HW 3 Gradient Descent, Regularization, and SGD

2025-03-15

Contents

Gradient Descent

In assignment 3, we implement gradient descent, both in its standard batch form and with L2 regularization (Ridge Regression). We also cover stochastic gradient descent (SGD), which is often used when dealing with large datasets.

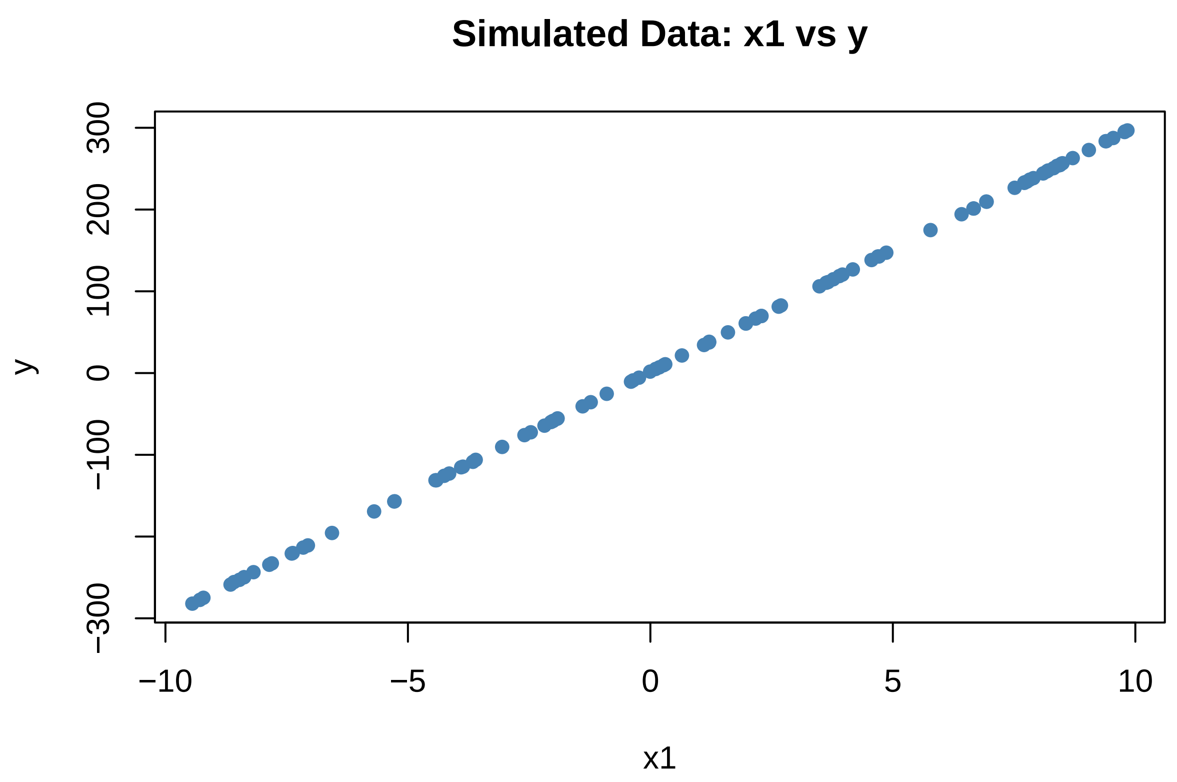
First step is simulating the data.

```r
#Code from the machine learning class

set.seed(112)

n <- 100
x1 <- runif(n, -1, 1) * 10
x0 <- rep(1, n)
x <- as.matrix(cbind(x0, x1))
y <- as.matrix(x1 * 30 + runif(n, 1, 2))
m <- nrow(y)

plot(x1, y, main = "Simulated Data: x1 vs y", xlab = "x1", ylab = "y", col = "steelblue", pch = 16)
```



