SVM Week 1

David Milmont

May 21, 2017

Loading Packages and Downloading Data - Creating Train, Test, Validate data sets

```r
library('kernlab')
library('RCurl')
```

```
## Loading required package: bitops
```

```r
library('ggplot2')
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```r
library('GGally')
library('mlr')
```

```
## Loading required package: ParamHelpers
```

```r
library('kknn')

file <- getURL('https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/v1/e39a3df780dacd5503df6a8322d72

data <- read.csv(textConnection(file), header=T, sep = "\t")

#Summary of data
summarizeColumns(data)
```

```
##    name    type na         mean         disp  median        mad   min
## 1    A1 integer  0    0.6896024    0.4630105   1.000   0.000000  0.00
## 2    A2 numeric  0   31.5783486   11.9817891  28.460  10.318896 13.75
## 3    A3 numeric  0    4.8305581    5.0232952   2.855   3.335850  0.00
## 4    A8 numeric  0    2.2416896    3.3691972   1.000   1.356579  0.00
## 5    A9 integer  0    0.5351682    0.4991434   1.000   0.000000  0.00
## 6   A10 integer  0    0.5611621    0.4966249   1.000   0.000000  0.00
## 7   A11 integer  0    2.4984709    4.9656552   0.000   0.000000  0.00
## 8   A12 integer  0    0.5382263    0.4989182   1.000   0.000000  0.00
## 9   A14 integer  0  180.0840979  168.3157190 160.000 148.260000  0.00
## 10  A15 integer  0 1012.7308869 5249.3206597   5.000   7.413000  0.00
## 11   R1 integer  0    0.4525994    0.4981291   0.000   0.000000  0.00
##         max nlevs
## 1      1.00     0
## 2     80.25     0
## 3     28.00     0
## 4     28.50     0
## 5      1.00     0
```

```
## 6       1.00     0
## 7      67.00     0
## 8       1.00     0
## 9    2000.00     0
## 10 100000.00     0
## 11      1.00     0
```

```
#Create data frame split
set.seed(546)

#shuffling to ensure randomness
data <- data[sample(nrow(data)),]

#Getting idea of sizes
nrow(data) * .60
```

```
## [1] 392.4
```

```
nrow(data) * .20
```

```
## [1] 130.8
```

```
nrow(data) * .20
```

```
## [1] 130.8
```

```
#Spliting Manually - KISS method
train <- data[1:394,]
test <- data[395:525,]
validate <- data[526:654,]
```

Visualization

Quick visual to see how the data is correlated

```
data[,11] <- as.factor(data[,11])
GGally::ggpairs(data[, c(2:4,9:11)], aes(colour=R1))
```

Creating the Model - SVM

Selected the polydot kernal as it has the highest accuracy. After runing MLR package for parameter hypertuning settled with C=2.73e+06 for 100% accuracy - This results in a very small margin hyperplane. I suspect in a real life scenerio this is not optimal as the use case would benefit from some margin of error for review scenerios.

```
#Creating matrix to train model
x <- as.matrix(data[,1:10])
#creating target
y <- data[,11]

model <- ksvm(y ~ x, type = "C-svc", kernal = "polydot", C=2.73e+06, scaled = TRUE, cross = 5)
```

```
a <- colSums(data[model@SVindex,1:10]*model@coef[[1]])
cat('a:',a,'\n')
```

```
## a: -298.9353 -16576.53 -4488.99 2362.042 536.0985 -382.8606 2740.088 -491.4585 8406.12 4124302
```

```
a0 <- sum(a*data[1,1:10]) - model@b
cat('a0:',a0,'\n')
```

```
## a0: 4127211276
```

```
pred <- predict(model,data[,1:10])
# pred
data$prediction <- pred

cat('SVM accuracy:',sum(pred == data[,11]) / nrow(data),'\n')
```

```
## SVM accuracy: 1
```

```
cat('offset:',b(model),'\n')
```

```
## offset: 8.856835
```

```
cat('error',error(model),'\n')
```

```
## error 0
```

```
kernelf(model)
```

```
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0916808764420663
```

