Case 1

Damage detection of the Valdemar platform model

Kasper Juul Jensen, Lars Roed Ingerslev, Maya Katrin Gussmann

March 27, 2016

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). Due to the fairly large amount of attributes the first and likely most important step in this case is dimension reduction. Section 2 describes the pre-processing steps we went through to reach an acceptable reduction in dimensions. Subsequently, as explained in section 3 and 4, we subjected the data to all appropriate models taught in the course and we selected the final damage class by the means of majority vote of the models.

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

Overfitting was controlled using a 10-fold cross validation, splitting the data into xTest, xTrain, yTest and yTrain. The folds were generated using `cvTools`[2]. In each fold xTrain was centered (but not scaled) and a PLS model was fitted to the training data (xTrain and yTrain), using `pls`[6]. The rows of each column in xTrain were shuffled and a second PLS model was fitted using the shuffled (randomized) xTrain. In the first PLS the components with an explained variance greater then the largest explained variance of the shuffled PLS were kept, in all folds the number of components kept were 4. Finally xTest was centered using the centers calculated on xTrain, and xTrain and xTest were rotated using the 4 components selected.

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

The first model we introduce is KNN. When applying KNN, finding an appropriate number of neighbors (k) is crucial. When k is too low the model is disposed to over fitting, hence we seek to find the simplest model which is the one with the highest k and lowest error rate. The appropriate number of neighbors is found with the means of leave one out cross validation on the training data. In this case the appropriate k varies from 13 to 16, but was 13 in 7 out of 10 folds. KNN performed very well with a mean error rate of 0.00, see table 3.1. In the final classification the k chosen was 15.

```r
KNN <- function(xTrain, yTrain, xTest, nNeighbours)
{
  require(class)
  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)
  print(K)

  KNNpred
}
```

3.2. Logistic regression

The logistic regression was performed using the `multinom` function [10]. To speed up the computation the variables were scaled from -1 to 1. The model is capable of simultaneously classifying all four cases. The error rate of the logistic regression was 0.005, see table 3.1, corresponding to a single misclassified observation.

```
logisticRegression <- function(xTrain, yTrain, xTest)
{
  require(nnet)
  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))

  logitPred
}
```

3.3. SVM

We used a support vector machine [7] 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.

```
SVM <- function(xTrain, yTrain, xTest)
{
  require(e1071)
  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)

  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, [9]. The mean error rate for CART was 0.005, which is slightly worse than for some of the other models, see table 3.1.

```
cart <- function(xTrain, yTrain, xTest)
{
  require(rpart)
  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"))

  cartPred
}
```

3.5. Boosting

Performing Classification tree with Boosting is an ensample method founded on bootstrapping (sampling with replacement). Boosting threes are grown in an adaptive manner where weights are put on misclassifications in order to reduce bias. Hence, output is a weighted average of all the trees which have been grown. Mean error rate for Booting in our case is 0.005, see table 3.1.

```
boost <- function(xTrain, yTrain, xTest)
{
  require(adabag)
  dat <- data.frame(y = yTrain, unclass(xTrain))
  model <- boosting(y ~ ., data = dat)

  boostPred <- predict(model, newdata = data.frame(xTest))$class

  boostPred
}
```

3.6. Random Forest

Finally random forrest classification was performed. 500 trees were grown using 1/3 of the variables and about 2/3 of the samples for each tree. The mean error rate for the random forest was 0.005.

```
rnForest <- function(xTrain, yTrain, xTest)
{
  require(randomForest)
  dat <- data.frame(y = yTrain, unclass(xTrain))
  model <- randomForest(y ~ ., data = dat)

  randomForestPred <- predict(model, newdata = data.frame(xTest))

