Lecture 8: Logistic regression

Matti Pirinen

25.8.2023

In this lecture, we study a relationship between a binary variable, such as a disease status, and one or more predictors, such as risk factors of a disease. We extend the linear regression model to binary outcomes by a modeling approach called **logistic regression**.

Modeling risk for binary outcomes through log-odds

When the outcome Y of an experiment/observational study is binary (can take only two values, denoted by 0 and 1), we typically want to model how the probability p_i, that the outcome is 1 for individual i, depends on the available knowledge of the values of predictor variables $X_1, \ldots, X_k$ for individual i. We call this probability "risk" although in general it can be any probability, not necessarily related to a disease risk.

Intuitively, extending our experience from the linear regression model, we would like to have a model for risk like

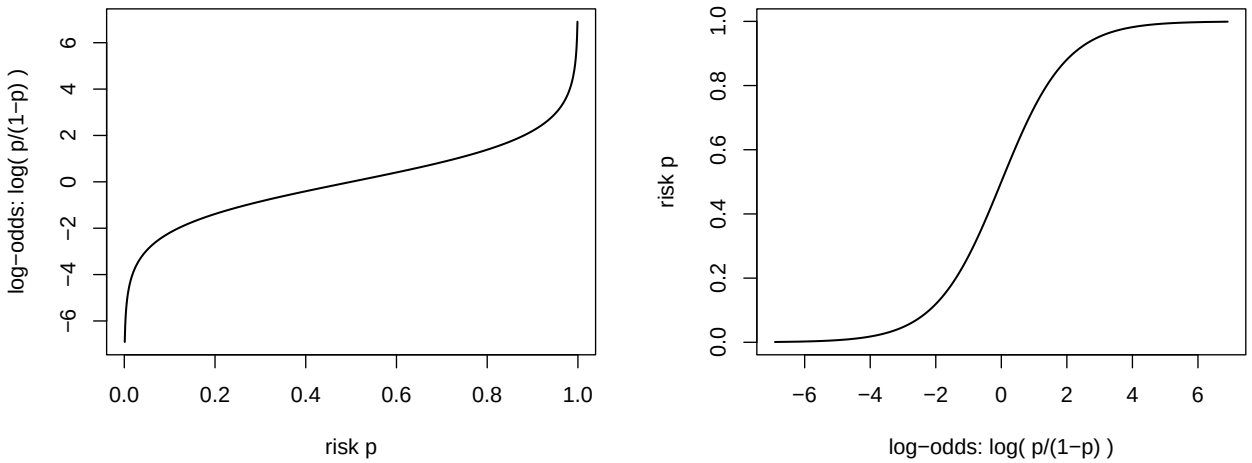
$$p_i = a + b\, x_i + \varepsilon_i,$$

from which we could estimate how a change in the predictor value x_i changes the risk p_i. This has two problems. First, we can't measure p_i but we only observe binary outcome y_i which is either 0 or 1 (e.g. for the patients we set $y_i = 1$ and for the healthy we set $y_i = 0$). Second, the risk value p_i is a probability and hence always between 0 and 1, whereas the linear model above doesn't impose such a restriction to the right hand side of the equation, which could result in inacceptable risk values from the model.

In this lecture, we leave the first problem for R to handle. The second problem is solved by transforming the outcome value p_i from the interval (0,1) to the whole real line by the **log-odds transformation**, where p_i is replaced by

$$\log\left(\frac{p_i}{1-p_i}\right).$$

```
par(mfrow = c(1,2))
p = seq(0.001, 0.999, 0.001)
y = log( p/(1-p) )
plot(p, y, xlab = "risk p", ylab = "log-odds: log( p/(1-p) )", t = "l", lwd = 1.5)
plot(y, p, xlab = "log-odds: log( p/(1-p) )", ylab = "risk p", t = "l", lwd = 1.5)
```



When p_i is the probability that an event of interest happens, then $1 - p_i$ is the probability that the event does not happen. The **odds** of the event are $p_i/(1 - p_i)$ and they tell how many times more likely the event is to happen than not to happen. For example, if risk is 50%, then the odds are 1:1 = 1 and if risk is 2/3 then the odds are 2:1 = 2. Large odds (and also large log-odds) correspond to very probable events and small odds (and also small log-odds) correspond to very unlikely events. Mathematically, the inverse of the log-odds transformation

$$\ell = \log\left(\frac{p}{1-p}\right) \qquad \text{is} \qquad p = \frac{\exp(\ell)}{1+\exp(\ell)}.$$

By taking the logarithm of odds, we have a quantity that can take any real number as its value. Hence, it is conceptually valid to use a regression model where

$$\log\left(\frac{p_i}{1-p_i}\right) = a + b\,x_i.$$

If b in this model is positive, then, as x increases, also the odds of the event increases and therefore also the probability of the event increases. If b is negative, then as x increases, also the odds of the event and hence also the probability of the event decreases. With the logistic regression model, for each pair of estimates of a and b, we can do a prediction of risk p_i for individual i given his/her value for predictor x_i.

Example 8.1

1. Risk of a disease is 10%. What are the odds of the disease? And log-odds of the disease? How does odds and log-odds change when risk changes from 10% to (100% - 10%) = 90%?

Odds are $p/(1-p) = 0.1/(1-0.1) = 0.1/0.9 = 1/9 \approx 0.111$. Log-odds are $\log(1/9) = -2.197$. If risk changes to 90%, then odds become $9/1 = 9$ and log-odds become $\log(9) = -\log(1/9) = 2.197$. Thus odds change to their inverse value, and log-odds changes the sign.

2. Suppose that log-odds of a disease follows a linear model $a + b\,x$, where x is measured BMI and parameter values are $a = -1$ and $b = 0.01$. Is increasing BMI a risk or protective factor for the disease? What is the risk of disease for an individual with bmi = 30? What about bmi = 40?

Since larger log-odds mean larger odds mean larger risk, a positive value of b means that increasing BMI increases the risk.

```r
a = -1
b = 0.01
x = c(30, 40)
logodds = a + b*x
p = exp(logodds)/(1 + exp(logodds))
data.frame(bmi = x, risk = p)
```

```
##   bmi      risk
## 1  30 0.3318122
## 2  40 0.3543437
```

Logistic regression in R

Let's use a dataset `Pima.tr` on 200 women from Pima population (a Native American population) from `MASS` package.

```r
library(MASS)
y = Pima.tr
str(y)
```

```
## 'data.frame':    200 obs. of  8 variables:
##  $ npreg: int  5 7 5 0 0 5 3 1 3 2 ...
##  $ glu  : int  86 195 77 165 107 97 83 193 142 128 ...
##  $ bp   : int  68 70 82 76 60 76 58 50 80 78 ...
##  $ skin : int  28 33 41 43 25 27 31 16 15 37 ...
##  $ bmi  : num  30.2 25.1 35.8 47.9 26.4 35.6 34.3 25.9 32.4 43.3 ...
##  $ ped  : num  0.364 0.163 0.156 0.259 0.133 ...
##  $ age  : int  24 55 35 26 23 52 25 24 63 31 ...
##  $ type : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 1 1 2 ...
```

This data frame contains the following columns:

Variable	Explanation
npreg	number of pregnancies
glu	plasma glucose concentration in an oral glucose tolerance test
bp	diastolic blood pressure (mm Hg)
skin	triceps skin fold thickness (mm)
bmi	body mass index
ped	diabetes pedigree function
age	age in years
type	Yes or No, for diabetic according to WHO criteria

