SLDM II - Homework 3

Matt Isaac

November 06, 2018

1. Maximum Likelihood Estimation (14 pts)

a. Let $X_1, \ldots, X_n$ be i.i.d. sample from a Poisson distribution with parameter λ, i.e.

$$P(X = x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

i. Write down the likelihood function $L(\lambda)$.

$$L(\lambda) = \prod_{i=1}^{n} f(X_i|\lambda)$$

$$L(\lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}$$

ii. Write down the log-likehood function $\ell(\lambda)$.

$$\ell(\lambda) = \log\left(\prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\right)$$

$$\ell(\lambda) = \sum_{i=1}^{n} \log\left(\frac{\lambda^{x_i} e^{-\lambda}}{x!}\right)$$

$$\ell(\lambda) = \sum_{i=1}^{n} \log(\lambda^{x_i} e^{-\lambda}) - \log(x!)$$

$$\ell(\lambda) = \sum_{i=1}^{n} \log(\lambda^{x_i}) + \log(e^{-x_i}) - \log(x!)$$

$$\ell(\lambda) = \sum_{i=1}^{n} x_i \log(\lambda) - \lambda - \log(x!)$$

$$\ell(\lambda) = log(\lambda)\left(\sum_{i=1}^{n} x_i\right) - n\lambda - \left(\sum_{i=1}^{n} \log(x!)\right)$$

iii. Find the MLE of the parameter λ.

$$\ell'(\lambda) = \frac{1}{\lambda}\sum_{i=1}^{n} x_i - n$$

Set derivative equal to 0, and solve for $\hat{\lambda}$:

$$\frac{1}{\hat{\lambda}}\sum_{i=1}^{n} x_i - n = 0$$

$$\frac{1}{\hat{\lambda}}\sum_{i=1}^{n} x_i = n$$

$$\frac{1}{\hat{\lambda}} = n\left(\sum_{i=1}^{n} x_i\right)^{-1}$$

$$\hat{\lambda} = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{X}$$

b. Let $X_1, \dots, X_n$ be an i.i.d. sample from an expoenetial distribution with the density function

$$p(x;\beta) = \frac{1}{\beta}e^{-\frac{x}{\beta}}, \quad 0, \leq x < \infty$$

Find the MLE of the parameter β. Given what you know about the role that β plays in the exponential distribution, does the MLE make sense? Why or why not?

To make the calculations simplier, let $\lambda = \frac{1}{\beta}$.

$$P(x;\lambda) = \lambda e^{-\lambda x}$$

$$\implies L(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$

$$L(\lambda) = \lambda^n \prod_{i=1}^{n} e^{-\lambda x_i}$$

$$\ell(\lambda) = \log\left(\lambda^n \prod_{i=1}^{n} e^{-\lambda x_i}\right)$$

$$\ell(\lambda) = \log(\lambda^n) + \sum_{i=1}^{n} \log(e^{-\lambda x_i})$$

$$\ell(\lambda) = n\log(\lambda) - \lambda\sum_{i=1}^{n} x_i$$

$$\ell'(\lambda) = \frac{n}{\lambda} - \sum_{i=1}^{n} x_i$$

Setting the derivative equal to 0, and solving for λ,

$$\frac{n}{\hat{\lambda}} - \sum_{i=1}^{n} x_i = 0$$

$$\frac{n}{\hat{\lambda}} = \sum_{i=1}^{n} x_i$$

$$\hat{\lambda} = \frac{n}{\sum\limits_{i=1}^{n} x_i}$$

When we substitute β back in for λ, we have

$$\frac{1}{\hat{\beta}} = \frac{n}{\sum\limits_{i=1}^{n} x_i}$$

$$\hat{\beta} = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{X}$$

This makes sense because the parameter λ in a exponential distribution describes the average time between events. The average value is the most likely value for the random variable to take on.

2. Consider training data $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)$ for binary classification and assume $y_i \in \{-1, 1\}$. Show that if $L(y, t) = \log(1 + \exp(-yt))$, then

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, \mathbf{w}^T\mathbf{x}_i + b)$$

is proportional to the neegative log-likelihood for logistic regression. Therefore ERM with the logistic loss is equivalent to the maximum likelihood approach to logistic regression.

