Computer lab 2 block 2 (732A99 Machine Learning)

Lennart Schilling (lensc874)

13 December 2018

Contents

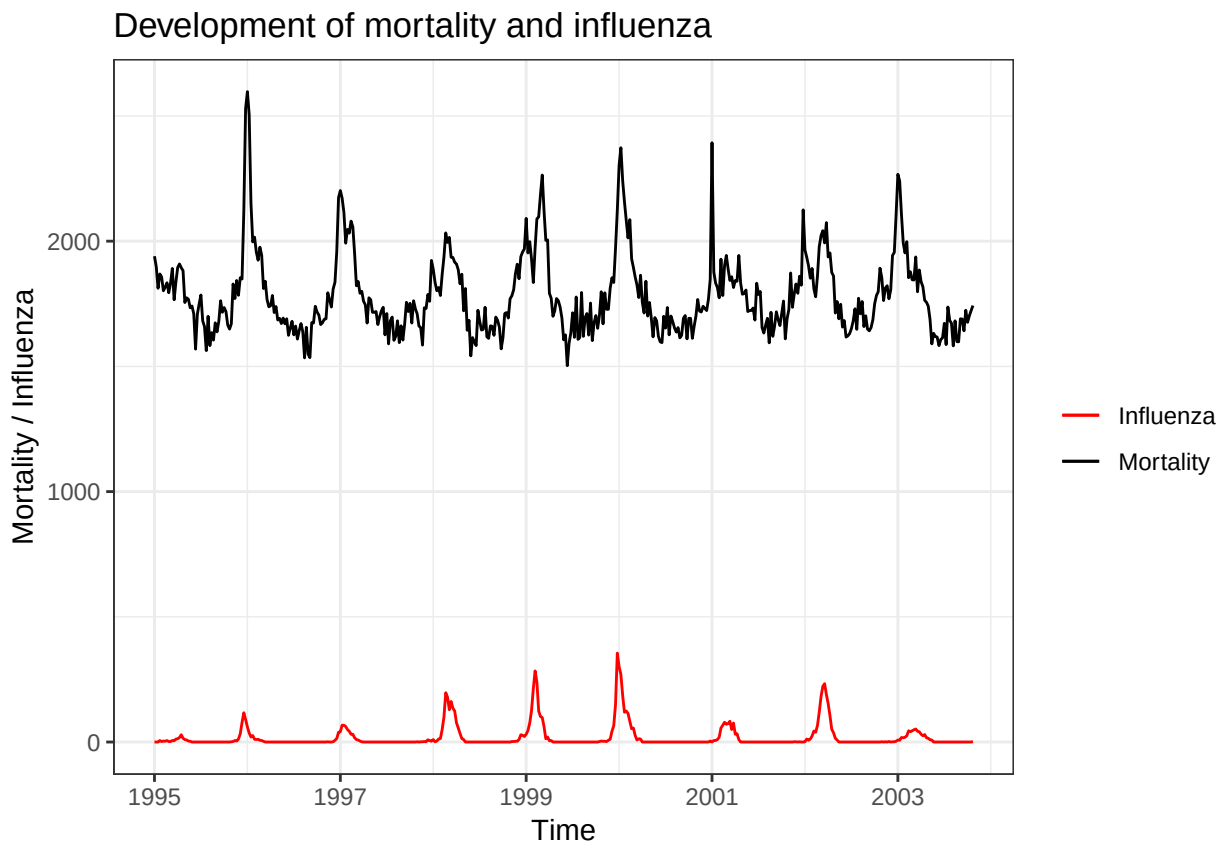
Assignment 1: Using GAM and GLM to examine the mortality rates

At first, the data from the Excel file *Influenza.xlsx* will be imported.

```
# importing data
library(readxl)
data = read_excel("Influenza.xlsx")
```

1.1

```
library(ggplot2)
# plotting development of mortality and influenza
ggplot(data = data,
       aes(x = Time)) +
  geom_line(aes(y = Mortality, colour = "Mortality")) +
  geom_line(aes(y = Influenza, colour = "Influenza")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "Development of mortality and influenza",
       y = "Mortality / Influenza",
       colour = "") +
  scale_colour_manual(values = c("red", "black")) +
  theme_bw()
```



In the plot it can be seen that for every time the amounts of influenza cases have a peak, the mortality also have a peak. This indicates that there is a certain similar pattern within the two distributions. Since influenza is a disease which could have an influence on the mortality, this similarity can be rated as logical.

1.2

```
library(mgcv)
# fitting gam model
gamModel = gam(formula = Mortality ~ Year + s(Week),
               data = data)
```

Using the default parameter settings within the *gam*-function implies that *Mortality* is normally distributed (*family=gaussian()*). Also, since *method = "GCV.Cp"*, this leads to the usage of GCV (*Generalized Cross Validation score*) related to the smoothing parameter estimation. The underlying probabilistic model can be written as:

