Statistical Learning, Homework #1

Elisa Paolazzi [MAT. 224424]

01/04/2022

R Setup

```
library(tidyverse)
library(tidymodels) # for models fitting
library(class) # for knn
library(caret) # for createDataPartition
library(psych) # for dummy.code
library(e1071) # for Naive Bayes
library(pROC)  # for ROC curve
library(yardstick) # for precision
library(dplyr) # for data manipulation
library(ggplot2) # for plots
library(ggpubr) # for plots' grid
```

1. Explore the data: what is the distribution of the response variable (breast)? Are there potential issues with any of the predictors? Do you need pre-processing or can you proceed with the data as is?

The first step of the analysis is to load the data (`breastfeed.Rdata`) and explore it:

```
# load the dataset
load("breastfeed.Rdata")

# copy the breastfeed dataframe to a new variable
breastfeed_df = breastfeed

# check dimension (number of columns and rows) of dataset
dim(breastfeed_df)
```

```
## [1] 139  10
```

```
# see column names
names(breastfeed_df)
```

```
##  [1] "breast"    "pregnancy" "howfed"    "howfedfr"  "partner"   "smokenow"
##  [7] "smokebf"   "age"       "educat"    "ethnic"
```

```
# see an extract of the dataset
as_tibble(breastfeed_df)
```

```
## # A tibble: 139 x 10
##    breast pregnancy howfed howfedfr partner smokenow smokebf   age educat ethnic
##    <fct>  <fct>     <fct>  <fct>    <fct>   <fct>    <fct>   <dbl>  <dbl> <fct>
##  1 Breast Beginning Breast Breast   Partner No       No         24     19 Non-w~
##  2 Breast Beginning Bottle Breast   Partner No       No         27     18 White
```

```
##  3 Bottle Beginning Breast Breast   Partner No       No         39     16 White
##  4 Bottle Beginning Breast Breast   Partner Yes      Yes        29     16 White
##  5 Breast Beginning Breast Breast   Partner No       No         21     21 White
##  6 Bottle Beginning Breast Bottle   Partner No       No         NA     28 White
##  7 Breast Beginning Breast Breast   Partner No       No         27     19 Non-w~
##  8 Breast Beginning Breast Breast   Partner No       No         27     22 Non-w~
##  9 Breast Beginning Breast Bottle   Partner Yes      Yes        20     19 Non-w~
## 10 Breast Beginning Breast Breast   Partner No       Yes        31     17 White
## # ... with 129 more rows
```

As we can see we are dealing with a dataset of 139 observations. For each observation we have 10 columns: as specified in the homework assignment the `breast` column contains a categorical (binary) response variable and the other 9 includes different types of predictors. We can also print a summary in order to better understand the content of the columns, and thus the nature of our predictors.

```
# see the summary
summary(breastfeed_df)
```

```
##     breast        pregnancy     howfed     howfedfr     partner    smokenow
##  Bottle: 39   End      :84   Bottle:59   Bottle:54   Single : 21   No :107
##  Breast:100   Beginning:55   Breast:80   Breast:85   Partner:118   Yes: 32
##
##
##
##
##
##  smokebf       age            educat            ethnic
##  No :88   Min.   :17.00   Min.   :14.00   White    :80
##  Yes:51   1st Qu.:25.00   1st Qu.:16.00   Non-white:59
##           Median :28.00   Median :17.00
##           Mean   :28.26   Mean   :18.15
##           3rd Qu.:32.00   3rd Qu.:19.00
##           Max.   :40.00   Max.   :38.00
##           NA's   :2       NA's   :2
```

We can notice that the categorical/dummy (`breast`, `pregnancy`, `howfed`, `howfedfr`, `partner`, `ethnic`, `smokenow`, `smokebf`) variables of the dataset are already correctly coded as factors and the levels correspond to those explained in the assignment description. Moreover the previous summary show us the presence of couple of NAs values in the numerical columns `age` and `educat`:

```
# check number of NAs
sum(is.na(breastfeed_df))
```

```
## [1] 4
```

```
# observations containing NA value for age:
breastfeed_df[is.na(breastfeed_df$age),]
```

```
##     breast pregnancy howfed howfedfr partner smokenow smokebf age educat ethnic
## 6   Bottle Beginning Breast   Bottle Partner       No      No  NA     28  White
## 125 Breast       End Breast   Bottle Partner       No      No  NA     16  White
```

```
# observations containing NA value for educat:
breastfeed_df[is.na(breastfeed_df$educat),]
```

```
##    breast pregnancy howfed howfedfr partner smokenow smokebf age educat
## 22 Bottle       End Breast   Bottle Partner       No      No  31     NA
## 46 Bottle       End Bottle   Bottle Partner       No      No  38     NA
##       ethnic
## 22 Non-white
## 46     White
```

In order to avoid affecting the fitting of future models, we remove observations containing missing values:

```
# remove rows containing NAs using dplyr
breastfeed_df = breastfeed_df %>%
  na.omit()
```

The decision to remove these observations also stems from the fact that they are fortunately a small group of observations (0.03% on the whole dataset). If it had been a larger number of NAs it would have been appropriate to fill in these missing values in some way (e.g, fill NAs with with the average of the corresponding class if the variable is numeric). Anyway, many models already exclude missing values from fit and prediction by default.

We can now look at the response variable which is the factor column `breast`.

```
# look again to the summary of the response column
summary(breastfeed_df$breast)
```

```
## Bottle Breast
##     36     99
```

```
# class percentage (empirical frequencies)
prop.table(table(breastfeed_df$breast))
```

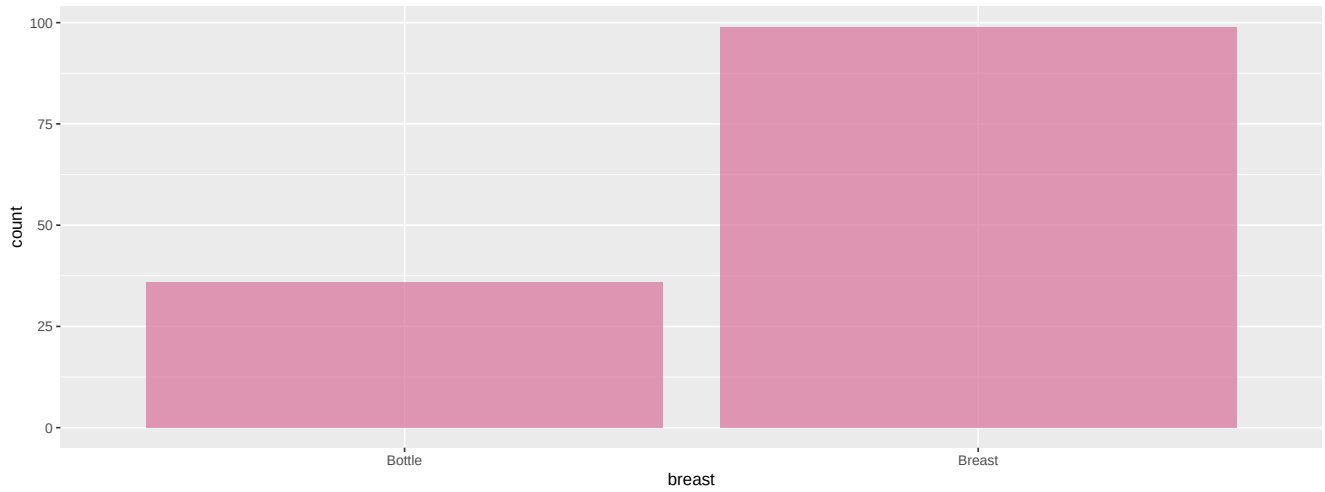
```
##
##    Bottle    Breast
## 0.2666667 0.7333333
```

The two class of the response variable are *Bottle* and *Breast*, which again correspond to the levels explained in the assignment:

- *Breast* class includes the cases "breastfeeding", "try to breastfeed" and "mixed breast- and bottle-feeding";
- *Bottle* class corresponds to "exclusive bottle-feeding".

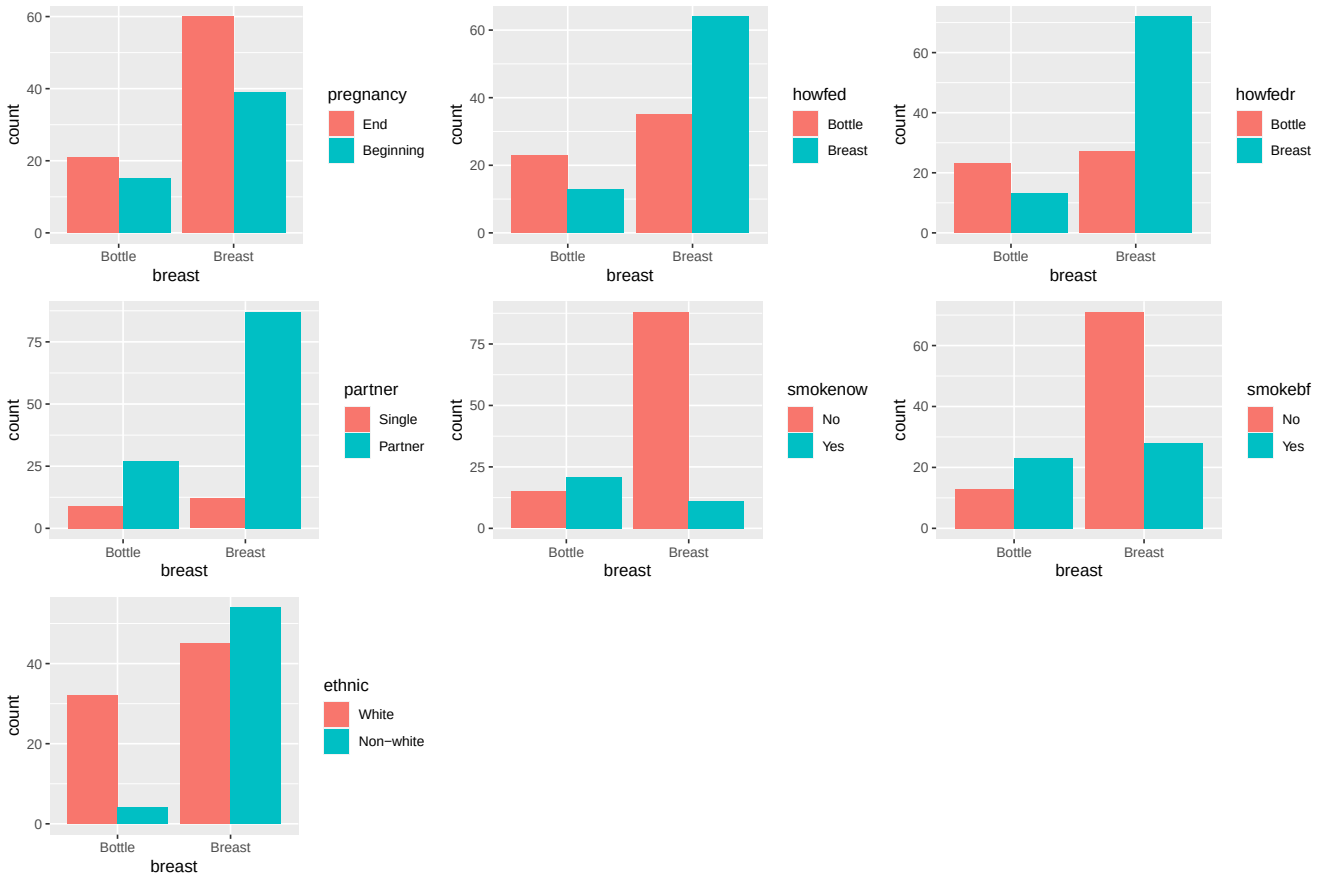
The categories seems to be quite unbalanced since 73.33% of observations belong to the *Breast* class and the the remaining 26.66% to the *Bottle* class. We can also show the unbalance of the dataset by means of a barplot:

```
# response variable barplot
ggplot(breastfeed_df, aes(x=breast)) +
  geom_bar(fill="#D77099", alpha=0.7)
```



