Lab 9

Statistical Computing, 36-350

Friday November 6, 2015

Agenda: Obtaining a good linear prediction model (Ridge Regression)

Part 1 (Warmup)

Recall that in linear regression we seek to find the linear relationship between a response variable and one or more explanatory variables. Say we have n data points $(y_i, X_i)_{i=1,\cdots,n}$ where y_i are scalar valued responses and $X_i = (X_{i1}\ X_{i2}\ \ldots\ X_{ip})$ are vector valued explanatory variables. We want to find the best vector of model coefficient β for:

$$y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip} + \mathcal{N}(0, \sigma^2)$$

The least squares solution minimizes the squared errors in the data, as you've already seen in lab 6 as a minimization over a 2D 'grid' of possible coefficient values of (β_0, β_1). We'll continue with this idea, but instead use gradient descent for the minimization.

First, load the file `ridgeregdat.Rdata` from the course website, which contains the two variables y and X which are the data you are to use for our problem.

```
load(url("http://www.stat.cmu.edu/~ryantibs/statcomp/labs/ridgeregdat.Rdata"))
```

Throughout, show all numerical results at 3 significant digits, using `signif()`.

1. Write the function, `squared.loss`, which takes the n-lengthed vector `beta` – the vector $\beta = (\beta_1\ \cdots\ \beta_p)$ – as an argument, and calculates the squared loss

$$\mathcal{L}_{\text{linear}}(\beta) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}))^2$$

   You may wish to refer to your work from the most recent homework, or lab 6.

2. Find a best-fit value for β by minimizing `squared.loss`, using `grad.descent()` from lecture, reproduced here. Use starting value of all zeros, and step size of 0.001 and maximum iterations of 200.

```
library(numDeriv) # to use the grad() function; make sure this is installed!

grad.descent = function(f, x0, max.iter=200, step.size=0.05,
  stopping.deriv=0.01, ...) {

  n = length(x0)
  xmat = matrix(0,nrow=n,ncol=max.iter)
  xmat[,1] = x0

  for (k in 2:max.iter) {
    # Calculate the gradient
    grad.cur = grad(f,xmat[,k-1],...)

    # Should we stop?
```

```
    if (all(abs(grad.cur) < stopping.deriv)) {
      k = k-1; break
    }

    # Move in the opposite direction of the grad
    xmat[,k] = xmat[,k-1] - step.size * grad.cur
  }

  xmat = xmat[,1:k] # Trim
  return(list(x=xmat[,k], xmat=xmat, k=k))
}
```

3. Run `lm` on the data and compare the results to what you found in 2. Do they agree with each other? (By the way, congratulations! Now you know 4 ways to fit a linear regression!)

Part 2 (Ridge Regression)

"Ridge regression" is a variant of the linear regression used in part 1. It also minimizes the squared errors (**notice!** this is a function of $\beta = (\beta_0 \ \cdots \ \beta_p)$, not y or X!), but with a penalty on $\sum_{i=1}^{p} \beta_j^2$

$$\mathcal{L}_{\text{ridge}}(\beta, \lambda) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where λ is an additional parameter in the model. Ridge regression is especially useful when you have an underdetermined system (there are more coefficients to estimate than there are datapoints), and when your explanatory variables are similar/duplicates or are highly correlated. We will now perform ridge regression on our data, with a fixed $\lambda = 0.1$.

4. Write the ridge regression objective function `ridge.loss()` as a function of the parameter of interest $\beta = (\beta_0, \cdots, \beta_p)$, and the penalty λ. The inputs to this function should be the vector `beta` (no default) and the scalar `lambda` (with default value of 0.01). You should be able to reuse some of your work from problem 1.

5. Evaluate `ridge.loss()` on the following β vectors. Which gives the smallest error? Why is searching in this manner is inefficient?

```
beta_try1 = c(rep(0.1, 29), 0.1)  # check that the returned value is 68523.59
beta_try2 = c(rep(0.1, 29), 0.11)
beta_try3 = c(rep(0.1, 29), 0.12)
beta_try4 = c(rep(0.1, 29), 0.13)
# you could continue this search for a long time..
```

6. Use `grad.descent()` to minimize `ridge.loss()`. Use all zeros as your starting value for β, a step size 0.001, and 10,000 maximum iterations.

7. What are the solutions for $\lambda = 1$? what about $\lambda = 3$? $\lambda = 10$ (use the same optimization settings)? You may find it useful to define a single-input function like this that overwrites (fixes) the `lambda` input to `ridge.loss()`.

```
ridge.loss1 = function(beta){return(ridge.loss(beta, lambda=1))}
```

Alternatively, you might just use the `...` argument in `grad.descent()`.

You should see your solutions "shrinking" toward zero as λ increases. From the ridge regression objective, can you explain why?. Plot the solutions as points along their place in the vector (first entry at x=1, second entry at x=2, and so forth). Use red for $\lambda = 0.1$ and blue for $\lambda = 1, 3, 10$ but with different shapes, using `pch`. Also plot `c(0,rep(c(3.7,-6.3,8.5),each=10))` as a black line – these are the coefficients that were actually used to generate the data. Which agree with the black line the most?