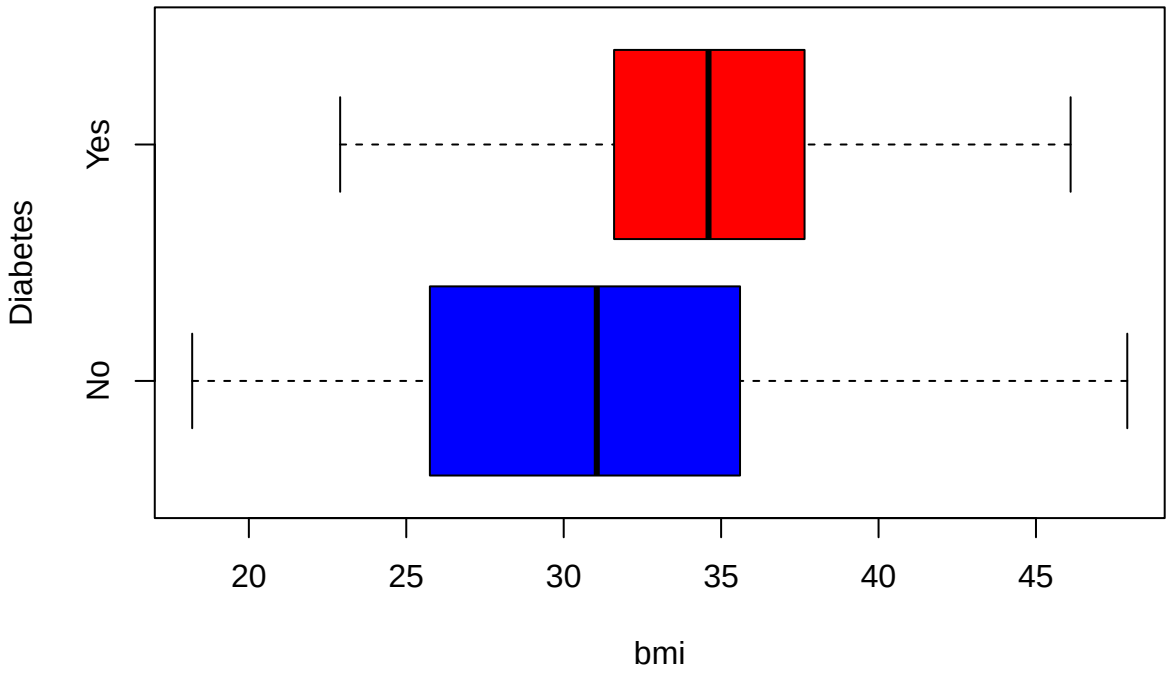
Our outcome variable is the binary diabetes status `type`.

```r
table(y$type)
```

```
##
##  No Yes
## 132  68
```

Let's start by seeing how `bmi` associates to `type` using a boxplot and a t-test. Since `type` is a factor (see above output from `str()`), we can make boxplots and the t-test of `bmi` with respect to the two levels of the factor by using the notation `bmi ~ type`.

```
boxplot(bmi ~ type, data = y, xlab = "bmi", ylab = "Diabetes",
        horizontal = TRUE, col = c("blue", "red")) #turns boxplot horizontal
```




```
t.test(bmi ~ type, data = y)
```

```
##
##  Welch Two Sample t-test
##
## data:  bmi by type
## t = -4.512, df = 171.46, p-value = 1.188e-05
## alternative hypothesis: true difference in means between group No and group Yes is not equal to 0
## 95 percent confidence interval:
##  -5.224615 -2.044547
## sample estimates:
##  mean in group No mean in group Yes
##          31.07424          34.70882
```

There seems to be a difference of about 3.5 kg/m^2, 95%CI (2.0..5.2) between the groups.

If we are given a value of BMI, what would be a risk of diabetes (in these data)? Let's use logistic regression. We use `glm( )` function that fits "generalized linear models", and we specify logistic regression by parameter `family = "binomial"`. This means that the outcome variable in regression is modeled as having a binomial distribution, where the success probability may depend on the values of predictors. For the model formula, the syntax is similar to the linear model `lm()`.

Important: If you forget to specify `family = "binomial"` in `glm( )`, then the function fits a linear model, not a logistic model!

```r
glm.1 = glm(type ~ bmi, data = y, family = "binomial")
summary(glm.1) #Always check from summary that you have fitted the correct model
```

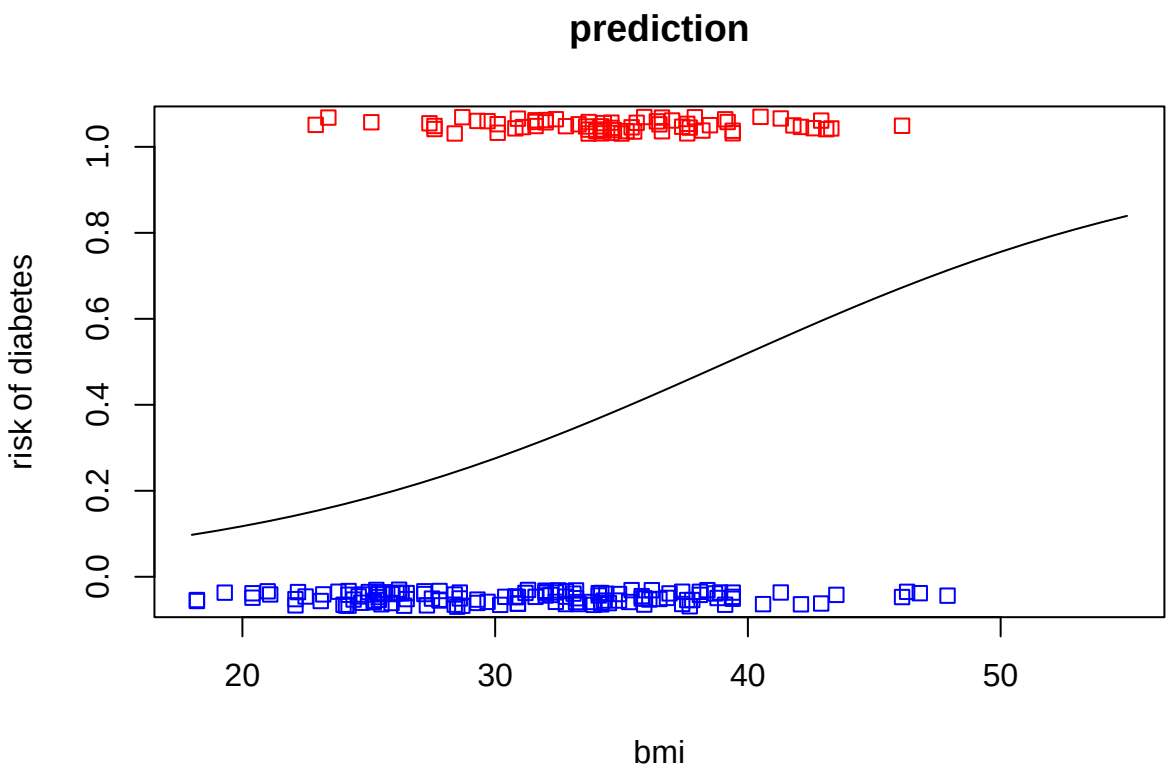
```
##
## Call:
## glm(formula = type ~ bmi, family = "binomial", data = y)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.11156    0.92806  -4.430 9.41e-06 ***
## bmi          0.10482    0.02738   3.829 0.000129 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 256.41  on 199  degrees of freedom
## Residual deviance: 239.97  on 198  degrees of freedom
## AIC: 243.97
##
## Number of Fisher Scoring iterations: 4
```

