svm_train_svm_tune.R

vikaskamath

2021-03-21

```
### Simple SVM Example

### Let's go ahead and use the (default data):

train = read.csv(file = "train.csv", header = TRUE )
test  = read.csv(file = "test.csv" , header = TRUE )
sol   = read.csv(file = "sol.csv", header = TRUE )

train$Id = NULL
test$Id  = NULL

y = sol$y

### WARNING: Do not use the full train data!  It will take too much time.
### Even though the underlying svm function uses a library written in C++, it is
###  still pretty slow.
### We'll use only a small fraction of the train data 3000 data points

n     = 3000
train = train[1:n,]

library(e1071)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
### Setting kernel = "linear" will do the hyperplace classification.
### We'll just use the defaults for the svm function for now except
###  probability = T will enable use to give predicted probabilities later on.
### Note that svm scales the data.

system.time({
svm_model = svm(as.factor(y) ~ ., data = train , kernel = "linear", probability = T)
})
```

```
##    user  system elapsed
##   1.072   0.009   1.080
```

```
### Quickly look at some results:

svm_model
```

```
##
## Call:
## svm(formula = as.factor(y) ~ ., data = train, kernel = "linear",
##     probability = T)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  405
```

```
svm_model$index
```

```
##   [1]    1   14   22   26   52   75   89  163  180  185  186  187  196  215  227
##  [16]  241  252  276  335  345  351  355  378  399  421  424  469  470  506  514
##  [31]  521  529  537  573  597  600  643  654  655  668  685  698  740  761  769
##  [46]  790  794  801  848  851  857  920  922  944  968  989  992 1017 1030 1045
##  [61] 1053 1085 1092 1147 1154 1168 1174 1187 1201 1232 1261 1267 1268 1278 1286
##  [76] 1297 1300 1312 1314 1339 1340 1401 1405 1423 1465 1486 1512 1581 1600 1615
##  [91] 1649 1663 1714 1731 1734 1767 1771 1790 1797 1798 1801 1804 1827 1839 1846
## [106] 1862 1867 1872 1878 1910 1912 1956 1958 1978 1980 1992 2025 2033 2043 2070
## [121] 2082 2083 2097 2106 2132 2152 2171 2220 2266 2278 2279 2281 2314 2315 2318
## [136] 2324 2337 2340 2409 2421 2430 2436 2464 2472 2477 2490 2496 2552 2560 2561
## [151] 2574 2597 2648 2677 2704 2708 2767 2824 2829 2894 2909 2932 2934 2945 2949
## [166] 2958 2971    3    7   37   44   69   96   97  124  152  156  158  166  184
## [181]  192  206  212  222  258  287  294  297  298  302  305  320  333  336  339
## [196]  353  370  400  404  423  449  454  459  463  475  479  481  493  501  528
## [211]  552  558  563  582  606  607  613  638  644  645  652  661  663  673  694
## [226]  706  717  736  742  757  762  771  773  781  803  812  832  878  883  892
## [241]  900  934  948  973  990  991  998 1010 1018 1044 1056 1067 1074 1083 1102
## [256] 1122 1123 1126 1132 1134 1156 1167 1194 1220 1229 1241 1247 1274 1288 1299
## [271] 1305 1315 1333 1366 1376 1413 1421 1428 1441 1442 1471 1489 1513 1514 1515
## [286] 1549 1550 1580 1586 1601 1608 1609 1630 1632 1637 1640 1680 1683 1688 1712
## [301] 1730 1735 1746 1751 1774 1792 1795 1803 1814 1815 1821 1830 1865 1869 1895
## [316] 1903 1916 1929 1959 1976 1998 2026 2048 2052 2055 2057 2061 2084 2114 2129
## [331] 2141 2153 2162 2170 2180 2186 2210 2225 2230 2237 2243 2253 2270 2287 2293
## [346] 2294 2332 2358 2370 2381 2387 2394 2402 2407 2435 2442 2443 2450 2456 2460
## [361] 2462 2469 2479 2497 2517 2578 2584 2610 2612 2620 2635 2656 2661 2667 2674
## [376] 2678 2679 2687 2688 2696 2702 2716 2720 2737 2740 2750 2752 2770 2790 2800
## [391] 2801 2803 2821 2837 2856 2892 2899 2911 2929 2977 2983 2987 2992 2993 2994
```

```
length(svm_model$index)
```

```
## [1] 405
```