To better investigate the categorical predictors with respect to the response variable (`breast`) we can create a grouped barchart for each of them:

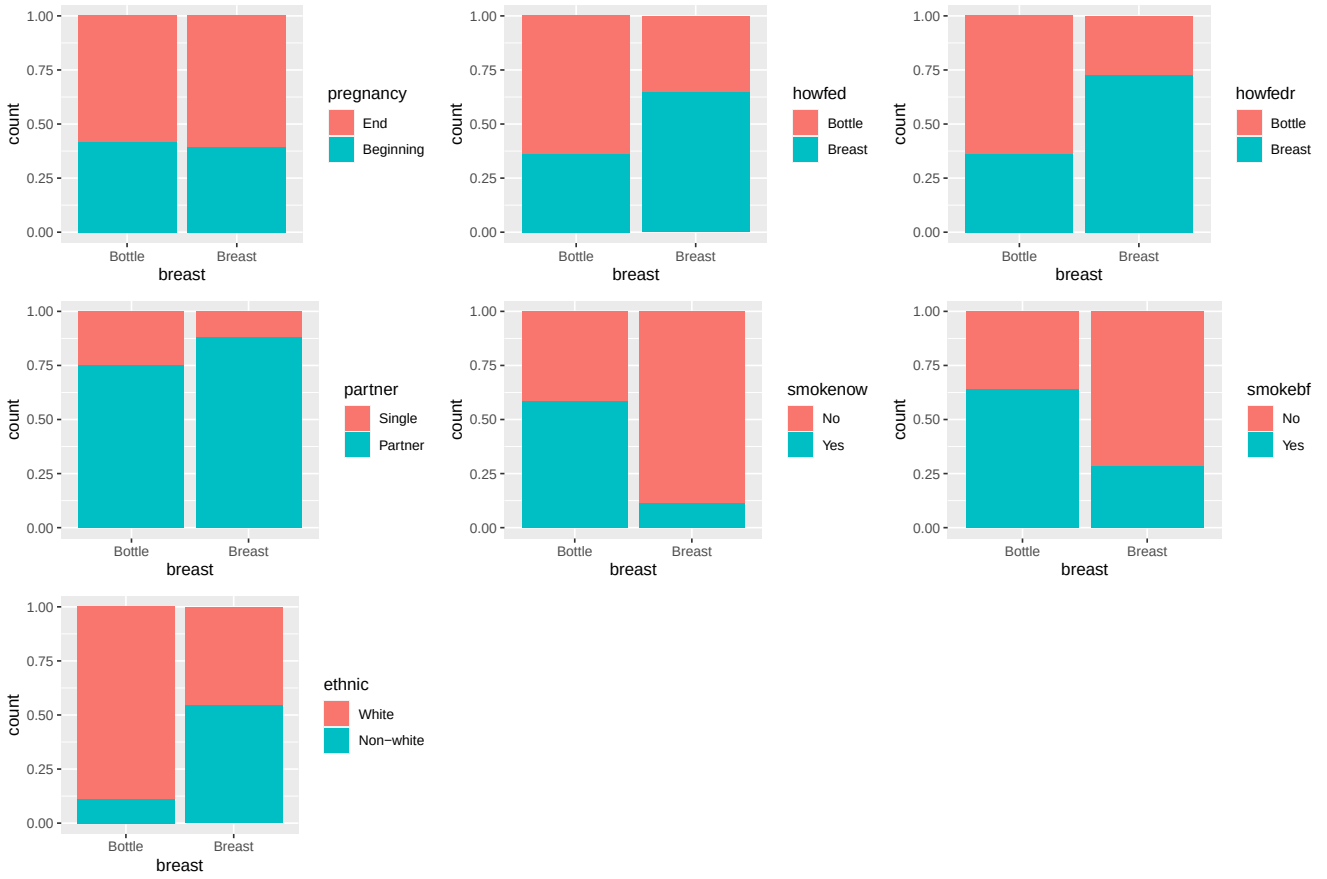
```r
# create a function to build each categorical predictor plot (grouped barchart)
cat_plots = function(column, leg_title) {
  plot = ggplot(breastfeed_df, aes(fill=column, x=breast)) +
  geom_bar(position="dodge") +
  guides(fill=guide_legend(title=as.character(leg_title)))
}
# build plots using the previous defined function
p1 = cat_plots(breastfeed_df$pregnancy, "pregnancy")
p2 = cat_plots(breastfeed_df$howfed, "howfed")
p3 = cat_plots(breastfeed_df$howfedfr, "howfedr")
p4 = cat_plots(breastfeed_df$partner, "partner")
p5 = cat_plots(breastfeed_df$smokenow, "smokenow")
p6 = cat_plots(breastfeed_df$smokebf, "smokebf")
p7 = cat_plots(breastfeed_df$ethnic, "ethnic")
# show plots in a grid
ggarrange(p1, p2, p3, p4, p5, p6, p7,
          ncol = 3, nrow = 3)
```



However, as the response variable categories are unbalanced, the previous plots are not very informative. Percent stacked barchart are a better alternative to show the distribution of the predictor variables in the two response categories:

```r
# create a function to build each categorical predictor plot (percent stacked barchart)
cat_plots_perc = function(column, leg_title) {
  plot = ggplot(breastfeed_df, aes(fill=column, x=breast)) +
  geom_bar(position="fill") +
  guides(fill=guide_legend(title=as.character(leg_title)))
}
# build plots using the previous defined function
p1 = cat_plots_perc(breastfeed_df$pregnancy, "pregnancy")
p2 = cat_plots_perc(breastfeed_df$howfed, "howfed")
p3 = cat_plots_perc(breastfeed_df$howfedfr, "howfedr")
```

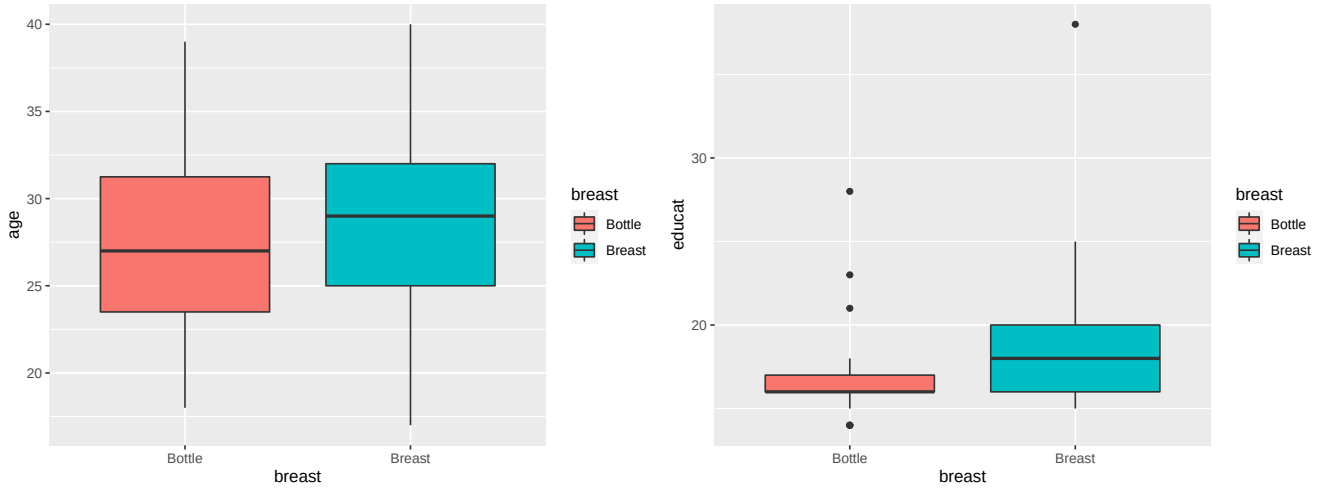
```
p4 = cat_plots_perc(breastfeed_df$partner, "partner")
p5 = cat_plots_perc(breastfeed_df$smokenow, "smokenow")
p6 = cat_plots_perc(breastfeed_df$smokebf, "smokebf")
p7 = cat_plots_perc(breastfeed_df$ethnic, "ethnic")
# show plots in a grid
ggarrange(p1, p2, p3, p4, p5, p6, p7,
          ncol = 3, nrow = 3)
```



As we can notice in the plot grid some predictors seem to have meaningful differences in their distribution within the response variable categories (e.g, `smokenow`, `ethnic`, `howfedr`, `howfed`, `smokebf`). This may already reveal something about the potential significance of the predictors in the upcoming models.

