Classification task: Naive Bayes (NB)

The prediction task: to predict which mobile phones can be soled for a high price. The rationale: help a company determine its price policy; more precisely, help it identify which phones it can offer to its customers on a high price.

The algorithm to use: Naive Bayes (NB)

Load and inspect the data

The data originates from the Mobile Price Classification dataset, where further information about the data set can be found, including the description of all the variables. Note: the original data set has been adapted for this class.

Load the data

```r
mobiles_ds <- read.csv("data/mobiles.csv")
```

Inspect the data

```r
str(mobiles_ds)
```

```
## 'data.frame':    2000 obs. of  13 variables:
##  $ battery_power: int  842 1021 563 615 1821 1859 1821 1954 1445 509 ...
##  $ clock_speed  : num  2.2 0.5 0.5 2.5 1.2 0.5 1.7 0.5 0.5 0.6 ...
##  $ dual_sim     : int  0 1 1 0 0 1 0 1 0 1 ...
##  $ int_memory   : int  7 53 41 10 44 22 10 24 53 9 ...
##  $ m_dep        : num  0.6 0.7 0.9 0.8 0.6 0.7 0.8 0.8 0.7 0.1 ...
##  $ mobile_wt    : int  188 136 145 131 141 164 139 187 174 93 ...
##  $ n_cores      : int  2 3 5 6 2 1 8 4 7 5 ...
##  $ pc           : int  2 6 6 9 14 7 10 0 14 15 ...
##  $ ram          : int  2549 2631 2603 2769 1411 1067 3220 700 1099 513 ...
##  $ talk_time    : int  19 7 9 11 15 10 18 5 20 12 ...
##  $ price_range  : int  1 2 2 2 1 1 3 0 0 0 ...
##  $ four_g       : int  0 1 1 0 1 0 1 0 0 1 ...
##  $ color        : chr  "black" "white" "white" "gray" ...
```

```r
summary(mobiles_ds)
```

```
##  battery_power     clock_speed       dual_sim        int_memory
##  Min.   : 501.0   Min.   :0.500   Min.   :0.0000   Min.   : 2.00
##  1st Qu.: 851.8   1st Qu.:0.700   1st Qu.:0.0000   1st Qu.:16.00
##  Median :1226.0   Median :1.500   Median :1.0000   Median :32.00
##  Mean   :1238.5   Mean   :1.522   Mean   :0.5095   Mean   :32.05
##  3rd Qu.:1615.2   3rd Qu.:2.200   3rd Qu.:1.0000   3rd Qu.:48.00
##  Max.   :1998.0   Max.   :3.000   Max.   :1.0000   Max.   :64.00
##      m_dep          mobile_wt        n_cores           pc
##  Min.   :0.1000   Min.   : 80.0   Min.   :1.00   Min.   : 0.000
##  1st Qu.:0.2000   1st Qu.:109.0   1st Qu.:3.00   1st Qu.: 5.000
##  Median :0.5000   Median :141.0   Median :4.00   Median :10.000
##  Mean   :0.5018   Mean   :140.2   Mean   :4.52   Mean   : 9.916
##  3rd Qu.:0.8000   3rd Qu.:170.0   3rd Qu.:7.00   3rd Qu.:15.000
##  Max.   :1.0000   Max.   :200.0   Max.   :8.00   Max.   :20.000
##       ram         talk_time      price_range       four_g
##  Min.   : 256   Min.   : 2.00   Min.   :0.00   Min.   :0.0000
##  1st Qu.:1208   1st Qu.: 6.00   1st Qu.:0.75   1st Qu.:0.0000
##  Median :2146   Median :11.00   Median :1.50   Median :1.0000
##  Mean   :2124   Mean   :11.01   Mean   :1.50   Mean   :0.5215
##  3rd Qu.:3064   3rd Qu.:16.00   3rd Qu.:2.25   3rd Qu.:1.0000
##  Max.   :3998   Max.   :20.00   Max.   :3.00   Max.   :1.0000
##     color
##  Length:2000
##  Class :character
##  Mode  :character
##
##
##
```

We will use the `price_range` variable to create the outcome variable:

```
table(mobiles_ds$price_range)
```

```
##
##   0   1   2   3
## 500 500 500 500
```

