Machine learning - Assignment 6

Alice Tivarovsky

2/27/2020

## Setup

Loading the necessary packages for data cleaning, modeling, partitioning, nhanes interface, and classification.

Comparison between Classification Trees, SVM and Logistic Regression

The posted article by Yu et al utilized NHANES data from 1999-2004 to predict diabetes and pre-diabetes using Support Vector Machines. You will conduct a similar analysis using data within the NHANES package in R. For this exercise, you will try to predict Diabetes using similar (although not all) variables. The available data is also slightly different, so you likely won’t get the same answers.

## Number 1: Loading NHANES, Restricting and Partitioning

Restrict the NHANES data to the list of 12 variables below. Partition the data into training and testing using a 70/30 split.

“Age”, “Gender”, “Race1”, “Education”, “HHIncome” (DEMO), “Weight”, “Height” (BMXBMI), “Pulse” (BPX), “Diabetes” (DIQ), “BMI” (BMXBMI), “PhysActive”, “Smoke100” (SMQ)

# Loading  
demo\_99 = nhanes\_load\_data("DEMO", "1999-2000")

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Downloading DEMO.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/DEMO.XPT

demo\_01 = nhanes\_load\_data("DEMO", "2001-2002")

## Downloading DEMO\_B.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/DEMO\_B.XPT

demo\_03 = nhanes\_load\_data("DEMO", "2003-2004")

## Downloading DEMO\_C.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/DEMO\_C.XPT

bmx\_99 = nhanes\_load\_data("BMX", "1999-2000")

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Downloading BMX.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/BMX.XPT

bmx\_01 = nhanes\_load\_data("BMX", "2001-2002")

## Downloading BMX\_B.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/BMX\_B.XPT

bmx\_03 = nhanes\_load\_data("BMX", "2003-2004")

## Downloading BMX\_C.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/BMX\_C.XPT

bpx\_99 = nhanes\_load\_data("BPX", "1999-2000")

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Downloading BPX.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/BPX.XPT

bpx\_01 = nhanes\_load\_data("BPX", "2001-2002")

## Downloading BPX\_B.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/BPX\_B.XPT

bpx\_03 = nhanes\_load\_data("BPX", "2003-2004")

## Downloading BPX\_C.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/BPX\_C.XPT

paq\_99 = nhanes\_load\_data("PAQ", "1999-2000")

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Downloading PAQ.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/PAQ.XPT

paq\_01 = nhanes\_load\_data("PAQ", "2001-2002")

## Downloading PAQ\_B.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/PAQ\_B.XPT

paq\_03 = nhanes\_load\_data("PAQ", "2003-2004")

## Downloading PAQ\_C.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/PAQ\_C.XPT

diq\_99 = nhanes\_load\_data("DIQ", "1999-2000")

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Downloading DIQ.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/DIQ.XPT

diq\_01 = nhanes\_load\_data("DIQ", "2001-2002")

## Downloading DIQ\_B.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/DIQ\_B.XPT

diq\_03 = nhanes\_load\_data("DIQ", "2003-2004")

## Downloading DIQ\_C.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/DIQ\_C.XPT

smq\_99 = nhanes\_load\_data("SMQ", "1999-2000")

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.  
## Cycle 1999-2000 doesn't always follow the normal naming conventions, so skipping the file suffix check.

## Downloading SMQ.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/SMQ.XPT

smq\_01 = nhanes\_load\_data("SMQ", "2001-2002")

## Downloading SMQ\_B.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/SMQ\_B.XPT

smq\_03 = nhanes\_load\_data("SMQ", "2003-2004")

