STAT40750 – Statistical Machine Learning - Assigment 2

Daniel Williams 21203054

22/03/2022

Exercise 1 - Question 1

Compute the percentage variation in the odds of testing positive for diabetes for the case where a subject varies her BMI from 28.9 to 26.4 (assuming that all the other predictors remain constant).

Odds of a positive diabetes case is
odds = p / 1 - p
odds = Pr(no diabetes) / pr(diabetes)
odds = exp(w_0 + w_1 x_1 + w_2 x_2 + …. w_n + x_n)
cofficients from question can be converted into odds ratio scale
coefficient for mass is positive… means higher mass = more likely diabetes
can convert coefficients into odds ratio scale by taking exponential

```
w_bmi <- 0.0161362
odds_ratio_bmi <- exp(w_bmi)
odds_ratio_bmi
```

```
## [1] 1.016267
```

```
unit_increase_bmi_odd <- (odds_ratio_bmi - 1) * 100
unit_increase_bmi_odd
```

```
## [1] 1.626709
```

```
# as BMI increases one unit, the odds of positive diabetes incraese by 1.63%

BMI_1 <- 26.4
BMI_2 <- 28.9
delta_bmi <- BMI_2 - BMI_1
delta_bmi
```

```
## [1] 2.5
```

```
# Change in BMI is 2.5 units, therefore use this and multiply by the above

Q1_answer <- delta_bmi  * unit_increase_bmi_odd
Q1_answer
```

```
## [1] 4.066773
```

```
# percentage variation in the odds of testing positive for diabetes for the
# case where a subject varies her BMI from 28.9 to 26.4 (assuming that all the
# other predictors remain constant).
# Answer   4.067 %
```

Exercise 1 - Question 2

Here we take omega zero as the coefficient of the constant / intercept from the question.
Vectors a,b,c are constructed as the results from each participant within the question.

The forumla from the notes is used, that the probability of a positive result, ie Pr(Y=1), is given by

p = exp(w_0 + sum of results & coefficiants) / 1 + exp(w_0 + sum of results & coefficiants) I use that formula to work out a probability of a positive test result for each patient.

```
w_0 <- -9.6616070
w <- c(-0.0285064, 0.0459003, 0.0005647, 0.0106388, -0.0019079, 0.0161362, 1.3170565, 0.05918
74 )
a <- c(5, 155, 72, 23, 105, 26.8, 0.323, 58)
b <- c(1, 143, 86, 31, 140, 31.6, 0.719, 23)
c <- c(0, 134, 94, 41, 204, 45.6, 0.270, 28)
w_a <- w * a
w_b <- w * b
w_c <- w * c
p_a <- (exp(w_0 + sum(w_a)))   / (1 + exp(w_0 + sum(w_a)))
p_b <- (exp(w_0 + sum(w_b)))   / (1 + exp(w_0 + sum(w_b)))
p_c <- (exp(w_0 + sum(w_c)))   / (1 + exp(w_0 + sum(w_c)))

p_a
```

```
## [1] 0.8437182
```

```
p_b
```

```
## [1] 0.4509256
```

```
p_c
```

```
## [1] 0.340236
```

Exercise 1, Question 3

I will determine the optimimum tau by testing all 4 options for their sensitivity & specificity
Highest rating on sensitivity + specificity will be optimum tau

