CS 229 - Machine Learning
Xiangliang Zhang
Computer Science (CS)/Statistics (STAT) Program
Computer, Electrical and Mathematical Sciences & Engineering (CEMSE) Division
King Abdullah University of Science and Technology (KAUST)

HOMEWORK III

Henrique Aparecido Laureano

Spring Semester 2018

Contents

Question 1

Show the derivation of negative log-likelihood of logistic regression problem.

$$\mathrm{NLL}(w) = -\sum_{i=1}^{N} \Big[t_i \ln \mu_i + (1 - t_i) \ln(1 - \mu_i) \Big]$$

<u>Solution:</u>

Target variable $t \in \{0, 1\}$:

$$p(t|x, w) = \mathrm{Bernoulli}(t|\mu(x)),$$

with $\mu(x)$ representing the parameter of the Bernoulli distribution $p(t = 1|x)$.

$$\mu(x) = \mathrm{sigm}(w^\top x) = \frac{1}{1 + \exp\{-w^\top x\}} \quad \Rightarrow \quad p(t|x, w) = \mathrm{Bernoulli}(t|\mathrm{sigm}(w^\top x)).$$

$$\text{Likelihood function} = \text{product of Bernoulli's} = \prod_{i=1}^{N} \mu_i^{t_i} (1 - \mu_i)^{1 - t_i}.$$

$$\text{Negative Log-Likelihood (NLL)} = -\sum_{i=1}^{N} \Big[t_i \ln \mu_i + (1 - t_i) \ln(1 - \mu_i) \Big].$$

Derivative of NLL on w:

$$\frac{\mathrm{d}\mathrm{NLL}(w)}{\mathrm{d}w} = -\sum_{i=1}^{N} \left[\frac{t_i}{\mu_i} + \frac{t_i - 1}{1 - \mu_i} \right] \frac{\mathrm{d}\mu_i}{\mathrm{d}w} = -\sum_{i=1}^{N} \left[\frac{t_i - \mu_i}{\mu_i(1 - \mu_i)} \right] \frac{\mathrm{d}\mu_i}{\mathrm{d}w} = \sum_{i=1}^{N} \left[\frac{\mu_i - t_i}{\mu_i(1 - \mu_i)} \right] \frac{\mathrm{d}\mu_i}{\mathrm{d}w},$$

with

$$\begin{aligned}
\frac{\mathrm{d}\mu_i}{\mathrm{d}w} = \frac{\mathrm{d}(1 + \exp\{-w^\top x_i\})^{-1}}{\mathrm{d}w} &= -(1 + \exp\{-w^\top x_i\})^{-2} \exp\{-w^\top x_i\}(-x_i) \\
&= \frac{\exp\{-w^\top x_i\}}{(1 + \exp\{-w^\top x_i\})^2} x_i \\
&= \frac{1}{1 + \exp\{-w^\top x_i\}} \left(1 - \frac{1}{1 + \exp\{-w^\top x_i\}} \right) x \\
&= \mu_i(1 - \mu_i) x_i.
\end{aligned}$$

Then,

$$\frac{\mathrm{d}\mathrm{NLL}(w)}{\mathrm{d}w} = \sum_{i=1}^{N} \frac{\mu_i - t_i}{\mu_i(1 - \mu_i)} \mu_i(1 - \mu_i) x_i = \sum_{i=1}^{N} (\mu_i - t_i) x_i.$$

□

Question 2

Show how to maximize NLL(w) **and find** $w^\star$ **by gradient descent. Or other difference solutions (you can get a bonus of 20 points if you find a different solution)**

Solution:

We already have the gradient of NLL(w) by the **Question 1**:

$$\frac{\mathrm{d}\mathrm{NLL}(w)}{\mathrm{d}w} = \sum_{i=1}^{N} (\mu_i - t_i) x_i.$$

We want maximize the **negative** log-likelihood (**N**LL), so we need to find the minimum of the function (in the best scenario, the unique global minimum).