$$Mortality = N(\mu, \sigma^2)$$

$$\hat{Mortality} = Intercept + \beta_1 Year + \epsilon + s(Week)$$

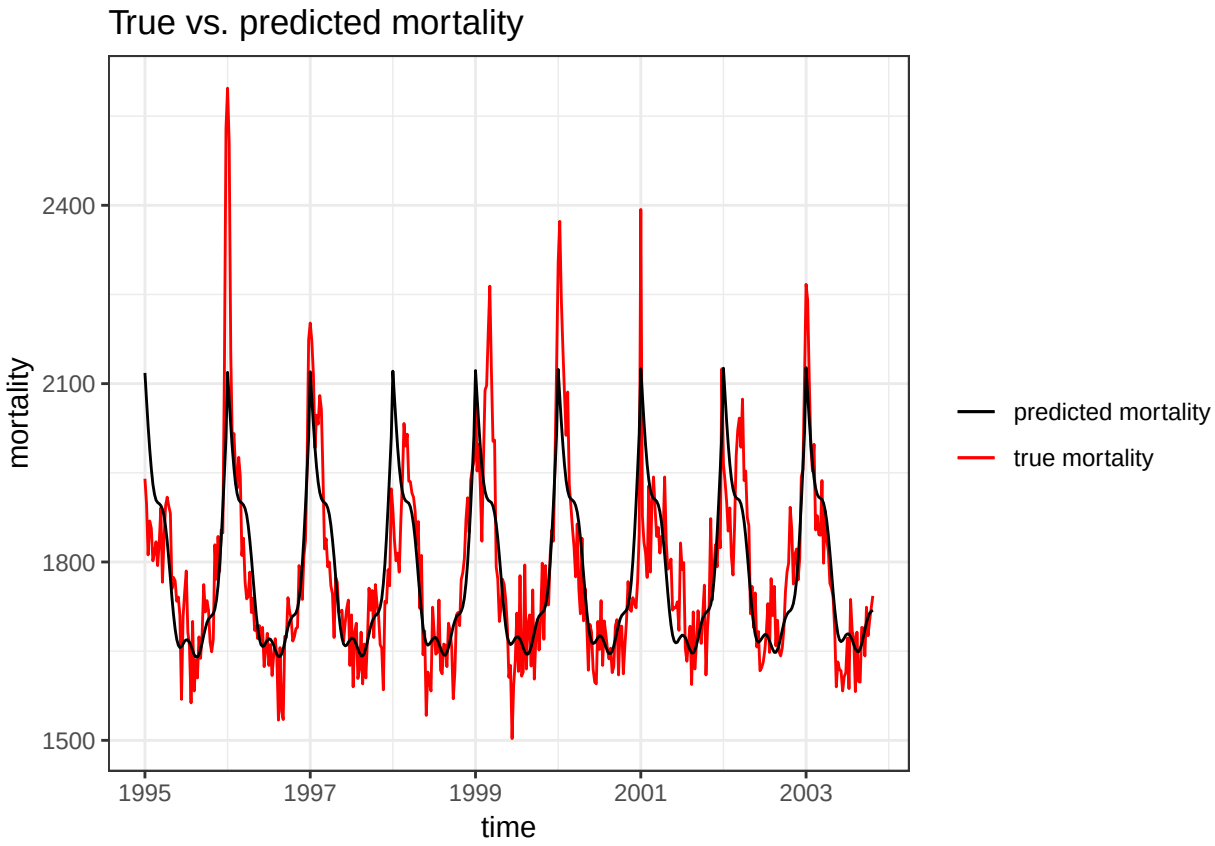
where

$$\epsilon = N(0, \sigma^2).$$

1.3

```
# creating data for plotting
plotData = as.data.frame(cbind(time = data$Time,
                               trueMortality = data$Mortality,
```

```
                                predictedMortality = round(gamModel$fitted.values)))
# plotting
ggplot(data = plotData,
       aes(x = time)) +
  geom_line(aes(y = trueMortality, colour = "true mortality")) +
  geom_line(aes(y = predictedMortality, colour = "predicted mortality")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "True vs. predicted mortality",
       y = "mortality",
       colour = "") +
  scale_colour_manual(values = c("black", "red")) +
  theme_bw()
```



In general, since the prediction and observed values are quite similar in most of the cases, the quality of the fit seems to be pretty good. Nevertheless, some of the peaks observed in the true mortality cannot be completely reproduced by the predicted mortality. It becomes clear that the distribution of the true mortality follows a specific pattern (regularly, there are peaks visible). That is why I would not say that there is a trend from one year to another.

In the next step, the output of the GAM model will be investigated.

```
# analysing model output
summary(gamModel)
```

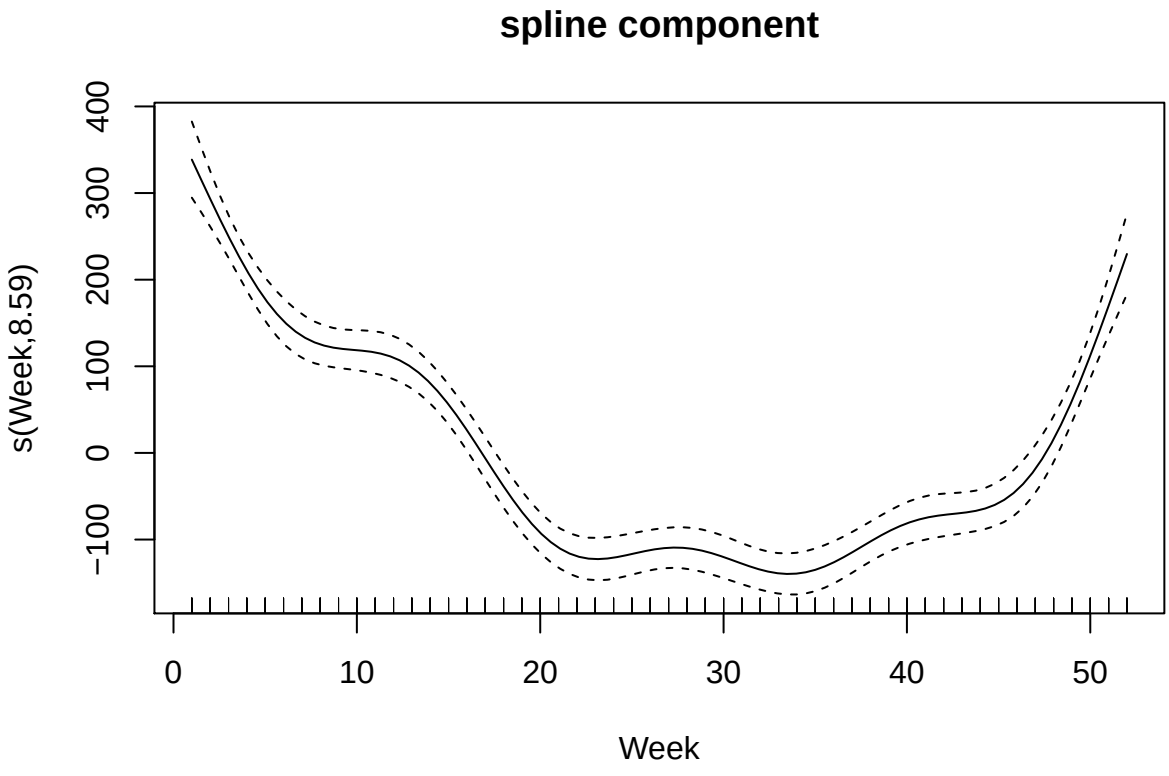
```
##
## Family: gaussian
## Link function: identity
##
```

```
## Formula:
## Mortality ~ Year + s(Week)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -652.060   3448.379  -0.189     0.85
## Year           1.219      1.725   0.706     0.48
##
## Approximate significance of smooth terms:
##           edf Ref.df     F p-value
## s(Week) 8.587  8.951 100.3  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.661   Deviance explained = 66.8%
## GCV = 9014.6  Scale est. = 8806.7    n = 459
```

Characterized by a p-values much higher than 0.05, the parametric coefficients (*Intercept* and *Year*) does not seem to be significant in the model. In contrast, the spline function of *Week* shows a p-value lower than 0.05. These infomartion combined leads to the assumption that within thus GAM model, *Mortality* will be predicted using only the spline component.