## Downloading SMQ\_C.XPT to /var/folders/5j/2zqbq7jx49g71545t3lq5bfh0000gn/T//Rtmp7jXP1o/SMQ\_C.XPT

# Joining  
data\_99 =   
 left\_join(demo\_99, bmx\_99, by = "SEQN") %>%   
 left\_join(bpx\_99, by = "SEQN") %>%   
 left\_join(paq\_99, by = "SEQN") %>%   
 left\_join(diq\_99, by = "SEQN") %>%   
 left\_join(smq\_99, by = "SEQN") %>%   
 select("SEQN", "RIDAGEYR", "RIAGENDR", "RIDRETH1", "DMDEDUC2", "INDHHINC", "BMXBMI", "BMXHT", "BMXWT", "BPXPLS", "DIQ010", "PAQ180", "SMQ020")   
  
data\_01 =   
 left\_join(demo\_01, bmx\_01, by = "SEQN") %>%   
 left\_join(bpx\_01, by = "SEQN") %>%   
 left\_join(paq\_01, by = "SEQN") %>%   
 left\_join(diq\_01, by = "SEQN") %>%   
 left\_join(smq\_01, by = "SEQN") %>%   
 select("SEQN", "RIDAGEYR", "RIAGENDR", "RIDRETH1", "DMDEDUC2", "INDHHINC", "BMXBMI", "BMXHT", "BMXWT", "BPXPLS", "DIQ010", "PAQ180", "SMQ020")   
  
data\_03 =   
 left\_join(demo\_03, bmx\_03, by = "SEQN") %>%   
 left\_join(bpx\_03, by = "SEQN") %>%   
 left\_join(paq\_03, by = "SEQN") %>%   
 left\_join(diq\_03, by = "SEQN") %>%   
 left\_join(smq\_03, by = "SEQN") %>%   
 select("SEQN", "RIDAGEYR", "RIAGENDR", "RIDRETH1", "DMDEDUC2", "INDHHINC", "BMXBMI", "BMXHT", "BMXWT", "BPXPLS", "DIQ010", "PAQ180", "SMQ020")   
  
final\_data = bind\_rows(data\_99, data\_01, data\_03) %>%   
 janitor::clean\_names() %>%   
 filter(diq010 %in% c(1,2)) %>%   
 mutate(diq010 = as.factor(diq010),   
 riagendr = as.factor(riagendr),   
 ridreth1 = as.factor(ridreth1),   
 dmdeduc2 = as.factor(dmdeduc2),   
 indhhinc = as.factor(indhhinc),   
 paq180 = as.factor(paq180),   
 smq020 = as.factor(smq020)) %>%   
 drop\_na()

Partitioning:

training\_data = final\_data$diq010 %>% createDataPartition(p = 0.7, list = F)  
train\_data = final\_data[training\_data, ]  
test\_data = final\_data[-training\_data, ]  
  
  
#Store outcome   
#alc\_cons\_train = train\_data$alc\_consumption  
#alc\_cons\_test = test\_data$alc\_consumption  
  
# store matrices excluding outcome  
#train = model.matrix(alc\_consumption~., train\_data)[,-1]  
#test = model.matrix(alc\_consumption~., test\_data)[,-1]

## Number 2: Prediction Models

Construct three prediction models to predict diabetes using the 11 features from NHANES. You will use the following three algorithms to create your prediction models:

1. Classification Tree
2. Support Vector Classifier (i.e. Support Vector Machine with a linear classifier)
3. Logistic regression.

## Classification Tree

train\_control = trainControl(method = "cv", number = 10)  
grid\_2 = expand.grid(cp = seq(0.0001, 0.011, by = 0.0001))  
tree\_diabetes = train(diq010 ~., data = train\_data, method = "rpart", trControl = train\_control, tuneGrid = grid\_2)  
tree\_diabetes$bestTune