The Hessian of NLL(w) is given by:

$$\begin{aligned} H = \frac{\mathrm{d}^2\mathrm{NLL}(w)}{\mathrm{d}w^2} &= \frac{\mathrm{d}\sum_{i=1}^{N}(\mu_i - t_i)x_i}{\mathrm{d}w} \\ &= \sum_{i=1}^{N} \frac{\mathrm{d}\mu_i}{\mathrm{d}w} x_i^\top \quad \text{(we calculated this in } \textbf{Question 1}\text{)} \\ &= \sum_{i=1}^{N} \mu_i(1-\mu_i) x_i x_i^\top \\ &= X^\top S X, \quad \text{with} \quad S = \begin{bmatrix} \mu_i(1-\mu_i) & \cdots & 0 \\ \dots & \ddots & \dots \\ 0 & \cdots & \mu_i(1-\mu_i). \end{bmatrix} \end{aligned}$$

μ_i is all positive. Therefore, H is positive definite.

Thus (**N**LL) is convex, and has a unique global minimum.

The gradient descent algorithm is given by searching $w^\star$ by

$$w^{k+1} = w^k - \eta g^k, \quad \text{with} \quad g^k = \frac{\mathrm{d}\mathrm{NLL}(w^k)}{\mathrm{d}w^k} = \sum_{i=1}^{N} (\mu_i - t_i) x_i.$$

Other solution for the task of find $w^\star$ **is the use IRLS** (**I**teratively **R**eweighted **L**east **S**quares), a special case of Newton's algorithm.

IRLS uses the second derivative and has the form

$$w^{k+1} = w^k - H^{-1} g^k, \quad \text{with} \quad g^k = \sum_{i=1}^{N} (\mu_i - t_i) x_i \quad \text{and} \quad H = X^\top S^k X.$$

□

Implementation Task

Data:

Please download data `logreg_data_binary.txt`. It includes four columns.
The first column coded the target variable of "apply to graduate school", unlikely (0), or likely (1).
The other three columns are three variables as follows:

1. parent, which is a 0/1 variable indicating whether at least one parent has a graduate degree,
2. public, which is a 0/1 variable where 1 indicates that the undergraduate institution is a public university and 0 indicates that it is a private university,
3. gpa, which is the student's grade point average.

```
# <r code> =============================================================== #
path <- "~/Dropbox/KAUST/machine_learning/hw3/"                    # file path
train <- read.table(paste0(path, "logreg_data_binary.txt"))     # loading data
names(train) <- c("target", "parent", "public", "gpa")  # creating variable names
# </r code> ============================================================== #
```

In other words, each undergraduate student is described by x, which is a 3-dim vector. Can we make a prediction of his/her target t =?

Learning method:

You can use gradient descent.

NOTE:

1. Data should be standardized, e.g., for one variable x using $x^{'} = (x - \text{mean}(x))/\text{std}(x)$ so that $x^{'}$ has mean$(x^{'}) = 0$ and std$(x^{'}) = 1$.
Standardization should be down for all three variables.

```
# <r code> =============================================================== #
std.train <- train                                          # standardization
for (i in 2:4)
  std.train[ , i] <- (std.train[ , i] - mean(train[ , i])) / sd(train[ , i])
# </r code> ============================================================== #
```

The testing data should be standardized by the mean and std obtained from the variable values in training data.

```
# <r code> =============================================================== #
test <- read.table(paste0(path, "test_data_binary.txt"))        # loading data
names(test) <- c("target", "parent", "public", "gpa")   # creating variable names
std.test <- test                                            # standardization
for (i in 2:4)
  std.test[ , i] <- (std.test[ , i] - mean(train[ , i])) / sd(train[ , i])
# </r code> ============================================================== #
```

2. **One more dimension with value 1 should be added to each example.**

```
# <r code> ================================================================= #
               # adding one more dimension with value 1 to train and test datasets
x.stdtrain <- as.matrix(cbind(intercept = 1, std.train[ , 2:4]))
x.stdtest  <- as.matrix(cbind(intercept = 1,  std.test[ , 2:4]))
# </r code> ================================================================ #
```

Task: Logistic Regression with Binary target

Implement the logistic regression algorithm for this binary classification problem.

