Mandatory assignment 2

Jonas Øren

```
library(dplyr)
```

Problem 1: Regression

We read the data

```
daphnia_data <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00505/qsar_aquatic_
# Name the variables. The last column, LC50, is the response
names(daphnia_data) <- c("TPSA", "SAacc", "H050", "MLOGP", "RDCHI", "GATS1p",
                         "nN", "C040", "LC50")
n <- nrow(daphnia_data)
```

And split into a training and a test set

```
set.seed(1)
train_idx <- sample(1:n, size = round(2/3*n), replace = FALSE)
test_idx <- (1:n)[-train_idx]
```

1. Analysis of count variables

Linear effect of count variables

```
# fit <- lm(LC50 ~ H050 + nN + C040, data = daphnia_data[train_idx, ])
fit_linear <- lm(LC50 ~ ., data = daphnia_data[train_idx, ])
```

Dichotmoized count variables

```
# Dochotomize count data to 0 (not present), and 1 (present)
dich_data <- daphnia_data
dich_data <- mutate_at(dich_data, c("H050", "nN", "C040"),
                       function(x) {x[x > 0] <- 1; return(factor(x))}
                       )
fit_dich <- lm(LC50 ~ ., data = dich_data[train_idx, ])
```

Summaries

Coefficients We first compare the two models by inspecting coefficient of the fitted regression models.

For the model with linear effects we have.

```
summary(fit_linear)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = daphnia_data[train_idx, ])
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6963 -0.7768 -0.1321  0.5957  4.8947
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.424483   0.307580   7.882 3.98e-14 ***
## TPSA         0.030984   0.003354   9.238  < 2e-16 ***
## SAacc       -0.016823   0.002673  -6.295 9.09e-10 ***
## H050         0.104049   0.074467   1.397  0.16321
## MLOGP        0.497885   0.079383   6.272 1.04e-09 ***
## RDCHI        0.498203   0.164706   3.025  0.00267 **
## GATS1p      -0.469568   0.194267  -2.417  0.01615 *
## nN          -0.284790   0.061188  -4.654 4.60e-06 ***
## C040         0.002479   0.113210   0.022  0.98254
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.215 on 355 degrees of freedom
## Multiple R-squared:  0.5014, Adjusted R-squared:  0.4902
## F-statistic: 44.62 on 8 and 355 DF,  p-value: < 2.2e-16
```

The model with dichotomized count variables gives the following output.

```r
summary(fit_dich)
```

```
##
## Call:
## lm(formula = LC50 ~ ., data = dich_data[train_idx, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5656 -0.7770 -0.1308  0.6153  5.0254
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.615321   0.318876   8.202 4.39e-15 ***
## TPSA         0.026486   0.003370   7.859 4.67e-14 ***
## SAacc       -0.013988   0.002296  -6.093 2.88e-09 ***
## H0501       -0.033537   0.156050  -0.215  0.82996
## MLOGP        0.477585   0.079174   6.032 4.07e-09 ***
## RDCHI        0.471415   0.169659   2.779  0.00575 **
## GATS1p      -0.528849   0.190036  -2.783  0.00568 **
## nN1         -0.202498   0.154128  -1.314  0.18975
## C0401       -0.140733   0.165083  -0.852  0.39451
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.248 on 355 degrees of freedom
## Multiple R-squared:  0.4744, Adjusted R-squared:  0.4626
## F-statistic: 40.05 on 8 and 355 DF,  p-value: < 2.2e-16
```

From the printouts we see that the linear effect of nN is significant while the dichotomized variable is not.

We also see that the coefficient for GATS1p in the dichotomized model have lower p-values than the coefficients in the model with linear effect of all variables. This effect changes between different test and training set combinations, and seem to be an effect of random variation more than from the model choice.

Prediction error as MSE in test data We now inspect the prediction error, measured as squared error on the test data set

$$\frac{1}{n}\sum_{i\in\mathcal{T}}(\hat{y}_i - y_i)^2,$$

where $\mathcal{T}$ is the set of indexes for the observations of the test set.

```r
# True respons on the test set
truth <- daphnia_data$LC50[test_idx]
# Predictions on the test set
prediction_linear_model <- predict(fit_linear, newdata = daphnia_data[test_idx, ])
prediction_dich_model <- predict(fit_dich, newdata = dich_data[test_idx, ])
cat(paste("Prediction error linear model:      ",
          round(mean((prediction_linear_model - truth)^2)
                                 , 2), "\n" ) )
```

```
## Prediction error linear model:      1.42
```

```r
cat(paste("Prediction error dichotomous model:",
          round(mean((prediction_dich_model - truth)^2)
                                 , 2) , "\n" ) )
```

```
## Prediction error dichotomous model: 1.43
```

We see that the model with linear effects for all the variables has a smaller prediction error on our test set.

From this analysis linear effects of count variables appear unambiguously better at predicting and explaining the outcome, meaning the amount of molecules also has an effect, and we should not just look at the presence of molecules.

2. Repeating the procedure for different test and training sets

```r
set.seed(2)
# We will sum up the errors to compute the average
lin_mod_error_sum <- 0
dic_mod_error_sum <- 0

for (i in 1:200) {
  # Create new test and training sets
  train_idx_i <- sample(1:n, size = round(2/3*n), replace = FALSE)
  test_idx_i <- (1:n)[-train_idx_i]

  # Fit the new models
  fit_linear <- lm(LC50 ~ ., data = daphnia_data[train_idx_i, ])
  fit_dich <- lm(LC50 ~ ., data = dich_data[train_idx_i, ])

  # Find prediction error

  truth <- daphnia_data$LC50[test_idx_i]
  prediction_linear_model <- predict(fit_linear, newdata = daphnia_data[test_idx_i, ])
  prediction_dich_model <- predict(fit_dich, newdata = dich_data[test_idx_i, ])

  lin_mod_error_sum <- lin_mod_error_sum + mean((prediction_linear_model - truth)^2)
  dic_mod_error_sum <- dic_mod_error_sum + mean((prediction_dich_model - truth)^2)
}

cat(paste("Average prediction error linear model:     ",
          round(lin_mod_error_sum/200, 3), "\n",
          "Average prediction error dichotomous model:",
          round(dic_mod_error_sum/200, 3), "\n"))
```

```
## Average prediction error linear model:      1.491
##  Average prediction error dichotomous model: 1.537
```

The model with dichotomized effects has on average a bigger prediction error than the model with linear effects. Dichotomizing the variables ignore the size of the effect, which discards much information. I would guess this will be responsible for the poorer prediction performance of dichotomized variables.

3. Variable selection

http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/154-stepwise-regression-essentials-in-r/

Stepwise selection

```r
full_model <- lm(LC50 ~ ., data = daphnia_data[train_idx, ])
n <- length(train_idx)

forwards_AIC <- step(full_model,
                     direction = "forward",
                     k = 2, # AIC
                     trace = 0
                     )
backwards_AIC <- step(full_model,
                     direction = "backward",
                     k = 2,
                     trace = 0
                     )
forwards_BIC <- step(full_model,
                     direction = "forward",
                     k = log(n),
                     trace = 0
                     )
backwards_BIC <- step(full_model,
                     direction = "backward",
                     k = log(n),
                     trace = 0
                     )

model_names <- c(  "Forwards  AIC: "
                 , "Backwards AIC: "
                 , "Forwards  BIC: "
                 , "Backwards BIC: "
)

lasso_CV <- glmnet::glmnet(x = model.matrix(~ . , data = daphnia_data[train_idx, ]),
                           y = daphnia_data$LC50[train_idx],
                           family = "gaussian")
```

```r
# Print formulas to compare selected models
cat(do.call(function(...) paste(..., sep = "\n")
            , as.list(diag(outer( model_names
                                , lapply( c(  forwards_AIC$terms[[3]]
                                            , backwards_AIC$terms[[3]]
                                            , forwards_BIC$terms[[3]]
                                            , backwards_BIC$terms[[3]]
                                            )
                                          , deparse # Convert formula to string
                                          )
                                , paste)))))
```

```
## Forwards  AIC:  TPSA + SAacc + H050 + MLOGP + RDCHI + GATS1p + nN + C040
## Backwards AIC:  TPSA + SAacc + H050 + MLOGP + RDCHI + GATS1p + nN
## Forwards  BIC:  TPSA + SAacc + H050 + MLOGP + RDCHI + GATS1p + nN + C040
## Backwards BIC:  TPSA + SAacc + MLOGP + RDCHI + GATS1p + nN
```

We see that selection based on BIC selects more sparse models. This is as expected as the BIC penalizes the dimensionality of the model more than AIC.

