LM using gradient descent

Rishabh Vaish

Linear Regression through Optimization

Linear regression is most popular statistical model, and the core technique for solving a linear regression is simply inverting a matrix:

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T y$$

However, lets consider alternative approaches to solve linear regression through optimization. We use a gradient descent approach. We know that $\hat{\beta}$ can also be expressed as

$$\hat{\beta} = argmin\ l(\beta) = argmin\ \frac{1}{2n} \sum_{i}^{2n} \left(y_i - x_i^T \beta\right) x_i$$

To perform the optimization, we will first set an initial beta value, say $\beta = 0$ for all entries, then proceed with the updating

$$\beta^{new} = \beta^{old} + \frac{\partial l(\beta)}{\partial \beta} \times \delta$$

where δ is some small constant, say 0.1. We will keep updating the beta values by setting β^{new} as the old value and calcuting a new one untill the difference between β^{new} and β^{old} is less than a prespecified threshold ϵ, e.g., $\epsilon = 10^{-6}$. You should also set a maximum number of iterations to prevent excessively long runing time.

R function mylm_g(x, y, delta, epsilon, maxitr) to implement this optimization version of linear regression. The output of this function is a vector of the estimated beta value

Created a mylm_g function in R which takes (x, y, delta, epsilon, maxitr) as inputs in matrix form of following dimensions - x is [n,p] y is [n,1]

The function implements gradient descent approach for optimisation and outputs the beta estimates in matrix of [1,p]

```
mylm_g <- function(x, y, delta, epsilon, maxitr){
  #beta matrix
  beta <- matrix(0,1,ncol(x))
  for(i in 1:maxitr){
    #gradient
    gradient <- (-1/nrow(x))*colSums((y - beta%*%t(x))%*%x)
    #calculating new beta
    beta_new <- beta - gradient*delta
    #checking distance between new and old beta
    beta_dist <- dist(rbind(beta, beta_new))
    #checking for threshold
    if (beta_dist < epsilon){
```

```
      break
    }
    #updating the beta values
    beta <- beta_new
  }
  return(beta)
}
```

Data implementation

Testing this function on the Boston Housing data from the `mlbench` package. Documentation is provided here if you need a description of the data. We will remove `medv`, `town` and `tract` from the data and use `cmedv` as the outcome. We will use a scaled and centered version of the data for estimation. Please also note that in this case, you do not need the intercept term. And you should compare your result to the `lm()` function on the same data. Experiment on different `maxitr` values to obtain a good solution. However your function should not run more than a few seconds.

```
library(mlbench)
data(BostonHousing2)
X = BostonHousing2[, !(colnames(BostonHousing2) %in% c("medv", "town", "tract", "cmedv"))]
X = data.matrix(X)
X = scale(X)
Y = as.vector(scale(BostonHousing2$cmedv))
```

Estimating beta using the function created above and comparing it with the beta values generated by using lm function. The values are similar. The difference between the 2 beta estimates is shown

```
# Setting the parameters
delta = 0.1
epsilon = 10^-7
maxitr = 5000

#Estimating beta using the function
beta <- mylm_g(X,Y,delta,epsilon,maxitr)

#creating dataframe and fitting lm
df <- data.frame(cbind(X,Y))
linear_model = lm(Y ~ lon+lat+crim+zn+indus+chas+nox+rm+age+dis+rad+tax+ptratio+b+lstat, data = df)
beta_lm <- matrix(linear_model$coefficients[2:length(linear_model$coefficients)], 1, ncol(X))

#comparing the 2 beta values
beta_compare <- cbind(t(beta),t(beta_lm),t(abs(beta-beta_lm)))
colnames(beta_compare) <- c("Beta", "Beta_LM", "Error")
beta_compare
```

```
##               Beta      Beta_LM        Error
##  [1,] -0.032316756 -0.032316441 3.147665e-07
##  [2,]  0.030244764  0.030245087 3.228742e-07
##  [3,] -0.097935206 -0.097935969 7.635663e-07
##  [4,]  0.118271740  0.118273098 1.357273e-06
##  [5,]  0.011386300  0.011390378 4.078428e-06
##  [6,]  0.071312754  0.071312253 5.001286e-07
```

```
##  [7,] -0.199702924 -0.199703772 8.477011e-07
##  [8,]  0.287233466  0.287232811 6.555650e-07
##  [9,]  0.007564405  0.007564852 4.470164e-07
## [10,] -0.321039510 -0.321039342 1.676267e-07
## [11,]  0.290840609  0.290850755 1.014614e-05
## [12,] -0.236514862 -0.236526155 1.129274e-05
## [13,] -0.206804452 -0.206804965 5.135301e-07
## [14,]  0.091235358  0.091235409 5.052057e-08
## [15,] -0.417972446 -0.417972819 3.727051e-07
```