Compulsory exercise 2: Group 64

TMA4268 Statistical Learning V2019

Martinussen, Jakob Gerhard & Wilson, Alm

12 April, 2019

Problem 1: Regression [6 points]

We start by importing the data set used in this problem.

```
all <- dget("https://www.math.ntnu.no/emner/TMA4268/2019v/data/diamond.dd")
dtrain <- all$dtrain
dtest <- all$dtest
```

Q1: Would you choose `price` or `logprice` as response variable? Justify your choice. Next, plot your choice of response pairwise with `carat`, `logcarat`, `color`, `clarity` and `cut`. Comment.

We choose `logprice` as the response variable, since we postulate that diamond qualities have a *multiplicative* effect on the price, not *additive*. A logarithmic response will result in such an interpretation of the regression coefficients. We now create the plots as instructed.

```
library(tidyverse)
library(ggplot2)
library(dplyr)
library(tibble)
library(tidyr)

# Combine both training and testing set for plotting
dall <- bind_rows(dtrain, dtest)

# Plot categorical covariates
dall %>%
  gather(key, value, c("cut", "color", "clarity")) %>%
  ggplot(aes(y = logprice)) +
  geom_boxplot(aes(x = value, col = key), alpha = 0.3) +
  facet_wrap(.~key, scales = "free", nrow=3)
```

```
# Plot continuous covariates
dall %>%
  gather(key, value, c("carat", "logcarat")) %>%
  ggplot(aes(y = logprice)) +
  geom_point(aes(x = value), alpha = 0.5) +
  facet_grid(.~key, scales = "free")
```

From figure 2 further down we observe that `logcarat` *seems* to be the covariate that has the most linear correlation with `logprice`, while `carat` does not. It is difficult to see any clear trends in the plots of the categorical covariates `clarity`, `color`, and `cut` in figure 1, also further down.

Q2: What is the predicted price of a diamond weighting 1 carat. Use the closest 20% of the observations.

We now use the local regression model $\texttt{logprice} = \beta_0 + \beta_1 \texttt{carat} + \beta_2 \texttt{carat}^2$ weighted by the tricube kernel K_{i0}.

```
model1 <- loess(logprice ~ carat + carat^2, data = dtrain, span = 0.2)
```

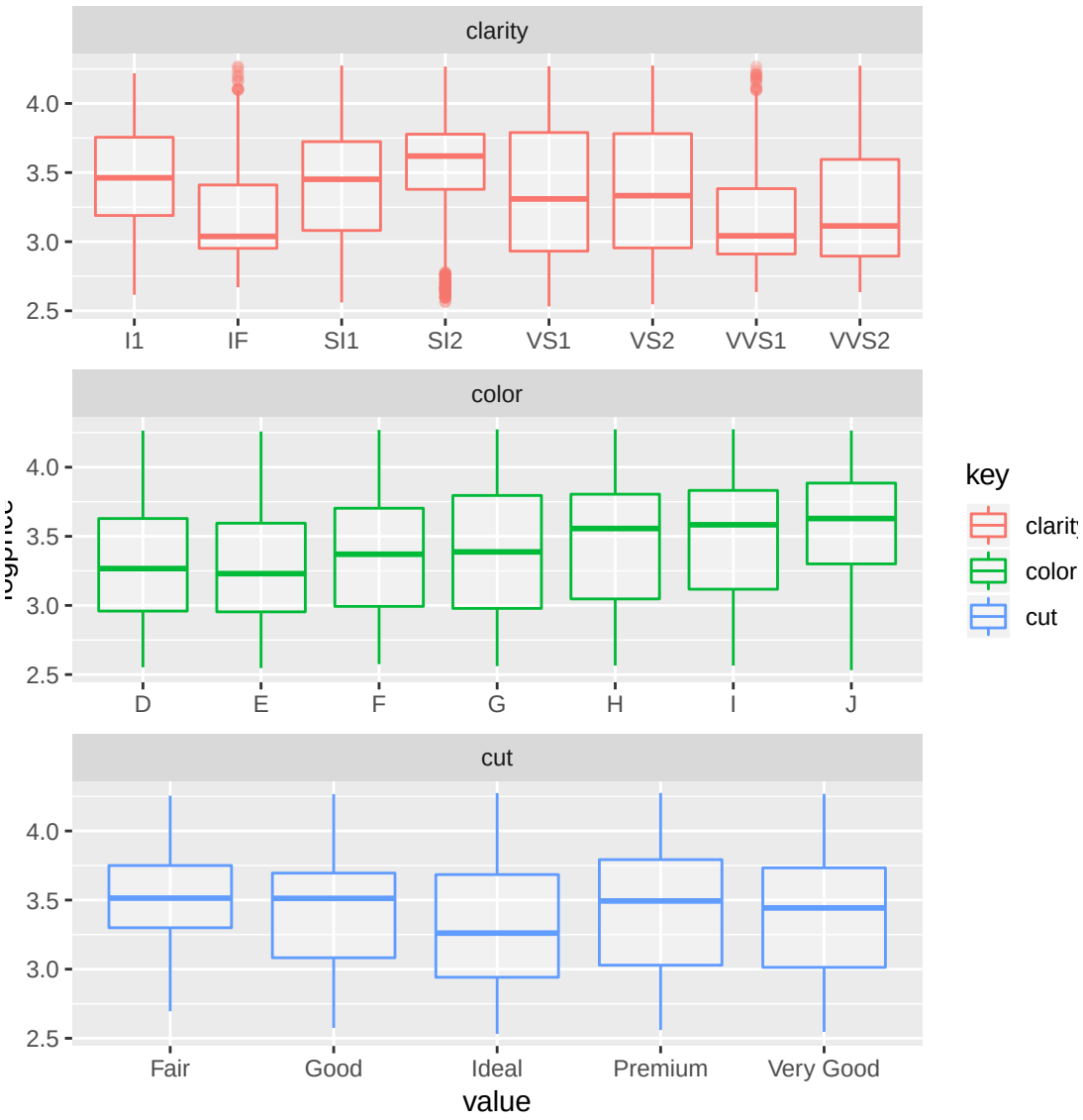


Figure 1: Boxplot for categorical covariates: clarity, color, and cut. Plotted against logprice on the y-axis.

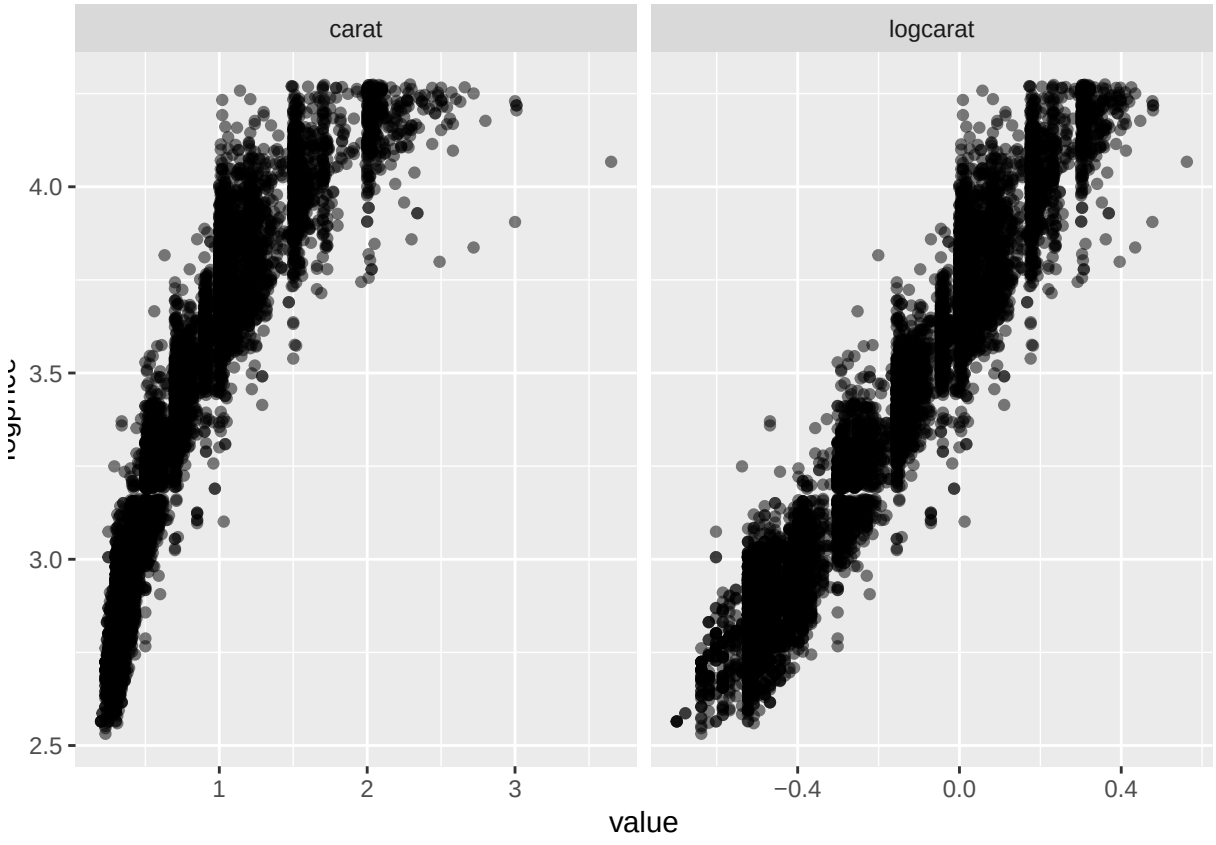


Figure 2: Scatterplot of carat and logcarat on the x-axis plotted against logprice on the y-axis.

We make a price prediction for a 1 carat diamond, using the closest 20% of the observations:

```
new_observation <- list(carat = 1)
price_prediction <- 10 ^ predict(model1, new_observation)
sprintf("$%.2f USD", price_prediction)
```

```
## [1] "$5100.67 USD"
```

So the prediction is that a 1 carat diamond will cost $\simeq$ \$5100 USD.

Q3: What choice of β_1, β_2 and K_{i0} would result in KNN-regression?

Let $\mathcal{N}_0$ be the index set for the K nearest (euclidean) neighbours of the predictor variable $\boldsymbol{x}_0$. Now assign the following values

$$\begin{aligned} \beta_1 &\leftarrow 0 \\ \beta_2 &\leftarrow 0 \\ K_{i0} &\leftarrow I(i \in \mathcal{N}_0). \end{aligned}$$

This yields the following expression to be minimized by the local regression

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i - \beta_2 x_i^2)^2 = \sum_{i \in \mathcal{N}_0} (y_i - \beta_0)^2,$$

The minimizer for this expression is $\beta_0 = \frac{1}{K}\sum_{i \in \mathcal{N}_0} y_i = \bar{y}_{\mathcal{N}_0}$. Here we have denoted $\bar{y}_{\mathcal{N}_0}$ as the mean of the responses of the K nearest neighbours to $\boldsymbol{x}_0$. This results in the following predictor

$$\hat{f}(\boldsymbol{x}_0) = \bar{y}_{\mathcal{N}_0},$$

which is in fact the predictor of a KNN-regression.

Q4: Describe how you can perform model selection in regression with AIC as criterion.

The Akaike Information Criterion (AIC) is defined as

$$\text{AIC} := -2 \cdot \ell(\hat{\beta}_M, \tilde{\sigma}^2) + 2(|M| + 1),$$

where $\hat{\beta}_M$ is the estimated coefficient vector for a model with M regression parameters, and $\ell(\cdot)$ is the logarithm of the maximum likelihood for that model. How you calculate the maximum likelihood depends on what type of regression you perform. We use $|M| + 1$, because we have to add the estimated variance of the error terms ϵ_i. For linear regression, AIC becomes:

$$\text{AIC} = \frac{1}{n\hat{\sigma}^2}(\text{RSS} + 2d\hat{\sigma}^2)$$

When we do model selection, i.e. picking between different candidate models, we can pick the model with the *lowest* value for AIC.

Q5: What are the main differences between using AIC for model selection and using cross-validation (with mean squared test error MSE)?

Both methods are intended to evaluate the predictive power of a model while still accounting for the fact that the model might be overfitted (and punishing accordingly).

AIC achieves this by using RSS as the measure of fit while *adding* a complexity term in order to adjust for overfitting. The entirety of the data is used for the measure.

Cross-validation with mean squared test error MSE does not require such an additional complexity term. It rather splits the data into training and validation sets, trains the model on the training set, and calculates the MSE with the validation test. That way it implicitly guards against overfitting since the validation set is not part of the model training procedure.

Here are some advantages/disadvantages:

- CV requires you to split your data set into *training* and *testing* subsets. This results in smaller sample sizes for fitting the models, and can result in an overestimation of errors.
- CV test MSE can have high variability for smaller sample sizes, since it is dependent on which observations are included/excluded.

Q6: See the code below for performing model selection with `bestglm()` based on AIC. What kind of contrast is used to represent `cut`, `color` and `clarity`? Write down the final best model and explain what you can interpret from the model.

```
library(bestglm)
ds <- as.data.frame(
  within(
    dtrain,
    {
      y = logprice    # setting response
      logprice = NULL # not include as covariate
      price = NULL    # not include as covariate
      carat = NULL    # not include as covariate
    }
  )
)

fit <- bestglm(Xy=ds, IC="AIC")$BestModel
summary(fit)
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.31283 -0.03717  0.00033  0.03564  0.58991
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.985969   0.042993  69.452  < 2e-16 ***
## logcarat      1.580675   0.029042  54.428  < 2e-16 ***
## cutGood       0.028558   0.005747   4.970 6.93e-07 ***
## cutVery Good  0.040480   0.005285   7.660 2.22e-14 ***
## cutPremium    0.042984   0.005295   8.118 5.90e-16 ***
## cutIdeal      0.054829   0.005223  10.498  < 2e-16 ***
```

```
## colorE       -0.021787   0.003031  -7.189 7.51e-13 ***
## colorF       -0.038892   0.003107 -12.519  < 2e-16 ***
## colorG       -0.067999   0.003024 -22.488  < 2e-16 ***
## colorH       -0.112815   0.003235 -34.877  < 2e-16 ***
## colorI       -0.165072   0.003686 -44.783  < 2e-16 ***
## colorJ       -0.223651   0.004543 -49.233  < 2e-16 ***
## claritySI2    0.174988   0.007933  22.060  < 2e-16 ***
## claritySI1    0.251631   0.007876  31.950  < 2e-16 ***
## clarityVS2    0.315556   0.007924  39.824  < 2e-16 ***
## clarityVS1    0.346409   0.008051  43.027  < 2e-16 ***
## clarityVVS2   0.402528   0.008233  48.894  < 2e-16 ***
## clarityVVS1   0.433326   0.008549  50.684  < 2e-16 ***
## clarityIF     0.473681   0.009154  51.745  < 2e-16 ***
## xx            0.069331   0.006610  10.489  < 2e-16 ***
## —
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05978 on 4980 degrees of freedom
## Multiple R-squared:  0.9815, Adjusted R-squared:  0.9814
## F-statistic: 1.39e+04 on 19 and 4980 DF,  p-value: < 2.2e-16
```

From the summary we can see that `cut`, `color`, and `clarity` are *dummy coded* with respectively `cutFair`, `colorD`, and `clarityI1` as reference categories.