Create the outcome variable - `top_price` - by considering that `price_range` value of 3 denotes top price:

```
mobiles_ds$top_price = ifelse(test = mobiles_ds$price_range == 3,
                              yes="Yes", no="No")
mobiles_ds$top_price[1:10]
```

```
##  [1] "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "No"
```

Transform the outcome variable into factor, as required for the classification task

```
mobiles_ds$top_price = as.factor(mobiles_ds$top_price)
mobiles_ds$top_price[1:10]
```

```
##  [1] No  No  No  No  No  No  Yes No  No  No
## Levels: No Yes
```

Remove from the data set the variable used for creating the outcome variable

```
mobiles_ds$price_range = NULL
```

Data preparation

To prepare the data for Naive Bayes algorithm we need to:

- transform categorical variables into factors
- examine the distribution of numerical variables: variables with normal distribution (if any) can be directly used, while those that are not normally distributed need to be discretized

First, transform categorical variables

```r
mobiles_ds$color = as.factor(mobiles_ds$color)
mobiles_ds$four_g = factor(mobiles_ds$four_g, levels = c(0,1), labels = c("No","Yes"))
mobiles_ds$dual_sim = factor(mobiles_ds$dual_sim, levels = c(0,1), labels = c("No","Yes"))
```

Next, check if numerical variables are normally distributed

```r
numeric_var_indices = c(1,2,4:10)
apply(mobiles_ds[,numeric_var_indices], 2, function(x) shapiro.test(x)$p.value >= 0.05)
```

```
## battery_power   clock_speed    int_memory         m_dep     mobile_wt
##         FALSE         FALSE         FALSE         FALSE         FALSE
##       n_cores            pc           ram     talk_time
##         FALSE         FALSE         FALSE         FALSE
```

None is normally distributed, meaning that we will have to discretize all numerical variables.

To do the discretization seamlessly, we'll use the *discretize* f. from the *bnlearn* R package.

```r
library(bnlearn)

?discretize
```

The `discretize` function performs *unsupervised discretization*, which consists of dividing the value range of a continuous variable into sub-ranges (intervals) without taking into account the outcome variable or any other information. Specifically, it splits the value range of a numerical variable into k intervals, where k is given as the input parameter. The function can do the splitting in different ways, among which the following two are often used:

- split into k intervals of (roughly) equal lengths - this is known as *interval discretization*
- split into k intervals so that all intervals have (roughly) equal number of values - this is known as *quantile discretization*

Quantile discretization is often recommended as it tends to lead to better results. So, we will use it here.

We will try to split all numerical variables into 5 intervals (bins) since this is a typically used number of bins for data sets of the size such as ours (if we had more observations, we could have opted for having more bins).

```r
mobiles_num = mobiles_ds[, numeric_var_indices]

# note that the discretize function requires a data frame with all numeric columns
apply(mobiles_num, 2, as.numeric) |> as.data.frame() -> mobiles_num
```

