HOMEWORK 10. ISYE 6501

Guillermo de la Hera Casado

October 27th, 2019

Question 14.1

Exploring the missing values

Let's first load the breast cancer data:

- The first column (sample code number) it's not of interest for our classification models. Therefore we will exclude it.
- The *class* variable is 2 for benign, 4 for malignant. Let's make it become 0 for benign, 1 for malignant.

```r
set.seed(101) # Set Seed so that same sample can be reproduced in future also
breastC <- read.table("breast-cancer-wisconsin.data.txt", header = FALSE,
                      col.names = c("sample_code",
                                    "clump_thick", "uni_cellsize", "uni_cellshape",
                                    "marg_adhesion", "epi_cellsize", "bare_nuclei",
                                    "bland_chro", "normal_nucleoli", "mitoses","type"),sep = ",")
breastC <- breastC[,2:ncol(breastC)]
breastC$type <- apply(breastC, 1, function(x) {ifelse(x[10] == 2, 0, 1)})
head(breastC)
```

```
##   clump_thick uni_cellsize uni_cellshape marg_adhesion epi_cellsize
## 1           5            1             1             1            2
## 2           5            4             4             5            7
## 3           3            1             1             1            2
## 4           6            8             8             1            3
## 5           4            1             1             3            2
## 6           8           10            10             8            7
##   bare_nuclei bland_chro normal_nucleoli mitoses type
## 1           1          3               1       1    0
## 2          10          3               2       1    0
## 3           2          3               1       1    0
## 4           4          3               7       1    0
## 5           1          3               1       1    0
## 6          10          9               7       1    1
```

```r
str(breastC)
```

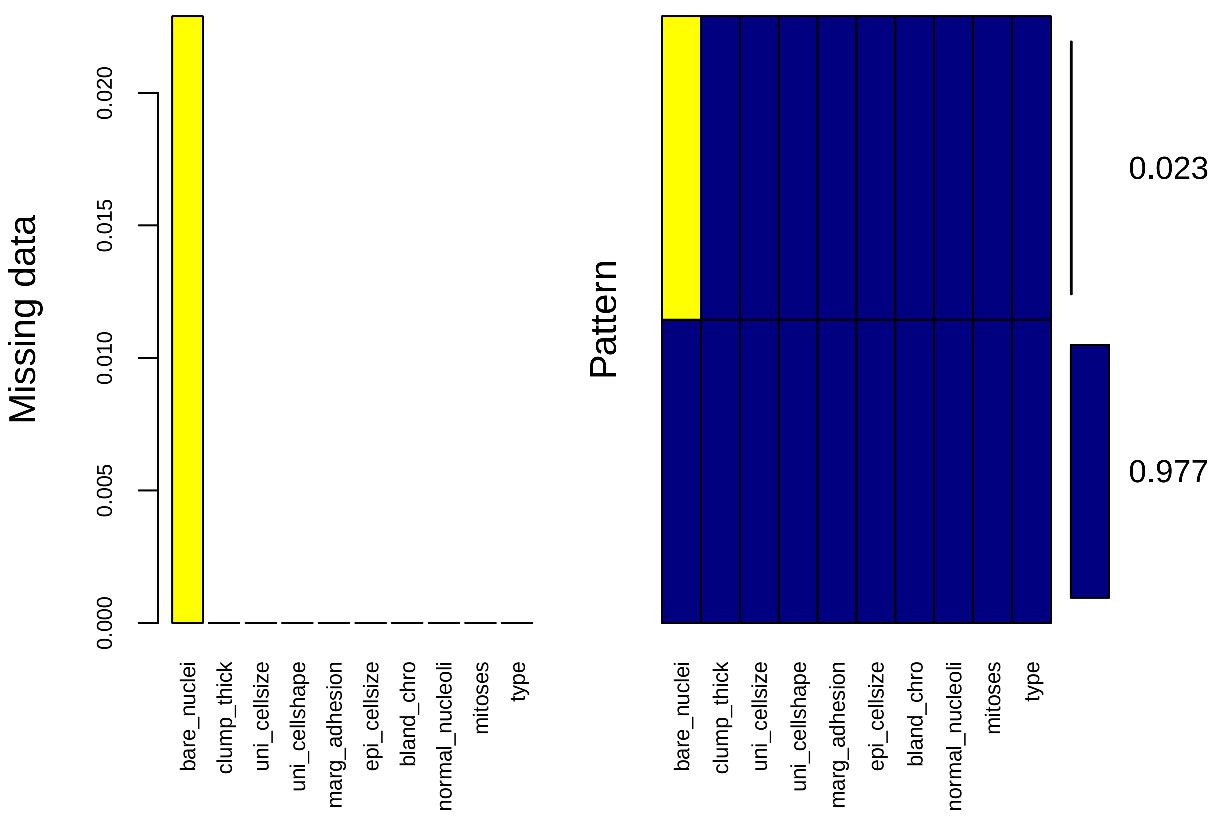
```
## 'data.frame':    699 obs. of  10 variables:
##  $ clump_thick    : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ uni_cellsize   : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ uni_cellshape  : int  1 4 1 8 1 10 1 2 1 1 ...
##  $ marg_adhesion  : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ epi_cellsize   : int  2 7 2 3 2 7 2 2 2 2 ...
##  $ bare_nuclei    : Factor w/ 11 levels "?","1","10","2",..: 2 3 4 6 2 3 3 2 2 2 ...
##  $ bland_chro     : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ normal_nucleoli: int  1 2 1 7 1 7 1 1 1 1 ...
##  $ mitoses        : int  1 1 1 1 1 1 1 1 5 1 ...
##  $ type           : num  0 0 0 0 0 1 0 0 0 0 ...
```

