Linear regression by gradient descent

July 26, 2012
By Christopher Bare

Share

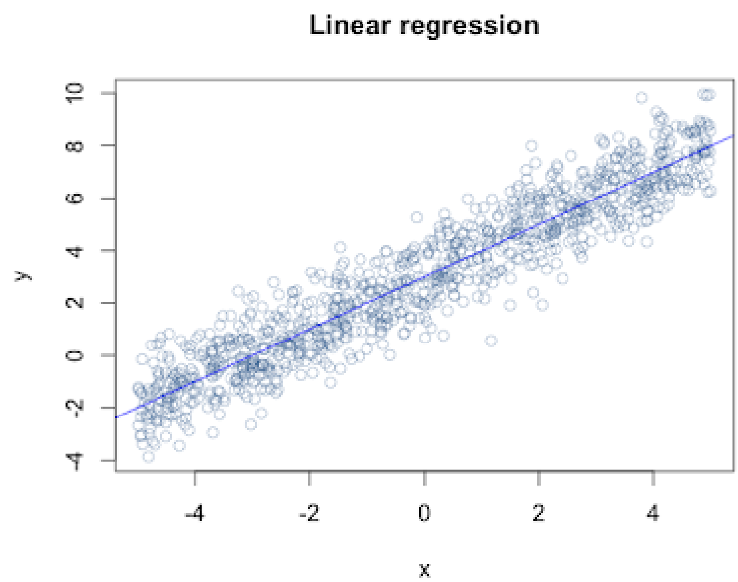
In Andrew Ng's Machine Learning class, the first section demonstrates gradient descent by using it on a familiar problem, that of fitting a linear function to data.

Let's start off, by generating some bogus data with known characteristics. Let's make y just a noisy version of x. Let's also add 3 to give the intercept term something to do.

```
# generate random data in which y is a noisy function of x
x <- runif(1000, -5, 5)
y <- x + rnorm(1000) + 3

# fit a linear model
res <- lm( y ~ x )
print(res)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     2.9930       0.9981
```

Fitting a linear model, we should get a slope of 1 and an intercept of 3. Sure enough, we get pretty close. Let's plot it and see how it looks.

```
# plot the data and the model
plot(x,y, col=rgb(0.2,0.4,0.6,0.4), main='Linear regression by gradient descent')
abline(res, col='blue')
```



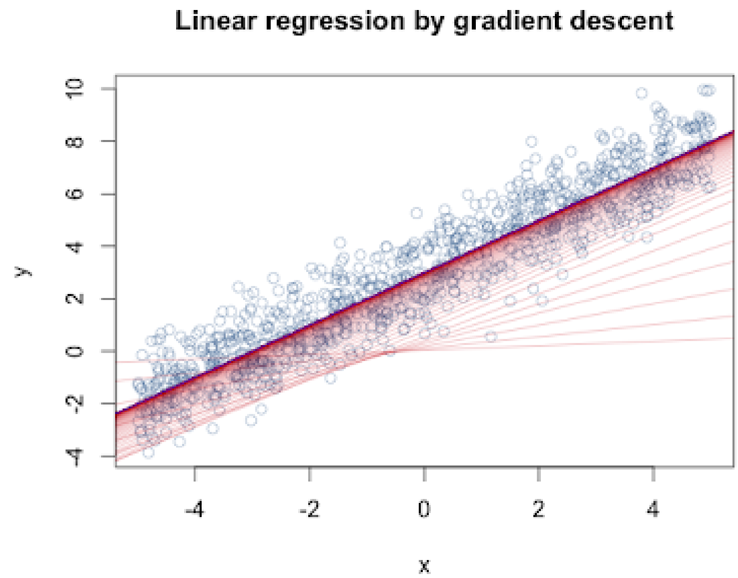
As a learning exercise, we'll do the same thing using gradient descent. As discussed previously, the main idea is to take the partial derivative of the cost function with respect to theta. That gradient, multiplied by a learning rate, becomes the update rule for the estimated values of the parameters. Iterate and things should converge nicely.

```
# squared error cost function
cost <- function(X, y, theta) {
  sum( (X %*% theta - y)^2 ) / (2*length(y))
}

# learning rate and iteration limit
alpha <- 0.01
num_iters <- 1000

# keep history
cost_history <- double(num_iters)
theta_history <- list(num_iters)

# initialize coefficients
theta <- matrix(c(0,0), nrow=2)

# add a column of 1's for the intercept coefficient
X <- cbind(1, matrix(x))

# gradient descent
for (i in 1:num_iters) {
  error <- (X %*% theta - y)
  delta <- t(X) %*% error / length(y)
  theta <- theta - alpha * delta
  cost_history[i] <- cost(X, y, theta)
  theta_history[[i]] <- theta
}

print(theta)

          [,1]
[1,] 2.9928978
[2,] 0.9981226
```

As expected, theta winds up with the same values as **lm** returned. Let's do some more plotting:

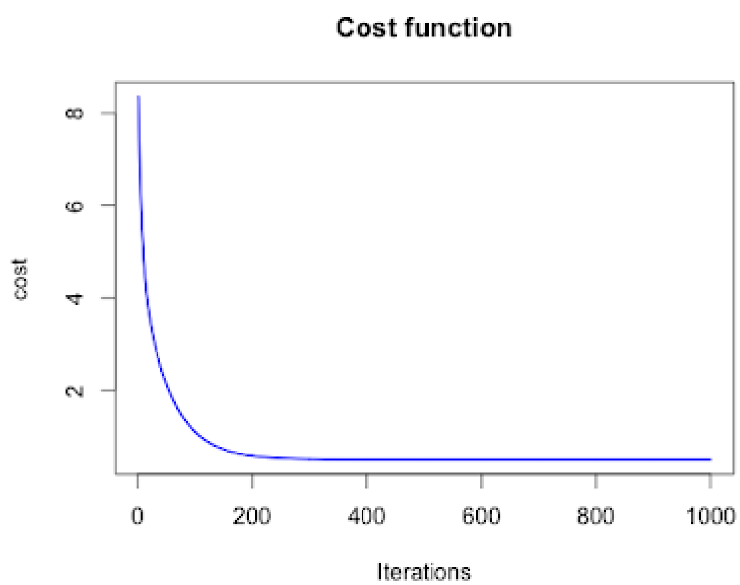
```
# plot data and converging fit
plot(x,y, col=rgb(0.2,0.4,0.6,0.4), main='Linear regression by gradient descent')
```

```
for (i in c(1,3,6,10,14,seq(20,num_iters,by=10))) {
  abline(coef=theta_history[[i]], col=rgb(0.8,0,0,0.3))
}
abline(coef=theta, col='blue')
```



Taking a look at how quickly the cost decreases, I might have done with fewer iterations.

```
plot(cost_history, type='line', col='blue', lwd=2, main='Cost function', ylab='cost', xlab='Iterations')
```



That was easy enough. The next step is to look into some of the more advanced [optimization methods]() available within R. I'll try to translate more of the [Machine Learning class]() into R. I know [others]() are doing that as well.