HW3 - SVM

Olivia Samples

6/30/2020

Dataset description

The description of the data can be found here http://archive.ics.uci.edu/ml/machine-learning-databases/tic-tac-toe/tic-tac-toe.names. The data is also available on canvas.

VARIABLE DESCRIPTIONS:

1. top-left-square: x,o,b
2. top-middle-square: x,o,b
3. top-right-square: x,o,b
4. middle-left-square: x,o,b
5. middle-middle-square: x,o,b
6. middle-right-square: x,o,b
7. bottom-left-square: x,o,b
8. bottom-middle-square: x,o,b
9. bottom-right-square: x,o,b
10. Class: positive,negative

Your tasks

Download/save and load the data (10 points) Start by cleaning the data (everything should be number, except the class) (10 points) Split the data into two sets (training and testing) (10 points) Use the train method from the caret library to figure out which svm Kernel (linear, polynominal, RBF) will work best with the training data. (30 points) Test the best svm Kernel with the testing data (10 points) 100 bootstrap for RBF kernel, report accuracy (10 points) and the 95% confidence interval of the accuracy (10 points). Submit your work using RMarkdown, including codes, output, and your explaination (10 points)

```
library(e1071)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(ROCR)
```

Read the csv file

```
tictac <- read.csv(file='tictac2.csv', head=TRUE, sep=",")
```

Now let's explore and clean the data.

```
any(is.na(tictac))
```

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

```
sapply(tictac, class)
```

```
##        c1        c2        c3        c4        c5        c6        c7        c8
## "integer" "integer" "integer" "integer" "integer" "integer" "integer" "integer"
##        c9     class
## "integer"  "factor"
```

```
summary(tictac)
```

```
##        c1              c2              c3              c4
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median :1.0000   Median :1.0000
##  Mean   :0.8643   Mean   :0.9165   Mean   :0.8643   Mean   :0.9165
##  3rd Qu.:1.0000   3rd Qu.:2.0000   3rd Qu.:1.0000   3rd Qu.:2.0000
##  Max.   :2.0000   Max.   :2.0000   Max.   :2.0000   Max.   :2.0000
##        c5              c6              c7              c8
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median :1.0000   Median :1.0000
##  Mean   :0.8121   Mean   :0.9165   Mean   :0.8643   Mean   :0.9165
##  3rd Qu.:1.0000   3rd Qu.:2.0000   3rd Qu.:1.0000   3rd Qu.:2.0000
##  Max.   :2.0000   Max.   :2.0000   Max.   :2.0000   Max.   :2.0000
##        c9             class
##  Min.   :0.0000   negative:332
##  1st Qu.:0.0000   positive:626
##  Median :1.0000
##  Mean   :0.8643
##  3rd Qu.:1.0000
##  Max.   :2.0000
```

As you can see, all of the data features are integers except for the class column which is a factor. There are no null values and the data does not need to be scaled.

Let's create 80-20 split of the data. 80% training data and 20% testing data.

```
library(caTools)
set.seed(101)
split <- sample.split(tictac$class, SplitRatio = 0.8)
train <- subset(tictac, split==TRUE)
test <- subset(tictac, split==FALSE, select =-class)
```

Now we are ready to run our svm classification

Sometimes we do not know ahead of time the appropriate setup for the kernel to be used. To handle this issue we need to use the train() method from the caret packages. Let us first extract the predictors and the classes from the training data

```
y <- train$class
x <- subset(train,select =-class)
```

To train svm for a Polynomial kernel using a 10-folds cross validation repeated 10-times

```
P_model <- train(x,y,method="svmPoly",tuneLength=5,trControl=trainControl(method='repeatedcv',number =
```

To train svm for a Linear kernel using a 10-folds cross validation repeated 10-times

```
L_model <- train(x,y,method="svmLinear",tuneLength=5,
trControl=trainControl(method='repeatedcv',number = 10,repeats = 10))
```

To train svm for a Radial kernel using a 10-folds cross validation repeated 10-times

```
R_model <- train(x,y,method="svmRadial",tuneLength=5,
trControl=trainControl(method='repeatedcv', number = 10,repeats = 10))
```