## cp  
## 50 0.005

tree\_diabetes

## CART   
##   
## 8244 samples  
## 12 predictor  
## 2 classes: '1', '2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 7419, 7420, 7419, 7420, 7420, 7420, ...   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa   
## 0.0001 0.8719045 0.129358789  
## 0.0002 0.8725110 0.129250554  
## 0.0003 0.8749381 0.131635211  
## 0.0004 0.8767579 0.128640004  
## 0.0005 0.8779710 0.126481540  
## 0.0006 0.8782137 0.125711774  
## 0.0007 0.8811250 0.123194316  
## 0.0008 0.8812462 0.123473025  
## 0.0009 0.8829450 0.121868547  
## 0.0010 0.8835512 0.118323034  
## 0.0011 0.8846434 0.114038328  
## 0.0012 0.8857348 0.110915167  
## 0.0013 0.8857348 0.110915167  
## 0.0014 0.8877977 0.099048039  
## 0.0015 0.8877977 0.099048039  
## 0.0016 0.8884045 0.097555903  
## 0.0017 0.8892535 0.096272218  
## 0.0018 0.8926489 0.096503698  
## 0.0019 0.8942252 0.092504471  
## 0.0020 0.8942252 0.092504471  
## 0.0021 0.8954387 0.076045889  
## 0.0022 0.8958023 0.073665649  
## 0.0023 0.8958023 0.067966404  
## 0.0024 0.8958023 0.059820934  
## 0.0025 0.8976227 0.054988793  
## 0.0026 0.8976227 0.054988793  
## 0.0027 0.8976227 0.054988793  
## 0.0028 0.8983508 0.044258150  
## 0.0029 0.8985936 0.037610679  
## 0.0030 0.8993217 0.030190110  
## 0.0031 0.9006552 0.023417352  
## 0.0032 0.9006552 0.021662320  
## 0.0033 0.9006552 0.021662320  
## 0.0034 0.9006552 0.021662320  
## 0.0035 0.9011403 0.010259039  
## 0.0036 0.9011403 0.010259039  
## 0.0037 0.9011403 0.004640633  
## 0.0038 0.9011403 0.004640633  
## 0.0039 0.9011403 0.004640633  
## 0.0040 0.9011403 0.004640633  
## 0.0041 0.9011403 0.004640633  
## 0.0042 0.9016252 0.001862278  
## 0.0043 0.9016252 0.001862278  
## 0.0044 0.9016252 0.001862278  
## 0.0045 0.9016252 0.001862278  
## 0.0046 0.9016252 0.001862278  
## 0.0047 0.9023533 0.003341726  
## 0.0048 0.9023533 0.003341726  
## 0.0049 0.9027174 0.004146543  
## 0.0050 0.9027174 0.004146543  
## 0.0051 0.9025961 0.000000000  
## 0.0052 0.9025961 0.000000000  
## 0.0053 0.9025961 0.000000000  
## 0.0054 0.9025961 0.000000000  
## 0.0055 0.9025961 0.000000000  
## 0.0056 0.9025961 0.000000000  
## 0.0057 0.9025961 0.000000000  
## 0.0058 0.9025961 0.000000000  
## 0.0059 0.9025961 0.000000000  
## 0.0060 0.9025961 0.000000000  
## 0.0061 0.9025961 0.000000000  
## 0.0062 0.9025961 0.000000000  
## 0.0063 0.9025961 0.000000000  
## 0.0064 0.9025961 0.000000000  
## 0.0065 0.9025961 0.000000000  
## 0.0066 0.9025961 0.000000000  
## 0.0067 0.9025961 0.000000000  
## 0.0068 0.9025961 0.000000000  
## 0.0069 0.9025961 0.000000000  
## 0.0070 0.9025961 0.000000000  
## 0.0071 0.9025961 0.000000000  
## 0.0072 0.9025961 0.000000000  
## 0.0073 0.9025961 0.000000000  
## 0.0074 0.9025961 0.000000000  
## 0.0075 0.9025961 0.000000000  
## 0.0076 0.9025961 0.000000000  
## 0.0077 0.9025961 0.000000000  
## 0.0078 0.9025961 0.000000000  
## 0.0079 0.9025961 0.000000000  
## 0.0080 0.9025961 0.000000000  
## 0.0081 0.9025961 0.000000000  
## 0.0082 0.9025961 0.000000000  
## 0.0083 0.9025961 0.000000000  
## 0.0084 0.9025961 0.000000000  
## 0.0085 0.9025961 0.000000000  
## 0.0086 0.9025961 0.000000000  
## 0.0087 0.9025961 0.000000000  
## 0.0088 0.9025961 0.000000000  
## 0.0089 0.9025961 0.000000000  
## 0.0090 0.9025961 0.000000000  
## 0.0091 0.9025961 0.000000000  
## 0.0092 0.9025961 0.000000000  
## 0.0093 0.9025961 0.000000000  
## 0.0094 0.9025961 0.000000000  
## 0.0095 0.9025961 0.000000000  
## 0.0096 0.9025961 0.000000000  
## 0.0097 0.9025961 0.000000000  
## 0.0098 0.9025961 0.000000000  
## 0.0099 0.9025961 0.000000000  
## 0.0100 0.9025961 0.000000000  
## 0.0101 0.9025961 0.000000000  
## 0.0102 0.9025961 0.000000000  
## 0.0103 0.9025961 0.000000000  
## 0.0104 0.9025961 0.000000000  
## 0.0105 0.9025961 0.000000000  
## 0.0106 0.9025961 0.000000000  
## 0.0107 0.9025961 0.000000000  
## 0.0108 0.9025961 0.000000000  
## 0.0109 0.9025961 0.000000000  
## 0.0110 0.9025961 0.000000000  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.005.