```
### svm_model$index tells us the row number of the support vectors.

### Let's get both the training and test AUC's

train_pred = predict(svm_model, newdata = train, probability = T)
y_train    = attr(train_pred, "probabilities")[,1]
head(y_train)
```

```
##          1          2          3          4          5          6
## 0.05557276 0.05557247 0.05557261 0.05557240 0.05557165 0.05557267
```

```
test_pred  = predict(svm_model, newdata = test, probability = T)
y_test     = attr(test_pred, "probabilities")[,1]
head(y_test)
```

```
##          1          2          3          4          5          6
## 0.05557257 0.05557266 0.05557246 0.05557240 0.05557264 0.05557249
```

```
par(mfrow=c(1,2))
roc(response = train$y, predictor = y_train, plot=T, col = "red")
```

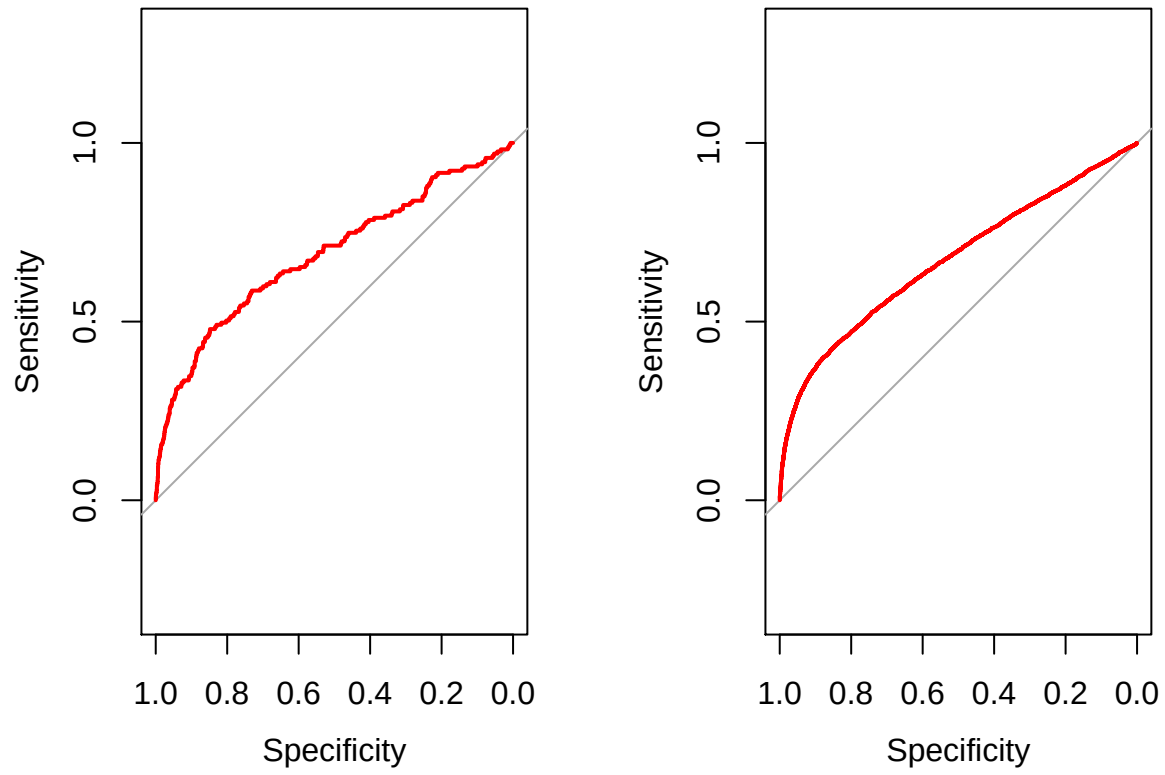
```
## Setting levels: control = n, case = y
```

```
## Setting direction: controls < cases
```

