Assignment39

John Bute

2024-11-19

R Markdown

5. Construct and evaluate ANN models for the Wines dataset

```
library(nnet)
Wine <- read.csv("C:/Users/johnb/Desktop/Machine Learning/data/wine.csv",
stringsAsFactors = TRUE)

set.seed(1)
Wine$Class = as.factor(Wine$Class)

index <- sample(1:nrow(Wine), 0.7 * nrow(Wine))
train_data <- Wine[index, ]
test_data <- Wine[-index, ]

ann_model <- nnet(Class ~ ., data = train_data, size = 10, maxit = 100)

## # weights:  173
## initial  value 215.368774
## iter  10 value 135.130137
## final  value 135.130134
## converged

summary(ann_model)

## a 13-10-3 network with 173 weights
## options were - softmax modelling
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1
##    0.54    0.00    0.53   -0.44    0.36    0.31    0.62    0.07    0.30   -0.16
## i10->h1 i11->h1 i12->h1 i13->h1
##   -0.56    0.60   -0.30    0.13
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
##   -0.55    0.48   -0.25    0.40   -0.33   -0.39    0.02   -0.32   -0.45    0.03
## i10->h2 i11->h2 i12->h2 i13->h2
##    0.09   -0.52   -0.34    0.31
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3
##    0.65   -0.55    0.37    0.63    0.45   -0.16    0.21    0.64    0.64   -0.22
## i10->h3 i11->h3 i12->h3 i13->h3
##   -0.32   -0.47   -0.25    1.99
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4
##    0.59    0.02   -0.34   -0.63   -0.12    0.50   -0.21   -0.52   -0.18    0.18
## i10->h4 i11->h4 i12->h4 i13->h4
##   -0.15    0.27    0.27    0.08
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5
##   -0.10   -0.07   -0.27    0.11    0.57   -0.50   -0.12   -0.40   -0.10   -0.51
## i10->h5 i11->h5 i12->h5 i13->h5
##   -0.06    0.62    0.37    0.61
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6  i9->h6
##   -0.04    0.15   -0.02   -0.55   -0.35    0.00   -0.18    0.61    0.03   -0.26
## i10->h6 i11->h6 i12->h6 i13->h6
##   -0.31    0.40    0.28   -0.47
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7  i9->h7
##   -0.61    0.36    0.17   -0.46   -0.61   -0.55   -0.17   -0.46   -0.28   -0.43
## i10->h7 i11->h7 i12->h7 i13->h7
```

```
##   -0.34   -0.45   -0.03    0.38
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8  i9->h8
##   -0.66    0.04    0.53   -0.18   -0.63   -0.51   -0.25   -0.48   -0.51   -0.39
## i10->h8 i11->h8 i12->h8 i13->h8
##   -0.38   -0.52    0.67   -0.24
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9  i9->h9
##    0.01    0.25   -0.56   -0.53   -0.63    0.60    0.24   -0.57   -0.01   -0.05
## i10->h9 i11->h9 i12->h9 i13->h9
##   -0.17    0.69   -0.45    0.44
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10
##    -0.60    -0.14    -0.50    -0.43     0.48     0.31    -0.33    -0.01
##  i8->h10  i9->h10 i10->h10 i11->h10 i12->h10 i13->h10
##    -0.58    -0.20     0.66     0.17     0.23    -0.26
##   b->o1  h1->o1  h2->o1  h3->o1  h4->o1  h5->o1  h6->o1  h7->o1  h8->o1  h9->o1
##   -0.70    0.13   -0.07    2.42   -0.13   -0.17   -0.32   -0.20    0.68   -0.86
## h10->o1
##   -0.14
##   b->o2  h1->o2  h2->o2  h3->o2  h4->o2  h5->o2  h6->o2  h7->o2  h8->o2  h9->o2
##    0.40   -0.63   -0.23    0.87   -0.52    0.08    0.66    0.65   -0.45    0.02
## h10->o2
##   -0.16
##   b->o3  h1->o3  h2->o3  h3->o3  h4->o3  h5->o3  h6->o3  h7->o3  h8->o3  h9->o3
##    0.85    0.28    0.56   -3.19    0.42    0.92   -0.02   -0.01    0.40    0.49
## h10->o3
##    0.67
```

```
ann_preds <- predict(ann_model, newdata = test_data, type = "class")
conf_matrix_ann <- table(Actual = test_data$Class, Predicted = ann_preds)
print(conf_matrix_ann)
```

```
##        Predicted
## Actual  2
##      1 20
##      2 22
##      3 12
```

```
test_error <- 1 - sum(diag(conf_matrix_ann)) / sum(conf_matrix_ann)
cat("Test Error Rate:", test_error, "\n")
```

```
## Test Error Rate: 0.6296296
```

Well, it seems that it predicts just one class, and failing to identify any other class. It is highly biased and no generalization at all.