The final best model is `logprice ~ 1 + logcarat cut + color + clarity + xx`, or in mathematical form

$$
\begin{aligned}
\widehat{\log(\text{price})}_i = &\hat{\beta}_0 + \hat{\beta}_{\text{logcarat}} x_{i,\text{logcarat}} \\
&+ \hat{\beta}_{\text{cutGood}} x_{i,\text{cutGood}} + \hat{\beta}_{\text{cutVeryGood}} x_{i,\text{cutVeryGood}} + \hat{\beta}_{\text{cutPremium}} x_{i,\text{cutPremium}} + \hat{\beta}_{\text{cutIdeal}} x_{i,\text{cutIdeal}} \\
&+ \hat{\beta}_{\text{colorE}} x_{i,\text{colorE}} + \hat{\beta}_{\text{colorF}} x_{i,\text{colorF}} + \hat{\beta}_{\text{colorG}} x_{i,\text{colorG}} \\
&+ \hat{\beta}_{\text{colorH}} x_{i,\text{colorH}} + \hat{\beta}_{\text{colorI}} x_{i,\text{colorI}} + \hat{\beta}_{\text{colorJ}} x_{i,\text{colorJ}} \\
&+ \hat{\beta}_{\text{claritySI2}} x_{i,\text{claritySI2}} + \hat{\beta}_{\text{claritySI1}} x_{i,\text{claritySI1}} + \hat{\beta}_{\text{clarityVS2}} x_{i,\text{clarityVS2}} + \hat{\beta}_{\text{clarityVS1}} x_{i,\text{clarityVS1}} \\
&+ \hat{\beta}_{\text{clarityVVS2}} x_{i,\text{clarityVVS2}} + \hat{\beta}_{\text{clarityVVS1}} x_{i,\text{clarityVVS1}} + \hat{\beta}_{\text{clarityIF}} x_{i,\text{clarityIF}} + \hat{\beta}_{\text{xx}} x_{i,\text{xx}}
\end{aligned}
$$

A `price` prediction for a covariate vector $\boldsymbol{x}_0$ is constructed in the following manner

$$
\widehat{\texttt{logprice}} = \hat{\boldsymbol{\beta}}^T \boldsymbol{x}_0 \implies \widehat{\texttt{price}} = 10^{\hat{\boldsymbol{\beta}}^T \boldsymbol{x}_0} = 10^{\beta_0} \cdot \prod_{i=1}^{19} 10^{\beta_i x_{0,i}}.
$$

Ten to the power of the regression coefficients are therefore of interest.

```
library(kableExtra)
beta_hat <- coefficients(fit)
10^beta_hat %>% kable(col.names = "$10^{\\hat{\\beta}_i}$", format = "pandoc")
```

	$10^{\hat{\beta}_i}$
(Intercept)	968.2091973
logcarat	38.0780598
cutGood	1.0679683
cutVery Good	1.0976911
cutPremium	1.1040377

	$10^{\hat{\beta}_i}$
cutIdeal	1.1345648
colorE	0.9510721
colorF	0.9143407
colorG	0.8550694
colorH	0.7712320
colorI	0.6837986
colorJ	0.5975150
claritySI2	1.4961931
claritySI1	1.7849721
clarityVS2	2.0680260
clarityVS1	2.2202846
clarityVVS2	2.5265526
clarityVVS1	2.7122279
clarityIF	2.9763272
xx	1.1730878

Several things can be interpreted from this. The `(Intercept)` represents a 1 carat diamond, i.e. 0 `logcarat`, with a *fair* cut, *D* color category, 0 mm in length, and *I1* clarity category. Such a diamond is predicted to cost $\simeq$ \$968 USD. Such a diamond clearly does not exist. All other covariates must be interpreted *relative to* this "reference diamond". For instance, since $10^{\hat{\beta}_{\texttt{cutIdeal}}} \approx 1.135$, we would expect a diamond with an *ideal cut* to cost 13.5% more than the "reference diamond", all other properties being equal.

Q7: Calculate and report the MSE of the test set using the best model (on the scale of `logprice`).

```
best_subset_test_mse <- mean((dtest$logprice - predict.lm(fit, dtest))^2)
print(best_subset_test_mse)
```

```
## [1] 0.003600305
```

The test MSE is $\simeq 0.0036$.

Q8: Build a model matrix for the covariates `~logcarat+cut+clarity+color+depth+table+xx+yy+zz-1`. What is the dimension of this matrix?

We build the design matrix by using the `model.matrix()` function.

```
lasso_formula <- ~logcarat+cut+clarity+color+depth+table+xx+yy+zz-1
model_matrix <- model.matrix(
  lasso_formula,
  data = dtrain
)
```

Since we use dummy coding of categorical covariates, each covariate with c categories result in $c-1$ columns in the design matrix. But in this case, we have no intercept, so one of the categorical covariates will have c columns, one for each covariate. We have 3 categorical variables, with 5, 8, and 7 categories. This results in 18 columns, instead of 17 if we had an intercept in our model. In addition, we have 6 continuous covariates. All together this results in 24 columns in the design matrix and n rows, where n equals the number of observations, in this case 5000. To confirm, we can check

```
dim(model_matrix)
```

```
## [1] 5000   24
```

This coincides.

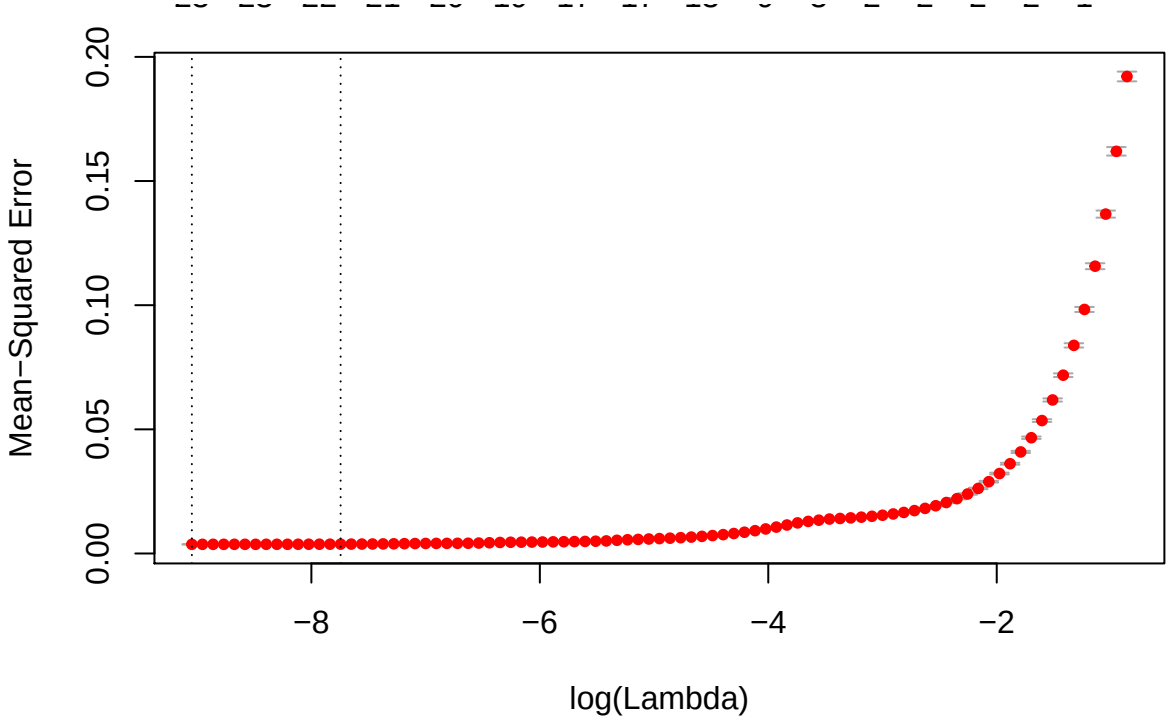
Q9: Fit a lasso regression to the diamond data with `logprice` as the response and the model matrix given in Q8. How did you find the value to be used for the regularization parameter?

We will fit a lasso regression model to the design matrix from **Q8**. In order to choose a suitable value for the tuning parameter λ, we use 10-fold cross validation and the 1 standard deviation rule as explained in exercise 1. We let `glmnet` choose a suitable range of λ values for us.

```
library(glmnet)
set.seed(0)
cv.out <- cv.glmnet(
  x = model_matrix,
  y = dtrain$logprice,
  alpha=1,          # Lasso
  nfolds=10,        # 10-fold CV
  standardize=TRUE  # Standardize coefficients, see: https://think-lab.github.io/d/205/#5
)
```

The cross validation MSE can be plotted against λ.

```
plot(cv.out)
```



The two vertical lines correspond to the λ with the least CV MSE (left) and the λ chosen with the 1SD-rule (right), with the specific values

```
print(cv.out$lambda.min)
print(cv.out$lambda.1se)
```

```
## [1] 0.0001177981
## [1] 0.000433306
```

The chosen regularization parameter is therefore $\lambda \approx 4.3 \cdot 10^{-4}$, which is within one standard deviation of $\lambda_{\min} \approx 1.2 \cdot 10^{-4}$.

The resulting degrees of freedom in this model is:

```
best_index <- which(cv.out$glmnet.fit$lambda == cv.out$lambda.1se)
degrees_of_freedom <- cv.out$glmnet.fit$df[best_index]
print(degrees_of_freedom)
```

```
## [1] 22
```

Which means that two of the regression coefficients are set to zero, and only 22 remain.

Q10: Calculate and report the MSE of the test set (on the scale of `logprice`).

We can now calculate the test MSE.

```
lasso_model <- cv.out$glmnet.fit
chosen_lambda <- cv.out$lambda.1se
xtest <- model.matrix(lasso_formula, data = dtest)
predictions <- predict(lasso_model, s = chosen_lambda, newx = xtest)
lasso_test_mse <- mean((dtest$logprice - predictions) ^ 2)
print(lasso_test_mse)
```

```
## [1] 0.003806364
```

The test MSE is $\approx 3.81 \cdot 10^{-3}$ on the scale of `logprice`.

Q11: A regression tree to model is built using a *greedy* approach. What does that mean? Explain the strategy used for constructing a regression tree.

A *greedy* approach means that the apparently best choice is taken at any one time, not considering the possibility of a suboptimal choice may be taken in order to achieve an *overall* better result further down the road. Once a decision has been made, no backtracking is performed, and the choice is "locked in".

The idea of a regression tree is to take n observation pairs $(\boldsymbol{x}_i, Y_i)$ and separating them into J partitions of the predictor space, $R_1, ..., R_J$. Predictions are made by identifying the region which a predictor falls within and then taking the average response of all the pairs of that region.

A prediction, $\hat{f}_{R_j}$, for $\boldsymbol{x}_0$ belonging to the region R_j, is therefore calculated as

$$\hat{f}_{R_j} = \frac{1}{|R_j|} \sum_{i:x_i \in R_j} y_i,$$

where $|R_j|$ is the number of observed pairs placed within region R_j.

Now a question remains; how to construct these regions? We use a greedy approach called *recursive binary splitting*. Define the parametrized regions

$$R_1(j, s) := \{\boldsymbol{x} \mid x_j < s\},$$
$$R_2(j, s) := \{\boldsymbol{x} \mid x_j \geq s\}.$$

We want to pick the predictor j and split-off point s that minimizes the sum of the residual sum of squares of both sub-partition, i.e.

$$(j, s) = \underset{(j,s)}{\operatorname{argmin}} \left\{ \sum_{i:x_i \in R_1(j,s)} (y_i - \hat{f}_{R_1(j,s)})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{f}_{R_2(j,s)})^2 \right\}$$

We choose (j, s) recursively without backtracking (thus greedy), until some stopping criterion is reached. The stopping criterion can be when the number of members of the resulting region is small enough, or when the reduction in RSS is smaller than some threshold.

Q12: Is a regression tree a suitable method to handle both numerical and categorical covariates? Elaborate.

Regression trees can handle categorical covariates as well, but there is now a change in how we view cut-points. In the numerical case, for a choice of region R_k and a predictor X_j, there exists at most $|R_k| - 1$ cut-points that will result in unique partitions of the region.

For a categorical covariate the number of cut-points depends on the number of levels l in the covariate, and if there is a sensible ordering of them. For example if the covariate is age demographic, then it makes sense to just use cut-points of the same type as in the numerical case. This results in at most $l - 1$ cut-points per region.

However, lacking such order as in the case of e.g. job type, this definition of a cut-point stops making sense. What should be done now is to consider all ways of partitioning the levels into two non-empty subsets R_1 and R_2. This changes the branching criteria from evaluating if $x_i < s$ to checking if $x_i \in L_1$. The number of "cut-points" per region thus becomes, in the worst case,

$$\frac{1}{2}\sum_{i=1}^{l-1}\binom{l}{i} = 2^{l-1} - 1$$

and checking each one of them quickly becomes intractable as l increases. Using regression trees in such cases should therefore be done with care.

Q13: Fit a (full) regression tree to the diamond data with `logprice` as the response (and the same covariates as for c and d), and plot the result. Comment briefly on you findings.

We now fit a full regression tree for the diamond data.

```
library(tree)
diamond_formula <- logprice~logcarat+cut+clarity+color+depth+table+xx+yy+zz-1
full_tree <- tree(
  diamond_formula,
  data = dtrain
)
```

The decision tree can be visualized together with the partition space. We will transform the `logprice` to `price` for easier interpretability.

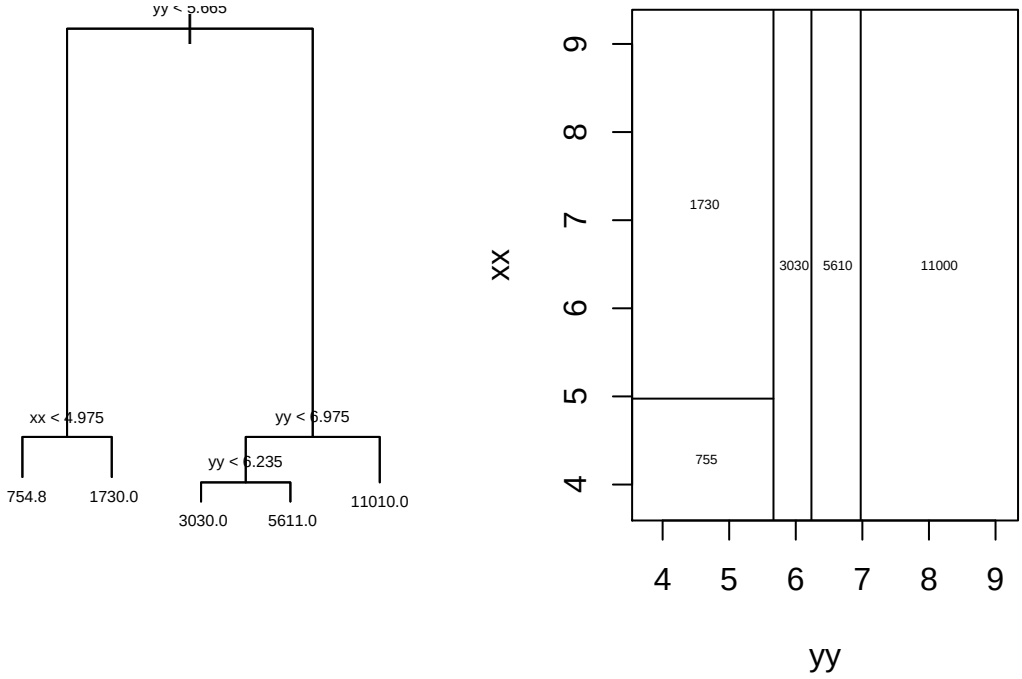
```
# For deep copy of data-structures
library(rlang)

# Plot side-by-side
par(mfcol = c(1, 2))

# Convert fitted values from logprice to price
plot_tree <- duplicate(full_tree, shallow = FALSE)
plot_tree$frame$yval <- 10 ^ plot_tree$frame$yval

# Plot the decision tree together with decision criteria
plot(plot_tree, type = "proportional")
text(plot_tree, pretty = 1, cex = 0.5)

# Plot the partitioning of the xx-yy plane
partition.tree(plot_tree, cex = 0.4)
```



The length of the edges in the tree on the left hand side are proportional to the reduction in RSS. We can see that the first split, `yy` < 5.665, by far reduces the RSS the most. The tree algorithm also decides that *above* this cut-point (to the right in the tree), only `yy` matters, and `xx` is ignored. The greater `xx` and `yy` is, the greater the predicted price. Lastly, it is interesting that solely `xx` and `yy` are chosen, all of the other covariates have been discarded. Using this model as a human is quite easy, as there are only two covariates to consider.