  randomForestPred
}
```

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. Surprisingly the majority vote performed worse than KNN and SVM, since a single observation was misclassified as type 0 in 4 of the 7 methods tested.

	error_test
KNN	0.00
logisticRegression	0.01
SVM	0.00
cart	0.01
boost	0.01
rnForest	0.01
majorityVote	0.01

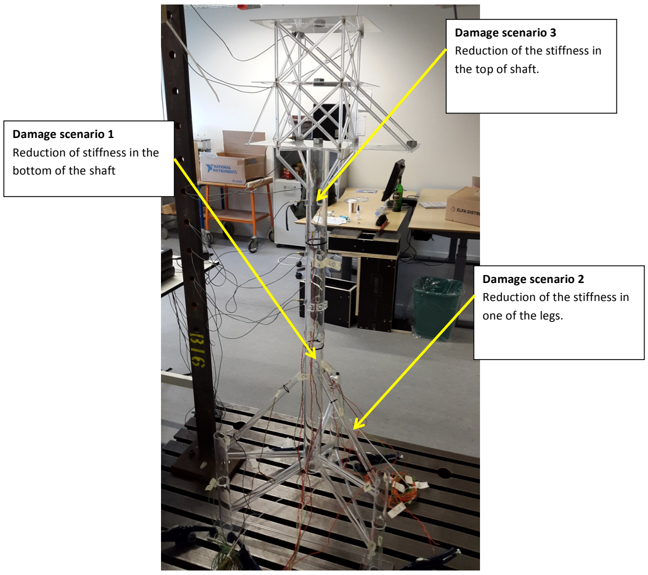
Table 3.1.: Mean error rates for different models over 10-fold cross validation.

	stderror_test
KNN	0.0000000000
logisticRegression	0.0000875977
SVM	0.0000000000
cart	0.0000875977
boost	0.0000875977
rnForest	0.0000875977
majorityVote	0.0000875977

Table 3.2.: Standard errors for different models over 10-fold cross validation.

4. Dimensions

It is intuitively pleasing that 4 dimensions were chosen to represent the data, since there are 4 classes. Howerver, while the dimensions separate the data quite well, the individual classes does not have a dimension that caputres their features, with the exception of class 0 that is separated from the other classes by the first dimension. A visualization of the 4 dimension can be seen in figure 4.1



Damage scenario 3
Reduction of the stiffness in the top of shaft.

Damage scenario 1
Reduction of stiffness in the bottom of the shaft

Damage scenario 2
Reduction of the stiffness in one of the legs.

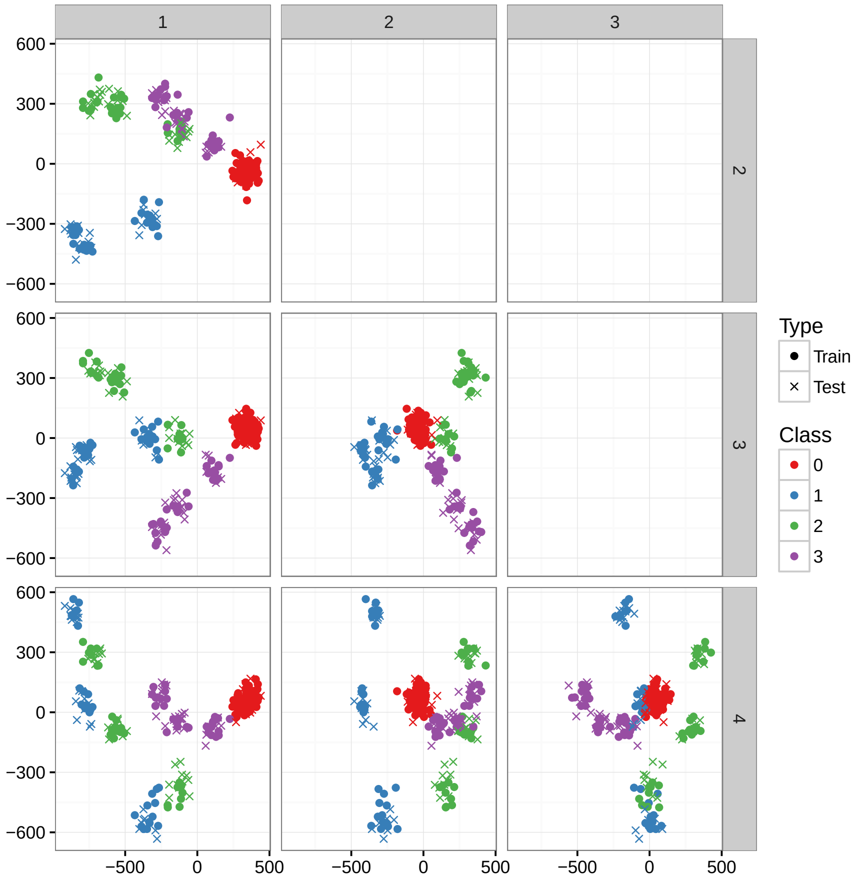


Figure 4.1.: Visualization of the four dimensions.

Appendices

A. Pre-processing Code

```r
library(reshape2)
library(data.table)
library(cvTools)
library(R.matlab)

load("Case1.RData")

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)
}


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

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

majorityVote <- function(dataObject)
{
  dataObject$predictions$majorityVote <- apply(
    dataObject$predictions, 1, function(x)names(which.max(table(x))))
  dataObject
}

getErrorRates <- function(dataObject)
{
  yTest <- dataObject$data$yTest
  nTest <- length(yTest)
  dataObject$errorRate <- apply(dataObject$predictions, 2, function(x){sum(x != yTest)/nTe
  dataObject
}

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

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

## [1] 13
## [1] 13
## [1] 13
## [1] 14
## [1] 13
## [1] 13
## [1] 13
## [1] 14
## [1] 13
## [1] 16
```

```
dataList <- lapply(dataList, doTest, logisticRegression)
```

```
## # 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, SVM)
dataList <- lapply(dataList, doTest, cart)
dataList <- lapply(dataList, doTest, boost)
dataList <- lapply(dataList, doTest, rnForest)
dataList <- lapply(dataList, majorityVote)

dataList <- lapply(dataList, getErrorRates)

getMeanErrorRates(dataList)
```

```
##                KNN logisticRegression                SVM
##        0.000000000        0.005263158        0.000000000
##               cart              boost           rnForest
##        0.005263158        0.005263158        0.005263158
##       majorityVote
##        0.005263158
```

R and R packages

[1] Esteban Alfaro, Matías Gámez, and Noelia García. adabag: An R package for classification with boosting and bagging. *Journal of Statistical Software*, 54(2):1–35, 2013.

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

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

[4] 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.

[5] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

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

[7] 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.

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

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

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

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

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