We also see that backward selection chooses a more sparse models than forward selection.

The most frugal method chose not to include H050 and C040 as predictors. The coefficients for these variables were also not significant in the full model fit.

All the methods seem to agree on which variables are the most important.

Prediction error We now compare the prediction errors of the models, computed on the test set.

```r
truth <- daphnia_data$LC50[test_idx]
models <- list(forwards_AIC, backwards_AIC, forwards_BIC, backwards_BIC)

for (i in 1:4) {
  predicted <- predict(models[[i]], newdata = daphnia_data[test_idx,])
  cat(paste(model_names[i], round(mean((truth - predicted)^2), 4), "\n"))
}
```

```
## Forwards  AIC:  1.4197
## Backwards AIC:  1.4198
## Forwards  BIC:  1.4197
## Backwards BIC:  1.3978
```

The most sparse model, chosen by backwards elimination using BIC, performed better than the more saturated models on the test set.

Backward selection has the advantage of accounting for the effect of other important variables in the model when deciding on whether to keep a variable in the model. This may be why this method chose the best model in this case.

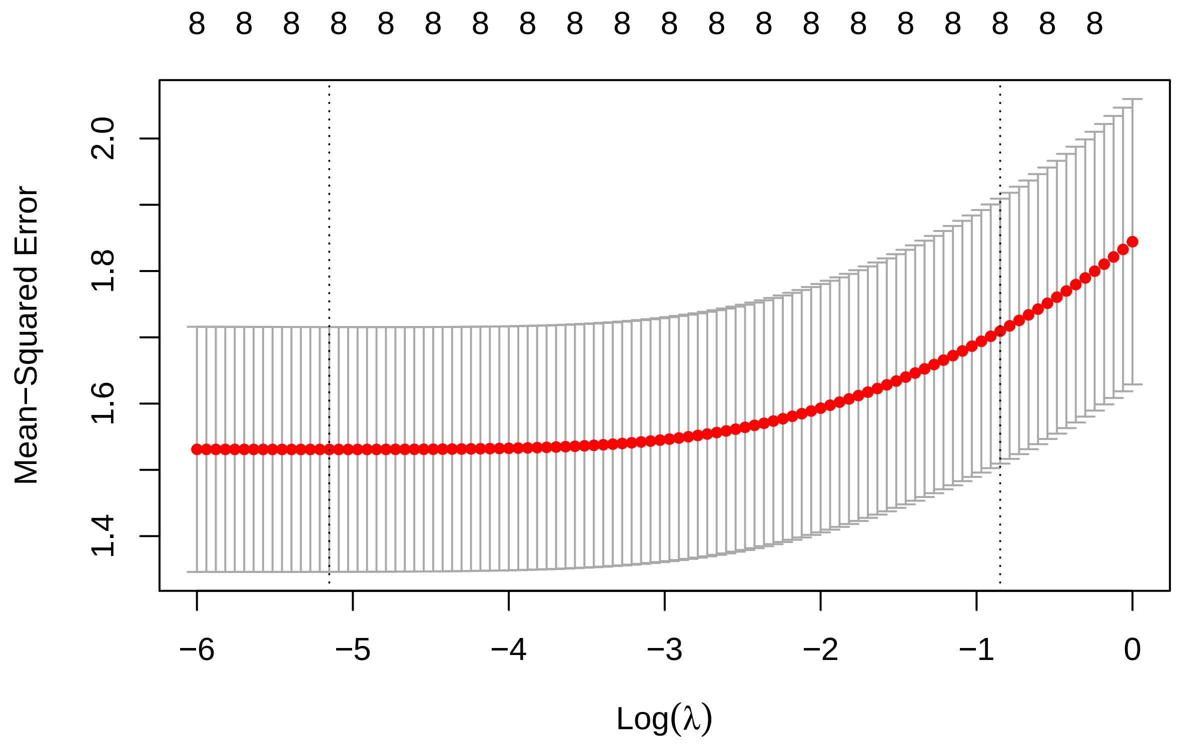
4. Ridge regression

Find tuning parameter

Cross validation The number of folds in cross validation is a bias variance tradeoff. Leave one out cross-validation has low bias but high variance, because the cross-validation sets are so similar to each other, we are close to only making one estimation of the error. The bias depends on sample size and the effect of sample size on model bias and variance. Because we have relatively many observations we will choose to use 10-fold cross validation.

We now find the optimal lambda according to deviance, and plot the error estimated for each lambda.

```r
library(glmnet)
set.seed(432)
x = model.matrix(LC50 ~ . - 1, data = daphnia_data[train_idx, ])
y = daphnia_data$LC50[train_idx]
cv_lambda <- cv.glmnet(x, y
                       , lambda = exp(seq(-6, 0, length.out = 100))
                       , nfolds = 10
                       , alpha = 0   # Ridge regression
                       )
plot(cv_lambda)
```



The dotted lines show the minimizing lambda, and the lambda chosen according to the "one standard deviation rule". The "one standard deviation rule" is to choose the most parsimonious model parameter less than one standard deviation from the minimizing value.

```r
print(cv_lambda$lambda.1se)
```

```
## [1] 0.428063
```

```r
print(mean(cv_lambda$cvsd))
```

```
## [1] 0.1894781
```

Cross validation chooses $\lambda = 0.42$. The estimated standard deviation of the cross validation estimate is 0.19.

