Case 1

Damage detection of the Valdemar platform model

Kasper Juul Jensen, Lars Roed Ingerslev, Maya Katrin Gussmann

March 21, 2016

Please upload a file `yhat.txt` containing predictions $\{0, 1, 2, 3\}$ as one long vector with newline separating the observations. The corresponding order is that given in the Cas1_tst (Xt), and a small report with a brief introduction/abstract, pre-processing, modeling and model assessment, plus your evaluation of the actual dimensions needed to describe the data/problem. (Max 5 A4 pages of text)

Please note that the deadline is mid-day.

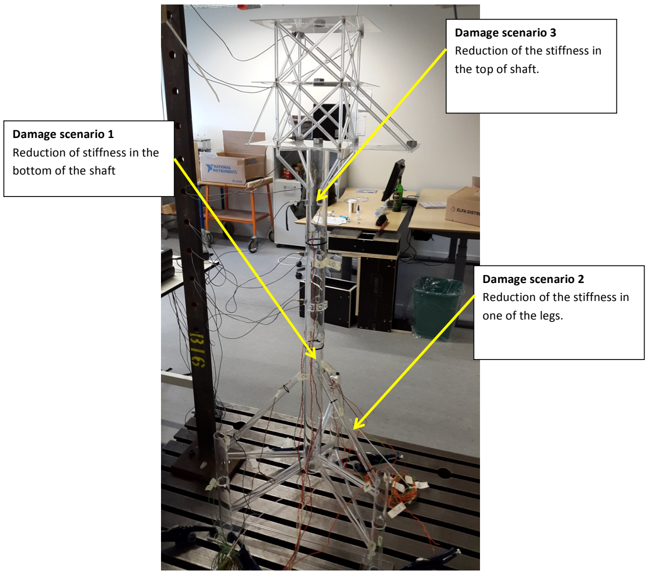
1. Introduction

There are three sensors on an offshore platform, that record data for detecting damage to the platform. Damage can occur at three different sites (see table 1.1) and for three different intensities (5%, 10%, 15%). The recorded data is given in the form of Frequency Response Functions (FRFs).

Class	Description
0	undamaged
1	damage in the bottom of the shaft
2	damage in one of the legs
3	damage in the top of the shaft

Table 1.1.: Damage classes.

In this case, 4092 samples of the three FRFs for 190 cases are given, together with their respective damage classes. The goal is to form a model to predict damage and damage class.



2. Pre-processing

2.1. Cross validation

2.2. PLS

3. Modeling

In the following sections we will look into different models. The final damage class will then be chosen based on a majority vote of the models.

3.1. KNN

```
KNNfn <- function(xTrain, yTrain, xTest, yTest, nNeighbours)
{
    errKNNcv <- numeric(nNeighbours)
    # Use leave one out CV of train set to select number of neighbours
    for (K in seq_len(nNeighbours))
    {
      KNNpred <- knn.cv(train = xTrain, cl = yTrain, k = K)
      errKNNcv[K] <- sum(as.integer(KNNpred != yTrain))
    }
    # The simplest model is the one with the most neighbours
    K <- max(which(errKNNcv == min(errKNNcv)))

    KNNpred <- knn(train = xTrain, test = xTest, cl = yTrain, k = K)
    errKNN <- sum(as.integer(KNNpred != yTest))
    names(errKNN) <- "KNN"

    print(K)

    list(errorRate = errKNN, predictions = KNNpred)
}
```

3.2. Logistic regression

```
logisticRegressionFn <- function(xTrain, yTrain, xTest, yTest){
    maxVal <- max(abs(xTrain))
    dat <- data.frame(y = yTrain, unclass(xTrain/maxVal))
    model <- multinom(y ~ ., data = dat)
    logitPred <- predict(model, newdata= data.frame(xTest/maxVal), type='probs')
```

```
    logitPred <- factor(apply(logitPred, 1, which.max) - 1, levels = c(0:3))
    errLogit <- sum(logitPred != yTest) / length(yTest)
    names(errLogit) <- "LogisticRegression"

    list(errorRate = errLogit, predictions = logitPred)
}
```