Next, we fit a linear model using the `lm` function in R

```
lm(y ~ x1)
```

```
##
## Call:
## lm(formula = y ~ x1)
##
## Coefficients:
## (Intercept)           x1
##        1.51        30.00
```

```
summary(lm(y ~ x[,2]))
```

```
##
## Call:
## lm(formula = y ~ x[, 2])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.51178 -0.21237 -0.00798  0.22105  0.49706
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.509666   0.027661   54.58   <2e-16 ***
## x[, 2]      30.004966   0.004657 6443.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2725 on 98 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 4.152e+07 on 1 and 98 DF,  p-value: < 2.2e-16
```

```r
solve(t(x) %*% x) %*% t(x) %*% y
```

```
##          [,1]
## x0  1.509666
## x1 30.004966
```

Batch Gradient Descent (No Regularization)

Batch Gradient Descent is an optimization technique used to find the parameters (theta) of a linear model by iteratively updating them. At each iteration, it calculates the gradient of the cost function using all training examples (hence the term batch). This is computationally efficient for small datasets and ensures stable convergence.

We implement batch gradient descent by defining a function to compute the gradient and a loop to update the parameters using this gradient.

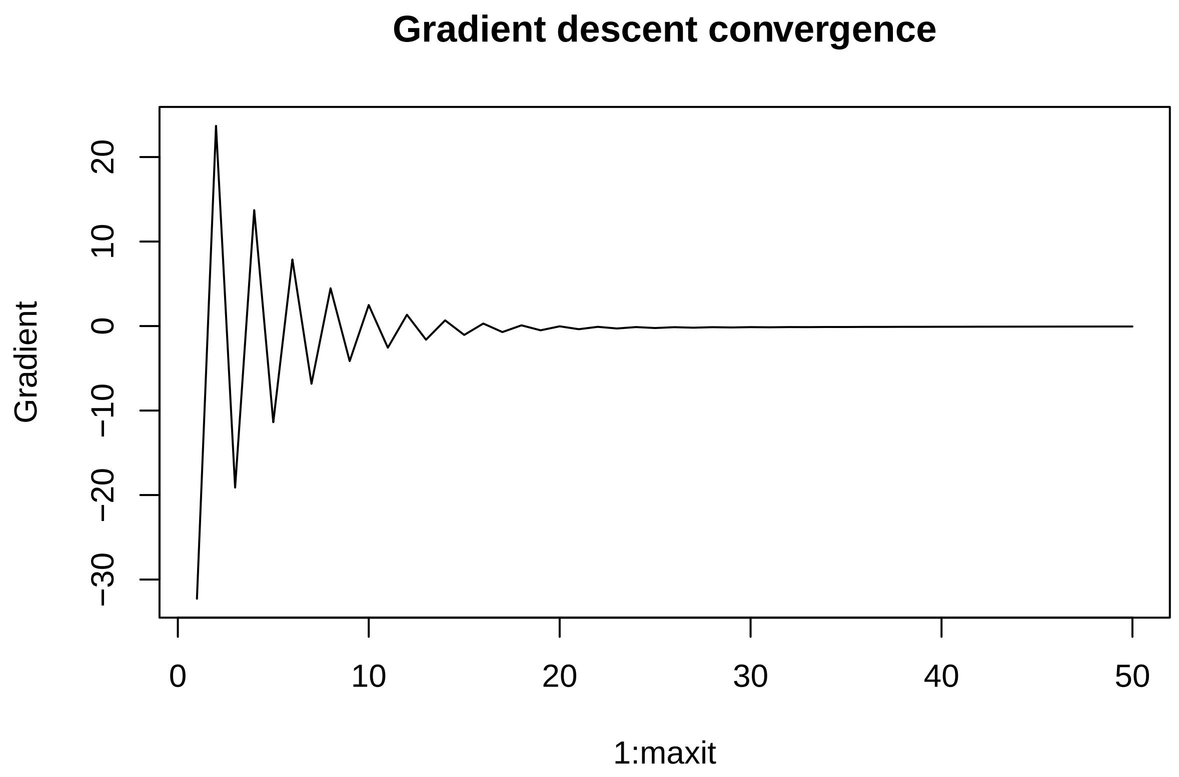
```r
## Gradient function
Gradient = function(x, y, theta) {
  m = nrow(y)
  gradient = (1 / m) * (t(x) %*% ((x %*% t(theta)) - y))
  return(t(gradient))
}

## Gradient Descent Update
Gradient_Descent_Update = function(x, y, maxit = 50, Alpha = 0.05) {
  theta = matrix(c(0, 0), nrow = 1)
  GRADVALS = matrix(0, nrow = 2, ncol = maxit)

  for (i in 1:maxit) {
    GRADVALS[, i] = Gradient(x, y, theta)
    theta = theta - Alpha * Gradient(x, y, theta)
  }

  plot(1:maxit, GRADVALS[1, ], type = 'l', main = "Gradient descent convergence", ylab = "Gradient")
  return(theta)
}

## Run GD
Gradient_Descent_Update(x, y)
```



```
##             x0       x1
## [1,] 1.458565 30.00644
```