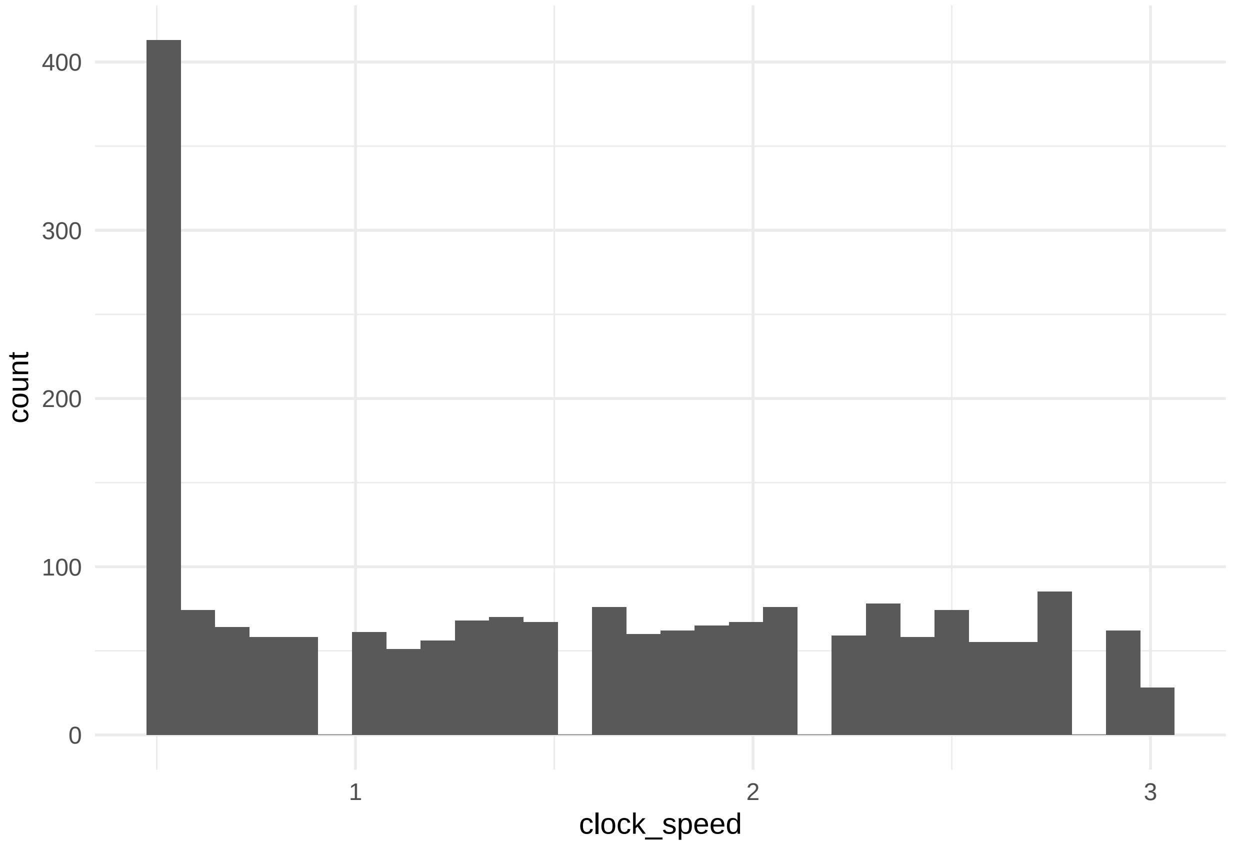
```
# mobiles_num_disc = discretize(data = mobiles_num,
#                               method = "quantile",
#                               breaks = 5)
```

Notice that the `clock_speed` variable could not be split into 5 segments of relatively equal frequencies. To understand why this is the case, we need to examine the distribution of this variable.

```
library(ggplot2)
```

```
ggplot(mobiles_num, aes(x = clock_speed)) + geom_histogram() + theme_minimal()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



From the plot, it is clear that this variable cannot be split into 5 intervals of (roughly) equal number of observations. It seems reasonable to try 3 bins.

```
mobiles_num_disc = discretize(data = mobiles_num,
                              method = "quantile",
                              breaks = c(5,3,5,5,5,5,5,5,5))
```

Let's see what the transformed variables look like, through summary statistics

```
summary(mobiles_num_disc)
```

```
##          battery_power         clock_speed    int_memory        m_dep
##  [501,781]      :401   [0.5,0.933333]:667   [2,13] :409   [0.1,0.2]:533
##  (781,1076]     :400   (0.933333,2]  :703   (13,25]:392   (0.2,0.4]:367
##  (1076,1395.4]  :399   (2,3]         :630   (25,38]:400   (0.4,0.6]:391
##  (1395.4,1698.2]:400                        (38,51]:416   (0.6,0.8]:408
##  (1698.2,1998]  :400                        (51,64]:383   (0.8,1]  :301
##     mobile_wt     n_cores         pc                    ram         talk_time
##  [80,103] :401   [1,2]:489   [0,4]  :492   [256,1016]     :400   [2,5]  :409
##  (103,128]:400   (2,4]:520   (4,8]  :362   (1016,1742.6]  :400   (5,9]  :439
##  (128,153]:412   (4,5]:246   (8,12] :403   (1742.6,2485.2]:400   (9,13] :407
##  (153,178]:401   (5,7]:489   (12,16]:369   (2485.2,3256.6]:400   (13,17]:430
##  (178,200]:386   (7,8]:256   (16,20]:374   (3256.6,3998]  :400   (17,20]:315
```

Merge the discretized variables with the rest of the columns (variables) from the *mobiles_ds* data frame.

