Question 2.1

Having worked in the plant maintenance industry, classification models can be quite a useful and effective way of triggering maintenance inspections. The most common, and highly expensive, problem is deciding on an optimal maintenance policy, based on the current condition of the equipment. A classification model could predict whether a component in the plant is 'healthy' or not, based on the parameters that are being monitored. Several predictors can be used, depending on the type of component. For example, rotating machinery are equipped with vibration sensors. The predictors that can be used from a vibration signal include the minimum, maximum value of the signal, skewness (and several other statistical moments), along with several key frequencies obtained by applying the Fourier transformation on the vibration signal.

Question 2.2 (1)

At first, the dataset is checked for imbalance to ensure that accuracy is indeed a valid metric. As shown in Fig. 1 the dataset is fairly balanced (45% of the observations have a target variable of zero, and the rest 55% present a value of one).

It is known that there are no missing values in the dataset (those have been already imputed for us), so there is no need to perform any checks.

The first step to make an SVM classifier involves the use of a linear kernel (vanilladot in the kernlab package).

The only parameter on a linear SVM is the soft margin constant λ – or C in the kernlab package. To identify a good estimate for λ several values were tried, based on a logarithmic scale, so as to allow us to identify a promising order of magnitude at first, and narrow down on a 'good' value for λ later on. The results are shown in Table 1.

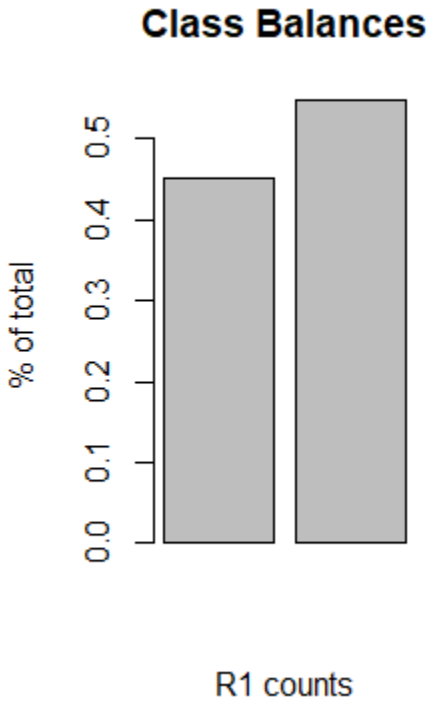

Figure 1: Target variable balance of the dataset

Table 1: SVM accuracy vs. Soft Margin Constant

Soft margin constant	Classification Accuracy
1e-3	0.8379
1e-2	0.8639
0.1	0.8639
1	0.8639
10	0.8639
25	0.8639
50	0.8639
1e2	0.8639
5e2	0.8639

It can be seen that for virtually any value of λ, the classification results remain constant to about 86.4%. For this classification accuracy (using a λ = 1), the following coefficients arise:

Table 2: SVM (linear) coefficients of each attribute

Attribute	N/A (constant term)	A1	A2	A3	A8
Coefficient	8.148382×10^{-2}	-1.1027×10^{-3}	-8.98×10^{-4}	-1.607×10^{-3}	2.904×10^{-3}
Attribute	A10	A11	A12	A14	A15
Coefficient	-2.985×10^{-3}	-2.035179×10^{-4}	-5.5048×10^{-4}	-1.2519×10^{-3}	0.10644

Using Table 2, the following hyperplane divides the space into two (c_i is the coefficient corresponding to the A_i attribute):

$$\left\{x : f(x) = c_0 + \sum_i c_i A_i = 0\right\}$$

The sign of the discriminant function f(x) denotes the side of the hyperplane the point (observation) is on.

Question 2.2 (2)

Now that a linear model has been established as a baseline, a non-linear kernel can be used to increase classification accuracy. As stated in (Asa Ben - Hur), between a Gaussian and a polynomial kernel, the Gaussian is better both in accuracy and convergence time. Following this advice, a Gaussian kernel was tested (rbfdot in kernlab), using a 2-dimensional grid-search to estimate a good combination of the soft margin constant and the classifier's hyperparameter (σ). The results are shown in Fig 2.

