**Example of Data Cleaning, Imputation, with Classification by KNN and SVM**

Brad Wallace

2020-03-20

# Clear environment  
rm(list = ls())  
  
# Setting the random number generator seed so that our results are reproducible  
set.seed(1)  
  
# First, read in the data  
data=read.table("F:/Documents/! IMPORTANT/ISYE 6501/Working/breast-cancer-wisconsin.data.txt",  
 col.names = c("Patient#", "Clump Thickness", "Uniformity of Cell Size",  
 "Uniformity of Cell Shape", "Marginal Adhesion",  
 "Single Epithelial cell SiZe", "Bare Nuclei",  
 "Bland Chromatin", "Normal Nucleoli",  
 "Mitoses", "Benign/Malignant"),  
 sep = ",", stringsAsFactors = FALSE, header = FALSE)  
  
# Count amount of "?" missing values  
sum(data=="?")

## [1] 16

# We know missing values are part of Variable #7, Bare Nuclei  
impute\_index <- which(data$Bare.Nuclei == "?")  
  
# Alternatively, we can search all columns if "?" is set to NA  
data\_with\_NA = data  
data\_with\_NA$Bare.Nuclei[impute\_index]= NA  
apply(is.na(data\_with\_NA), 2, which)

## $Patient.  
## integer(0)  
##   
## $Clump.Thickness  
## integer(0)  
##   
## $Uniformity.of.Cell.Size  
## integer(0)  
##   
## $Uniformity.of.Cell.Shape  
## integer(0)  
##   
## $Marginal.Adhesion  
## integer(0)  
##   
## $Single.Epithelial.cell.SiZe  
## integer(0)  
##   
## $Bare.Nuclei  
## [1] 24 41 140 146 159 165 236 250 276 293 295 298 316 322 412 618  
##   
## $Bland.Chromatin  
## integer(0)  
##   
## $Normal.Nucleoli  
## integer(0)  
##   
## $Mitoses  
## integer(0)  
##   
## $Benign.Malignant  
## integer(0)

# The results confirm that all missing values are in V7, Bare Nuclei  
  
# % of how much data we are missing - No more than 5% per factor  
paste(length(impute\_index)/nrow(data)\*100, "%")

## [1] "2.28898426323319 %"

data\_clean <- data[-impute\_index,]  
data\_missing <- data[impute\_index,]  
  
# Force V7 to integer type for functions, linear regression model, etc.  
data$Bare.Nuclei <- as.integer(data$Bare.Nuclei)

## Warning: NAs introduced by coercion

data\_clean$Bare.Nuclei <- as.integer(data\_clean$Bare.Nuclei)  
  
data$Patient.[impute\_index]

## [1] 1057013 1096800 1183246 1184840 1193683 1197510 1241232 169356 432809  
## [10] 563649 606140 61634 704168 733639 1238464 1057067

# Since we want to eventually use this data to classify V11, it is important to check out  
# if the missing data is biased in some way.  
  
table(data$Benign.Malignant)

##   
## 2 4   
## 458 241

table(data\_clean$Benign.Malignant)

##   
## 2 4   
## 444 239

table(data\_missing$Benign.Malignant)

##   
## 2 4   
## 14 2

sum(data$Benign.Malignant == 2)/nrow(data)

## [1] 0.6552217

sum(data\_clean$Benign.Malignant == 2)/nrow(data\_clean)

## [1] 0.6500732

sum(data\_missing$Benign.Malignant == 2)/nrow(data\_missing)

## [1] 0.875

# It appears that the Benign/Malignant ratio is higher in the missing data at 0.875 = 87.5% Benign,   
# however the split in the clean\_data, which will be used for training, has....  
# ... 65% Benign = 2 and 35% Malignant = 4. Therefore, a high-quality model should be able to predict  
# 2 or 4 reliably with data having a 35/65 split between cancer classifications  
  
#table(data\_clean)  
  