Batch Gradient Descent with Ridge Regularization (lambda = 0.1)

We then incorporate L2 regularization (Ridge regression) into the gradient descent framework. Regularization is a technique used to prevent overfitting, especially when dealing with highly correlated predictors or small datasets. One common regularization method is Ridge Regression, which adds a penalty term to the cost function to shrink coefficients.

Below, we modify our gradient descent function to include a regularization term (L2 penalty) controlled by a hyperparameter lambda. regularization effect.

```r
## Gradient with L2 regularization (ridge)
Gradient_L2 = function(x, y, theta, lambda) {
  m = nrow(y)
  reg_term = lambda * t(theta)
  gradient = (1 / m) * (t(x) %*% ((x %*% t(theta)) - y)) + reg_term
  return(t(gradient))
}

## Gradient descent with ridge regularization
Gradient_Descent_Ridge = function(x, y, lambda = 0.1, maxit = 50, Alpha = 0.05) {
  theta = matrix(c(0, 0), nrow = 1)
  for (i in 1:maxit) {
    theta = theta - Alpha * Gradient_L2(x, y, theta, lambda)
  }
  return(theta)
}

## Run ridge descent
Gradient_Descent_Ridge(x, y, lambda = 0.1)
```

```
##             x0      x1
## [1,] 1.414241 29.9229
```

Advantages and Disadvantages of Regularization

Advantages of Regularization (e.g., Ridge):

- Prevents overfitting by penalizing large coefficients.
- Especially helpful when predictors are highly correlated (multicollinearity).
- Improves generalization on unseen data.

Disadvantages:

- Introduces bias in parameter estimates.
- Choosing the regularization parameter (lambda) can be tricky.
- May underfit if lambda is too high.

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a variation of gradient descent where the model updates the parameters using only one randomly selected training sample at a time.

This leads to faster but noisier convergence. While SGD may require more epochs to reach optimal solutions, it is highly scalable and well-suited for large datasets. The implementation here demonstrates that even with limited data, SGD can still recover reasonable parameter estimates.

```r
## Stochastic Gradient Descent
SGD = function(x, y, Alpha = 0.01, maxit = 100) {
  m = nrow(y)
  theta = matrix(c(0, 0), nrow = 1)

  for (i in 1:maxit) {
    for (j in 1:m) {
      idx = sample(1:m, 1)  ## Randomly pick one data point
      x_j = matrix(x[idx, ], nrow = 1)
      y_j = y[idx, , drop = FALSE]
      gradient = (t(x_j) %*% ((x_j %*% t(theta)) - y_j))
      theta = theta - Alpha * t(gradient)
    }
  }
  return(theta)
}
```

```r
## Run SGD
SGD(x, y)
```

```
##          [,1]     [,2]
## [1,] 1.49882 30.03505
```