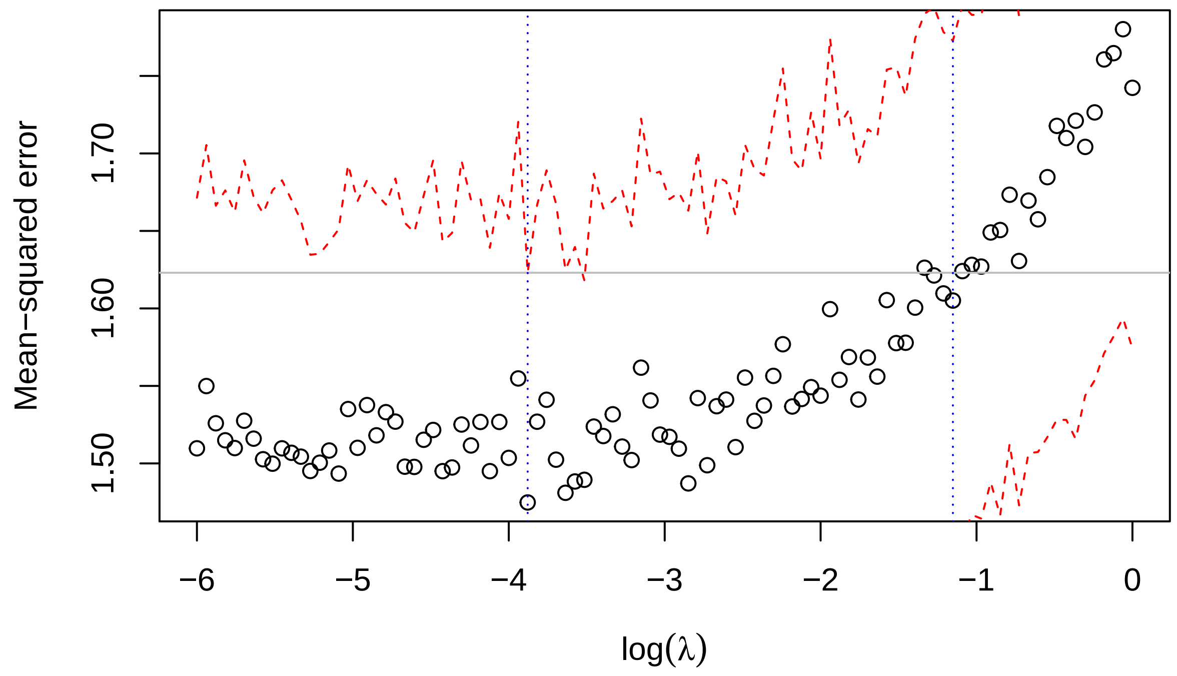
Bootstrapping We will now estimate the optimal λ with respect to deviance using bootstrapping.

```r
# The chosen grid of lambda-values, to optimize over
lda_seq <- exp(seq(-6, 0, length.out = 100))

deviance_estimate <- NULL
error_SD_estimate <- NULL
```

```
for (lda in 1:100) {
  MSE_boot <- NULL
  for (i in 1:100) {
    id_boot <- sample(1:nrow(daphnia_data), replace = TRUE)

    # Select bootstrap sample
    x = model.matrix(LC50 ~ . -1, data = daphnia_data[id_boot, ])
    y = daphnia_data$LC50[id_boot]
    # Store mean squared prediction error
    MSE_boot[i] <- mean( (predict(glmnet(x, y, family = "gaussian", alpha = 0,
                                         lambda = lda_seq[lda])
                                  , newx = model.matrix(LC50 ~ . -1, data = daphnia_data[-id_boot, ])
                                  , newy = daphnia_data$LC50[-id_boot]
                                  )
                          - daphnia_data$LC50[-id_boot])^2
                         )
  }
  # Average over error for this lambda
  deviance_estimate[lda] <- mean(MSE_boot)
  # Estimate standard deviation of errors.
  error_SD_estimate[lda] <- sd(MSE_boot)
}
```

We plot the estimated expected prediction error for each lambda.

```
plot(log(lda_seq), deviance_estimate, xlab=expression(log(lambda)),
     ylab="Mean-squared error")
lines(log(lda_seq), deviance_estimate + error_SD_estimate, lty=2, col="red")
lines(log(lda_seq), deviance_estimate - error_SD_estimate, lty=2, col="red")
id_lowest_error <- which.min(deviance_estimate)
one_sd_from_min <- deviance_estimate[id_lowest_error] + error_SD_estimate[id_lowest_error]
abline(h=one_sd_from_min,
       lty=1, col="grey")

# Find lambda according to "one standard deviation rule"
boot_lda <- max(lda_seq[deviance_estimate < one_sd_from_min])
abline(v=log(lda_seq[id_lowest_error]), lty=3, col="blue")
abline(v=log(boot_lda), lty=3, col="blue")
```



The red bands are estimated standard deviation of the estimated error. The blue lines shows the minimizing lambda and the lambda chosen by the "one standard deviation rule".

```r
print(boot_lda)
```

```
## [1] 0.3161574
```

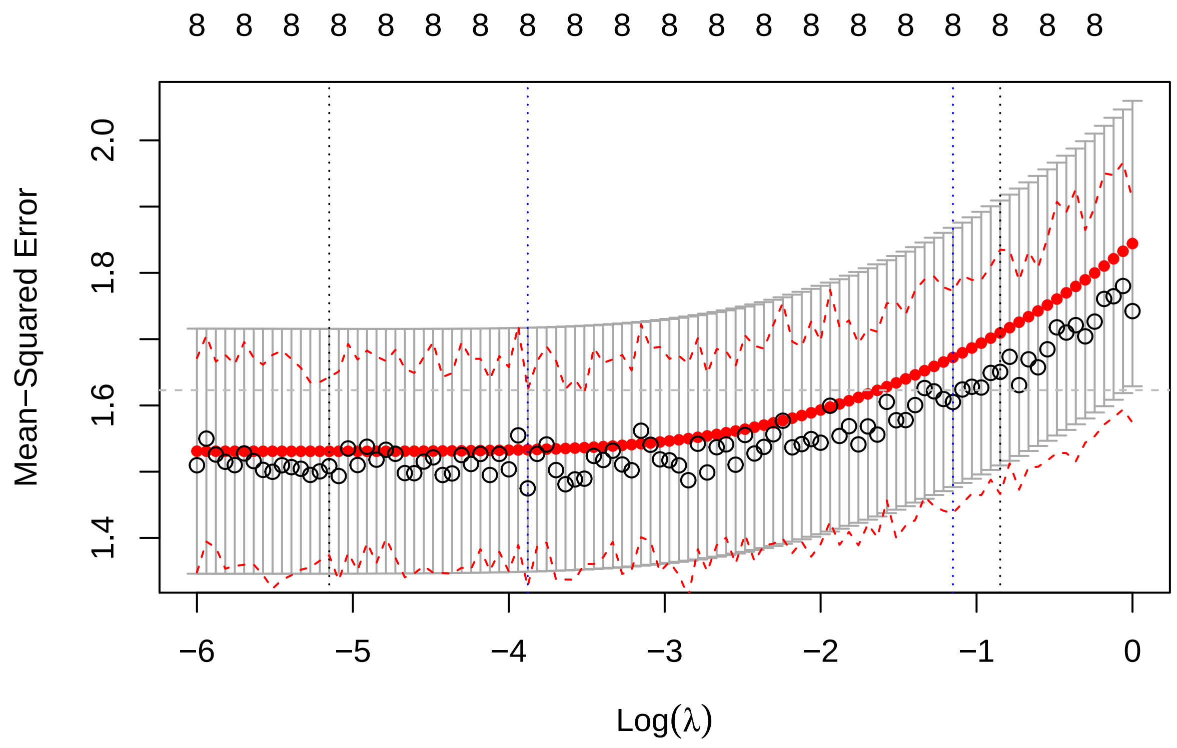
```r
print(mean(error_SD_estimate))
```

```
## [1] 0.1592806
```

The bootstrapping procedure chooses $\lambda = 0.32$. The estimated standard deviation of the bootstrap estimates is 0.16.

In order to compare the two procedures we now superimpose the two plots of estimated errors.

```r
plot(cv_lambda)
points(log(lda_seq), deviance_estimate, xlab=expression(log(lambda)),
       ylab="Mean-squared error")
lines(log(lda_seq), deviance_estimate + error_SD_estimate, lty=2, col="red")
lines(log(lda_seq), deviance_estimate - error_SD_estimate, lty=2, col="red")
id_lowest_error <- which.min(deviance_estimate)
one_sd_from_min <- deviance_estimate[id_lowest_error] + error_SD_estimate[id_lowest_error]
abline(h=one_sd_from_min,
       lty=2, col="grey")

# Find lambda according to "one standard deviation rule"
boot_lda <- max(lda_seq[deviance_estimate < one_sd_from_min])
abline(v=log(lda_seq[id_lowest_error]), lty=3, col="blue")
abline(v=log(boot_lda), lty=3, col="blue")
```



We see that in this case the procedures obtained a similar result. There is still some difference, and it could have been higher. There is much variability in each procedure, and the somewhat arbitrary choice of number of folds and number of bootstrap iterations will effect this.

