# STAT 775: Machine Learning
# HW 05

Terence Henriod

March 8, 2015

**Abstract**

In this assignment, logistic regression and neural networks of single neurons are used for simple digit image classification.

# 1 Exercise 01

## 1.1 Problem Statement

Using the zip code digit data from the ESL website, use Logistic Regression to perform binary classification between 2s and 3s, 4s and 5s, and 7s and 9s.

## 1.2 Results

The classification results are as follows:

| | Classified as 2 | Classified as 3 | % Correct |
|---|---|---|---|
| Actual class 2 | 188 | 10 | 94.9 |
| Actual class 3 | 6 | 160 | 96.4 |
| Overall Correct Prediction Rate | | | 95.6 |

| | Classified as 4 | Classified as 5 | % Correct |
|---|---|---|---|
| Actual class 4 | 198 | 2 | 99.0 |
| Actual class 5 | 6 | 154 | 96.3 |
| Overall Correct Prediction Rate | | | 97.8 |

| | Classified as 7 | Classified as 9 | % Correct |
|---|---|---|---|
| Actual class 7 | 139 | 8 | 94.6 |
| Actual class 9 | 4 | 173 | 97.7 |
| Overall Correct Prediction Rate | | | 96.3 |

## 1.3 Code

The following R code was used to perform the LDA and classification:

```
#
# Initial Setup
#
setwd("C:/Users/Terence/Documents/GitHub/STAT775/HW05")

#
# Data Cleaning
#
DATA.PATH <- "../DataSets/zip.data/"
ZIP.TRAIN.FILE.NAME <- paste0(DATA.PATH, "zip.train")
ZIP.TEST.FILE.NAME <- paste0(DATA.PATH, "zip.test")

create.data.tuple <- function(file.path.name, class0, class1) {
  data <- read.table(file.path.name)
  data <- subset(data, V1 == class0 | V1 == class1)

  labels <- matrix(0, nrow(data), 1)
  for (i in 1:nrow(data)) {
    if (data[i, 1] == class1) {
      labels[i, 1] <- 1.0
    }
  }

  data <- data[, -1]
```

```
  data.tuple <- list(
    labels = data.matrix(labels),
    observations = data.matrix(data)
  )
  return(data.tuple)
}

train.data <- create.data.tuple(ZIP.TRAIN.FILE.NAME, 4, 5)
test.data <- create.data.tuple(ZIP.TEST.FILE.NAME, 4, 5)


#
# Logistic Regression Pieces
#
logistic.predict <- function(x, b) {
  #
  # Args:
  #   x: a p x 1 column vector of observation values
  #   b: a p X 1 column vector of model weights (betas)

  e <- exp(t(b) %*% rbind(1.0, x))
  return(e / (1.0 + e))
}

newton.raphson.iteration <- function(X, y, old.betas, step.size = 0.5) {
  #
  # Args:
  #   X: N x p matrix of observations
  #   y: column vector of N target labels
  #   old.betas: column vector of (p + 1) model parameters
  #   step.size: a rate controlling parameter

  p <- matrix(0, nrow = nrow(y), ncol = 1)
  for (i in 1:nrow(y)) {
    p[i, 1] <- logistic.predict(x = data.matrix(X[i, ]), b = old.betas)
  }

  W <- diag(1, nrow(y), nrow(y))
  for (i in 1:nrow(y)) {
    W[i, i] <- p[i, 1] * (1.0 - p[i, 1])