The spline component will be plotted:

```
# plotting spline component
plot(gamModel, main = "spline component")
```



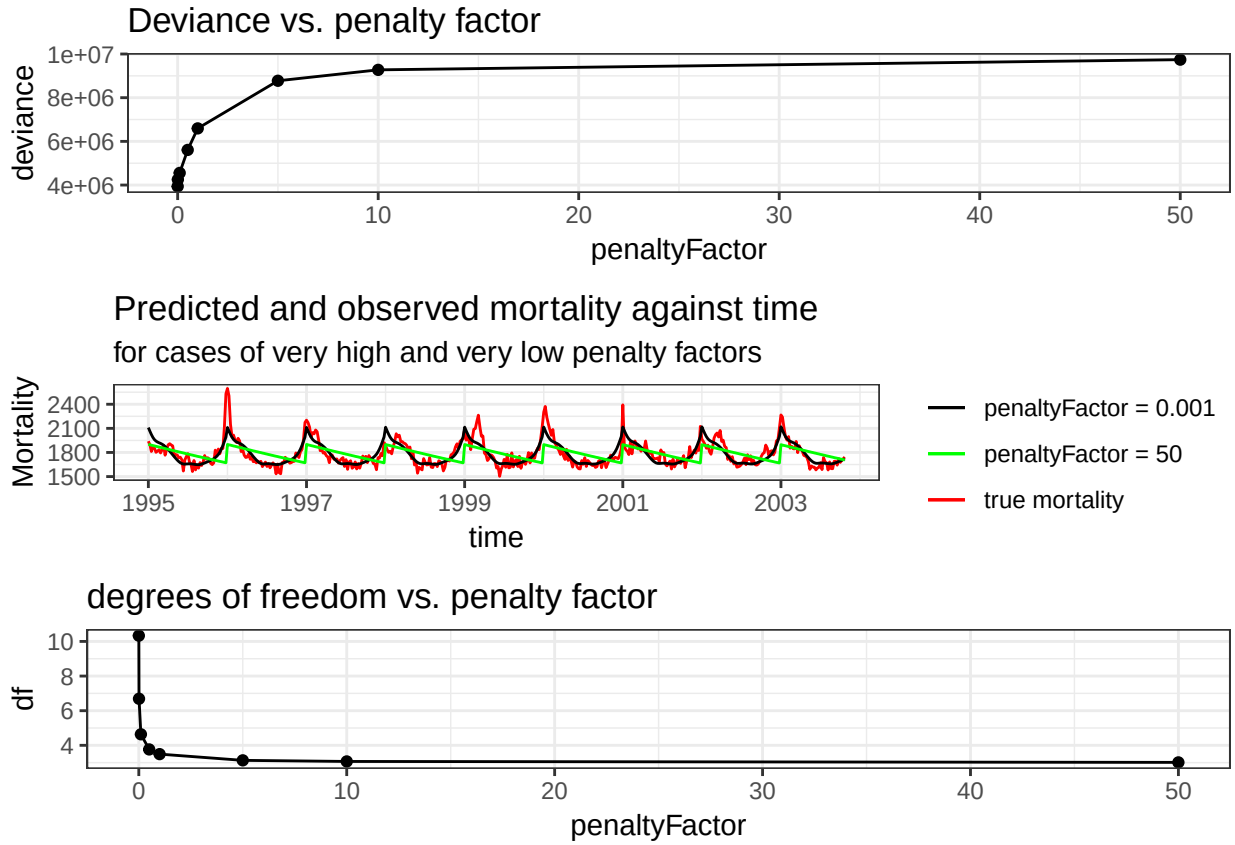
As it can be seen in the plot, the target variable of the model (*Mortality*) varies a lot for different weeks

within a year. As a result, during winter, the mortality seems to be predicted as much higher than during summer. The fact that people get diseases much faster during winter explains this result.

1.4

```
library(gridExtra)
# setting up empty data frames for loop
predictionsDiffPenalty = data.frame(cbind(time = data$Time, trueMortality = data$Mortality))
deviancesDiffPenalty = setNames(data.frame(matrix(ncol = 2, nrow = 0)),
                                c("penaltyFactor", "deviance"))
dfDiffPenalty = setNames(data.frame(matrix(ncol = 2, nrow = 0)), c("penaltyFactor", "df"))
# calculating prediction values and deviances for gam models with different penalty factors
for (penaltyFactor in c(0.001, 0.01, 0.1, 0.5, 1, 5, 10, 50)) {
  # creating gam model
  gamModel = gam(formula = Mortality ~ Year + s(Week,
                                                k = length(unique(data$Week)),
                                                sp = penaltyFactor),
                 data = data)
  # adding prediction values to predictionsDiffPenalty
  predictionsDiffPenalty = cbind(predictionsDiffPenalty,
                                 round(gamModel$fitted.values))
  colnames(predictionsDiffPenalty)[ncol(predictionsDiffPenalty)] =
    paste0("penaltyFactor_", penaltyFactor)
  # adding deviance to deviancesDiffPenalty
  deviancesDiffPenalty = rbind(deviancesDiffPenalty,
                               cbind(penaltyFactor, deviance = gamModel$deviance))
  # adding degrees of freedom to dfDiffPenalty
  dfDiffPenalty = rbind(dfDiffPenalty,
                        cbind(penaltyFactor, df = sum(gamModel$edf)))
}
# plotting results
grid.arrange(
  ggplot(data = deviancesDiffPenalty, aes(x = penaltyFactor, y = deviance)) +
    geom_line() +
    geom_point() +
    theme_bw() +
    labs(title = "Deviance vs. penalty factor"),
  ggplot(data = predictionsDiffPenalty, aes(x = time)) +
    geom_line(aes(y = trueMortality, colour = "true mortality")) +
    geom_line(aes(y = penaltyFactor_0.001, colour = "penaltyFactor = 0.001")) +
    geom_line(aes(y = penaltyFactor_50, colour = "penaltyFactor = 50")) +
    scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
    labs(title = "Predicted and observed mortality against time",
         subtitle = "for cases of very high and very low penalty factors",
         y = "Mortality",
         colour = "") +
    scale_colour_manual(values = c("black", "green", "red")) +
    theme_bw(),
  ggplot(data = dfDiffPenalty, aes(x = penaltyFactor, y = df)) +
    geom_line() +
    geom_point() +
    theme_bw() +
    labs(title = "degrees of freedom vs. penalty factor"),
```

```
  nrow = 3
)
```