There may be a better t (ie not one of the 4 given), but we don't have the data.

```
t_1_TP <- 330
t_1_TN <- 539
t_1_FP <- 265
t_1_FN <- 42

t_3_TP <- 291
t_3_TN <- 624
t_3_FP <- 180
t_3_FN <- 81

t_5_TP <- 221
t_5_TN <- 734
t_5_FP <- 70
t_5_FN <- 151

t_7_TP <- 148
t_7_TN <- 777
t_7_FP <- 27
t_7_FN <- 224


t_1_sensitivity <- t_1_TP / (t_1_TP + t_1_FN)
t_3_sensitivity <- t_3_TP / (t_3_TP + t_3_FN)
t_5_sensitivity <- t_5_TP / (t_5_TP + t_5_FN)
t_7_sensitivity <- t_7_TP / (t_7_TP + t_7_FN)

t_1_specificity <- t_1_TN / (t_1_FP + t_1_TN)
t_3_specificity <- t_3_TN / (t_3_FP + t_3_TN)
t_5_specificity <- t_5_TN / (t_5_FP + t_5_TN)
t_7_specificity <- t_7_TN / (t_7_FP + t_7_TN)


t_1_total <- t_1_sensitivity + t_1_specificity
t_3_total <- t_3_sensitivity + t_3_specificity
t_5_total <- t_5_sensitivity + t_5_specificity
t_7_total <- t_7_sensitivity + t_7_specificity

t_1_total
```

```
## [1] 1.557495
```

```
t_3_total
```

```
## [1] 1.558377
```

```
t_5_total
```

```
## [1] 1.507021
```

```
t_7_total
```

```
## [1] 1.364267
```

Best tau from the options is t = 0.3
This has the highest sum of sensitivity & specificity.
Using t = 0.3, on the 3 subjects from part two.

if p > t, predict diabetes

using p_a, p_b, and p_c from q2

p_a > t, therefore predict diabetes
p_b > t, therefore predict diabetes p_c > t, therefore predict diabetes

Exercise 2 - Question 1 & 2

Firstly installing the required packages, and loading the data

```
library(rpart)
library(partykit)
load("C:/Users/Dan/Documents/Data Science/Masters/Stat ML/Assignment 2/data_influencer.RData"
)
```

Seperating the data into two sections a training set & a test set.

```
data0 <- data_influencer
N <- nrow(data0)   # setting variable with the number of rows of all the data
set.seed(63) # setting a seed to allow replicatable results
test <- sample(1:N, N*0.2) # randomly sampling 20% of the row numbers in our data
data_test <- data0[test,]  # subsetting with these values to create our testing data

# test data will not be used in the model fitting procedure, but instead used to validate the
model performance later

train <- setdiff(1:N, test) # the training data is then the remaining rows (ie, the ones we w
ont use for testing), this is 80% of the data and will be used to learn the parameters of the
classifiers (trees / regressions).

data <- data0[train,]  # subsetting with these values to create our training data
N_train <- nrow(data)  # checking the ammount of lines within our training data
```

Writing a customised function which will be used to assess the accuracy of the 3 classifiers below, it will be used to create a table of predicted class labels, and actual class labels of the test data, and then by computing its diagonal [correctly predicted results], divided by the total number of results, we get a measure of classification accuracy.

```
class_acc <- function(y, yhat) {
  tab <- table(y, yhat)
  return( sum(diag(tab))/sum(tab) )}
```

Setting some constants, and vectors needed within the main for loop. I am performing a K fold cross validation procedure. Here the number K represents the number of folds in the data, these are equally sized. In turn the model is fit to K-1 folds, and then tested on the fold which was dropped from the fitting process. The accuracy of the classificaiton is measured each time.

```
K <- 5  # number of folds
R <- 50 # number of replications of the K fold procedure
out <- vector("list", R)  # empty list of length R set up for the loop output
```