Regarding the numerical predictors, we can show their distribution within the categories of the response variable by means of a boxplot:

```
# create a function to build each numerical predictor boxplot
num_plots = function(column, lab) {
  ggplot(breastfeed_df, aes(x=breast, y=column, fill=breast)) +
  geom_boxplot() +
  ylab(lab)
}
# build plots using the previous defined function
pn1 = num_plots(breastfeed_df$age, "age")
pn2 = num_plots(breastfeed_df$educat, "educat")
# show plots in a grid
ggarrange(pn1, pn2,
          ncol = 2, nrow = 1)
```



Looking at the numerical predictors is now useful to explore the relationships between them and see whether they are correlated.

```
# plot age vs educat
ggplot(breastfeed_df, aes(x=age, y=educat)) +
  geom_point(colour="#D77099", alpha=0.7, size=3) +
  geom_smooth(method='lm',formula=y~x, colour="#858585")
```


```
# correlation between age and educat
cor(dplyr::select(breastfeed_df, age, educat))
```

```
##               age    educat
## age     1.0000000 0.2001478
## educat 0.2001478 1.0000000
```

As we can see also from the scatterplot, there is a slight positive correlation (0.2) between `age` and `educat`. As this is a reasonable and rather slight correlation, it should not compromise the fitting of the models and the significance of the predictors themselves.

2. Split the data into (reproducible) training and test sets. Given the class imbalance, you could aim for sets that have the same imbalance with respect to the outcome variable. In order to do this, you could either perform the splitting manually on each class, or use dedicated functions (for example, `caret::createDataPartition(labels, p=train_size)`, with `train_size` a number between 0 and 1 representing the percentage of data you would like to use for training.

Dividing the dataset in two sets is essential: the models will be fitted on the training set data and these fitted models will be used to make prediction on the test data, also comparing the results with the true responses. In our

case, where the response classes are unbalanced, it is important to fit and test our models on subsets that reproduce this imbalance situation.

```
# set seed for reproducible results
set.seed(42)

# create partitions that have the same imbalance with respect to the outcome variable
# (setting the train size to 70%)
train = caret::createDataPartition(breastfeed_df$breast, p=0.7, list = F,)

# split train and test sets based on the previous defined partition
breastfeed_train = breastfeed_df[train,]
breastfeed_test = breastfeed_df[-train,]

# verify that the classes' imbalance is maintained
summary(breastfeed_train$breast)
```

```
## Bottle Breast
##     26     70
```

```
# class percentage (empirical frequencies)
prop.table(table(breastfeed_train$breast))
```

```
##
##    Bottle    Breast
## 0.2708333 0.7291667
```

After these manual verification, we have shown that we have successfully maintained the unbalance of the classes in the partitions.

3. Fit the following GLM model:

$$\begin{aligned}\text{logit}(\text{E}(\text{breast})) = \beta_0 &+ \beta_1 \text{pregnancy} + \beta_2 \text{howfed} + \beta_3 \text{howfedfr} \\ &+ \beta_4 \text{partner} + \beta_5 \text{age} + \beta_6 \text{educat} + \beta_7 \text{ethnic} + \beta_8 \text{smokenow} + \beta_9 \text{smokebf}\end{aligned}$$

Discuss the `summary` and the interpretation of the model in the context of the study.

Logistic regression is a statistical model that uses Logistic function to model the conditional probability, for binary cases, we calculate the conditional probability of the dependent variable Y (`breast`), given independent variable X (the vector of predictors). It uses logit function, also referred to as log-odds, which is the logarithm of odds. Hence, the logit function estimates probabilities between 0 and 1, and its parameter β can be estimated using the maximum likelihood estimation(MLE), which searches for the parameters that best fit the joint probability of the predictors.

As first model we fit a logistic regression including all predictors on the train set. We will use `tidymodels` for this purpose:

```
# build the model specification
lr_spec = logistic_reg() %>%
    set_engine("glm") %>%
    set_mode("classification")

# fit the model on all predictors (using "~ ." to fit on all of them)
lr_fit = lr_spec %>%
    fit(breast ~ .,
        data = breastfeed_train)

# show fit summary
```

```
lr_fit %>%
    pluck("fit") %>%
    summary()
```

```
##
## Call:
## stats::glm(formula = breast ~ ., family = stats::binomial, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7056  -0.5575   0.2442   0.5718   2.5198
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -4.57455    2.45420  -1.864   0.0623 .
## pregnancyBeginning -0.52753    0.63473  -0.831   0.4059
## howfedBreast        0.15033    0.70098   0.214   0.8302
## howfedfrBreast      1.22698    0.65077   1.885   0.0594 .
## partnerPartner      0.61210    0.82141   0.745   0.4562
## smokenowYes        -2.47934    1.07925  -2.297   0.0216 *
## smokebfYes          1.03285    1.02668   1.006   0.3144
## age                 0.05477    0.05923   0.925   0.3552
## educat              0.14850    0.13077   1.136   0.2561
## ethnicNon-white     2.28216    0.88694   2.573   0.0101 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 112.144  on 95  degrees of freedom
## Residual deviance:  73.031  on 86  degrees of freedom
## AIC: 93.031
##
## Number of Fisher Scoring iterations: 6
```

The first section of the output show us the original call to the `glm()` function (the formula that we write for our model). In the Deviance Residuals section it give us a summary of the deviance residuals: their values are satisfactory since they are close to be centered on 0 and are roughly symmetrical.

The Coefficients section contains all the coefficient value from β_0 (intercept) to β_9 and the related standard error, z value (computed trough the Wald's test: $\frac{coefficientestimate}{standarderror}$) and p-value, which test coefficients not to be statistically different from zero (how significant they are).

In this case the predictors `howfedfr`, `smokenow` and `ethnic` seems to be significant, since they have a quite low p-value (even if the p-value of `howfedfr` is just above the 0.05 threshold, it is still included in the most significant items for this model). In particular the logit coefficients represent the change in the log odds of the outcome for a one unit increase in the predictor variable:

- if friends' mothers breastfed their babies, versus they fed their baby with bottle (baseline) changes the log odds of breast by 1.23;
- being a smoker now, versus being a non-smoker changes the log odds by -2.48;
- belonging to a non-white ethnicity, versus belonging to a white ethnicity changes the log odds by 2.28.

Next to the coefficients estimates we see the default dispersion parameter used for this logistic regression, which represents the fixed value ϕ for binomial density function in the EDF's rewriting settings.

Below the Coefficients section the summary includes indices of fit: the null and deviance residuals and the AIC. These measures of model fit represent the significance of the overall model, testing whether the model with predictors fits significantly better than a model with just an intercept (i.e., a null model). Quite high null and residual deviance means that the null model does not explain the data pretty well, but in general these values are useful in model comparison. Finally we have AIC value which, in this context, is just the residual deviance adjusted for the number

of parameters in the model. Of course also AIC represents an important criterion to compare this model with future fittings: a lower AIC represents a better fit in terms of amount of variation explained and number of parameters used (complexity).

Lastly the number of Fisher Scoring iterations tells us how quickly the `glm()` function converged on the maximum likelihood estimates for the coefficients.

Once the model has been fitted, we can finally test it on the test set:

```
# make prediction on test set data
predict(lr_fit, new_data=breastfeed_test, type="prob")
```

```
## # A tibble: 39 x 2
##    .pred_Bottle .pred_Breast
##           <dbl>        <dbl>
##  1      0.0354         0.965
##  2      0.105          0.895
##  3      0.818          0.182
##  4      0.0770         0.923
##  5      0.00345        0.997
##  6      0.298          0.702
##  7      0.144          0.856
##  8      0.0407         0.959
##  9      0.770          0.230
## 10      0.783          0.217
## # ... with 29 more rows
```

```
# make prediction using test set data and save the predicted class in a new variable
glm_pred = predict(lr_fit, new_data=breastfeed_test, type="class")

# display the ground truth besides the model predictions
augment(lr_fit, new_data=breastfeed_test)[c(1,11)]
```

```
## # A tibble: 39 x 2
##    breast .pred_class
##    <fct>  <fct>
##  1 Breast Breast
##  2 Breast Breast
##  3 Bottle Bottle
##  4 Breast Breast
##  5 Breast Breast
##  6 Breast Breast
##  7 Breast Breast
##  8 Breast Breast
##  9 Bottle Bottle
## 10 Bottle Bottle
## # ... with 29 more rows
```

```
# display confusion matrix
augment(lr_fit, new_data=breastfeed_test) %>%
    conf_mat(truth=breast, estimate=.pred_class)
```

```
##           Truth
## Prediction Bottle Breast
##     Bottle      8      4
##     Breast      2     25
```

```
# compute accuracy:
augment(lr_fit, new_data=breastfeed_test) %>%
    accuracy(truth=breast, estimate=.pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.846
```

```
# save accuracy for further comparison
lr_fit_accuracy = augment(lr_fit, new_data=breastfeed_test) %>%
    accuracy(truth=breast, estimate=.pred_class)
```

Given the 84.6% accuracy, the model seems to perform quite well on the test set. As we can see in the confusion matrix the model fails to classify 4 observations in the *Breast* category and 2 observations in the *Bottle* category. However, we must remember that we are dealing with imbalanced classes; it is therefore useful to also investigate the accuracy of the individual classes (precision) in order to understand whether the model tends to misclassify one class more than the other:

```
# "Breast" class precision
lr_fit_breast_precision = augment(lr_fit, new_data=breastfeed_test) %>%
    yardstick::precision(truth=breast, estimate=.pred_class, event_level="second")

# "Bottle" class precision
lr_fit_bottle_precision = augment(lr_fit, new_data=breastfeed_test) %>%
    yardstick::precision(truth=breast, estimate=.pred_class, event_level="first")
message("Breast class precision: ", lr_fit_breast_precision[3])
```

```
## Breast class precision: 0.925925925925926
```

```
message("Bottle class precision: ", lr_fit_bottle_precision[3])
```

```
## Bottle class precision: 0.666666666666667
```

As we can see, the precision is much lower for the smaller class (*Bottle*), which means that the model is more prone to misclassify that class.