Recall that

$$\eta(\mathbf{x}) = P(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}+b}}$$

$$1 - \eta(\mathbf{x}) = P(Y = -1|\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}+b}}$$

Thus,

$$f^*(\mathbf{x}) = \frac{1}{1 + e^{-y(\mathbf{w}^T\mathbf{x}+b)}}$$

$$L(y, \mathbf{w}^T\mathbf{x} + b) = \prod_{i=1}^{n} \frac{1}{1 + e^{-y(\mathbf{w}^T\mathbf{x_i}+b)}}$$

$$\ell(y, \mathbf{w}^T\mathbf{x} + b) = \sum_{i=1}^{n} \log(1) - \log\left(1 + e^{-y(\mathbf{w}^T\mathbf{x_i}+b)}\right)$$

$$-\ell(y, \mathbf{w}^T\mathbf{x} + b) = \sum_{i=1}^{n} \log\left(1 + e^{-y(\mathbf{w}^T\mathbf{x_i}+b)}\right)$$

The only difference between the negative log-likelihood, and the given loss function ($\frac{1}{n}\sum\limits_{i=1}^{n} L(y_i, \mathbf{w}^T\mathbf{x}_i + b)$) is the constant, $\frac{1}{n}$. So, these two are proportional.

3.

a. Show that if f is strictly convex, then f has at most one global minimizer.

Let f be strictly convex, and suppose that f has two global minimums, located at $\mathbf{x}$ and $\mathbf{y}$. In other words, $f(x) = f(y) = z$, where z is a global minimum. Recall the definition of strict convexity:

$$f(t\mathbf{x} + (1-t)\mathbf{y}) < tf(\mathbf{x}) + (1-t)f(\mathbf{y}),$$

where $t \in \{0, 1\}$. Let $f(t\mathbf{x} + (1-t)\mathbf{y}) = f(w)$.

$$\implies f(w) = f(t\mathbf{x} + (1-t)\mathbf{y}) < tf(\mathbf{x}) + (1-t)f(\mathbf{y}),$$

$$\implies f(w) < tz + (1-t)z)$$

$$\implies f(w) < tz + (1-t)z)$$

$$\implies f(w) < z$$

This indicates that we found a point, w, at which the function evaluated, $f(w)$ is less than z, which we defined as a global minimum. Thus, we have reached a contradiction, as we cannot have a value less than a global minimum. Thus, by contradiction we have shown that if f is strictly convex, it must have at most 1 global minimum.

b. Prove that the sum of 2 convex functions is convex.

Let $f(x)$ and $g(x)$ be two convex functions, both twice differentable. Due to the properties of twice differntable functions, we know that $\nabla^2 f(x^*)$ and $\nabla^2 g(y^*)$ both exist, and are positive semi-definite matrices. We also know that x^* and y^* and local minimums. By the definition of semi-definite matrices, we also know that for some vector

$$\mathbf{z}^T(\nabla^2 f(x^*))\mathbf{z} \geq 0 \text{ for all } \mathbf{z} \in \mathbb{R}^d$$

$$\mathbf{z}^T(\nabla^2 g(y^*))\mathbf{z} \geq 0 \text{ for all } \mathbf{z} \in \mathbb{R}^d$$

.

$$\implies \mathbf{z}^T(\nabla^2 f(x^*))\mathbf{z} + \mathbf{z}^T(\nabla^2 g(y^*))\mathbf{z} \geq 0$$

$$\implies \mathbf{z}^T(\nabla^2 f(x^*)) + (\nabla^2 g(y^*))\mathbf{z} \geq 0,$$

Which implies that $\nabla^2 f(x^*)) + (\nabla^2 g(y^*))$ is also positive semmi-definite. This is the hessian of $f(x^*) + g(y*)$. Since the hessian of $f(x^*) + g(y*)$ is positive semi-definite, $f(x^*) + g(y*)$ is convex.

c. Consider the function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$, where A is a symmetric $d \times d$ matrix. Derive the Hessian of f. Under what conditions on A is f convex? Strictly convex?

