STAT 775: Machine Learning
HW 07

Terence Henriod

April 14, 2015

Abstract

In this assignment, we use random forrests to classify hand-written digits that are typically hard to discern from one another for classifiers.

1 Exercise 01

1.1 Problem Statement

Using the zip.data data set from the ESL website, reduce the dimensionality to 40 with PCA, and perform classification using "a collection of random trees." Perform binary classification on 7s vs. 9s, 3s vs. 5s, and 0s vs. 8s.

1.2 Difficulty In Implementation

I had a lot of difficulty implementing a good random tree classifier. The runtime was awful (10 minutes to train 3 trees), the trees were large (greater tree depth than number of training examples), and classification performance was poor (70-80%). I am not sure what went wrong, so for a sanity check I made use of the R package `randomForest`. It operates much faster and with much better classification accuracy. Both sets of results are shown below. Unfortunately, at this time, I may not be able to spend the time to make a fully effective implementation, and I figure that my time is better spent moving on to final projects. If using a library is insufficient and I should redo this for full credit, please let me know.

1.3 randomForest Package Results

1.3.1 The Tables

The confusion matrices for classification are shown below.

7s vs. 9s:

Actual/Prediction	7	9	% Correct
7	140	7	95.2
9	3	174	98.3
Overall			96.9

3s vs. 5s:

Actual/Prediction	3	5	% Correct
3	148	18	89.1
5	10	150	93.7
Overall			91.4

0s vs. 8s:

Actual/Prediction	0	8	% Correct
0	356	3	99.1
8	9	157	94.6
Overall			97.7

1.3.2 The Code

The following R code was used to perform classification:

```
library('randomForest')

tests <- list(
  list(7, 9),
```

```
  list(3, 5),
  list(0, 8)
)

for (test in tests) {
  data <- get.data(
    train.file = ZIP.TRAIN.FILE.NAME,
    test.file = ZIP.TEST.FILE.NAME,
    class.1 = 0,
    class.2 = 8,
    num.principle.components = 50
  )

  rf <- randomForest(x = data$train[, -(1:2)], y = data$train[, 1])
  data$test$prediction <- round(predict(rf, data$test[, -(1:2)]))

  print(compute.confusion.matrix(
    predictions = data$test$prediction,
    targets = data$test$target,
    num.classes = 2
  )
}
```

1.4 My Results

1.4.1 The Tables

The confusion matrices for classification are shown below.

7s vs. 9s:

Actual/Prediction	7	9	% Correct
7	109	38	74.1
9	74	103	58.1
Overall			65.4

3s vs. 5s:

Actual/Prediction	3	5	% Correct
3	115	51	69.3
5	87	73	45.6
Overall			57.7

0s vs. 8s:

