Dataset

- Download the file MASQ.Rda from the GitHub repo.

- Read it into R: `load("MASQ.Rda")`

- The dataset ($N = 3,597$) contains 8 variables:
  - `D_DEPDYS`: Response variable. An indicator for whether a depressive or dysthymic disorder is present, or not. Verify that it is coded as a factor.
  - `leeftijd`: Age in years.
  - `geslacht`: Gender, make sure it is coded as a factor.
  - Subscale scores from the Mood and Anxiety Symptom Questionnaire:
    * `AD`: Anhedonic Depression, a 22-item total score.
    * `AA`: Anxious Arousal, a 17-item total score.
    * `GDD`: General Distress Depression, a 12-item total score.
    * `GDA`: General Distress Anxiety, an 11-item total score.
    * `GDM`: General Distress Mixed, a 15-item total score.

- Select 80% of the data for training, and 20% for testing, e.g.:

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

- Then you can select the training and test datasets as follows:

```
MASQ[train, ] ## training set
MASQ[-train, ] ## test set
```

Exercise 1: Fit a smoothing spline

- Use function `gam()` from package `mgcv` to fit a smoothing spline of `AD` to predict `D_DEPDYS`:

- You only need to specify the `formula`, `data` and `method` arguments.

- In the model `formula`, specify a cubic spline basis for the smoothing spline: `D_DEPDYS ~ s(AD, bs = "cr")`

- Specify `method = "REML"` to use restricted maximum likelihood estimation.

- Apply the `summary` and `plot` methods on the fitted model to inspect the resulting model. Describe the effect of the AD scale score on the probability of having a depressive disorder.

Exercise 2: Fit a GAM with multiple predictor variables

- Use function `gam` from package `mgcv`.

- Fit a GAM to the training observations, using all predictors in the dataset. Specify a smoothing spline for each numeric predictors, using funciton `s()`. Factors can be included in the model formula as usual.

- You can simply use the default settings of function `s()`, as these generally tend to work well.

- Use the `summary` and `plot` methods to inspect and interpret the fitted model. Which predictor variables seem to be most important? Which variables' effects deviate from a linear effect?

- Use the `predict` method to compute predicted probabilities for the test observations. In order to obtain probabilities, specify `type = "response"` (by default, predictions are returned on the scale of the linear predictor for binomial outcomes).

- Compute the Brier score (squared error loss on predicted probabilities) for the test observations.

- Compute predicted class labels for the test observations, by assigning observations with $p > .5$ to the target class.

- Compute the mis-classification rate for the test observations.

Exercise 3: Fit SVMs

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

- Use functions `tune()` and `svm()` from package `e1071`.
- Predict `D_DEPDYS` using all other variables.
- 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 4: Fit a conditional inference tree

- Fit a conditional inference tree to the `MASQ` training data using the `partykit` library:
  ```
  ct <- ctree(D_DEPDYS ~ ., data = MASQ[train, ])
  ```

- Inspect the resulting tree by plotting it. Which are the important predictors and how do they affect the response?

- Compute predicted probabilities for the test observations using `predict`. To obtain probabilities, specify `type = "prob"`. Note that you obtain a matrix of predictions, use only the second column as it gives the probability for belonging to the target class (first column gives probability for belonging to the reference class).

- Compute the misclassification rate and Brier score.

Exercise 5: Fit a bagged ensemble and random forest

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 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, apply `plot()` to the fitted ensemble 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 6: Fit a gradient-boosting ensemble

- 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)
```

- Inspect `grid` by printing it.

- Remember to set the random seed before the analysis.

- Perform the cross-validation as follows:

```
gbmFit <- train(D_DEPDYS ~ .,
                 data = MASQ[-train, ], tuneGrid = grid,
                distribution = "bernoulli", method = "gbm")
```

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

- Inspect the results using:

```
gbmFit
gbmFit$bestTune
plot(gbmFit)
```

- 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.