Q14: Calculate and report the MSE of the test set (on the scale of `logprice`).

We can now calculate the test MSE on the scale of `logprice`.

```
tree_predictions <- predict(full_tree, dtest)
tree_test_mse <- mean((tree_predictions - dtest$logprice)^2)
print(tree_test_mse)
```

```
## [1] 0.01715548
```

The test MSE is $\approx 1.72 \cdot 10^{-2}$, a whole order of magnitude greater than the lasso model fitted earlier. It should be noted, though, that this model is *substantially* less complex than the lasso model.

Q15: Explain the motivation behind bagging, and how bagging differs from random forest? What is the role of bootstrapping?

A problem with decision trees is that they often portray a large amount of variance, since small changes in the predictors will often result in very different trees. One solution to this problem is *bootstrap aggregation*, more commonly known as *bagging*.

The idea is to draw a sufficiently large amount, B, bootstrap samples from the training set and fit a tree, $\hat{f}_b(\boldsymbol{x})$, for every single replicate, X_b. The resulting predictor, $\hat{f}_{\text{avg}}(\boldsymbol{x})$ is the *average* of all those trees,

$$\hat{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x}).$$

The motivation is this, assume that we have B i.i.d. observations of the random variable X with mean μ and variance σ^2. The unbiased mean estimator, $\bar{X} = \frac{1}{B}\sum_{b=1}^{B} X_b$, has a variance of

$$\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} X_b\right) = \frac{1}{B^2}\sum_{b=1}^{B}\text{Var}(X_b) = \frac{\sigma^2}{B}$$

The variance of the estimator is thus inversely proportional to the number of independent samples B. This effect can be *simulated* by generating B bootstrap data sets; they are not completely independent but is better than nothing.

Bagging is not the end all solution, however. If there is a really strong predictor in the training set, then this predictor will end up being used in the first split in most of the bootstrap replicated trees. This results in the trees becoming highly correlated, and we lose much of the variance reduction which we expect from the averaging of the trees. A *random forest* aims to solve this by randomly removing a fraction of the covariates in each split, instead of using *all* the covariates as we do in *bagging*. This is intended to decorrelate the trees.

Q16: What are the parameter(s) to be set in random forest, and what are the rules for setting these?

There are two parameters of interest when creating a *random forest*. The first is the number of trees, B. This is *not* a tuning parameter, however, and the larger the better (ignoring computational cost). Pick a value that is sufficiently large while being computationally feasible.

The other parameter is m, the number of covariates to be considered in each split. First off, we require $m < p$, of course. For very correlated predictors, m is chosen to be small. A good *a priori* rule-of-thumb is to choose $m \approx \sqrt{p}$ for classification and $m \approx p/3$ for regression. This has been derived from simulations.

Q17: Boosting is a popular method. What is the main difference between random forest and boosting?

A random forest is created by averaging B trees fitted in parallel with the use of bootstrapping. The trees are built independently of each other.

Boosting is different in that it is a *sequential* method. Trees are built sequentially, each fitted on the residuals of an accumulated model of the previous trees. For a more rigorous explanation, see the Module 8 part of **Q29**.

Q18: Fit a random forest to the diamond data with `logprice` as the response (and the same covariates as before). Comment on your choice of parameter (as decribed in Q16).

We fit a random forest with the same formula.

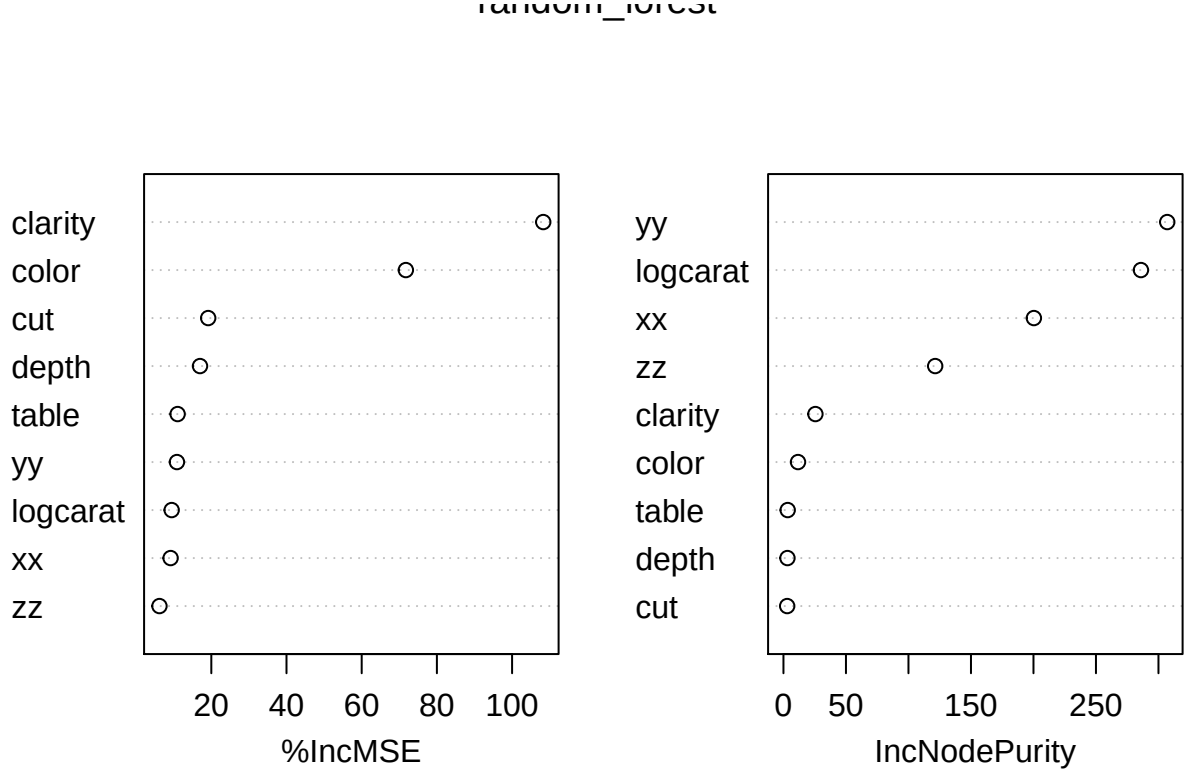
```
set.seed(0)
random_forest <- randomForest(
  diamond_formula,
  data = dtrain,
  ntree = 100,
  mtry = 9 / 3,
  importance = TRUE
)
```

We have here chosen `mtry = 9 / 3 = 3` since we have a continuous response and this is a regression tree ($p = 9$). With other words, we are using the rule-of-thumb $m \approx p/3$.

Q19: Make a variable importance plot and comment on the plot. Calculate and report the MSE of the test set (on the scale of `logprice`).

In order to interpret this *ensemble method* we can make a variable importance plot.

```
varImpPlot(random_forest)
```



The two variable importance metrics (percentage increase in MSE and increase in node purity) give quite different results. This is often more common for models with categorical covariates with many levels. Not much can be said, as they don't share any opinion of which variables are the most important.

We can now calculate the MSE for the test set on the scale of `logprice`.

```
random_forest_predictions <- predict(random_forest, dtest)
random_forest_test_mse <- mean((random_forest_predictions - dtest$logprice)^2)
print(random_forest_test_mse)
```

```
## [1] 0.002811009
```

We have a test MSE of $\approx 2.81 \cdot 10^{-3}$.

Q20: Finally, compare the results from c (subset selection), d (lasso), e (tree) and f (random forest): Which method has given the best performance on the test set and which method has given you the best insight into the relationship between the price and the covariates?

We can now compare the test MSE scores of the four previous models.

```
test_mse <- tibble(
  best_subset = best_subset_test_mse,
  lasso = lasso_test_mse,
  tree = tree_test_mse,
  random_forest = random_forest_test_mse
)
t(test_mse) %>% kable(col.names = c("Test MSE"), format = "pandoc")
```

	Test MSE
best_subset	0.0036003
lasso	0.0038064
tree	0.0171555

	Test MSE
random_forest	0.0028110

We can see that the random forest has performed the best on the test set. The random forest is difficult to interpret though, especially since the two importance plots do not coincide. The tree model is much simpler to interpret, it condenses all the information down to just two covariates, `xx` and `yy`, making it easy to print out on a piece of paper and taking it with you to the jeweler.

Problem 2: Unsupervised learning [3 points]

Q21: What is the definition of a principal component score, and how is the score related to the eigenvectors of the matrix $\hat{\boldsymbol{R}}$.

Assume that you have an $n \times p$ design matrix $\boldsymbol{X}$, with a standardized version $\boldsymbol{Z}$ where the columns have been centered (having mean 0) and scaled (having variance 1). We want to linearly transform the set of (possibly) correlated covariates into a new orthogonal basis where the *new* variables are linearly uncorrelated while retaining as much of the information (variance) in the data as possible. These new variables, $\boldsymbol{W}_m$, are called *principal components* and the problem can be formally defined as trying to solve

$$\begin{aligned} &\boldsymbol{W}_m = \boldsymbol{Z}\boldsymbol{\phi}_m = \sum_{j=1}^{p} \phi_{mj} \text{Col}_j(\boldsymbol{Z}), && \forall m : ||\boldsymbol{\phi}_m||_2 = 1 \\ &\text{Var}(\boldsymbol{W}_i) \geq \text{Var}(\boldsymbol{W}_j) && \forall i, j : i < j \\ &\boldsymbol{W}_i \perp \boldsymbol{W}_j, && \forall i, j : i \neq j. \end{aligned} \tag{1}$$

The new variables are linear combination of the original columns. Since the components are sorted in decreasing order of variance contributed, we can subset the first M components in order to reduce the dimensionality of the original data set while retaining the maximum amount of information in the data set.

We will now explain how to solve this problem. The estimated covariance matrix, $\hat{\boldsymbol{R}} \in \mathbb{R}^{p \times p}$, for the standardized design matrix, $\boldsymbol{Z}$, is given as

$$\hat{\boldsymbol{R}} = \frac{1}{n-1} \sum_{i=1}^{n} (\boldsymbol{Z}_i - \bar{\boldsymbol{Z}})(\boldsymbol{Z}_i - \bar{\boldsymbol{Z}})^T.$$

In this case $\bar{Z} = 0$, since we have centered the design matrix. We know that $\text{rank}(\hat{\boldsymbol{R}}) = \min(p,\ n-1)$, which we will denote as r. Since $\hat{\boldsymbol{R}}$ is a symmetric and real, it is guaranteed to have an eigendecomposition. We can therefore diagonalize $\hat{\boldsymbol{R}}$ as

$$\hat{\boldsymbol{R}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1},$$

where $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, ..., \lambda_r, 0, ..., 0) \in \mathbb{R}^{p \times p}$ is the diagonal eigenvalue matrix sorted in *decreasing* order and $\boldsymbol{V} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_p]$ is a matrix which columns consists of the respective *eigenvectors* of $\hat{\boldsymbol{R}}$. It can be shown that $\boldsymbol{\phi} = [\phi_1\ ...\ \phi_p] = \boldsymbol{V}$ solves equation (1).

The *principal component scores* are the new values we attain in this transformed space. The transformation is given as $\boldsymbol{Z}\boldsymbol{\phi} = \boldsymbol{Z}\boldsymbol{V} \in \mathbb{R}^{p \times p}$, where only the r first columns are nonzero. We therefore end up in practice with r nonzero principal component scores for each of the original observations.

Q22: Explain what is given in the plot with title "First eigenvector". Why are there only $n = 64$ eigenvectors and $n = 64$ principal component scores?

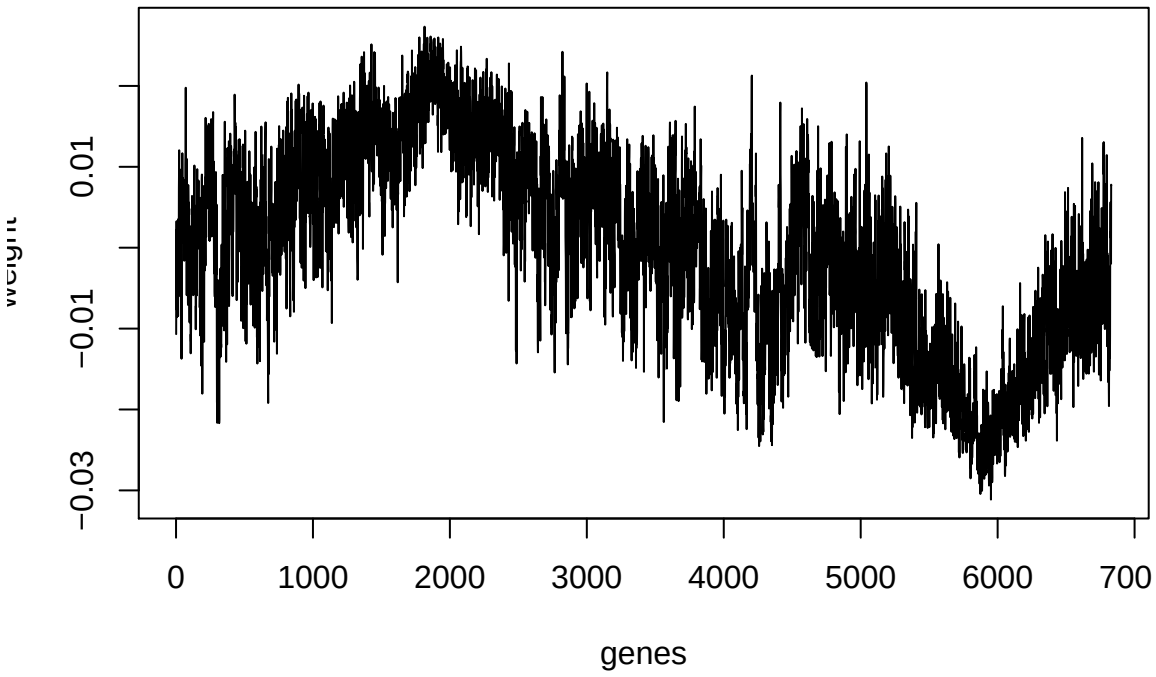


Figure 3: First eigenvector.

```
library(ElemStatLearn)
X <- t(nci)  # n times p matrix
table(rownames(X))

ngroups <- length(table(rownames(X)))
cols <- rainbow(ngroups)
cols[c(4, 5, 7, 8, 14)] <- "black"
pch.vec <- rep(4, 14)
pch.vec[c(4, 5, 7, 8, 14)] <- 15:19

colsvsnames <- cbind(cols, sort(unique(rownames(X))))
colsamples <- cols[match(rownames(X), colsvsnames[,2])]
pchvsnames <- cbind(pch.vec, sort(unique(rownames(X))))
pchsamples <- pch.vec[match(rownames(X), pchvsnames[,2])]

Z <- scale(X)
pca <- prcomp(Z)
plot(1:dim(X)[2], pca$rotation[,1], type="l", xlab="genes", ylab="weight")
```

```
##
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA
##           7           5           7           1           1           6
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE
##           1           1           8           9           6           2
##       RENAL     UNKNOWN
##           9           1
```

Figure 3 shows the values of of the first eigenvector of $\hat{\boldsymbol{R}}$. With other words, it shows the entries of ϕ_1, how to linearly combine the columns of $\boldsymbol{Z}$ in order to get the first principal component score, which explains the

most variance of all the principal components.