Actual/Prediction	0	8	% Correct
0	336	23	93.6
8	72	94	56.6
Overall			81.9

1.4.2 The Code

The following R code was used to perform classification:

```
##############
# STAT775: Machine Learning
#
# HW07
# Exercise 01
#
# Using the zip.data data set from the ESL website, reduce the dimensionality
# to $40$ with PCA, and perform classification using ‘‘a collection of random
# trees." Perform binary classification on 7s vs. 9s, 3s vs. 5s, and 0s vs. 8s.
###############

#
# Initial Setup
#
setwd("C:/Users/Terence/Documents/GitHub/STAT775/HW07")

#
# 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")

construct.classification.frame <- function(data, targets) {
  return(
    data.frame(
      'target' = targets,
      'prediction' = rep(0, length(targets)),
      data
    )
  )
}

get.data <- function(
  train.file, test.file, class.1, class.2, num.principle.components = 50) {
  # Args:
  #   train.file:
  #   test.file:
  #   class.1:
  #   class.2
  #   num.principle.components:
  #

  train <- read.table(train.file)
  train <- subset(train, V1 == class.1 | V1 == class.2)
  for (i in 1:nrow(train)) {
    train[i, 1] <- if (train[i, 1] == class.1) {1} else {2}
  }

  test <- read.table(test.file)
  test <- subset(test, V1 == class.1 | V1 == class.2)
  for (i in 1:nrow(test)) {
    test[i, 1] <- if (test[i, 1] == class.1) {1} else {2}
  }
```

```
  pca.model <- prcomp(train[, -1])

  train[, -1] <- predict(pca.model, train[, -1])
  train <- train[, 1:(num.principle.components + 1)]
  train <- construct.classification.frame(
    targets = train[, 1],
    data = train[, -1]
  )

  test[, -1] <- predict(pca.model, test[, -1])
  test <- test[, 1:(num.principle.components + 1)]
  test <- construct.classification.frame(
    targets = test[, 1],
    data = test[, -1]
  )

  return(list(
    train = train,
    test = test
  ))
}

#
# Data Presentation and Evaluation
#

compute.confusion.matrix <- function(predictions, targets, num.classes = 2) {
  confusion.matrix <- matrix(0, nrow = num.classes + 1, ncol = num.classes + 1)
  for (i in 1:length(predictions)) {
    confusion.matrix[targets[[i]], predictions[[i]]] <-
      confusion.matrix[targets[[i]], predictions[[i]]] + 1
  }
  confusion.matrix[num.classes + 1, num.classes + 1] <-
    sum(diag(confusion.matrix)) / sum(confusion.matrix)
  for (i in 1:num.classes) {
    confusion.matrix[i, num.classes + 1] <-
      confusion.matrix[i, i] / sum(confusion.matrix[i, 1:num.classes])
    confusion.matrix[num.classes + 1, i] <-
      confusion.matrix[i, i] / sum(confusion.matrix[1:num.classes, i])
  }
  return(confusion.matrix)
}


#
# Random Discriminant
#
require(ppls)

fit.gaussian <- function(x, n) {
  # Args:
  #   x: a list or 1-dimensional vector
  #   n: the total number of observations in the set the members of x come from
```

```
  gaussian.model <- list(
    mu = mean(x),
    sigma = var(x),
    prior = length(x) / n
  )

  if (gaussian.model$prior == 0) {
    gaussian.model$mu <- 1
    gaussian.model$sigma <- 1
  } else if (is.na(gaussian.model$sigma)) {
    gaussian.model$sigma <- 1  # arbitrary, hack for when fittint to 1 element
  }

  return(gaussian.model)
}

relative.gaussian.pdf <- function(model, x) {
  # Args:
  #   model:
  #   x:

  if (model$prior == 0) {
    return(0)
  }

  scaling.factor <- 1 / (sqrt(2 * pi) * model$sigma)
  exponent <- -(((x - model$mu) ^ 2) / (2 * model$sigma ^ 2))
  probability <- scaling.factor * exp(exponent)

if (any(is.na(probability))) {
  print(model)
  print(x)
}

  return(probability)
}

construct.random.discriminant <- function(classification.frame) {
  # Args:
  #   data: labeled data in rows

  dimensionality <- ncol(classification.frame) - 2

  projection.line <- normalize.vector(
    matrix(runif(n = dimensionality, min = -5, max = 5), ncol = 1)
  )

  data.c1 <- data.matrix(subset(classification.frame, target == 1)[, -(1:2)])
  data.c2 <- data.matrix(subset(classification.frame, target == 2)[, -(1:2)])

  projections.c1 <- data.c1 %*% projection.line
  projections.c2 <- data.c2 %*% projection.line
```

```
  model <- list(
    projection.line = projection.line,
    decision.boundary = NULL,
    class.1.model = NULL,
    class.2.model = NULL
  )

  if (nrow(data.c1) <= 1 || nrow(data.c1) <= 1) {
    model$decision.boundary <-
      0.5 * (mean(projections.c1) + mean(projections.c2))
  }

  model$class.1.model <- fit.gaussian(
    x = projections.c1,
    n = nrow(classification.frame)
  )
  model$class.2.model <- fit.gaussian(
    x = projections.c2,
    n = nrow(classification.frame)
  )

  return(model)
}


discriminant.predict <- function(model, c.frame) {
  # Args:
  #   model: an object that represents an LDA model
  #   x: the data to be classified via LDA

  if (nrow(c.frame) > 0) {