We can identify quite easily that something is going on with column: **bare_nuclei**. It gets the value: *?* for some of the observations. Let's give to these rows value = NA as it may be easier to work with it:

```
breastC$bare_nuclei <- apply(breastC, 1, function(x) {ifelse(x[6] == "?", NA, as.integer(x[6]))})
```

Let's now explore where the missing values are and the percentage out of total observations. For that task, we will use **MICE** library, that stands for *Multivariate Imputation by Chained Equations* and also **VIM**, that stands for *Visualization and Imputation of Missing Values*:

```
mice_plot <- aggr(breastC, col=c('navyblue','yellow'),
                    numbers=TRUE, sortVars=TRUE,
                    labels=names(breastC), cex.axis=.7,
                    gap=3, ylab=c("Missing data","Pattern"))
```



```
##
##  Variables sorted by number of missings:
##         Variable      Count
##      bare_nuclei 0.02288984
##      clump_thick 0.00000000
##     uni_cellsize 0.00000000
##    uni_cellshape 0.00000000
##    marg_adhesion 0.00000000
##     epi_cellsize 0.00000000
##       bland_chro 0.00000000
##  normal_nucleoli 0.00000000
##          mitoses 0.00000000
##             type 0.00000000
```

From the results, we can see that the only column with missing values is *bare_nuclei* and the percentage of missing values is only **2.3%**. As we learnt from the lectures, a value of 5% or higher missing data points would require further investigation and question the validity of the imputation. Since the value is smaller than 5%, we can move forward and explore imputation methods.

With regards to whether there is any bias on the missing values, we could start by looking at the relationship between the missing values and the class:

```
prop.table(table(breastC$bare_nuclei, breastC$type, useNA="ifany"),1)
```

```
##
##                   0          1
##   1    0.96268657 0.03731343
##   2    0.70000000 0.30000000
##   3    0.50000000 0.50000000
##   4    0.31578947 0.68421053
##   5    0.33333333 0.66666667
##   6    0.00000000 1.00000000
##   7    0.12500000 0.87500000
##   8    0.09523810 0.90476190
##   9    0.00000000 1.00000000
##   10   0.02272727 0.97727273
##   <NA> 0.87500000 0.12500000
```

As we can see, there is no clear bias on NA values towards a specific class. *Bare_nuclei* mode is: 1, and shows a majority of benign observations. NA distribution just follows the same logic.