```
# identify the columns to add as the difference between the vectors with variable names
cols_to_add <- setdiff(names(mobiles_ds), names(mobiles_num_disc))

cbind(mobiles_ds[,cols_to_add], mobiles_num_disc) -> mobiles_final
```

```
str(mobiles_final)
```

```
## 'data.frame':    2000 obs. of  13 variables:
##  $ dual_sim     : Factor w/ 2 levels "No","Yes": 1 2 2 1 1 2 1 2 1 2 ...
##  $ four_g       : Factor w/ 2 levels "No","Yes": 1 2 2 1 2 1 2 1 1 2 ...
##  $ color        : Factor w/ 3 levels "black","gray",..: 1 3 3 2 3 1 3 1 2 3 ...
##  $ top_price    : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 1 ...
##  $ battery_power: Factor w/ 5 levels "[501,781]","(781,1076]",..: 2 2 1 1 5 5 5 5 4 1 ...
##  $ clock_speed  : Factor w/ 3 levels "[0.5,0.933333]",..: 3 1 1 3 2 1 2 1 1 1 ...
##  $ int_memory   : Factor w/ 5 levels "[2,13]","(13,25]",..: 1 5 4 1 4 2 1 2 5 1 ...
##  $ m_dep        : Factor w/ 5 levels "[0.1,0.2]","(0.2,0.4]",..: 3 4 5 4 3 4 4 4 4 1 ...
##  $ mobile_wt    : Factor w/ 5 levels "[80,103]","(103,128]",..: 5 3 3 3 3 4 3 5 4 1 ...
##  $ n_cores      : Factor w/ 5 levels "[1,2]","(2,4]",..: 1 2 3 4 1 1 5 2 4 3 ...
##  $ pc           : Factor w/ 5 levels "[0,4]","(4,8]",..: 1 2 2 3 4 2 3 1 4 4 ...
##  $ ram          : Factor w/ 5 levels "[256,1016]","(1016,1742.6]",..: 4 4 4 4 2 2 4 1 2 1 ...
##  $ talk_time    : Factor w/ 5 levels "[2,5]","(5,9]",..: 5 2 2 3 4 3 5 1 5 3 ...
```

To make things easier later, we'll change the order of variables in the data frame, so that the outcome variable is the last one.

```
mobiles_final = mobiles_final[,c(1:3,5:13,4)]
```

The data set is now ready and we can proceed to splitting the data into training and test sets and building models.

IMPORTANT NOTE: for now, we are considering all variables as potentially relevant for building a prediction model. In a week or two, we will learn how to select variables that are truly relevant candidates for model building.

Train-test split

We will split the data into training and test sets in the same way we did before.

```r
library(caret)
```

```r
set.seed(2024)
training_indices = createDataPartition(mobiles_final$top_price, p = 0.8, list = FALSE)
train_ds = mobiles_final[training_indices, ]
test_ds = mobiles_final[-training_indices, ]
```

Create a prediction model using the Naive Bayes algorithm

We will start by loading the **e1071** package that offers a handy function (**naiveBayes**) for building a classification model using the NB algorithm.

For explanation of the NB algorithm, see the lecture slides.

```r
library(e1071)

?naiveBayes
```

```r
nb1 <- naiveBayes(top_price ~ ., data = train_ds)
```

Print the model to see the conditional probabilities that the `naiveBayes` function computed for each variable.