Solution:

```
# <r code> ================================================================= #
gd <- function(x, target) {                               # gd: gradient descent
  w = w.new = matrix(numeric(4))            # coefficient matrix, dimension 4 x 1
  eta = .1                                                             # costant
  nll = numeric(1)               # object to keep the nll values at each iteration
  n = length(target)                                               # sample size
  for (i in 1:500) {                      # fixing the number of iterations in 500
    mu = 1 / ( 1 + exp(-x %*% w) )                                  # computing \mu
    grad = (1 / n) * t(mu - target) %*% x                  # computing the gradient
    w.new = t(w) - eta * grad     # computing the new values of the coefficients w
    mu.new = 1 / ( 1 + exp(-x %*% t(w.new)) )        # computing \mu with the new w
      # computing and keeping the nll (negative log-likelihood) at each iteration
    nll[i] = - sum( target * log(mu.new) + (1 - target) * log(1 - mu.new) )
    # convergence criterion: diference in w between iterations smaller than 0.0001
    if (i > 2) if (all( abs(w.new - t(w)) < 1e-4 )) break
    w = t(w.new)                                     # the new w became the older w
  }           # returning the estimate w, the number of iterations and nll values
  return(list(w = w.new, i = i, nll = nll))
}                                   # Gradient Descent for the Logistic Regression
gd.lr <- gd(x = x.stdtrain, target = std.train[ , 1])
# </r code> ================================================================ #
```

