Data Mining: Assignment Two

Aaron Gonzales

October 24, 2014

1 Neural Network

Reading the data, optimizing for speed:

```
tab5rows <- read.table("~/Dropbox/cs/datamining/data-mining-is-fun/assignments/two/data.txt", header = FALSE, nr
classes <- sapply(tab5rows, class)
data <- read.table("~/Dropbox/cs/datamining/data-mining-is-fun/assignments/two/data.txt", header = FALSE, colCla
```

(a)

Using the nnet package and a small function to tabulate our predictions.

```
library(nnet)
nrow(data)

## [1] 10000

#Define class 'labels' for our set.
targets <- class.ind(c(rep('0', 5000), rep('1',5000)))
#test_data <- smalldata[sample(nrow(smalldata), 5000),]
test.cl <- function(true, pred) {
    actual <- max.col(true) - 1
    predicted <- max.col(pred) - 1
    table(actual, predicted)

}
```

```
smalldata <- subset(data, select = c(V1,V2, V3, V4))
targets <- class.ind(c(rep('0', 5000), rep('1',5000)))
#test_data <- smalldata[sample(nrow(smalldata), 5000),]
accuracy_rate <- NULL
for (i in 1:2) {
  train <- c(sample(1:5000, 450), sample(5000:10000,450))
  neural_net <- nnet(smalldata[train,], targets[train,], size=10, rang = 0.1, maxit =200, MaxNWts = 26000)
  a <- test.cl(targets[-train,], predict(neural_net, smalldata[-train,]))
  print(a)
  # error rate
  accuracy_rate[i] = 1-(a[1,2]+a[2,1]) / (a[1,1]+a[2,2])
  print("finished run ")
}

## # weights:  72
## initial  value 451.334452
## iter  10 value 327.221188
## iter  20 value 322.725060
## iter  30 value 235.049688
## iter  40 value 232.869533
## iter  50 value 230.007035
## iter  60 value 228.008672
## iter  70 value 228.007333
## iter  80 value 226.020105
## iter  90 value 226.007900
## iter 100 value 226.007554
```

```
## iter 110 value 224.008560
## iter 120 value 224.005990
## iter 130 value 224.005402
## iter 140 value 224.001424
## iter 150 value 222.064803
## iter 160 value 222.006044
## iter 170 value 222.005349
## iter 180 value 222.005230
## final  value 222.004955
## converged
##       predicted
## actual    0    1
##      0 3962  588
##      1  487 4063
## [1] "finished run "
## # weights:  72
## initial  value 458.135487
## iter  10 value 258.639580
## iter  20 value 251.627515
## iter  30 value 247.832327
## iter  40 value 246.249208
## iter  50 value 244.375720
## iter  60 value 241.291073
## iter  70 value 232.740171
## iter  80 value 228.119845
## iter  90 value 226.389288
## iter 100 value 224.882510
## iter 110 value 217.967684
## iter 120 value 214.202867
## iter 130 value 208.250834
## iter 140 value 169.769658
## iter 150 value 163.722617
## iter 160 value 159.576286
## iter 170 value 156.004103
## iter 180 value 153.376762
## iter 190 value 143.111016
## iter 200 value 140.891957
## final  value 140.891957
## stopped after 200 iterations
##       predicted
## actual    0    1
##      0 4109  441
##      1  740 3810
## [1] "finished run "

accuracy_rate

## [1] 0.8660 0.8509

mean(accuracy_rate)

## [1] 0.8585
```

```r
# Vector for accuracy rate on each run
accuracy_rate <- NULL
# train the network 10 times on 90 percent of each sample
for (i in 1:10) {
  # train is made from sampling at random from each half of the
  # data
  train <- c(sample(1:5000, 4500), sample(5000:10000,4500))
  neural_net <- nnet(data[train,], targets[train,],
                     size=10,
                     rang = 0.1,
                     maxit =100,
                     MaxNWts = 26000)
  # hold values in table and print them
  # tests prediction with the non-sampled remnants of the data
  a <- test.cl(targets[-train,], predict(neural_net, data[-train,]))
  print(a)
  # error rate
  accuracy_rate[i] = 1-(a[1,2]+a[2,1]) / (a[1,1]+a[2,2])