  }

  X.b <- cbind(1.0, X)  # X with 1s prepended for bias term

  z <- (X.b %*% old.betas) +
       (solve(W + diag(0.0001, nrow(W), ncol(W))) %*% (y - p))

  step <- solve((t(X.b) %*% W %*% X.b) + diag(0.0001, ncol(X.b), ncol(X.b))) %*%
          t(X.b) %*% (y - p)
  step <- step * step.size

  return(old.betas + step)
}
```

```
train.logistic.model <- function(X, y, n.iterations = 75, step.size = 0.5) {
  #
  # Args:
  #   X: N x p matrix of observations, observations are rows
  #   y: N x 1 column vector of class labels {0, 1}
  #   n.iterations: the number of approximation iterations to perform
  #   step.size: used to slow the rate of convergence to prevent overshooting

  # start with model parameters at 0 - arbitrary starting point
  betas <- matrix(0, nrow = ncol(X) + 1, 1)

  # iterate until parameter updates become arbitrarily small
  old.betas <- betas + 1.0
  i <- 0
  while (i < n.iterations) {
    betas <- newton.raphson.iteration(
      X = X,
      y = y,
      old.betas = betas,
      step.size = step.size
    )

    # TODO: use method for finding convergence, not just arbitrary iterations

    i <- i + 1
  }

  return(betas)
}

#
# Testing
#
test.cases <- list(
  list(2, 3),
  list(4, 5),
  list(7, 9)
)

for (case in test.cases) {
  train <- create.data.tuple(ZIP.TRAIN.FILE.NAME, case[[1]], case[[2]])
  model.betas <- train.logistic.model(train$observations, train$labels)

  test <- create.data.tuple(ZIP.TEST.FILE.NAME, case[[1]], case[[2]])
  confusion.matrix <- matrix(0, 2, 2 + 1)
  rownames(confusion.matrix) <- c(case[[1]], case[[2]])
  colnames(confusion.matrix) <- c(case[[1]], case[[2]], '%.Correct')
  correct <- 0

  for (i in 1:nrow(test$labels)) {
    prediction <- logistic.predict(
      x = data.matrix(test$observations[i, ]),
      b = model.betas
```

```
    )
    prediction <- round(prediction, 0)

    if (test$labels[[i]] == 0) {
      if (prediction == 0) {
        confusion.matrix[1, 1] <- confusion.matrix[1, 1] + 1
      } else {
        confusion.matrix[1, 2] <- confusion.matrix[1, 2] + 1
      }
    } else {
      if (prediction == 0) {
        confusion.matrix[2, 1] <- confusion.matrix[2, 1] + 1
      } else {
        confusion.matrix[2, 2] <- confusion.matrix[2, 2] + 1
      }
    }

    if (prediction == test$labels[[i]]) {
      correct <- correct + 1
    }
  }
  print (correct / nrow(test$labels))
  confusion.matrix[1, 3] <- confusion.matrix[1, 1] /
                            (confusion.matrix[1, 1] + confusion.matrix[1, 2])
  confusion.matrix[2, 3] <- confusion.matrix[2, 2] /
                            (confusion.matrix[2, 1] + confusion.matrix[2, 2])
  print(confusion.matrix)
}
```

