IIT Kanpur

Convex Optimisation Term Paper

Stochastic Gradient Descent

Anupreet Porwal
Vibhuti Mahajan

Supervisor:
Prof. Ketan Rajawat

April 17, 2016

Stochastic Gradient Descent

Anupreet Porwal, Vibhuti Mahajan

Abstract

With the growing size of datasets and increased emphasis on large scale learning in the past few decades, stochastic gradient descent has gained popularity for large dataset. In this report, we look at application of the same algorithm to estimate the parameter in logistic regression,SVM and K means algorithm, and compare the time complexity and efficiency of these algorithm with other readily available packages in R.

Keywords: Credit Scoring

1. Introduction

Gradient Descent is an algorithm for solving minimising problems of the form $min_\theta J(\theta)$. For a function plotted as shown in figure 1, what gradient descent does is it starts at some point on the surface of the function and in each iteration it goes downhill in the direction of the steepest descent, until it reaches the minimum.

`Algorithm`
Repeat until convergence {
$\theta_j = \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$
}

This is the update equation of a particular parameter θ_j, and α is called the learning rate. This update equation is carried out simultaneously for all the parameters. Hence this is a simple algorithm in which on every step we take partial derivative of the optimisation objective with respect to our parameters and take a small step in the direction of decreasing gradient. We usually take α to be a constant for practical purposes, because although initially a variable alpha might take large steps but as the algorithm progresses, the learning rate becomes smaller and smaller. Hence, no need to slowly decrease the α
There are different variants of gradient descent. We list below two of them

namely, Batch gradient descent and stochastic gradient descent.
We concern ourselves with *batch gradient descent* when the algorithm looks at the entire training set at a time, i.e. for a squared loss error $\frac{d}{d\theta_j}J(,a) = \frac{1}{m}\Sigma_i^m(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ in the inner for loop of the gradient descent algorithm, where $h_\theta(x)$ is the hypothesis function.

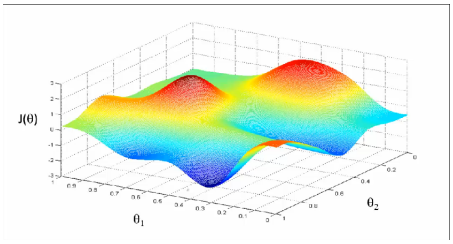


Figure 1: Gradient Descent

1.1. Data Preprocessing

Sometimes the features of the raw data are highly variant from each other and needs to be made comparable before proceeding with evaluation. Usually this preprocessing involves subtracting the mean from the features and diving by standard deviation (estimates in this case). This need arises from the fact that for a variant data the contours for descent are long ellipsoids and thus the algorithm needs to take very large steps in the direction of the steepest descent. On the other hand, a normalised data has more or less shperical contours and thus the descent algorithm is way faster in this case due to relatively small step size.

2. Stochastic Gradient Descent

An alternative algorithm that sometimes runs much faster than stochastic gradient descent is called the stochastic gradient descent (SGD). We begin our analysis with gradient descent. For linear models like SVM, regression etc., loss function can be defined as

$$min_\theta J(\theta) = \min_{\omega,b} J(\omega, b) = \frac{1}{m}\Sigma_i^m(h_{\omega,b}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\|\|\omega\|\|^2$$

where, ω is slope parameter and b is the intercept. The first term is the averaged square error and second term is the regularisation parameter added

to prevent over-fitting. Hence, on each iteration, the update equations for a particular parameter are:

$$\omega_j = \omega_j - \alpha \frac{d}{d\omega_j} J(\omega, b)$$

$$b = b - \alpha \frac{d}{db} J(\omega, b)$$

$$\omega_j = \omega_j - \alpha \frac{1}{m} \Sigma_i^m (h_{\omega,b}(x^{(i)}) - y^{(i)}) x_j^{(i)} - \alpha\lambda\omega_j \tag{1}$$

If m, the number of training samples is large, then to compute this derivative, we have to sum over all the training samples. In contrast, SGD looks at only 1 training sample at a time. SGD works as follows:
`Algorithm`
Repeat{
Randomly shuffle the data
For i = 1 to m{

$$\omega_j = \omega_j - \alpha(h_{\omega,b} x^{(i)} - y^{(i)}) x_j - \alpha\lambda\omega_j$$

$$b = b - \alpha(h_{\omega,b} x^{(i)} - y^{(i)})$$

} }
The intiuition behind this algorithm to work is that although the descent mechanism might proceed in a zig-zag, sometimes even away from the direction of maximum gradient, but eventually the algorithm reaches in a close vicinity of the optimal value. Also, the algorithm never really attains the minimum and parameters oscillate near the optimal value. But, this algorithm is way faster than conventional batch gradient descent for larger data sets as it avoids redundant calculations for large large datasets and hence improves the efficiency.
The workplan of this paper will follow incorporating gradient descent in K means algorithm, logistic regression model and SVM classifier and comparing the time taken to conventional ways.