Main for loop for K fold cross validation procedure.

```
for ( r in 1:R ) { # running through 50 replications of the below
  acc <- matrix(NA, K, 3) # setting up an empty accuracy matrix with dimensions of K rows, fo
r each fold, and 3 columns for each of the models we are asked to fit in the question.

  folds <- rep( 1:K, ceiling(N/K) ) # creates number sequence 1,2,3,4,5 used to split the dat
a into K folds
  folds <- sample(folds) # randomises these assignments between data points
  folds <- folds[1:N_train] # subset these to get the 80% of the data we need to train on
  for ( k in 1:K ) {
    # first two lines separate our our data into the k-1 training folds, and the 1 validation
fold

    train_fold <- which(folds != k)
    validation <- setdiff(1:N_train, train_fold)

    # fitting C1 – Logistic regression model, with τ = 0.5.
    fit_log_1 <- glm(Choice ~ . , data = data, subset = train_fold, family = "binomial")

    # fitting C2 - Classification tree with default complexity settings
    fitct1 <- rpart(Choice ~ ., data = data, subset = train_fold)

    # fitting C3 - Classification tree with increased complexity (compared to C2)
    # Changed minsplit, and cp to achieve this, minsplit increases the number of branches by
 reducing the minimum number below which you will not split further, and a lower CP directly
 drives more nodes, higher complexity

    fitct2 <- rpart(Choice ~ ., data = data, subset = train_fold, minsplit = 15, cp = 0.005)

    # 3 code chunks using the trained classifiers, and testing them on the validation fold, t
au in the first instance being 0.5 as requested in the question. The results of which are the
n stored in the k'th column within the accuracy matrix created earlier, comparing actual clas
s labels, and predicted ones.

    pred_log_1 <- predict(fit_log_1, type = "response", newdata = data[validation,])
    pred_log_1 <- ifelse(pred_log_1 > 0.5, 1, 0)
    acc[k,1] <- class_acc(pred_log_1, data$Choice[validation])

    pred_ct_1 <- predict(fitct1, type = "class", newdata = data[validation,])
    acc[k,2] <- class_acc(pred_ct_1, data$Choice[validation])

    pred_ct_2 <- predict(fitct2, type = "class", newdata = data[validation,])
    acc[k,3] <- class_acc(pred_ct_2, data$Choice[validation])
  }
out[[r]] <- acc
}
```

Numerical summaries of the 3 models performance

```
avg <- t( sapply(out, colMeans) )
head(avg, 5)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.7594405 0.7906029 0.7781613
## [2,] 0.7541687 0.7945024 0.7698134
## [3,] 0.7586931 0.8004266 0.7835043
## [4,] 0.7548039 0.7721493 0.7773446
## [5,] 0.7466815 0.7897125 0.7714455
```

Above we calculate an average of the column means, so for each of the 50 replications, we see the classification accuracy of the 3 classifiers (cols), here we have limited the response to the first 5 replications purely to save space. We can see that the classifiers are fairly close in their performance measures.

```
mean_acc <- colMeans(avg)
mean_acc
```

```
## [1] 0.7568982 0.7905736 0.7760705
```