Coefficient for `bmi` is the increase in log-odds of diabetes per one unit increase in `bmi` and shows a clear statistical association with P ~ 1e-4. Let's demonstrate what this means in terms of the risk of diabetes by applying `predict( )` function for a sequence of `bmi` values. The default output from `predict( )` on logistic regression object is in log-odds but we can ask the prediction directly as probabilities by specifying `type = "response"`.

```r
bmi.grid = 18:55
data.in = data.frame(bmi = bmi.grid)
y.pred = predict(glm.1, newdata = data.in, type = "response") #type = "response" predicts probabilities
```

Let's plot these predictions as a function of bmi, and let's add to the figure also the bmi distributions of the observed cases and controls of diabetes using `stripchart()`.

```r
plot(bmi.grid, y.pred, ylim = c(-0.05,1.05), t = "l",
     main = "prediction", ylab = "risk of diabetes", xlab = "bmi")
stripchart(y$bmi[y$type == "No"], at = -0.05, add = T, method = "jitter", jitter = 0.02, col = "blue")
stripchart(y$bmi[y$type == "Yes"], at = 1.05, add = T, method = "jitter", jitter = 0.02, col = "red")
```



We drew the predictions up to `bmi = 55`, but we see that original data included few observations above 45, and hence we shouldn't expect the model to be reliable after that. We see that the risk raises from about 10% at `bmi < 20` up to 60% at `bmi > 40`. While the relationship in this BMI range is close to linear, we still see that there is some non-linearity in the prediction curve, which is a difference from the corresponding linear regression model.

Is this a good model? Let's define some measures for how well the model explains the data. We use **misclassification error** and **ROC curve** for this purpose.

Missclassification error We can predict the class (0 or 1) of each sample based on the model, for example, by predicting the class to be 1 if the predicted risk > 0.5 and otherwise predicting the class to be 0, and then see which proportion of the predictions are wrong. This proportion of wrong predictions is called the missclassification rate. Let's apply it to our current data. We can use `predict(glm.1, type = "response")` to produce risks and then threshold them into 0 or 1 according to whether the risk is < 0.5 or > 0.5.

```
pred.1 = predict(glm.1, type = "response") #if newdata is not specified, then predictions are given for
y.pred.1 = ifelse(pred.1 < 0.5, 0, 1) #if pred.1[i] < 0.5 then sets y.pred.1[i]=0 else sets y.pred.1[i]
table(y.pred.1, y$type) #cross-tabulate predictions vs. true type
```

```
##
## y.pred.1  No Yes
##        0 120  57
##        1  12  11
```

We see that missclassification error is $\frac{57+12}{200} = 0.345$. Two measures often used in medical research are

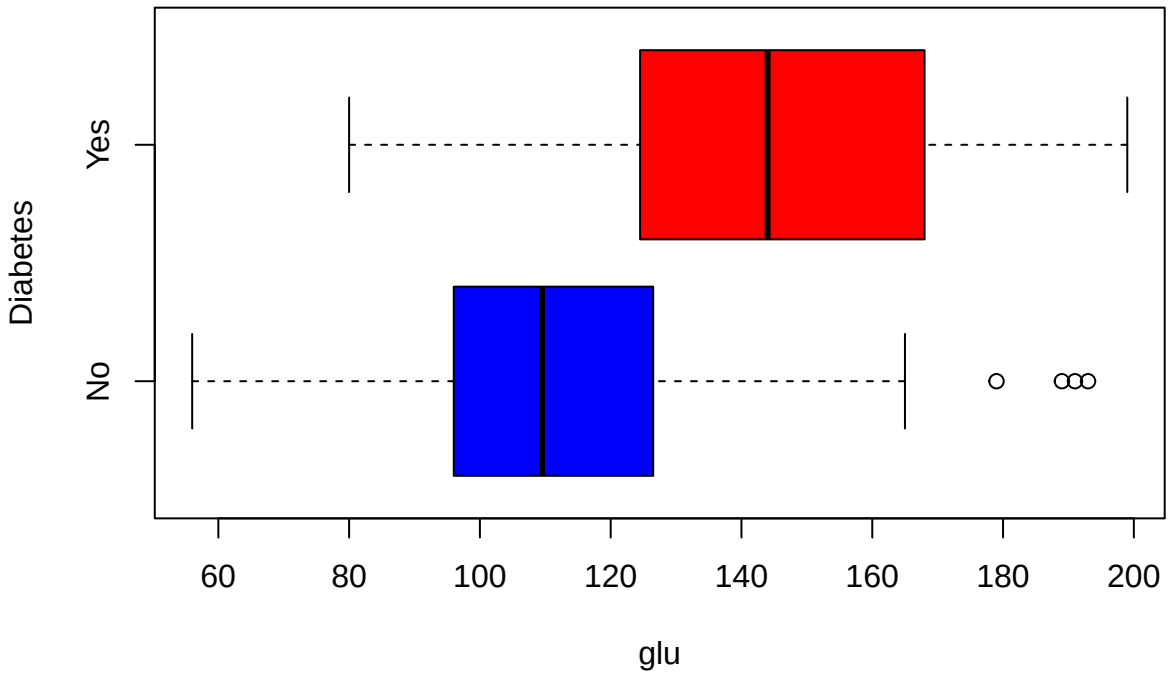
- *sensitivity*, the proportion of true 1s that are correctly labeled as 1s: $\frac{11}{57+11} = 0.162$ and

- *specificity*, the proportion of true 0s that are correctly labeled as 0s: $\frac{120}{120+12} = 0.909$.

The classifier is perfect if and only if it has both sensitivity and specificity equal to 1. Here our model has a very low sensitivity of 16% and a higher specificity of 91%. Low sensitivity means that the model doesn't really detect well the true diabetes cases.

Let's add `glu` to the model. It is expected to predict strongly diabetes, since it is used to diagnose diabetes.

```
boxplot(glu ~ type, data = y, xlab = "glu", ylab = "Diabetes", horizontal = T, col = c("blue","red"))
```




```
t.test(glu ~ type, data = y)
```

```
##
##  Welch Two Sample t-test
##
## data:  glu by type
## t = -7.3856, df = 121.76, p-value = 2.081e-11
## alternative hypothesis: true difference in means between group No and group Yes is not equal to 0
## 95 percent confidence interval:
##  -40.51739 -23.38813
## sample estimates:
##  mean in group No mean in group Yes
##          113.1061          145.0588
```

```
glm.2 = glm(type ~ bmi + glu, data = y, family = "binomial")
summary(glm.2)
```

```
##
## Call:
## glm(formula = type ~ bmi + glu, family = "binomial", data = y)
```

```
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.216106   1.346965  -6.100 1.06e-09 ***
## bmi          0.090016   0.031268   2.879  0.00399 **
## glu          0.035716   0.006311   5.659 1.52e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 256.41  on 199  degrees of freedom
## Residual deviance: 198.47  on 197  degrees of freedom
## AIC: 204.47
##
## Number of Fisher Scoring iterations: 4
```