3. Applications

In this section, we look at objective of different models discussed, look at their mathematical formulations followed by SGD update equations for corresponding model.

3.1. ***Multi class Logistic Regression- Application 1***

Any multi class classification problem can be formulated using multiple one vs all logistic regression model. We, therefore, look at the case of this model with when there are only two classes. Given a data point $x \in \mathbb{R}^n$, we model the probability of Y being a label to be a label $y \in \{-1, 1\}$.

$$P(Y = y|x, w, b) = \frac{1}{1 + exp(-y(w^T x + b))} \tag{2}$$

Thus, the log likelihood $L(w, b)$ is given by

$$L(w, b) = -\sum_{i=1}^{m} log(1 + exp(-y(w^T x + b))) \tag{3}$$

Our optimisation problem now becomes

$$max_{w,b} L(w, b) \iff min_{w,b} - L(w, b) = \sum_{i=1}^{m} log(1 + exp(-y(w^T x + b))) \tag{4}$$

Thus our update equation for Stochastic gradient descent algorithm now becomes
For any random training example x_*, y_*

$$\begin{aligned} w_{t+1} &= w_t + x_* y_* \frac{\eta}{1 + exp(y_* w^T x_*)} \\ b_{t+1} &= b_t + y_* \frac{\eta}{1 + exp(y_* w^T x_*)} \end{aligned} \tag{5}$$

3.2. ***Linear Support Vector Machine- Application 2***

Another conventional classification algorithm is that of SVM. Linear SVM tries to find a linear boundary to separate the training set examples of the two different categories with a maximum margin. thus our aim, here becomes to find a w such that

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{w^T w}{2} \\ \text{subject to} \quad & y_i(w^T x_i) >= 1 \; \forall i = 1, \ldots, m. \end{aligned}$$

where x is a feature vector belonging to $\mathbb{R}^d$ and y denotes the class assignment {1,-1}. Thus here, our update equation for SGD becomes
For a random training example y_t, w_t

$$w_{t+1} = w_t - \gamma \begin{cases} \lambda w & \text{if } y_t w^T x_t > 1 \\ \lambda w - y_t x_t, & \text{otherwise} \end{cases} \tag{6}$$

Again, as in the case of Logistic regression, any multi class classification problem can be thought of as solving multiple one vs all problem and there theory of linear SVM with only two classes can be used.

3.3. ***K Means Clustering- Application 3***

Changing the course of action, we turned our attention to the most common and widely used k-means clustering. As given in the 1967 paper of Macqueen, the algorithm proceeds by minimising the inter-cluster distance i.e.:

$$Q = min_k \frac{1}{2}(z - w_k)^2 \tag{7}$$

where $z, w_1, w_2, ..., w_k \in R^d$ and $n_1, n_2, ..., n_k \in N$, initially 0.
SGD changes th update equation as follows:

$$k* = argmin_k (z_t - w_k)^2 \tag{8}$$

$$n_k* \leftarrow n_k * +1 \tag{9}$$

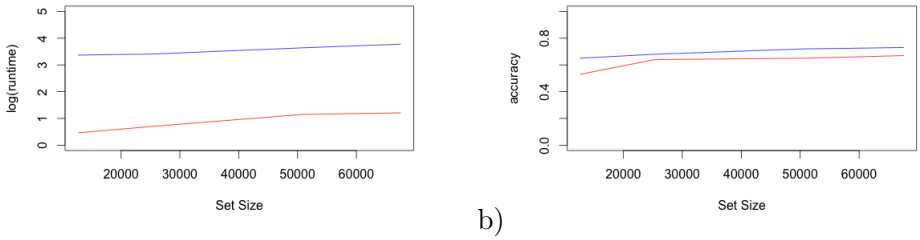
$$w_k* \leftarrow w_k * \tfrac{1}{n_k*} (z_t - w_k*) \tag{10}$$

4. Results

We use a connect-4 dataset, which is freely available on UCI machine learning repository that contains all legal 8-ply positions(nrows=65337, ncol=42) in the game of connect-4 in which neither player has won yet, and in which the next move is not forced. The outcome is a categorical variable with 3 values: "win", "lose" or "draw". Results are tabulated in table 1.
Also, Figure 2 a) depicts the comparison of running time of logistic regression (blue) v/s sgd-Logistic Regression (Red) on a logarithmic scale for different training set size. It is clearly evident that sgd-logreg is faster by atleast an order of 10. 2 b) is the comparison of accuracies for different set sizes, and for high dimension data, accuracy of sgd-logreg converges towards that of logistic regression.