To see the result of the training you can do

```
P_model
```

```
## Support Vector Machines with Polynomial Kernel
##
## 767 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 690, 691, 691, 691, 690, 690, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     Accuracy   Kappa
##   1       1e-03  0.25  0.6532157  0.000000000
##   1       1e-03  0.50  0.6532157  0.000000000
##   1       1e-03  1.00  0.6532157  0.000000000
##   1       1e-03  2.00  0.6532157  0.000000000
##   1       1e-03  4.00  0.6532157  0.000000000
##   1       1e-02  0.25  0.6532157  0.000000000
##   1       1e-02  0.50  0.6532157  0.000000000
##   1       1e-02  1.00  0.6532157  0.000000000
##   1       1e-02  2.00  0.6671434  0.068493924
##   1       1e-02  4.00  0.6797183  0.234532236
##   1       1e-01  0.25  0.7256929  0.272282916
##   1       1e-01  0.50  0.6309698  0.148591087
##   1       1e-01  1.00  0.6167248  0.123153115
##   1       1e-01  2.00  0.6176459  0.124895777
##   1       1e-01  4.00  0.6189446  0.127316268
##   1       1e+00  0.25  0.6176459  0.124895777
##   1       1e+00  0.50  0.6189446  0.127316268
##   1       1e+00  1.00  0.6189446  0.127316268
##   1       1e+00  2.00  0.6208810  0.131129352
##   1       1e+00  4.00  0.6208810  0.131129352
##   1       1e+01  0.25  0.6208810  0.131129352
##   1       1e+01  0.50  0.6208810  0.131129352
##   1       1e+01  1.00  0.6208810  0.131129352
##   1       1e+01  2.00  0.6208810  0.131129352
##   1       1e+01  4.00  0.6208810  0.131129352
##   2       1e-03  0.25  0.6532157  0.000000000
##   2       1e-03  0.50  0.6532157  0.000000000
##   2       1e-03  1.00  0.6532157  0.000000000
##   2       1e-03  2.00  0.6532157  0.000000000
##   2       1e-03  4.00  0.6532157  0.000000000
##   2       1e-02  0.25  0.6532157  0.000000000
```

```
##   2       1e-02  0.50  0.6532157  0.000000000
##   2       1e-02  1.00  0.6809505  0.120276099
##   2       1e-02  2.00  0.6797011  0.229751691
##   2       1e-02  4.00  0.6779976  0.241066212
##   2       1e-01  0.25  0.7794212  0.488085030
##   2       1e-01  0.50  0.8288191  0.619312704
##   2       1e-01  1.00  0.8693310  0.714151271
##   2       1e-01  2.00  0.8890306  0.758940977
##   2       1e-01  4.00  0.9111874  0.808441345
##   2       1e+00  0.25  0.9856574  0.967717003
##   2       1e+00  0.50  0.9856574  0.967717003
##   2       1e+00  1.00  0.9856574  0.967717003
##   2       1e+00  2.00  0.9856574  0.967717003
##   2       1e+00  4.00  0.9856574  0.967717003
##   2       1e+01  0.25  0.9856574  0.967717003
##   2       1e+01  0.50  0.9846167  0.965433938
##   2       1e+01  1.00  0.9812297  0.958334006
##   2       1e+01  2.00  0.9778343  0.951042838
##   2       1e+01  4.00  0.9749566  0.944636965
##   3       1e-03  0.25  0.6532157  0.000000000
##   3       1e-03  0.50  0.6532157  0.000000000
##   3       1e-03  1.00  0.6532157  0.000000000
##   3       1e-03  2.00  0.6532157  0.000000000
##   3       1e-03  4.00  0.6532157  0.000000000
##   3       1e-02  0.25  0.6532157  0.000000000
##   3       1e-02  0.50  0.6524247  0.006178065
##   3       1e-02  1.00  0.7117541  0.288060827
##   3       1e-02  2.00  0.7009388  0.288373480
##   3       1e-02  4.00  0.7303959  0.360749012
##   3       1e-01  0.25  0.8878432  0.755832977
##   3       1e-01  0.50  0.9142860  0.814879601
##   3       1e-01  1.00  0.9465113  0.883819494
##   3       1e-01  2.00  0.9727216  0.939676293
##   3       1e-01  4.00  0.9788767  0.952859462
##   3       1e+00  0.25  0.9214098  0.827565567
##   3       1e+00  0.50  0.9214098  0.827565567
##   3       1e+00  1.00  0.9214098  0.827565567
##   3       1e+00  2.00  0.9214098  0.827565567
##   3       1e+00  4.00  0.9214098  0.827565567
##   3       1e+01  0.25  0.9521559  0.895247796
##   3       1e+01  0.50  0.9521559  0.895247796
##   3       1e+01  1.00  0.9521559  0.895247796
##   3       1e+01  2.00  0.9521559  0.895247796
##   3       1e+01  4.00  0.9521559  0.895247796
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 2, scale = 1 and C = 0.25.
```

```
L_model
```

```
## Support Vector Machines with Linear Kernel
##
## 767 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 690, 690, 691, 690, 690, 690, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6175253  0.1253164
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
R_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 767 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 691, 690, 691, 690, 690, 691, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7003843  0.1931102
##   0.50  0.7926482  0.5048245
##   1.00  0.8819603  0.7362794
##   2.00  0.9360971  0.8617122
##   4.00  0.9658332  0.9249670
##
## Tuning parameter 'sigma' was held constant at a value of 0.06693486
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.06693486 and C = 4.
```