Let's see the missclassification rate in the updated model.

```
pred.2 = predict(glm.2, type = "response")
y.pred.2 = ifelse(pred.2 < 0.5, 0, 1)
table(y.pred.2, y$type) #cross-tabulate predictions vs. true type
```

```
##
## y.pred.2  No Yes
##        0 116  31
##        1  16  37
```

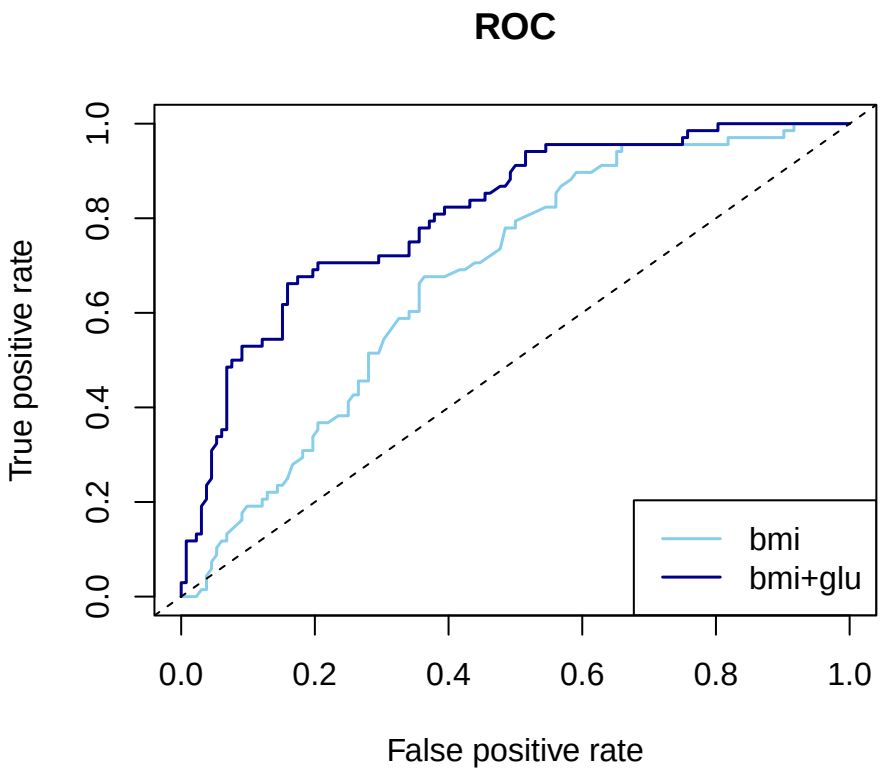
Missclassification rate is now $\frac{31+16}{200} = 0.235$ and has clearly fallen from its earlier value of 0.345 when BMI was alone in the model.

Missclassification rate is an intuitive measure, but it has a downside that it is defined by a particular fixed risk threshold (usually 0.5). We can get more information about a classifier, if we can see its performance throughout all possible risk cutoffs. The ROC curve gives us such information.

ROC curve In *receiver operating characteristics* (ROC) curve, we run through all possible risk thresholds to classify samples into 1s and 0s and, for each threshold, we compute the true positive rate (= sensitivity, the proportion of true 1s that are labeled as 1s) against false positive rate (=1-specificity, the proportion of true 0s that are labeled as 1s). Let's draw ROC curves from models 1 and 2 above, and discuss the curves.

```
p.thresh = seq(-0.001, 1.001, 0.001) #risk thresholds for which ROC is computed
#We make roc.1 and roc.2 matrices here and then fill them in the for-loop below
roc.1 = matrix(NA, ncol = 2, nrow = length(p.thresh)) #will have col1 = FP rate and col2 = TP rate
roc.2 = matrix(NA, ncol = 2, nrow = length(p.thresh))

for(ii in 1:length(p.thresh)){ #goes through all risk thresholds

  p = p.thresh[ii] # now p is the current risk threshold

  y.pred.1 = ifelse(pred.1 < p, 0, 1) #0-1 predictions based on current risk threshold 'p'
  y.pred.2 = ifelse(pred.2 < p, 0, 1)

  roc.1[ii,1] = sum( y$type == "No" & y.pred.1 == 1) / sum( y$type == "No" ) #false positive rate
```

```
  roc.1[ii,2] = sum( y$type == "Yes" & y.pred.1 == 1) / sum( y$type == "Yes") #true positive rate

  roc.2[ii,1] = sum( y$type == "No" & y.pred.2 == 1) / sum( y$type == "No" )
  roc.2[ii,2] = sum( y$type == "Yes" & y.pred.2 == 1) / sum( y$type == "Yes")
}

plot(NULL, xlim = c(0,1), ylim = c(0,1), main = "ROC",
     xlab = "False positive rate", ylab = "True positive rate") #Makes an empty plot
lines(roc.1, col = "skyblue", lwd=1.5) #adds ROC of model 1
lines(roc.2, col = "darkblue", lwd=1.5) #adds ROC of model 2

legend("bottomright", leg = c("bmi", "bmi+glu"), col = c("skyblue", "darkblue"), lwd = 1.5)
abline(a = 0, b = 1, lty = 2) #adds diagonal line Y = X
```



An ROC curve always starts from the origin (0,0), corresponding to the risk threshold of 1.0, at where no sample is yet classified as outcome 1 (because all estimated risks are below 1.0). If the two groups were completely separable in their estimated risk values, then ROC curve would climb straight up to the point (0,1) at the threshold that for the first time completely separated the two groups. From there, it would then move horizontally towards the point (1,1) as the risk thresholds get closer to 0.0. It would reach the point (1,1) at latest when the risk threshold reaches 0.0 because then every observation is classified as 1. If the risk estimates were just some random values, we would expect the ROC curve to follow closely the diagonal.

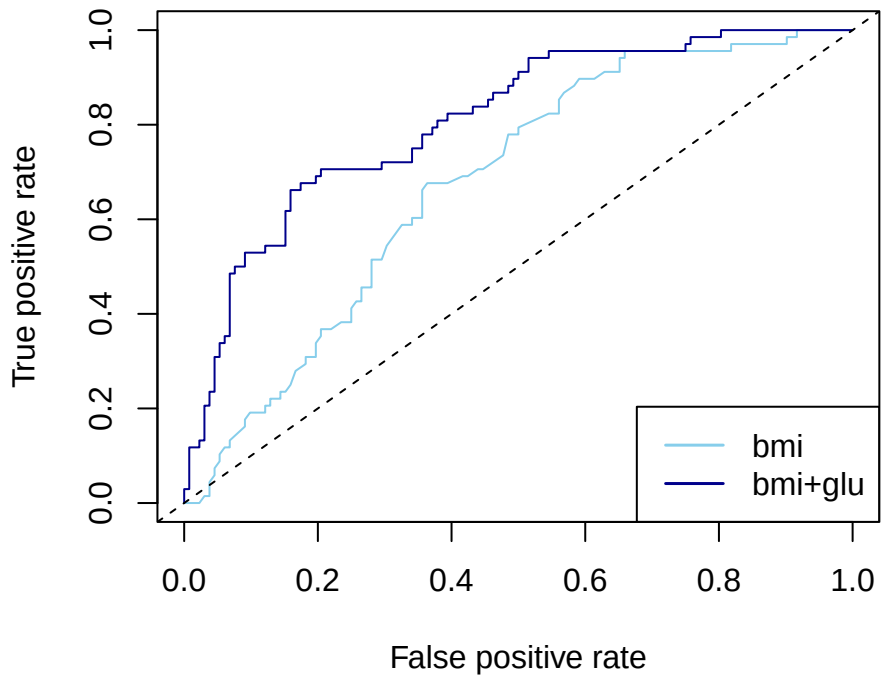
As a conclusion, the better the classifier is, more quickly its ROC jumps up vertically and more slowly it moves horizontally. Another way to express this is that we would like our classifier to have simultaneously a high true positive rate AND a low false positive rate and such classifiers reside near the top left corner of the ROC plot. Mathematically, it can be shown that the **area under the curve (AUC)** of an ROC curve corresponds to the probability that a randomly chosen case (a true 1) has a higher risk estimate than a randomly chosen control (a true 0). Thus, AUC is a single number that summarizes the performance of a classifier.