To begin with, we will calculate the gradient, $\nabla f(\mathbf{x})$.

Finding the gradient of the first term:

$$\nabla(\frac{1}{2}\mathbf{x}^T A\mathbf{x}) = \nabla\left(\sum_{j=1}^{d}\sum_{i=1}^{d} A_{ij}\mathbf{x_i}\mathbf{x_j}\right)$$

Using the properties found in the matrix calculus resource (and since A is symmetric),

$$\nabla(\frac{1}{2}\mathbf{x}^T A\mathbf{x}) = 2A\mathbf{x}.$$

Next,

$$\nabla(\mathbf{b}^T\mathbf{x}) = \mathbf{b}^T.$$

Since we need $\mathbf{b}$ to be a column vector, we will write is as

$$\nabla(\mathbf{b}^T\mathbf{x}) = \mathbf{b}.$$

Lastly,

$$\nabla c = 0.$$

Putting these indivual gradients together,

$$\nabla f(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$$

Movint on to the Hessian,

$$\nabla^2 f(\mathbf{x}) = A.$$

Recalling the properties of the Hessian of a function, $f(\mathbf{x})$ is convex if A is positive semidefinite, and if A is positive definite, $f(\mathbf{x})$ is strictly convex.

d. Let $J(\theta)$ be a twice continuously differentiable function. Derive the update step for Newton's method from the second order approximation of $J(\theta)$.

From the lecture notes, the 2nd order approximation of $J(\theta)$ is

$$J(\theta) \approx J(\theta_t) + \nabla J(\theta_t)^T(\theta - \theta_t) + \frac{1}{2}(\theta - \theta_t)^T \nabla^2 J(\theta_t)(\theta - \theta_t)$$

Expanding,

$$J(\theta) \approx J(\theta_t) + [\nabla J(\theta_t)]\theta - [\nabla J(\theta_t)]\theta_t + \frac{1}{2}[\theta(\nabla^2 J(\theta_t)) - \theta_t^T(\nabla^2 J(\theta_t))][\theta - \theta_t]$$

$$J(\theta) \approx J(\theta_t) + [\nabla J(\theta_t)]\theta - [\nabla J(\theta_t)]\theta_t + \frac{1}{2}[\theta_t(\nabla^2 J(\theta_t))\theta - \theta_t^T(\nabla^2 J(\theta_t))\theta + \theta_t^T(\nabla^2 J(\theta_t))\theta_t - \theta^T(\nabla^2 J(\theta_t))\theta_t]$$

$$\nabla J(\theta) \approx [J(\theta_t)]^T + \frac{1}{2}\frac{2}{1}\theta_t(\nabla^2 J(\theta_t)) - \frac{1}{2}\theta_t^T(\nabla^2 J(\theta_t)) - \frac{1}{2}\nabla^2 J(\theta_t)\theta_t$$

$$\nabla J(\theta) \approx \nabla J(\theta_t) + (\nabla^2 J(\theta_t))\theta_t - \frac{1}{2}(\nabla^2 J(\theta_t))\theta_t - \frac{1}{2}(\nabla^2 J(\theta_t))\theta_t$$

$$\nabla J(\theta) \approx \nabla J(\theta_t) + (\nabla^2 J(\theta_t))\theta_t - (\nabla^2 J(\theta_t))\theta_t$$

Set equal to 0 to minimize:

$$0 = \nabla J(\theta_t) + (\nabla^2 J(\theta_t))\theta - (\nabla^2 J(\theta_t))\theta_t$$

$$(\nabla^2 J(\theta_t))\theta = -\nabla J(\theta_t) + (\nabla^2 J(\theta_t))\theta_t$$

$$\theta_{t-1} = \theta_t - [\nabla^2 J(\theta_t)]^{-1}[\nabla J(\theta_t)]$$

4. Determine a formula for the gradient and the Hessian of the regularized logistic regression objective function. Argue that the objective function ($J(\theta) = -\ell(\theta) + \lambda||\theta||^2$) is convex when $\lambda \geq 0$ and that for $\lambda > 0$ is strictly convex.