6. Re-run the ANN models incorporating 10 hidden layers with 30 nodes. How much more time does it take to run a "bigger" neural network on the Wine dataset? Let us compare both model's run times:

```
library(nnet)
library(microbenchmark)

first_ann_benchmark <- microbenchmark(
ann_model <- nnet(Class ~ ., data = train_data, size = 10, maxit = 100),
times = 10L
)
```

```
## # weights:  173
## initial  value 144.310736
## iter  10 value 123.614907
## iter  20 value 100.890702
## iter  30 value 80.875859
## iter  40 value 74.174068
## iter  50 value 60.066808
## iter  60 value 43.865311
```

```
## iter  70 value 24.922794
## iter  80 value 19.438731
## iter  90 value 15.428540
## iter 100 value 13.573137
## final  value 13.573137
## stopped after 100 iterations
## # weights:  173
## initial  value 188.594100
## final  value 135.130134
## converged
## # weights:  173
## initial  value 173.203244
## final  value 135.130134
## converged
## # weights:  173
## initial  value 156.889627
## final  value 135.130133
## converged
## # weights:  173
## initial  value 135.589664
## final  value 135.130134
## converged
## # weights:  173
## initial  value 147.344042
## final  value 135.130134
## converged
## # weights:  173
## initial  value 150.832781
## final  value 135.130134
## converged
## # weights:  173
## initial  value 140.709882
## final  value 135.130134
## converged
## # weights:  173
## initial  value 209.059997
## final  value 135.130134
## converged
## # weights:  173
## initial  value 147.833100
## iter  10 value 121.007508
## iter  20 value 96.293464
## iter  30 value 96.276626
## final  value 96.276595
## converged
```

```
print(first_ann_benchmark)
```

```
## Unit: milliseconds
##                                                                    expr
##  ann_model <- nnet(Class ~ ., data = train_data, size = 10, maxit = 100)
##       min       lq     mean   median       uq     max neval
##  2.905001 3.004401 5.481871 3.180951 3.604901 21.9333    10
```

So, it takes about 13 miliseconds to run the smaller model. What about the bigger one

```
library(nnet)
library(microbenchmark)

big_ann_benchmark <- microbenchmark(
ann_model <- nnet(Class ~ ., data = train_data, size = 30, maxit = 500),
times = 10L
)
```

```
## # weights:  513
## initial  value 236.471417
## final  value 135.130134
```

```
## converged
## # weights:  513
## initial  value 239.262237
## final  value 135.130134
## converged
## # weights:  513
## initial  value 219.097102
## iter  10 value 131.872426
## iter  20 value 64.314334
## iter  30 value 60.431177
## iter  40 value 60.007864
## iter  50 value 57.829738
## iter  60 value 56.838110
## iter  70 value 56.757665
## iter  80 value 55.637476
## iter  90 value 55.356439
## iter 100 value 55.201881
## iter 110 value 54.092181
## iter 120 value 52.326306
## iter 130 value 52.018193
## iter 140 value 51.972187
## iter 150 value 51.767332
## iter 160 value 51.348508
## iter 170 value 51.185806
## iter 180 value 51.004440
## iter 190 value 47.748294
## iter 200 value 26.087473
## iter 210 value 9.441419
## iter 220 value 7.083028
## iter 230 value 4.409302
## iter 240 value 0.769179
## iter 250 value 0.009451
## iter 260 value 0.000605
## final  value 0.000085
## converged
## # weights:  513
## initial  value 390.963771
## iter  10 value 135.130134
## iter  10 value 135.130134
## iter  10 value 135.130134
## final  value 135.130134
## converged
## # weights:  513
## initial  value 141.371714
## iter  10 value 135.130140
## iter  20 value 103.961145
## iter  30 value 94.252781
## iter  40 value 94.222307
## final  value 94.222209
## converged
```

```
## # weights:  513
## initial  value 187.739133
## final  value 135.130134
## converged
## # weights:  513
## initial  value 188.967699
## iter  10 value 135.130134
## iter  10 value 135.130134
## iter  10 value 135.130134
## final  value 135.130134
## converged
## # weights:  513
## initial  value 322.617980
## iter  10 value 121.054351
## iter  20 value 68.800790
## iter  30 value 63.606752
## iter  40 value 61.971744
## iter  50 value 60.894231
## iter  60 value 59.264826
## iter  70 value 58.302398
## iter  80 value 57.930437
## iter  90 value 57.282789
## iter 100 value 56.092305
## iter 110 value 41.634976
## iter 120 value 16.695990
## iter 130 value 10.029500
## iter 140 value 5.437891
## iter 150 value 1.603264
## iter 160 value 0.109639
## iter 170 value 0.002275
## iter 180 value 0.000503
## iter 190 value 0.000239
## iter 200 value 0.000114
## final  value 0.000097
## converged
## # weights:  513
## initial  value 262.557645
## final  value 135.130134
## converged
## # weights:  513
## initial  value 159.330308
## final  value 135.130133
## converged

print(first_ann_benchmark)

## Unit: milliseconds
##                                                                    expr
##  ann_model <- nnet(Class ~ ., data = train_data, size = 10, maxit = 100)
```

```
##       min       lq     mean   median       uq     max neval
##  2.905001 3.004401 5.481871 3.180951 3.604901 21.9333    10
```

```
print(big_ann_benchmark)
```

```
## Unit: milliseconds
##                                                                      expr
##  ann_model <- nnet(Class ~ ., data = train_data, size = 30, maxit = 500)
##       min       lq     mean   median       uq      max neval
##  7.575001 8.698501 43.73688 9.478151 33.9459 192.1392    10
```

