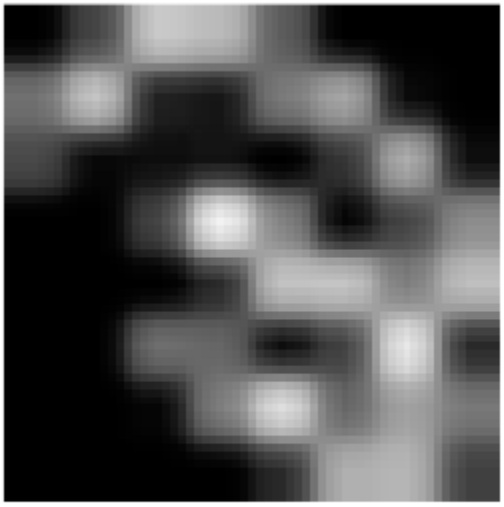
STA414 Assignment 2 Problem 6

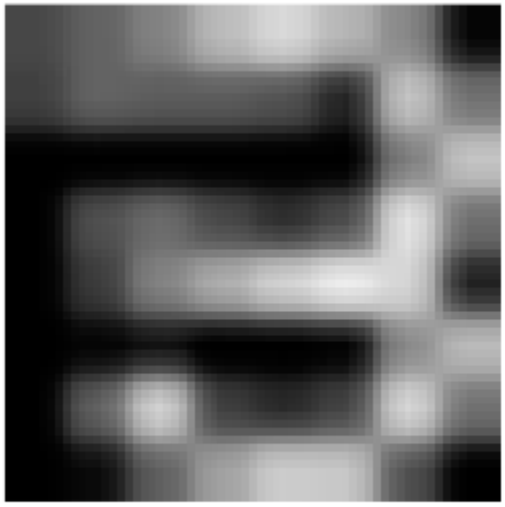
2015-11-04

Rui Qiu, #999292509

A. Warm up example

To see how the data looks like, we first visualize the dataset. First we convert two files into two different large matrices (for training set, it's a 800 by 65 matrix; for test set, it's 400 by 65). Then we pick some lines from these two matrices, and each line (ignoring the last column since it's a label) represents a 'digit' to be plotted. And finally we plot this as a raster plot. Note that the raw plots are 90 degree clockwise rotated, though we fixed the problem manually.





We plotted the 1st digit in training set and the 64th digit in test set as two examples.

B. Performance evaluation

- First we need to state that the results of computing `log p(t|x)` for each of the training and test cases

under three required models would be extremely space-consuming in this assignment. By quick calculation, the result contains 3 by 1200 = 3600 different values. Consider the fact that those values are not that important in this assignment, we skip the output of this part. However, we have still kept the command which generate these results in our code, but commented out.

- Average conditional log likelihood are concluded in the following two tables.
    - On **training data**.

	logistic regression	**Gaussian-conditional**	**regularized Gaussian-conditional**
Number 3	-0.04906491	-0.1351515	-0.04910520
Number 5	-0.05110875	-0.2767125	-0.05387059
Overall	-0.05008683	-0.2059320	-0.05148789

    - On **testing data**.

	logistic regression	**Gaussian conditional**	**regularized Gaussian-conditional**
Number 3	-0.2397528	-0.03298787	-0.006468977
Number 5	-0.1361371	-1.26257979	-0.212511104
Overall	-0.1879449	-0.64778383	-0.109490041

- Errors
    - On **training data**.

	logistic regression	Gaussian conditional	regularized Gaussian-conditional
Number 3	count = 7, rate = 1.750%	count = 5, rate = 1.250%	count = 3, rate = 0.750%
Number 5	count = 5 , rate = 1.250%	count = 6, rate = 1.500%	count = 4, rate = 1.000%
Overall	count = 12, rate = 1.500%	count = 11, rate = 1.375%	count = 7, rate = 0.875%

- On **testing data**.

	logistic regression	Gaussian conditional	regularized Gaussian-conditional
Number 3	count = 11 , rate = 5.500%	count = 2, rate = 1.000%	count = 0, rate = 0.000%
Number 5	count = 7 , rate = 3.500%	count = 9, rate = 4.500%	count = 6, rate = 3.000%
Overall	count = 18, rate = 4.500%	count = 11, rate = 2.750%	count = 6, rate = 1.500%

- The regularization helps to reduce the overall error rate in both data sets. In fact, the error rate of Number 3 is less than its counterpart of Number 5 in both datasets. Overall error rate in training data is less than that in testing data, probably because the size of training data is larger.