From the lecture slides, we obtain the following equation for $J(\theta)$:

$$J(\theta) = -\sum_{i=1}^{n}\left[y_i \log\left(\frac{1}{1+e^{-\theta^T\mathbf{x}_i}}\right) + (1-y_i)\log\left(\frac{e^{-\theta^T\mathbf{x}_i}}{1+e^{-\theta^T\mathbf{x}_i}}\right)\right] + \lambda||\theta||^2$$

$$\implies J(\theta) = -\sum_{i=1}^{n}\left[y_i(0 - \log(1+e^{-\theta^T\mathbf{x}_i}) + (1-y_i)(\log(e^{-\theta^T\mathbf{x}_i}) - \log(1+e^{-\theta^T\mathbf{x}_i}))\right] + \lambda||\theta||^2$$

$$\implies J(\theta) = -\sum_{i=1}^{n}\left[-y_i\log(1+e^{-\theta^T\mathbf{x}_i}) - \theta^T\mathbf{x}_i - \log(1+e^{-\theta^T\mathbf{x}_i}) + y_i\theta^T\mathbf{x}_i + y_i\log(1+e^{-\theta^T\mathbf{x}_i}))\right] + \lambda||\theta||^2$$

$$\implies J(\theta) = -\sum_{i=1}^{n}\left[-\theta^T\mathbf{x}_i - \log(1+e^{-\theta^T\mathbf{x}_i}) + y_i\theta^T\mathbf{x}_i\right] + \lambda||\theta||^2$$

Then, the gradient, $\nabla J(\theta)$ is

$$\implies \nabla J(\theta) = -\sum_{i=1}^{n}\left[-\mathbf{x}_i + y_i\mathbf{x}_i + \frac{e^{-\theta^T\mathbf{x}_i}\mathbf{x}_i}{1+e^{-\theta^T\mathbf{x}_i}}) + y_i\mathbf{x}_i\right] + \lambda||\theta||^2$$

Next, calculating the hessian:

$$\nabla^2 J(\theta) = -\sum_{i=1}^{n}\left[x_i\frac{(1+e^{-\theta^T\mathbf{x}_i})(e^{-\theta^T\mathbf{x}_i})(-\mathbf{x}_i) - \left[(e^{-\theta^T\mathbf{x}_i})(e^{-\theta^T\mathbf{x}_i})(-\mathbf{x_i})\right]}{(1+e^{-\theta^T\mathbf{x}_i})^2}\right] + 2\lambda I$$

To simplify the algebra, let $A = e^{-\theta^T\mathbf{x}_i}$ and $B = -\mathbf{x}_i$. Then,

$$\nabla^2 J(\theta) = -\sum_{i=1}^{n}\left[x_i\frac{(1+A)(AB) - \left[A^2B\right]}{(1+A)^2}\right] + 2\lambda I$$

$$\nabla^2 J(\theta) = -\sum_{i=1}^{n}\left[x_i\frac{AB + A^2B - A^2B}{(1+A)^2}\right] + 2\lambda I$$

$$\nabla^2 J(\theta) = -\sum_{i=1}^{n}\left[x_i\frac{AB}{(1+A)^2}\right] + 2\lambda I$$

$$\nabla^2 J(\theta) = -\sum_{i=1}^{n}\left[x_i\frac{AB}{(1+A)^2}\right] + 2\lambda I$$

$$\nabla^2 J(\theta) = -\sum_{i=1}^{n}\left[x_i\frac{e^{-\theta^T\mathbf{x}_i}(-\mathbf{x}_i)}{(1+e^{-\theta^T\mathbf{x}_i})^2}\right] + 2\lambda I$$

$$\nabla^2 J(\theta) = \sum_{i=1}^{n}\left[x_i^2\frac{e^{-\theta^T\mathbf{x}_i}}{(1+e^{-\theta^T\mathbf{x}_i})^2}\right] + 2\lambda I$$

