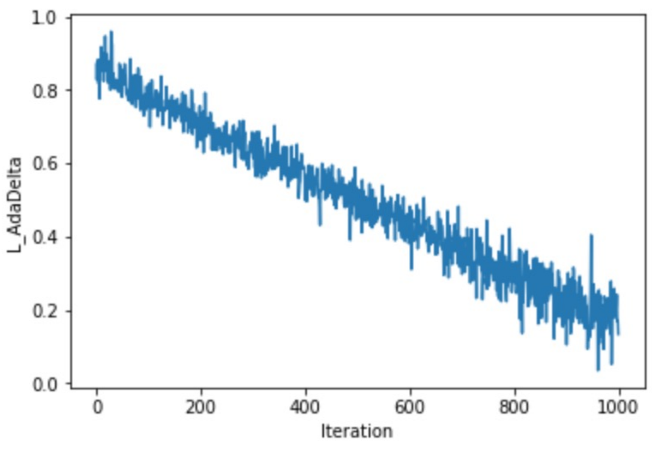

Fig. 3. Logistic Regression's $L_{AdaDelta}$

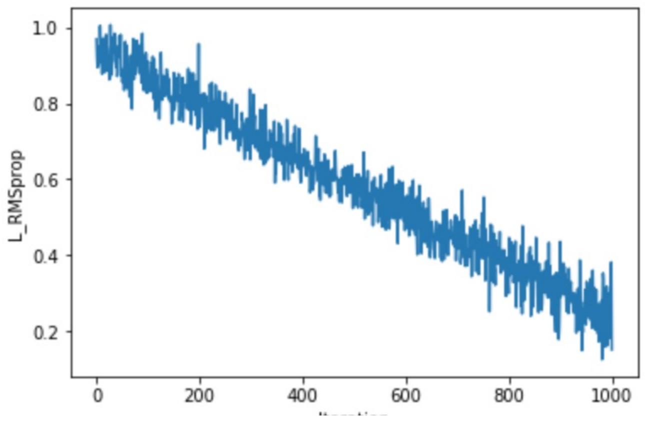

Fig. 4. Logistic Regression's $L_{RMSprop}$

IV. Conclusion

In this experiments, I have implemented a logistic regression model which behaves well using stochastic gradient descent optimization method. I also made a linear SVM classification method with penalty which classifies the data very well. As I implemented those models and methods from head to tail, I faced a lot of bugs which really confused me and depressed me. As soon as I made my first linear regression model, the works afterwords become steady. As you can see my implementations behave wired. Although I spend lots of time

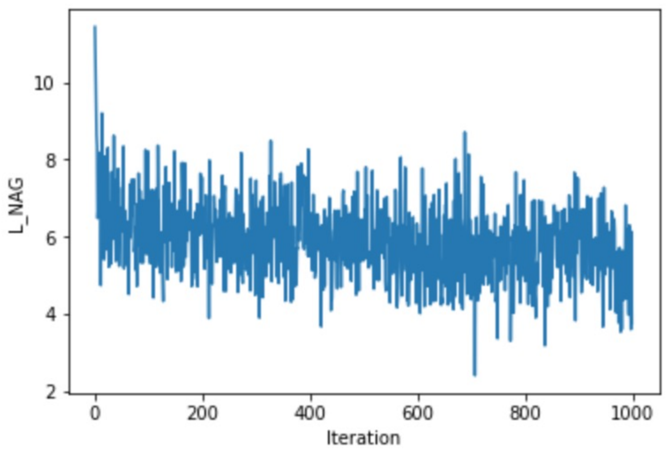

Fig. 5. Linear Classification's L_{NAG}

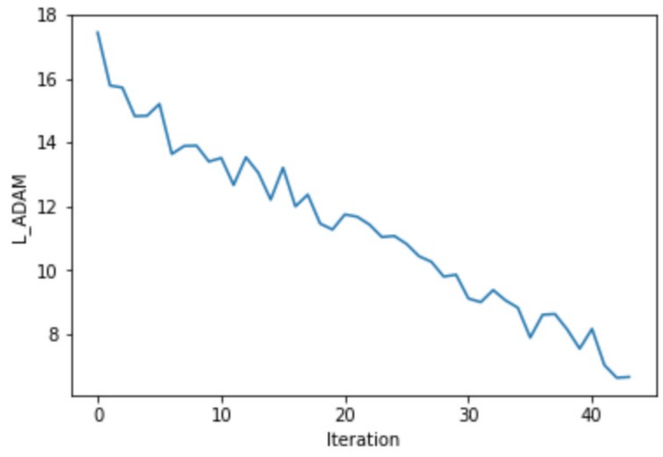

Fig. 6. Linear Classification's L_{Adam}

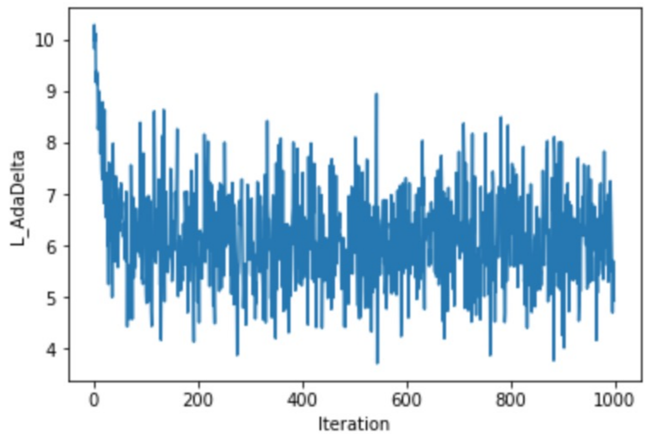

Fig. 7. Linear Classification's $L_{AdaDelta}$

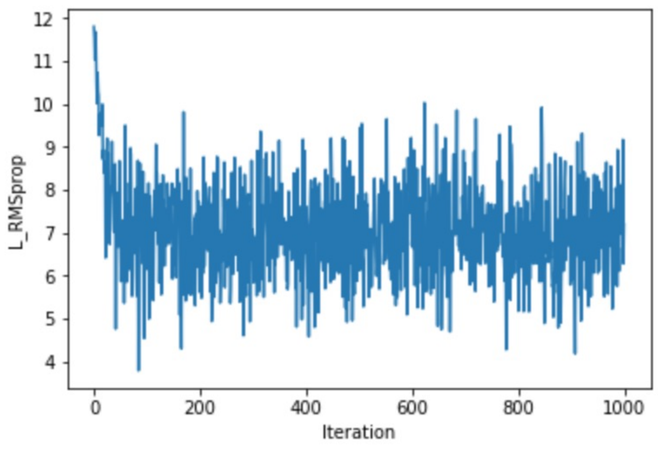

Fig. 8. Linear Classification's $L_{RMSprop}$

to debug, I still can't figure that out. I wish take some of assignment lessons from you to help me fix my model's wired behavior