Exercise 1 - Fit cubic and natural splines

Load the Boston Housing data:

```
library("MASS")

data(Boston)
```

Set up a cubic spline basis for variable `lstat`:

```
library("splines")

basis <- bs(Boston$lstat, df = 5)
```

(a) Where are the knots located? (you can see this if you print `basis`)

(b) How many basis functions were generated?

(c) Create a plot with the value of each basis function on the y-axis and the `lstat` variable on the x-axis. Hint: First reorder the observations in the dataset `Boston <- Boston[order(Boston$lstat), ]` and then use function `matplot`.

Exercise 1 - Continued

(d) Regress response variable `medv` on the cubic spline just created, e.g.:

```
library("gam")

mod_df5 <- gam(medv ~ bs(lstat, df = 5), data = Boston)
```

Also fit a cubic spline model with 8 df.

Compare the two fitted models, using functions `summary()`, `plot()` (or `plot.Gam()`; also make sure to specify `residuals = TRUE`) and `BIC()`. Which of the two models fits best?

(e) Fit two natural cubic splines, one with 3 df and one with 5 df to the data from Exercise 1. Hint: Use function `ns()` instead of `bs()` to set up the spline basis; use function `gam()` to fit the model as before.

Inspect, interpret and compare the fitted models using `summary()`,`plot()` and `BIC()`. Which model fits best? Is the natural spline an improvement over the cubic spline?

Exercise 2 - Fit a smoothing spline

- Fit smoothing splines to the data from Exercise 1. Again use function `gam` to fit the model, but use function `s()` instead of `bs` or `ns` to fit a smoothing spline. Specify a high (> 100) once, and then a low value (between 3 and 8) for the `df` argument.

- Inspect, interpret and compare the fitted models using `summary()`, `plot()`, `BIC()`. Do the smoothing splines provide an improvement over the natural and cubic splines fitted earlier?

Exercise 3 - Multiple predictor variables, binary outcome

- Download the file MASQ.txt from the GitHub repo. Read it into R:
  `MASQ <- read.table("MASQ.txt", header=TRUE)`

- The dataset ($N = 3597$) contains the following variables:
  - `D_TOT`: the total number of diagnoses present (ignore for today)
  - `D_DEPDYS`: Response variable of interest; an indicator for whether a depressive or dysthymic disorder is present, or not. Make sure it is coded as a factor.
  - `leeftijd` (age)
  - `geslacht` (gender; make sure it is coded as a factor)
  - Subscale scores from the Mood and Anxiety Symptom Questionnaire:
    * `AD`: Anhedonic Depression (22 item scale score)
    * `AA`: Anxious Arousal items (17 item scale score)
    * `GDD`: General Distress Depression (12 item scale score)
    * `GDA`: General Distress Anxiety (11 item scale score)

  * GDM: General Distress Mixed (15 item scale score)

- Select 80% training, 20% test observations, e.g.:

```
set.seed(1)
train <- sample(1:nrow(MASQ), size = nrow(MASQ)*.8)
```

- Fit a GAM on the training observations, to predict D_DEPDYS using all other variables (exclude D_TOT, it would make the prediction problem too easy). Use function gam from package mgcv:

```
detach("package:gam", unload=TRUE)
library("mgcv")
GAM <- gam(D_DEPDYS   s(AD) + s(AA) + s(GDD) + s(GDA) +
s(GDM), data = MASQ[train, ])
```

- Use the summary and plot functions to evaluate the fitted model and curves. Which predictor variables seem to be most important? Which variables' effects deviate from a linear effect?

Exercise 4 - Fit SVMs to MASQ data

Fit SVMs on the training observations of the MASQ data and compare performance with the GAMs using the test observations.

- Use functions `tune()` and `svm()` from package `e1071`.