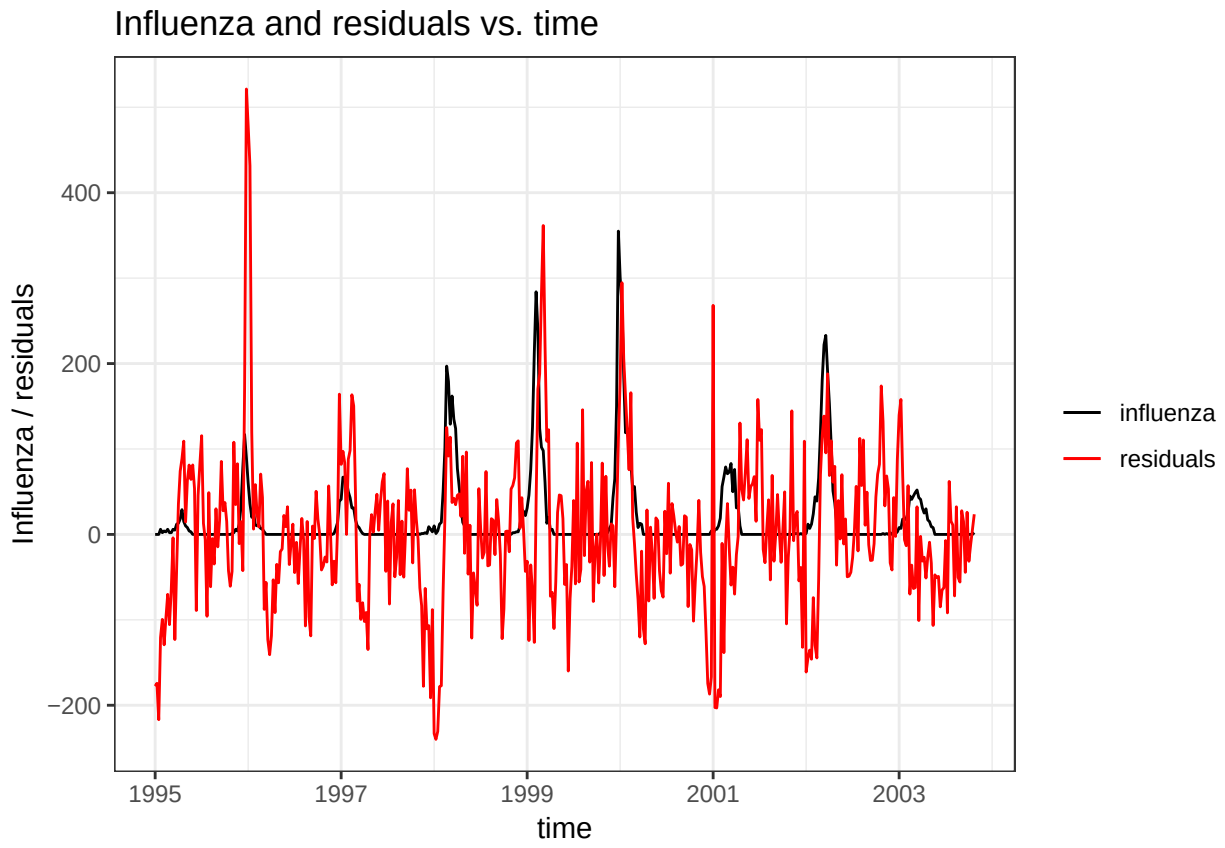


Within the three different plots, observations related to the model using different penalty fators can be made. It can be seen that the higher the penalty factor, the higher the deviance and the lower the degrees of freedom. Here it has to be added that both the deviance and degrees of freedom converge to specific values and do not change in a remarkable way anymore for an increasing penalty factor bigger than 10. Furtermore, the plot in the middle refers to a comparison between observed and predicted mortality using different penalty factors. Here, two extreme values for the penalty factor were chosen and it shows that for the very small penalty factor (0.001) the prediction seem to be much closer to reality than for the model with an extreme high penaly factor (50).

1.5

```
# fitting model from step 2
gamModel = gam(formula = Mortality ~ Year + s(Week),
               data = data)
# creating data for plotting
plotData = as.data.frame(cbind(time = data$Time,
                               influenza = data$Influenza,
                               residuals = gamModel$residuals))
ggplot(data = plotData,
       aes(x = time)) +
  geom_line(aes(y = influenza, colour = "influenza")) +
  geom_line(aes(y = residuals, colour = "residuals")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
```

```
  labs(title = "Influenza and residuals vs. time",
       y = "Influenza / residuals",
       colour = "") +
  scale_colour_manual(values = c("black", "red")) +
  theme_bw()
```



Within the plot, it does not seem as if there would be a correlation between the residuals and the influenza. Both graphs follow different patterns.

1.6

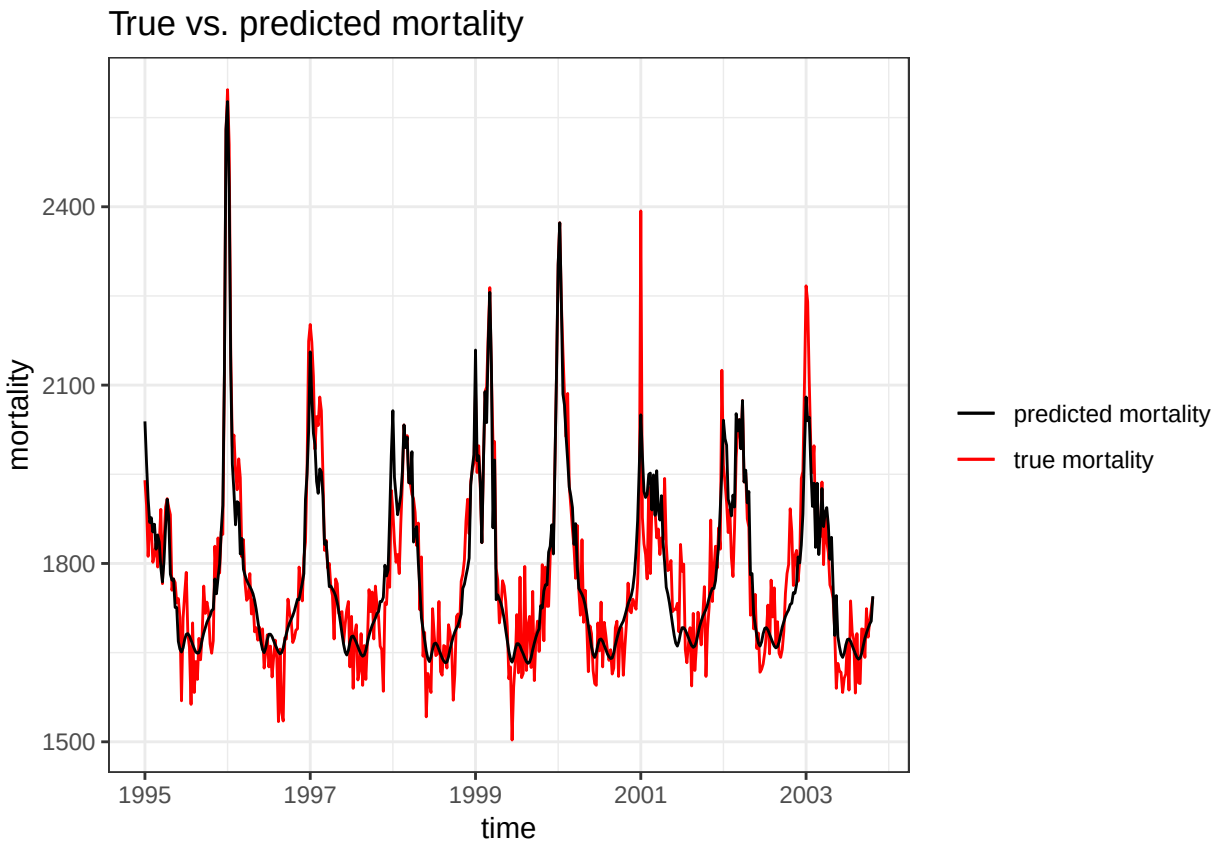
```
# fitting gam model
gamModel = gam(formula = Mortality ~
                 s(Year, k = length(unique(data$Year))) +
                 s(Week, k = length(unique(data$Week))) +
                 s(Influenza, k = length(unique(data$Influenza))),
                 data = data)
# analysing model output
summary(gamModel)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## Mortality ~ s(Year, k = length(unique(data$Year))) + s(Week,
```

```
##     k = length(unique(data$Week))) + s(Influenza, k = length(unique(data$Influenza)))
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1783.8        3.2   557.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                 edf Ref.df      F p-value
## s(Year)       4.663  5.677  1.487   0.181
## s(Week)      14.641 18.248 18.533  <2e-16 ***
## s(Influenza) 69.729 72.840  5.599  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Rank: 134/144
## R-sq.(adj) =  0.819   Deviance explained = 85.4%
## GCV = 5846.7  Scale est. = 4699.8    n = 459
```