For our dataset, we have $r = \min(p,\ n-1) = \min(6830,\ 64-1) = 63$. There should therefore be $n = 63$ eigenvectors, and $n = 63$ principal component scores for each observation. The R-function `prcomp()` returns 64, but the last eigenvector is the first of the $n - r$ eigenvectors which are spanned by Null(Z). We would therefore expect $\lambda_{64} = 0$, which we can check

```
pca$sdev[64]^2
```

```
## [1] 1.530027e-28
```

This is for all intents and purposes equal to 0, which is what we expected.

Q23: How many principal components are needed to explain 80% of the total variance in $\boldsymbol{Z}$? Why is `sum(pca$sdev^2)=p`?

If we transform $\boldsymbol{Z}$ into the first M principal components, the fraction of the original variance that is kept is

$$R^2 = \frac{\sum_{i=1}^{M} \lambda_i}{\sum_{j=1}^{p} \lambda_j}$$

We can calculate these values by retrieving `pca$sdev`

```
library(magrittr)  # For is_weakly_greater_than()
total_variance <- sum(pca$sdev^2)
sprintf("Total variance: %.0f", total_variance)

cummulative_variance <- cumsum(pca$sdev^2)
proportional_variance <- cummulative_variance / total_variance
sufficient_components <- proportional_variance %>%
  is_weakly_greater_than(0.8) %>%
  which() %>%
  first()
sprintf(
  "80 percent of variance is explained by the first %.0f principal components",
  sufficient_components
)
```

```
## [1] "Total variance: 6830"
## [1] "80 percent of variance is explained by the first 32 principal components"
```

We can explain 80% of the variance with the 32 first principal components. Since the principal component analysis is performed on the *standardized* covariance matrix where the variance for each covariate has been scaled to 1 the total variance becomes $\sum_{i=1}^{p} 1 = p$.

Q24: Study the PC1 vs PC2 plot, and comment on the groupings observed. What can you say about the placement of the `K262`, `MCF7` and `UNKNOWN` samples? Produce the same plot for two other pairs of PCs and comment on your observations.

```
plot_pca <- function(first, second) {
  plot(
    pca$x[,first],
    pca$x[,second],
    xlab = paste("PC", first, sep = ""),
    ylab = paste("PC", second, sep = ""),
    pch = pchsamples,
    col = colsamples
  )
  legend(
```

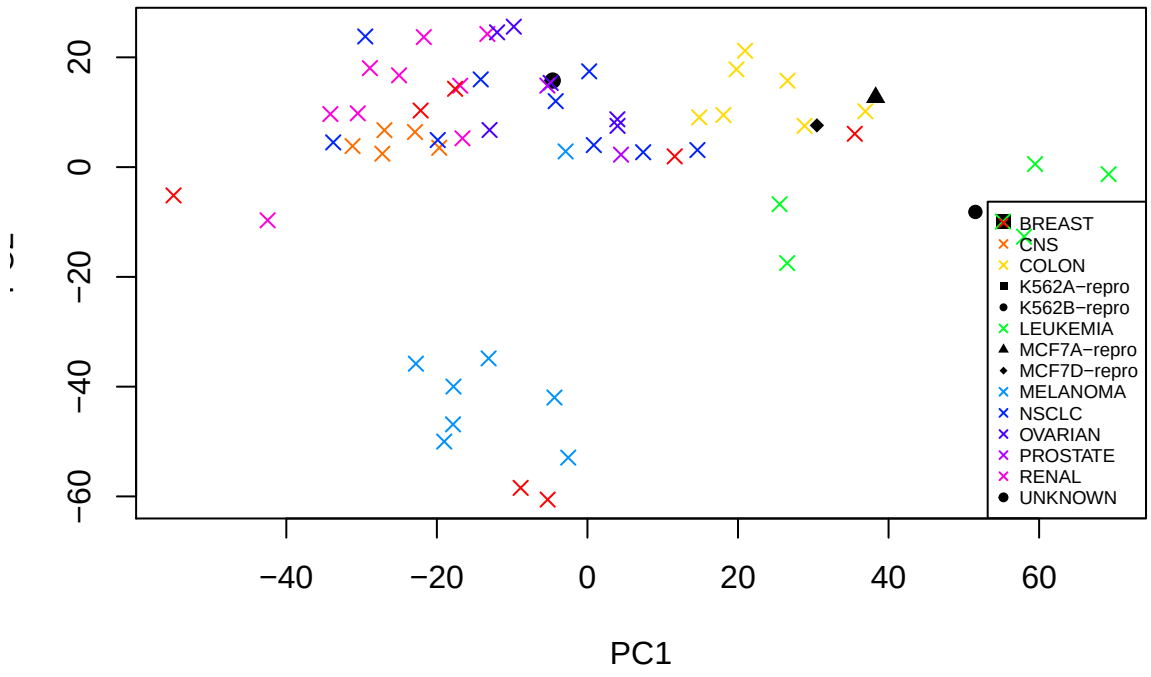


Figure 4: PC1 scores plotted against PC2 scores

```
    "bottomright",
    legend = colsvsnames[,2],
    cex = 0.55,
    col = cols,
    pch = pch.vec
  )
}

plot_pca(1, 2)
```

`MELANOMA`, `LEUKEMIA`, and `COLON` are the cancer tumor cells that portrays the greatest degree of clustering. The `K562{A,B}` cells, which are known to be leukemia cells, are located in the vicinity of the `LEUKEMIA` cluster, which makes sense. The `MCF7{A,D}`, which are known to be breast cancer cells, are located in the `COLON` cluster. Perhaps these cancer types are similar in some way? The `UNKNOWN` sample seems to be clustered close to `PROSTATE`, `NSCLC`, and `OVARIAN`. Perhaps `UNKNOWN` is an `NSCLC` sample, because there are most `NSCLC` samples in this region?

Below we have plotted the first and third principal components against each other (figure 5), as well as the first and fourth (figure 6). Similar trends are still noticeable for `K562A-repro` and `K562B-repro`, and the `UNKOWN` sample. For `MCF7A-repro` and `MCF7D-repro` these two plots show that their association with the `COLON`-cluster is not as strong as we saw in the plot of the first two PC's.

```
plot_pca(1, 3)
```

```
plot_pca(1, 4)
```

Q25:: Explain what it means to use Euclidean distance and average linkage for hierarchical clustering.

A cluster, C_k, is considered to be good if it has a *within-cluster variation* that is as small as possible. The within-cluster variation, $W(C_k)$, is defined in terms of the *squared* Euclidean distance between the members of the cluster

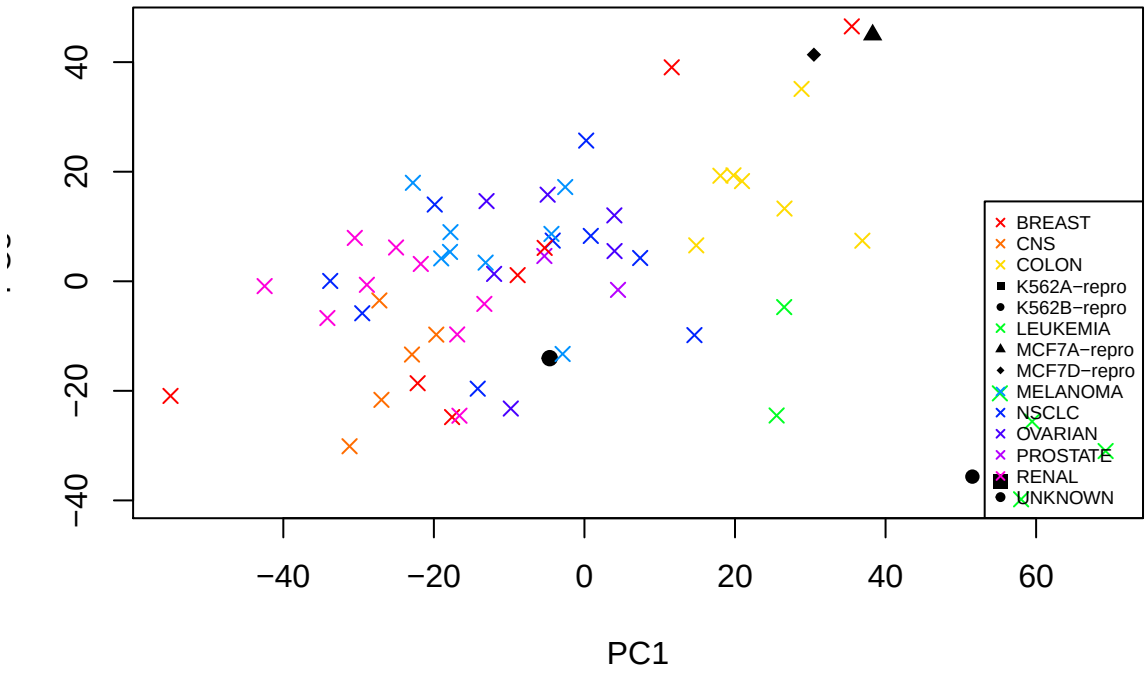


Figure 5: First PCA plotted against third PCA.

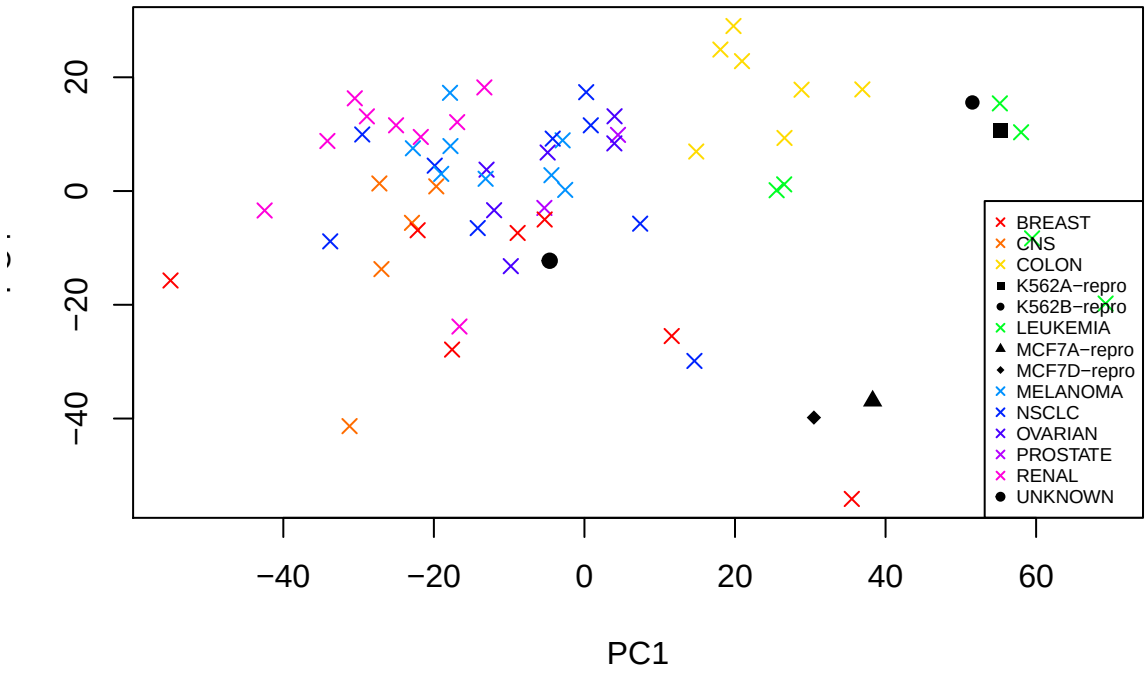


Figure 6: First PCA plotted against fourth PCA.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

where $|C_k|$ is the cardinality of the cluster index set. Thus we want to find a cluster superset $C = \{C_1, ..., C_k\}$ which minimizes $\sum_{k=1}^{K} W(C_k)$. Several algorithms exists for finding such cluster sets, one of which is the *hierarchical clustering algorithm*. The algorithm goes as follows:

1. Start with all n observations, i.e. the bottom of the *dendogram* (see next task). Consider each observation to be its own cluster.
2. `while number of clusters` > 1:
   (a) Fuse the two clusters that are the most similar to each other.
   (b) Decrement `number of clusters` by 1.

A question remains to be answered; how to define *similarity* between clusters? Consider two clusters consisting of a single observation each, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. We can use the *euclidean distance* $||\boldsymbol{x}_1 - \boldsymbol{x}_2||_2$ as a similarity measure.

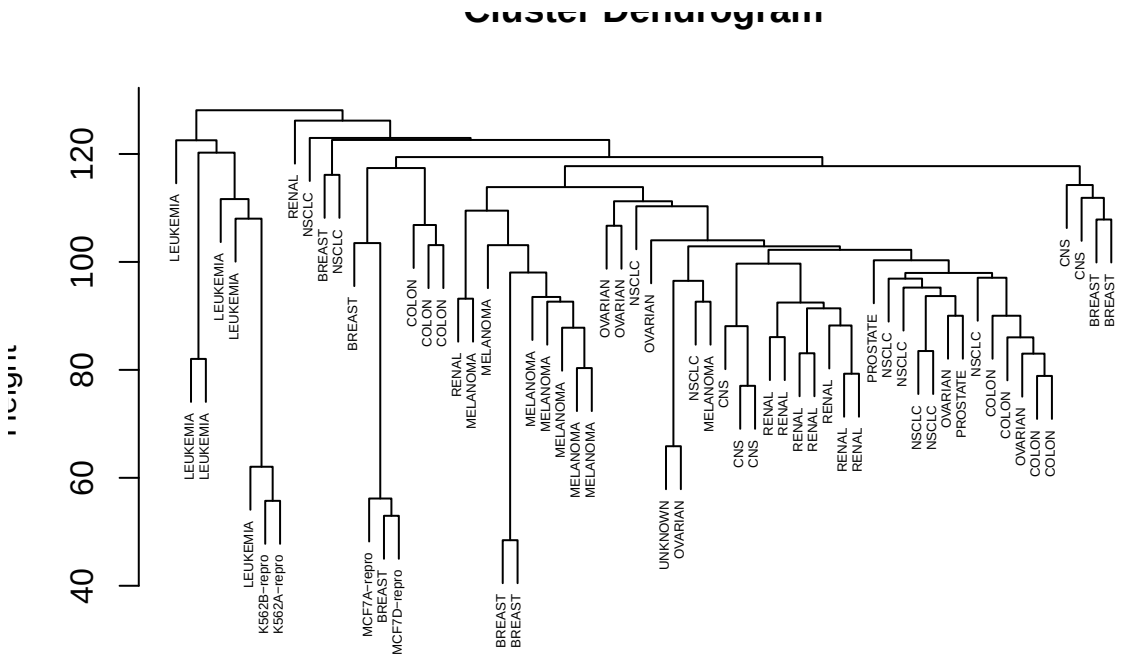
But how do we measure the similarity between two clusters with cardinalities that are not equal to 1? We need to generalize the concept of similarity to *groups* of observations. This is called *linkage*. There are no straight forward answers here, but one proposal solution is to use *mean inter-cluster dissimilarity*, aka. "average linking". Define the average distance measure D_a between two clusters C_1 and C_2 as

$$D_a(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \sum_{(x_i, x_j) \in C_1 \times C_2} ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2,$$

where $C_1 \times C_2$ is the Cartesian product between the two cluster sets. This is now what we want to minimize when we fuse two clusters of arbitrary cardinality.