##### 14.1.1.1 Mean imputation #####  
mean = mean(as.numeric(data\_clean$Bare.Nuclei))  
mean\_rounded = round(mean)  
# replace the missing values  
data\_mean\_impute <- data  
data\_mean\_impute$Bare.Nuclei[impute\_index] = mean\_rounded  
data\_mean\_impute$Bare.Nuclei[impute\_index]

## [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

# data\_mean\_impute is now ready for use!  
  
##### 14.1.1.2 Mode imputation #####  
mode = as.integer(which.max(data\_clean$Bare.Nuclei))  
# replace the missing values  
data\_mode\_impute <- data  
data\_mode\_impute$Bare.Nuclei[impute\_index] = mode  
data\_mode\_impute$Bare.Nuclei[impute\_index]

## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

# data\_mode\_impute is now ready for use!  
  
##### 14.1.2 Linear Regression imputation #####  
  
# 10 fold CV Linear Regression, 80% Train & 20% Test (for measuring quality)  
  
# Load 'caret' package for CV-lm  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

# Random split 80% Data for Training and 20% for Test.   
  
train\_index <- sample.int(n = nrow(data\_clean), size = floor(0.80\*nrow(data\_clean)), replace = F)  
data\_train <- data\_clean[train\_index,]  
data\_test <- data\_clean[-train\_index,]  
  
# Perform 10-fold CV, repeat 10, Linear Regression on all variables except...  
# ... Patient Number and Response Variable (Classification)  
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)  
model\_regression <- train(Bare.Nuclei~.,   
 data=data\_train[,c(-1,-11)],   
 trControl=ctrl,  
 method="lm",  
 preProcess = c("center", "scale"))  
  
# Assess 10-fold CV, repeat 10, best model quality  
summary(model\_regression$finalModel)

##   
## Call:  
## lm(formula = .outcome ~ ., data = dat)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -9.4228 -0.9791 -0.3274 0.8308 8.6726   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 3.60806 0.10107 35.699 < 2e-16 \*\*\*  
## Clump.Thickness 0.69291 0.13580 5.103 4.66e-07 \*\*\*  
## Uniformity.of.Cell.Size -0.01061 0.26767 -0.040 0.96841   
## Uniformity.of.Cell.Shape 0.83021 0.25734 3.226 0.00133 \*\*   
## Marginal.Adhesion 0.84491 0.15126 5.586 3.70e-08 \*\*\*  
## Single.Epithelial.cell.SiZe 0.17918 0.15817 1.133 0.25779   
## Bland.Chromatin 0.82429 0.17191 4.795 2.11e-06 \*\*\*  
## Normal.Nucleoli 0.06086 0.15770 0.386 0.69973   
## Mitoses -0.13538 0.11692 -1.158 0.24744   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 2.362 on 537 degrees of freedom  
## Multiple R-squared: 0.5931, Adjusted R-squared: 0.587   
## F-statistic: 97.83 on 8 and 537 DF, p-value: < 2.2e-16

predict\_test <- predict(model\_regression, newdata=data\_test)  
  
SSE = sum((predict\_test - data\_test[,7])^2)  
SStot = sum((data\_test[,7] - mean(data\_test[,7]))^2)  
R2 <- 1 - SSE/SStot  
R2

## [1] 0.700676

R2\_adj = 1 - ((1 - R2)\*(nrow(data\_train)-1)/(nrow(data\_train)-8-1))  
R2\_adj

## [1] 0.6962168

# Predict missing data values for Variable 7, Bare Nuclei  
regression\_pred <- predict(model\_regression, newdata=data\_missing)  
  
# Round the regression prediction values to integers  
regression\_pred\_rounded = round(regression\_pred)  
# replace the missing values  
data\_regression\_impute <- data  
data\_regression\_impute$Bare.Nuclei[impute\_index] = regression\_pred\_rounded  
data\_regression\_impute$Bare.Nuclei[impute\_index]

