Rcane-A Statistical R package to calculate parameter estimates of Linear Regression

December 10, 2017

Abstract

This document provides an introduction to the statistical package Rcane which calculates parameter estimates of Linear Regression.

Authors: Siddhesh Acharekar | Shivayogi Biradar | Akshay Suresh | Hsiangwei Chao

1 Summary

Linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables). Numeric optimization is one of the approaches to obtain the coefficients of linear regression for which the residual sum of squares error is minimum. This package offers users a set of commonly used numeric optimization algorithms for parameter estimation in linear regression. Each algorithm will perform differently on different types of data enabling users to choose the algorithm as per their performance and computation standards. Rcane performs parameter estimation using the following 4 algorithms - Batch Gradient Descent, Mini-batch Gradient Descent, Stochastic Gradient Descent, and Coordinate Descent.

2 Methods

2.1 Introduction to Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. Gradient Descent is based on the observation that if the multi-variable function is defined and differentiable in the neighborhood of a point, the function decreases fastest if one goes from the point in the direction of the negative gradient. This implies that for a point a defined in $f(X)$, $f(X)$ would decrease fastest if it goes in the direction of $- \bigtriangledown f(a)$. This can be illustrated as follows:

$$a_{n+1} \leftarrow a_n - \alpha \cdot \nabla f(a)$$

$f(a)$ could be any convex function. In case of Gradient Descent in Linear models, $f(a)$ is a convex loss function (residual sum of squares). α is called the learning rate. An important concept to learn while understanding gradient descent is learning rate. Learning rate is used in gradient descent algorithms to determine how quickly the parameters converge to the local or global minima.

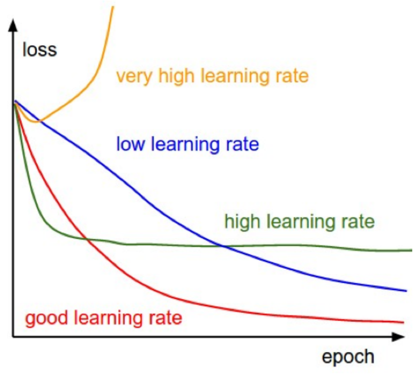


Figure 1: Loss vs. Epoch for different learning rates

Learning rate plays an important role in gradient descent algorithms. This is because if the learning rate is set at a very high value, the algorithm would make drastic updates in the parameter estimates. This would cause the algorithm to overshoot the parameter estimates in each iteration and never converge. On the other hand, if the learning rate is set a very low value, the algorithm would take longer time to converge. This is illustrated in Figure 1. Rcane performs parameter estimation using the following 4 algorithms - Batch Gradient Descent, Mini-batch Gradient Descent, Stochastic Gradient Descent, and Coordinate Descent.

2.2 Batch Gradient Descent

Batch Gradient Descent computes the gradient of the cost function with respect to the parameters θ for the entire training dataset. The algorithm for Batch Gradient Descent can be given as follows:

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta)$$

2.2.1 Batch Gradient Descent With Bold Driver

Depending on the choice of cost function, Gradient Descent faces different problems. Having sum of squared errors as a cost function causes the value of $\nabla_\theta J(\theta)$ to get larger and larger as the size of the training dataset increases. Intuitively, an adaptive α can improve the algorithm in this case. The idea behind this is that the farther the parameter are from optimal values, the faster the cost function value should move towards the minimum and thus α should be larger. Similarly, the closer the cost function value gets to the minimum the smaller α should be.

Rcane applies the Bold Driver method to attain an adaptive α. The approach used by Bold Driver is - after each iteration, the current loss function value ($J_i(\theta)$) is compared to the previous loss function value ($J_{i-1}(\theta)$). If the cost function has decreased, α is increased by