3.3. SVM

We used a support vector machine [5] with a linear kernel to build a model from the training data. Then, classes for the test data were predicted and the error rate calculated. The mean error rate for this model was 0.00, see table 3.1. For the given training and test data and pre-processing, the SVM is performing very well.

```
svmFn <- function(xTrain, yTrain, xTest, yTest){
    dat <- data.frame(y = yTrain, unclass(xTrain))
    colnames(dat) <- c('y', colnames(xTrain))
    model <- svm(formula = y ~ ., data = dat, kernel = "linear")

    svmPred <- predict(model, xTest)
    errSVM <- sum(svmPred != yTest) / length(yTest)
    names(errSVM) <- "SVM"

    list(errorRate = errSVM, predictions = svmPred)
}
```

3.4. CART

For the classification and regression trees we used the `rpart` function to fit a classification tree and `prune` to prune it, if necessary, [7]. The mean error rate for CART was 0.01, which is slightly worse than for some of the other models, see table 3.1.

```
cartFn <- function(xTrain, yTrain, xTest, yTest)
{
    dat <- data.frame(y = yTrain, unclass(xTrain))
    model <- rpart(y ~ ., data = dat, method = "class")
    pfit<- prune(model, cp= model$cptable[which.min(model$cptable[,"xerror"]),"CP"])

    cartPred <- predict(pfit, newdata = data.frame(xTest), type = c("class"))
    errCart <- sum(cartPred != yTest) / length(yTest)
```

```
    names(errCart) <- "CART"

    list(errorRate = errCart, predictions = cartPred)
}
```

3.5. Boosting

3.6. Random Forest

3.7. Majority Vote

For the final class selection, we compared all model predictions and chose the class that was predicted by most models. In case of a tie, the first used method (see table 3.1) would be chosen. For the R-code of the majority vote, please see appendix B.

	x
KNN	0.00
LogisticRegression	0.01
SVM	0.00
CART	0.01
majorityVote	0.00

Table 3.1.: Mean error rates for different models over 10-fold cross validation.

4. Dimensions

Appendices

A. Pre-processing

```
library(reshape2)
library(data.table)
library(cvTools)
library(class)
library(nnet)
library(e1071)
library(pls)
library(rpart)


load("Case1.RData")

setup <- data.table(
    sensor = rep(rep(c("S1", "S2", "S3"), each = 4092), 2),
    freq = rep(1:4092, 6),
    part = rep(c("real", "imaginary"), each = 4092*3)
)
setup[, idx:=seq_len(nrow(setup))]


set.seed(1)
nObs <- nrow(Xtr)
nFolds <- 10
folds <- cvFolds(nObs, nFolds)

subsetData <- function(i, x, y, folds)
{
    ind <- folds$which == i
    xTrain <- x[!ind, ]
    yTrain <- y[!ind]
    xTest <- x[ind, ]
    yTest <- y[ind]
    list(xTrain = xTrain,
         yTrain = yTrain,
         xTest = xTest,
         yTest = yTest)
```

```r
}


rotateData <- function(xTrain, yTrain, xTest, yTest)
{
    # Center, but do no scale x
    xTrain <- scale(xTrain, scale = FALSE)
    xTest <- scale(xTest, center = attr(xTrain, "scaled:center"), scale = FALSE)

    # Reshape y to be a 4 column matrix and do PLS
    y <- as.data.frame(lapply(levels(yTrain), function(x) as.integer(yTrain == x)))
    pls <- plsr(as.matrix(y) ~ xTrain)

    # shuffle xTrain, do PLS again and
    # find the maximum explained variance by the shuffled data.
    xShuf <- apply(xTrain, 2, function(x)x[sample(length(x))])
    plsShuf <- plsr(as.matrix(y) ~ xShuf)
    cutOff <- max(explvar(plsShuf))

    nComp <- max(which(explvar(pls) > cutOff))
    pls2 <- plsr(as.matrix(y) ~ xTrain, ncomp = nComp)
    # Select only components with a higher explained variance
    xTrainRot <- pls2$scores
    xTestRot <- predict(pls2, xTest, type = "scores")

    list(xTrain = xTrainRot,
         yTrain = yTrain,
         xTest = xTestRot,
         yTest = yTest)
}

makeSkeleton <- function(i, x, y, folds)
{
    data <- subsetData(i, x, y, folds)
    list(
        i = i,
        data = with(data, rotateData(xTrain, yTrain, xTest, yTest)),
        predictions = data.frame(matrix(nrow = length(data$yTest), ncol = 0)),
        errorRate = numeric()
    )
}

dataList <- lapply(seq_len(nFolds), makeSkeleton, Xtr, class_tr, folds)
```

```
save(dataList, folds, file = "preprocessed.RData")
```