Q26:: Perform hierarchical clustering with Euclidean distance and average linkage on the scaled gene expression in `Z`. Observe where our samples labelled as K562, MCF7 and `UNKNOWN` are placed in the dendrogram. Which conclusions can you draw from this?

```
dissimilarity_structure <- dist(Z, method = "euclidean")
hc_model <- hclust(dissimilarity_structure, method = "average")
plot(hc_model, cex = 0.45)
```



dissimilarity_structure
hclust (*, "average")

The two `K562` cells are first clustered together, and afterwards with all the other `LEUKEMIA` samples. This is what we would expect, as they should be similar. The `MCF7` cells, which are known to be breast cancer cells, are also clustered with `BREAST` cells, which is also expected from what we know *a priori*.

Lastly, the `UNKNOWN` cell is considered to be relatively similar to an `OVARIAN` sample. Perhaps this cell sample is ovarian cancer?

Q27:: Study the R-code and plot below. Here we have performed hierarchical clustering based on the first 64 principal component instead of the gene expression data in `Z`. What is the difference between using all the gene expression data and using the first 64 principal components in the clustering? We have plotted the dendrogram together with a heatmap of the data. Explain what is shown in the heatmap. What is given on the horizontal axis, vertical axis, value in the pixel grid?

In task **Q26** observations are clustered according to their respective euclidean distances in the original covariate vector space $\mathbb{R}^p$, i.e.

$$||\boldsymbol{z}_1 - \boldsymbol{z}_2||_2 = \sqrt{\sum_{j=1}^{p}(z_{1j} - z_{2j})^2}$$

When performing hierarchical clustering on the principal component scores, the distance measure is defined over the principal component space $\mathbb{R}^r$ instead.

$$||\boldsymbol{w}_1 - \boldsymbol{w}_2||_2 = \sqrt{\sum_{j=1}^{r}(w_{1j} - w_{2j})^2}$$

The 64^{th} principal component score will always be ≈ 0, so it will not contribute to the distance. Which different does this make? Observe the following

$$
\begin{aligned}
||\boldsymbol{w}_1 - \boldsymbol{w}_2||_2 &= ||\boldsymbol{z}_1^T\boldsymbol{V} - \boldsymbol{z}_2^T\boldsymbol{V}||_2 = ||(\boldsymbol{z}_1 - \boldsymbol{z}_2)^T\boldsymbol{V}||_2 \\
&= \left[(\boldsymbol{z}_1 - \boldsymbol{z}_2)^T \boldsymbol{V} \left((\boldsymbol{z}_1 - \boldsymbol{z}_2)^T \boldsymbol{V}\right)^T\right]^{1/2} \\
&= \left[(\boldsymbol{z}_1 - \boldsymbol{z}_2)^T \boldsymbol{V}\boldsymbol{V}^T (\boldsymbol{z}_1 - \boldsymbol{z}_2)\right]^{1/2} \\
&\underset{(a)}{=} \left[(\boldsymbol{z}_1 - \boldsymbol{z}_2)^T (\boldsymbol{z}_1 - \boldsymbol{z}_2)\right]^{1/2} \\
&= ||\boldsymbol{z}_1 - \boldsymbol{z}_2||_2
\end{aligned}
$$

In (a) we have used the fact that $\boldsymbol{V}$ is a unitary matrix, such that $\boldsymbol{V}\boldsymbol{V}^T = I$.

We can see that the distances have been preserved in the principal component space. This can be confirmed numerically by calculating the largest percentage difference in the respective euclidean distances:

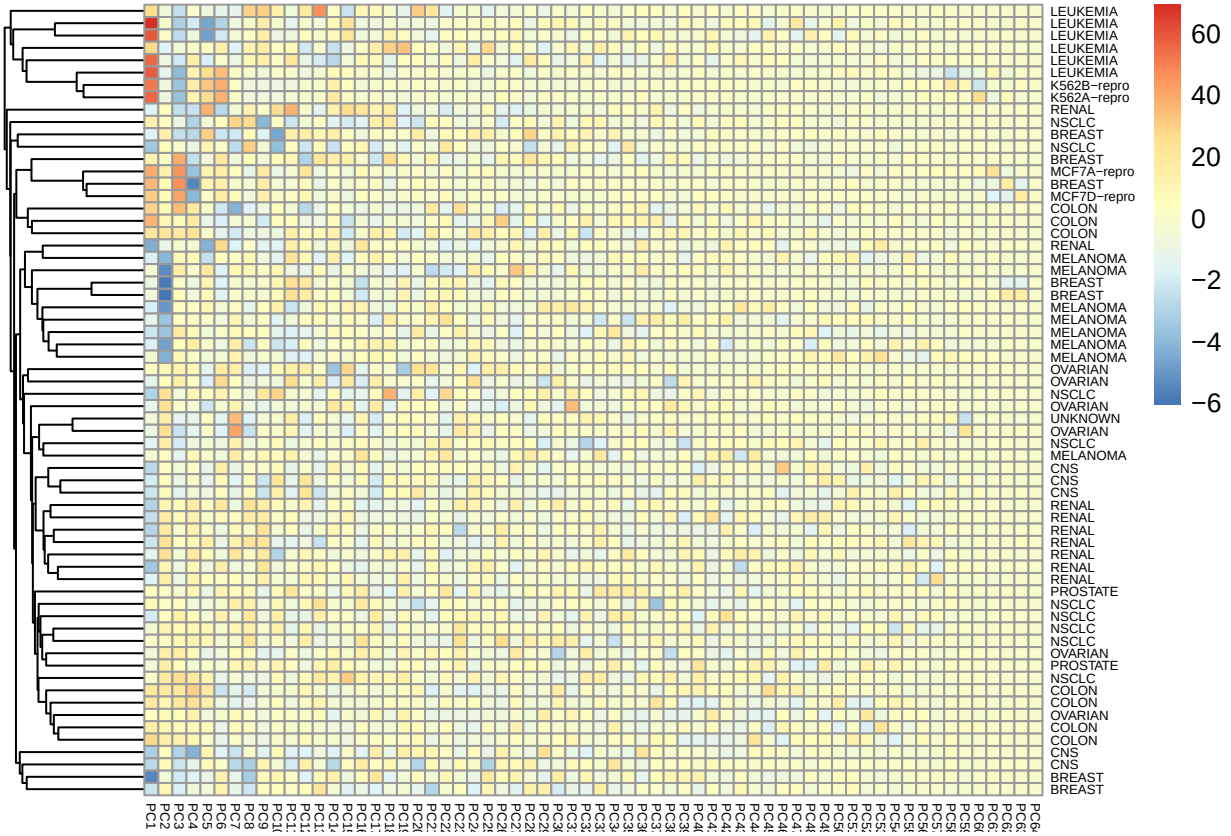
```
max_percentage_difference <- max(100 * (dist(pca$x) - dist(Z)) / dist(Z))
print(max_percentage_difference)
```

```
## [1] 3.501195e-13
```

Only at most a $3.5 \cdot 10^{-13}$ percent difference between the two spaces, which can be attributed to numerical errors. So in conclusion, doing clustering in the PC-space makes absolutely no difference, besides possible different numerical costs. It may also help with visualization, as we will now show.

We now plot the principal component scores as a heatmap.

```
library(pheatmap)
par(mfcol=c(1, 1))
npcs <- 64
pheatmap(
  pca$x[,1:npcs],
  scale = "none",
  cluster_col = FALSE,
  cluster_row = TRUE,
  clustering_distance_rows = "euclidean",
  clustering_method = "average",
  fontsize_row = 5,
  fontsize_col = 5
)
```



The hierarchical clustering algorithm can be seen on the left vertical axis. The horizontal axis shows the principal components, while the right vertical axis shows the observations. The pixels show the principal component scores of each observation, red pixels indicate great positive scores while blue values indicate great negative scores. Here it becomes apparent that the first principal component has the most variance, and that the variance decreases as we move right. The last column, which is the 64th principal components, is clearly populated by zeros, as explained earlier.

Problem 3: Flying solo with diabetes data [6 points]

First we import the data set.

```
flying <- dget("flying.dd")
ctrain <- flying$ctrain %>% as_tibble()
ctest <- flying$ctest %>% as_tibble()
```

Q28: Start by getting to know the *training data*, by producing summaries and plots. Write a few sentences about what you observe and include your top 3 informative plots and/or outputs.

`summary()`

First it is practical to get a quick summary of the data structure of the training set by using `summary()`:

```
summary(ctrain)
```

```
##     diabetes          npreg             glu              bp
##  Min.   :0.0000   Min.   : 0.000   Min.   : 57.0   Min.   : 24.00
##  1st Qu.:0.0000   1st Qu.: 1.000   1st Qu.:100.0   1st Qu.: 64.00
##  Median :0.0000   Median : 2.000   Median :117.0   Median : 70.00
##  Mean   :0.3333   Mean   : 3.437   Mean   :122.7   Mean   : 71.23
```

```
##  3rd Qu.:1.0000   3rd Qu.: 5.000   3rd Qu.:143.0   3rd Qu.: 78.00
##  Max.   :1.0000   Max.   :17.000   Max.   :199.0   Max.   :110.00
##       skin             bmi              ped              age
##  Min.   : 7.00   Min.   :18.20   Min.   :0.0850   Min.   :21.00
##  1st Qu.:21.00   1st Qu.:27.88   1st Qu.:0.2532   1st Qu.:23.00
##  Median :29.00   Median :32.90   Median :0.4075   Median :28.00
##  Mean   :29.22   Mean   :33.09   Mean   :0.4789   Mean   :31.48
##  3rd Qu.:37.00   3rd Qu.:37.12   3rd Qu.:0.6525   3rd Qu.:37.25
##  Max.   :60.00   Max.   :67.10   Max.   :2.1370   Max.   :70.00
```

First of, we observe that *all* the columns are of numeric types. This is okay for the covariates, as they are considered to be numeric, but we prefer to encode the `diabetes` response as a factor variable, so we will do this straight away.

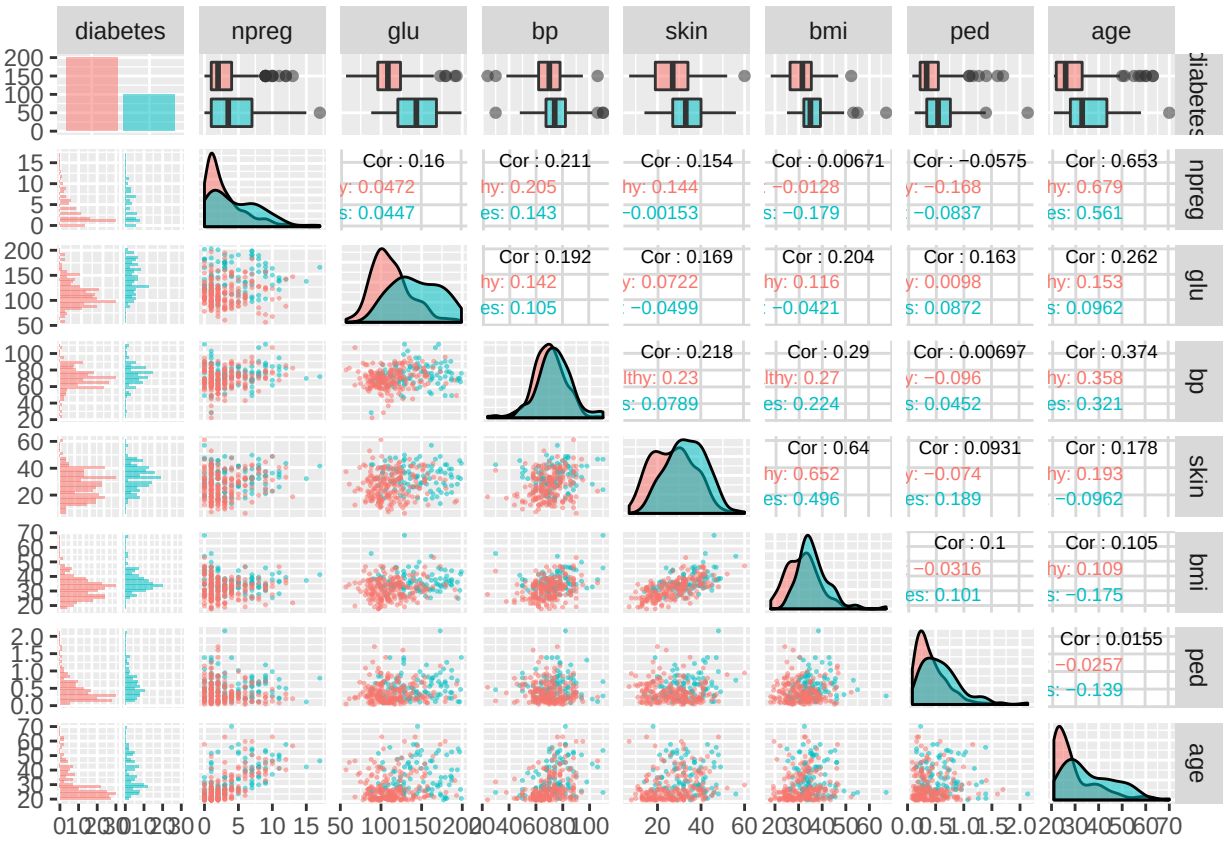
```
ctrain$diabetes <- as.factor(ctrain$diabetes)
ctest$diabetes <- as.factor(ctest$diabetes)
ctrain$diabetes %<>% recode("0" = "healthy", "1" = "diabetes")
ctest$diabetes %<>% recode("0" = "healthy", "1" = "diabetes")
```

Otherwise, we can see that one third of the observations have `diabetes = 1`. Also, the median number of pregnancies is 2 and the average is 3.437, so we have a positive skew in pregnancies. The `age` column spans from 21 to 70, which is good, since most age segments are represented. The dataset is not standardized in any way. Lastly, the average and median `bmi` is ≈ 33, which is considered to be obese, which makes sense in a sample set where 33% of the observations have diabetes.