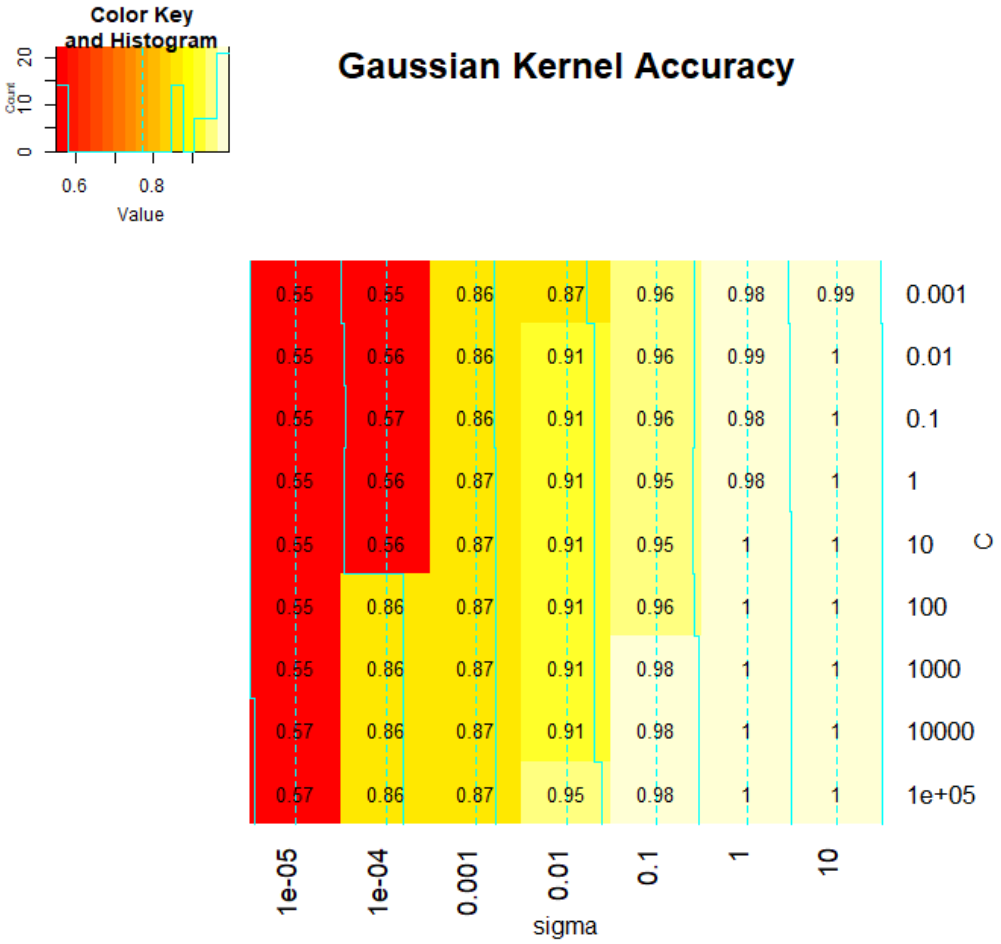

Figure 2: SVM w/ Gaussian Kernel - accuracy, C, σ

It can be seen that several combinations of the two parameters result in a perfect classification accuracy. Without a testing dataset however, it is not possible to get a true estimate of the classifier's accuracy. It is indeed possible, that high accuracies are the result of sever overfitting. The R script of this analysis can be found in Appendix A.