	LREG	SGD-LREG	SVM	SGD-SVM	KMEANS	SGD-KM
Time	47.23	5.31	8178.65	27.46	2.24	0.19
Accuracy	73.68	65.36	81.72	74.13	–	–

Table 1: Comparison Table



a) b)

Figure 2: Blue: Simple Logistic Regression, Red: SGD- Logistic Regression

5. Conclusion

In this paper we looked at the application of Stochastic gradient descent algorithm in different models including Logistic regression and Support Vector Machine(SVM). We conclude that as the dataset size increases, the machine learning modelling techniques can be limited by the computational time that it take to find the optimum values of parameter and SGD performs well in this context. However, like every algorithm, SGD has its disadvantages too. SGD is sensitive to feature scaling. Also, while batch gradient converges to a minimum, SGD randomness allows it to new and a potentially better local optima however it may cause problems as it will keep oscillating and overshooting. However, many variants have been proposed to deal with this problem like decreasing the learning rate with time may shrink the overshooting part. Therefore, when applied properly, SGD can increase the computational efficiency with little or no change in accuracy.

6. References

1. Zhang, Tong. "Solving large scale linear prediction problems using stochastic gradient descent algorithms." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.
2. Bottou, Lon. "Stochastic gradient descent tricks." Neural Networks: Tricks of the Trade. Springer Berlin Heidelberg, 2012. 421-436.
3. Tran, Dustin, Panos Toulis, and Edoardo M. Airoldi. "Stochastic gradient descent methods for estimation with large data sets." arXiv preprint arXiv:1509.06459 (2015)
4. https://archive.ics.uci.edu/ml/datasets/Connect-4
5. R Libraries Used: `psych, e1071, sgd, RSofia, RevoScaleR`

Appendix

Code for logistic regression using GLM

```
library(e1071)

library(psych)

connect.4 <- read.csv("F:/Academics/8th Sem/EE609A-Convex/TermPaper/connect-4.data",
header=FALSE)

connect.4<-as.matrix(connect.4)

x<-connect.4[,-43]

classes<-connect.4[,43]

y=classes

#y<-ifelse(y=='win', 1, ifelse(y=='draw', 0, -1))

y<-as.vector(y)

factorData=matrix(ncol=ncol(x)*3, nrow=nrow(x))

factorData[is.na(factorData)]<-0

for(i in 1:dim(x)[1])  # for each row

{

 for(j in 1:dim(x)[2]) # for each column

 {

  if(x[i,j]=='b')

  {factorData[i,3*j] = 1 }

  else if(x[i,j]=='x')

  {

   factorData[i,3*j-1] =1

  }

  else if(x[i,j]=='o'){

   factorData[i,3*j-2]=1

  }
```

```
  }
}
#model=svm(x=factorData, y=y, cross=5, type='C-classification')

ptm <- proc.time()
ywin<-ifelse(y=='win', 1, 0)
d1<-as.data.frame(cbind(factorData,ywin))
m1 <- glm(ywin~.,data=d1, family="binomial")
summary(m1)
p1<-predict(m1, as.data.frame(factorData))
p1<-ifelse(p1<=0.5,0,1)

ydraw<-ifelse(y=='draw', 1, 0)
d2<-as.data.frame(cbind(factorData,ydraw))
m2 <- glm(ydraw~., data=d2, family="binomial")
summary(m2)
p2<-predict(m2, as.data.frame(factorData))
p2<-ifelse(p2<=0.5,0,1)

yloss<-ifelse(y=='loss', 1, 0)
d3<-as.data.frame(cbind(factorData,yloss))
m3 <- glm(yloss~., data=d3, family="binomial")
summary(m3)
p3<-predict(m3, as.data.frame(factorData))
p3<-ifelse(p3<=0.5,0,1)
```

```
bind<-cbind(p1, p2, p3)

classnames = c('win', 'draw', 'loss')

colnames(bind)<-classnames

a<-colnames(bind)[apply(bind,1,which.max)]

b<-cbind(bind,a)

b<-as.data.frame(b)

b<-as.matrix(table(b[,4],connect.4[,43]))


tr(b)/length(y)

proc.time() – ptm
```