C. Appendix

```
library(grid)

train <- as.matrix(read.csv(file="./digitstrain.txt", header=FALSE, sep=","))
test <- as.matrix(read.csv(file="./digitstest.txt", header=FALSE, sep=","))

# 0. Warm-up example

colnames(train) <- NULL
colnames(test) <- NULL
plot(as.raster(matrix(train[1, 1:64], ncol = 8, nrow = 8)))
```

```r
plot(as.raster(matrix(test[64, 1:64], ncol = 8, nrow = 8)))

# One can use the previous command to plot any 'digit' in either training or
# test sets. The only thing need to do is to change the set name and the
# certain index of the 'digit'.

# reassign two data sets as tables
train <- read.table(file="./digitstrain.txt", sep=",")
test <- read.table(file="./digitstest.txt", sep=",")

# 1. LOGISTIC REGRESSION

N <- nrow(train)
M <- nrow(test)

sigmoid <- function(t, x) {
  b <- 1 / (1 + exp(-t %*% x))
  return(b)
}

s <- function(data, w) {
  learn <- c(rep(0, 64))
  for (i in 1:800) {
    temp <- as.numeric(data[i, 1:64])
    learn <- learn + (as.numeric(sigmoid(w, temp)) - data[i, 65]) * temp
  }
  return(learn)
}

w0 <- c(rep(0, 64))

# define a function to update w, with data set and number of iterations given
update_w <- function(data, num) {
  w <- w0
  for (i in 1:num) {
    w <- w - 0.01 * s(data, w) # set learning rate alpha to be 0.01
  }
  return(w)
}

# since it might take hundreds or thousands of steps to get a really 'fitting' w,
# we manually set number of iterations, say 500
wfit <- update_w(train, 500)
wfit
```

```r
# 1.1 Evaluation of logistic

logp_logistic_reg <- function(t, x) {
  if (t == 0) {
    l <- log(1 - sigmoid(wfit, x))
  } else {l <- log(sigmoid(wfit, x))}
  return(l)
}

logp_logistic_train <- matrix(nrow = 800, ncol = 2)
colnames(logp_logistic_train) <- c("logp0", "logp1")
for (i in 1:2) {
  for (j in 1:800) {
    logp_logistic_train[j, i] <- logp_logistic_reg(i-1, as.numeric(train[j, 1:64]))
  }
}

# logp_logistic_train

logp_logistic_test <- matrix(nrow = 400, ncol = 2)
colnames(logp_logistic_test) <- c("logp0", "logp1")
for (i in 1:2) {
  for (j in 1:400) {
    logp_logistic_test[j, i] <- logp_logistic_reg(i-1, as.numeric(test[j, 1:64]))
  }
}

# logp_logistic_test

# 1.2 Average of logistic

train_0_logistic <- mean(logp_logistic_train[1:400, 1])
train_1_logistic <- mean(logp_logistic_train[401:800, 2])
train_logistic <- mean(c(train_0_logistic, train_1_logistic))
test_0_logistic <- mean(logp_logistic_test[1:200, 1])
test_1_logistic <- mean(logp_logistic_test[201:400, 2])
test_logistic <- mean(c(test_0_logistic, test_1_logistic))

print(c(train_0_logistic, train_1_logistic, train_logistic))
print(c(test_0_logistic, test_1_logistic, test_logistic))

# 1.3 Classify

train_logistic_pre <- c(1:800)
test_logistic_pre <- c(1:400)
```

```r
for (i in 1:800) {
  if (logp_logistic_train[i, 1] > logp_logistic_train[i, 2]) {