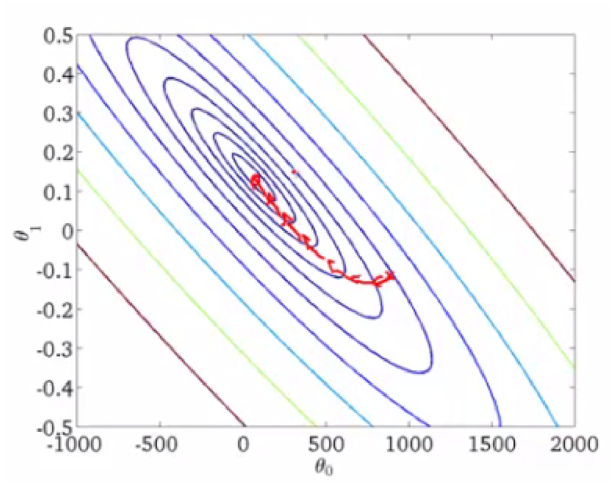


Figure 2: Convergence using Batch Gradient Descent

10%. If the cost function has increased, α is decreased by 50%. The biggest advantage this offers is that the algorithm never overshoots the local minimum value of the cost function and converges in most cases.

```
coef(BatchGradientDescent(x, y, alpha=10))
```

```
## (Intercept)          x1
##         NaN         NaN
```

```
coef(BatchGradientDescent(x, y, alpha=10, boldDriver = TRUE))
```

```
## (Intercept)          x1
##    3.000008    1.999922
```

2.3 Stochastic Gradient Descent

Stochastic gradient descent, in contrast, performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$. The algorithm for stochastic gradient descent can be given as follows:

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

2.3.1 Stochastic Gradient Descent With AdaGrad

The effort to address data redundancy in Stochastic Gradient Descent leads to a new problem for most real datasets that no optimization algorithm library addresses. Datasets tend to be sparse and frequency of features differs often, so applying the same learning rate to every parameter update does not lead to attaining an optimal minumum. So as to account for the rarely occuring features, we decide not to update all parameters to the same extent but perform a large update for rarely occuring features and vice versa.

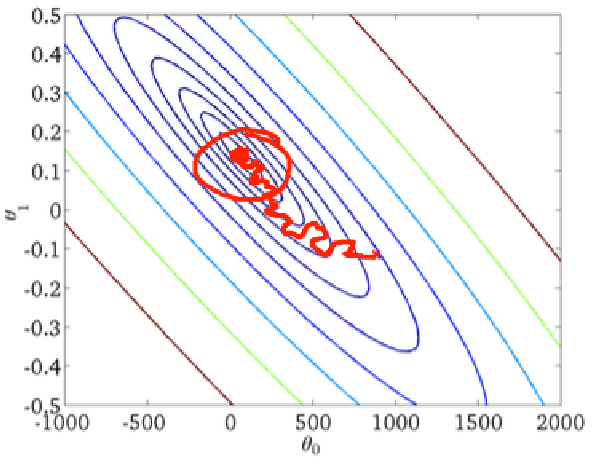


Figure 3: Convergence using Stochastic Gradient Descent

This is the idea behind AdaGrad. In its update rule, AdaGrad modifies α at each iteration t for every parameter θ_i based on the previous gradients that have been computed:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Here $G_{t,ii}$ is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t θ_i up to t while ϵ is a smoothing term that avoids division by zero. In Rcane $\epsilon = 10^{-8}$. Another observed benefit of AdaGrad is it eliminates the need to manually tune α. Rcane uses a default value of *0.1* and lets the algorithm handle the rest.

```
coef(StochasticGradientDescent(x, y, alpha=10000))
```

```
## (Intercept)          x1
##         NaN         NaN
```

```
coef(StochasticGradientDescent(x, y, alpha=10000, AdaGrad=TRUE))
```

```
## (Intercept)          x1
##           3           2
```

2.4 Mini-Batch Gradient Descent

Mini-batch Gradient Descent finally takes the best of both worlds and performs an update for a certain batch of n training examples:

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

This way, it a) reduces the variance of the parameter updates, which can lead to more stable convergence; and b) can make use of highly optimized matrix operations common to state-of-the-art deep learning libraries that make computing the gradient with respect to a mini-batch very efficient. Common mini-batch sizes range between 50 and 256, but can vary for different applications.

2.5 Coordinate Descent

Coordinate descent is an optimization algorithm that successively minimizes along coordinate directions to find the minimum of a function. At each iteration, the algorithm determines a coordinate or coordinate block via a coordinate selection rule, then exactly or inexactly minimizes over the corresponding coordinate hyperplane while fixing all other coordinates or coordinate blocks. A line search along the coordinate direction can be performed at the current iteration to determine the appropriate step size.