What about the row indexes for the missing values? It doesn't look like the rows with empty values are consecutive or follow a pattern;

```
na_index <- which(is.na(breastC$bare_nuclei))
na_index
```

```
##  [1]  24  41 140 146 159 165 236 250 276 293 295 298 316 322 412 618
```

Use the mean/mode imputation

From the data structure we could infer that our predictors take values from 1 to 10 and would be likely categorical and not continuous. From the documentation it's not clear whether there is any ordering. For these reasons I have decided to use the mode as imputation method. Let's work out the mode and assign it to the missing values:

```
table(breastC$bare_nuclei)
```

```
##
##   1   2   3   4   5   6   7   8   9  10
## 402  30  28  19  30   4   8  21   9 132
```

```
breastC_mode <- data.frame(breastC)
breastC_mode[na_index,6] <- 1
breastC_mode[na_index,6]
```

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

As we can see now, **we have made all the 16 missing values = 1**, that is the mode.

Let's look at the performance for this imputation method. As measure for success I have chosen *recall (sensitivity)*. The reason why is that in this context **we want to minimize False Negatives** (patient with breast cancer being incorrectly diagnosed as benign). The formula is: TP / (TP+FN)

If we use KKNN as classification algorithm, we will find out during validation that the best combination is k=7 and a triangular kernel. If we use this configuration on the whole training set to predict the test set, we will find out that **recall is 100% and accuracy = 97.1%**

```
sample <- sample.split(breastC_mode$type, SplitRatio = 0.8)
trainSet <- subset(breastC_mode, sample == TRUE)
testSet <- subset(breastC_mode, sample == FALSE)

valModel<- train.kknn(as.factor(type)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale=FALSE, distance=2)
print(valModel$best.parameters$kernel)
```

```
## [1] "triangular"
```

```
print(valModel$best.parameters$k)
```

```
## [1] 7
```

```
model <- kknn(as.factor(type)~., train=trainSet,
              test = testSet, kernel = "triangular", k=7, scale = FALSE, distance=2)
caret::confusionMatrix(table(model$fit, testSet$type), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##      0  1
##   0 88  0
##   1  4 48
##
##                Accuracy : 0.9714
##                  95% CI : (0.9285, 0.9922)
##     No Information Rate : 0.6571
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9378
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 1.0000
##             Specificity : 0.9565
##          Pos Pred Value : 0.9231
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3429
##          Detection Rate : 0.3429
##    Detection Prevalence : 0.3714
##       Balanced Accuracy : 0.9783
##
##        'Positive' Class : 1
##
```

Use regression to input values from the missing data

Let's take the class variable out of the predictors, so that it's not used for the imputation. Otherwise we would be introducing bias at the time of determining the quality of the model.

We will build a linear regression with the remaining variables, being *bare_nuclei* the response. We will be careful to use for training the model the instances where data is not missing, and then use the model to predict the values for the missing instances.

```r
breastC_reg <- data.frame(breastC[,1:9])
breastC_regper <- data.frame(breastC[,1:9])

breastLM <- lm(formula = bare_nuclei~., data = breastC_reg[-na_index,])
summary(breastLM)
```

```
##
## Call:
## lm(formula = bare_nuclei ~ ., data = breastC_reg[-na_index, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.7316 -0.9426 -0.3002  0.6725  8.6998
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -0.616652   0.194975  -3.163  0.00163 **
## clump_thick      0.230156   0.041691   5.521 4.83e-08 ***
## uni_cellsize    -0.067980   0.076170  -0.892  0.37246
## uni_cellshape    0.340442   0.073420   4.637 4.25e-06 ***
## marg_adhesion    0.339705   0.045919   7.398 4.13e-13 ***
## epi_cellsize     0.090392   0.062541   1.445  0.14883
## bland_chro       0.320577   0.059047   5.429 7.91e-08 ***
## normal_nucleoli  0.007293   0.044486   0.164  0.86983
## mitoses         -0.075230   0.059331  -1.268  0.20524
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.274 on 674 degrees of freedom
## Multiple R-squared:  0.615,  Adjusted R-squared:  0.6104
## F-statistic: 134.6 on 8 and 674 DF,  p-value: < 2.2e-16
```

```r
predicted <- predict(breastLM, breastC_reg[na_index,])
breastC_reg[na_index, 6] <- round(predicted,0)
breastC_reg[na_index,6]
```

```
##  [1] 5 8 1 2 1 2 3 2 2 6 1 2 6 2 1 1
```

```r
breastC_reg$type <- breastC[,10]
```

As we can see now, we have assigned to each one of the 16 missing values the result of the linear regression model, rounding up so it's an integer between 1 and 10.