    train_logistic_pre[i] <- 0
  } else {
    train_logistic_pre[i] <- 1
  }
}

for (i in 1:400) {
  if (logp_logistic_test[i, 1] > logp_logistic_test[i, 2]) {
    test_logistic_pre[i] <- 0
  } else {
    test_logistic_pre[i] <- 1
  }
}

# 1.4 Error counting and error rate

train_3_e <- sum(abs(train[1:400, 65] - train_logistic_pre[1:400]))
train_5_e <- sum(abs(train[401:800, 65] - train_logistic_pre[401:800]))
train_e <- train_3_e + train_5_e
test_3_e <- sum(abs(test[1:200, 65] - test_logistic_pre[1:200]))
test_5_e <- sum(abs(test[201:400, 65] - test_logistic_pre[201:400]))
test_e <- test_3_e + test_5_e

print(c(train_3_e, train_3_e/400, train_5_e, train_5_e/400, train_e, train_e/800))
print(c(test_3_e, test_3_e/200, test_5_e, test_5_e/200, test_e, test_e/400))


# 2. GAUSSIAN-CONDITIONAL

## setup based on training set
zero <- train[train[, 65] == 0, ]
one <- train[train[, 65] == 1, ]
N0 <- nrow(zero)
N1 <- nrow(one)
pi0 <- N0 / N
pi1 <- N1 / N
mu0 <- sapply(zero[, 1:64], mean)
mu1 <- sapply(one[, 1:64], mean)
s0 <- var(zero[, 1:64]) * (N0 - 1) / N0 #Sigma
s1 <- var(one[, 1:64]) * (N1 - 1) / N1 #Sigma

# 2.1 Evaluation of Gaussian-conditional
# use log-posterior function
```

```r
lnp_x <- function(x, mu, sigma) {
  d <- length(mu)
  lnp <- -(0.5 * d) * log(2*pi) - 0.5 * log(det(sigma)) - 0.5 * t(x - mu) %*% solve(sig
  return(lnp)
}

logp_gauss <- function (t, x) {
  temp <- lnp_x(x, mu0, s0) + log(pi0) - lnp_x(x, mu1, s1) - log(pi1)
  if (t == 0) {
    lnp <- -log((1 + exp(-temp))) # this is actually a sigmoid function, but we don't c
  } else {
    lnp <- -log((1 + exp(temp)))
  }
  return(lnp)
}

logp_gauss_train <- matrix(nrow = 800, ncol = 2) # just like before
colnames(logp_gauss_train) <- c("logp0", "logp1")
for (i in 1:2) {
  for (j in 1:800) {
    logp_gauss_train[j, i] <- logp_gauss(i-1, as.numeric(train[j, 1:64]))
  }
}

# logp_gauss_train

logp_gauss_test <- matrix(nrow = 400, ncol = 2)
colnames(logp_gauss_test) <- c("logp0", "logp1")
for (i in 1:2) {
  for (j in 1:400) {
    logp_gauss_test[j, i] <- logp_gauss(i-1, as.numeric(test[j, 1:64]))
  }
}

# logp_gauss_test

# 2.2 Average of Gaussian-conditional
train_0_g <- mean(logp_gauss_train[1:400, 1])
train_1_g <- mean(logp_gauss_train[401:800, 2])
train_g <- mean(c(train_0_g, train_1_g))
test_0_g <- mean(logp_gauss_test[1:200, 1])
test_1_g <- mean(logp_gauss_test[201:400, 2])
test_g <- mean(c(test_0_g, test_1_g))
```

```
print(c(train_0_g, train_1_g, train_g))
print(c(test_0_g, test_1_g, test_g))

# 2.3 Classify
train_gauss_pre <- c(1:800)
test_gauss_pre <- c(1:400)
for (i in 1:800) {
  if (logp_gauss_train[i, 1] > logp_gauss_train[i, 2]) {
    train_gauss_pre[i] <- 0
  } else {train_gauss_pre[i] <- 1}
}

for (i in 1:400) {
  if (logp_gauss_test[i, 1] > logp_gauss_test[i, 2]) {