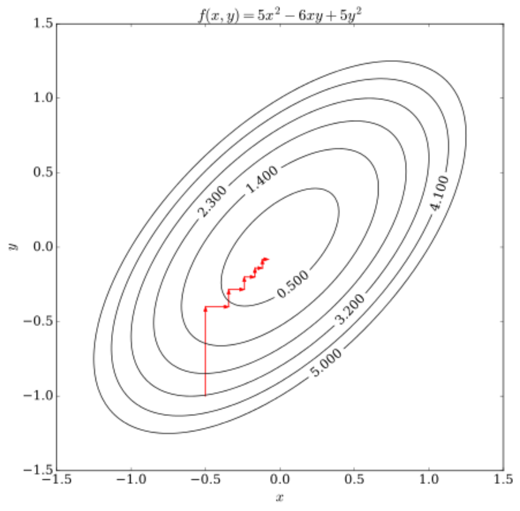


Figure 4: Convergence using Coordinate Descent

2.6 Package installation and usage

This package is open source and available on GitHub (https://github.com/sureshaks/rcane).

Package installation

```
install_github("sureshaks/rcane", build_vignettes=TRUE)
```

Package usage

```
library(rcane)
```

```
?rlm
```

rlm() with Batch Gradient Descent can be used using the following command:

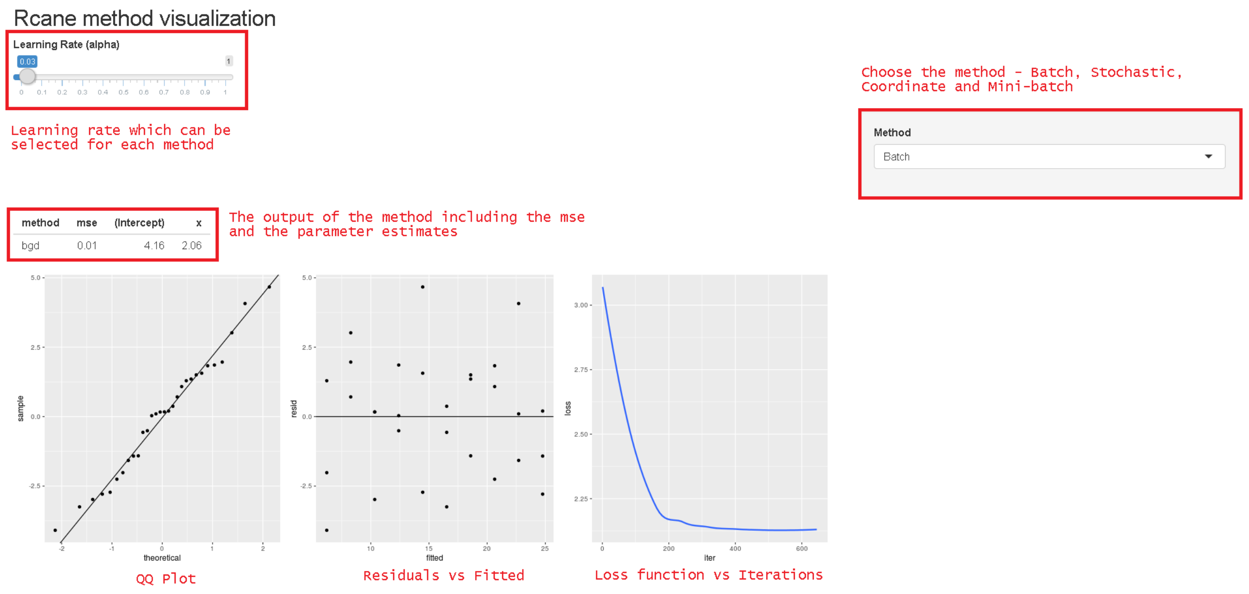
```
# batch gradient descent
rlm(formula = y ~ x, data = mydata, method = "bgd")
```

More information on package functions and usage can be found in the Vignette.

```r
vignette("rcane", package="rcane")
```

2.7 Rcane Shiny App

As seen in the previous sections, these algorithms may return different parameter estimates for different datasets. The Rcane Shiny App makes it convenient for a user to select the appropriate method and the parameters for the dataset and observe the performance of the method. An illustration of the Shiny App is given below:



Rcane method visualization

Learning Rate (alpha): 0.03

Method: Batch

method	mse	(Intercept)	x
bgd	0.01	4.16	2.06

Learning rate which can be selected for each method
Choose the method - Batch, Stochastic, Coordinate and Mini-batch
The output of the method including the mse and the parameter estimates
QQ Plot
Residuals vs Fitted
Loss function vs Iterations
sample
theoretical
resid
fitted
loss
iter

Figure 5: Rcane Shiny App

The shiny app is available on https://siddheshacharekar.shinyapps.io/rcane_shiny/. The source code is available on https://github.com/SiddheshAcharekar/rcane_shiny.