Code for Logistic regression using SGD

```
library(e1071)

library(sgd)

library(psych)

connect.4 <- read.csv("F:/Academics/8th Sem/EE609A-Convex/TermPaper/connect-4.data",
header=FALSE)

connect.4<-as.matrix(connect.4)

x<-connect.4[,-43]

classes<-connect.4[,43]

y=classes

#y<-ifelse(y=='win', 1, ifelse(y=='draw', 0, -1))

y<-as.vector(y)

factorData=matrix(ncol=ncol(x)*3, nrow=nrow(x))

factorData[is.na(factorData)]<-0

for(i in 1:dim(x)[1])  # for each row

{

  for(j in 1:dim(x)[2]) # for each column
```

```
 {
   if(x[i,j]=='b')
   {factorData[i,3*j] = 1 }
   else if(x[i,j]=='x')
   {
     factorData[i,3*j-1] =1
   }
   else if(x[i,j]=='o'){
     factorData[i,3*j-2]=1
   }
 }
}
#model=svm(x=factorData, y=y, cross=5, type='C-classification')

ptm <- proc.time()
ywin<-ifelse(y=='win', 1, 0)
d1<-as.data.frame(cbind(factorData,ywin))
m1 <- sgd(ywin~.,data=d1, family="binomial", model="glm",model.control=binomial(link="logit"),
        sgd.control=list(reltol=1e-5, npasses=200),
        lr.control=c(scale=1, gamma=1, alpha=30, c=1))
summary(m1)
p1<-predict(m1, as.matrix(d1))

p1<-ifelse(p1<=0.5,0,1)
ydraw<-ifelse(y=='draw', 1, 0)
d2<-as.data.frame(cbind(factorData,ydraw))
m2 <- sgd(ydraw~., data=d2, family="binomial",model="glm",model.control=binomial(link="logit"),
```

```
        sgd.control=list(reltol=1e-5, npasses=200),
        lr.control=c(scale=1, gamma=1, alpha=30, c=1))
summary(m2)
p2<-predict(m2, as.matrix(d2))
p2<-ifelse(p2<=0.5,0,1)

yloss<-ifelse(y=='loss', 1, 0)
d3<-as.data.frame(cbind(factorData,yloss))


m3 <- sgd(yloss~., data=d3, family="binomial",model="glm",model.control=binomial(link="logit"),
        sgd.control=list(reltol=1e-5, npasses=200),
        lr.control=c(scale=1, gamma=1, alpha=30, c=1))
summary(m3)
p3<-predict(m3, as.matrix(d3))
p3<-ifelse(p3<=0.5,0,1)


bind<-cbind(p1, p2, p3)
classnames = c('win', 'draw', 'loss')
colnames(bind)<-classnames
a<-colnames(bind)[apply(bind,1,which.max)]
b<-cbind(bind,a)
b<-as.data.frame(b)
b<-as.matrix(table(b[,4],connect.4[,43]))

tr(b)/length(y)
```

```
proc.time() - ptm
```

Code for SVM

```
library(e1071)

connect.4 <- read.csv("F:/Academics/8th Sem/EE609A-Convex/TermPaper/connect-4.data",
header=FALSE)

connect.4<-as.matrix(connect.4)

x<-connect.4[,-43]

classes<-connect.4[,43]

y=classes

#y<-ifelse(y=='win', 1, ifelse(y=='draw', 0, -1))

y<-as.vector(y)

factorData=matrix(ncol=ncol(x)*3, nrow=nrow(x))

factorData[is.na(factorData)]<-0

for(i in 1:dim(x)[1])  # for each row

{

  for(j in 1:dim(x)[2]) # for each column

  {

    if(x[i,j]=='b')

    {factorData[i,3*j] = 1 }

    else if(x[i,j]=='x')

    {

      factorData[i,3*j-1] =1

    }

    else if(x[i,j]=='o'){

      factorData[i,3*j-2]=1

    }

  }

}
```

```
#model=svm(x=factorData, y=y, cross=5, type='C-classification')



ywin<-ifelse(y=='win', 1, 0)

m1 <- svm(x=factorData, y=ywin, cross=5, type='C-classification')

summary(m1)

p1<-predict(m1, factorData)



ydraw<-ifelse(y=='draw', 1, 0)

m2 <- svm(x=factorData, y=ydraw, cross=5, type='C-classification')

summary(m2)

p2<-predict(m2, factorData)

yloss<-ifelse(y=='loss', 1, 0)

m3<- svm(x=factorData, y=yloss, cross=5, type='C-classification')

summary(m3)

p3<-predict(m3, factorData)

bind<-cbind(p1, p2, p3)

classnames = c('win', 'draw', 'loss')

colnames(bind)<-classnames

a<-colnames(bind)[apply(bind,1,which.max)]

b<-cbind(bind,a)

b<-as.data.frame(b)

table(b[,4],connect.4[,43])
```