  print("finished run ")
}
```

```
## # weights:  10032
## initial  value 4519.044769
## iter  10 value 1920.675214
## iter  20 value 1821.884527
## iter  30 value 1815.212459
## iter  40 value 1805.378522
## iter  50 value 1801.001336
## iter  60 value 1800.994814
## iter  70 value 1800.992528
## iter  80 value 1800.987958
## iter  90 value 1800.974425
## iter 100 value 1800.787020
## final  value 1800.787020
## stopped after 100 iterations
##       predicted
## actual   0   1
##      0 497   2
##      1 100 401
## [1] "finished run "
## # weights:  10032
## initial  value 4538.520491
## iter  10 value 2644.230764
## iter  20 value 2643.225081
## final  value 2643.223756
## converged
##       predicted
## actual   0   1
##      0 460  40
##      1 105 396
## [1] "finished run "
## # weights:  10032
## initial  value 4505.217915
## iter  10 value 3840.630191
```

```
## final  value 3840.000021
## converged
##       predicted
## actual   0   1
##      0 359 141
##      1  69 432
## [1] "finished run "
## # weights:  10032
## initial  value 4484.199195
## iter  10 value 1817.144627
## iter  20 value 1804.726962
## iter  30 value 1018.263056
## iter  40 value 276.145805
## iter  50 value 149.648442
## iter  60 value 126.339842
## iter  70 value 121.059186
## iter  80 value 118.527459
## iter  90 value 112.441895
## iter 100 value 108.802717
## final  value 108.802717
## stopped after 100 iterations
##       predicted
## actual   0   1
##      0 456  44
##      1  18 483
## [1] "finished run "
## # weights:  10032
## initial  value 4551.293441
## iter  10 value 2640.951304
## iter  20 value 2595.035711
## iter  30 value 2581.910447
## iter  40 value 2571.735794
## iter  50 value 2161.942237
## iter  60 value 2140.505889
## iter  70 value 2119.834916
## iter  80 value 2116.977686
## iter  90 value 2116.945339
## iter 100 value 2113.720418
## final  value 2113.720418
## stopped after 100 iterations
##       predicted
## actual   0   1
##      0 479  21
##      1 105 396
## [1] "finished run "
## # weights:  10032
## initial  value 4541.914191
## iter  10 value 2404.984418
## iter  20 value 2401.999199
## iter  20 value 2401.999193
## iter  20 value 2401.999188
## final  value 2401.999188
## converged
##       predicted
```

```
## actual   0   1
##      0 473  27
##      1 122 379
## [1] "finished run "
## # weights:  10032
## initial  value 4521.727445
## final  value 2358.000000
## converged
##       predicted
## actual   0   1
##      0 459  41
##      1  98 403
## [1] "finished run "
## # weights:  10032
## initial  value 4548.811518
## final  value 2484.999902
## converged
##       predicted
## actual   0   1
##      0 460  40
##      1  94 406
## [1] "finished run "
## # weights:  10032
## initial  value 4601.836850
## iter  10 value 2265.309924
## iter  20 value 2238.066232
## iter  30 value 2196.172340
## iter  40 value 2187.630918
## iter  50 value 2149.644828
## iter  60 value 2148.036092
## iter  70 value 2147.035635
## iter  80 value 2139.833175
## iter  90 value 2138.220191
## iter 100 value 2121.847738
## final  value 2121.847738
## stopped after 100 iterations
##       predicted
## actual   0   1
##      0 476  24
##      1 108 393
## [1] "finished run "
## # weights:  10032
## initial  value 4565.683485
## final  value 2482.000989
## converged
##       predicted
## actual   0   1
##      0 439  61
##      1  83 418
## [1] "finished run "
```

```
# prints vector of rates
accuracy_rate
```

```
##  [1] 0.8864 0.8306 0.7345 0.9340 0.8560 0.8251 0.8387 0.8453 0.8481 0.8320
```

```
# mean of the accuracy rate
mean(accuracy_rate)

## [1] 0.8431
```

Neural network...