As you can see, this Polynomial Kernel performs best. We will use it to predict on our test data.

```
pred <- predict(P_model,test)
```

For evaluation purpose, let's extract the classes for the testing data.

```
newtest <- subset(tictac, split==FALSE)$class
```

To see the confusion matrix, let us compare what is in pred with what is in y.

```
confusionMatrix(pred,newtest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       61        0
##   positive        5      125
##
##                Accuracy : 0.9738
##                  95% CI : (0.94, 0.9914)
##     No Information Rate : 0.6545
```

```
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9411
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 0.9242
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9615
##              Prevalence : 0.3455
##          Detection Rate : 0.3194
##    Detection Prevalence : 0.3194
##       Balanced Accuracy : 0.9621
##
##        'Positive' Class : negative
##
```

Now let's look at the 95% confidence interval for accuracy and AUC. We use 100 resamples for bootstrap here and the polynomial kernel.

```
# n <- 100
# accuracy =rep(0,n)
#
# for (i in 1:n) {
#   set.seed(i+100)
#   new_index <- sample(c(1:length(tictac$class)), length(tictac$class), replace=TRUE)
#   new_sample <- tictac[new_index,]
#   split <- sample.split(new_sample$class, SplitRatio = 0.8)
#   train <- subset(new_sample, split==TRUE)
#   test.x <- subset(subset(new_sample, split==FALSE), select=-class)
#   test.y <- subset(new_sample, split==FALSE)$class
#
#   svm.p <- train(class~.
#                  ,data=train
#                  ,method="svmPoly"
#                  ,tuneLength = 5
#                  ,trControl=trainControl(method='repeatedcv',repeats = 5
#                                          ,classProbs=TRUE)
#                   )
#
#   ## accuracy
#   pred <- predict(svm.p,test.x)
#   c <- confusionMatrix(pred,test.y)
#   accuracy[i] <-c[3]$overall[1]
#
# }
#
# # 95% confidence interval for accuracy
# accuracy.mean<- mean(accuracy)
# accuracy.me <- qnorm(0.975) * sd(accuracy)/sqrt(length(accuracy))
# accuracy.lci <- accuracy.mean - accuracy.me
# accuracy.uci <- accuracy.mean + accuracy.me
#
# # The accuracy is below:
```

```r
# accuracy.mean
#
# #The Confidence Interval is below:
# c(accuracy.lci, accuracy.uci)
```

We will compute the accuracy of our best performing kernel: Polynomial kernel.

```r
accuracy <- confusionMatrix(pred,newtest)$overall[1]
# The accuracy is below:
accuracy
```

```
## Accuracy
## 0.973822
```