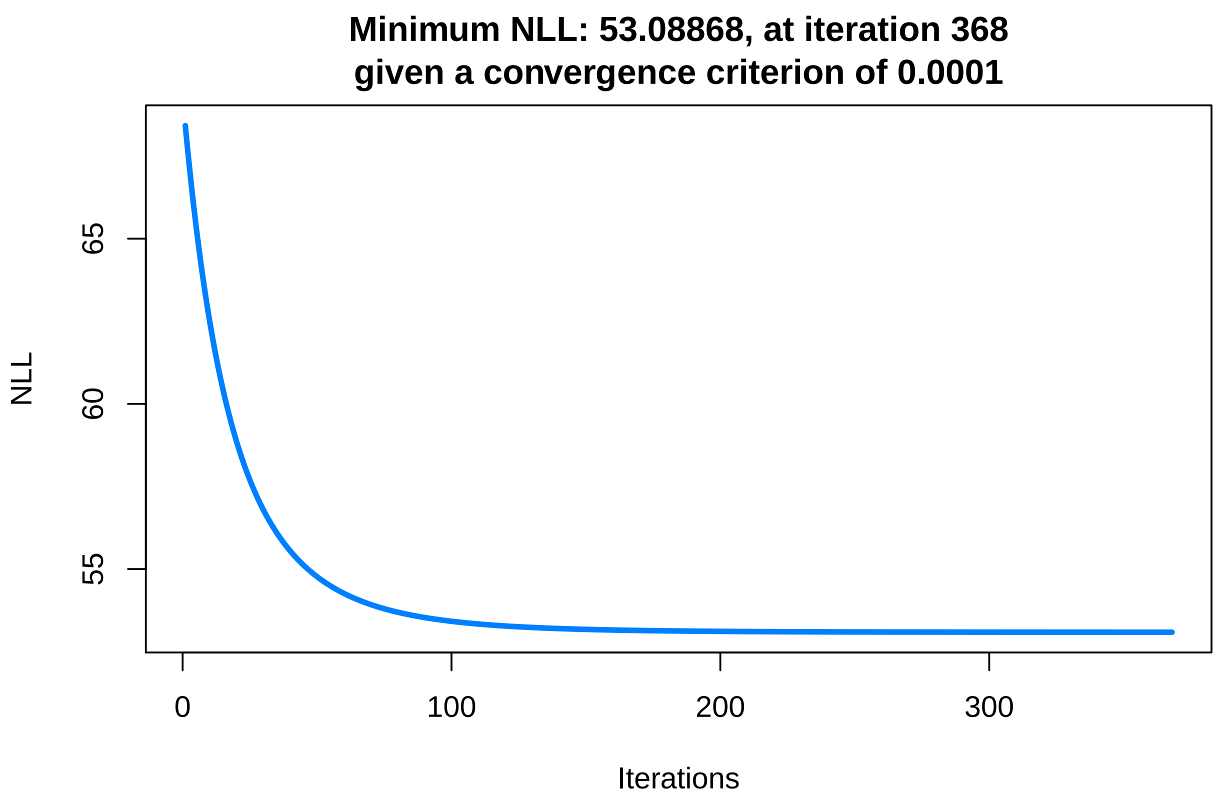
□

1)

Show the decreasing of NLL (negative log-likelihood) function with the increasing of iteration numbers.

Solution:

```r
# <r code> ================================================================== #
par(mar = c(4, 4, 3, 1) + .1)                          # graphical definitions
plot(gd.lr$nll, type = "l", lwd = 3, col = "#0080ff"     # plotting the nll values
     , xlab = "Iterations", ylab = "NLL"
     , main = paste0("Minimum NLL: ", round(min(gd.lr$nll), 5)
                     , ", at iteration ", gd.lr$i
                     , "\ngiven a convergence criterion of 0.0001"))
# </r code> ================================================================= #
```



□

2)

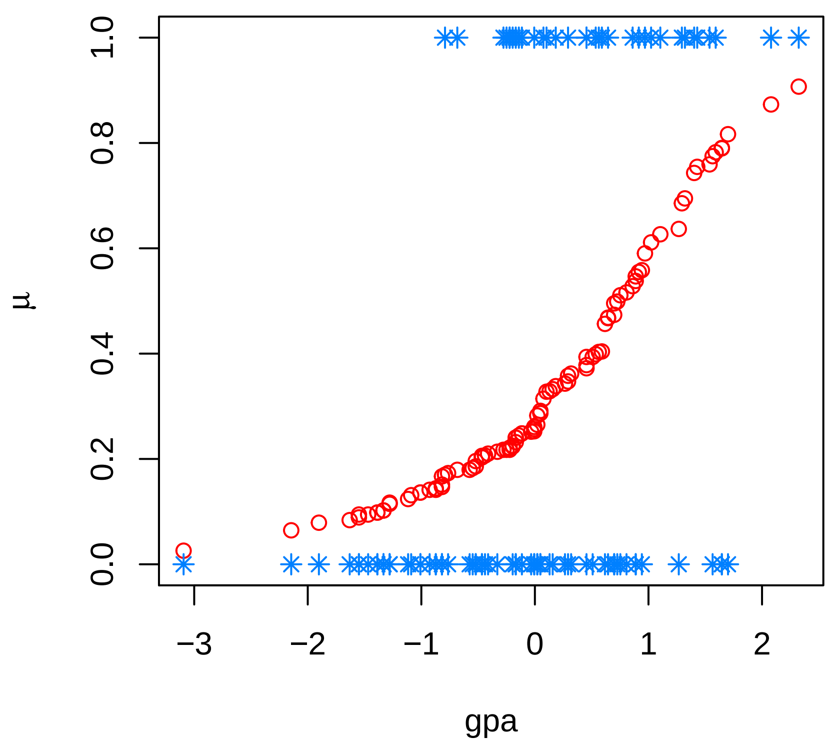
Give the results of obtained coefficient, w.