```r
print(nb1)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   No  Yes
## 0.75 0.25
##
## Conditional probabilities:
##      dual_sim
## Y            No       Yes
##   No  0.4966667 0.5033333
##   Yes 0.4700000 0.5300000
##
##      four_g
## Y            No       Yes
##   No  0.4916667 0.5083333
##   Yes 0.4575000 0.5425000
##
##      color
## Y         black      gray     white
##   No  0.4491667 0.2883333 0.2625000
##   Yes 0.4525000 0.2975000 0.2500000
##
##      battery_power
## Y      [501,781] (781,1076] (1076,1395.4] (1395.4,1698.2] (1698.2,1998]
##   No  0.2291667  0.2183333     0.2116667       0.1783333     0.1625000
##   Yes 0.0975000  0.1700000     0.1825000       0.2475000     0.3025000
##
##      clock_speed
## Y      [0.5,0.933333] (0.933333,2]     (2,3]
##   No       0.3308333    0.3475000 0.3216667
##   Yes      0.3300000    0.3575000 0.3125000
##
##      int_memory
## Y         [2,13]   (13,25]   (25,38]   (38,51]   (51,64]
##   No  0.2066667 0.2125000 0.1941667 0.1925000 0.1941667
##   Yes 0.1750000 0.1875000 0.1900000 0.2425000 0.2050000
##
##      m_dep
## Y      [0.1,0.2] (0.2,0.4] (0.4,0.6] (0.6,0.8]   (0.8,1]
##   No  0.2666667 0.1883333 0.1833333 0.2066667 0.1550000
##   Yes 0.2825000 0.1650000 0.1900000 0.2100000 0.1525000
##
##      mobile_wt
## Y       [80,103] (103,128] (128,153] (153,178] (178,200]
##   No  0.1966667 0.1908333 0.2050000 0.2083333 0.1991667
##   Yes 0.2375000 0.2050000 0.2075000 0.1950000 0.1550000
##
##      n_cores
## Y          [1,2]     (2,4]     (4,5]     (5,7]     (7,8]
##   No  0.2500000 0.2666667 0.1075000 0.2333333 0.1425000
##   Yes 0.2400000 0.2375000 0.1375000 0.2625000 0.1225000
##
##      pc
## Y          [0,4]     (4,8]    (8,12]   (12,16]   (16,20]
##   No  0.2533333 0.1800000 0.1933333 0.1916667 0.1816667
##   Yes 0.2375000 0.2025000 0.1900000 0.1650000 0.2050000
##
##      ram
## Y      [256,1016] (1016,1742.6] (1742.6,2485.2] (2485.2,3256.6] (3256.6,3998]
##   No  0.26833333    0.26500000      0.25666667      0.17583333    0.03416667
##   Yes 0.00000000    0.00000000      0.02500000      0.28500000    0.69000000
##
##      talk_time
## Y          [2,5]     (5,9]    (9,13]   (13,17]   (17,20]
##   No  0.2166667 0.2183333 0.2025000 0.2141667 0.1483333
##   Yes 0.1975000 0.2325000 0.2000000 0.1925000 0.1775000
```

For each input variable, we have conditional probabilities for each possible value of the variable given a particular value of the outcome variables. For example, in case of the `n_cores` variable, if a mobile has top price (top_price="Yes"), the probability that the mobile has 7 or 8 cores is 0.12. Or in case of the `color` variable, if a phone is not in the top price category (top_price="No"), the probability that the phone is white is 0.26.

We will proceed to evaluate the model on the test set, as was done before (for classification trees).

First, make the predictions on the test data set

```r
nb1_pred <- predict(nb1, newdata = test_ds, type = 'class')

head(nb1_pred)
```

```
## [1] No  No  Yes No  No  Yes
## Levels: No Yes
```

Then, create the confusion matrix:

```r
nb1_cm <- table(true = test_ds$top_price, predicted = nb1_pred)
nb1_cm
```

```
##      predicted
## true   No Yes
##   No  293   7
##   Yes  16  84
```

Load the function for computing the evaluation measures from the util.R script

```r
source("util.R")
```

```r
compute_eval_measures_v1(nb1_cm)
```

```
##  accuracy precision    recall        F1
##    0.9425    0.9231    0.8400    0.8796
```

Finding optimal probability threshold using ROC curves

When doing predictions using the initial model (nb1), we relied on the default classification threshold of 0.5. That is, the `predict` function used that threshold to associate class labels ("Yes", "No") with each observation based on its posterior probability estimated by the NB model (recall the Bayesian formula that NB uses). However, we do not need to use the value of 0.5, but can use any other probability value as the threshold. To find the optimal one, we use the ROC curve (see the lecture slides for further information about the ROC curve).

ROC curve enables us to choose the probability threshold that will result in the desired *specificity* vs *sensitivity* trade-off.

Sensitivity (*True Positive Rate - TPR*) measures the proportion of positives that are correctly predicted as such; it is the same as *Recall*

Specificity (*True Negative Rate - TNR)*) measures the proportion of negatives that are correctly predicted as such.

By selecting the probability threshold, we make a decision if we want to favour the prediction of positive class (Sensitivity), the prediction of negative class (Specificity), or we want a model that seeks to maximize both measures.

To make use of a ROC curve to find the optimal probability threshold, we need to:

- For each observation, compute the probability of having each class value ('Yes' / 'No')

- Use the computed probabilities to create a ROC curve
- Use the ROC curve to choose the classification (probability) threshold

First, for the observations in the test set, compute probabilities for each class value

```
nb1_pred_prob <- predict(nb1, newdata = test_ds, type = "raw")
head(nb1_pred_prob)
```