Thus, the hessian is

$$\nabla^2 J(\theta) = \sum_{i=1}^{n}\left[x_i x_i^T\frac{e^{-\theta^T\mathbf{x}_i}}{(1+e^{-\theta^T\mathbf{x}_i})^2}\right] + 2\lambda I$$

5. Given training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, yn)$, define the empirical risk for either a regression or classification problem as

$$\hat{R}(f_\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f_\theta(x_i)).$$

Write pseudocode describing how you would implement stochastic gradient descent to minimize $\hat{R}(f_\theta)$ with respect to θ. Assume a fixed mini-batch size of m and assume that the step size α is fixed for each epoch.

psudo code:

```
set threshold (i.e. stopping point)
datacopy = random permutation of data
initialize current threshold
initailize theta
initialize magnitude of the gradient of Rhat
alpha = fixed step size
while(magnitude of the gradient of Rhat > threshold){
  m = fixed mini-batch size
  for(i in 1:m){
    batch = take first m observations from datacopy (can cut from datacopy, since it is a copy)
    create empty vector called losses to hold terms from summation
    for(i in 1:n/m)
    {
      losses[i] = result of L(y_i, f(theta, x_i))
    }
    Rhat = (1/n) * sum(losses)
    Find gradient of Rhat
    Adjust theta by taking step of size alpha in the direction of the gradient
    Update the magnitude of the gradient of Rhat
  }
}
```

6. Implement Newton's method to find a minimizer of the regularized negative log likelihood for logistic regression: $J(\theta) = -\ell(\theta) + \lambda||\theta||$. Try setting $\lambda = 10$. Use the first 2000 examples as training data and the last 1000 as test data.

a. Report the test error, your termination criterion (you may choose), how you initialized θ_0, and the value of the objective function at the optimum.

I obtained a test error of 5.5% (94.5% correctly classified).
$\theta_0 = \begin{bmatrix} b & w_1 & w_2 & \dots & w_d \end{bmatrix}^T$ was initialized with $b = 1$, $w_1 = w_2 = \cdots = w_d$.
The value of the objective function at the optimized value of θ was -223733.80.

b.

See figure 1.

I defined "confidence", C, as the absolute value of the probability obtained from logistic regression minus 0.5 ($C = |P(Y|X) - 0.5|$). The observations with the least amount of confidence will be those with probabilities close to 0.5. Thus, this definition of confidence finds those values that are furthest from 0.5. $P(Y|X) \in [0, 1]$, which implies $C \in [0, 0.5]$. The highest values of C correspond to the predictions with the highest confidence.

c. Include your well-organized, clearly commented code.

See below:

```
library(R.matlab)
library(dplyr)
library(geometry)
library(pracma)

# Read in and format data into dataframe with column "Y" and columns "X_1",
# "X_2", ...
```