Above we see that both classifiers seem better than random (they are clearly above the diagonal) and the model with `glu` is a better classifier since it is above the other curve and hence also covers more area under the curve.

See illustration of ROC curves: https://kennis-research.shinyapps.io/ROC-Curves/.

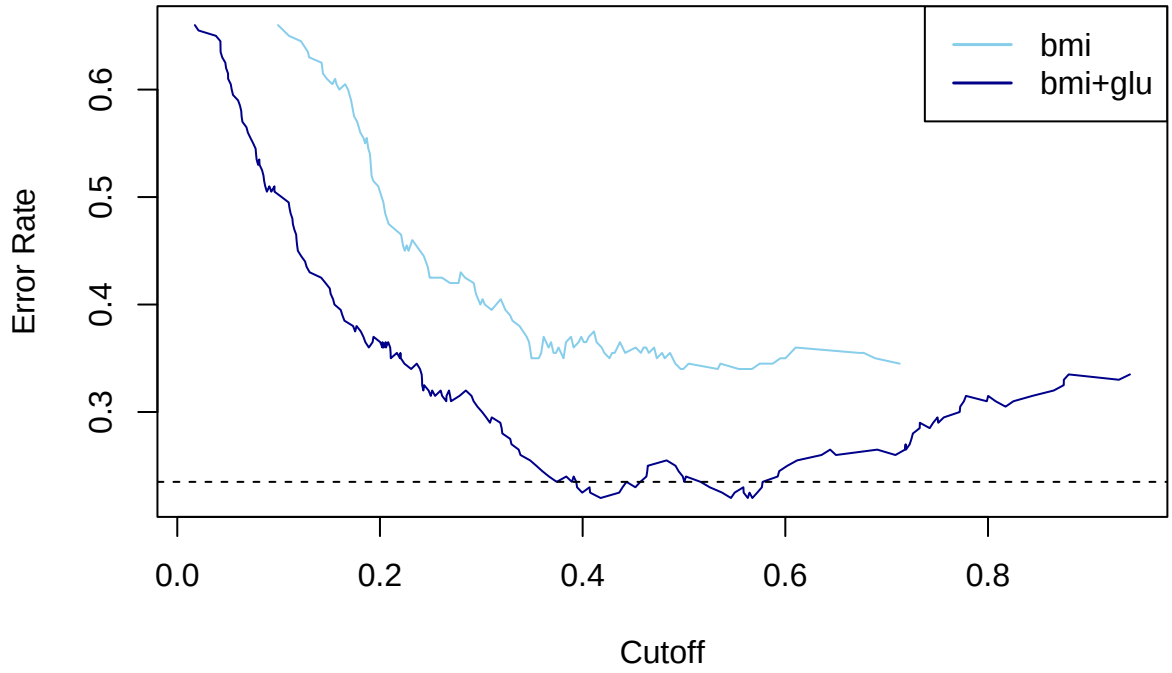
ROCR package Let's do the same tasks with the `ROCR` package (install if first, see the code below). We need two vectors for each ROC curve: `predictions` which in logistic regression gives the predicted risk, and `labels` which gives the true 0/1 labels. These are given to the `predictions( )` function to produce an ROCR object for performance evaluation. We can then assess `performance( )`, for example, by making an ROC curve using `measure = "tpr"` and `x.measure = "fpr"`.

```
#install.packages("ROCR")
library(ROCR)
pred.1 = predict(glm.1, type = "response") #repeat risk predictions from model glm.1
rocr.pred.1 = prediction(pred.1, labels = y$type) #ROCR prediction object
roc.perf.1 = performance(rocr.pred.1, measure = "tpr", x.measure = "fpr") #ROCR performance object
plot(roc.perf.1, col = "skyblue")

pred.2 = predict(glm.2, type = "response")
rocr.pred.2 = prediction(pred.2, labels = y$type)
roc.perf.2 = performance(rocr.pred.2, measure = "tpr", x.measure = "fpr")
plot(roc.perf.2, col = "darkblue", add = TRUE) #add to existing plot

abline(a = 0, b = 1, lty = 2) #diagonal corresponding to a random assignment
legend("bottomright", leg = c("bmi", "bmi+glu"), col = c("skyblue", "darkblue"), lwd = 1.5)
```



We can also get the missclassification rate using `ROCR` by asking `measure = "err"`. Let's plot it as a function of the risk cutoff and mark the value 0.235, that we computed manually above for `glm.2` model at the cutoff of 0.5.

```r
plot(performance(rocr.pred.2, measure = "err"), col = "darkblue")
plot(performance(rocr.pred.1, measure = "err"), col = "skyblue", add = T)
legend("topright", leg = c("bmi", "bmi+glu"), col = c("skyblue", "darkblue"), lwd = 1.5)
abline(h = 0.235, lty = 2)
```



We see that the smallest missclassification rates occur around the risk threshold region (0.4,. . . ,0.6).