```
##               No         Yes
## [1,] 0.99817421 0.001825786
## [2,] 0.99767378 0.002326219
## [3,] 0.08234195 0.917658053
## [4,] 0.96589390 0.034106103
## [5,] 0.98929205 0.010707953
## [6,] 0.06704617 0.932953829
```

Second, create a ROC curve, using the `roc` function from the **pROC** package

```
library(pROC)

?roc
```

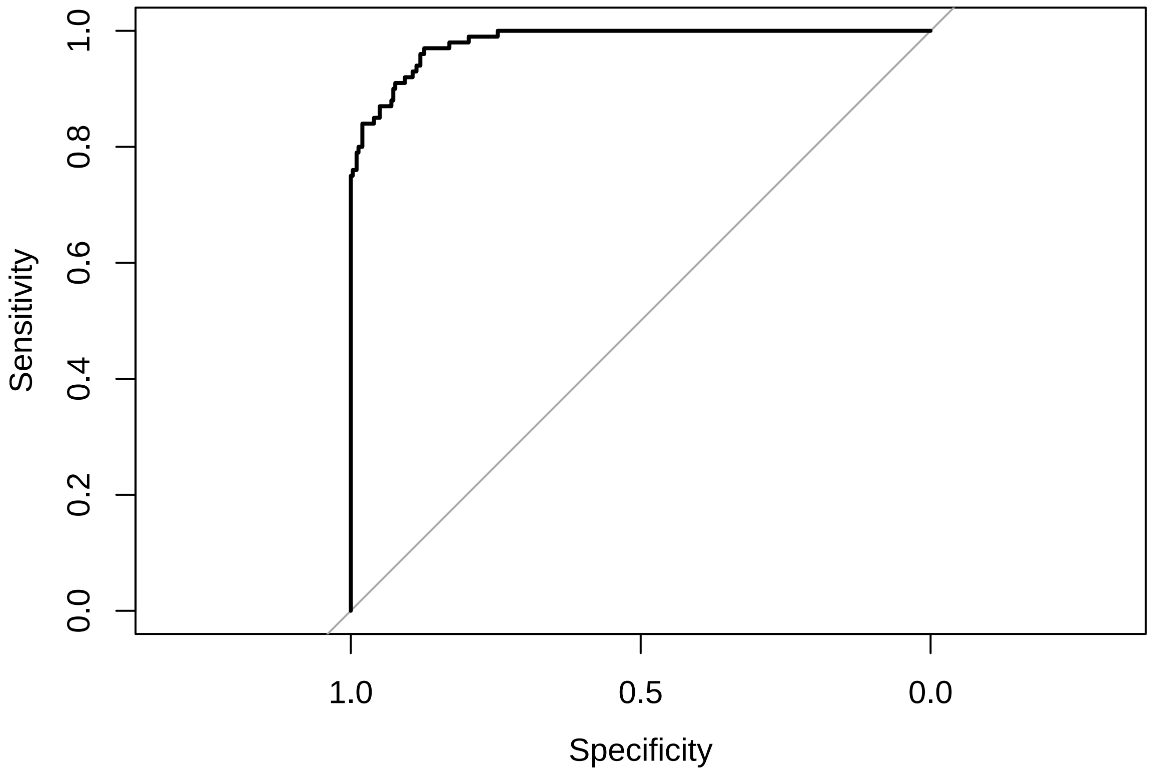
```
nb1_roc <- roc(response = as.numeric(test_ds$top_price),
               predictor = nb1_pred_prob[,2],
               levels = c(1, 2))
```

```
## Setting direction: controls < cases
```

Note: the `roc` function uses somewhat peculiar terminology (originating from experimental research): *controls* stands for observations of the negative class, while *cases* stands for observations of the positive class

Plot the ROC curve

```
plot.roc(nb1_roc)
```



Local maximas of the ROC curve serve as candidates for the classification threshold. To identify those candidates and how the selection of each one would affect the performance of the classifier, we will use the `coords` function. In particular, for each local maximum (= threshold candidate), the function computes a set of evaluation measures (defined by the `ret` parameter) that characterise the classification performance that would result from choosing that local maximum as the classification threshold:

```
nb1_coords <- coords(nb1_roc,
                     ret = c("accuracy", "spec", "sens", "precision", "thr"),
                     x = "local maximas",
                     transpose = FALSE)
nb1_coords
```