Still, we can now refit the model using only the significant predictors to see whether accuracy and classes' precision will increase:

```
# fit the model on the significant predictors
lr_fit2 = lr_spec %>%
    fit(breast ~ howfedfr + smokenow + ethnic,
        data = breastfeed_train)

# show fit summary
lr_fit2 %>%
    pluck("fit") %>%
    summary()
```

```
##
## Call:
## stats::glm(formula = breast ~ howfedfr + smokenow + ethnic, family = stats::binomial,
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7249  -0.5892   0.2223   0.6549   1.9166
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      0.07171    0.46327   0.155  0.87699
## howfedfrBreast   1.35874    0.56489   2.405  0.01616 *
```

```
## smokenowYes     -1.73475    0.60488  -2.868  0.00413 **
## ethnicNon-white  2.25748    0.73169   3.085  0.00203 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 112.144  on 95  degrees of freedom
## Residual deviance:  78.824  on 92  degrees of freedom
## AIC: 86.824
##
## Number of Fisher Scoring iterations: 5
```

The coefficient values have been adjusted and the p-values are now even lower. In addition, the latter model has a lower AIC value than the previous one, showing a better fitting. Finally, this model converges slightly faster on the maximum likelihood estimates for the coefficients (lower number of Fisher Scoring iterations).

Let us check whether these insights actually lead to an increase in accuracy and classes' precision on the test set:

```
# make prediction using test set data and save the predicted class in a new variable
glm_pred = predict(lr_fit2, new_data=breastfeed_test, type="class")

# display the ground truth besides the model predictions
augment(lr_fit2, new_data=breastfeed_test)[c(1,11)]
```

```
## # A tibble: 39 x 2
##    breast .pred_class
##    <fct>  <fct>
##  1 Breast Breast
##  2 Breast Breast
##  3 Bottle Bottle
##  4 Breast Breast
##  5 Breast Breast
##  6 Breast Breast
##  7 Breast Breast
##  8 Breast Breast
##  9 Bottle Bottle
## 10 Bottle Bottle
## # ... with 29 more rows
```

```
# confusion matrix
augment(lr_fit2, new_data=breastfeed_test) %>%
    conf_mat(truth=breast, estimate=.pred_class)
```

```
##           Truth
## Prediction Bottle Breast
##     Bottle      8      2
##     Breast      2     27
```

```
# accuracy:
augment(lr_fit2, new_data=breastfeed_test) %>%
    accuracy(truth=breast, estimate=.pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.897
```

```
# save accuracy for further comparison
lr_fit_accuracy_2 = augment(lr_fit2, new_data=breastfeed_test) %>%
    accuracy(truth=breast, estimate=.pred_class)
```

As expected, accuracy increased to 89.7%. Let's check also precision:

```
# "Breast" class precision
lr_fit2_breast_precision = augment(lr_fit2, new_data=breastfeed_test) %>%
    yardstick::precision(truth=breast, estimate=.pred_class, event_level="second")

# "Bottle" class precision
lr_fit2_bottle_precision = augment(lr_fit2, new_data=breastfeed_test) %>%
    yardstick::precision(truth=breast, estimate=.pred_class, event_level="first")
message("Breast class precision: ", lr_fit2_breast_precision[3])
```

```
## Breast class precision: 0.931034482758621
```

```
message("Bottle class precision: ", lr_fit2_bottle_precision[3])
```

```
## Bottle class precision: 0.8
```

Precision increased as well in both classes, particularly in the *Bottle* class which previously scored only 0.67.

4. Fit a k-nn classifier, by performing a careful selection of the tuning parameter k.

Once the results of the logistic regression have been analysed, we can now fit a k-nn classifier. K-nn is a nonparametric classifiers used in pattern recognition for classifying objects based on the characteristics of objects close to the one being considered. By means of such classifier an observation is assigned to the most common class among its k nearest neighbours: therefore it is important to find the appropriate number of k (neighbours to be considered) that allows to correctly classify as many observations as possible.

For the sake of completeness, we will fit and test both the full model (using all predictors) and the model that includes only the most significant predictors from the logistic regression (`howfedfr`, `smokenow`, `ethnic`).

So let's start with the complete one. Since k-nn is based on distances to identify observations near to each other, in order to prevent the numerical column values from affecting the distance calculation, we have to standardise these predictors. We standardise training and test sets separately so as to avoid using information from the different sets in the standardisation operation of each.

```
# standardize numerical columns separately for train and test set and store the results in
# new dataframes
breastfeed_knn_train = breastfeed_train %>%
  mutate_if(is.numeric, scale)

breastfeed_knn_test = breastfeed_test %>%
  mutate_if(is.numeric, scale)

# verify the standardization on some example values within the train and test set.
var(breastfeed_train$age) # previous variance
```

```
## [1] 29.75164
```

```
var(breastfeed_knn_train$age) # after standardisation variance
```

```
##      [,1]
## [1,]    1
```

```r
var(breastfeed_test$educat) # previous variance
```

```
## [1] 15.4143
```

```r
var(breastfeed_knn_test$educat) # after standardisation variance
```

```
##      [,1]
## [1,]    1
```

```r
mean(breastfeed_train$educat) # previous mean
```

```
## [1] 18.19792
```

```r
mean(breastfeed_knn_train$educat) # after standardisation mean
```

```
## [1] -4.335273e-16
```

```r
mean(breastfeed_test$age) # previous mean
```

```
## [1] 28.05128
```

```r
mean(breastfeed_knn_test$age) # after standardisation mean
```

```
## [1] 3.560631e-17
```

We will use a for loop on a k values vector to store the results and thus find the best k number. In order to compare the optimal k number's models we also test the model on the training set itself.

```r
# set seed for reproducible results
set.seed(42)

# number of ks
kvec = c(seq(1:30))

# initialize dataframes for storing train and test accuracy
results_df_train = data.frame(k_number = kvec, type="train", error = 0, accuracy=0, precision_breast=0, pr
results_df_test = data.frame(k_number = kvec, type="test", error = 0, accuracy=0, precision_breast=0, prec

# iterate over the k number vector
for (k in kvec) {

  # compose the model specification
  knn_spec = nearest_neighbor(neighbors=k) %>%
    set_mode("classification") %>%
    set_engine("kknn")

  # fit the knn classifier
  knn_fit = knn_spec %>%
    fit(breast ~., data=breastfeed_knn_train)

  # train accuracy
  res = augment(knn_fit, new_data=breastfeed_knn_train) %>%
    accuracy(truth=breast, estimate=.pred_class) # accuracy

  # store results
  results_df_train$error[k] = 1 - res$.estimate
```

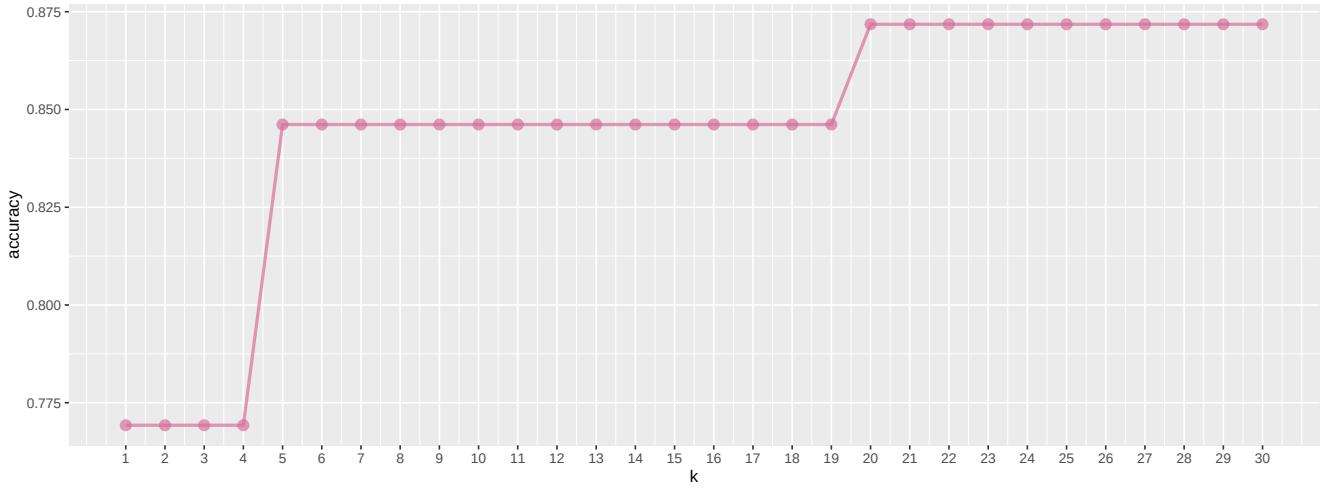
```
  results_df_train$accuracy[k] = res$.estimate

  # train precision
  prec_br = augment(knn_fit, new_data=breastfeed_knn_train) %>%
     yardstick::precision(truth=breast, estimate=.pred_class, event_level="second")
  prec_bo = augment(knn_fit, new_data=breastfeed_knn_train) %>%
     yardstick::precision(truth=breast, estimate=.pred_class, event_level="first")

  # store results
  results_df_train$precision_breast[k] = prec_br$.estimate
  results_df_train$precision_bottle[k] = prec_bo$.estimate

  # test accuracy
  res = augment(knn_fit, new_data=breastfeed_knn_test) %>%
    accuracy(truth=breast, estimate=.pred_class) # accuracy

  # store results
  results_df_test$error[k] = 1 - res$.estimate
  results_df_test$accuracy[k] = res$.estimate

  # test precision
  prec_br = augment(knn_fit, new_data=breastfeed_knn_test) %>%
     yardstick::precision(truth=breast, estimate=.pred_class, event_level="second")
  prec_bo = augment(knn_fit, new_data=breastfeed_knn_test) %>%
     yardstick::precision(truth=breast, estimate=.pred_class, event_level="first")

  # store results
  results_df_test$precision_breast[k] = prec_br$.estimate
  results_df_test$precision_bottle[k] = prec_bo$.estimate

}

# visualize test's results ordered by error
head(results_df_test[order(results_df_test$error),])
```

```
##    k_number type     error  accuracy precision_breast precision_bottle
## 20       20 test 0.1282051 0.8717949        0.9285714        0.7272727
## 21       21 test 0.1282051 0.8717949        0.9285714        0.7272727
## 22       22 test 0.1282051 0.8717949        0.9285714        0.7272727
## 23       23 test 0.1282051 0.8717949        0.9285714        0.7272727
## 24       24 test 0.1282051 0.8717949        0.9285714        0.7272727
## 25       25 test 0.1282051 0.8717949        0.9285714        0.7272727
```

The best k-value seems to be 20 and above, reaching an accuracy of 87.2%. On the other hand, the lowest accuracy (76.9%) is achieved by the smallest values (which represent the most complex models), while intermediate accuracy (84.6%) is achieved by values between 5 and 19. In terms of the precision of the two classes, it improves with the best k numbers, especially for the minor class (*Bottle*), which increases from 0.53 to 0.73.