3 Results

The performance of the package in terms of accuracy is evaluated on a real dataset. This dataset contains a subset of the fuel economy data that the EPA makes available on http://fueleconomy.gov. It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

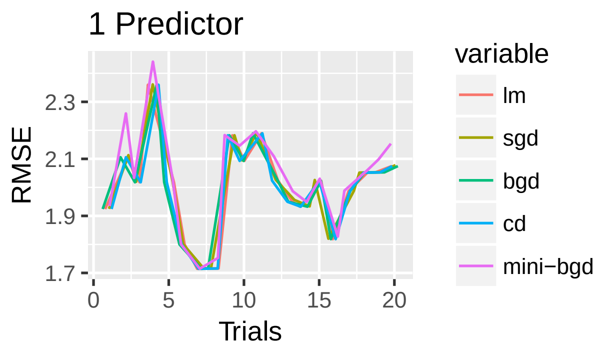
A linear model is fitted using the rlm() method to predict highway miles per gallon given the city miles per gallon. The comparison of different models can be given as follows:

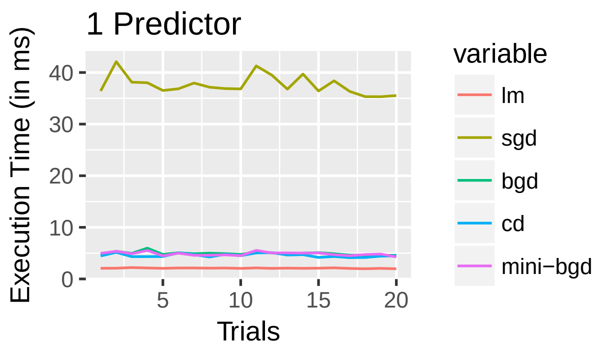
```
##                              methods Intercept     displ
## 1         Standard lm() implementation  35.69765 -3.530589
## 2               Batch Gradient Descent  35.69728 -3.530495
## 3  Batch Gradient Descent with Bold Driver  35.69764 -3.530587
## 4              Stochastic Gradient Descent  35.81298 -3.581527
## 5 Stochastic Gradient Descent with AdaGrad  34.62920 -2.656946
## 6               Minibatch Gradient Descent  35.69765 -3.530589
## 7                       Coordinate Descent  35.69765 -3.530589
```

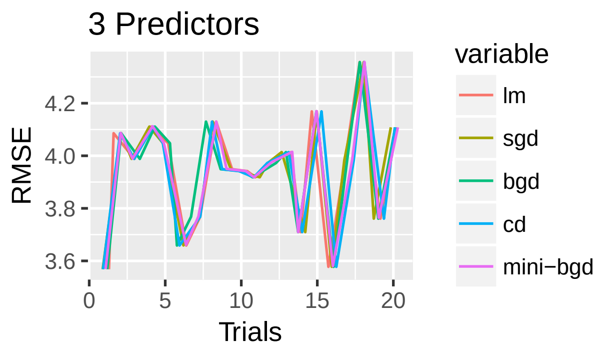
The performance in terms of accuracy and execution time is evaluated on 3 different kinds of datasets:

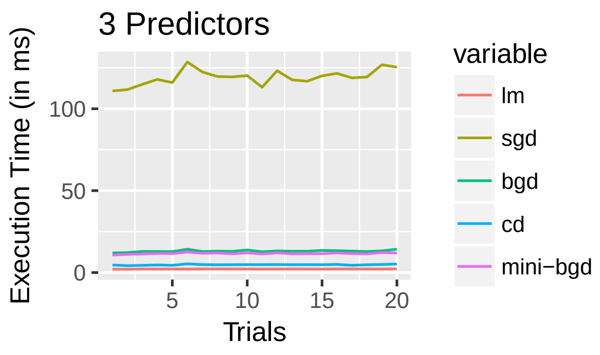
- 1 predictor, 1 response variable
- 3 predictors, 1 response variable
- 3 predictors (out of which 2 are highly correlated), 1 response variable