Looking at the validation phase, it advises a triangular kernel with 10 neighbours. If we check the final performance of this method, **it provides a very similar performance when compared to mean/mode imputation.**

```r
sample <- sample.split(breastC_reg$type, SplitRatio = 0.8)
trainSet <- subset(breastC_reg, sample == TRUE)
testSet <- subset(breastC_reg, sample == FALSE)

valModel<- train.kknn(as.factor(type)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale=FALSE, distance=2)
```

```
print(valModel$best.parameters$kernel)
```

```
## [1] "triangular"
```

```
print(valModel$best.parameters$k)
```

```
## [1] 10
```

```
model <- kknn(as.factor(type)~., train=trainSet,
               test = testSet, kernel = "triangular", k=10, scale = FALSE, distance=2)
caret::confusionMatrix(table(model$fit, testSet$type), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##       0  1
##    0 88  0
##    1  4 48
##
##                Accuracy : 0.9714
##                  95% CI : (0.9285, 0.9922)
##     No Information Rate : 0.6571
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9378
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 1.0000
##             Specificity : 0.9565
##          Pos Pred Value : 0.9231
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3429
##          Detection Rate : 0.3429
##    Detection Prevalence : 0.3714
##       Balanced Accuracy : 0.9783
##
##        'Positive' Class : 1
##
```

Use regression with perturbation to input values from the missing data

As explained during office hours we will use a random normal distribution in which the predicted values are the means and the standard deviation is the standard deviation of the predicted values. We will use the *rnorm* function.

Applying the perturbation makes some values become zero or negative, for those, we adjust to value = 1, as we want all to be in the range [1-10]

```
sd(predicted)
```

```
## [1] 2.246945
```

```
norm_per <- round(rnorm(length(na_index),predicted, sd(predicted)),0)
norm_per
```

```
##  [1]  7  8  1  1 -1  3  4  5  5  4  2  1  8  3  2 -1
```

```r
norm_per <- sapply(norm_per, function(x) {ifelse(x <= 0, 1, x)})
breastC_regper[na_index,6] <- norm_per
breastC_regper[na_index,6]
```

```
##  [1] 7 8 1 1 1 3 4 5 5 4 2 1 8 3 2 1
```

```r
breastC_regper$type <- breastC[,10]
```

Looking at the validation phase, it advises a triangular kernel with 10 neighbours. **If we check the final performance of this method, it provides a bit better accuracy (98.5%), but at the cost of dropping recall from 100% to 97.9%**

```r
sample <- sample.split(breastC_regper$type, SplitRatio = 0.8)
trainSet <- subset(breastC_regper, sample == TRUE)
testSet <- subset(breastC_regper, sample == FALSE)

valModel<- train.kknn(as.factor(type)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale=FALSE, distance=2)
print(valModel$best.parameters$kernel)
```

```
## [1] "triangular"
```

```r
print(valModel$best.parameters$k)
```

```
## [1] 10
```

```r
model <- kknn(as.factor(type)~., train=trainSet,
              test = testSet, kernel = "triangular", k=10, scale = FALSE, distance=2)
caret::confusionMatrix(table(model$fit, testSet$type), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##      0  1
##   0 91  1
##   1  1 47
##
##                Accuracy : 0.9857
##                  95% CI : (0.9493, 0.9983)
##     No Information Rate : 0.6571
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9683
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9792
##             Specificity : 0.9891
##          Pos Pred Value : 0.9792
##          Neg Pred Value : 0.9891
##              Prevalence : 0.3429
##          Detection Rate : 0.3357
##    Detection Prevalence : 0.3429
##       Balanced Accuracy : 0.9841
##
##        'Positive' Class : 1
```

```
##
```

Take data that remains when missing values are removed

It's quite straightforward to remove the rows where there is missing values.