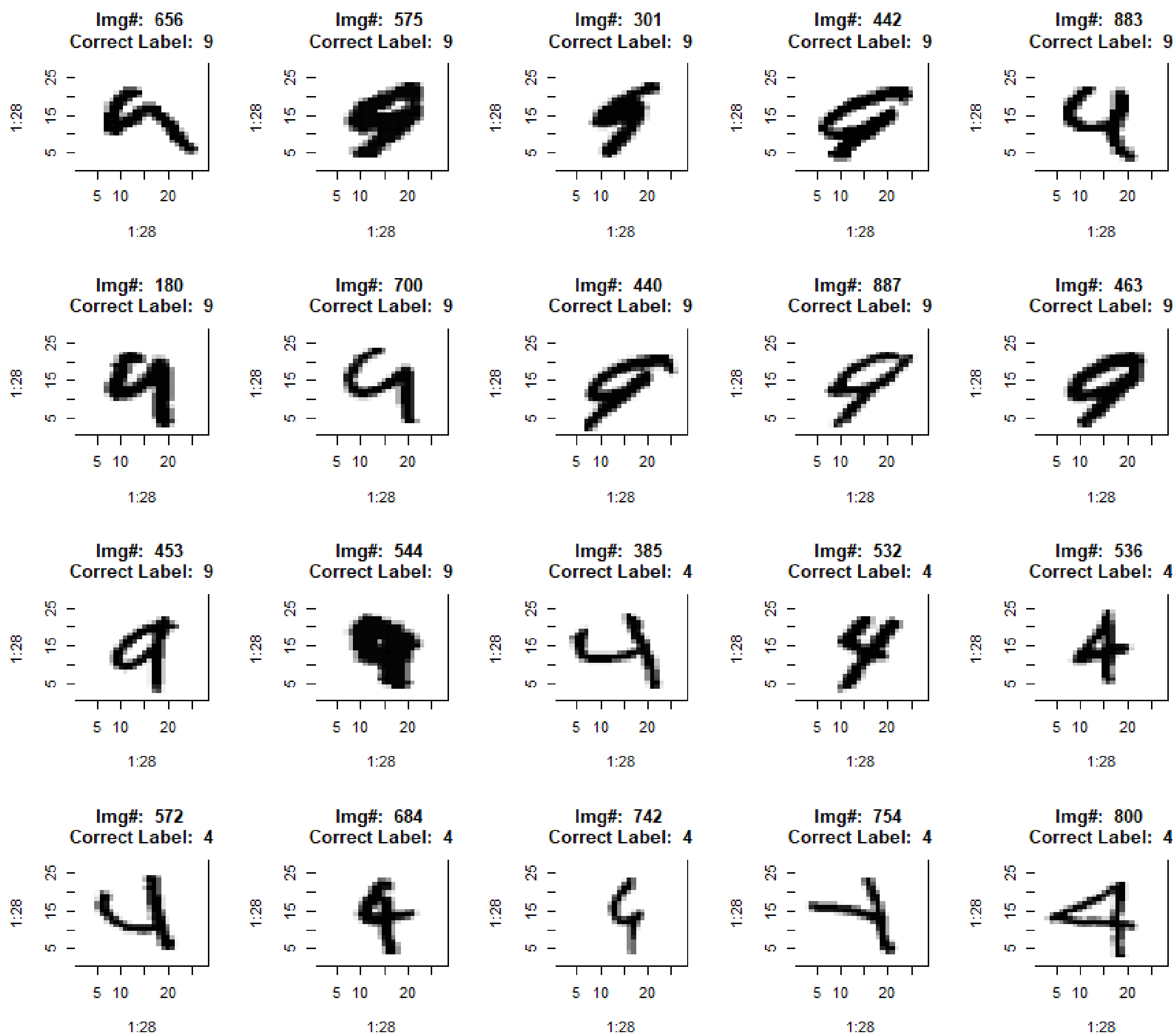
Img#: 656
Correct Label: 9
Img#: 575
Correct Label: 9
Img#: 301
Correct Label: 9
Img#: 442
Correct Label: 9
Img#: 883
Correct Label: 9
Img#: 180
Correct Label: 9
Img#: 700
Correct Label: 9
Img#: 440
Correct Label: 9
Img#: 887
Correct Label: 9
Img#: 463
Correct Label: 9
Img#: 453
Correct Label: 9
Img#: 544
Correct Label: 9
Img#: 385
Correct Label: 4
Img#: 532
Correct Label: 4
Img#: 536
Correct Label: 4
Img#: 572
Correct Label: 4
Img#: 684
Correct Label: 4
Img#: 742
Correct Label: 4
Img#: 754
Correct Label: 4
Img#: 800
Correct Label: 4