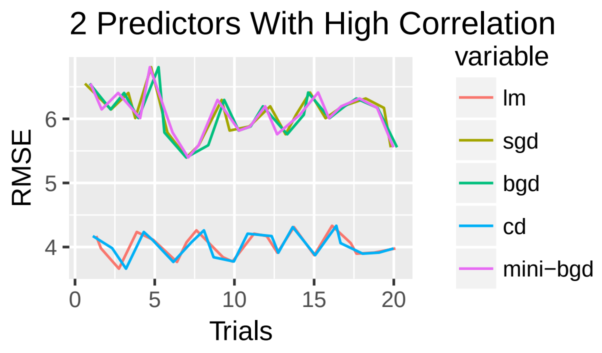
The results can be illustrated below:

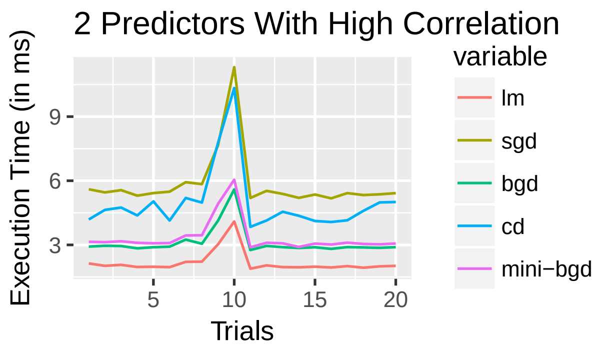












4 Discussion

The key learnings from the project are as follows:

- The algorithms perform differently on different learning rates and different kinds of data. From the graphs above, it can be seen that the 4 algorithms perform similarly in terms of accuracy but Coordinate Descent outperforms other gradient descent aalgorithms when the predictors are highly correlated. In terms of execution time, Stochastic Gradient Descent is slower in comparision to other algorithms.
- Choosing the best learning rate is challenging - a high learning rate may overshoot and a low learning rate would make the algorithms very slow taking a lot of time to converge.
- Stochastic Gradient Descent may not always converge. This is due to the fact that Stochastic Gradient Descent updates the parameters for each record which makes it more susceptible to random noise in the data.
- Mini-batch gradient descent involves choosing the correct batch size along with learning rate.

Rcane can be improved with the following enhancements:

- Performance improvements with C++ implementations.
- Implementation of Logistic Regression using Gradient Descent Algorithms.
- Implementation of Regression models with statistical regularization.
- Integration of Gradient Descent Algorithms with neural networks.

5 Statement Of Contribution

Siddhesh Acharekar - Assisted in topic selection for the project

- Assisted in the development of project proposal
- Implemented Mini-batch gradient descent
- Implemented the Shiny App
- Assisted in the development of the presentation
- Assisted in the development of the report

Shivayogi Biradar - Assisted in topic selection for the project

- Assisted in the development of project proposal
- Implemented Coordinate descent
- Performed profiling and benchmarks of the 4 algorithms
- Assisted in the development of the presentation

- Assisted in the development of the report

Hsiangwei Chao - Assisted in topic selection for the project

- Assisted in the development of project proposal
- Implemented standard Stochastic descent along with Stochastic Gradient Descent with AdaGrad
- Performed profiling and benchmarks of the 4 algorithms
- Wrote Unit tests for testing the functionality of the algorithms
- Assisted in the development of the presentation
- Assisted in the development of the report

Akshay Suresh - Assisted in topic selection for the project

- Assisted in the development of project proposal
- Set up the GitHub repository and integrated Travis CI for continuous integration
- Implemented standard Batch Gradient Descent along with Batch Gradient Descent with BoldDriver
- Developed the vignettes for the package
- Assisted in the development of the presentation
- Assisted in the development of the report

6 Appendix

Rcane - https://github.com/sureshaks/rcane

Rcane Shiny - https://github.com/SiddheshAcharekar/rcane_shiny