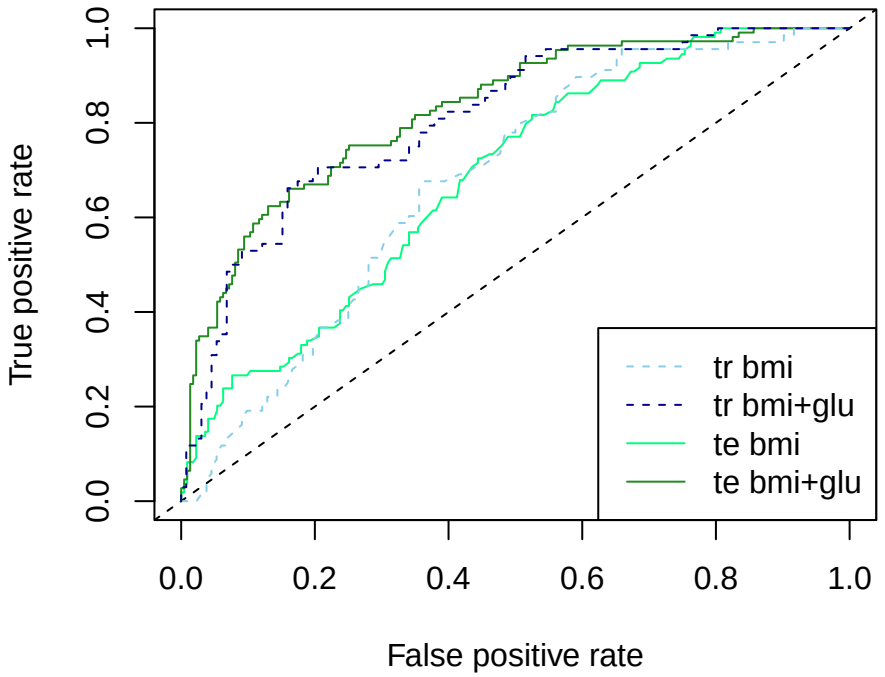
Test data and cross-validation

Our model assessment above has a potential problem because we first fitted the regression model in the data set `y` and then we tested how well the model was doing using the **same** data set `y`. This procedure may yield an overestimation of the performance of the model compared to what the performance would be in some **different** test data set. This is because the method to fit the model maximizes the fit of the model to the available data set, also called the **training data** set since it is used for "training" the model parameters. The same model could, however, perform much worse in an unseen **test data** set, because the test data was not available to fit the model in the first place. The severity of this **overfitting problem** depends on the context and is likely very small in our case where the model has only a couple of predictors, but it could become very severe, if we had used hundreds of predictors in the regression model. Therefore, in general, a reliable performance assessment of a model should be done in an independent test data set that was not used when the model was trained. This principle of separate training and test data is a cornerstone of modern data science and machine learning applications.

Luckily, we have an independent test data `Pima.te` that contains 332 more women from the same Pima population that we used above as our training data. Let's see how our models perform in the test data.

```r
library(ROCR)
pred.te.1 = predict(glm.1, newdata = Pima.te, type = "response") #.te = "test"
rocr.pred.te.1 = prediction(pred.te.1, labels = Pima.te$type)
roc.perf.te.1 = performance(rocr.pred.te.1, measure = "tpr", x.measure = "fpr")
plot(roc.perf.te.1, col = "springgreen")
```

```r
pred.te.2 = predict(glm.2, newdata = Pima.te, type = "response")
rocr.pred.te.2 = prediction(pred.te.2, labels = Pima.te$type)
roc.perf.te.2 = performance(rocr.pred.te.2, measure = "tpr", x.measure = "fpr")
plot(roc.perf.te.2, col = "forestgreen", add = T)

abline(a = 0, b = 1, lty = 2)
plot(roc.perf.1, col = "skyblue", add = T, lty = 2)
plot(roc.perf.2, col = "darkblue", add = T, lty = 2)

legend("bottomright", legend = c("tr bmi","tr bmi+glu","te bmi","te bmi+glu"),
       col = c("skyblue","darkblue","springgreen","forestgreen"), lty = c(2,2,1,1))
```



The ROC curves show that the performance is fairly similar in the training and test data, which tells that we have not overfitted our model in this example but the model performance generalizes as such also to unseen data from the same population.

It is important to be able to test the model performance in test data that are independent from the training data. But in the same time it seems wasteful to put aside a large proportion of the available data for testing purposes since then we can't use all available information to fit the model. To tackle this problem, we can use the technique of **cross-validation**.

Cross-validation In cross-validation, we split the data set into K parts (typically $K = 5$ or $K = 10$). Then we fit the target model K separate times, by using one of the K parts as test data and the rest as training data. Finally, we average the performance over the K different test data results. This procedure gives a reliable estimate about how the particular model would perform in independent test data, but in the same time we have used all the data to inform the model fitting. We would then choose the model that performs the best in the cross-validated performance metric and fit that model using the whole data set to serve as our model of choice. If you are interested to learn more, watch https://www.youtube.com/watch?v=fSytzGwwBVw.

Example 8.2: Logistic regression on biopsy data

Here we consider `biopsy` data set from `MASS` package.

We aim to develop a logistic regression model in order to assist the patient's medical team in determining whether the tumor is malignant or not. This breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. He assessed biopsies of breast tumours for 699 patients up to 15 July 1992; each of nine attributes has been scored on a scale of 1 to 10, and the outcome is also known.

Name	Explanation
ID	sample code
thick	clump thickness
u.size	uniformity of cell size
u.shape	uniformity of cell shape
adhsn	marginal adhesion
s.size	single epithelial cell size
nucl	bare nuclei (16 values are missing)
chrom	bland chromatin
n.nuc	normal nucleoli
mit	mitoses
type	"benign" or "malignant"

```
library(MASS)
y = biopsy #working copy to y
names(y) = c("ID","thick", "u.size", "u.shape", "adhsn", "s.size", "nucl", "chrom", "n.nuc", "mit", "ty
str(y)
```

```
## 'data.frame':    699 obs. of  11 variables:
##  $ ID     : chr  "1000025" "1002945" "1015425" "1016277" ...
##  $ thick  : int  5 5 3 6 4 8 1 2 2 4 ...
##  $ u.size : int  1 4 1 8 1 10 1 1 1 2 ...
##  $ u.shape: int  1 4 1 8 1 10 1 2 1 1 ...
##  $ adhsn  : int  1 5 1 1 3 8 1 1 1 1 ...
##  $ s.size : int  2 7 2 3 2 7 2 2 2 2 ...
##  $ nucl   : int  1 10 2 4 1 10 10 1 1 1 ...
##  $ chrom  : int  3 3 3 3 3 9 3 3 1 2 ...
##  $ n.nuc  : int  1 2 1 7 1 7 1 1 1 1 ...
##  $ mit    : int  1 1 1 1 1 1 1 1 5 1 ...
##  $ type   : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
```

We will remove ID column and we will remove 16 samples with some missing data using `na.omit()`. The working data are assigned to `y`.

```
y = na.omit(y) #remove rows with missing observations
y$ID = NULL #remove ID column
head(y)
```

```
##   thick u.size u.shape adhsn s.size nucl chrom n.nuc mit      type
## 1     5      1       1     1      2    1     3     1   1    benign
## 2     5      4       4     5      7   10     3     2   1    benign
## 3     3      1       1     1      2    2     3     1   1    benign
```