- Predict `D_DEPDYS` using all other variables, except `D_TOT` as predictors. Probably easiest from now on to simply remove that column from the dataset:

```
MASQ <- MASQ[ , -9]
```

- Fit a support vector classifier (i.e., SVM with linear kernel). First tune the cost parameter using function `tune()`:

```
cost <- c(.001, .01, .1, 1, 5, 10, 100)
tune.out <- tune(svm, D_DEPDYS ~ ., data = MASQ[train, ],
kernel = "linear", ranges = list(cost = cost))
```

- With the optimal budget (cost), now fit the SVM:

```
svmfit <- svm(D_DEPDYS ~ ., data = MASQ[train,],
kernel = "linear", cost = ...)
```

- Compute the misclassification rate on the test observations using function `predict`.

- Now fit a support vector machine with radial basis kernel. First obtain optimal values for the gamma and cost parameters:

```
gamma <- c(0.5, 1, 2, 3, 4))
tune.out <- tune(svm, D_DEPDYS ~ ., data = MASQ[train, ],
kernel = "radial", ranges = list(
cost = cost, gamma = gamma))
```

- Now fit the SVM with optimal cost and gamma values (use function `svm` like before and specify `kernel = "radial"`)

- Compute the misclassification rate on the test observations using function `predict`.

Exercise 5 - Fit conditional inference tree to MASQ data

Fit a `ctree` to the `MASQ` training data using the `partykit` library:

```
library("partykit")
```

```
ct <- ctree(D_DEPDYS ~ ., data = MASQ[train, ])
```

Compute predicted probabilities for the test observations, and evaluate the misclassification rate and brier score.

Exercise 6 - Fit random forest to MASQ data

Fit a bagged ensemble and a random forest to the `D_DEPDYS` outcome. Use the `randomForest()` function (from package with the same name):

- Make sure to set the random seed, first. Make sure to exclude `D_TOT` as a possible predictor. Make sure to specify `importance = TRUE` in the call to `randomForest()`.
- For the bagged ensemble, set argument `mtry` to p, for the random forest, set `mtry` to $\sqrt{p}$.
- For both (bagged and RF) ensembles, use `plot()` to see how the OOB error decreases as a function of the number of trees in the ensemble.
- For both ensembles, use functions `varImpPlot()` and `importance()` to inspect variable importances.
- Use function `partialPlot()` to inspect the effect of each predictor variable. Check the documentation of this function to see what arguments `pred.data` and `x.var` do.

- Compute predicted probabilities using function `predict`. Compare performance with the models fitted earlier.

Exercise 7 - Fit boosted tree ensemble to MASQ data

- Use function `gbm()` (from package of same name).
- Make sure to set the random seed, first.
- Type `?gbm` to inspect the possible and default settings. Check out the meaning of arguments `distribution`, `n.trees`, `interaction.depth`, `bag.fraction` and `shrinkage`.
- Fit 1,000 trees, specify a learning rate (shrinkage) of .01 and an interaction depth of 4, and specify a Bernoulli distribution. Make sure you code the response variable as 0-1.
- Use function `predict()` to evaluate predictive accuracy on test observations.
- Use function `plot()`, `summary()` and `gbm.perf()` to inspect and interpret the result. Compute predicted probabilities using function `predict`. Compare performance with the models fitted earlier.

Exercise 7 - Tuning boosting parameters

Optimize the parameter settings for the boosted ensemble using the training data, and function `train()` from package **`caret`** (short for classification and regression training):

- Specify a grid of tuning parameter values like:

```
grid <- expand.grid(
shrinkage = c(.1, .01, .001),
n.trees = c(10, 100, 1000),
interaction.depth = 1:4,
n.minobsinnode = 10)
grid
```

- Remember to set the random seed before the analysis.

Exercise 7 - Tuning boosting parameters

- Perform the cross-validation like:

```
gbmFit <- train(D DEPDYS ~ .,
data = MASQ[-train, ], tuneGrid = grid,
distribution = "bernoulli", method = "gbm",
verbose = FALSE)
gbmFit
gbmFit$bestTune
```

- Note that `train()` wants a binary factor, not a 0-1 coded response variable.

- Use the best-performing parameter values to fit a new boosted ensemble. Compare the performance on the test data with the earlier boosted ensemble.

- Note that you could optimize the parameters of function `randomForest()` in a similar fashion. Check out `?train` and the `method` and `metric` arguments.