5. Generalized additive models

We load the required package.

```
library(gam)
```

And fit a relatively complex GAM with smoothing splines for each variable. The complexity parameter, effective degrees of freedom of the splines is set to five, a moderately complex model. Effective degrees of freedom is linked to the penalty on the second derivative of the smoothing splines. One degree of freedom produces an almost linear fit.

```
# Fit the GAM
fit_GAM <- gam(LC50 ~ s(TPSA, 5)
           + s(SAacc, 5)
           + s(H050, 5)
           + s(MLOGP, 5)
           + s(RDCHI, 5)
           + s(GATS1p, 5)
           + s(nN, 5)
           + s(C040, 5)
           , family = gaussian
           , data = daphnia_data[train_idx, ])
summary(fit_GAM)
```

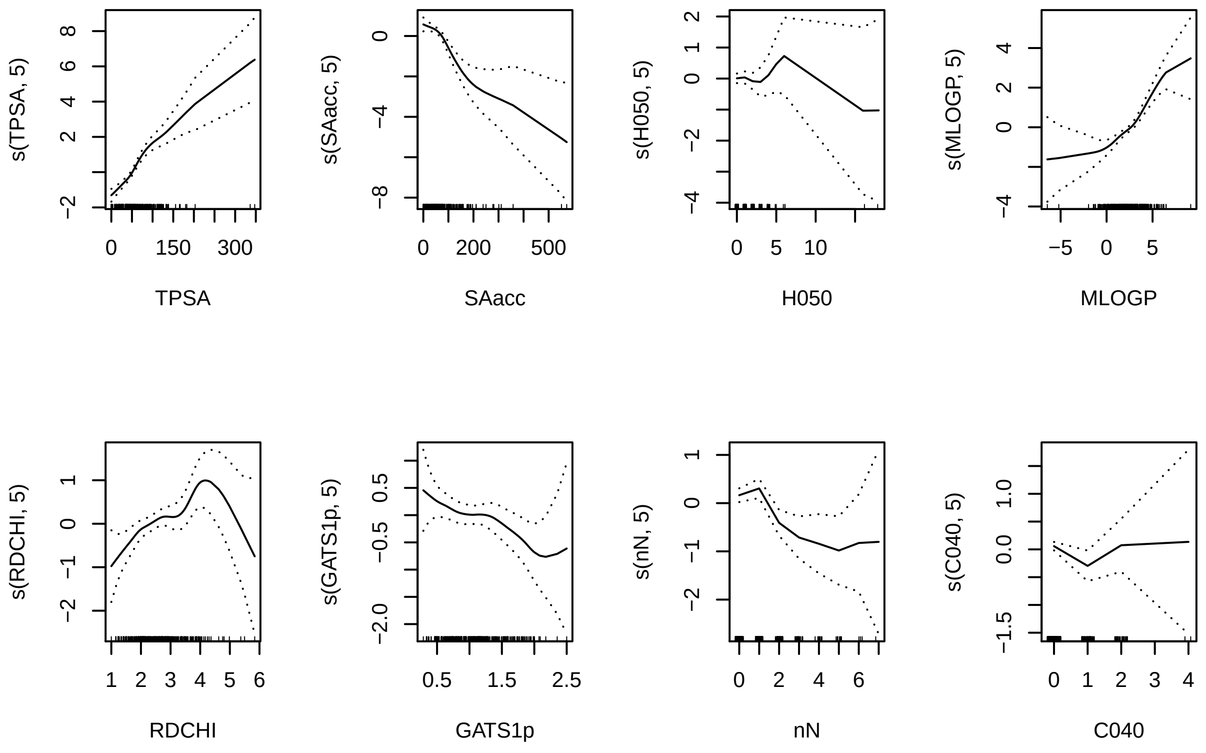
```
##
## Call: gam(formula = LC50 ~ s(TPSA, 5) + s(SAacc, 5) + s(H050, 5) +
##     s(MLOGP, 5) + s(RDCHI, 5) + s(GATS1p, 5) + s(nN, 5) + s(C040,
##     5), family = gaussian, data = daphnia_data[train_idx, ])
## Deviance Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -4.00384 -0.71521 -0.05359  0.54987  4.88077
##
## (Dispersion Parameter for gaussian family taken to be 1.304)
##
##     Null Deviance: 1051.494 on 363 degrees of freedom
## Residual Deviance: 423.7867 on 325.0007 degrees of freedom
## AIC: 1168.342
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##               Df Sum Sq Mean Sq  F value    Pr(>F)
## s(TPSA, 5)     1   0.16    0.16   0.1239    0.7250
## s(SAacc, 5)    1  63.06   63.06  48.3613 1.954e-11 ***
## s(H050, 5)     1  32.46   32.46  24.8942 9.886e-07 ***
## s(MLOGP, 5)    1 321.78  321.78 246.7724 < 2.2e-16 ***
## s(RDCHI, 5)    1   1.99    1.99   1.5266    0.2175
## s(GATS1p, 5)   1   8.32    8.32   6.3831    0.0120 *
## s(nN, 5)       1  23.23   23.23  17.8115 3.168e-05 ***
## s(C040, 5)     1   0.86    0.86   0.6629    0.4161
## Residuals    325 423.79    1.30
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##              Npar Df Npar F     Pr(F)
## (Intercept)
## s(TPSA, 5)         4 3.0897 0.0161553 *
## s(SAacc, 5)        4 4.3721 0.0018597 **
## s(H050, 5)         4 1.0634 0.3746106
## s(MLOGP, 5)        4 5.0081 0.0006272 ***
## s(RDCHI, 5)        4 5.1467 0.0004946 ***
## s(GATS1p, 5)       4 0.6754 0.6094389
## s(nN, 5)           4 3.0382 0.0175966 *
## s(C040, 5)         2 1.9989 0.1371493
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that, except for C040, all parameters have a significant effect either in the parametric or in the nonparametric effect, where the nonparametric effect is simply the linear effect.

We now plot the parameters, to inspect the nonlinearity of the estimated effects.

```r
par(mfrow=c(2,4))
plot(fit_GAM, se=T)
```



The algorithm does not seem to fit very nonlinear effects at the bulk of the observations. There might be a slight nonlinear effect for the categorical variables H050, nN and C040. Perhaps this would better be modeled by factor variables, but at the cost of added complexity and risk of some overfitting.

We now fit a less complex GAM for comparison. We set the estimated degrees of freedom for the splines to one.

```
# Fit the GAM
linear_GAM <- gam(LC50 ~ s(TPSA, 1.1)
            + s(SAacc, 1.1)
            + s(H050, 1.1)
            + s(MLOGP, 1.1)
            + s(RDCHI, 1.1)
            + s(GATS1p, 1.1)
            + s(nN, 1.1)
            + s(C040, 1.1)
            , family = gaussian
            , data = daphnia_data[train_idx, ])
summary(linear_GAM)
```

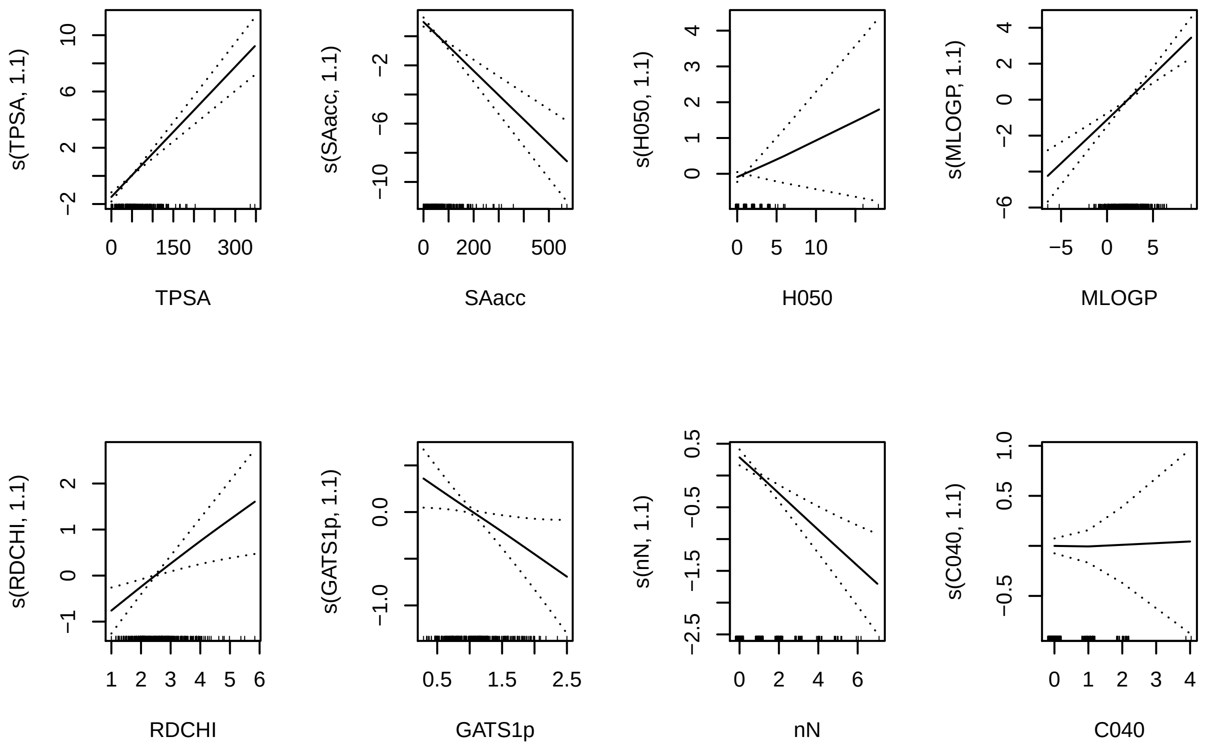
```
##
## Call: gam(formula = LC50 ~ s(TPSA, 1.1) + s(SAacc, 1.1) + s(H050, 1.1) +
##     s(MLOGP, 1.1) + s(RDCHI, 1.1) + s(GATS1p, 1.1) + s(nN, 1.1) +
##     s(C040, 1.1), family = gaussian, data = daphnia_data[train_idx,
##     ])
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6852 -0.7718 -0.1383  0.5809  4.8916
##
## (Dispersion Parameter for gaussian family taken to be 1.4723)
##
```