```
##    accuracy specificity sensitivity precision  threshold
## 1    0.8100   0.7466667        1.00 0.5681818 0.04946290
## 2    0.8450   0.7966667        0.99 0.6187500 0.08127616
## 3    0.8675   0.8300000        0.98 0.6577181 0.16324659
## 4    0.8975   0.8733333        0.97 0.7185185 0.25243527
## 5    0.9000   0.8800000        0.96 0.7272727 0.26009164
## 6    0.9000   0.8866667        0.94 0.7343750 0.27946244
## 7    0.9025   0.8933333        0.93 0.7440000 0.29700079
## 8    0.9100   0.9066667        0.92 0.7666667 0.34112224
## 9    0.9200   0.9233333        0.91 0.7982456 0.37434496
## 10   0.9200   0.9266667        0.90 0.8035714 0.38400411
## 11   0.9175   0.9300000        0.88 0.8073394 0.39842941
## 12   0.9300   0.9500000        0.87 0.8529412 0.43968848
## 13   0.9325   0.9600000        0.85 0.8762887 0.47863838
## 14   0.9450   0.9800000        0.84 0.9333333 0.51733844
## 15   0.9400   0.9866667        0.80 0.9523810 0.63893227
## 16   0.9400   0.9900000        0.79 0.9634146 0.68715928
## 17   0.9375   0.9966667        0.76 0.9870130 0.71827304
## 18   0.9375   1.0000000        0.75 1.0000000 0.73984462
```

We choose the threshold based on what we want to achieve - maximum sensitivity (recall) or maximum precision or balanced specificity and sensitivity or … The choice depends on the requirements of each classification task.

In the context of the current task - distinguishing between top priced phones and those that are not, we would be probably interested in having both high specificity and sensitivity (= high prediction rate of both positive and negative classes), while also not harming precision too much (= not making many false positives), we can choose the threshold given in the 9th or 10th row in the `nb1_coords` data frame:

```
opt_threshold = nb1_coords$threshold[9]
```

Note: if we wanted to choose the threshold that would maximize the sum of sensitivity and specificity measures - one of the often used criteria for the threshold selection - in the `coords` f. we should set the value of the `x` parameter to "best" and the value of the `best.method` parameter to "youden" (see below)

```
# coords(nb1_roc,
#        ret = c("accuracy", "spec", "sens", "precision", "thr"),
#        x = "best",
#        best.method = "youden") -> nb1_youden
```

Now, we use the chosen threshold to make predictions: for each observation, if the probability of belonging to the positive class is equal to or above the chosen classification threshold, the label of the positive class is assigned to the observation

```
ifelse(nb1_pred_prob[,2] >= opt_threshold, "Yes", "No") |> as.factor() -> nb1_opt_pred
```

```
head(nb1_opt_pred)
```

```
## [1] No  No  Yes No  No  Yes
## Levels: No Yes
```

Now that we have each observation associated with a class label, we can create the confusion matrix and compute evaluation measures:

```
nb1_opt_cm <- table(true = test_ds$top_price, predicted = nb1_opt_pred)
compute_eval_measures_v1(nb1_opt_cm)
```

```
##  accuracy precision    recall        F1
##    0.9200    0.7982    0.9100    0.8505
```

Note that the results are as expected, that is, as given in the 9th row of the above given (`nb1_coords`) data frame.

Compare the performance with the default and the new threshold:

```r
rbind(compute_eval_measures_v1(nb1_cm),
      compute_eval_measures_v1(nb1_opt_cm)) |>
  round(digits = 4) |>
  as.data.frame()
```

```
##   accuracy precision recall     F1
## 1   0.9425    0.9231   0.84 0.8796
## 2   0.9200    0.7982   0.91 0.8505
```

Compared to the initial predictions, we have significant increase in recall and at the same time significant drop in precision. In other words, we are making more mistakes of the false positive type - that is, classifying phones as top priced even when they are not such - while at the same time making less mistakes of the false negative type - that is, missing to identify phones that are top priced. Accuracy has dropped somewhat, but that performance measure is not of much importance in unbalanced data sets (= data sets where one class is far more represented that the other) such as the one we have here.

Final note: The use of ROC curve to select the optimal classification threshold can be applied to any classifier that computes probabilities of class membership (= probabilities of belonging to each of the classes), it is not restricted to NB.