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

# data\_regression\_impute is now ready for use!  
  
# Further modeling --->  
# Linear Regression could be undertaken with PCA and/or removal of non-significant factors  
  
##### 14.1.3 CV-Linear Regression With Perturbation #####  
  
# Additional Perturbation of normal distribution (mean=0, sd=1), no scaling factor  
perturb <- rnorm(length(impute\_index),0,1)  
reg\_pred\_with\_perturb <- regression\_pred + perturb  
  
# Clip (correct) the perturbation values to range [1,11]   
reg\_pred\_perturb\_corrected <- pmax(1, pmin(reg\_pred\_with\_perturb, 10))  
  
# Round the clipped perturbation values to integers  
reg\_pred\_perturb\_corr\_rounded <- round(reg\_pred\_perturb\_corrected)  
# replace the missing values  
data\_regression\_impute\_perturb <- data  
data\_regression\_impute\_perturb$Bare.Nuclei[impute\_index] = reg\_pred\_perturb\_corr\_rounded  
data\_regression\_impute\_perturb$Bare.Nuclei[impute\_index]

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

# data\_regression\_impute\_perturb is now ready for use!  
  
# Further modeling --->  
# Perturbation could take other distribution forms e.g. uniform, binomial etc.  
# Perturbation could also be scaled by a coefficient to   
# increase/decrease effect of "randomness"  
  
##### 14.1.4.2 Classification Modeling #####  
# Using data that remains after data points with missing values are removed  
# i.e. data\_clean. Predict Classification of V11 (Benign/Malignant) utilizing  
# all Variables except patient number (V1)  
  
##### K-nearest neighbors model with k-fold cross validation #####  
  
kmax <- 35  
  
# note that the double use of "k" (k-nearest neighbors and k-fold cross validation) can be confusing  
  
data\_clean\_no\_labels <- data\_clean  
colnames(data\_clean\_no\_labels) <- c("V1","V2","V3","V4","V5","V6","V7",  
 "V8","V9","V10","V11")  
  
colnames(data\_missing) <- c("V1","V2","V3","V4","V5","V6","V7",  
 "V8","V9","V10","V11")  
  
  
# Split it into 80% training data and 20% test data  
data\_train\_1 <- data\_clean\_no\_labels[train\_index,]  
data\_test\_1 <- data\_clean\_no\_labels[-train\_index,]  
  
  
knn\_fit <- train(as.factor(V11)~V2+V3+V4+V5+V6+V7+V8+V9+V10,  
 data = data\_train\_1,  
 method = "knn",  
 trControl = trainControl(  
 method="repeatedcv", # k-fold cross validation  
 number=10, # number of folds (k in cross validation)  
 repeats=10), # number of times to repeat k-fold cross validation  
 preProcess = c("center", "scale"), # standardize the data  
 tuneLength = kmax) # max number of neighbors (k in nearest neighbor)  
  
# Assess 10-fold CV, repeat 10, best model quality  
# print knn model summary   
knn\_fit

## k-Nearest Neighbors   
##   
## 546 samples  
## 9 predictor  
## 2 classes: '2', '4'   
##   
## Pre-processing: centered (9), scaled (9)   
## Resampling: Cross-Validated (10 fold, repeated 10 times)   
## Summary of sample sizes: 491, 492, 491, 491, 492, 492, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.9710633 0.9374366  
## 7 0.9692352 0.9334765  
## 9 0.9705113 0.9362457  
## 11 0.9716259 0.9385633  
## 13 0.9699759 0.9351358  
## 15 0.9741882 0.9441828  
## 17 0.9736427 0.9430250  
## 19 0.9736495 0.9430469  
## 21 0.9729188 0.9414549  
## 23 0.9725518 0.9405694  
## 25 0.9714542 0.9381701  
## 27 0.9708951 0.9369342  
## 29 0.9697941 0.9344449  
## 31 0.9694134 0.9336003  
## 33 0.9677736 0.9299437  
## 35 0.9668646 0.9278967  
## 37 0.9674066 0.9291198  
## 39 0.9659487 0.9258100  
## 41 0.9654033 0.9245834  
## 43 0.9639386 0.9212954  
## 45 0.9632046 0.9197085  
## 47 0.9622821 0.9176550  
## 49 0.9621070 0.9172131  
## 51 0.9617400 0.9163282  
## 53 0.9613696 0.9155177  
## 55 0.9610060 0.9146975  
## 57 0.9611912 0.9150802  
## 59 0.9610093 0.9146657  
## 61 0.9610093 0.9146657  
## 63 0.9611945 0.9150585  
## 65 0.9613728 0.9153973  
## 67 0.9608274 0.9141709  
## 69 0.9606456 0.9137561  
## 71 0.9604604 0.9132927  
## 73 0.9599116 0.9121239  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 15.