```
##     Null Deviance: 1051.494 on 363 degrees of freedom
## Residual Deviance: 521.4791 on 354.1999 degrees of freedom
## AIC: 1185.451
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df Sum Sq Mean Sq  F value    Pr(>F)
## s(TPSA, 1.1)     1.0   3.99    3.99   2.7075 0.1007650
## s(SAacc, 1.1)    1.0  70.15   70.15  47.6507 2.363e-11 ***
## s(H050, 1.1)     1.0  19.39   19.39  13.1732 0.0003258 ***
## s(MLOGP, 1.1)    1.0 385.37  385.37 261.7517 < 2.2e-16 ***
## s(RDCHI, 1.1)    1.0   5.50    5.50   3.7334 0.0541311 .
## s(GATS1p, 1.1)   1.0   8.14    8.14   5.5312 0.0192275 *
## s(nN, 1.1)       1.0  31.94   31.94  21.6914 4.537e-06 ***
## s(C040, 1.1)     1.0   0.00    0.00   0.0009 0.9754339
## Residuals      354.2 521.48    1.47
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                Npar Df Npar F   Pr(F)
## (Intercept)
## s(TPSA, 1.1)       0.1 1.3571 0.10505
## s(SAacc, 1.1)      0.1 2.0883 0.08708 .
## s(H050, 1.1)       0.1 2.2998 0.08310 .
## s(MLOGP, 1.1)      0.1 4.1817 0.05920 .
## s(RDCHI, 1.1)      0.1 4.8283 0.05373 .
## s(GATS1p, 1.1)     0.1 0.5636 0.14194
## s(nN, 1.1)         0.1 1.9172 0.09063 .
## s(C040, 1.1)       0.1 1.6537 0.09677 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There is not much increase in AIC from the more complex model, indicating that there is not much gained from the nonlinear effects. It might appear that C040 now has a significant linear effect, but if we inspect the plot of the parameter, we see that the parameter is estimated to be near 0 for all values of C040.

Plots

```r
par(mfrow=c(2,4))
plot(linear_GAM, se=T)
```



6. Regression trees

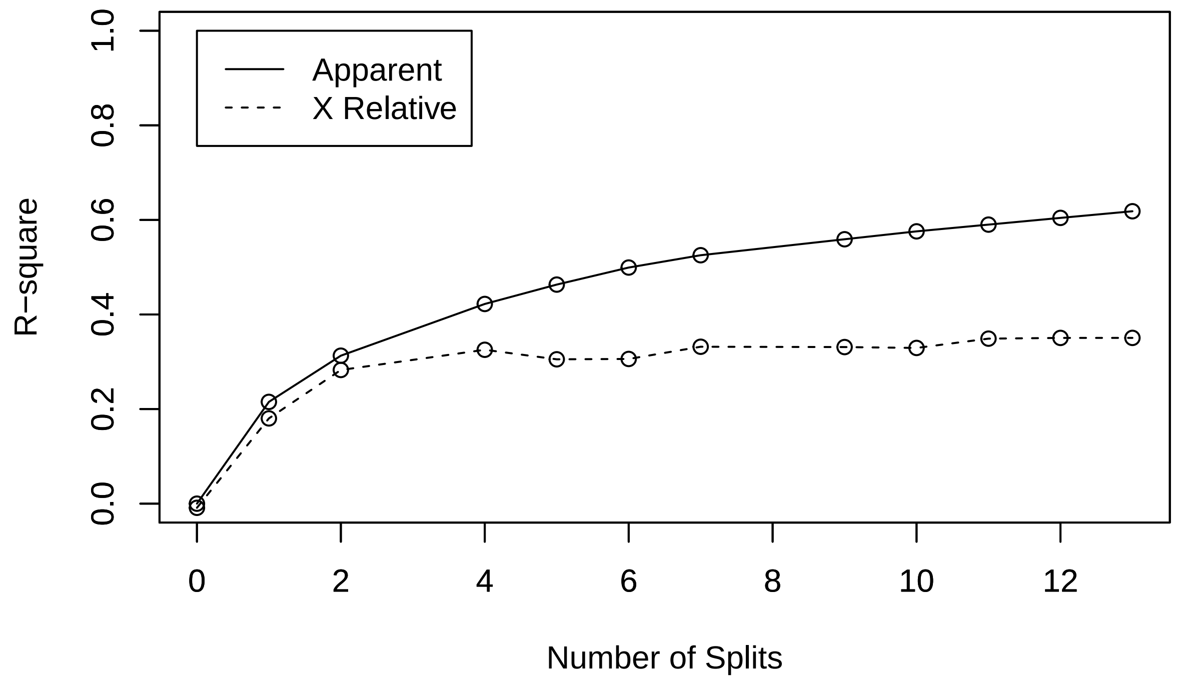
We will use the `rpart` package to compute regression trees. This package uses cross validation to estimate