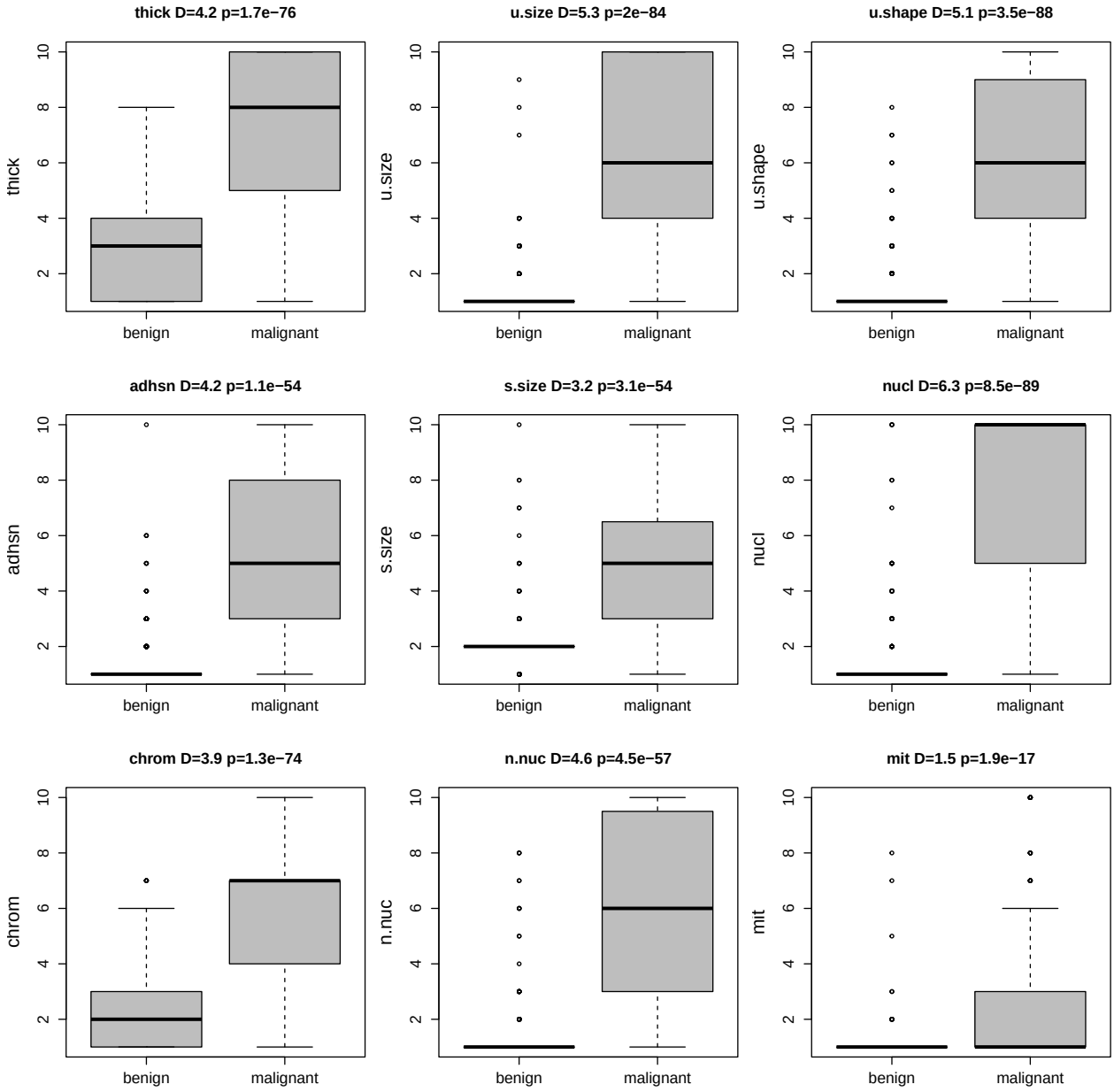
```
## 4     6      8       8     1      3    4     3     7   1    benign
## 5     4      1       1     3      2    1     3     1   1    benign
## 6     8     10      10     8      7   10     9     7   1 malignant
```

```
nrow(y)
```

```
## [1] 683
```

Let's visualize the variables.

```
par(mfrow = c(3,3)) #3x3 plotting area for 9 variables
par(mar = c(3,4,4,1)) #crop a bit away from default margins for each figure
for(ii in 1:9){
  tt = t.test(y[,ii] ~ y$type)
  boxplot(y[,ii] ~ y$type, xlab = "", ylab = names(y)[ii],
          cex.lab = 1.5, cex.main = 1.3, cex.axis = 1.2, col ="gray",
          main = paste0(names(y)[ii]," D=",signif(tt$estimate[2]-tt$estimate[1],2),
                        " p=",signif(tt$p.value,2)))
}
```

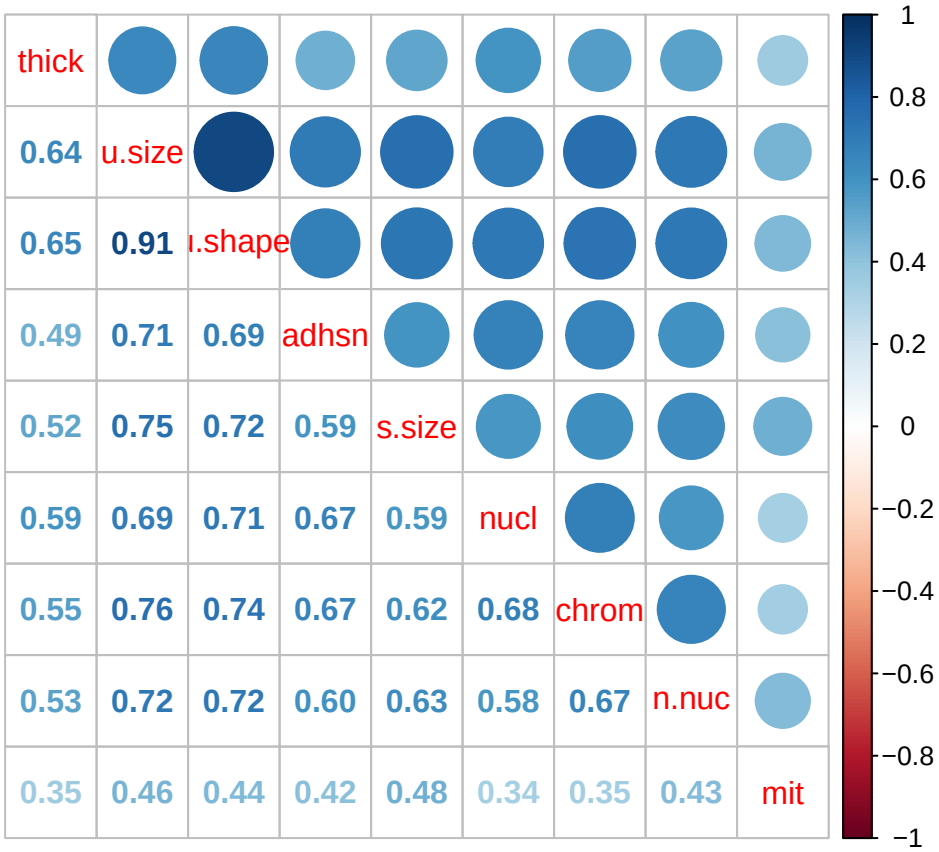


We see that for all 9 variables there is a clear difference between malignant and benign cases. Let's see the correlations between the 9 variables.

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
corr = cor(y[,1:9])
corrplot.mixed(corr)
```



Quite strong positive correlations among the variables.

Let's split the data to 30% for testing and 70% for training. It is important to do the split at random since the order of the samples in the data frame may be associated with some of the variables whereas we want that our training and test data are similar to each other with respect to the distribution of every variable.

```
ind = sample(1:nrow(y), size = round(0.3*nrow(y))) #random sample from 1,...,683 of size 0.3*683
y.tr = y[-ind,] #70% for training
y.te = y[ind,] #30% for testing
```

Let's fit the model with all variables. We can use dot "." to denote using all variables except the outcome variable as predictors.