# best k neighbors  
knn\_fit$bestTune

## k  
## 6 15

# accuracy linked with best k neighbors kknn model  
max(knn\_fit$results$Accuracy)

## [1] 0.9741882

predict\_test\_1 <- predict(knn\_fit, newdata=data\_test\_1)  
  
# print predictions  
predict\_test\_1

## [1] 2 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 4 4 2 2 2 4  
## [38] 4 4 4 4 2 4 2 4 4 2 2 4 2 2 4 4 4 2 4 4 2 2 4 4 4 2 4 4 4 2 4 2 4 2 4 2 2  
## [75] 4 2 2 2 4 2 2 2 4 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2  
## [112] 2 2 2 2 2 2 2 4 2 4 4 2 2 4 2 4 2 2 2 2 2 2 2 2 2 2  
## Levels: 2 4

# print test set accuracy  
sum(predict\_test\_1 == data\_test\_1[,11])/nrow(data\_test\_1)

## [1] 0.9489051

##### Support Vector Machine with k-fold cross validation #####  
c\_values <- 10 # 10 algorithm-generated C levels   
  
svm\_fit <- train(as.factor(V11)~V2+V3+V4+V5+V6+V7+V8+V9+V10,  
 data = data\_clean\_no\_labels,  
 method = "svmLinear",  
 trControl = trainControl(  
 method="repeatedcv", # k-fold cross validation  
 number=10, # number of folds (k in cross validation)  
 repeats=20), # number of times to repeat k-fold cross validation  
 preProcess = c("center", "scale"), # standardize the data  
 tuneLength = c\_values)  
  
# Assess 10-fold CV, repeat 10, best model quality  
# print svm model summary   
svm\_fit

## Support Vector Machines with Linear Kernel   
##   
## 683 samples  
## 9 predictor  
## 2 classes: '2', '4'   
##   
## Pre-processing: centered (9), scaled (9)   
## Resampling: Cross-Validated (10 fold, repeated 20 times)   
## Summary of sample sizes: 614, 614, 615, 614, 615, 615, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.9664208 0.9263654  
##   
## Tuning parameter 'C' was held constant at a value of 1

# best C value  
svm\_fit$bestTune

## C  
## 1 1

# accuracy linked with best svm model  
max(svm\_fit$results$Accuracy)

## [1] 0.9664208

predict\_test\_2 <- predict(svm\_fit, newdata=data\_test\_1)  
  
# print predictions  
predict\_test\_2

## [1] 2 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 4 2 4 4 2 2 2 4  
## [38] 4 4 4 4 2 4 2 4 4 2 2 4 2 2 4 4 4 2 4 4 2 2 4 4 4 2 4 4 4 2 4 2 4 2 4 2 2  
## [75] 4 2 2 2 4 2 2 2 4 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2  
## [112] 2 2 2 2 2 2 2 4 2 4 4 2 2 4 2 4 2 2 2 2 2 2 2 2 2 2  
## Levels: 2 4

# print test set accuracy  
sum(predict\_test\_2 == data\_test\_1[,11])/nrow(data\_test\_1)

## [1] 0.9562044