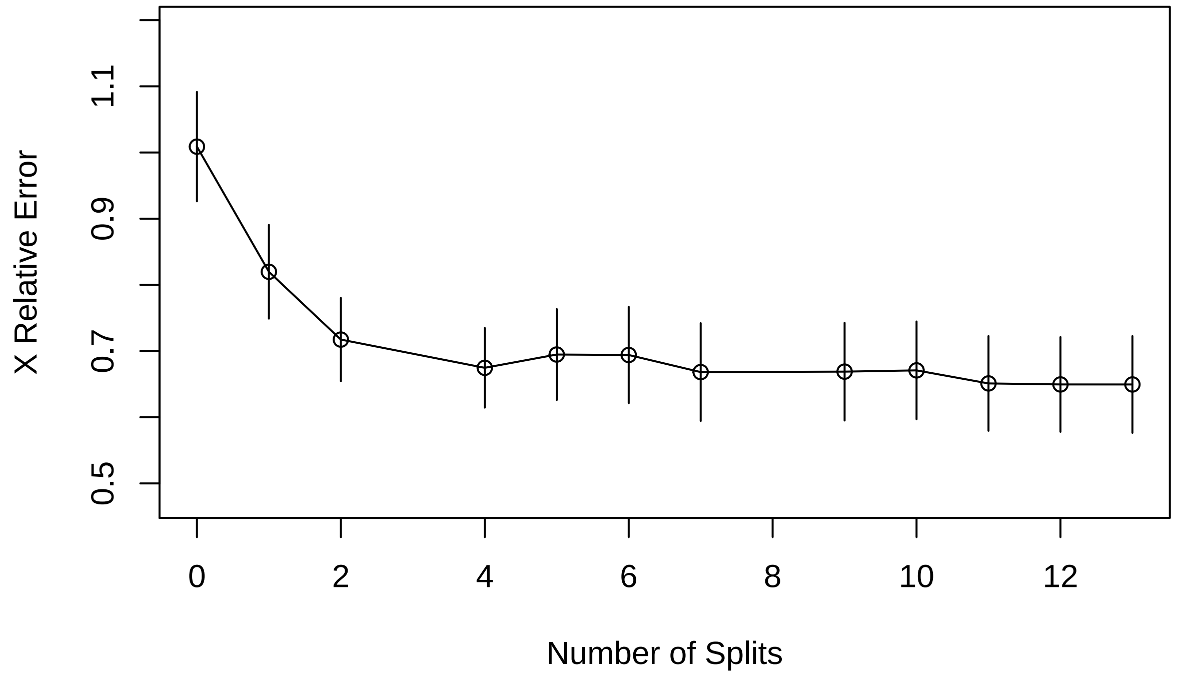
```r
library(rpart)
```

```r
fitted_tree <- rpart(LC50 ~ ., data = daphnia_data[train_idx, ], method = "anova",
                     control = rpart.control(xval = 10)  # 10-fold cross validation for pruning
                     )
```

```r
rsq.rpart(fitted_tree)
```

```
##
## Regression tree:
## rpart(formula = LC50 ~ ., data = daphnia_data[train_idx, ], method = "anova",
##     control = rpart.control(xval = 10))
##
## Variables actually used in tree construction:
## [1] GATS1p H050   MLOGP  RDCHI  SAacc  TPSA
##
## Root node error: 1051.5/364 = 2.8887
##
## n= 364
##
##          CP nsplit rel error  xerror     xstd
## 1  0.215155      0   1.00000 1.00882 0.082650
## 2  0.097755      1   0.78485 0.81976 0.070779
## 3  0.054664      2   0.68709 0.71733 0.062764
## 4  0.040970      4   0.57776 0.67467 0.060171
## 5  0.036055      5   0.53679 0.69475 0.068709
## 6  0.026068      6   0.50074 0.69406 0.072921
## 7  0.016877      7   0.47467 0.66819 0.073959
## 8  0.016688      9   0.44091 0.66887 0.073857
## 9  0.014289     10   0.42423 0.67077 0.073860
## 10 0.014216     11   0.40994 0.65107 0.071569
## 11 0.014048     12   0.39572 0.64953 0.071581
## 12 0.010000     13   0.38167 0.64950 0.073047
```





```
# rpart.control
# rpart.object
# .summary
# .print
```

The algorithm has chosen not to use the count variables nN, and C040. I could not find out why, as it has chosen to keep H050, which is also a count variable.

From the plot of improvement in R-square, and reduction in relative error we see that the biggest improvement is in the first two splits.

We now use 10-fold cross validation to choose the optimal tuning parameter `cp` according to mean squared error.

```
library(caret)
```

```
set.seed(432)
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 10)
tree_fit <- train(LC50 ~ ., data = daphnia_data[train_idx, ],
                 method = 'rpart',
                 trControl = fitControl,
                 ## This last option is actually one
                 ## for gbm() that passes through
                 )
```

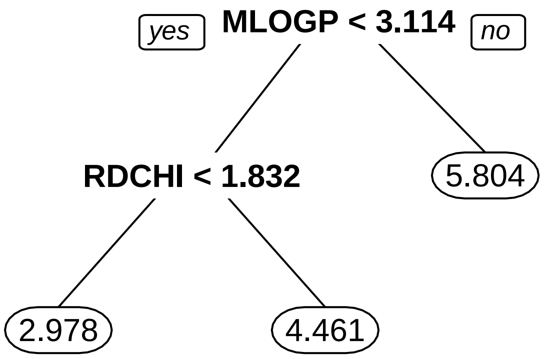
```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
tree_fit
```

```
## CART
##
## 364 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 328, 327, 327, 328, 327, 328, ...
## Resampling results across tuning parameters:
##
##   cp          RMSE      Rsquared   MAE
##   0.05466436  1.449255  0.2920311  1.119398
##   0.09775489  1.498163  0.2404114  1.154781
##   0.21515491  1.654410  0.1231236  1.271848
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.05466436.
```

Cross validation chooses `cp = 0.05466436`. We prune the model using this complexity parameter.

```
pruned <- rpart::prune(fitted_tree, cp = 0.05466436)
# Uses the `rpart.plot` package to plot the tree properly
rpart.plot::prp(pruned, digits = 4)
```

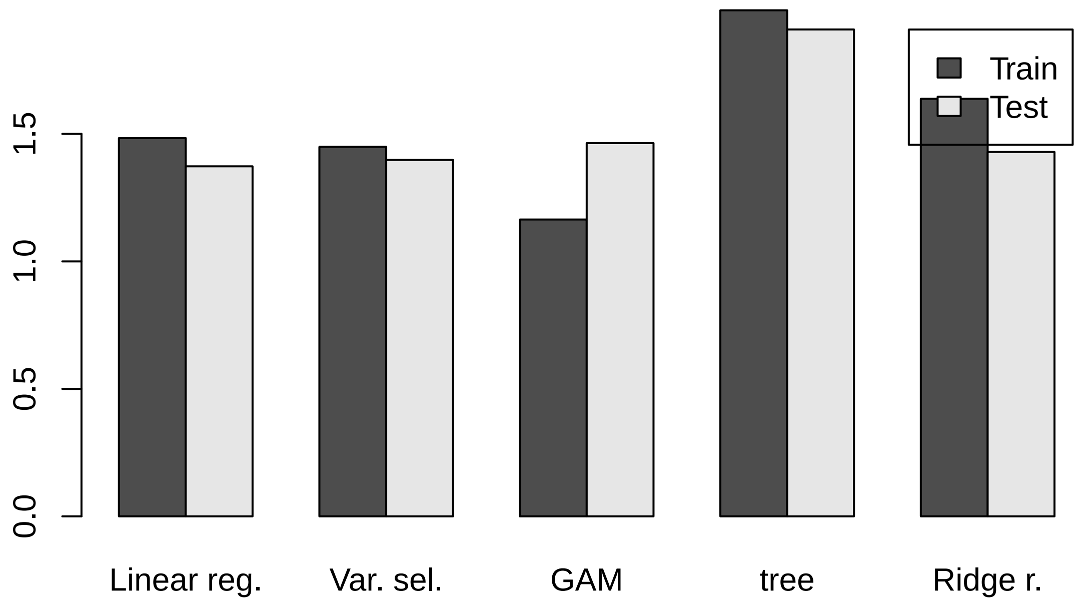


The selected model only uses two splits, as expected.