We can visualize the test set results in terms of accuracy and number of k:

```
# plot test set results (accuracy vs number of k)
ggplot(results_df_test, aes(x=k_number, y=accuracy)) +
    geom_line(colour="#D77099", alpha=0.7, size=1) +
    geom_point(colour="#D77099", alpha=0.7, size=3) +
    scale_x_continuous(breaks = results_df_test$k_number) +
    xlab("k")
```



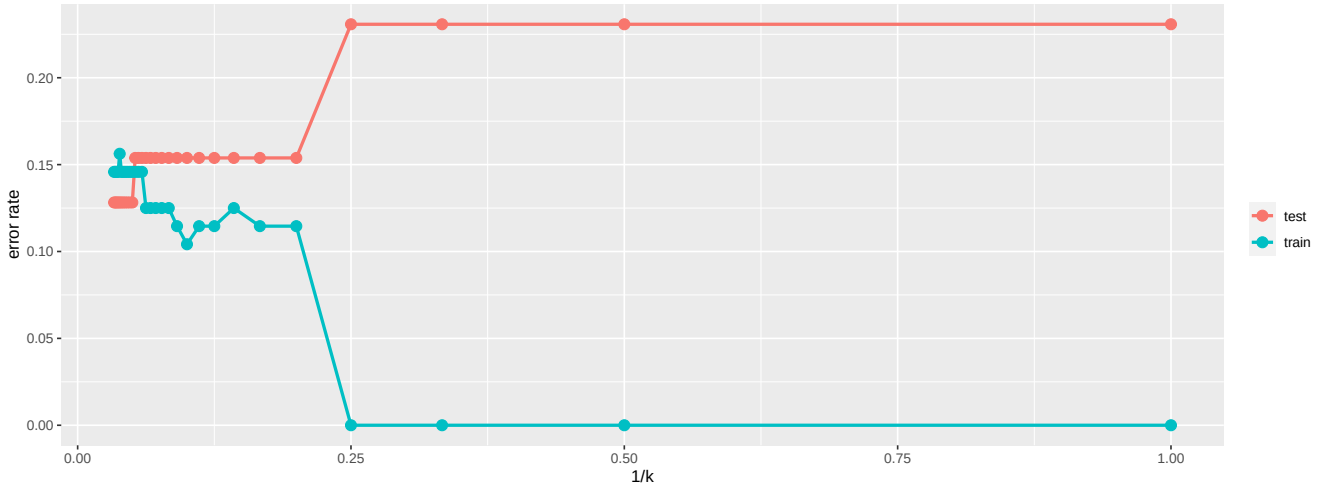
Obviously, the performances on the training set are less prone to errors; but unlike the test set, their accuracy increases with model's complexity (lower k values), reaching the 100%. In this case the decision surface adapts perfectly to the data and we would risk to overfit the data and fitting peculiarity of the data that are not of the true underlying mechanism, so the model will not perform well on new data (test set).

We can visualise and compare performances on the train and test set using *error rate vs 1/k* plot:

```
# put results of train and test sets together in a new dataframe
results_df = rbind(results_df_train, results_df_test)
# show the results ordered by the lowest error
head(results_df[order(results_df$error),], 20)
```

```
##    k_number  type     error  accuracy precision_breast precision_bottle
## 1         1 train 0.0000000 1.0000000        1.0000000        1.0000000
## 2         2 train 0.0000000 1.0000000        1.0000000        1.0000000
## 3         3 train 0.0000000 1.0000000        1.0000000        1.0000000
## 4         4 train 0.0000000 1.0000000        1.0000000        1.0000000
## 10       10 train 0.1041667 0.8958333        0.9054054        0.8636364
## 5         5 train 0.1145833 0.8854167        0.9041096        0.8260870
## 6         6 train 0.1145833 0.8854167        0.9041096        0.8260870
## 8         8 train 0.1145833 0.8854167        0.8933333        0.8571429
## 9         9 train 0.1145833 0.8854167        0.8933333        0.8571429
## 11       11 train 0.1145833 0.8854167        0.8933333        0.8571429
## 7         7 train 0.1250000 0.8750000        0.8918919        0.8181818
## 12       12 train 0.1250000 0.8750000        0.8815789        0.8500000
## 13       13 train 0.1250000 0.8750000        0.8815789        0.8500000
## 14       14 train 0.1250000 0.8750000        0.8815789        0.8500000
## 15       15 train 0.1250000 0.8750000        0.8815789        0.8500000
## 16       16 train 0.1250000 0.8750000        0.8815789        0.8500000
## 50       20  test 0.1282051 0.8717949        0.9285714        0.7272727
## 51       21  test 0.1282051 0.8717949        0.9285714        0.7272727
## 52       22  test 0.1282051 0.8717949        0.9285714        0.7272727
## 53       23  test 0.1282051 0.8717949        0.9285714        0.7272727
```

```
# plot error rates vs 1/k
ggplot(results_df, aes(x=1/k_number, y=error, color=as.factor(type), fill=as.factor(type), group=as.factor
  geom_line(size=1) +
  geom_point(shape=21, size=3) +
  xlab("1/k") +
  ylab("error rate") +
  theme(legend.title=element_blank())
```



Considering what we pointed out earlier, we can see that the error rate in the training set becomes smaller and smaller as the complexity of the model increases, reaching 0, while the error in the test set will tend to increase with complex models.

We can now fit the model using only the significant predictors previously identified in the fit of the logistic regression. We then test the new model on the test set to see whether it achieves a higher accuracy than the previous one.

```
# set seed for reproducible results
set.seed(42)

# number of ks
kvec = c(seq(1:30))

# initialize dataframes for storing train and test accuracy
results_df_test_2 = data.frame(k_number = kvec, type="test", error = 0, accuracy=0,
                               precision_breast=0, precision_bottle=0)

for (k in kvec) {

  # compose the model specification
  knn_spec = nearest_neighbor(neighbors=k) %>%
    set_mode("classification") %>%
    set_engine("kknn")

  # fit the knn classifier
  knn_fit = knn_spec %>%
    fit(breast ~ howfedfr + smokenow + ethnic, data=breastfeed_knn_train)

  # test accuracy
  res = augment(knn_fit, new_data=breastfeed_knn_test) %>%
    accuracy(truth=breast, estimate=.pred_class) # accuracy

  # store results
  results_df_test_2$error[k] = 1 - res$.estimate
  results_df_test_2$accuracy[k] = res$.estimate

  # test precision
  prec_br = augment(knn_fit, new_data=breastfeed_knn_test) %>%
     yardstick::precision(truth=breast, estimate=.pred_class, event_level="second")
  prec_bo = augment(knn_fit, new_data=breastfeed_knn_test) %>%
     yardstick::precision(truth=breast, estimate=.pred_class, event_level="first")

  # store results
  results_df_test_2$precision_breast[k] = prec_br$.estimate
  results_df_test_2$precision_bottle[k] = prec_bo$.estimate
```

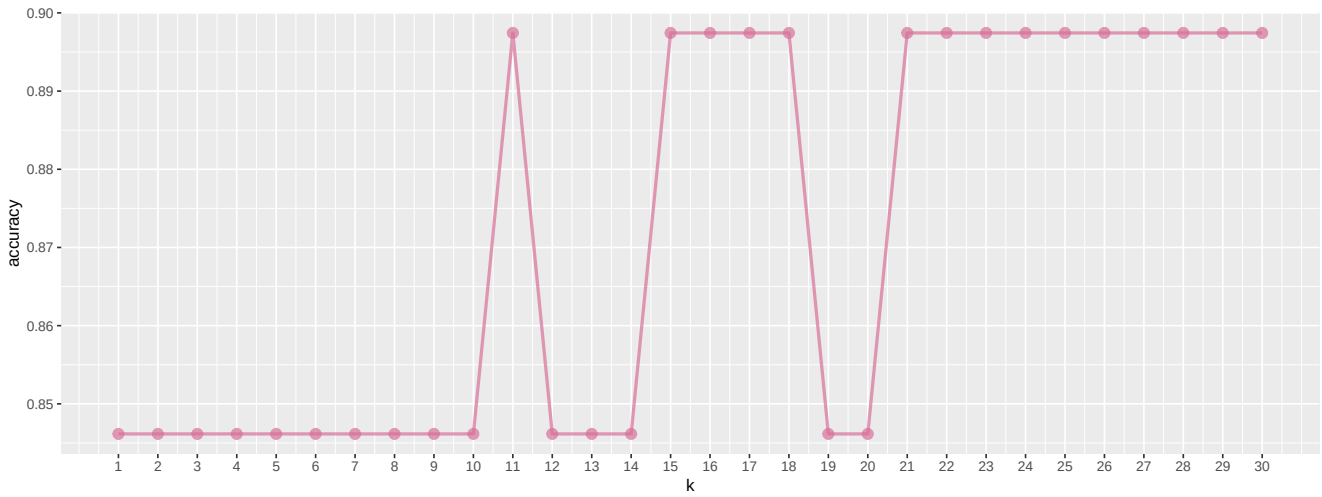
```
}

# visualize test's results ordered by error
head(results_df_test_2[order(results_df_test_2$error),])
```

```
##    k_number type     error  accuracy precision_breast precision_bottle
## 11       11 test 0.1025641 0.8974359        0.9310345              0.8
## 15       15 test 0.1025641 0.8974359        0.9310345              0.8
## 16       16 test 0.1025641 0.8974359        0.9310345              0.8
## 17       17 test 0.1025641 0.8974359        0.9310345              0.8
## 18       18 test 0.1025641 0.8974359        0.9310345              0.8
## 21       21 test 0.1025641 0.8974359        0.9310345              0.8
```

Prediction accuracy on the test set increased to 89.7%. In this case the optimal values for k seem to be 11, between 15 and 18 and from 21 and higher, while the other values achieve the lowest accuracy (84.6%). The precision of the two classes has also increased compared to the previous model, particularly for the *Bottle* class which goes from a 0.73 to a 0.80 (the same of the logistic regression).