Looking at the p-value of the smoothed term for *Influenza* (much smaller than 5%), it can be concluded that the mortality seems to be influenced by the outbreaks of influenza.

```
# creating data for plotting
plotData = as.data.frame(cbind(time = data$Time,
                               trueMortality = data$Mortality,
                               predictedMortality = round(gamModel$fitted.values)))
# plotting
ggplot(data = plotData,
       aes(x = time)) +
  geom_line(aes(y = trueMortality, colour = "true mortality")) +
  geom_line(aes(y = predictedMortality, colour = "predicted mortality")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "True vs. predicted mortality",
       y = "mortality",
       colour = "") +
  scale_colour_manual(values = c("black", "red")) +
  theme_bw()
```



Comparing the observed and predicted mortality, this model definitely seems to lead to the most accurate results. The peaks are well predicted and also the general pattern between both lines is very similar.

Assignment 2: High-dimensional methods

At first, the data from the file *data.csv* will be imported.

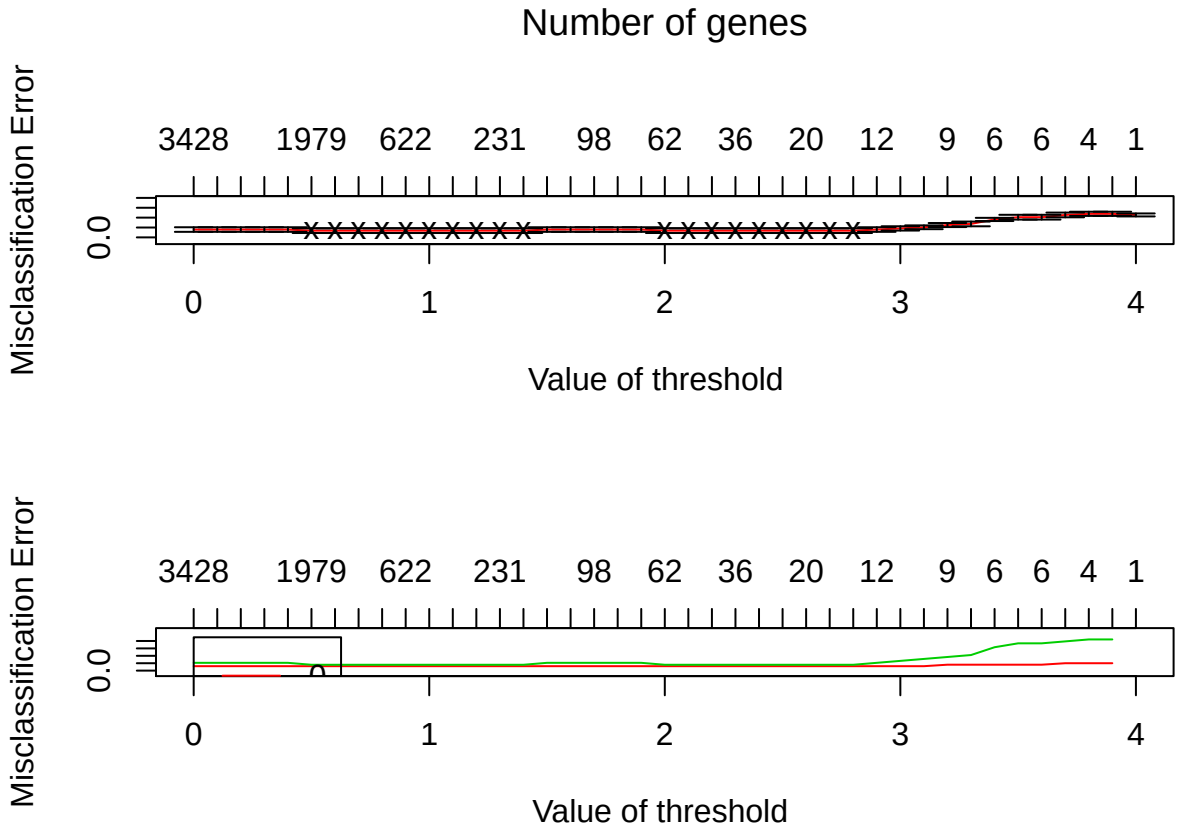
```
# importing data
data = read.csv2("data.csv")
```

2.1

```
library(pamr)
# dividing data into train and test set
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.7))
train = data[id,]
test = data[-id,]
# correcting rownames of train data
rownames(train) = 1:nrow(train)
# extracting x and y for train data
x = t(train[,-which(colnames(train) == "Conference")])
y = train[,which(colnames(train) == "Conference")]
# creating list using x and y for modelling
mydata = list(x = x,
```

```
              y = as.factor(y),
              geneid = as.character(1:nrow(x)),
              genenames = rownames(x))
# performing nearest shrunken centroid classification of training data
model = pamr.train(mydata, threshold = seq(0,4, 0.1))
# cross-validating nearest shrunken centroid classifier
set.seed(12345)
cvmodel = pamr.cv(model, mydata)
```

```
# plotting misclassification error vs. threshold
pamr.plotcv(cvmodel)
```



