Matt Dickenson `mcd31`
STA561/CS571 — Fall 2013 **Homework 2** Due: 23 September, 2013

Homework Notes: I did not work with anyone else on this homework or refer to resources other than the course notes, textbook, and course Piazza page.

Problem 1

A Using the normal equation $\hat{\beta} = (X^T X)^{-1}(X^T Y)$, we can calculate $\hat{\beta}$ in $\mathcal{R}$ with the following program (after loading the data as `lindata`):

Listing 1: R Code for 1A

```
X.prime.X = t(X) %*% X
X.prime.X.inverse = solve(X.prime.X)
X.prime.Y = t(X) %*% Y
beta.hat = X.prime.X.inverse %*% X.prime.Y
beta.hat #=> 69.942167, 15.119109, 0.321028

# B
linreg = function(X, Y){
```

Letting β_i correspond to X_i and β_0 represent the intercept, $\hat{\beta}_0 = 69.94, \hat{\beta}_1 = 15.12, \hat{\beta}_2 = 0.32$.

B We can also estimate β using online stochastic gradient descent using $\mathcal{R}$'s `optim` function for minimization:

Listing 2: R Code for 1B

```
    sq.resid = residuals^2
    rss = sum(sq.resid)
    return(rss)
  }
  results = optim(rep(0, ncol(X)), RSS,
    hessian=TRUE, method="BFGS", x=X, y=Y)
  list(beta=results$par,
    vcov=solve(results$hessian),
    converged=results$convergence==0)
}
model = linreg(X, Y)
model$beta #=> 69.942167, 15.119109, 0.321028

# C
ridgereg = function(X, Y, sigma.sq=1, tau.sq=1){
```

Using this method we reach the same results as above: $\hat{\beta}_0 = 69.94, \hat{\beta}_1 = 15.12, \hat{\beta}_2 = 0.32$.

C A third method we can use to estimate β is ridge regression, using the following $\mathcal{R}$ code:

Listing 3: R Code for 1C

```
  lambda.id = lambda * diag(D)
  X.prime.X = t(X) %*% X
  lambda.X.prime.X.inverse = solve(lambda.id + X.prime.X)
  X.prime.Y = t(X) %*% Y
  beta.hat = lambda.X.prime.X.inverse %*% X.prime.Y
  return(beta.hat)
}
beta.ridge = ridgereg(X, Y)
beta.ridge #=> 65.9129285, 14.3521921, 0.3478858

# D
# a
```

From this approach we get the estimates $\hat{\beta}_1 = 3.00, \hat{\beta}_2 = 0.78$.

D Table 1 presents the residual sum of squares (RSS) for the training data using each of the methods above.

Table 1: Training set RSS for linear regression methods

Method	RSS
Normal equations	98,729.3
Online stochastic gradient descent	98,729.3
Ridge regression	99,005.9

E Table 2 presents the residual sum of squares (RSS) for the test data using each of the methods above.

Table 2: Test set RSS for linear regression methods

Method	RSS
Normal equations	11,573.2
Online stochastic gradient descent	11,573.2
Ridge regression	11,523.5

F We can now take the predicted blood pressure for a hypothetical female weighing 135 pounds:

Table 3: Predicted values

Method	$\mathbb{E}[Y \mid X = [1, 135]^T, \hat{\beta}]$
Normal equations	128.4
Online stochastic gradient descent	128.4
Ridge regression	127.2

G PLOTS HERE

H Of the three methods used above, ridge regression is the least likely to overfit the data. This is because the λ term (using the notation in the MLAPP textbook) helps to regularize the coefficients. This leads to higher RSS but also makes the coefficients less susceptible to outliers in the training data, as evidenced by the lower RSS using the training set data. The other two methods provide no such protection against outliers, and are thus more likely to overfit the training data.

I The researchers should use the ridge regresion results. One reason for this is its robustness (relative to the other two methods) to overfitting. This is especially pertinent given the small n of the training data. However, the similarity in the coefficients should help allay any concerns the researchers may have over the differences in the three methods.

Matt Dickenson `mcd31`
STA561/CS571 — Fall 2013 **Homework 2** Due: 23 September, 2013

Problem 2

A The following $\mathcal{R}$ code implements the IRLS algorithm as presented in the MLAPP textbook (after loading the data):

Listing 4: R Code for 2A

```r
sigm = function(eta){
  out = exp(eta)/( exp(eta) + 1)
  return(out)
}

irls = function(X, Y, epsilon=1/1e10){
  D = ncol(X)
  N = nrow(X)

  w = matrix(0, nrow=D, ncol=D)
  w0 = log(mean(Y)/(1-mean(Y)))
  eta = mu = s = z = matrix(NA, nrow=N, ncol=D)

  converged = FALSE
  numiters = 0
  while(!converged){
    last.w = w
    for(i in 1:N){
      xi = matrix(X[i, ], ncol=1)
      eta[i, ] = w0 + (t(w) %*% xi)
      mu[i, ] = sigm(eta[i, ])
      s[i, ] = mu[i, ] * (1-mu[i, ])
      z[i, ] = eta[i, ] + ((Y[i, ] - mu[i, ])/s[i, ])
    }
    S = matrix(0, nrow=N, ncol=N)
    for(i in 1:N){
      S[i, i] = s[i]
    }
    first.term = solve(t(X) %*% S %*% X)
    second.term = t(X) %*% S %*% z
    w =  first.term %*% second.term
    numiters = numiters + 1
    diff = w - last.w
    smalldiff = TRUE
    for(i in 1:D){
      smalldiff = smalldiff & (diff[i, i] < epsilon)
    }
```

```
      if(smalldiff){
        converged = TRUE
        output = paste("converged after ", numiters, " iterations", sep="")
        print(output)
      }
      if(numiters > 100){
        print("reached max iterations")
        break
      }
    }
    return(w)
}

irls(X, Z)
```