```
glm.full = glm(type ~ ., data = y.tr, family = "binomial") # . uses all variables as predictors (except
summary(glm.full)
```

```
##
## Call:
## glm(formula = type ~ ., family = "binomial", data = y.tr)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.4784     1.5456  -6.780  1.2e-11 ***
## thick         0.6324     0.1987   3.183  0.00146 **
## u.size        0.2110     0.3312   0.637  0.52404
## u.shape       0.3769     0.3601   1.047  0.29525
```

```
## adhsn         0.2052     0.1637   1.254  0.20996
## s.size       -0.1999     0.2189  -0.913  0.36104
## nucl          0.4102     0.1283   3.197  0.00139 **
## chrom         0.4545     0.2235   2.034  0.04195 *
## n.nuc         0.3750     0.1752   2.141  0.03229 *
## mit           0.4055     0.4499   0.901  0.36737
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 624.576  on 477  degrees of freedom
## Residual deviance:  56.585  on 468  degrees of freedom
## AIC: 76.585
##
## Number of Fisher Scoring iterations: 8
```

```
pred.full = predict(glm.full, type = "response")
y.pred.full = ifelse(pred.full < 0.5, 0, 1)
tab.full = table(y.pred.full, y.tr$type)
tab.full
```

```
##
## y.pred.full benign malignant
##           0    302         4
##           1      4       168
```

```
(tab.full[1,2] + tab.full[2,1]) / sum(tab.full)
```

```
## [1] 0.0167364
```

The missclassification rate is only 1.7%. Model looks very good in training data. But how is it possible that the individual predictors were not associated with the outcome with very low P-values in the logistic regression summary? This is because when the variables are highly correlated, their effects are difficult to separate from each other statistically, and hence the uncertainty of the individual model coefficients remains large when all of the variables are included in the same model. This leads to high P-values for each variable. However, when the model is applied to the data, it can still make very good predictions even though the values of individual coefficients remain uncertain. This gives an important lesson that, when there are correlated variables in the model, high P-values of individual variables do not necessarily mean that the variables are not predictive of the outcome value.

For comparison, let's also make a simpler model by including only two variables `thick` and `chrom`.

```
glm.2 = glm(type ~ thick + chrom, data = y.tr, family = "binomial")
summary(glm.2)
```

```
##
## Call:
## glm(formula = type ~ thick + chrom, family = "binomial", data = y.tr)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)  -9.7240     0.9989  -9.735  < 2e-16 ***
## thick         0.9530     0.1277   7.462 8.55e-14 ***
## chrom         1.2495     0.1627   7.680 1.60e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 624.58  on 477  degrees of freedom
## Residual deviance: 135.94  on 475  degrees of freedom
## AIC: 141.94
##
## Number of Fisher Scoring iterations: 7
```

```
pred.2 = predict(glm.2, type = "response")
y.pred.2 = ifelse(pred.2 < 0.5, 0, 1)
tab.2 = table(y.pred.2, y.tr$type)
tab.2
```

```
##
## y.pred.2 benign malignant
##        0    299        13
##        1      7       159
```

```
(tab.2[1,2] + tab.2[2,1])/sum(tab.2)
```

```
## [1] 0.041841
```

This model has a bit worse missclassification rate but much lower P-values for the individual predictors.

Let's then see how the two models work in test data.

```
pred.te.full = predict(glm.full, newdata = y.te, type = "response")
y.pred.te.full = ifelse(pred.te.full < 0.5, 0, 1)
tab.full = table(y.pred.te.full, y.te$type)
tab.full
```

```
##
## y.pred.te.full benign malignant
##              0    133         5
##              1      5        62
```

```
(tab.full[1,2] + tab.full[2,1]) / sum(tab.full)
```

```
## [1] 0.04878049
```

```
pred.te.2 = predict(glm.2, newdata = y.te, type = "response")
y.pred.te.2 = ifelse(pred.te.2 < 0.5, 0, 1)
tab.2 = table(y.pred.te.2, y.te$type)
tab.2
```

```
##
## y.pred.te.2 benign malignant
##           0    135        11
##           1      3        56
```

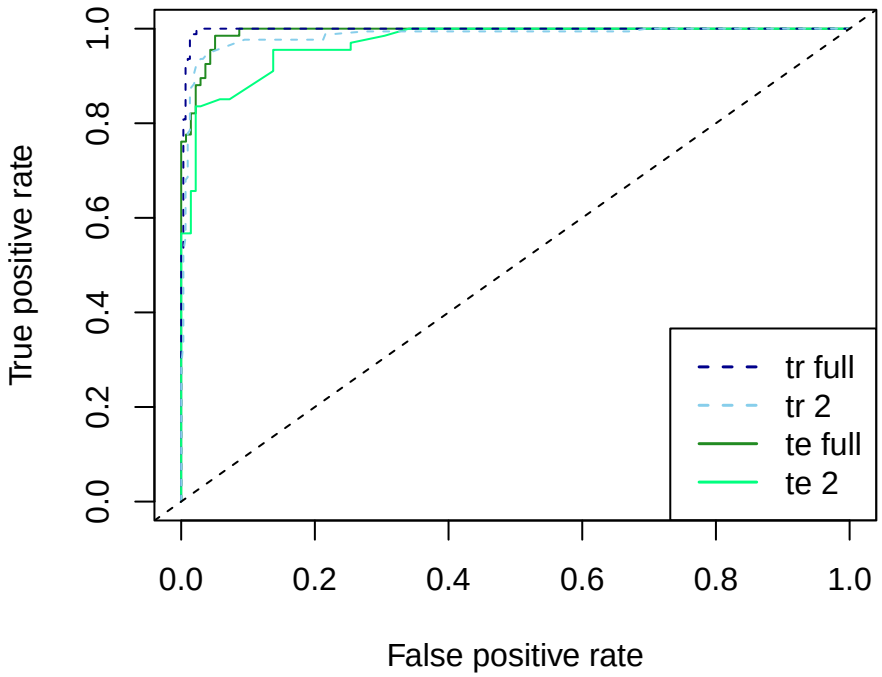
```
(tab.2[1,2] + tab.2[2,1]) / sum(tab.2)
```

```
## [1] 0.06829268
```

Error rates ares slightly larger in test data than in training data but the difference between the models stays similar: the full model has a smaller missclassification error than the simpler model.

Let's look at the ROCs.

```
#Test data ROCs:
# Full model
rocr.pred.te.full = prediction(pred.te.full, labels = y.te$type)
roc.perf.te.full = performance(rocr.pred.te.full, measure = "tpr", x.measure = "fpr")
plot(roc.perf.te.full, col = "forestgreen")

# 2 predictor model
rocr.pred.te.2 = prediction(pred.te.2, labels = y.te$type)
roc.perf.te.2 = performance(rocr.pred.te.2, measure = "tpr", x.measure = "fpr")
plot(roc.perf.te.2, col = "springgreen", add = T)

#Training data ROCs:
# Full model
rocr.pred.tr.full = prediction(pred.full, labels = y.tr$type)
roc.perf.tr.full = performance(rocr.pred.tr.full, measure = "tpr", x.measure = "fpr")
plot(roc.perf.tr.full, col="darkblue", add = T, lty = 2)

# 2 predictor model
rocr.pred.tr.2 = prediction(pred.2, labels = y.tr$type)
roc.perf.tr.2 = performance(rocr.pred.tr.2, measure = "tpr", x.measure = "fpr")
plot(roc.perf.tr.2, col="skyblue", add = T, lty = 2)

abline(a = 0, b = 1, lty = 2) #diagonal for random assignment
legend("bottomright", legend = c("tr full","tr 2","te full","te 2"),
       col = c("darkblue","skyblue","forestgreen","springgreen"), lty = c(2,2,1,1), lwd = 1.5)
```



We see that the full model performs better also in terms of ROC curve, both in training and test data. In general, the full model seems a very good classifier in this data set.

Extra: Understanding overfitting and difference between training and testing error

In our examples, there was no dramatic difference between missclassification errors in training vs. test sets. This is because our logistic regression model was a very simple model with only a few predictors compared to the number of available observations. To understand better why it is so important to consider a separate test data set, you can watch another video from StatsQuest: https://www.youtube.com/watch?v=EuBBz3bI-aA