Question 2.2 (3)

Using the kknn package, a K Nearest Neighbors (KNN) classifier was trained on the entire dataset, with a maximum value of k equal to 80. The mean squared error of the classifier, against different values of k, can be seen in Fig. 3

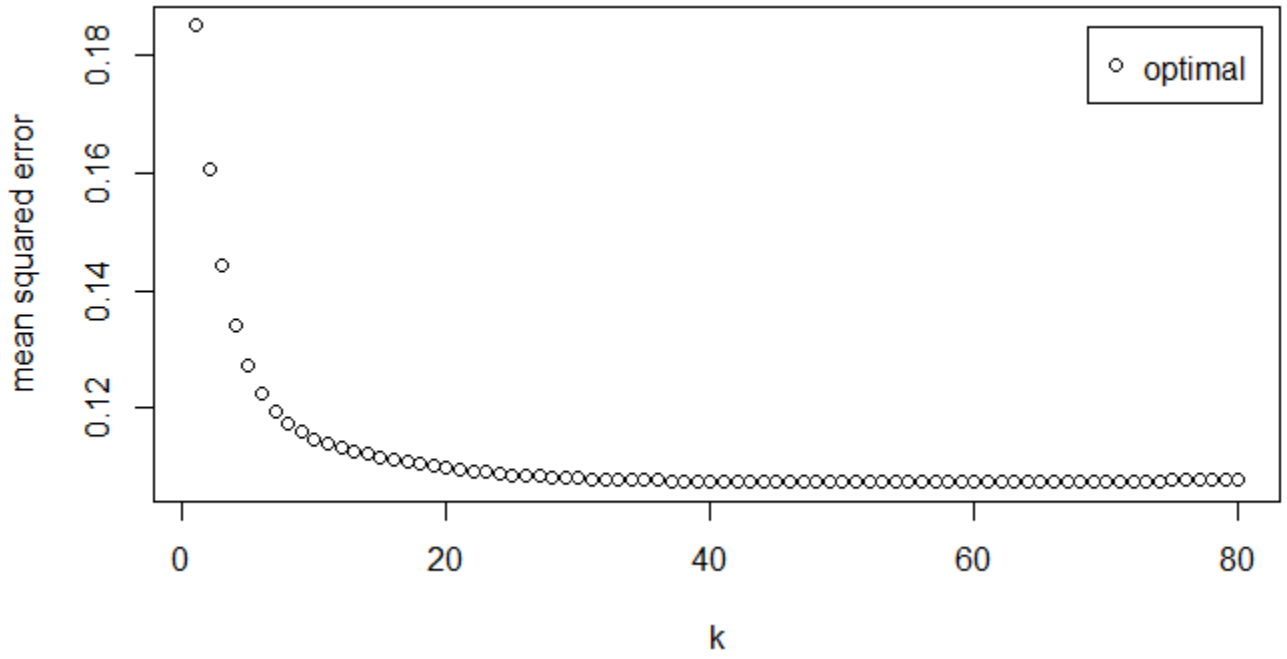

Figure 3: KNN classifier, mean squared error for different values of the parameter k

The optimal value of k was found to be equal to 58, with an accuracy of 87.76%, although it is not visible in the Fig. 3. It is interesting to note that this is slightly higher than the accuracy achieved with the linear SVM model. The corresponding R script can be found in Appendix B.

Question 3.1 (a)

Although the kknn package provides an off-the-shelf command to perform cross validation using a KNN classifier (cv.kknn(formula, data, kcv ...)) as described in https://www.rdocumentation.org/packages/class/versions/7.3-14/topics/knn.cv , a small script was written to replicate its functionality, shown in Appendix C. A 10-fold cross validation was performed (without withholding a small portion of the dataset as the testing set). The accuracy of the classifiers – for different values of k – was estimated as the average accuracy across all 10 validation sets. This average accuracy as a function of k can be seen in Fig. 4.

Two important conclusions can be drawn from Fig. 4:

- The average accuracy on the validation sets has dropped, compared with the model of question 2.2(3) by 3.5% - the validation accuracy is 84.24% for this model
- The optimum value of k differs a lot compared to the previous model. The value of k that provides the best results in this case is 14, far from the value of 80, as discussed in question 2.2(3).