`ggpairs()`

Next up we can use `ggpairs()` to get a really quick overview of the distributional properties of the data and how they correlate.

```
library(GGally)
ggpairs(
  ctrain,
  mapping = aes(color = diabetes, alpha = 0.5),
  upper = list(continuous = wrap("cor", size = 2.5)),
  lower = list(continuous = wrap("points", size = 0.2))
)
```

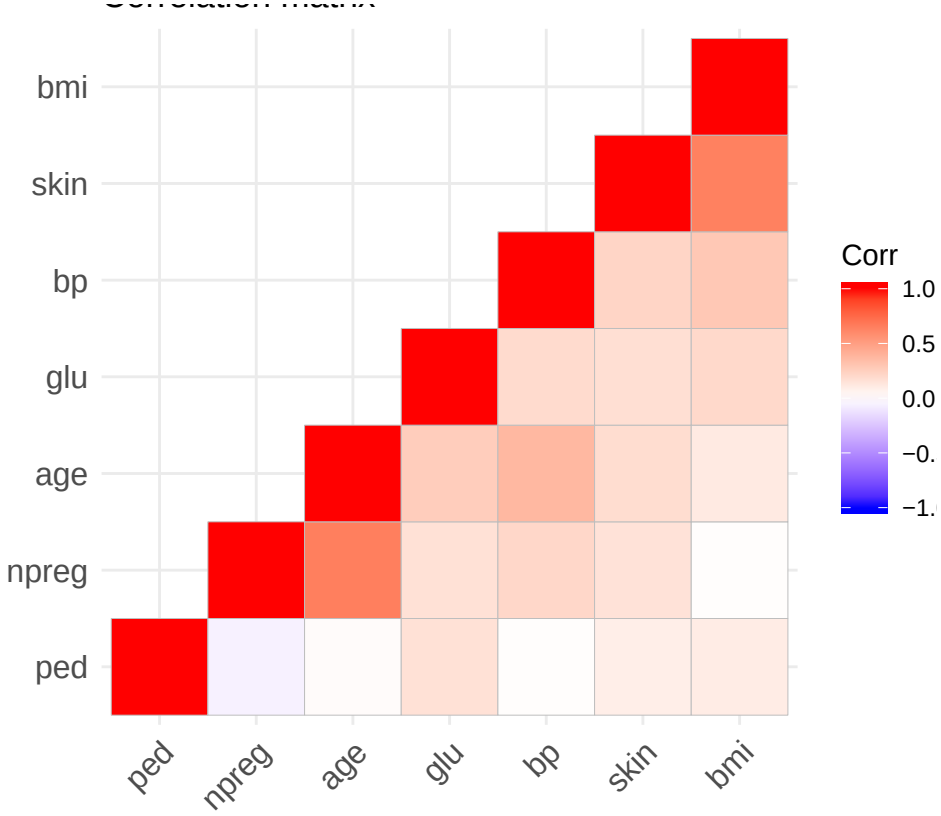


Here, `diabetes = 0` is shown in red and `diabetes = 1` is shown in blue. We can now see that diabetes sufferers are overrepresented in the right tail of all the covariates (see the diagonal), especially `glu`.

`ggcorrplot()`

We can take a closer look at the covariance matrix of the design matrix with `ggcorrplot()`.

```
library(ggcorrplot)
library(corrr)
ctrain %>%
  select(-diabetes) %>%
  cor() %>%
  ggcorrplot(
    type = "lower",
    hc.order = TRUE,
    title = "Correlation matrix",
    show.diag = TRUE
  )
```



What is interesting here is the fact that Cor(`age`, `npreg`) and Cor(`skin`, `bmi`) are the greatest amongst the correlations, with Cor(`age`, `bp`) following somewhat behind. Age being correlated with number of pregnancies makes sense. Skin thickness is highly correlated with obesity, which BMI is intended to measure, it therefore makes sense that also these are correlated.

Q29: Use different methods to analyse the data. In particular use

Module 4: Classification

We can choose between logistic regression, LDA, or QDA. The two latter methods assumes that the class-specific distributions of the covariates are approximately Gaussian. The boxplots and histograms in the `ggpairs`-plot show us that this assumption does not hold. We will therefore analyze the data using logistic regression.

The logistic regression method comes from the diagnostic paradigm of classification. This paradigm consists of methods where the goal is to estimate the posterior distribution $P(Y = k|X = x)$ for each class $k \in C$, where C is the set of classes the response Y can belong to. Logistic regression requires independence between observational pairs $(Y_i, \boldsymbol{x}_i)$.

The diabetes classification is a two-class problem, $C = \{0, 1\}$, and we write $P(Y_i = 1|\mathbf{X} = \mathbf{x_i}) = p(\mathbf{x_i})$ for brevity. Since we are modelling probabilities we need $p(\mathbf{x_i}) \in [0, 1]$. The most common function form used for achieving this is the sigmoid function $\sigma(x) = \frac{e^x}{1+e^x}$. Using this the logistic regression model becomes

$$p(\boldsymbol{x}_i) = \frac{\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\}}{1 + \exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\}} = \frac{\exp\left\{\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}\right\}}{1 + \exp\left\{\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}\right\}} \in [0, 1]$$

With $P(Y_i = 0|\mathbf{X} = \mathbf{x_i}) = 1 - p(\mathbf{x_i})$. The coefficients β are now estimated using MLE, where the coefficients that maximizes the likelihood function

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}}\ \ell(\boldsymbol{\beta}; \boldsymbol{x}, \boldsymbol{y})$$

$$\ell(\boldsymbol{\beta}; \boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{n} p\left(\boldsymbol{x}_i \mid \boldsymbol{\beta}\right)^{y_i} \left(1 - p\left(\boldsymbol{x}_i \mid \boldsymbol{\beta}\right)\right)^{1-y_i}$$

are chosen.

As in linear regression, we need to decide on the number of parameters to include in our model. One way to decide on which subset of the p available predictors to include is best subset selection. This is performed exhaustively below with AIC as selection criterion.

```
library(magrittr)  # For use_series()
library(caret)  # For confusionMatrix()
library(e1071)  # For ConfusionMatrix()
library(purrr)  # For map_dbl()
set.seed(0)

# Create bestglm-compliant dataframe
Xy <- as.data.frame(
  within(
    ctrain,
    {
      y = diabetes    # setting response
      diabetes = NULL # not include as covariate
    }
  )
)

# Logistic model selection based on AIC
logistic_model <- bestglm(
    Xy = Xy,
    family = binomial,
    IC = "AIC",
    method = "exhaustive"
  ) %>%
  use_series(BestModel)
```

We can take a look at the resulting model with `summary()`:

```
summary(logistic_model)
```

```
##
## Call:
## glm(formula = y ~ ., family = family, data = Xi, weights = weights)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8277  -0.6312  -0.3269   0.5922   2.2182
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -11.196794   1.356891  -8.252  < 2e-16 ***
## npreg         0.114009   0.061917   1.841 0.065577 .
## glu           0.034711   0.005759   6.028 1.66e-09 ***
```

```
## bmi           0.107235   0.024922   4.303 1.69e-05 ***
## ped           1.969892   0.523686   3.762 0.000169 ***
## age           0.032784   0.019115   1.715 0.086322 .
## —
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 381.91  on 299  degrees of freedom
## Residual deviance: 255.89  on 294  degrees of freedom
## AIC: 267.89
##
## Number of Fisher Scoring iterations: 5
```

The resulting model formula is `diabetes ~ npreg + glu + bmi + ped + age` where only `npreg` and `age` are considered insignificant at a significance level of $\alpha = 0.05$, but significant at $\alpha = 0.1$.

To interpret the fitted model it is useful to define the notion of odds. This is just the ratio of the estimated probabilities of the two classes. After taking the logarithm of the odds, and doing some algebraic manipulation we get that

$$\log\left(\frac{\hat{p}(\boldsymbol{x}_i)}{1-\hat{p}(\boldsymbol{x}_i)}\right) = \boldsymbol{x}_i^T\hat{\boldsymbol{\beta}},$$

which shows that the log odds is linear in the predictors when using the sigmoid function. This means that by increasing one of the predictors x_{ij} by one unit, while holding everything else constant, the predicted log odds will increase by $\hat{\beta}_j$ units. Equivalently, the odds will increase by a factor of $e^{\hat{\beta}_j}$. The linear relation between our independent variables and tho log odds is one of our model assumptions.

```
coefficients(logistic_model) %>%
  exp() %>%
  kable(format = "latex", col.names = c("exp(beta)"))
```

	exp(beta)
(Intercept)	0.0000137
npreg	1.1207618
glu	1.0353208
bmi	1.1131957
ped	7.1698986
age	1.0333275

Here we see, for instance, that for every additional pregnancy we expect a $\approx 12.1\%$ increase in the predicted odds of having diabetes, everything else being equal.

Under the Bayes classifier in a two-class problem an observation $\boldsymbol{x}_i$ is assigned the class for which the predicted posterior is greater than 0.5. Using this, we can evaluate the performance of the model on the test set. The results are shown below, along with a plot of the ROC-curve and the corresponding AUC.

```
# Create diabetes predictions
logistic_predictions <- predict(logistic_model, ctest, type = "response") %>%
  map_lgl(~ .x > 0.5) %>%
  if_else("diabetes", "healthy") %>%
  as_factor()

# Create confusion matrix
logistic_confmat <- confusionMatrix(
  data = logistic_predictions,
```

```
  reference = ctest$diabetes
)
print(logistic_confmat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction healthy diabetes
##   healthy      137       29
##   diabetes      18       48
##
##                Accuracy : 0.7974
##                  95% CI : (0.7399, 0.8472)
##     No Information Rate : 0.6681
##     P-Value [Acc > NIR] : 9.374e-06
##
##                   Kappa : 0.5262
##
##  Mcnemar's Test P-Value : 0.1447
##
##             Sensitivity : 0.8839
##             Specificity : 0.6234
##          Pos Pred Value : 0.8253
##          Neg Pred Value : 0.7273
##              Prevalence : 0.6681
##          Detection Rate : 0.5905
##    Detection Prevalence : 0.7155
##       Balanced Accuracy : 0.7536
##
##        'Positive' Class : healthy
##
```

The overall misclassification rate of the model is $\approx 20.3\%$.

Module 8: Trees (and forests)

How it works

We now want to fit a *boosted* tree to the data set. A boosted tree is a sequential learning model that learns by "modifying" itself based on the remaining residuals in each step. Here is how it works algorithmically:

1. Set $\hat{f}^{(0)}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, ..., B$, repeat:
   (a) Fit a *new* $\hat{f}^{(b)}$ with d splits, from the remaining residuals of the previous iteration. This results in $d + 1$ leaf nodes.
   (b) Update $\hat{f}$ by adding a shrunken down version of this new tree:
   $$\hat{f} \leftarrow \hat{f} + \lambda \hat{f}^{(b)}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^{(b)}(x_i)$$
3. The boosted model is $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^{(b)}(x)$.

Parameters

We will use the package `gbm` in order to fit this boosted tree, but we need to pick the following tuning parameters:

- The number of trees grown, B. If this is too low, not all of the information is used, while if it is too large we will overfit the model. `gbm` calls this paramater `n.trees`.
- The shrinkage parameter, λ, determines how fast the the algorithm learns. A lower value will require more trees in order to offset the slow learning. `gbm` calls this parameter `shrinkage`.
- The interaction depth, d, is the number of splits in each tree. Greater values imply a greater degree of interaction between the covariates, while $d = 1$ results in an *additive* model. `gbm` calls this parameter `interaction.depth`.
- Additionally, `gbm` requires the parameter `n.minobsinnode` which constrains the minimal size of the leaf nodes. That way the algorithm can't put "troublesome" residuals into very small leaf nodes in order to reduce such residuals, that would lead to overfitting. We will denote this parameter as m.

The question remains, how to choose these parameters? We will use 10-fold cross validation in conjunction with a grid search in order to pick suitable tuning parameters from our training set.

The R-package `caret` enables us to specify the cross validation with `trainControl()`:

```
library(caret)
cross_validation <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  allowParallel = TRUE
)
```

We can also specify the grid on which to search for the best tuning parameters. We will choose:

$$\begin{aligned}\boldsymbol{B} &= [10, 20, 30, 50, 100] \\ \boldsymbol{d} &= [1, 3, 5, 10] \\ \boldsymbol{\lambda} &= [0.2, 0.15, 0.1, 0.05, 0.01, 0.001] \\ \boldsymbol{m} &= [10, 15, 20, 30]\end{aligned}$$

where the grid to search upon becomes $\boldsymbol{B} \times \boldsymbol{d} \times \boldsymbol{\lambda} \times \boldsymbol{m}$. This can also be specified in `caret` with the function `expand.grid()`:

```
tune_grid <- expand.grid(
 n.trees = c(10, 20, 30, 50, 100),
 interaction.depth = c(1, 3, 5, 10),
 shrinkage = c(0.2, 0.15, 0.1, 0.05, 0.01, 0.001),
 n.minobsinnode = c(10, 15, 20, 30)
)
```

Fitting the model

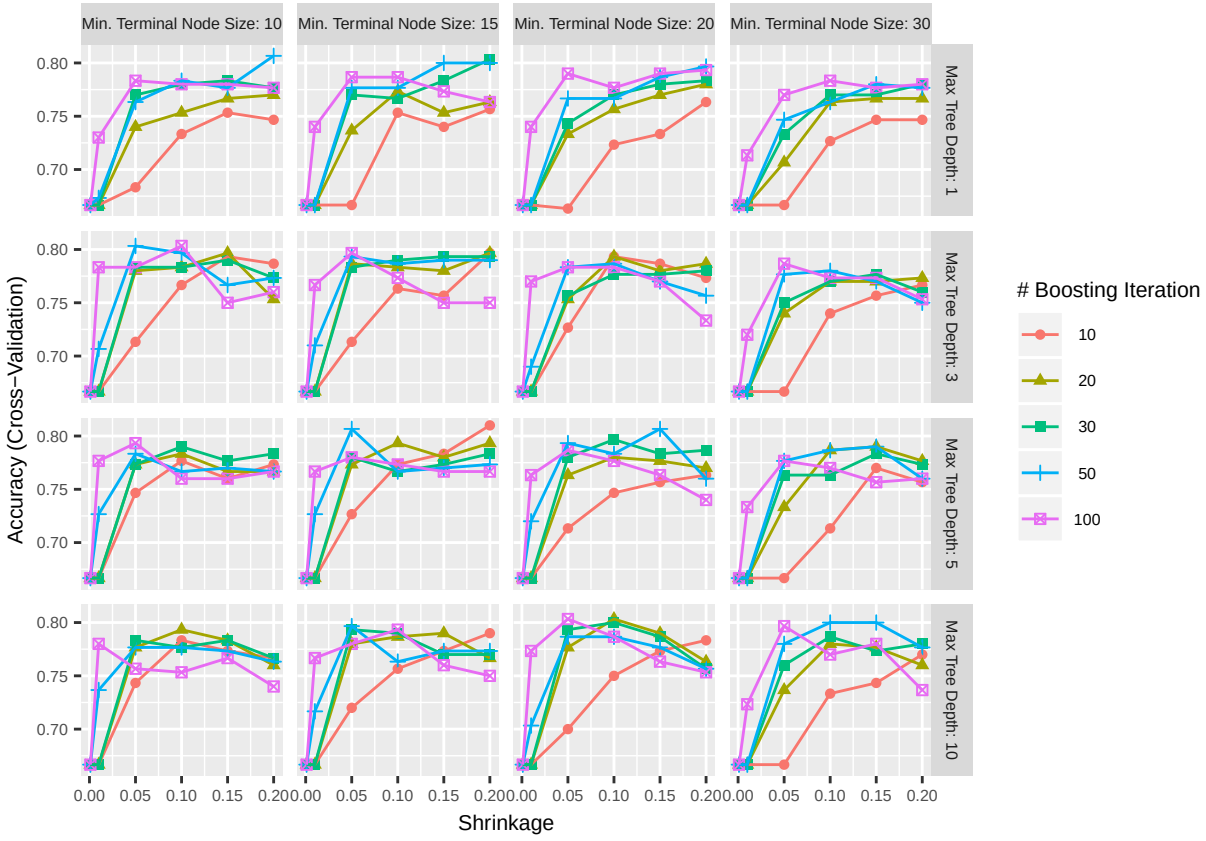
We can now instruct `caret` to train a boosted tree model with a search upon this grid. The model with the best cross-validated accuracy is chosen.

```
set.seed(0)
boost_model <- caret::train(
  form = diabetes ~ .,
```

```
  data = ctrain,
  method = "gbm",
  trControl = cross_validation,
  tuneGrid = tune_grid,
  verbose = FALSE
)
```