Figure 1:

```r
df <- R.matlab::readMat("mnist_49_3000.mat") %>% lapply(t) %>% lapply(as_tibble)
colnames(df[[1]]) <- sprintf("X_%s",seq(1:ncol(df[[1]])))
colnames(df[[2]]) <- c("Y")
df <- bind_cols(df) %>% select(Y, everything())

### Function Definitions

objFunc <- function(xs, y, theta, lambda){
  # objFunc() calculates the negative log likelihood with the added
  # regularization term.
  #
  # Args:
  #   xs: dataframe of training predictor variables (i.e. feature vectors)
  #   y:  vector of training response variables. Must be of the same length as xs.
  #   lambda: value of lambda in regularization term
  #
  # Returns:
  #   The J(theta) = negative log likelihood for logistic regression with regulizer
  #   term added on.
  terms <- double(nrow(xs))
  for(i in 1:nrow(xs)){
    xi <- as.numeric(xs[i,])
    yi <- as.numeric(y$Y[i])
    t1 <- yi*log(1/(1 + exp(dot(-theta, xi))))
    t2 <- (1-yi)*log((exp(dot(-theta, xi)))/(1 + exp(dot(-theta, xi))))
    terms[i] <- t1 + t2
  }
  Jmin <- (-1 * sum(terms)) + (lambda * dot(theta, theta))
  return(Jmin)
}

calcmu <- function(theta, xi){
  # calculates mu as defined in
  # http://www.cs.cmu.edu/~mgormley/courses/10701-f16/slides/lecture5.pdf
  #
  # Note: The classification did not work when using the hessian formulas
  # derived in #4. I think they are mostly correct, but we could not get them
  # to work in the code. As a last resort, we tried the formulas for the hessian
  # found on these slides, and achieved 94.5% correct classification.
  #
  # Args:
  #   theta: vector containing intercept (b) and weights (w)
  #   xi: vector of predictor variables (one observation ("row"))
  #
  #   theta and xi must be vectors of same length
  #
  # Returns:
  #   The optimized value of theta
  denom <- (1 + exp(dot(-theta, xi)))
  return(1/denom)
}

mellonGH <- function(xmat, yvec, theta){
```

```r
  # Calculates the Gradient and Hessian
  # Args:
  #   xmat: matrix of predictor variables (including column of 1's)
  #   yvec: dataframe (tibble?) of response values
  #   theta: vector of b and w's.
  #
  # Returns:
  #   list containing gradient (grad) and hessian (hess)
  lambda <- 1
  mus <- double(nrow(xmat))
  for(i in 1:nrow(xmat)){
    xi <- as.numeric(xmat[i,])
    mus[i] <- calcmu(theta, xi)
  }
  grad <- t(xmat) %*% (mus - as.numeric(yvec$Y)) + 2*lambda*theta
  ds <- mus * (1 - mus)
  D <- diag(ds)
  XT <- t(sapply(xmat, as.numeric))
  X <- sapply(xmat, as.numeric)
  tlambI <- diag(rep(2 * lambda, 785))
  hess <- XT %*% D %*% X + tlambI

  results <- list("grad" = grad, "hess" = hess)
  return(results)
}

newtonsMethodinit <- function(maxitr, predictors, response){
  # Kicks off newtonsMethod() recursive function
  # Args:
  #   maxitr: maximum iterations for Newton's Method
  #   predictors: dataframe of predictor variables
  #   response: dataframe (tibble?) of response values
  #
  # Returns:
  #   The optimized value of theta

  b = 1 # initial guess of b
  ws = rep(0, 784) # initial guess of w's
  inittheta <- c(b, ws)

  theta <- newtonsMethod(theta = inittheta, nitr = 0,
                         maxitr = maxitr, xs = predictors, y = response)
  return(theta)
}

newtonsMethod <- function(theta, nitr, maxitr, xs, y){
  # Recursive function that implements Newton's method
  # Args:
  #   theta: vector of w's and b's
  #   nitr: current iteration number
  #   maxitr: maximum number of iterations for Newton's method
  #   xs: dataframe of predictor variables
  #   y: dataframe (tibble?) of response
```

```r
  #
  # Returns:
  #   The optimized value of theta, or calls itself again.
  if(nitr >= maxitr){
    return(theta)
  } else {
    print(paste("Iteration Number: ", nitr))