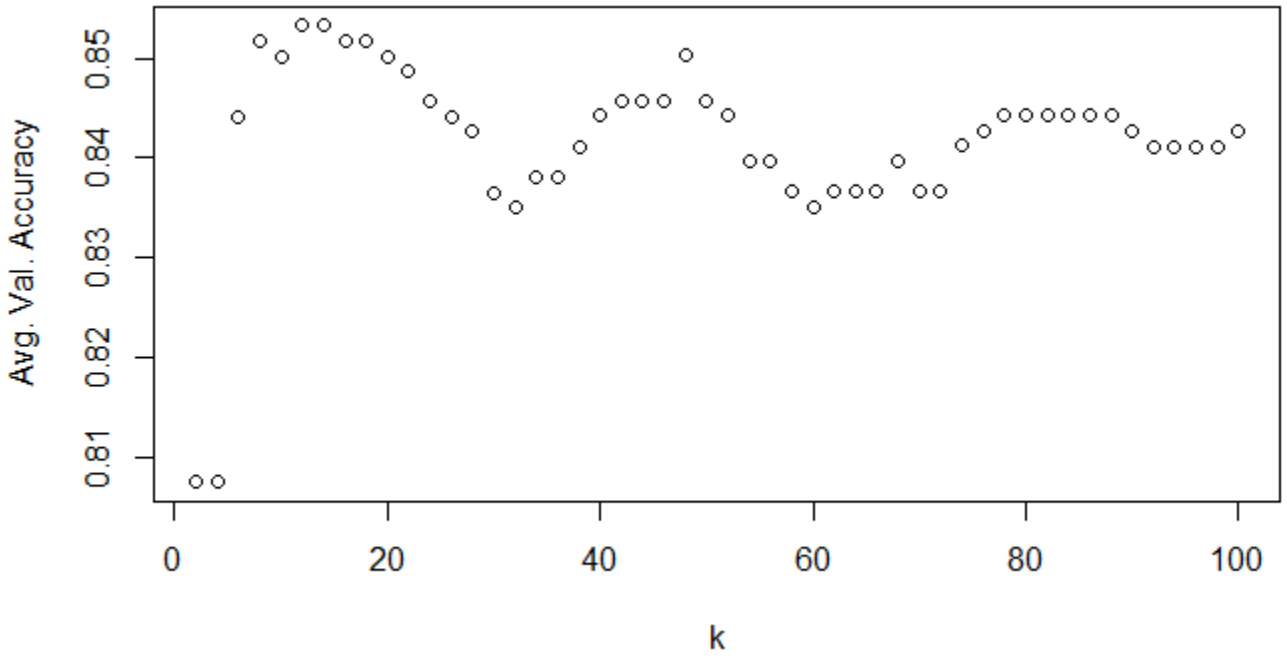

Figure 4: Average validation-set-accuracy for different values of k

Question 3.1 (b)

Finally, a KNN classifier was used again for the train / test / validation split, for comparison purposes with the models created using the previous approaches. From the original dataset, the percentages shown in Table 3 were used.

Table 3: Train, Test, Validation split percentages

Subset	Percentage (%)
Training	50
Test	25
Validation	25

All the models created (for different values of k), were trained using the training set, their accuracy on the validation set was calculated, so as to choose the best model, whose performance, finally, was estimated using the test set. The accuracy of the different classifiers on the validation set can be seen in Fig. 5. In this case, the optimal value for k is 10, not far from – but not equal to either – the optimal value found from the previous approach. The optimal classifier's accuracy in this case was found to be equal to 82.65% *on the test set*, which

is 1.59% lower compared to the accuracy estimated in the 10-fold cross validation. The R script developed can be found in Appendix D.