```
##
## Call:
## roc.default(response = train$y, predictor = y_train, plot = T,     col = "red")
##
## Data: y_train in 2833 controls (train$y n) < 167 cases (train$y y).
## Area under the curve: 0.6874
```

```
roc(response = y,       predictor = y_test , plot=T, col = "red")
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = y, predictor = y_test, plot = T, col = "red")
##
## Data: y_test in 69971 controls (y 0) < 5029 cases (y 1).
## Area under the curve: 0.6719
```

```
par(mfrow=c(1,1))

### By the way, something strange happens when you set n = 3000.
### The results get worse!

train = read.csv( "train.csv", header = TRUE )
test  = read.csv( "test.csv",  header = TRUE )
sol   = read.csv( "sol.csv",   header = TRUE )

train$Id = NULL
test$Id  = NULL

y = sol$y

###
### At random, pick 5000 rows of the train data.
### You can change  5000 to say 1000 if your tuning takes too much time.
### We could have just taken the first 5000 rows as well.
```

```
set.seed(1)

n     = 5000
ind   = sample(1:nrow(train), size = n, replace = F)
train = train[ind,]

### Do you remember what a very large or very small cost does?
### Do you remember what a very large or very small gamma does?

svm_grid = expand.grid( cost = c(0.1 ,1 ,10 ,100, 1000 ),
                        gamma = c(0.001, 0.1, 0.5 ,2,10,50) )
dim(svm_grid)
```

```
## [1] 30  2
```

```
m = dim(svm_grid)[1]
m
```

```
## [1] 30
```

```
svm_auc  = rep(0, m)
svm_auc
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
no_of_folds = 2
set.seed(2000)
index_values = sample(1:no_of_folds, size = nrow(train), replace = TRUE)


system.time({

  for (i in 1:m)
  {

    tmp_auc     = rep(0, no_of_folds)

    for (j in 1:no_of_folds)
    {
      index_out     = which(index_values == j)
      left_out_data = train[  index_out, ]
      left_in_data  = train[ -index_out, ]

      ###
      ### The default kernel is radial basis so we do not need to explicitly state it.
      ###

      tmp_model     = svm( as.factor(y) ~ ., data = left_in_data, cost = svm_grid$cost[i],
                           gamma = svm_grid$gamma[i], probability = T)

      ###
```