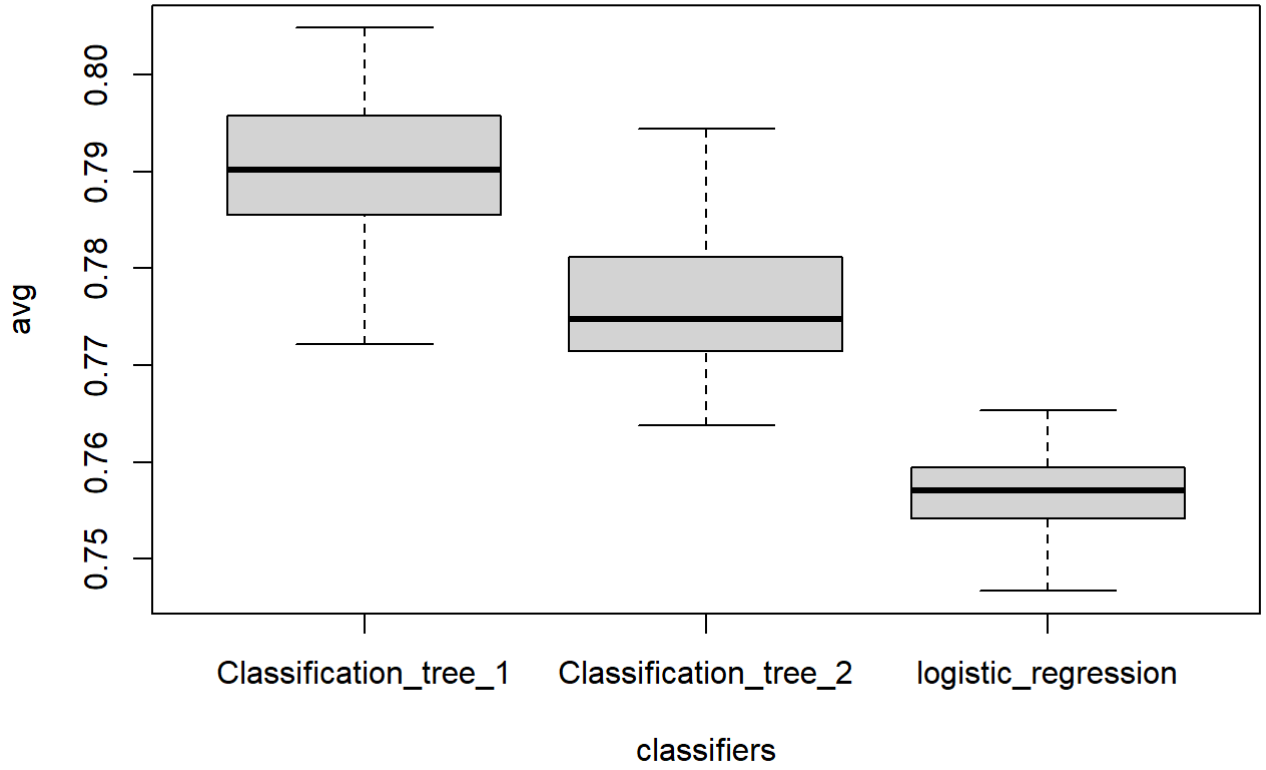
Above is the average accuracy across the 50 replications of the K fold cross validation, we can see that classifier 2 seems to have performed the best.

```
sd_acc <- apply(avg, 2, sd)/sqrt(R)
sd_acc
```

```
## [1] 0.0005519893 0.0009990565 0.0009811962
```

Relatively low errors are found in the results, born out of our cross validation approach, and high replication value.

```
classifiers <- c("logistic_regression", "Classification_tree_1", "Classification_tree_2")
mat <- data.frame( avg = c(avg), classifiers = rep(classifiers, each = R) )
boxplot(avg ~ classifiers, mat)
```



Graphical representation of the numerical summaries above. We see that based on this K fold cross validation method, the first classification tree, with the default complexity settings has performed the best.

This tree has a classification accuracy of ~ 0.79, meaning that 79% of the time it correctly predicts who is the most influential for new pairs of individuals in the social network.

Relatively speaking, looking at the length of the IQR, and the min / max vales, along with the standard deviation of the prediction accuracies we can see that we have a fairly precise result, only varying by a maximum of ~4%, evidently this would be very dependant on the situation, but for this task, we can be confident that we have trained a suitable classifier.

The logisitic regression result has the lowest variability in its estimates.

There is no overlap of the IQR's between tree_1 and tree_2 allowing us to say with some level of confidence that tree_1 is the better performing classifier.

```
fit_best <- rpart(Choice ~ ., data = data)
pred_best <- predict(fit_best, type = "class", newdata = data_test)
class_acc(pred_best, data_test$Choice)
```

```
## [1] 0.7708333
```

Above we see that when we test the best classifier (classification_tree_1), on the whole of the test data (20% of all the data), we get a predictive performance measure of 0.771. This is really towards the bottom end of the results from the k fold results. We expect to correctly identify 77 in 100 of the most influential within new pairs of people.

The estimate of the generalisation error is 23% 0.23

This is potentially a result of using the k fold system, which trades computational complexity for bias in the estimated error.