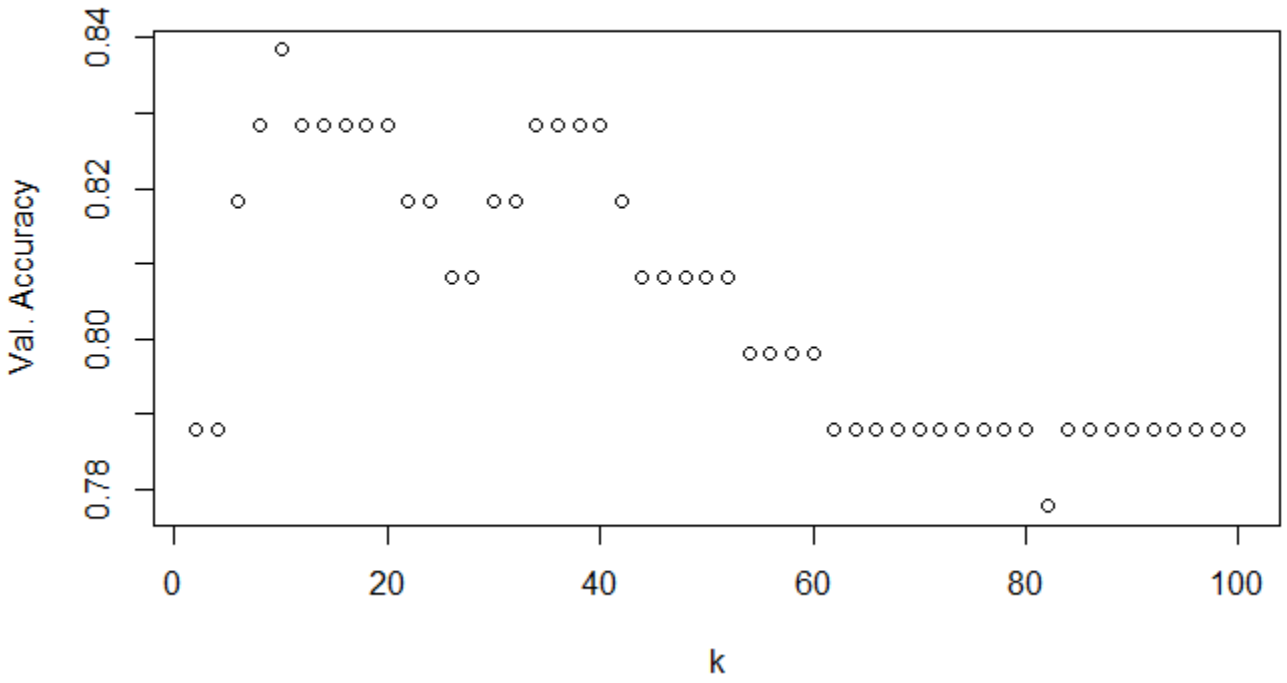

Figure 5: Validation set accuracy for different values of k

Conclusions

The most crucial conclusion that can be drawn from this analysis is the importance of performing a correct evaluation of the model. Using different (or worse, wrong) approaches can lead to severe misinterpretation to the evaluation metric of the model, its performance, and its optimal structure.

Appendix A: R script for Question 2.2 (1), (2)

```
# GTx: ISYE6501x
# SU18: Introduction to Analytics Modeling
# Homework - Week 1

#   Question 2.2
#------------------

# Set the working directory
setwd("C:/Users/Miltos/Desktop/Homework 1/")

# If the kernlab package is not installed, go on and install it
if (!"kernlab" %in% rownames(installed.packages()))
{
  install.packages("kernlab")
}
library(kernlab)

# Read the .txt with headers
txt <- read.table("2.2credit_card_data-headersSummer2018.txt", header = TRUE)
txt <- as.matrix(txt)

# Split target / attributes
x <- txt[, 1:10]
y <- txt[, 11]

# Check for imbalance
barplot(c(sum(y == 1) / length(y), sum(y == 0) / length(y)), xlab = "R1 counts", ylab = "% of total", main="Class Balances")

# Initialize an empty matrix to store results on
```

```
results.matrix <- NULL

# Iterate over different lambdas
for (C in c(1e-3, 1e-2, 1e-1, 1, 10, 25, 50, 1e2, 5e2))
{
  # Train the model
  model <- ksvm(x, y, type = "C-svc", kernel = "vanilladot", C = C, scaled = TRUE)
  # (not so efficient to rescale the data everytime...)

  # append results to the results.matrix
  results.matrix <- rbind(results.matrix, c(C, 1 - model@error))
}

# Train the model using C = 1
model <- ksvm(x, y, type = "C-svc", kernel = "vanilladot", C = 1, scaled = TRUE)

# calculate a1 - am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a
# calculate a0
a0 <- -model@b
a0

# Initialize an empty matrix to store results on
results.matrix_gaussian <- NULL

# Make a list with parameter values
Cs <- c(1e-3, 1e-2, 1e-1, 1, 10, 1e2, 1e3, 1e4, 1e5)
sigmas <- c(1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10)

# Iterate over different lambdas
for (C in Cs)
{
```

```
  # Iterate over different sigmas
  for (sigma in sigmas)
  {
    # Train the model
    model <- ksvm(x, y, type = "C-svc", kernel = "rbfdot", C = C, sigma = sigma, scaled = TRUE)

    # append results to the results.matrix
    results.matrix_gaussian <- rbind(results.matrix_gaussian, c(C, sigma, 1 - model@error))
  }
}

# Reshape matrix to plot a heatmap
q <- results.matrix_gaussian[,3]
dim(q) <- c(9,7)

# plot heatmap
heatmap.2(q, cellnote=round(q, 2), Colv = NA, Rowv = NA, labRow = Cs,
          labCol = sigmas, main = "Gaussian Kernel Accuracy", ylab = "C",
          xlab = "sigma", notecol = "black")
```