We can show how the value of accuracy varies according to the values of k by means of a plot:

```
# plot test set results (accuracy vs number of k)
ggplot(results_df_test_2, aes(x=k_number, y=accuracy)) +
    geom_line(colour="#D77099", alpha=0.7, size=1) +
    geom_point(colour="#D77099", alpha=0.7, size=3) +
    scale_x_continuous(breaks = results_df_test_2$k_number) +
    xlab("k")
```



As we can see, the accuracy values for the best and worst ks oscillate slightly (from 0.846 to 0.897); this is probably due to just one or a few more correctly classified observations.

Nevertheless, it is important to note that in order to find the best k-value, a k-fold cross-validation or a nested cross-validation would be more appropriate.

5. Fit a Naïve Bayes classifier.

The last model we will fit is the Naïve Bayes classifier. Naïve Bayes is a classifier which uses the Bayes theorem, assuming conditional independence of predictors. It basically calculate the a-priori class probabilities and the conditional probabilities, and the observation is classified to the class that leads to the largest a-posteriori probability.

As before, we will fit and test both a full model (including all predictors) and a model using only the most significant predictors.

In this case we can use non-standardised sets, as the `naiveBayes()` function (which assumes independence of the predictor variables, and Gaussian distribution) already performs standardisation internally.

```r
# fit Naive Bayes using all predictors
naive_bayes_fit = naiveBayes(breast ~ ., data=breastfeed_train)

# see the fit output
naive_bayes_fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##    Bottle    Breast
## 0.2708333 0.7291667
##
## Conditional probabilities:
##         pregnancy
## Y              End Beginning
##   Bottle 0.5769231 0.4230769
##   Breast 0.6000000 0.4000000
##
##         howfed
## Y           Bottle    Breast
##   Bottle 0.6153846 0.3846154
##   Breast 0.3571429 0.6428571
##
##         howfedfr
## Y           Bottle    Breast
##   Bottle 0.6153846 0.3846154
##   Breast 0.2571429 0.7428571
##
##         partner
## Y           Single   Partner
##   Bottle 0.2307692 0.7692308
##   Breast 0.1142857 0.8857143
##
##         smokenow
## Y               No       Yes
##   Bottle 0.5000000 0.5000000
##   Breast 0.8714286 0.1285714
##
##         smokebf
## Y               No       Yes
##   Bottle 0.4230769 0.5769231
##   Breast 0.7285714 0.2714286
##
##         age
## Y            [,1]     [,2]
##   Bottle 26.84615 5.780604
##   Breast 28.72857 5.280320
##
##         educat
## Y            [,1]     [,2]
##   Bottle 16.76923 3.010750
##   Breast 18.72857 2.631589
##
##         ethnic
## Y           White Non-white
##   Bottle 0.8846154 0.1153846
```

```
##   Breast 0.4428571 0.5571429
```

As we can see, the output of the fit first returns the original call to the function. In the subsequent section, the a-priori probabilities are shown; note also that we had already calculated them by hand in the data exploration. In the last section of the output the model creates and shows the conditional probability for each predictor separately.

Let us now see how the model performs on the test set:

```
# get posterior probabilities on test set
predict(naive_bayes_fit, newdata=breastfeed_test, type="raw")
```

```
##            Bottle     Breast
##  [1,] 0.005850281 0.99414972
##  [2,] 0.177284991 0.82271501
##  [3,] 0.952400211 0.04759979
##  [4,] 0.398473977 0.60152602
##  [5,] 0.120887193 0.87911281
##  [6,] 0.244353307 0.75564669
##  [7,] 0.039706454 0.96029355
##  [8,] 0.013071813 0.98692819
##  [9,] 0.966120059 0.03387994
## [10,] 0.957887368 0.04211263
## [11,] 0.024259921 0.97574008
## [12,] 0.016331220 0.98366878
## [13,] 0.007250718 0.99274928
## [14,] 0.978580855 0.02141915
## [15,] 0.901889462 0.09811054
## [16,] 0.050800969 0.94919903
## [17,] 0.967173529 0.03282647
## [18,] 0.891529999 0.10847000
## [19,] 0.010580658 0.98941934
## [20,] 0.052662902 0.94733710
## [21,] 0.003253181 0.99674682
## [22,] 0.004669946 0.99533005
## [23,] 0.038632162 0.96136784
## [24,] 0.766782005 0.23321799
## [25,] 0.974005517 0.02599448
## [26,] 0.027286903 0.97271310
## [27,] 0.909656077 0.09034392
## [28,] 0.007921728 0.99207827
## [29,] 0.031870561 0.96812944
## [30,] 0.453891632 0.54610837
## [31,] 0.176362374 0.82363763
## [32,] 0.897900003 0.10210000
## [33,] 0.024541403 0.97545860
## [34,] 0.150306803 0.84969320
## [35,] 0.006354568 0.99364543
## [36,] 0.532040895 0.46795911
## [37,] 0.895881467 0.10411853
## [38,] 0.195939806 0.80406019
## [39,] 0.382485964 0.61751404
```

```
# get predicted label
nb_pred = predict(naive_bayes_fit, newdata=breastfeed_test)

# confusion matrix
table(nb_pred, breastfeed_test$breast)
```

```
##
## nb_pred  Bottle Breast
##   Bottle      8      5
##   Breast      2     24
```

```r
# accuracy
nb_fit_accuracy = mean(nb_pred == breastfeed_test$breast)
nb_fit_accuracy
```

```
## [1] 0.8205128
```

```r
# build a df containing ground truth and predictions
nb_prev_vs_truth = breastfeed_test %>%
  select(breast) %>%
  mutate(nb_prediction = nb_pred)

# "Breast" class precision
yardstick::precision(nb_prev_vs_truth, truth=breast, estimate=nb_prediction, event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.923
```

```r
# "Bottle" class precision
yardstick::precision(nb_prev_vs_truth, truth=breast, estimate=nb_prediction, event_level="first")
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.615
```

As we can see, the model achieves an accuracy of 82.1%, failing to classify 7 observations correctly. Of the misclassified observations, most belonged to the *Bottle* category, which in fact only achieved a precision of 0.62, while the other class shows a quite good precision of 0.92.

We now fit the model using only significant predictors:

```r
# fit Naive Bayes using significant predictors
naive_bayes_fit_2 = naiveBayes(breast ~ howfedfr + smokenow + ethnic, data=breastfeed_train)

# see the fit output
naive_bayes_fit_2
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##    Bottle    Breast
## 0.2708333 0.7291667
##
## Conditional probabilities:
##         howfedfr
## Y           Bottle    Breast
##   Bottle 0.6153846 0.3846154
##   Breast 0.2571429 0.7428571
##
##         smokenow
## Y               No       Yes
##   Bottle 0.5000000 0.5000000
```

```
##   Breast 0.8714286 0.1285714
##
##         ethnic
## Y            White Non-white
##   Bottle 0.8846154 0.1153846
##   Breast 0.4428571 0.5571429
```

Obviously, the a-priori probability of the two classes is unchanged, while the conditional probabilities with respect to the predictors are adjusted.

Let's see how this last model performs on the test set:

```
# get posterior probabilities on test set
predict(naive_bayes_fit_2, newdata=breastfeed_test, type="raw")
```

```
##           Bottle    Breast
##  [1,] 0.02234105 0.9776589
##  [2,] 0.18060133 0.8193987
##  [3,] 0.59901691 0.4009831
##  [4,] 0.09553426 0.9044657
##  [5,] 0.09553426 0.9044657
##  [6,] 0.18060133 0.8193987
##  [7,] 0.18060133 0.8193987
##  [8,] 0.09553426 0.9044657
##  [9,] 0.87349783 0.1265022
## [10,] 0.87349783 0.1265022
## [11,] 0.02234105 0.9776589
## [12,] 0.02234105 0.9776589
## [13,] 0.02234105 0.9776589
## [14,] 0.87349783 0.1265022
## [15,] 0.50464909 0.4953509
## [16,] 0.18060133 0.8193987
## [17,] 0.87349783 0.1265022
## [18,] 0.87349783 0.1265022
## [19,] 0.02234105 0.9776589
## [20,] 0.09553426 0.9044657
## [21,] 0.02234105 0.9776589
## [22,] 0.02234105 0.9776589
## [23,] 0.18060133 0.8193987
## [24,] 0.59901691 0.4009831
## [25,] 0.87349783 0.1265022
## [26,] 0.02234105 0.9776589
## [27,] 0.50464909 0.4953509
## [28,] 0.02234105 0.9776589
## [29,] 0.09553426 0.9044657
## [30,] 0.18060133 0.8193987
## [31,] 0.18060133 0.8193987
## [32,] 0.59901691 0.4009831
## [33,] 0.02234105 0.9776589
## [34,] 0.18060133 0.8193987
## [35,] 0.02234105 0.9776589
## [36,] 0.50464909 0.4953509
## [37,] 0.87349783 0.1265022
## [38,] 0.18060133 0.8193987
## [39,] 0.50464909 0.4953509
```

```
# get predicted label
nb_pred_2 = predict(naive_bayes_fit_2, newdata=breastfeed_test)

# confusion matrix
table(nb_pred_2, breastfeed_test$breast)
```

```
##
## nb_pred_2 Bottle Breast
##    Bottle      9      5
##    Breast      1     24
```

```
# accuracy
nb_fit_accuracy_2 = mean(nb_pred_2 == breastfeed_test$breast)
nb_fit_accuracy_2
```

```
## [1] 0.8461538
```

```
# build a df containing ground truth and predictions
nb_prev_vs_truth_2 = breastfeed_test %>%
  select(breast) %>%
  mutate(nb_prediction = nb_pred_2)

# "Breast" class precision
yardstick::precision(nb_prev_vs_truth_2, truth=breast, estimate=nb_prediction, event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary          0.96
```

```
# "Bottle" class precision
yardstick::precision(nb_prev_vs_truth_2, truth=breast, estimate=nb_prediction, event_level="first")
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.643
```

Performance on the test set actually increased to 84.6%, correctly classifying one more observation than the previous model. Since these observation belongs to the *Bottle* class, also the class precision has increased.