As we see, the mean time for the bigger model was 101 milliseconds, while for the smaller model, it was 14 milliseconds.

Let us see if the bigger model performs better

```
set.seed(123)
big_ann_model <- nnet(Class ~ ., data = train_data, size = 30, maxit = 500)
```

```
## # weights:  513
## initial  value 148.254270
## iter  10 value 133.479661
## iter  20 value 94.329901
## iter  30 value 81.278339
## iter  40 value 26.537377
## iter  50 value 25.660925
## iter  60 value 25.540541
## iter  70 value 22.992134
## iter  80 value 22.496830
## iter  90 value 20.180725
## iter 100 value 20.105207
## iter 110 value 20.090700
## iter 120 value 20.016665
## iter 130 value 20.013450
## iter 140 value 19.982257
## iter 150 value 19.839588
## iter 160 value 19.832572
## iter 170 value 19.824955
## iter 180 value 19.823714
## iter 190 value 19.823241
## iter 200 value 19.821929
## iter 210 value 19.820938
## iter 220 value 19.820600
## iter 230 value 15.564400
## iter 240 value 15.316459
## iter 250 value 15.293021
## iter 260 value 15.267555
## iter 270 value 15.228925
## iter 280 value 15.219195
## iter 290 value 15.218384
## iter 300 value 15.218296
```

```
## iter 310 value 15.217997
## iter 320 value 15.216966
## iter 330 value 15.210446
## iter 340 value 15.169683
## iter 350 value 15.125378
## iter 360 value 14.836218
## iter 370 value 14.610837
## iter 380 value 10.957461
## iter 390 value 10.654106
## iter 400 value 10.640421
## iter 410 value 10.520044
## iter 420 value 10.465496
## iter 430 value 10.366535
## iter 440 value 10.217303
## iter 450 value 10.047114
## iter 460 value 10.031337
## iter 470 value 10.031141
## iter 480 value 10.031033
## iter 490 value 10.030555
## iter 500 value 9.879454
## final  value 9.879454
## stopped after 500 iterations
```

```
big_ann_preds <- predict(big_ann_model, newdata = test_data, type = "class")
conf_matrix_big_ann <- table(Actual = test_data$Class, Predicted =
big_ann_preds)
print(conf_matrix_big_ann)
```

```
##       Predicted
## Actual  1  2  3
##      1 19  1  0
##      2  2 18  2
##      3  1  0 11
```

```
test_error <- 1 - sum(diag(conf_matrix_big_ann)) / sum(conf_matrix_big_ann)
cat("Test Error Rate:", test_error, "\n")
```

```
## Test Error Rate: 0.1111111
```

We achieve an error rate of 11%, which is certainly way better than what we previously got.