We use the caret (classification and regression tool) package with doMC(Parallelization for R) to train an svm (libsvm package) with a radial basis function.

```
library(caret)

## Loading required package:  lattice

library(doMC)

## Loading required package:  foreach
## Loading required package:  iterators
## Loading required package:  parallel

# class labels don't need to be dummy coded like they did for the aNN
data_svm <- data
labels <- c(rep(0,5000), rep(1,5000))
data_svm$labels <- labels
data_svm$labels <- as.factor(data_svm$labels)

# subsetting into train and test sets.
index <- 1:nrow(data_svm)
testindex <- sample(index, trunc(length(index)*30/100))
testset <- data_svm[testindex,]
trainset <- data_svm[-testindex,]

# training the model with a radial basis function.
# uses the lables ~ . to say 'the class labels are defined in the varible
# 'labels'. 10-fold cross validation is performed in the model and reported later and
# doesn't need to be done by hand.
system.time(
  model <- train(labels ~ .,
                 data = trainset,
                 method="svmRadial",
                 trControl=trainControl(method='cv', number = 10)
                 )
)

## Loading required package:  kernlab

##    user  system elapsed
## 1108.72   10.25 1123.19

print(model)

## Support Vector Machines with Radial Basis Function Kernel
##
## 7000 samples
## 1000 predictors
##    2 classes: '0', '1'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 6300, 6299, 6300, 6301, 6301, 6300, ...
##
## Resampling results across tuning parameters:
##
##   C    Accuracy  Kappa  Accuracy SD  Kappa SD
##   0.2  1         1      5e-04        9e-04
##   0.5  1         1      5e-04        9e-04
##   1.0  1         1      5e-04        9e-04
##
## Tuning parameter 'sigma' was held constant at a value of 0.0005534
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 0.0005534 and C = 0.25.
```

```
# prediction.
prediction <- predict(model, testset[,-1001])
tab <- table(pred = prediction, true = testset$labels)
plot(model)
confusionMatrix(model)
```

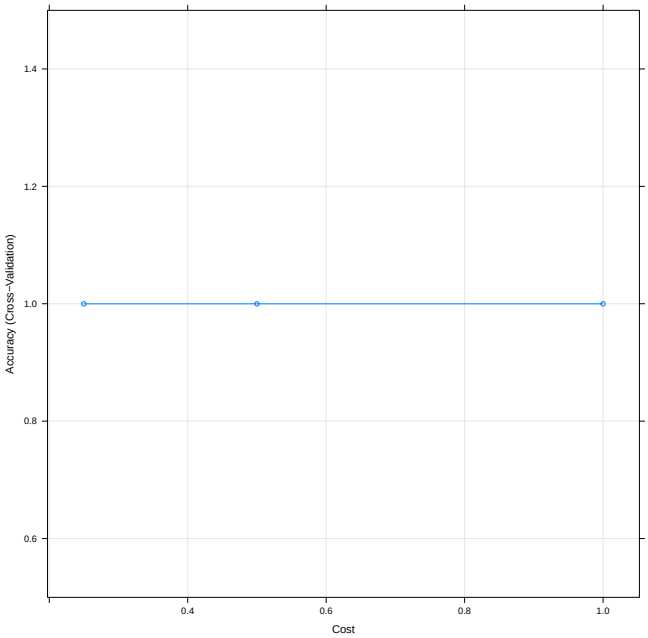
```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentages of table totals)
##
##           Reference
## Prediction    0    1
##          0 50.2  0.0
##          1  0.0 49.7
```

```
print(tab)
```

```
##     true
## pred    0    1
##    0 1482    0
##    1    0 1518
```

```
# model accuracy
print(1-(tab[1,2]+tab[2,1]) / (tab[1,1]+tab[2,2]))
```

```
## [1] 1
```




```
labels <- c(
  rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500),
        rep('0',500),
        rep('1',500)
)

targets <- class.ind(c(rep('0', 5000), rep('1',5000)))
```