6. Evaluate the performance of the methods and compare the results.

After fitting and testing different models, we can finally compare their performances.

We first compare the models in terms of accuracy/error rate on test set's performance:

```
# models labels
models_lab = c("logistic regression (all predictors)",
               "logistic regression (only significant predictors)",
               "Naïve Bayes (all predictors)",
               "Naïve Bayes (only significant predictors)")
# models accuracy
models_acc = c(lr_fit_accuracy$.estimate, lr_fit_accuracy_2$.estimate,
               nb_fit_accuracy, nb_fit_accuracy_2)

# initialize dataframe for storing models' accuracy
model_results = data.frame(model = models_lab, accuracy=models_acc)

# add different k k-nn all predictors model results
temp_knn = results_df_test %>%
  select(k_number, accuracy) %>%
  group_by(accuracy) %>% # aggregate k with same accuracy
  mutate(k_number = paste("k-nn (all predictors) k:", min(k_number), "-",
```

```
                          max(k_number))) %>%
  rename(model = k_number) %>%
  distinct(model, accuracy)

# add different k k-nn only significant predictors model results
temp_knn_2 = results_df_test_2 %>%
  select(k_number, accuracy) %>%
  # aggregate k with same accuracy
  mutate(k_number = ifelse(accuracy == max(accuracy),
                           "k-nn (only significant predictors) k: 11, 15-18, 21-30",
                           "k-nn (only significant predictors) k: 1-10, 12-14, 19-20")) %>%
  rename(model = k_number) %>%
  distinct(model, accuracy)

# combine all the results
model_accuracy = rbind(model_results, temp_knn, temp_knn_2)
# show table using knitr for a better appearance on the PDF file
knitr::kable(model_accuracy[order(-model_accuracy$accuracy),])
```

	model	accuracy
2	logistic regression (only significant predictors)	0.8974359
9	k-nn (only significant predictors) k: 11, 15-18, 21-30	0.8974359
7	k-nn (all predictors) k: 20 - 30	0.8717949
1	logistic regression (all predictors)	0.8461538
4	Naïve Bayes (only significant predictors)	0.8461538
6	k-nn (all predictors) k: 5 - 19	0.8461538
8	k-nn (only significant predictors) k: 1-10, 12-14, 19-20	0.8461538
3	Naïve Bayes (all predictors)	0.8205128
5	k-nn (all predictors) k: 1 - 4	0.7692308

As we can see from the ranking, the logistic regression and best ks' k-nn models that include only the most significant predictors achieve the highest accuracy (89.7%). Then we find the best ks' complete k-nn model, which reaches 87.2% accuracy. This is followed by a group of models obtaining 84.6% accuracy (complete logistic regression model, only significant predictors' Naïve Bayes model, complete k-nn with k range from 5 to 19 and only significant predictors' k-nn with k range 1-10, 12-14, 19-20). As the last models we have the complete Naïve Bayes and the worst k's complete k-nn, which achieve 82% and 76.9% accuracy respectively. Moreover, as we have seen in the individual models' sections, the increase in accuracy is associated with an increase in precision, especially for the smallest class (*Bottle*), which achieves a maximum of 0.8 in the models with the highest accuracy; however, the largest class (*Breast*) still achieves higher precision values, exceeding the 0.90 threshold in the best models.

Comparing models in terms of accuracy can certainly give us a clear idea of which models are the most successful and efficient; however, it is also important to consider other ways of assessing model performance. In this respect we must remember that the models were tested using the classical probability cut-off of 0.5 and the response variable classes are imbalanced. For these reasons, in evaluating these model is also useful to see how the performances change with the probability cutoff, evaluating the effectiveness of the model with different classification threshold. The ideal situation will find that sweet spot between a low overall error rate and a low per-class error rate (precision). In this respect the ROC curve is a useful tool that allow us to test several thresholds for the posterior probabilities and save the performance metrics (recall/true positive rate and true negative rate) of the model for each threshold, summarizing the two types of errors for all cut-offs.

We can easily calculate the true positive and negative rates by looking at the confusion matrix of the models' prediction and its values. Defining matrix value as true positive, true negative, false positive and false negative it is important to specify that the response value's label chosen to be "positive" will be *Breast* and, consequently, the "negative" label will be *Bottle*. For each model we then calculate the true positive (sensitivity) and true negative rates (specificity) associated with each probability cut-off threshold (between 0 and 1):

```
# get the augmented test set for logistic regression (all predictors)
augmented_ts = augment(lr_fit, new_data=breastfeed_test)
# compute the ROC object:
roc_lr_fit = augmented_ts %>%
```

```
    roc_curve(truth=breast, estimate=.pred_Breast, event_level="second")
roc_lr_fit$model = "logistic regression (all predictors)"

# get the augmented test set for logistic regression (only significant predictors)
augmented_ts = augment(lr_fit2, new_data=breastfeed_test)
# compute the ROC object:
roc_lr_fit2 = augmented_ts %>%
    roc_curve(truth=breast, estimate=.pred_Breast, event_level="second")
roc_lr_fit2$model = "logistic regression (only significant predictors)"

# for knn we take the best k's models:
# set seed for reproducible results
set.seed(42)
# fit the knn classifier (all predictors)
knn_spec = nearest_neighbor(neighbors=20) %>%
  set_mode("classification") %>%
  set_engine("kknn")
knn_fit = knn_spec %>%
  fit(breast ~ ., data=breastfeed_knn_train)
# get the augmented test set
augmented_ts_knn = augment(knn_fit, new_data=breastfeed_knn_train)
# compute the ROC object:
roc_knn = augmented_ts_knn %>%
    roc_curve(truth=breast, estimate=.pred_Breast, event_level="second")
roc_knn$model = "k-nn model (all predictors)"

# fit the knn classifier (only significant predictors)
knn_spec_2 = nearest_neighbor(neighbors=11) %>%
  set_mode("classification") %>%
  set_engine("kknn")
knn_fit_2 = knn_spec_2 %>%
  fit(breast ~ howfedfr + smokenow + ethnic, data=breastfeed_knn_train)
# get the augmented test set
augmented_ts_knn = augment(knn_fit_2, new_data=breastfeed_knn_train)
# compute the ROC object:
roc_knn2 = augmented_ts_knn %>%
    roc_curve(truth=breast, estimate=.pred_Breast, event_level="second")
roc_knn2$model = "k-nn model (only significant predictors)"

# put together the results
roc_mod = rbind(roc_lr_fit, roc_lr_fit2, roc_knn, roc_knn2)
```