Appendix B: R script for Question 2.2 (3)

```
# GTx: ISYE6501x
# SU18: Introduction to Analytics Modeling
# Homework - Week 1

#   Question 2.3
#------------------

# Set the working directory
setwd("C:/Users/Miltos/Desktop/Homework 1/")

# If the kknn package is not installed, go on and install it
if (!"kknn" %in% rownames(installed.packages()))
{
  install.packages("kknn")
}
library(kknn)

# Read the .txt with headers
txt <- read.table("2.2credit_card_data-headersSummer2018.txt", header = TRUE)
txt <- as.matrix(txt)

# Split target / attributes
x <- txt[, 1:10]
y <- txt[, 11]

# Train the model via leave-one-out
model <- train.kknn(y~., data.frame(x), kmax = 80) # scale = TRUE by default
model
plot(model)
```

```
# Get the results
pred <- predict(model, data.frame(x))
results.df <- data.frame(y, round(pred))
colnames(results.df) <- c("actual", "predicted")
accuracy <- sum(results.df$actual == results.df$predicted) / nrow(results.df)
accuracy
```

Appendix C: R script for Question 3.1 (a)

```
# GTx: ISYE6501x
# SU18: Introduction to Analytics Modeling
# Homework - Week 1

#   Question 3.1
#------------------

# Set the working directory
setwd("C:/Users/Miltos/Desktop/Homework 1/")

# If the kknn package is not installed, go on and install it
if (!"kknn" %in% rownames(installed.packages()))
{
  install.packages("kknn")
}
library(kknn)

# Read the .txt with headers
txt <- read.table("2.2credit_card_data-headersSummer2018.txt", header = TRUE)
txt <- as.matrix(txt)

# The following can be used in the KKNN package: cv.kknn(formula, data, kcv = 10, ...)
# It would be interesting to replicate it...

#shuffle the data before splitting (to ensure diversity)
txt<-txt[sample(nrow(txt)), ]

#Create 10 equally size folds
folds <- cut(seq(1, nrow(txt)), breaks = 10, labels = FALSE)
```

```
# Try different values of k to find the best model
kappas <- seq(2, 100, by = 2)
# Initialize an empty matrix to store results on
results.matrix <- NULL

for (k in kappas)
{
 # Create an empty variable which will contain the individual accuracy of each fold
 accuracy <- NULL

 #Perform 10 fold cross validation, without using a test set (not explicitly stated in the HW to use a test set here)
 for(i in 1:10)
 {
  #Split the data by fold using the which() function
  testIndexes <- which(folds == i,arr.ind = TRUE)
  testData <- txt[testIndexes, ]
  trainData <- txt[-testIndexes,]
  x_train <- data.frame(trainData[, 1:10])
  y_train <- trainData[, 11]
  x_test <- data.frame(testData[, 1:10]) # test referes to the validation set here
  y_test <- testData[, 11]

  # Train the knn model on the train data
  model <- train.kknn(y_train ~., x_train, ks = k) # scale = TRUE by default

  # And computre its accuracy on the validation data
  pred <- predict(model, x_test)
  accuracy <- rbind(accuracy, sum(y_test == round(pred)) / nrow(x_test))
 }
```

```
  accuracy <- mean(accuracy) # or median?

  # append results to the results.matrix

  results.matrix <- rbind(results.matrix, c(k, accuracy))

}


# Check results

plot(results.matrix, xlab = "k", ylab = "Avg. Val. Accuracy")


# Same exact procedure applies to the Support Vector Machine
```