We can plot the cross-validated accuracy for each point on the grid:

```
ggplot(
  boost_model,
  metric = "Accuracy",
  nameInStrip = TRUE
) + theme(text = element_text(size=8))
```



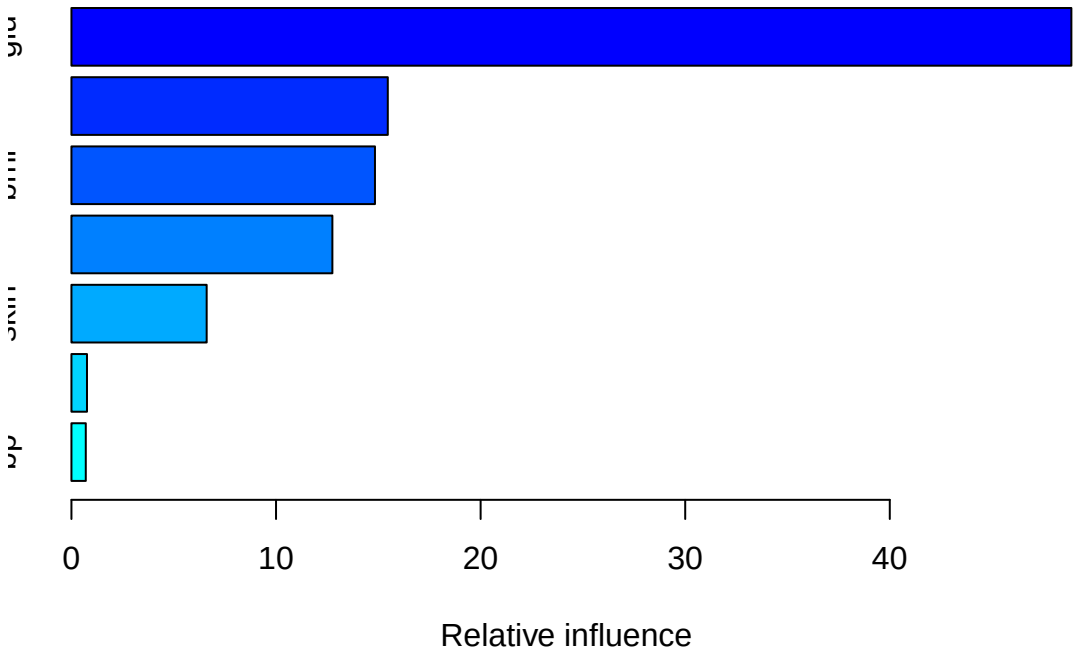
We can get the specific point on the grid which is considered optimal:

```
boost_model$bestTune
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 446      10                 5       0.2             15
```

$B = 10$, $d = 5$, $\lambda = 0.2$, and $m = 15$ performs best, so this is what is chosen in the final model. We can create a variable importance plot of the final model

```
final_model <- boost_model$finalModel
summary(final_model)
```




```
##         var    rel.inf
## glu     glu 48.8760656
## age     age 15.4624943
## bmi     bmi 14.8362051
## ped     ped 12.7539190
## skin   skin  6.6094913
## npreg npreg  0.7605521
## bp       bp  0.7012727
```

`glu` (glucose) is considered the most important covariate according to our final model. This coincides with what the authors would expect as blood sugar levels are highly related to diabetes. We can calculate the confusion matrix of this model tested against our test data set.

```
boost_probabilities <- predict(
    boost_model,
    newdata = ctest,
    type = "prob"
  ) %>%
  pull(diabetes)
boost_predictions <- boost_probabilities %>%
  map_lgl(~ .x > 0.5) %>%
  if_else("diabetes", "healthy") %>%
  as_factor()

boost_confmat <- confusionMatrix(
  data = boost_predictions,
  reference = ctest$diabetes
)
print(boost_confmat)
```

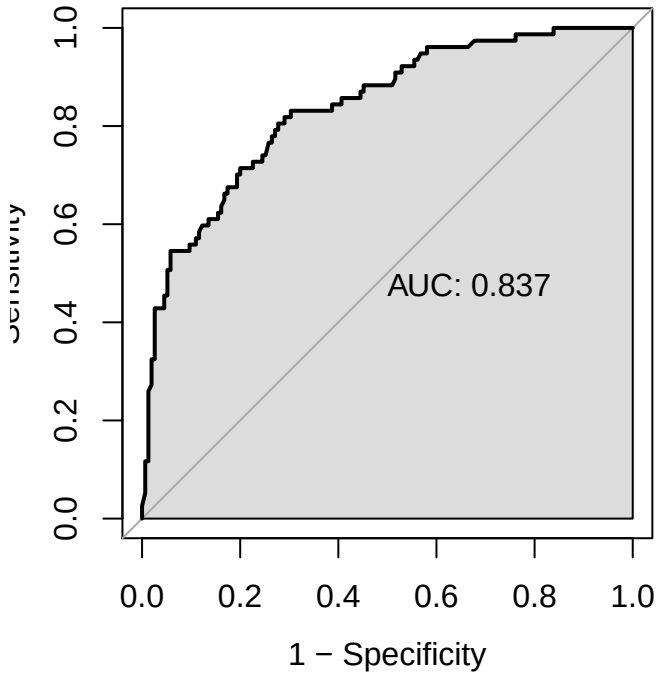
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction healthy diabetes
```

```
##   healthy      138       33
##   diabetes      17       44
##
##                Accuracy : 0.7845
##                  95% CI : (0.7259, 0.8356)
##     No Information Rate : 0.6681
##     P-Value [Acc > NIR] : 6.596e-05
##
##                   Kappa : 0.4872
##
##  Mcnemar's Test P-Value : 0.03389
##
##             Sensitivity : 0.8903
##             Specificity : 0.5714
##          Pos Pred Value : 0.8070
##          Neg Pred Value : 0.7213
##              Prevalence : 0.6681
##          Detection Rate : 0.5948
##    Detection Prevalence : 0.7371
##       Balanced Accuracy : 0.7309
##
##        'Positive' Class : healthy
##
```

The result here is an overall accuracy of $\approx 78.5\%$, i.e. a misclassification rate of 21.5%. Interestingly, we have a $\approx 89\%$ sensitivity, so the model is able to correctly predict diabetes for nine out of ten diabetes sufferers.

Finally, we can plot the ROC-curve for the model and report the AUC.

```
library(pROC)
roc(
  response = ctest$diabetes,
  predictor = boost_probabilities,
  plot = TRUE,
  auc.polygon = TRUE,
  print.auc = TRUE,
  legacy.axes = TRUE,
  xlim = c(1, 0)
)
```




```
##
## Call:
## roc.default(response = ctest$diabetes, predictor = boost_probabilities,     plot = TRUE, auc.polygon = TRUE, pr
##
## Data: boost_probabilities in 155 controls (ctest$diabetes healthy) < 77 cases (ctest$diabetes diabetes).
## Area under the curve: 0.8371
```

Here we see that the ROC curve is far from being diagonal and AUC $\approx 0.84 > 0.5$, which a random guessing model would have. It can therefore be considered to have explaining power.

Module 9: Support vector machines

We will now implement a *support vector machine* with a *linear kernel* for the diabetes data set.

How it works

First off, let us notationally recode the `diabetes` response vector such that $y_i = 1$ indicates a patient with diabetes, as before, but $y_i = -1$ for non-diabetes patients. Now, define a p-dimensional *hyperplane* as

$$\beta_0 + \boldsymbol{x}^T\boldsymbol{\beta} = 0,$$

where $\boldsymbol{\beta} = [\beta_1, ..., \beta_p]^T$ is the plane's normal vector. Assuming that $||\boldsymbol{\beta}||_2 = 1$, then the value of $\beta_0 + \boldsymbol{x}_i^T\boldsymbol{\beta}$ gives the distance from the plane to $\boldsymbol{x}_i$. The value is positive if the point $\boldsymbol{x}_i$ lies on *one* side of the hyperplane, while it is negative if it lies on the *other* side. Now the idea is to find a hyperplane which satisfies

$$y_i(\beta_0 + \boldsymbol{x}_i^T\boldsymbol{\beta}) > 0,$$

such that we can construct a classifier for a new observation $\boldsymbol{x_0}$ as

$$\hat{f}(x) = \begin{cases} 1 & , \text{ if } \beta_0 + \boldsymbol{x_0}^T\boldsymbol{\beta} > 0 \\ -1 & , \text{ otherwise.} \end{cases}$$

Ideally we would like to find a *maximal margin hyperplane* which perfectly divides the two classes of observations from each other and is the farthest away from the training observations as possible. This distance is what we call the *margin*, and we denote this length as M.

Now in practice, such a perfect dividing hyperplane seldom exists. We must allow some training observations to lie within this margin or even lie on the wrong side of the hyperplane. We introduce the concept of *slack* in order to formulate an optimization problem that finds a *soft-margin classifier* instead

$$
\begin{aligned}
\underset{\beta_0,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{argmax}} \quad & M, \text{ subject to...} \\
||\boldsymbol{\beta}||_2 =& 1, \\
y_i(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i) \geq& M(1-\epsilon_i), \qquad (\forall i = 1, ..., n) \\
\epsilon_i \geq& 0, \\
\sum_i^n \epsilon_i \leq& C.
\end{aligned}
$$

$\epsilon_1, ..., \epsilon_n$ are the *slack variables* which allows the optimization algorithm to place points on the wrong side of the margin. If ϵ_i equals zero, the observation lies on the right side of the *margin*, and if it is greater than zero it does not. Lastly, if it is greater than one, it lies on the wrong side of the *hyperplane* and will therefore be wrongly classified.

C is here a tuning parameter that restricts the optimization algorithm in its pursuit of the widest possible margin, as the aggregate sum of the slack variables can't exceed this limit. With other words, no more than C training observations may be wrongly classified.

Parameter selection

The tuning parameter C will again be found with a grid search (actually a line search) with 10-fold cross validation with accuracy as the selection metric yet again. We will search over the points $\boldsymbol{C} = [0.001, 0.01, 0.1, 1, 2, 3, 4, 5, 10, 30]$.

```
svm_tune_grid <- expand.grid(
  C = c(0.001, 0.01, 0.1, 1, 2, 3, 4, 5, 10, 30)
)
```

Fitting the model

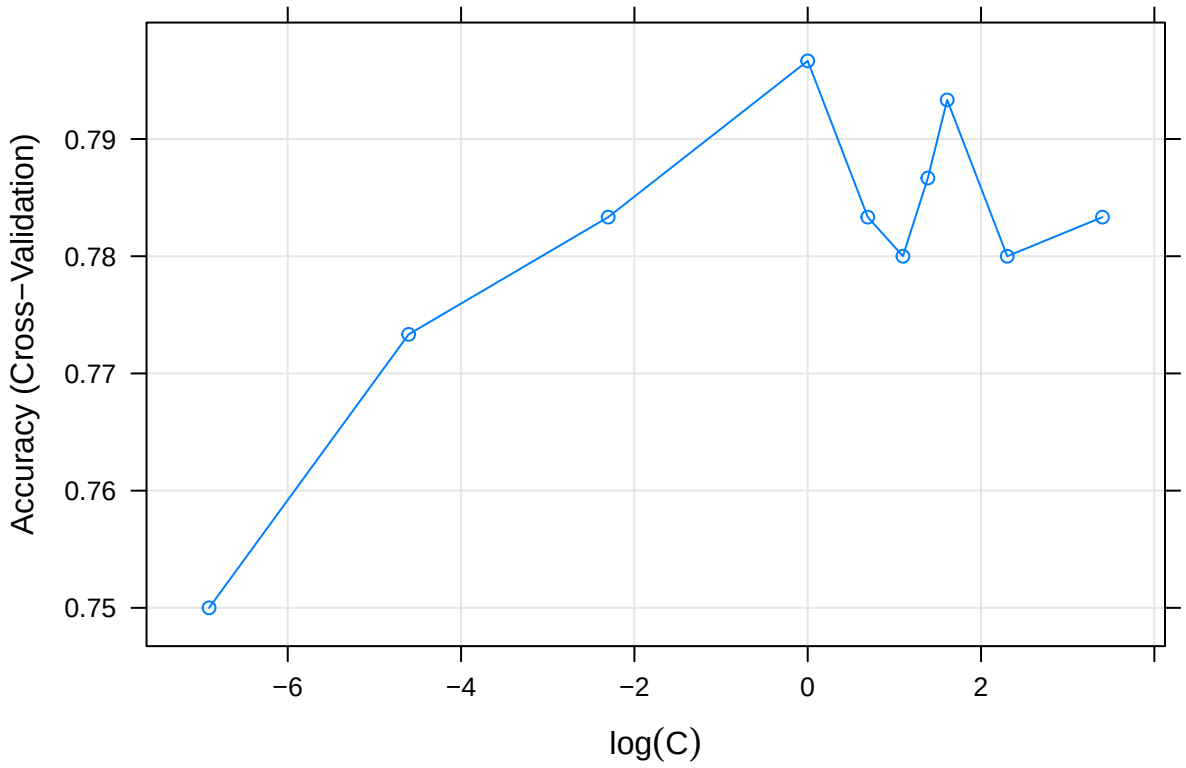
We will now perform the grid search and retrieve the best model.

```
set.seed(0)
svm_model <- caret::train(
  form = diabetes ~ .,
  data = ctrain,
  method = "svmLinear",
  trControl = cross_validation,
  preProcess = c("center", "scale"),
  tuneGrid = svm_tune_grid
)
```

Notice that we decided to standardize the covariates in the `preProcess` parameter. This is important since the euclidean norm is involved in the optimization, and we do not want the covariates with the greater variance to dominate the distance measures.

The result of the line search can now be plotted.

```
svm_plot <- plot(svm_model, xTrans = log)
svm_plot$xlab = expression(log(C))
svm_plot
```



We have here logarithmically transformed the C-axis. $C = 1$ seems to be the tuning parameter that wins out, and this can be verified

```
svm_model$bestTune
```

```
##   C
## 4 1
```

Yes, $C = 1$ is the parameter that we end up choosing.

Model assessment

We can now calculate the confusion matrix of this model against our test set.