7. Comparing all the models

```r
predicted_train <- list(  predict(fit_linear, newdata = daphnia_data[train_idx, ])
                  , predict(backwards_BIC, newdata = daphnia_data[train_idx, ])
                  , predict(fit_GAM, newdata = daphnia_data[train_idx, ])
                  , predict(pruned, newdata = daphnia_data[train_idx, ])
                  , predict(cv_lambda,
                           newx = model.matrix(LC50 ~ . - 1, data = daphnia_data[train_idx, ]),
                           newy = daphnia_data$LC50[train_idx])
                  )
predicted_test <- list(  predict(fit_linear, newdata = daphnia_data[test_idx, ])
                       , predict(backwards_BIC, newdata = daphnia_data[test_idx, ])
                       , predict(fit_GAM, newdata = daphnia_data[test_idx, ])
                       , predict(pruned, newdata = daphnia_data[test_idx, ])
                       , predict(cv_lambda,
                                newx = model.matrix(LC50 ~ . - 1, data = daphnia_data[test_idx, ]),
                                newy = daphnia_data$LC50[test_idx])
)
RSS_train <- NULL
RSS_test <- NULL
for (i in 1:5) {
  RSS_test[i] <-  mean( ( predicted_test[[i]]  - daphnia_data$LC50[test_idx]  )^2 )
  RSS_train[i] <- mean( ( predicted_train[[i]] - daphnia_data$LC50[train_idx] )^2 )
}
```

```r
# plot(meanResidents, axes=FALSE, xlab="dorms")
models <- c(  "Linear reg."
            , "Var. sel."
            , "GAM"
            , "tree"
            , "Ridge r."
            )
barplot(rbind(RSS_train, RSS_test), beside=T, names.arg = models,
        legend.text = c("Train", "Test"),
        )
```



We see that all the models except the regression tree perform very similarly on the test set. The generalized linear model performs better on the training data, but performs worse on the test set, indicating that it has overfit the data. The regression tree performs the worst on both sets, indicating that a linear model adopted by the other models will model the underlying mechanism well.

Problem 2. Classification

Unfortunately I ran out of time to do this problem properly. I will go over it more thoroughly later.

```
# Clear the R environment
rm(list=ls())
library(mlbench)
data(PimaIndiansDiabetes)
```

Split data

We split the data set into a trianing and test set. The method `createDataPartition` preserves the overall class distribution of the predictor.

```
library(caret)
trainIndex <- createDataPartition(PimaIndiansDiabetes$diabetes, p = 2/3,
                                  list = FALSE,
                                  times = 1)

# Check if class proportions are equal
a <- summary(PimaIndiansDiabetes$diabetes)
a[1]/a[2]
```

```
##      neg
## 1.865672
```

```
a <- summary(PimaIndiansDiabetes$diabetes[trainIndex])
a[1]/a[2]
```

```
##      neg
## 1.865922
```

```r
a <- summary(PimaIndiansDiabetes$diabetes[-trainIndex])
a[1]/a[2]
```

```
##      neg
## 1.865169
```

1. k-neares neighbours

We find the number of neighbours using 5-fold cross validation, using the caret package for convenience.

```r
set.seed(432)
fitControl <- trainControl(method = "repeatedcv",
                           number = 5, # 5 folds
                            repeats = 10 # repeated ten times
                           )
knn_fit_5f <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'knn',
                 trControl = fitControl,
                 preProcess = c("center", "scale"),
                 tuneLength = 20
                 )
knn_fit_5f
```

```
## k-Nearest Neighbors
##
## 513 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 410, 411, 410, 411, 410, 410, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7372091  0.4006693
##    7  0.7362153  0.3933669
##    9  0.7315398  0.3802693
##   11  0.7385282  0.3915649
##   13  0.7379457  0.3873118
##   15  0.7391242  0.3841498
##   17  0.7447895  0.3940715
##   19  0.7424671  0.3869104
##   21  0.7455757  0.3928811
##   23  0.7457929  0.3908493
##   25  0.7426842  0.3799029
##   27  0.7467675  0.3879950
##   29  0.7453969  0.3817833
##   31  0.7473596  0.3863507
##   33  0.7446279  0.3797701
##   35  0.7403368  0.3652744
##   37  0.7430743  0.3717202
##   39  0.7401427  0.3615872
##   41  0.7430706  0.3663321
##   43  0.7409175  0.3588486
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 31.
```

```r
set.seed(432)
fitControl <- trainControl(method = "LOOCV")
knn_fit_loo <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'knn',
                 trControl = fitControl,
                 preProcess = c("center", "scale"),
                 tuneLength = 20
                 )
knn_fit_loo
```

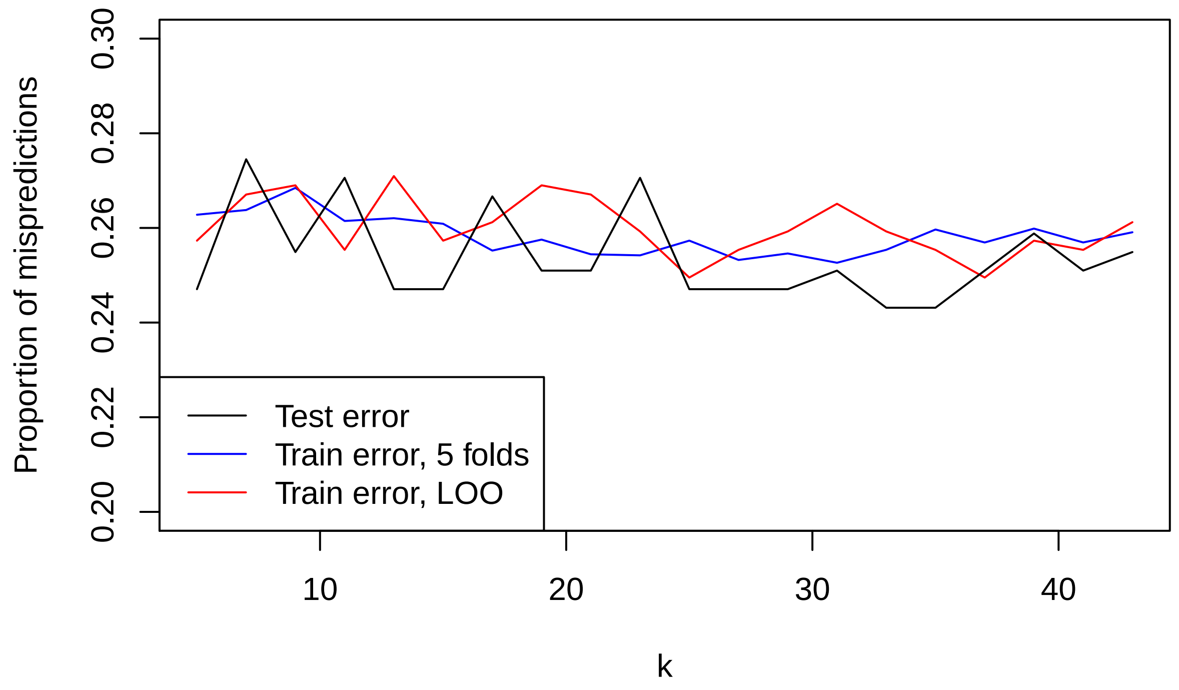
```
## k-Nearest Neighbors
##
## 513 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 512, 512, 512, 512, 512, 512, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7426901  0.4154956
##    7  0.7329435  0.3909106
##    9  0.7309942  0.3839822
##   11  0.7446394  0.4096559
##   13  0.7290448  0.3667060
##   15  0.7426901  0.3994253
##   17  0.7387914  0.3835462
##   19  0.7309942  0.3580057
##   21  0.7329435  0.3617607
##   23  0.7407407  0.3821348
##   25  0.7504873  0.4095070
##   27  0.7446394  0.3948238
##   29  0.7407407  0.3803954
##   31  0.7348928  0.3619286
##   33  0.7407407  0.3715494
##   35  0.7446394  0.3809998
##   37  0.7504873  0.3960376
##   39  0.7426901  0.3789346
##   41  0.7446394  0.3792272
##   43  0.7387914  0.3622718
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 37.
```

```r
# Noramlize test and train data based on the training data, with the help of the caret package
normalized_data <- predict(preProcess(PimaIndiansDiabetes[trainIndex, ],
                                      method = c("center", "scale")),
                           newdata = PimaIndiansDiabetes)
```

```r
predicted_knn <- function(k) {
  # Calculates prediction on the test set of a kNN-model
  # trained on the test set using k neighbours
  class::knn(train = normalized_data[trainIndex, -9],
             test = normalized_data[-trainIndex, -9],
             cl = normalized_data$diabetes[trainIndex],
             k = k
             )
}

k_vals <- knn_fit_5f$results[["k"]]

test_error <- NULL
for (i in 1:length(k_vals)) {
  test_error[i] <- 1 - mean(predicted_knn(k=k_vals[i])
                             == PimaIndiansDiabetes$diabetes[-trainIndex])
}
```