B. Majority vote

```
doTest <- function(dataObject, f, ...)
{
    name = as.character(substitute(f))
    res <- with(dataObject$data, f(xTrain, yTrain, xTest, yTest, ...))
    dataObject$predictions[[name]] <- res$predictions
    dataObject$errorRate <- c(dataObject$errorRate, res$errorRate)
    dataObject
}

majorityVote <- function(dataObject)
{
    dataObject$predictions$majorityVote <- apply(
      dataObject$predictions, 1, function(x)names(which.max(table(x))))
    dataObject$errorRate["majorityVote"] <- sum(
      dataObject$predictions$majorityVote != dataObject$data$yTest)
    dataObject
}

getMeanErrorRates <- function(dataList)
{
    dataList <- lapply(dataList, majorityVote)

    err <- do.call('rbind', lapply(dataList, '[[', 'errorRate'))
    err <- colMeans(err)
    err
}

dataList <- lapply(dataList, doTest, KNNfn, nNeighbours = 20)

## [1] 13
## [1] 14
## [1] 13
## [1] 13
## [1] 13
## [1] 14
## [1] 13
```

```
## [1] 13
## [1] 13
## [1] 15
```

```r
dataList <- lapply(dataList, doTest, logisticRegressionFn)
```

```
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.706105
## iter  20 value 0.009737
## iter  30 value 0.002467
## final  value 0.000090
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.573304
## iter  20 value 0.042724
## iter  30 value 0.000305
## final  value 0.000078
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.410803
## iter  20 value 0.004693
## iter  30 value 0.000727
## final  value 0.000084
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.911910
## iter  20 value 0.030692
## iter  30 value 0.000778
## final  value 0.000075
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.716921
## iter  20 value 0.057360
## iter  30 value 0.000271
## final  value 0.000070
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.728738
```

```
## iter  20 value 0.037782
## iter  30 value 0.000495
## final  value 0.000048
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.823359
## iter  20 value 0.053523
## iter  30 value 0.000331
## final  value 0.000085
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.648413
## iter  20 value 0.067015
## iter  30 value 0.000232
## final  value 0.000059
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.723918
## iter  20 value 0.022144
## final  value 0.000079
## converged
## # weights:  24 (15 variable)
## initial  value 237.056336
## iter  10 value 1.658182
## iter  20 value 0.020666
## final  value 0.000072
## converged
```

```
dataList <- lapply(dataList, doTest, svmFn)
dataList <- lapply(dataList, doTest, cartFn)
```

```
getMeanErrorRates(dataList)
```

```
##                  KNN LogisticRegression                SVM
##          0.000000000        0.005263158        0.000000000
##                 CART                KNN LogisticRegression
##          0.005263158        0.000000000        0.005263158
##                  SVM               CART       majorityVote
##          0.000000000        0.005263158        0.000000000
```

R and R packages

[1] Andreas Alfons. *cvTools: Cross-validation tools for regression models*, 2012. R package version 0.3.2.

[2] David B. Dahl. *xtable: Export tables to LaTeX or HTML*, 2014. R package version 1.7-4.

[3] M Dowle, A Srinivasan, T Short, S Lianoglou with contributions from R Saporta, and E Antonyan. *data.table: Extension of Data.frame*, 2015. R package version 1.9.6.

[4] Bjørn-Helge Mevik, Ron Wehrens, and Kristian Hovde Liland. *pls: Partial Least Squares and Principal Component Regression*, 2015. R package version 2.5-0.

[5] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2015. R package version 1.6-7.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[7] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2015. R package version 4.1-10.

[8] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

[9] Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007.

[10] Yihui Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2015. R package version 1.11.