```
      ### This next line is a bit awkward but it basically extracts the predicted
      ### probabilities.
      ###

      tmp_pred     = attr(predict(tmp_model, newdata = left_out_data, probability = T),
                          "probabilities")[,1]

      tmp_auc[j]   = roc(response = left_out_data$y, predictor = tmp_pred, plot=F)$auc[1]

    }

    svm_auc[i]     = mean(tmp_auc)

  }

})
```

```
## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases
```

```
## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases
```

```
## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases
```

```
## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases
```

```
## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls < cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases
```

```
## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

## Setting levels: control = n, case = y

## Setting direction: controls > cases

##    user  system elapsed
##  44.728   0.150  44.884
```

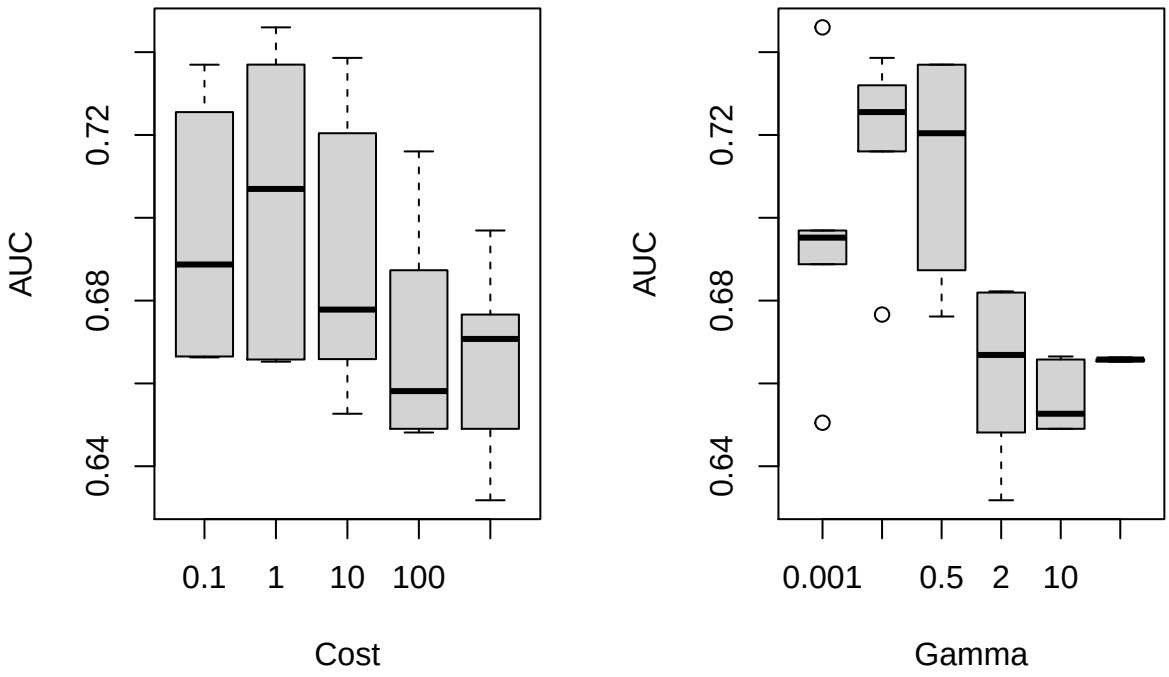
```
results     = cbind(svm_grid, svm_auc)
results
```

```
##     cost gamma   svm_auc
## 1  1e-01 1e-03 0.6952029
## 2  1e+00 1e-03 0.7459880
## 3  1e+01 1e-03 0.6887832
## 4  1e+02 1e-03 0.6505127
## 5  1e+03 1e-03 0.6969299
## 6  1e-01 1e-01 0.7255246
## 7  1e+00 1e-01 0.7319970
## 8  1e+01 1e-01 0.7386183
## 9  1e+02 1e-01 0.7160311
## 10 1e+03 1e-01 0.6766322
## 11 1e-01 5e-01 0.7369764
## 