Code for SVM using SGD

```
library(e1071)

library(psych)

library(RSofia)

connect.4 <- read.csv("F:/Academics/8th Sem/EE609A-Convex/TermPaper/connect-4.data",
header=FALSE)

connect.4<-as.matrix(connect.4)

x<-connect.4[,-43]

classes<-connect.4[,43]

y=classes

#y<-ifelse(y=='win', 1, ifelse(y=='draw', 0, -1))

y<-as.vector(y)

factorData=matrix(ncol=ncol(x)*3, nrow=nrow(x))

factorData[is.na(factorData)]<-0

for(i in 1:dim(x)[1])  # for each row

{

 for(j in 1:dim(x)[2]) # for each column

 {

  if(x[i,j]=='b')

  {factorData[i,3*j] = 1 }

  else if(x[i,j]=='x')

  {

   factorData[i,3*j-1] =1

  }

  else if(x[i,j]=='o'){

   factorData[i,3*j-2]=1

  }

 }
```

```
}
#model=svm(x=factorData, y=y, cross=5, type='C-classification')


ptm <- proc.time()
ywin<-ifelse(y=='win', 1, 0)
d1<-as.data.frame(cbind(factorData,ywin))
m1 <- sofia(ywin~.,data=d1, learner_type = "sgd-svm")
summary(m1)
p1<-predict(m1, as.data.frame(factorData), prediction_type = 'logistic')
m<-range(p1)
q<-(m[2]+m[1])/2
p1<-ifelse(p1<=q,0,1)


ydraw<-ifelse(y=='draw', 1, 0)
d2<-as.data.frame(cbind(factorData,ydraw))
m2 <- sofia(ydraw~., data=d2, learner_type = "sgd-svm")
summary(m2)
p2<-predict(m2, as.data.frame(factorData), prediction_type='logistic' )
m<-range(p2)
q<-(m[2]+m[1])/2
p2<-ifelse(p2<=q,0,1)


yloss<-ifelse(y=='loss', 1, 0)
d3<-as.data.frame(cbind(factorData,yloss))
m3 <- sofia(yloss~., data=d3, learner_type = "sgd-svm")
summary(m3)
p3<-predict(m3, as.data.frame(factorData), prediction_type='logistic')
```

```
m<-range(p2)
q<-(m[2]+m[1])/2
p3<-ifelse(p3<=q,0,1)


bind<-cbind(p1, p2, p3)
classnames = c('win', 'draw', 'loss')
colnames(bind)<-classnames
a<-colnames(bind)[apply(bind,1,which.max)]
b<-cbind(bind,a)
b<-as.data.frame(b)
b<-as.matrix(table(b[,4],connect.4[,43]))

tr(b)/length(y)
proc.time() – ptm
```

Code for K means(both with and without SGD)

```
ptm <- proc.time()
fit1<-kmeans(factorData, algorithm = "MacQueen", centers=3)
proc.time() - ptm

ptm <- proc.time()
library(RevoScaleR)
y<-ifelse(y=='win', 1, ifelse(y=='draw', 0, -1))
d4<-as.data.frame(cbind(factorData,y))
fit2<-rkmeans(y~.,data=d4, centers=3)
proc.time() - ptm
```

```
#obtained by running the procedure multiple times

xhat <- c(67557, 50668, 25344, 12672)

yhat1<- c(43.67, 38.01, 30.20, 28.97)

yhat2 <- c(3.35, 3.15, 2.02, 1.59)


plot(xhat, y=log(yhat2),type="l",col="red", xlab='Set Size', ylab = 'log(runtime)', ylim = c(0,5))

lines(xhat, y=log(yhat1),type="l",col="blue")

y1<- c(0.73, 0.72, 0.68, 0.65)

y2<- c(0.67, 0.65, 0.64, 0.53)

plot(xhat, y=y2,type="l",col="red", xlab='Set Size', ylab = 'accuracy', ylim = c(0,1))

lines(xhat, y=y1,type="l",col="blue")
```