    test_gauss_pre[i] <- 0
  } else {test_gauss_pre[i] <- 1}
}

# 2.4 Error counting and error rate

e_g_train_3 <- sum(abs(train[1:400, 65] - train_gauss_pre[1:400]))
e_g_train_5 <- sum(abs(train[401:800, 65] - train_gauss_pre[401:800]))
e_g_train <- e_g_train_3 + e_g_train_5
e_g_test_3 <- sum(abs(test[1:200, 65] - test_gauss_pre[1:200]))
e_g_test_5 <- sum(abs(test[201:400, 65] - test_gauss_pre[201:400]))
e_g_test <- e_g_test_3 + e_g_test_5

print(c(e_g_train_3, e_g_train_3/400, e_g_train_5,
e_g_train_5/400, e_g_train, e_g_train/800))
print(c(e_g_test_3, e_g_test_3/200, e_g_test_5,
e_g_test_5/200, e_g_test, e_g_test/400))

# 3. REGULARIZED GAUSSIAN-CONDITIONAL

# Basically the same settings for this, except the fact that sigmas' are different due
s0_r <- s0 + 0.01 * diag(64)
s1_r <- s1 + 0.01 * diag(64)

# 3.1 Evaluation of regularized Gaussian-conditional

logp_r <- function (t, x) {
  temp <- lnp_x(x, mu0, s0_r) + log(pi0) - lnp_x(x, mu1, s1_r) - log(pi1)
  if (t == 0) {
    lnp <- -log((1 + exp(-temp)))
  } else {
```

```
    lnp <- -log((1 + exp(temp)))
  }
  return(lnp)
}

logp_r_train <- matrix(nrow = 800, ncol = 2)
colnames(logp_r_train) <- c("logp0", "logp1")
for (i in 1:2) {
  for (j in 1:800) {
    logp_r_train[j, i] <- logp_r(i-1, as.numeric(train[j, 1:64]))
  }
}

# logp_r_train

logp_r_test <- matrix(nrow = 400, ncol = 2)
colnames(logp_r_test) <- c("logp0", "logp1")
for (i in 1:2) {
  for (j in 1:400) {
    logp_r_test[j, i] <- logp_r(i-1, as.numeric(test[j, 1:64]))
  }
}

# logp_gauss_test

# 3.2 Average of regularized Gaussian-conditional
train_0_r <- mean(logp_r_train[1:400, 1])
train_1_r <- mean(logp_r_train[401:800, 2])
train_r <- mean(c(train_0_r, train_1_r))
test_0_r <- mean(logp_r_test[1:200, 1])
test_1_r <- mean(logp_r_test[201:400, 2])
test_r <- mean(c(test_0_r, test_1_r))

print(c(train_0_r, train_1_r, train_r))
print(c(test_0_r, test_1_r, test_r))

# 3.3 Classify

train_r_pre <- c(1:800)
test_r_pre <- c(1:400)
for (i in 1:800) {
  if (logp_r_train[i, 1] > logp_r_train[i, 2]) {
    train_r_pre[i] <- 0
  } else {train_r_pre[i] <- 1}
}
```

```
for (i in 1:400) {
  if (logp_r_test[i, 1] > logp_r_test[i, 2]) {
    test_r_pre[i] <- 0
  } else {test_r_pre[i] <- 1}
}

# 3.4 Error counting and error rate

e_r_train_3 <- sum(abs(train[1:400, 65] - train_r_pre[1:400]))
e_r_train_5 <- sum(abs(train[401:800, 65] - train_r_pre[401:800]))
e_r_train <- e_r_train_3 + e_r_train_5
e_r_test_3 <- sum(abs(test[1:200, 65] - test_r_pre[1:200]))
e_r_test_5 <- sum(abs(test[201:400, 65] - test_r_pre[201:400]))
e_r_test <- e_r_test_3 + e_r_test_5

print(c(e_r_train_3, e_r_train_3/400, e_r_train_5,
e_r_train_5/400, e_r_train, e_r_train/800))
print(c(e_r_test_3, e_r_test_3/200, e_r_test_5,
e_r_test_5/200, e_r_test, e_r_test/400))
```