    gh <- mellonGH(xmat = xs, yvec = y, theta = theta)
    grad <- gh$grad
    hess <- gh$hess

    # grad <- gradJ(xmat = xs, yvec = y, theta = theta)
    # hess <- hessJ(xs = xs, theta = theta)

    theta.1 <- theta - (inv(hess) %*% grad)
    nitr <- nitr + 1
    return(newtonsMethod(theta = theta.1, nitr = nitr,
                         maxitr = maxitr, xs = xs, y = y))
  }
}

predictNum <- function(xs, yi, theta){
  # Conducts logistic regression to predict classes, using parameter
  # estimates in theta.
  # Args:
  #   xs: dataframe of predictor variables (without column of 1's)
  #   yi: dataframe (tibble?) of response
  #   theta: vector of b and w's
  #
  # Returns:
  #   list containing a dataframe of probabilities, predicted value,
  #   actual value, and correct indicator, as well as pcc
  #   (percent correctly classified)
  w <- theta[2:length(theta)]
  b <- theta[1]
  probs <- double(nrow(xs))
  predict <- double(nrow(xs))
  correct <- double(nrow(xs))
  for(i in 1:nrow(xs)){
    xi <- as.numeric(xs[i,])
    p <- 1 / (1 + exp(-dot(w, xi) + b))
    probs[i] <- p
    if(p > 0.5){
      predict[i] <- 1
    } else{
      predict[i] <- -1
    }
    if(as.numeric(predict[i]) == as.numeric(yi$Y[i])){
      correct[i] = 1
    } else {
      correct[i] = 0
    }
```

```r
  }
  result <- data.frame(probs, predict, Y = yi$Y, correct)
  pcc <- sum(correct)/nrow(xs)
  lst <- list("results" = result, "pcc" = pcc)

  return(lst)
}

# create and format training data
train <- dplyr::slice(df, 1:2000)
trainPred <- select(train, -Y)
X0 <- rep(1, 2000) # add column of 1's to line up with intercept
trainPred <- cbind(X0, trainPred)
trainResp <- resp <- select(train, Y)

# create and format test data
test <- dplyr::slice(df, 2001:3000)
testPred <- select(test, -Y)
X0 <- rep(1000) # add column of 1's to line up with intercept
testPred <- cbind(X0, testPred)
testResp <- select(test, Y)

# Estimate theta from training data
thetatrain <- newtonsMethodinit(maxitr = 2, predictors = trainPred,
                                response = trainResp)

# Predict classes of test data
pred <- predictNum(xs = select(testPred, -X0), yi = testResp,
                   theta = thetatrain)
results <- pred$results
pcc <- pred$pcc
pcc # view PCC

# Calculate minimum value of Objective function
minOF <- objFunc(xs = trainPred, y = trainResp, theta = thetatrain, lambda = 1)

# Get top 20 "most sure but misclassified" observations

incorrect <- results[results$correct == 0,] # filter out correctly classified
incorrect$confidence <- abs(incorrect$probs - 0.5) # calculated 'confidence' metric
incorrect <- incorrect[order(-incorrect$confidence, incorrect$probs), ] # sort by confidence
incorrect.20 <- incorrect[1:20,] # take top 20 most confident, incorrect classifications.
incorrect.20$index <- row.names(incorrect.20)

# read data in (again) and format for visualization
mnist <- readMat("mnist_49_3000.mat")
imgList = list()
for(i in seq(1, length(mnist$x), by = 784)) {
  img <- matrix(mnist$x[i:(i + 783)], nrow = 28, byrow = TRUE)
  img <- t(apply(img, 2, rev))
  imgList[[length(imgList)+1]] = img
}
```

```r
# create container
test.imgList <- imgList[2001:3000]


par(mfrow = c(4,5))
# Visualize top 20 mis-classified
for(i in row.names(incorrect.20)){
  print(i)
  correctLabel = "intial"
  if(incorrect.20$Y[which(incorrect.20$index == i)] == -1){
    correctLabel = "4"
  } else {
    correctLabel = "9"
  }
  image(1:28, 1:28, test.imgList[[as.numeric(i)]],
        col=gray((255:0)/255),
        main = paste("Img#: ", i, "\nCorrect Label: ", correctLabel))
}
```

7. How long did this assignment take you?

Somewhere in the neighborhood of 33-35 hours.

8. Type up homework solutions:

Check.