12 1e+00 5e-01 0.7369941
## 13 1e+01 5e-01 0.7204256
## 14 1e+02 5e-01 0.6873384
## 15 1e+03 5e-01 0.6761632
## 16 1e-01 2e+00 0.6822323
## 17 1e+00 2e+00 0.6819429
## 18 1e+01 2e+00 0.6668816
## 19 1e+02 2e+00 0.6481517
## 20 1e+03 2e+00 0.6317870
## 21 1e-01 1e+01 0.6665103
## 22 1e+00 1e+01 0.6657743
## 23 1e+01 1e+01 0.6526750
## 24 1e+02 1e+01 0.6490356
## 25 1e+03 1e+01 0.6490354
## 26 1e-01 5e+01 0.6662918
## 27 1e+00 5e+01 0.6652512
## 28 1e+01 5e+01 0.6658701
## 29 1e+02 5e+01 0.6657811
## 30 1e+03 5e+01 0.6653282
```

```
best_result = results[which.max(svm_auc),]
best_result
```

```
##   cost gamma  svm_auc
## 2    1 0.001 0.745988
```

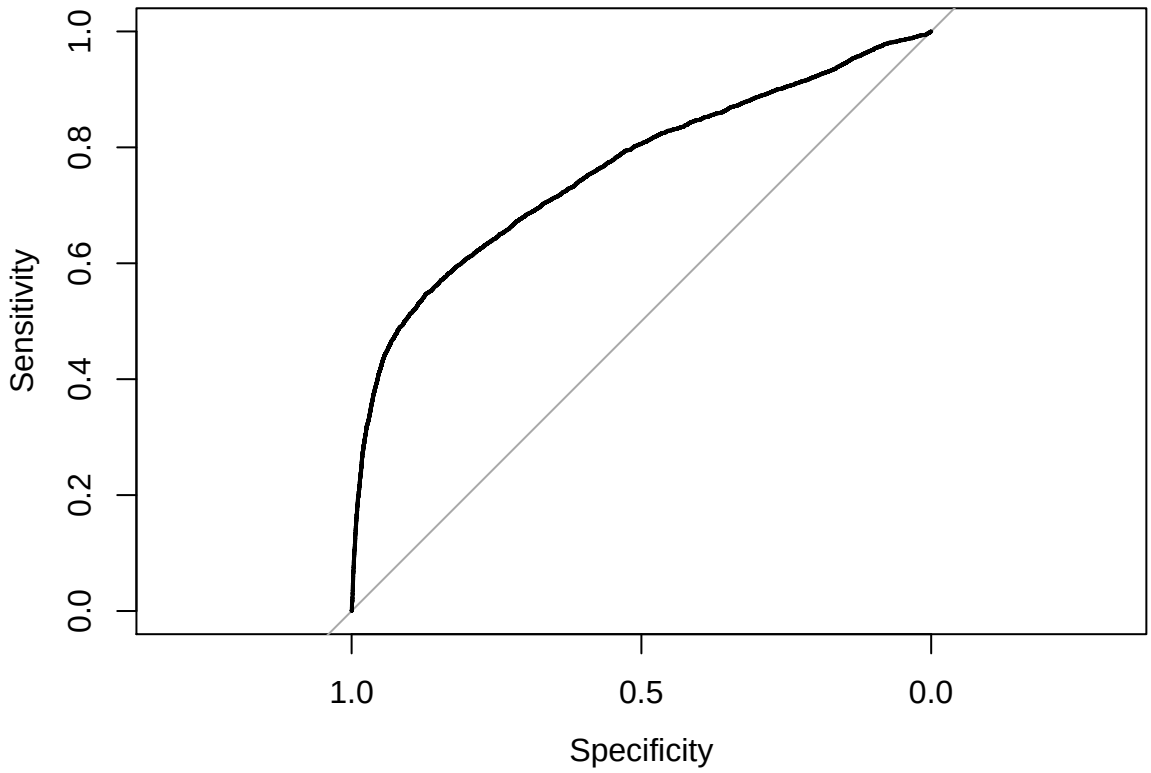
```
par( mfrow = c(1,2))
boxplot( svm_auc ~ cost,  data = results, xlab = "Cost", ylab = "AUC")
boxplot( svm_auc ~ gamma, data = results, xlab = "Gamma", ylab = "AUC")
```




```
par( mfrow = c(1,1))

# the max value of AUc according to the trend is Cost = 0.1 and Gamma = 0.5

final_model = svm(as.factor(y) ~ ., data = train, cost = best_result$cost,
                  gamma = best_result$gamma, probability = T)

svm_y = attr(predict(final_model, newdata = test, probability = T),
             "probabilities")[,1]

roc_svm = roc(response = y, predictor = svm_y, plot=T, col = "black")
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
roc_svm$auc
```

```
## Area under the curve: 0.7605
```

```
## The ROC curve result is 0.7596, not bad!
```