Kernal Selection

Checking different kernals and their affect on prediction accuracy - selected polydot for final model

```
kernals <- c('rbfdot','polydot','vanilladot','tanhdot','laplacedot','besseldot','anovadot','splinedot',

for(kernal in kernals){
model <- ksvm(x, y, type = "C-svc", kernal = kernal, C=100, scaled = TRUE, cross = 5)
pred <- predict(model,data[,1:10])
cat('\n',kernal,'pred: ', sum(pred == data[,11]) / nrow(data))
}
```

```
##
##  rbfdot pred:  0.9587156
##  polydot pred:  0.9525994
##  vanilladot pred:  0.9541284
##  tanhdot pred:  0.9541284
##  laplacedot pred:  0.9587156
##  besseldot pred:  0.9541284
##  anovadot pred:  0.9480122
##  splinedot pred:  0.9541284
##  stringdot pred:  0.9571865
```

Paramater Hypertuning

Utilized the MLR package for parameter tuning, selected C=2.73e+06 for highest prediction accuracy

```r
#Trying mlr package for parameter hypertuning

trainTask <- makeClassifTask(data = data, target = 'R1')
trainTask
```

```
## Supervised task: data
## Type: classif
## Target: R1
## Observations: 654
## Features:
## numerics  factors  ordered
##       10        1        0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Classes: 2
##   0   1
## 358 296
## Positive class: 0
```

```r
learner <- makeLearner("classif.ksvm")

ksvm <- makeLearner("classif.ksvm", predict.type = "response")
getParamSet("classif.ksvm")
```

```
##                        Type  len    Def
## scaled              logical    -   TRUE
## type               discrete    -  C-svc
## kernel             discrete    - rbfdot
## C                   numeric    -      1
## nu                  numeric    -    0.2
## epsilon             numeric    -    0.1
## sigma               numeric    -      -
## degree              integer    -      3
## scale               numeric    -      1
## offset              numeric    -      1
## order               integer    -      1
## tol                 numeric    -  0.001
## shrinking           logical    -   TRUE
## class.weights numericvector <NA>      -
## fit                 logical    -   TRUE
## cache               integer    -     40
##                                              Constr Req Tunable Trafo
## scaled                                            -   -    TRUE     -
## type              C-svc,nu-svc,C-bsvc,spoc-svc,kbb-svc   -    TRUE     -
## kernel        vanilladot,polydot,rbfdot,tanhdot,lap...   -    TRUE     -
## C                                          0 to Inf   Y    TRUE     -
## nu                                         0 to Inf   Y    TRUE     -
## epsilon                                 -Inf to Inf   Y    TRUE     -
## sigma                                      0 to Inf   Y    TRUE     -
## degree                                     1 to Inf   Y    TRUE     -
## scale                                      0 to Inf   Y    TRUE     -
## offset                                  -Inf to Inf   Y    TRUE     -
## order                                   -Inf to Inf   Y    TRUE     -
## tol                                        0 to Inf   -    TRUE     -
```

```
## shrinking                                              -    -    TRUE     -
## class.weights                                  0 to Inf    -    TRUE     -
## fit                                                    -    -   FALSE     -
## cache                                          1 to Inf    -    TRUE     -
```

```
set_cv <- makeResampleDesc("CV",iters = 3L)
pssvm <- makeParamSet(
  makeNumericParam("C", lower = -10, upper = 10, trafo = function(x) 10^x),
  makeNumericParam("sigma", lower = -10, upper = 10, trafo = function(x) 10^x)
)


ctrl = makeTuneControlRandom(maxit = 200L)

res <- tuneParams(ksvm, task = trainTask, resampling = set_cv, par.set = pssvm, control = ctrl)

t.svm <- setHyperPars(ksvm, par.vals = res$x)

par.svm <- train(ksvm, trainTask)
predict.svm <- predict(par.svm, trainTask)

res
```

```
## Tune result:
## Op. pars: C=8.36e+07; sigma=0.00261
## mmce.test.mean=   0
```