Looking at the validation phase, it advises a triangular kernel with 9 neighbours. **If we check the final performance of this method, it provides the worst accuracy (96.3%) and the worst recall (0.937)**

The big learning here is to be careful when dropping rows due to empty values. The other predictors that have valid figures may be very relevant to predict the response and by dropping rows we are providing the model with less predictive power.

```r
breastC_del <- data.frame(breastC)
breastC_del <- breastC_del[-na_index,]

sample <- sample.split(breastC_del$type, SplitRatio = 0.8)
trainSet <- subset(breastC_del, sample == TRUE)
testSet <- subset(breastC_del, sample == FALSE)

valModel<- train.kknn(as.factor(type)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale=FALSE, distance=2)
print(valModel$best.parameters$kernel)
```

```
## [1] "triangular"
```

```r
print(valModel$best.parameters$k)
```

```
## [1] 9
```

```r
model <- kknn(as.factor(type)~., train=trainSet,
               test = testSet, kernel = "triangular", k=9, scale = FALSE, distance=2)
caret::confusionMatrix(table(model$fit, testSet$type), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##       0  1
##    0 87  3
##    1  2 45
##
##                Accuracy : 0.9635
##                  95% CI : (0.9169, 0.988)
##     No Information Rate : 0.6496
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9194
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9375
##             Specificity : 0.9775
##          Pos Pred Value : 0.9574
##          Neg Pred Value : 0.9667
##              Prevalence : 0.3504
##          Detection Rate : 0.3285
##    Detection Prevalence : 0.3431
##       Balanced Accuracy : 0.9575
##
##        'Positive' Class : 1
##
```

Use a binary variable to indicate when data is missing

We will create a new binary variable called *isMissing*. It will be equal to one when *bare_nuclei* value is NA, or zero otherwise. In this case we need to scale the data, since we are introducing a new predictor that is not anymore in the range [1,10] We also need to specify *bare_nuclei* equal to 0 in the cases when it's NA. Otherwise, kknn function doesn't support NA values during prediction.

Looking at the validation phase, it advises a triangular kernel with 30 neighbours. **If we check the final performance of this method, it provides also a worse accuracy (95.7%) and recall (0.937)**

```
breastC_bin <- data.frame(breastC)
breastC_bin$isMissing <- apply(breastC_bin, 1, function(x) {ifelse(is.na(x[6]), 1, 0)})
breastC_bin[na_index,6] <- 0
sample <- sample.split(breastC_bin$type, SplitRatio = 0.8)
trainSet <- subset(breastC_bin, sample == TRUE)
testSet <- subset(breastC_bin, sample == FALSE)
valModel<- train.kknn(as.factor(type)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale=TRUE, distance=2)
print(valModel$best.parameters$kernel)
```

```
## [1] "triangular"
```

```
print(valModel$best.parameters$k)
```

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

```
model <- kknn(as.factor(type)~., train=trainSet,
              test = testSet, kernel = "triangular", k=30, scale = TRUE, distance=2)
caret::confusionMatrix(table(model$fit, testSet$type), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
##      0  1
##   0 89  3
##   1  3 45
##
##                Accuracy : 0.9571
##                  95% CI : (0.9091, 0.9841)
##     No Information Rate : 0.6571
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9049
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9375
##             Specificity : 0.9674
##          Pos Pred Value : 0.9375
##          Neg Pred Value : 0.9674
##              Prevalence : 0.3429
##          Detection Rate : 0.3214
##    Detection Prevalence : 0.3429
##       Balanced Accuracy : 0.9524
##
##        'Positive' Class : 1
##
```

Question 15.1. Describe a real life situation in which optimization would be appropiate.

Optimization is very applicable to marketing campaigns. There is usually:

- Few variables to optimize e.g. outbound agent performance, number of campaigns…
- Constraints: total budget for campaigns, maximum number of agents available….
- Function to optimize: it could be based on revenue maximization.

It would be really valuable to implement it in my current company, as everyone wants to obtain the maximum revenue out of marketing activities at the lowest budget.