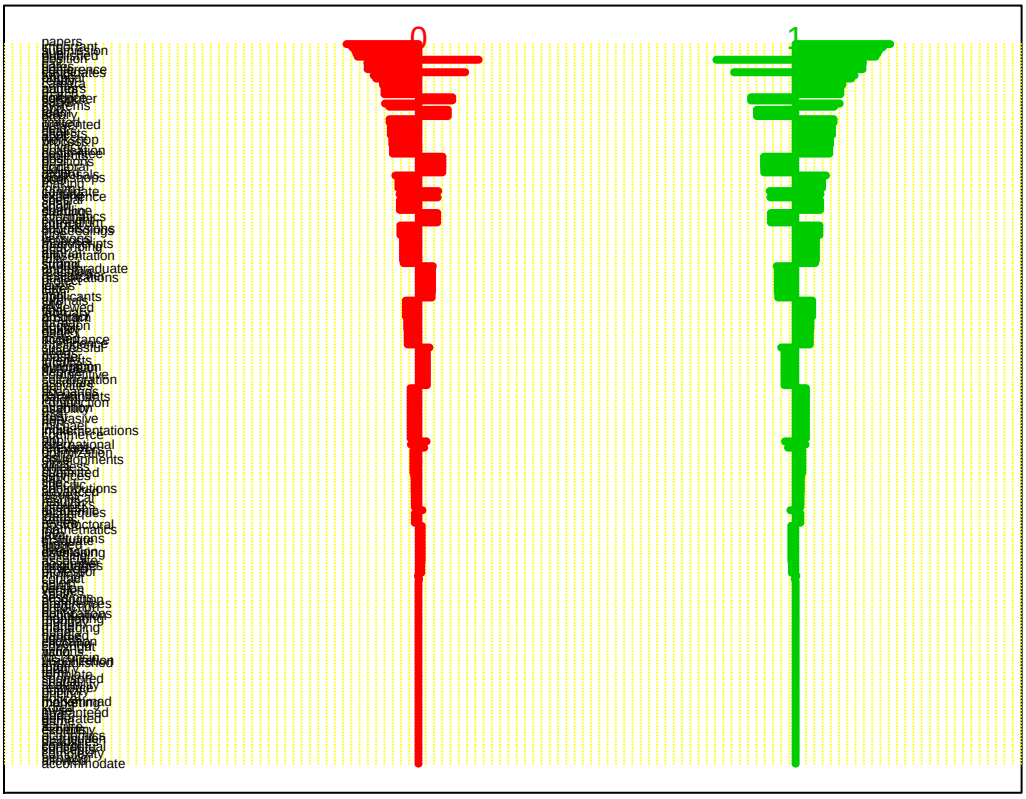
In the plots it can be seen that the threshold with the minimum misclassification error has to be roughly between 0.5 to 1.4 or between 2.0 and 2.8. To find the exact thresholds with the mimunum error, we extract the information from the created *cvmodel*.

```
# identifying thresholds with minimum misclassification error
cvmodel$threshold[which(cvmodel$error == min(cvmodel$error))]
```

```
##  [1] 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 2.0 2.1 2.2 2.3 2.4 2.5 2.6
## [18] 2.7 2.8
```

Our guess has been confirmed. Therefore, the optimal threshold is one of the values between 0.5 to 1.4 or between 2.0 and 2.8. For integrity reasons, so that I reproduce the same results as my group mates, I decide to choose the threshold 1.3 for the further processes. Of course, I could have also chosen any other value between 0.5 and 1.4 or between 2.0 and 2.8.

```
# plotting centroid plot for for threshold
pamr.plotcen(model, mydata, threshold = 1.3)
```




```
# identifying most contributing features
selectedFeatures = as.data.frame(pamr.listgenes(model, mydata, threshold = 1.3))
```

```
nrow(selectedFeatures)
```

```
## [1] 231
```

In total, the model with a threshold of 1.3 leads to 231 selected features. The ten most contributing features can be seen in the table:

```
knitr::kable(setNames(as.data.frame(colnames(data[,selectedFeatures$id[1:10]])), "feature"))
```

feature
active
aachen
aicit
X2012call
advance
adami
ambient
anastasia
X10th
ambitious

Furthermore, the created model will be used to classify the test data.

```r
# correcting rownames of test data
rownames(test) = 1:nrow(test)
# extracting x and y for test data
xTest = t(test[,-which(colnames(test) == "Conference")])
yTest = test[,which(colnames(test) == "Conference")]
# using trained model to classify test data with optimal threshold of 1.3
testClassification  = pamr.predict(model, newx = xTest, threshold = 1.3)
# evaluating model (function will be used in 2.2 as well)
evaluatingModel = function(yTrue, yFitted, saveError = F, nFeatures = F, model = NULL) {
  # calculating missclassification error
  print(paste0("Misclassification error: ", mean(yTrue != yFitted)))
  # creating confusion matrix
  print(table(y = yTrue, yFit = yFitted))
  # calculating number of contributing features
  if (isTRUE(nFeatures)) {
    coefficients = coef(model, s = "lambda.min")
    if (is.list(coefficients)) {
      print(paste0("Number of contributing features: ", length(coefficients[[1]])))
    } else {
      print(paste0("Number of contributing features: ", length(coefficients@i)))
      knitr::kable(setNames(data.frame(coefficients@Dimnames[[1]][coefficients@i + 1]),
                            "feature"))
    }
  }
  if (isTRUE(saveError)) {
    return(mean(yTrue != yFitted))
  }
}
evaluatingModel(yTest, testClassification)
```

```
## [1] "Misclassification error: 0.1"
##    yFit
## y    0  1
##   0 10  0
##   1  2  8
```

The miscassification error of 0.1 indicates that the model classified the test data with a high accuracy.