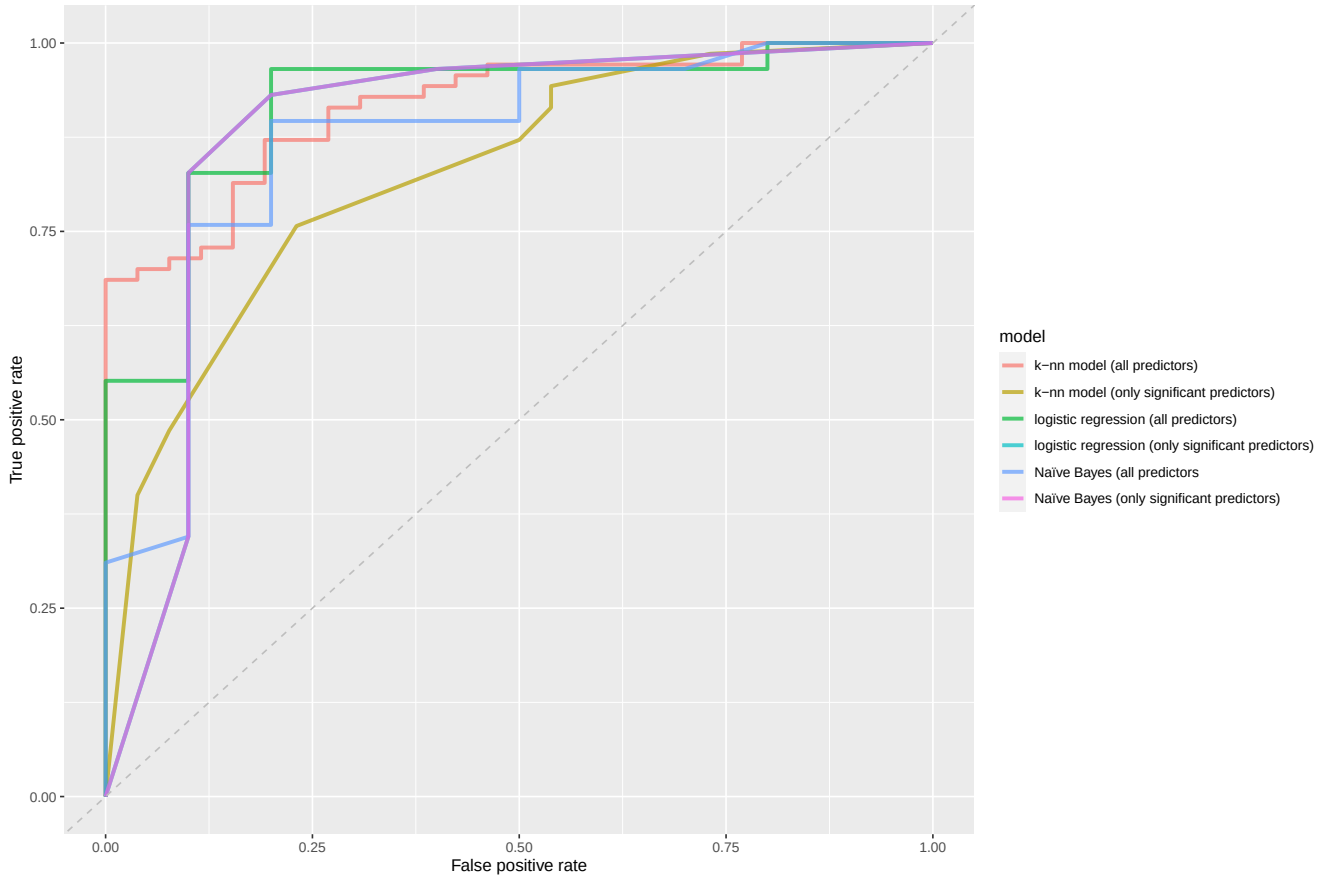
In order to compute Naïve Bayes ROC object we need a different procedure (since we did not use `tidymodels`):

```
# function to compute nb roc object
nb_roc = function(nb_mod, mod_name) {
  # placeholders for storing sensitivity, specificity and threshold
  all_sens = all_spec = c()
  threshold = c()
  # get predictions' probabilities
  nb_prob = as.data.frame(predict(nb_mod, newdata=breastfeed_test, type="raw"))
  # keep only "positive" probabilities
  nb_prob = nb_prob$Breast

  # assign test set labels
  y_ts = breastfeed_test$breast

  # varing cut off from 0 to 1 so to have 500 different values of cutoff
  for(prob_cutoff in seq(0, 1, length.out=500)) {

      sens = sum(nb_prob >= prob_cutoff & y_ts == "Breast") /
```

```
        sum(y_ts == "Breast") # compute sensitivity
      fpr = sum(nb_prob < prob_cutoff & y_ts == "Bottle") /
        sum(y_ts == "Bottle") # compute specificity

      all_sens = append(all_sens, sens)
      all_spec = append(all_spec, fpr)
      threshold = append(threshold, prob_cutoff)
  }
  # putting results in the roc object df
  roc_nb = data.frame(.threshold=threshold, specificity=all_spec,
                      sensitivity=all_sens, model=mod_name)
}

# compute roc objects
roc_nb = nb_roc(naive_bayes_fit, "Naïve Bayes (all predictors")
roc_nb2 = nb_roc(naive_bayes_fit_2, "Naïve Bayes (only significant predictors)")

# add roc object to the all models' one
roc_mod = rbind(roc_mod, roc_nb, roc_nb2)
```

Finally we can plot the models' ROC curves together:

```
ggplot(roc_mod, aes(x=1-specificity, y=sensitivity, group=model, color=model)) +
  # geom_point() +
  geom_path(size=1.2, alpha=0.7) +
  xlab("False positive rate") +
  ylab("True positive rate") +
  # line corresponding to the "no information" classifier (e.g., random guess)
  geom_abline(linetype = "dashed", color="gray")
```



The ROC plot gives us a graphical idea of how the models perform with various probability cut-offs and make easy to identify the best decision threshold for each model (represented by the nearest point to the top left edges of the

graph). In these terms a way to quantify the performance of each model is to compute the area under the ROC curve, named AUC:

```
# function to calculate AUC for logistic regression and k-nn models
mod_auc = function(mod, mod_name) {
  # get the augmented test set and compute AUC
  auc = augment(mod, new_data=breastfeed_test) %>%
    roc_auc(breast, .pred_Breast, event_level="second")
  # tidy up the model's AUC df
  auc = auc %>%
    select(.estimate) %>%
    rename(AUC = .estimate) %>%
    mutate(model = mod_name)
}

# compute AUC
lr_fit_auc = mod_auc(lr_fit, "logistic regression (all predictors)")
lr_fit2_auc = mod_auc(lr_fit2, "logistic regression (only significant predictors)")
knn_fit_auc = mod_auc(knn_fit, "k-nn (all predictors)")
knn_fit_2_auc = mod_auc(knn_fit_2, "k-nn (only significant predictors)")

# function to calculate AUC for Naïve Bayes models
mod_auc_nb = function(nb_roc, mod_name) {
  # initialize model's AUC df
  nb_auc = data.frame(AUC=0, model=mod_name)
  # calculate AUC (area under the curve)
  height = (nb_roc$sensitivity[-1] + nb_roc$sensitivity[-length(nb_roc$sensitivity)]) / 2
  width = -diff(1-nb_roc$specificity)
  nb_auc$AUC = sum(height*width)
  nb_auc
}

# compute AUC
nb_fit_auc = mod_auc_nb(roc_nb, "Naïve Bayes (all predictors)")
nb_fit_2_auc = mod_auc_nb(roc_nb2, "Naïve Bayes (only significant predictors)")

# putting the models' AUC values together
models_auc = rbind(lr_fit_auc, lr_fit2_auc, knn_fit_auc, knn_fit_2_auc, nb_fit_auc, nb_fit_2_auc)
# show table using knitr for a better appearance on the PDF file
knitr::kable(models_auc[order(-models_auc$AUC),])
```

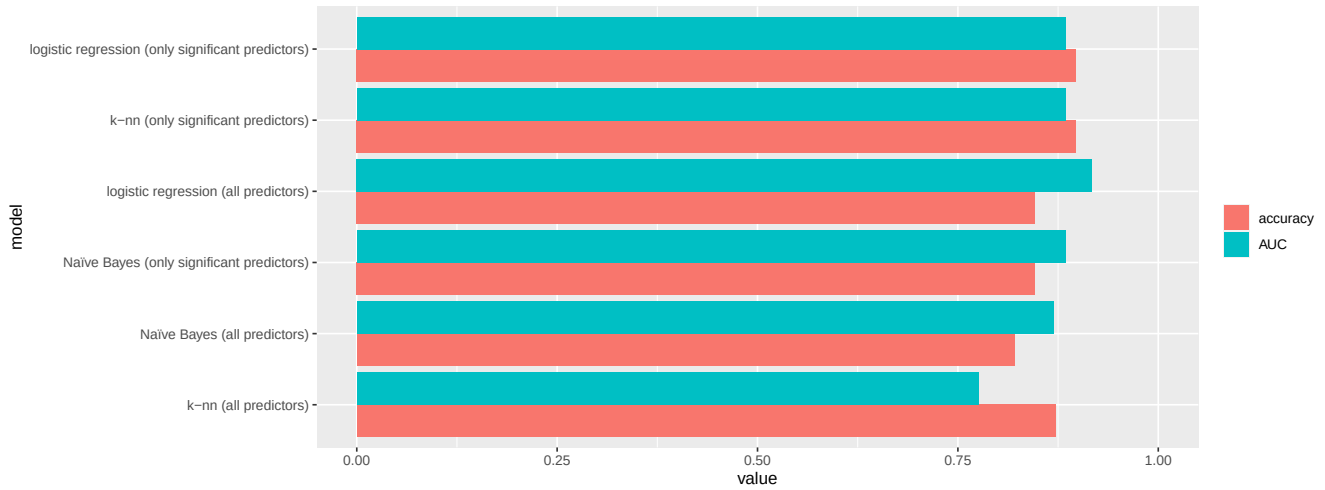
AUC	model
0.9172414	logistic regression (all predictors)
0.8844828	logistic regression (only significant predictors)
0.8844828	k-nn (only significant predictors)
0.8844828	Naïve Bayes (only significant predictors)
0.8689655	Naïve Bayes (all predictors)
0.7758621	k-nn (all predictors)

In terms of AUC, the best model is the logistic regression including all predictors (point 2), which reaches 0.92; this is followed by the logistic regression, k-nn and Naïve Bayes classifiers including only significant predictors, which reach 0.88. We can see that the ranking is different from the one based on accuracy: this shows us how the models behave differently with different probability cut-offs.

We can summarise the evaluation of the models in a stacked barplot showing the accuracy and AUC values for each model:

```
# tidy up accuracy df deleting minor models and prepare to be plotted together with AUC df
final_model_accuracy = model_accuracy %>%
  mutate(model = ifelse(model == "k-nn (all predictors) k: 20 - 30", "k-nn (all predictors)",
```

```
                        ifelse(model == "k-nn (only significant predictors) k: 11, 15-18, 21-30",
                               "k-nn (only significant predictors)", model))) %>%
  filter(!(model == "k-nn (all predictors) k: 1 - 4" |
             model == "k-nn (all predictors) k: 5 - 19" |
             model == "k-nn (only significant predictors) k: 1-10, 12-14, 19-20")) %>%
  rename(value = accuracy) %>%
  mutate(type = "accuracy")

# prepare data to be plotted together with accuracy df
final_model_auc = models_auc %>%
  rename(value = AUC) %>%
  mutate(type = "AUC")

# put together AUC and accuracy values
model_evaluation_df = rbind(final_model_accuracy, final_model_auc)

# plot AUC and accuracy for each model
ggplot(model_evaluation_df, aes(fill=type, y=value, x=reorder(model, value))) +
  geom_bar(position="dodge", stat="identity") +
  coord_flip() +
  ylim(0:1) +
  xlab("model") +
  theme(legend.title=element_blank())
```



The best models in terms of both AUC and accuracy seem to be logistic regression and k-nn. On the other hand, the Naive Bayes classifier obtains a good AUC value (equal to the other models) but obtaining slightly lower accuracy values than the competitors.

As a final remark, it should also be noted that the test and training set partitions and the models' performances may differ slightly depending on the chosen seeds.