Appendix D: R script for Question 3.1 (b)

```
# GTx: ISYE6501x
# SU18: Introduction to Analytics Modeling
# Homework - Week 1

#   Question 3.1b
#------------------

# Set the working directory
setwd("C:/Users/Miltos/Desktop/Homework 1/")

# If the kknn package is not installed, go on and install it
if (!"kknn" %in% rownames(installed.packages()))
{
  install.packages("kknn")
}
library(kknn)

# Read the .txt with headers
txt <- read.table("2.2credit_card_data-headersSummer2018.txt", header = TRUE)
txt <- as.matrix(txt)

# The following can be used in the KKNN package: cv.kknn(formula, data, kcv = 10, ...)
# It would be interesting to replicate it...

#shuffle the data before splitting (to ensure diversity)
txt<-txt[sample(nrow(txt)), ]

#Create 10 equally size folds
folds <- cut(seq(1, nrow(txt)), breaks = 10, labels = FALSE)
```

```
# Try different values of k to find the best model
kappas <- seq(2, 100, by = 2)

# Initialize an empty matrix to store results on
results.matrix <- NULL

# Split data into training, test, and alidation sets
# Selecting 70% of data as sample from total 'n' rows of the data randomly
sample <- sample.int(n = nrow(txt), size = floor(.7*nrow(txt)), replace = F)
train <- txt[sample, ]
rest  <- txt[-sample, ]

# Further split the test set in half, to get the validation set
sample <- sample.int(n = nrow(rest), size = floor(.5*nrow(rest)), replace = F)
test  <- rest[sample, ]
val  <- rest[-sample, ]

x_train <- data.frame(train[, 1:10])
y_train <- train[, 11]
x_test <- data.frame(test[, 1:10]) # test referes to the validation set here
y_test <- test[, 11]
x_val <- data.frame(val[, 1:10])
y_val <- val[, 11]

for (k in kappas)
{
  # Create an empty variable which will contain the individual accuracy of each fold
  accuracy <- NULL
```

```
  # Train the model on the train set
  model <- train.kknn(y_train ~., x_train, ks = k) # scale = TRUE by default

  # Get the model's accuracy on the validation set
  pred <- predict(model, x_val)
  accuracy <- rbind(accuracy, sum(y_val == round(pred)) / nrow(x_val))
  # append results to the results.matrix
  results.matrix <- rbind(results.matrix, c(k, accuracy))
}

# Check results
plot(results.matrix, xlab = "k", ylab = "Val. Accuracy")

# Sort the results according to accuracy
results.matrix <- results.matrix[sort.list(results.matrix[,2], decreasing = TRUE), ]

# Train the best model on the train set
model <- train.kknn(y_train ~., x_train, ks = results.matrix[1, 1]) # scale = TRUE by default
# No need to check its accuracy again on the validation set...

# Use the test data to estimate the performance of the best model
pred <- predict(model, x_test)
accuracy <- sum(y_test == round(pred)) / nrow(x_test)
accuracy # Wow... VERY different from the validation accuracy

# Same exact procedure applies to the Support Vector Machine
```

Bibliography

Asa Ben - Hur, J. W. (n.d.). *A user's guide to Support Vector Machines.*