2.2

```r
# changing class for x and xTest from data.frame to matrix
x = as.matrix(train[,-which(colnames(train) == "Conference")])
xTest = as.matrix(test[,-which(colnames(test) == "Conference")])
```

a. Elastic Net

```r
library(glmnet)
# fitting elastic net using cross-validation
cvElastic = cv.glmnet(x = x, y = y, alpha = 0.5, family = "binomial")
# using trained model to classify test data
testClassificationElasticNet = predict.cv.glmnet(cvElastic, newx = xTest,
                                                  s = "lambda.min", type = "class")
# evaluating model
evaluatingModel(yTest, testClassificationElasticNet, nFeatures = T, model = cvElastic)
```

```
## [1] "Misclassification error: 0.1"
##    yFit
## y    0  1
##   0 10  0
##   1  2  8
## [1] "Number of contributing features: 35"
```

b. Support Vector Machine

```
library(kernlab)
# fitting svm model
svmModel = ksvm(x, y, kernel="vanilladot", scale = FALSE, type = "C-svc")
```

```
##  Setting default kernel parameters
```

```
# using trained model to classify test data
testClassificationSVM <- predict(svmModel, xTest, type="response")
# evaluating model
evaluatingModel(yTest, testClassificationSVM, nFeatures = T, model = svmModel)
```

```
## [1] "Misclassification error: 0.05"
##    yFit
## y    0  1
##   0 10  0
##   1  1  9
## [1] "Number of contributing features: 43"
```

c. Comparison of the models with the results of the nearest shrunken centroids

```
knitr::kable(as.data.frame(
  cbind("Nearest Shrunken Centroid Model" =
          evaluatingModel(yTest, testClassification, saveError = T),
        "ElasticNet" =
          evaluatingModel(yTest, testClassificationElasticNet, saveError = T),
        "SVM" =
          evaluatingModel(yTest, testClassificationSVM, saveError = T))))
```

```
## [1] "Misclassification error: 0.1"
##    yFit
## y    0  1
##   0 10  0
##   1  2  8
## [1] "Misclassification error: 0.1"
##    yFit
## y    0  1
##   0 10  0
##   1  2  8
## [1] "Misclassification error: 0.05"
##    yFit
## y    0  1
##   0 10  0
##   1  1  9
```

Nearest Shrunken Centroid Model	ElasticNet	SVM
0.1	0.1	0.05

Based on the comparison of the misclassification rates, the *SVM*-model is the most accurate model. That is why I prefer this model.

2.3

```
pValue = c()
for (i in 1:ncol(data[,-which(colnames(data) == "Conference")])){
  x = data[,i]
  testResult = t.test(x ~ Conference,
                      data = data,
                      alternative = "two.sided")
  pValue[i] = testResult$p.value
}
pValue = as.data.frame(pValue)
pValue$reject_flag = as.factor(ifelse(pValue$pValue <0.05, "Retain", "Drop"))
pValue$column_index = row.names(pValue)
keep = ifelse(pValue$reject_flag == "Retain", as.numeric(pValue$column_index), NA)
keep = na.omit(keep)
rejected = colnames(data[,keep])
rejected
```

```
##   [1] "abstract"        "academic"        "acceptance"      "accepted"        "access"          "acm
##  [28] "bio"             "call"            "calls"           "camera"          "canada"          "can
##  [55] "contributions"   "copyright"       "covering"        "cross"           "curriculum"      "dat
##  [82] "expected"        "experience"      "extension"       "feature"         "february"        "fig
## [109] "include"         "included"        "india"           "infrastructures" "initially"       "ins
## [136] "letter"          "levels"          "limited"         "liu"             "looking"         "mad
## [163] "ontologies"      "opportunity"     "optimization"    "org"             "organizers"      "org
## [190] "privacy"         "proceedings"     "process"         "professor"       "proficiency"     "pro
## [217] "scalability"     "scenarios"       "science"         "scope"           "security"        "ser
## [244] "taiwan"          "takes"           "tasks"           "teaching"        "team"            "tec
## [271] "versions"        "vienna"          "visualization"   "vitae"           "wang"            "wir
```

All listed features have a p-value lower than 5%. Based on this information, for all of these ones a significant influence on *Conference* will be assumed.