Solution:

```r
# <r code> ================================================================== #
gd.lr$w                                              # obtained coefficients, w
# </r code> ================================================================= #
```

```
      intercept    parent   public       gpa
[1,] -0.7952539 0.4605697 0.243584 0.8039067
```

```r
# <r code> =================================================================== #
                                   # function to compute \mu, the sigmoid function
mu <- function(x, w) 1 / ( 1 + exp(-x %*% t(w)) )
mu.train <- mu(x = x.stdtrain, w = gd.lr$w)   # computing mu for the train dataset
par(mar = c(4, 4, 1, 1) + .1)                            # graphical definitions
plot(sort(mu.train) ~ sort(std.train[ , 4])          # plotting the estimated curve
     , col = 2, ylim = c(0, 1), xlab = "gpa", ylab = expression(mu))
                                             # plotting corresponding target points
points(target ~ gpa, std.train, pch = 8, col = "#0080ff")
# </r code> ================================================================== #
```



□

3)

Download test data at `test_data_binary.txt`. How many target labels of test data are correctly predicted by the learned w?

Solution:

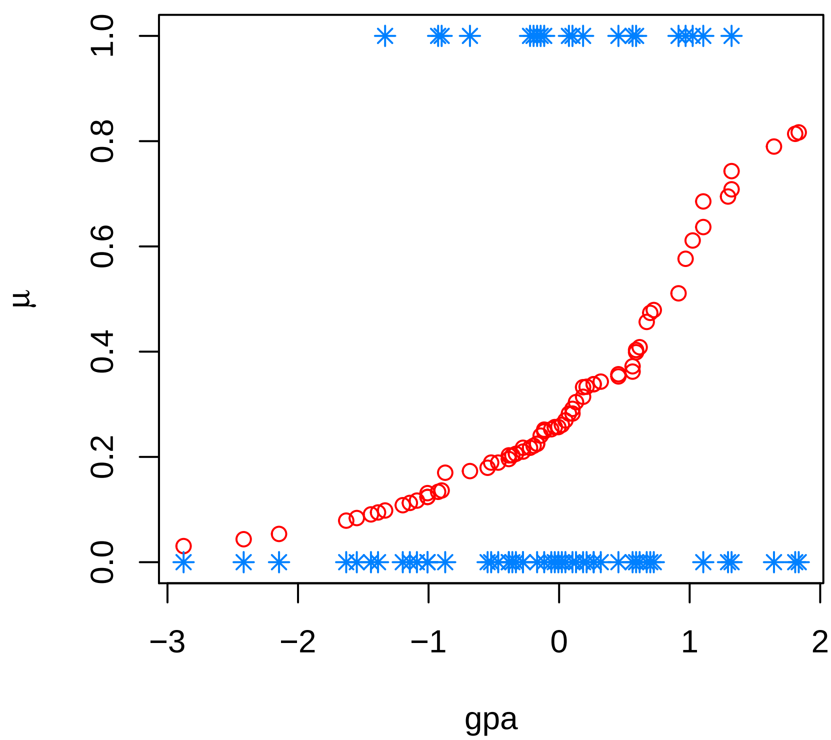
```r
# <r code> =================================================================== #
mu.test <- mu(x = x.stdtest, w = gd.lr$w)       # computing mu for the test dataset
                                  # if \mu > 0.5 => target = 1, else => target = 0
target.pred <- ifelse(mu.test > .5, 1, 0)
                             # comparing the predicted labels with the real labels
```

```r
table(target.pred == std.test[ , 1])
# </r code> ================================================================== #
```

```
FALSE  TRUE 
   21    49 
```

⇒ 49 of 70 are predicted correctly. Hit rate: 70% correct.

```r
# <r code> =================================================================== #
par(mar = c(4, 4, 1, 1) + .1)                              # graphical definitions
plot(sort(mu.test) ~ sort(std.test[ , 4])           # plotting the estimated curve
     , col = 2, ylim = c(0, 1), xlab = "gpa", ylab = expression(mu))
                                          # plotting corresponding target points
points(target ~ gpa, std.test, pch = 8, col = "#0080ff")
# </r code> ================================================================== #
```



■