Training KKNN - Using Test, Train, And Validate data sets - results are KKNN accuracy: 0.9160305

```
train.knn <- kknn(formula = formula(train$R1~.), train = train, test = test, k = 7, distance = 1, kernel

fit <- fitted(train.knn)

table(test$R1, fit)
```

```
##    fit
##      0 0.0127439975524816 0.0405412744063402 0.0727517584925532
##   0 38                  2                  3                  1
##   1  4                  0                  0                  0
##    fit
##     0.0854957560450348 0.111364854928861 0.124108852481343
##   0                  0                 1                 1
##   1                  1                 0                 0
##    fit
##     0.160270316835722 0.213555588794544 0.241936733479724
##   0                 1                 1                 2
##   1                 0                 0                 0
##    fit
##     0.245766072880757 0.301944494419796 0.324920443723405
##   0                 0                 1                 0
##   1                 1                 0                 1
##    fit
##     0.342485768826136 0.373135061856799 0.385879059409281
```

```
##    0                 0                 3                 1
##    1                 2                 0                 0
##     fit
##      0.397672202215958 0.413676336263139 0.426420333815621 0.48449991678566
##    0                 0                 1                 0                1
##    1                 1                 0                 1                0
##     fit
##      0.499172092308174 0.537785188744482 0.541369182098166
##    0                 1                 0                 1
##    1                 1                 1                 0
##     fit
##      0.573579666184379 0.586690650651343 0.597792949684554
##    0                 0                 0                 0
##    1                 1                 1                 1
##     fit
##      0.602327797784042 0.614120940590719 0.618901134737556
##    0                 0                 1                 0
##    1                 1                 1                 1
##     fit
##      0.626864938143201 0.642869072190382 0.644770233621383
##    0                 0                 0                 0
##    1                 3                 2                 1
##     fit
##      0.687823553829076 0.726436650265384 0.730265989666417
##    0                 0                 0                 0
##    1                 1                 1                 2
##     fit
##      0.754233927119243 0.758063266520276 0.770807264072758
##    0                 0                 0                 0
##    1                 1                 1                 1
##     fit
##      0.775342112172246 0.799188408757938 0.839729683164278
##    0                 1                 1                 1
##    1                 0                 0                 3
##     fit
##      0.848093870664799 0.886706967101107 0.888635145071139
##    0                 0                 0                 1
##    1                 2                 1                 1
##     fit
##      0.914504243954965 0.946714728041178 0.95945872559366 0.987256002447518
##    0                 1                 1                 1                0
##    1                 0                 0                 1                1
##     fit
##       1
##    0  1
##    1 22
```

```
(fit.train1 <- train.kknn(R1 ~ ., train, kmax = 3,
    kernel = "optimal", distance = 1))
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = train, kmax = 3, distance = 1,     kernel = "optimal")
##
## Type of response variable: continuous
```

```
## minimal mean absolute error: 0.1991646
## Minimal mean squared error: 0.1514675
## Best kernel: optimal
## Best k: 3
```

```r
(fit.train2 <- train.kknn(R1 ~ ., train, kmax = 4,
    kernel = "optimal", distance = 1))
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = train, kmax = 4, distance = 1,     kernel = "optimal")
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1991646
## Minimal mean squared error: 0.1408923
## Best kernel: optimal
## Best k: 4
```

```r
(fit.train3 <- train.kknn(R1 ~ ., train, kmax = 5,
    kernel = "optimal", distance = 1))
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = train, kmax = 5, distance = 1,     kernel = "optimal")
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1991646
## Minimal mean squared error: 0.134424
## Best kernel: optimal
## Best k: 5
```

```r
(fit.train4 <- train.kknn(R1 ~ ., train, kmax = 6,
    kernel = "optimal", distance = 1))
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = train, kmax = 6, distance = 1,     kernel = "optimal")
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1991646
## Minimal mean squared error: 0.130609
## Best kernel: optimal
## Best k: 6
```

```r
# table(predict(fit.train1, validate), validate$R1)
# table(predict(fit.train2, validate), validate$R1)
# table(predict(fit.train3, validate), validate$R1)
# table(predict(fit.train4, validate), validate$R1)

# plot(fit.train1)
# plot(fit.train2)
# plot(fit.train3)
# plot(fit.train4)

cat('KKNN accuracy:',sum(fit == test[,11]) / nrow(test),'\n')
```

```
## KKNN accuracy: 0.4580153
```