We inspect the test error, using proportion of mispredictions as a measure.

```r
plot(k_vals, 1 - knn_fit_5f$results[["Accuracy"]], ylim = c(0.2, 0.3),
     col = "blue", type = "l",
     xlab = "k", ylab = "Proportion of mispredictions")
lines(k_vals, 1 - knn_fit_loo$results[["Accuracy"]], col = "red")
lines(k_vals, test_error, col = "black")
legend("bottomleft", legend = c("Test error", "Train error, 5 folds", "Train error, LOO"),
       lty = 1,
       col = c("black", "blue", "red"))
```



We see that leave-one-out CV has higher variance than 5-fold CV, as expected. Both seem to aim for the same error estimate, but bots seem to underestimate the true error in the data.

The test error seem to be minimized for k = 12, a value far below the chosen k values, that would fit the model more closely to the data. But this minimizing value seem to be the product of random variation, and I would expect the "one standard deviation" rule to imply a far higher value for k, seeng as the error does not change much for k up to 40.

2. GAM for classification

```
set.seed(432)
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 2,
)
gam_fit <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'gamboost',
                 # trControl = fitControl,
                 preProcess = c("center", "scale")
                 # , tuneLength = 10
)
gam_fit
```

I was not able to figure this method out properly.

3. Classification trees

Bagging tree

```
library(ipred)
set.seed(432)
fit_tree_bagging <- bagging(diabetes ~ ., data = normalized_data[trainIndex, ], coob = T)

fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 3)
tree_fit <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'rpart',
                 trControl = fitControl,
                 )
```

```
pruned_bagging <- prune(tree = fit_tree_bagging, cp = tree_fit$bestTune[[1]])
print(pruned_bagging)
```

```
##
## Bagging classification trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = diabetes ~ ., data = normalized_data[trainIndex,
##     ], coob = T)
##
## Out-of-bag estimate of misclassification error:  0.2593
```

Random forest

```
set.seed(432)
random_forest <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'cforest',
                 trControl = fitControl,
                 preProcess = c("center", "scale")
                 # , tuneLength = 10
)
random_forest
```

```
## Conditional Inference Random Forest
##
## 513 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 462, 461, 461, 462, 462, 462, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7660246  0.4467666
##   5     0.7706647  0.4760419
##   8     0.7673585  0.4711747
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

Neural network

```
set.seed(432)
nnet <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'nnet',
                 trControl = fitControl,
                 preProcess = c("center", "scale")
                 # , tuneLength = 10
)
```

```
nnet
```

```
## Neural Network
##
## 513 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 462, 461, 461, 462, 462, 462, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.7503967  0.4581427
##   1     1e-04  0.7465008  0.4416737
##   1     1e-01  0.7543318  0.4545871
##   3     0e+00  0.7342227  0.4216123
##   3     1e-04  0.7393645  0.4147406
##   3     1e-01  0.7570382  0.4523895
##   5     0e+00  0.7198180  0.3736577
##   5     1e-04  0.7225229  0.3880984
##   5     1e-01  0.7452705  0.4296136
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

Boosted tree

```
set.seed(432)
ada_fit <- train(diabetes ~ ., data = PimaIndiansDiabetes[trainIndex, ],
                 method = 'ada',
                 trControl = fitControl,
                 preProcess = c("center", "scale")
                 # , tuneLength = 10
)
ada_fit
```

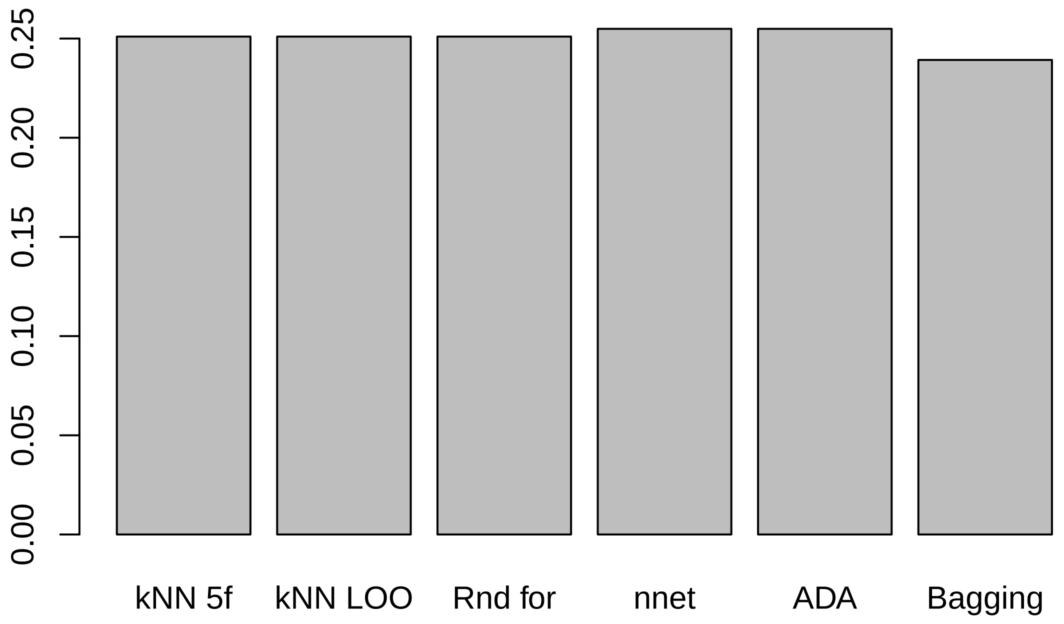
```
## Boosted Classification Trees
##
## 513 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 462, 461, 461, 462, 462, 462, ...
## Resampling results across tuning parameters:
##
##   maxdepth  iter  Accuracy   Kappa
##   1          50   0.7621669  0.4442987
##   1         100   0.7699734  0.4669529
##   1         150   0.7706772  0.4681411
##   2          50   0.7621533  0.4521226
##   2         100   0.7647305  0.4600412
##   2         150   0.7608592  0.4537007
##   3          50   0.7706893  0.4798451
##   3         100   0.7641393  0.4647329
##   3         150   0.7628205  0.4661858
##
## Tuning parameter 'nu' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were iter = 50, maxdepth = 3 and nu = 0.1.
```

4. Choice of method

```
predicted_test <- lapply(
  list(knn_fit_5f,
       knn_fit_loo,
       gam_fit,
       random_forest,
       nnet,
       ada_fit
  ),
  function (x) predict(x,
                       newdata = PimaIndiansDiabetes[-trainIndex, ])
)
```

```
## Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
## args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
## Linear extrapolation used.

## Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
## args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
## Linear extrapolation used.

## Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
## args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
## Linear extrapolation used.
```

```
predicted_test[[7]] <- predict( pruned_bagging, newdata = normalized_data[-trainIndex, ])

errors <- sapply(predicted_test,
                 function(x) 1- mean(x == PimaIndiansDiabetes$diabetes[-trainIndex]))

models <- c(  "kNN 5f"
            , "kNN LOO"
            , "GAM"
            , "Rnd for"
            , "nnet"
            , "ADA"
            , "Bagging"
            )
barplot(errors[-3], beside=F, names.arg = models[-3])
```



I was not able to properly implement the GAM procedure, so we will simply compare the other models. The difference in their performance is so small that the choice is almost arbitrary. Still the neural network performed slightly better than the rest on the test set, and is therefore the recommended model.