```
svm_predictions <- predict(
  svm_model,
  ctest
)
svm_confmat <- confusionMatrix(
  data = svm_predictions,
  reference = ctest$diabetes
)
svm_confmat
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction healthy diabetes
##   healthy      141       33
##   diabetes      14       44
##
##                Accuracy : 0.7974
##                  95% CI : (0.7399, 0.8472)
##     No Information Rate : 0.6681
##     P-Value [Acc > NIR] : 9.374e-06
##
##                   Kappa : 0.513
##
##  Mcnemar's Test P-Value : 0.00865
##
##             Sensitivity : 0.9097
##             Specificity : 0.5714
##          Pos Pred Value : 0.8103
##          Neg Pred Value : 0.7586
##              Prevalence : 0.6681
##          Detection Rate : 0.6078
##    Detection Prevalence : 0.7500
##       Balanced Accuracy : 0.7406
##
##        'Positive' Class : healthy
##
```

The resulting accuracy is $\approx 79.74\%$, with a sensitivity of approximately 91% and specificity of 57%. We therefore have a misclassification rate of $\approx 20\%$. The model is therefore pretty good at predicting diabetes for diabetes patients but it has a relatively high false positive rate. If the downside of not detecting diabetes is much worse than the downside of predicting diabetes for a patient that is really healthy, this could be considered a good model. If the opposite is the case, then it can be considered a bad model.

Module 11: Neural networks

We will now implement a *single-hidden-layer neural network* for the data set.

How it works

Structure of the neural network

The explanation for how a neural network works is best accompanied with an illustration, see figure 7.

The blue x_i nodes on the left hand side in the figure are the so-called *input nodes*. In our case, this is `bmi`, `glu` and so on. The `bias` nodes represent constant intercepts.

The input data is fed into the hidden layer nodes by weighted summation. Each arrow is a term in the sum, with an associated weight α_{jm}. The hidden node applies a *hidden layer activation* function ϕ_h on the input before it sends it forward into the next layer. So the input to the hidden layer node z_i therefore becomes

$$z_i(\boldsymbol{x}) = \phi_h \left(\alpha_{0i} + \sum_{j=1}^{p} \alpha_{ji} x_j \right)$$

We will use the *sigmoid function* as the activation function:

$$\phi_h(x) = \frac{1}{1 + \exp(-x)}$$

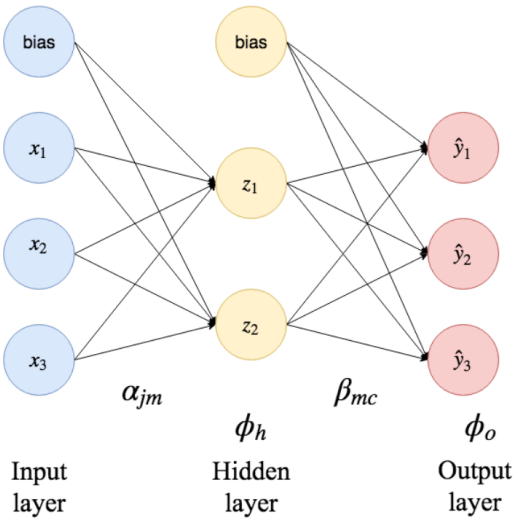


Figure 7: Single-hidden-layer neural network

Since we have a single-hidden-layer network, the hidden nodes feed their output to the *output layer* in the same fashion, together with a new bias node and new weights β_{mc}. The output activation function, ϕ_o, must also be chosen and we use the sigmoid function yet again, i.e. $\phi_o = \phi_h$. In our case, we only have one node in the output layer, as we have a binary classification problem. The value of the output node therefore becomes

$$\hat{y}_1 = \phi_0 \left(\beta_{01} + \sum_{j=1}^{w} \beta_{j1} z_j \right)$$

Here w denotes the number of nodes in the hidden layer, often called the *width* of the neural network. This is a tuning parameter. Generally, you would also be able to add additional hidden layers to the network, increasing its *width*. But the `R`-package `nnet` only support single-hidden-layer networks, so we will not be able to tune this parameter.

The optimization problem

Now, how do we decide the weights of the edges, $\boldsymbol{\theta} := [\alpha\text{'s}, \beta\text{'s}]$? We want to maximize the log-likelihood of the statistical model. We define an objective function, $J(\boldsymbol{\theta})$, a scaled version of the *binomial cross-entropy loss*, which is the negative of the binomial log likelihood in scaled form

$$J(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \left(y_i \ln(\hat{y}_1(\boldsymbol{x}_i)) + (1 - y_i) \ln(1 - \hat{y}_1(\boldsymbol{x}_i)) \right)$$

We seek to find the $\boldsymbol{\theta}$ which minimizes this loss function. In practice, this is done with the `BFGS` method of `optim` by `nnet`.

Additionally, we should mention that `nnet` supports adding a L_2-penalty, λ, to the loss function for regularization purposes. This is similar to what we have seen in ridge regression models earlier. The loss function therefore becomes

$$J(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \left(y_i \ln(\hat{y}_1(\boldsymbol{x}_i)) + (1 - y_i) \ln(1 - \hat{y}_1(\boldsymbol{x}_i)) \right) + \frac{1}{n} \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2$$

In neural networks λ is known as *weight decay*, and this is the second and last tuning parameter in this model.

Parameter selection

We will perform a grid search over the tuning parameter space $\boldsymbol{w} \times \boldsymbol{\lambda} = [1, 2, ..., 11] \times [0, 0.1, ..., 0.5]$ with the same method as explained before.

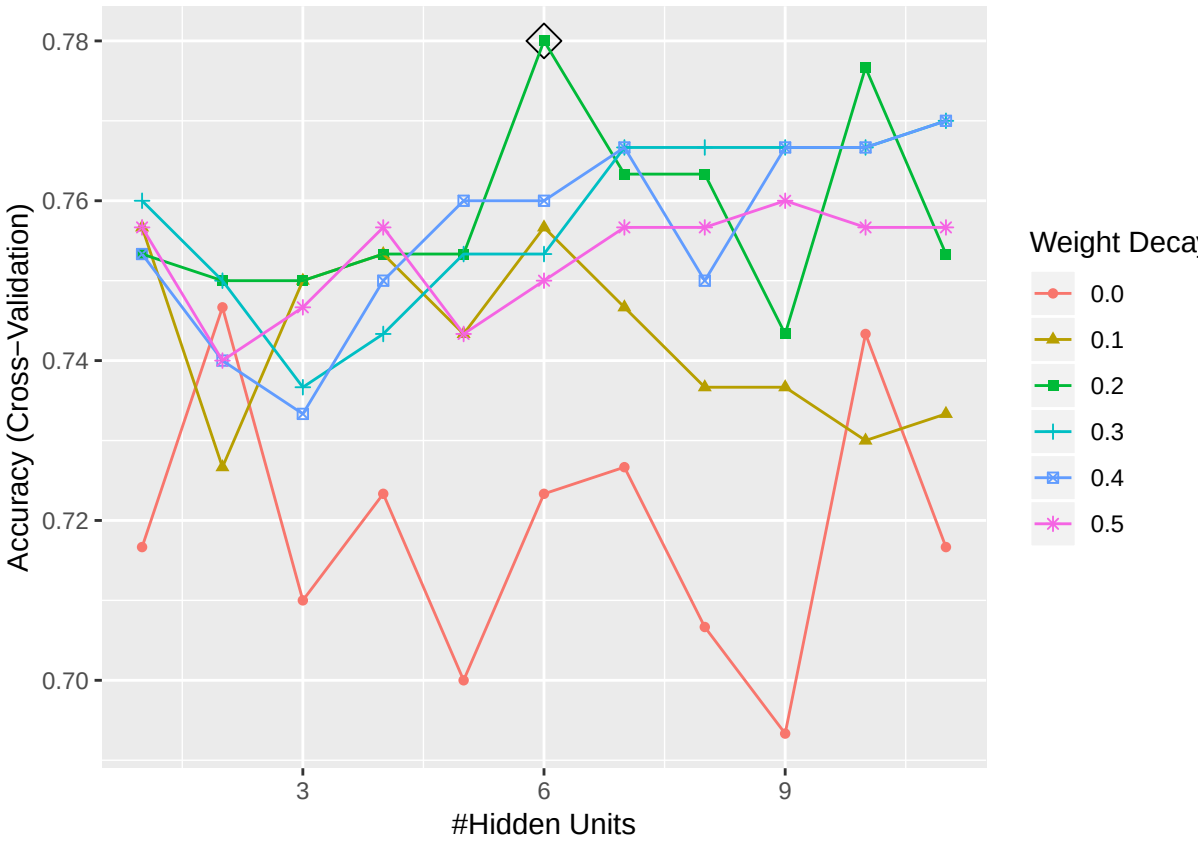
```
nn_tune_grid <- expand.grid(
  size = seq(from = 1, to = 11, by = 1),
  decay = seq(from = 0, to = 0.5, by = 0.1)
)
```

We can now perform this grid search and retrieve the best model. The covariates need not really be centered and scaled, but it often helps with the convergence rate of the optimization algorithm, as explained in this StackOverflow answer.

```
set.seed(42)
nn_model <- caret::train(
  diabetes ~ .,
  data = ctrain,
  method = "nnet",
  preProcess = c("center", "scale"),
  trControl = cross_validation,
  tuneGrid = nn_tune_grid,
  linout = FALSE,
  trace = FALSE
)
```

The result of the grid search algorithm can be plotted.

```
ggplot(nn_model, highlight = TRUE)
```



By inspection we see that 6 hidden nodes with a weight decay of 0.2 is optimal, which can be confirmed:

```
nn_model$bestTune
```

```
##    size decay
## 33    6   0.2
```

The resulting weights of each edge can be printed:

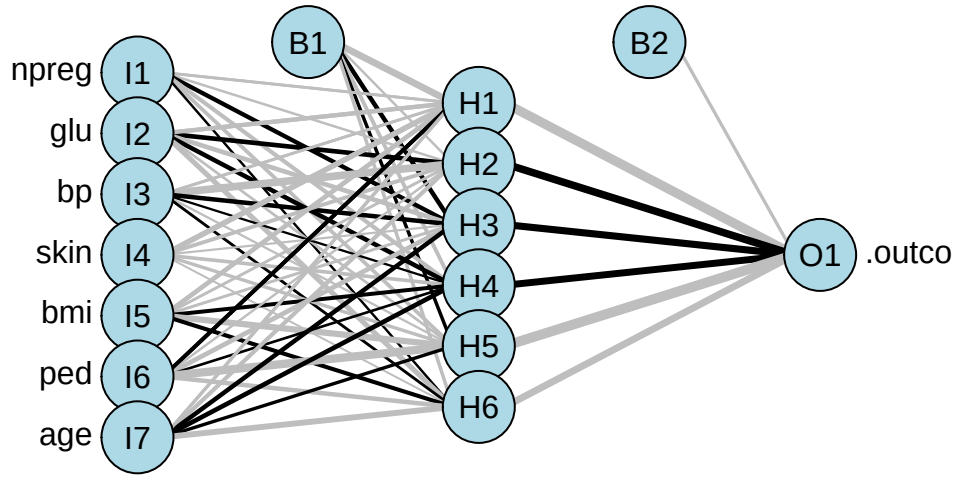
```
summary(nn_model$finalModel)
```

```
## a 7-6-1 network with 55 weights
## options were - entropy fitting  decay=0.2
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1
##  -1.96  -0.34  -0.92  -0.35  -1.31  -0.97   0.77  -0.51
##  b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2
##  -0.22  -0.22   1.07  -2.27  -0.77  -0.39  -0.19  -0.91
##  b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3
##   1.01   0.95  -1.07   1.02  -0.10  -0.41  -1.12   1.02
##  b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4
##  -0.77  -0.87   1.03   0.14  -0.66   0.80   0.28   1.09
##  b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5
##   0.60  -0.73  -0.32  -0.13  -0.58  -1.70  -2.55   0.57
##  b->h6 i1->h6 i2->h6 i3->h6 i4->h6 i5->h6 i6->h6 i7->h6
##  -0.62   0.20  -1.27   0.36  -0.01   0.87  -1.01  -1.58
##  b->o h1->o h2->o h3->o h4->o h5->o h6->o
## -0.54 -2.99  2.22  2.06  2.00 -3.22 -2.02
```

This is hard to parse, so we will use a plot method implemented by the GitHub user `Peque` avaliable here in

order to visualize the neural network:

```
library(devtools)
source_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/85e14f3a292e126f1454864427e3a189c2fe
plot.nnet(nn_model$finalModel)
```



Grey edges represent negative weights, while the black ones represent positive weights. The thickness of each line is proportional to the magnitude of the weights.

We can now test the neural net on our testing data set:

```
nn_probabilities <- predict(nn_model, ctest, type = "prob")
nn_predictions <- predict(nn_model, ctest)
nn_confmat <- confusionMatrix(
  data = nn_predictions,
  reference = ctest$diabetes,
)
nn_confmat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction healthy diabetes
##   healthy      132       31
##   diabetes      23       46
##
##                Accuracy : 0.7672
##                  95% CI : (0.7075, 0.82)
##     No Information Rate : 0.6681
##     P-Value [Acc > NIR] : 0.0006343
##
##                   Kappa : 0.4611
##
##  Mcnemar's Test P-Value : 0.3408032
##
##             Sensitivity : 0.8516
##             Specificity : 0.5974
##          Pos Pred Value : 0.8098
##          Neg Pred Value : 0.6667
##              Prevalence : 0.6681
##          Detection Rate : 0.5690
```

```
##    Detection Prevalence : 0.7026
##       Balanced Accuracy : 0.7245
##
##        'Positive' Class : healthy
##
```

We have a misclassification rate of $\approx 23.3\%$.

Q30: Conclude with choosing a winning method, and explain why you mean that this is the winning method.

Let us gather all the calculated test accuracies, sensitivities, and specificities of our four models.

```
models <- list(
  "logistic" = logistic_confmat,
  "boost" = boost_confmat,
  "svm" = svm_confmat,
  "nnet" = nn_confmat
)

comparison <- tibble(
  method = character(),
  accuracy = double(),
  sensitivity = double(),
  specificity = double()
)

for (model in names(models)) {
  comparison %<>% add_row(
    method = model,
    accuracy = models[[model]]$overall["Accuracy"],
    sensitivity = models[[model]]$byClass["Sensitivity"],
    specificity = models[[model]]$byClass["Specificity"]
  )
}

comparison %>% kable(format = "pandoc", digits = 3)
```

method	accuracy	sensitivity	specificity
logistic	0.797	0.884	0.623
boost	0.784	0.890	0.571
svm	0.797	0.910	0.571
nnet	0.767	0.852	0.597

We can now see that the logistic and svm models have comparable accuracies of $\approx 79.7\%$, but with different sensitivities and specificities. Which model we choose as the "best" model depends on multiple factors and which objectives we want to meet. For instance, if early detection of diabetes is paramount, we would prefer high sensitivity at the cost of lower specificity. With this in mind, the `svm` model might be preferable.

But we choose the logistic regression model to be the "winning model" as it allows for easier inference. The model can be used to explain which factors that are correlated to diabetes, and you can say things such as "the odds of getting diabetes is expected to increase $X\%$ for every additional `BMI` point, everything else being equal".