# 2 Exercise 02

## 2.1 Problem Statement

Using the zip code digit data from the ESL website, use Single Neuron Neural Networks to perform binary classification between 2s and 3s, 4s and 5s, and 7s and 9s.

## 2.2 Results

The classification results (with 1000 training iterations and a step size of 0.1) are as follows:

| | Classified as 2 | Classified as 3 | % Correct |
|---|---|---|---|
| Actual class 2 | 191 | 7 | 96.5 |
| Actual class 3 | 4 | 162 | 97.6 |
| Overall Correct Prediction Rate | | | 97.0 |

| | Classified as 4 | Classified as 5 | % Correct |
|---|---|---|---|
| Actual class 4 | 194 | 6 | 97.0 |
| Actual class 5 | 4 | 156 | 97.5 |
| Overall Correct Prediction Rate | | | 97.2 |

| | Classified as 7 | Classified as 9 | % Correct |
|---|---|---|---|
| Actual class 7 | 138 | 9 | 93.9 |
| Actual class 9 | 4 | 173 | 97.7 |
| Overall Correct Prediction Rate | | | 96.0 |

## 2.3 Code

The following R code was used to perform the LDA and classification:

```
#
# Initial Setup
#
setwd("C:/Users/Terence/Documents/GitHub/STAT775/HW05")

#
# Data Cleaning
#
DATA.PATH <- "../DataSets/zip.data/"
ZIP.TRAIN.FILE.NAME <- paste0(DATA.PATH, "zip.train")
ZIP.TEST.FILE.NAME <- paste0(DATA.PATH, "zip.test")

create.data.tuple <- function(file.path.name, class0, class1) {
  data <- read.table(file.path.name)
  data <- subset(data, V1 == class0 | V1 == class1)

  labels <- matrix(0, nrow(data), 1)
  for (i in 1:nrow(data)) {
    if (data[i, 1] == class1) {
      labels[i, 1] <- 1.0
    }
  }

  data <- data[, -1]
```

```
  data.tuple <- list(
    labels = data.matrix(labels),
    observations = data.matrix(data)
  )
  return(data.tuple)
}

#
# Neuron Structure
#
create.neuron <- function(dim, activation.f, activation.f.derivative) {
#
# Args:
#   dim: an integer dimensionality of the inputs (do not include bias)
#   activation.f: the activation function of the neuron
#   activation.f.derivative: a the function that is the derivative of activation.f

  w <- matrix(rand(), dim + 1, 1)

  return(list(
    w = w,
    f = activation.f,
    f.prime = activation.f.derivative,
    output = 0,
    error.prime = 9999
  ))
}


sigmoid.f <- function(x) {
  return(1.0 / (1.0 + exp(-x)))
}

sigmoid.f.prime <- function(x) {
  return(sigmoid.f(x) * (1.0 - sigmoid.f(x)))
}

weighted.sum.inputs <- function(b, x) {
#
# Args:
#   x: a p x 1 column vector of feature values
#   b: a (p + 1) x 1 column vector of input weights
  return((t(b[-1, ]) %*% x) - b[1,1])
}


#
#
#

test.cases <- list(
  list(2, 3),
  list(4, 5),
  list(7, 9)
```

```
)

case <- test.cases[[1]]
i <- 1
learn.rate <- 0.1

for (case in test.cases) {
  train.data <- create.data.tuple(ZIP.TRAIN.FILE.NAME, case[[1]], case[[2]])
  W <- matrix(runif(ncol(train.data$observations) + 1))

  k <- 1000
  while (k > 0) {
    for (i in 1:nrow(train.data$labels)) {
      x <- data.matrix(train.data$observations[i, ])
      y <- train.data$labels[[i]]
      input <- weighted.sum.inputs(x = x, b = W)
      o <- sigmoid.f(input)
      e <- y - o
      W <- W + learn.rate * (rep(((o * (1.0 - o)) * e), 257) * rbind(1.0, x))




#       D <- sigmoid.f.prime(input) * e
#       W <- W - rep((learn.rate * o * D), 257)
    }

    k <- k - 1
  }

  test.data <- create.data.tuple(ZIP.TEST.FILE.NAME, case[[1]], case[[2]])
  confusion.matrix <- matrix(0, 2, 2 + 1)
  rownames(confusion.matrix) <- c(case[[1]], case[[2]])
  colnames(confusion.matrix) <- c(case[[1]], case[[2]], '%.Correct')
  correct <- 0

  for (i in 1:nrow(test.data$labels)) {
    x <- data.matrix(test.data$observations[i, ])
    prediction <- round(sigmoid.f(weighted.sum.inputs(x = x, b = W)))

    if (test.data$labels[[i]] == 0) {
      if (prediction == 0) {
        confusion.matrix[1, 1] <- confusion.matrix[1, 1] + 1
      } else {
        confusion.matrix[1, 2] <- confusion.matrix[1, 2] + 1
      }
    } else {
      if (prediction == 0) {
        confusion.matrix[2, 1] <- confusion.matrix[2, 1] + 1
      } else {
        confusion.matrix[2, 2] <- confusion.matrix[2, 2] + 1
      }
    }
```

```
    if (prediction == test.data$labels[[i]]) {
      correct <- correct + 1
    }
  }
  print(correct / nrow(test.data$labels))
  confusion.matrix[1, 3] <- confusion.matrix[1, 1] /
                            (confusion.matrix[1, 1] + confusion.matrix[1, 2])
  confusion.matrix[2, 3] <- confusion.matrix[2, 2] /
                            (confusion.matrix[2, 1] + confusion.matrix[2, 2])
  print(confusion.matrix)
}
```