Appendix

```
# importing data
library(readxl)
data = read_excel("Influenza.xlsx")
library(ggplot2)
# plotting development of mortality and influenza
ggplot(data = data,
       aes(x = Time)) +
  geom_line(aes(y = Mortality, colour = "Mortality")) +
  geom_line(aes(y = Influenza, colour = "Influenza")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "Development of mortality and influenza",
```

```
       y = "Mortality / Influenza",
       colour = "") +
  scale_colour_manual(values = c("red", "black")) +
  theme_bw()
library(mgcv)
# fitting gam model
gamModel = gam(formula = Mortality ~ Year + s(Week),
               data = data)
# creating data for plotting
plotData = as.data.frame(cbind(time = data$Time,
                               trueMortality = data$Mortality,
                               predictedMortality = round(gamModel$fitted.values)))
# plotting
ggplot(data = plotData,
       aes(x = time)) +
  geom_line(aes(y = trueMortality, colour = "true mortality")) +
  geom_line(aes(y = predictedMortality, colour = "predicted mortality")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "True vs. predicted mortality",
       y = "mortality",
       colour = "") +
  scale_colour_manual(values = c("black", "red")) +
  theme_bw()
# analysing model output
summary(gamModel)
# plotting spline component
plot(gamModel, main = "spline component")
library(gridExtra)
# setting up empty data frames for loop
predictionsDiffPenalty = data.frame(cbind(time = data$Time, trueMortality = data$Mortality))
deviancesDiffPenalty = setNames(data.frame(matrix(ncol = 2, nrow = 0)),
                                c("penaltyFactor", "deviance"))
dfDiffPenalty = setNames(data.frame(matrix(ncol = 2, nrow = 0)), c("penaltyFactor", "df"))
# calculating prediction values and deviances for gam models with different penalty factors
for (penaltyFactor in c(0.001, 0.01, 0.1, 0.5, 1, 5, 10, 50)) {
  # creating gam model
  gamModel = gam(formula = Mortality ~ Year + s(Week,
                                                k = length(unique(data$Week)),
                                                sp = penaltyFactor),
                 data = data)
  # adding prediction values to predictionsDiffPenalty
  predictionsDiffPenalty = cbind(predictionsDiffPenalty,
                                 round(gamModel$fitted.values))
  colnames(predictionsDiffPenalty)[ncol(predictionsDiffPenalty)] =
    paste0("penaltyFactor_", penaltyFactor)
  # adding deviance to deviancesDiffPenalty
  deviancesDiffPenalty = rbind(deviancesDiffPenalty,
                               cbind(penaltyFactor, deviance = gamModel$deviance))
  # adding degrees of freedom to dfDiffPenalty
  dfDiffPenalty = rbind(dfDiffPenalty,
                        cbind(penaltyFactor, df = sum(gamModel$edf)))
}
# plotting results
```

```
grid.arrange(
  ggplot(data = deviancesDiffPenalty, aes(x = penaltyFactor, y = deviance)) +
    geom_line() +
    geom_point() +
    theme_bw() +
    labs(title = "Deviance vs. penalty factor"),
  ggplot(data = predictionsDiffPenalty, aes(x = time)) +
    geom_line(aes(y = trueMortality, colour = "true mortality")) +
    geom_line(aes(y = penaltyFactor_0.001, colour = "penaltyFactor = 0.001")) +
    geom_line(aes(y = penaltyFactor_50, colour = "penaltyFactor = 50")) +
    scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
    labs(title = "Predicted and observed mortality against time",
         subtitle = "for cases of very high and very low penalty factors",
         y = "Mortality",
         colour = "") +
    scale_colour_manual(values = c("black", "green", "red")) +
    theme_bw(),
  ggplot(data = dfDiffPenalty, aes(x = penaltyFactor, y = df)) +
    geom_line() +
    geom_point() +
    theme_bw() +
    labs(title = "degrees of freedom vs. penalty factor"),
  nrow = 3
)
# fitting model from step 2
gamModel = gam(formula = Mortality ~ Year + s(Week),
               data = data)
# creating data for plotting
plotData = as.data.frame(cbind(time = data$Time,
                               influenza = data$Influenza,
                               residuals = gamModel$residuals))
ggplot(data = plotData,
       aes(x = time)) +
  geom_line(aes(y = influenza, colour = "influenza")) +
  geom_line(aes(y = residuals, colour = "residuals")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "Influenza and residuals vs. time",
       y = "Influenza / residuals",
       colour = "") +
  scale_colour_manual(values = c("black", "red")) +
  theme_bw()
# fitting gam model
gamModel = gam(formula = Mortality ~
                 s(Year, k = length(unique(data$Year))) +
                 s(Week, k = length(unique(data$Week))) +
                 s(Influenza, k = length(unique(data$Influenza))),
                 data = data)
# analysing model output
summary(gamModel)

# creating data for plotting
plotData = as.data.frame(cbind(time = data$Time,
                               trueMortality = data$Mortality,
```

```
                                predictedMortality = round(gamModel$fitted.values)))
# plotting
ggplot(data = plotData,
       aes(x = time)) +
  geom_line(aes(y = trueMortality, colour = "true mortality")) +
  geom_line(aes(y = predictedMortality, colour = "predicted mortality")) +
  scale_x_continuous(breaks=c(1995,1997,1999,2001,2003)) +
  labs(title = "True vs. predicted mortality",
       y = "mortality",
       colour = "") +
  scale_colour_manual(values = c("black", "red")) +
  theme_bw()
# importing data
data = read.csv2("data.csv")
library(pamr)
# dividing data into train and test set
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.7))
train = data[id,]
test = data[-id,]
# correcting rownames of train data
rownames(train) = 1:nrow(train)
# extracting x and y for train data
x = t(train[,-which(colnames(train) == "Conference")])
y = train[,which(colnames(train) == "Conference")]
# creating list using x and y for modelling
mydata = list(x = x,
              y = as.factor(y),
              geneid = as.character(1:nrow(x)),
              genenames = rownames(x))
# performing nearest shrunken centroid classification of training data
model = pamr.train(mydata, threshold = seq(0,4, 0.1))
# cross-validating nearest shrunken centroid classifier
set.seed(12345)
cvmodel = pamr.cv(model, mydata)
# plotting misclassification error vs. threshold
pamr.plotcv(cvmodel)
# identifying thresholds with minimum misclassification error
cvmodel$threshold[which(cvmodel$error == min(cvmodel$error))]
# plotting centroid plot for for threshold
pamr.plotcen(model, mydata, threshold = 1.3)
# identifying most contributing features
selectedFeatures = as.data.frame(pamr.listgenes(model, mydata, threshold = 1.3))
nrow(selectedFeatures)
knitr::kable(setNames(as.data.frame(colnames(data[,selectedFeatures$id[1:10]])), "feature"))
# correcting rownames of test data
rownames(test) = 1:nrow(test)
# extracting x and y for test data
xTest = t(test[,-which(colnames(test) == "Conference")])
yTest = test[,which(colnames(test) == "Conference")]
# using trained model to classify test data with optimal threshold of 1.3
testClassification  = pamr.predict(model, newx = xTest, threshold = 1.3)
```

```r
# evaluating model (function will be used in 2.2 as well)
evaluatingModel = function(yTrue, yFitted, saveError = F, nFeatures = F, model = NULL) {
  # calculating missclassification error
  print(paste0("Misclassification error: ", mean(yTrue != yFitted)))
  # creating confusion matrix
  print(table(y = yTrue, yFit = yFitted))
  # calculating number of contributing features
  if (isTRUE(nFeatures)) {
    coefficients = coef(model, s = "lambda.min")
    if (is.list(coefficients)) {
      print(paste0("Number of contributing features: ", length(coefficients[[1]])))
    } else {
      print(paste0("Number of contributing features: ", length(coefficients@i)))
      knitr::kable(setNames(data.frame(coefficients@Dimnames[[1]][coefficients@i + 1]),
                            "feature"))
    }
  }
  if (isTRUE(saveError)) {
    return(mean(yTrue != yFitted))
  }
}
evaluatingModel(yTest, testClassification)
# changing class for x and xTest from data.frame to matrix
x = as.matrix(train[,-which(colnames(train) == "Conference")])
xTest = as.matrix(test[,-which(colnames(test) == "Conference")])
library(glmnet)
# fitting elastic net using cross-validation
cvElastic = cv.glmnet(x = x, y = y, alpha = 0.5, family = "binomial")
# using trained model to classify test data
testClassificationElasticNet = predict.cv.glmnet(cvElastic, newx = xTest,
                                                 s = "lambda.min", type = "class")
# evaluating model
evaluatingModel(yTest, testClassificationElasticNet, nFeatures = T, model = cvElastic)
library(kernlab)
# fitting svm model
svmModel = ksvm(x, y, kernel="vanilladot", scale = FALSE, type = "C-svc")
# using trained model to classify test data
testClassificationSVM <- predict(svmModel, xTest, type="response")
# evaluating model
evaluatingModel(yTest, testClassificationSVM, nFeatures = T, model = svmModel)
knitr::kable(as.data.frame(
  cbind("Nearest Shrunken Centroid Model" =
          evaluatingModel(yTest, testClassification, saveError = T),
        "ElasticNet" =
          evaluatingModel(yTest, testClassificationElasticNet, saveError = T),
        "SVM" =
          evaluatingModel(yTest, testClassificationSVM, saveError = T))))
pValue = c()
for (i in 1:ncol(data[,-which(colnames(data) == "Conference")])){
  x = data[,i]
  testResult = t.test(x ~ Conference,
                      data = data,
                      alternative = "two.sided")
```

```
  pValue[i] = testResult$p.value
}
pValue = as.data.frame(pValue)
pValue$reject_flag = as.factor(ifelse(pValue$pValue <0.05, "Retain", "Drop"))
pValue$column_index = row.names(pValue)
keep = ifelse(pValue$reject_flag == "Retain", as.numeric(pValue$column_index), NA)
keep = na.omit(keep)
rejected = colnames(data[,keep])
rejected
```