    if (is.null(model$decision.boundary)) {
      predictions <- list()
      for (i in 1:nrow(c.frame)) {
        projection <- data.matrix(c.frame[i, -(1:2)]) %*% model$projection.line
        p.class.1 <- relative.gaussian.pdf(
          model = model$class.1.model,
          x = projection
        )
        p.class.2 <- relative.gaussian.pdf(
          model = model$class.2.model,
          x = projection
        )

        predictions[[i]] <- if (p.class.1 >= p.class.2) {1} else {2}
      }
      c.frame$prediction <- predictions
    } else {
      for(i in 1:nrow(c.frame)) {
        projection <- data.matrix(c.frame[i, -(1:2)]) %*% model$projection.line
        if (projection <= model$decision.boundary) {
          c.frame$prediction[[i]] <- 1
        } else {
          c.frame$prediction[[i]] <- 2
```

```
      }
    }
  }
}

  return(c.frame)
}


#
# Tree
#
compute.entropy <- function(c.frame) {
  # Args:
  #   c.frame:

  n <- nrow(c.frame)

  if (n == 0) {
    return(0)
  }

  p.1 <- (nrow(subset(c.frame, target == 1))) / n
  p.2 <- 1 - p.1

  if (p.1 == 0 || p.2 == 0) {
    return(0)
  }

  entropy <- -((p.1 * log2(p.1)) + (p.2 * log2(p.2)))
  return(entropy)
}

construct.random.tree <- function(c.frame, entropy.threshold = 0.1) {
  # Args:
  #   data: a frame of labeled data
  #   stopping.criterion: a function that will determine if the tree should
  #                       continue building. Could be entropy, gini index, etc.

  entropy <- compute.entropy(c.frame)

  if (nrow(c.frame) <= 1 || entropy <= entropy.threshold){
    return(NULL)
  }

  discriminant.model <- construct.random.discriminant(c.frame)

  c.frame <- discriminant.predict(
    model = discriminant.model,
    c.frame = c.frame
  )

  predicted.as.1 <- subset(c.frame, prediction == 1)
  predicted.as.1$prediction <- rep(0, nrow(predicted.as.1))
```

```
  class.1.branch <- construct.random.tree(c.frame = predicted.as.1)

  predicted.as.2 <- subset(c.frame, prediction == 2)
  predicted.as.2$prediction <- rep(0, nrow(predicted.as.2))
  class.2.branch <- construct.random.tree(c.frame = predicted.as.2)

  tree <- list(
    model = discriminant.model,
    entropy = entropy,
    class.1.branch = class.1.branch,
    class.2.branch = class.2.branch
  )

  return(tree)
}

tree.predict <- function(tree, c.frame) {
  if (!is.null(tree)) {
    c.frame <- discriminant.predict(tree$model, c.frame)

    predicted.as.1 <- subset(c.frame, prediction == 1)
    predicted.as.2 <- subset(c.frame, prediction == 2)

    c.frame <- rbind(
      tree.predict(tree$class.1.branch, predicted.as.1),
      tree.predict(tree$class.2.branch, predicted.as.2)
    )
  }
  return(c.frame)
}


#
# Main
#
library('randomForest')

tests <- list(
  list(7, 9),
  list(3, 5),
  list(0, 8)
)

for (test in tests) {
  data <- get.data(
    train.file = ZIP.TRAIN.FILE.NAME,
    test.file = ZIP.TEST.FILE.NAME,
    class.1 = 0,
    class.2 = 8,
    num.principle.components = 50
  )

  rf <- randomForest(x = data$train[, -(1:2)], y = data$train[, 1])
  data$test$prediction <- round(predict(rf, data$test[, -(1:2)]))
```

```
  print(compute.confusion.matrix(
    predictions = data$test$prediction,
    targets = data$test$target,
    num.classes = 2
  ))
}

run.trial <- function(
  class.1, class.2, num.features = 50, num.trees = 5, entropy.threshold = 0.1) {

  print('Fetching Data...')
  data <- get.data(
    train.file = ZIP.TRAIN.FILE.NAME,
    test.file = ZIP.TEST.FILE.NAME,
    class.1 = class.1,
    class.2 = class.2,
    num.principle.components = num.features
  )

  print('Constructing Forrest...')
  forrest <- list()
  for (i in 1:num.trees) {
    forrest[[i]] <- construct.random.tree(
      c.frame = data$train,
      entropy.threshold = entropy.threshold
    )
  }

  print('Making Predictions...')
  predictions <- list()
  for (i in 1:num.trees) {
    predictions[[i]] <- tree.predict(forrest[[i]], data$test)[, 1:2]
  }

  print('Tabulating Votes...')
  votes.1 <- rep(0, nrow(predictions[[1]]))
  votes.2 <- rep(0, nrow(predictions[[1]]))
  for (j in 1:nrow(predictions[[1]])) {
    for (i in 1:num.trees) {
      if (predictions[[i]]$prediction[[j]] == 1) {
        votes.1[[j]] <- votes.1[[j]] + 1
      } else {
        votes.2[[j]] <- votes.2[[j]] + 1
      }
    }
  }

  consensus <- list()
  for (j in 1:length(votes.1)) {
    if (votes.1[[j]] >= votes.2[[j]]) {
      consensus[[j]] <- 1
    } else {
      consensus[[j]] <- 2
```

```
    }
  }

  print(
    compute.confusion.matrix(
      predictions = consensus,
      targets = predictions[[1]]$target)
  )
}

for (test in tests) {
  run.trial(
    class.1 = test[[1]],
    class.2 = test[[2]],
    num.features = 50,
    entropy.threshold = 0.001
  )
}
```