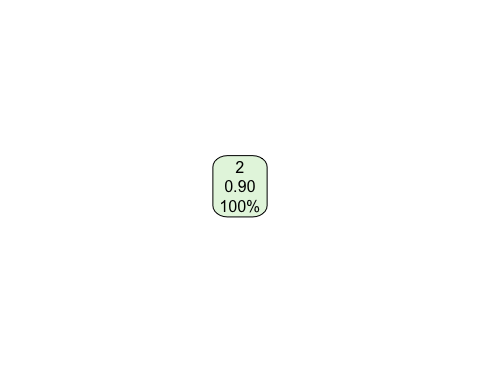
varImp(tree\_diabetes)

## rpart variable importance

## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf  
  
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning  
## -Inf

## only 20 most important variables shown (out of 39)  
##   
## Overall  
## dmdeduc29 NaN  
## indhhinc4 NaN  
## indhhinc2 NaN  
## smq0209 NaN  
## paq1802 NaN  
## bmxwt NaN  
## paq1809 NaN  
## bmxbmi NaN  
## indhhinc9 NaN  
## indhhinc99 NaN  
## smq0202 NaN  
## dmdeduc22 NaN  
## ridreth15 NaN  
## riagendr2 NaN  
## indhhinc10 NaN  
## paq1804 NaN  
## indhhinc8 NaN  
## dmdeduc25 NaN  
## ridreth12 NaN  
## bpxpls NaN

rpart.plot(tree\_diabetes$finalModel)



The optimal Cp for highest was 0.0047.

## Support Vector Classifier

svm\_diabetes = svm(diq010 ~ ., data = train\_data, kernel="linear", cost = 1, scale = TRUE)  
print(svm\_diabetes)

##   
## Call:  
## svm(formula = diq010 ~ ., data = train\_data, kernel = "linear",   
## cost = 1, scale = TRUE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 2069

new\_data = train\_data[-11]  
  
svm\_pred = predict(svm\_diabetes, new\_data)  
table(svm\_pred, train\_data$diq010)

##   
## svm\_pred 1 2  
## 1 0 0  
## 2 803 7441

misClasificError = mean(svm\_pred != train\_data$diq010, na.rm = T)  
print(paste('Accuracy Model 1',1 - misClasificError))

## [1] "Accuracy Model 1 0.902595827268316"

features = new\_data  
outcome = train\_data$diq010  
  
svm\_tune = tune(svm, diq010 ~ ., data = train\_data, kernel="linear", range=list(cost=10^(-1:1)))  
  
summary(svm\_tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 0.1  
##   
## - best performance: 0.09740644   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.1 0.09740644 0.01439658  
## 2 1.0 0.09740644 0.01439658  
## 3 10.0 0.09740644 0.01439658

Accuracy of the SVM model was 0.902595827268316.