Classification

Vladan Devedzic

(Installing and) Loading the required R packages

```
# install.packages("ggplot2")
library(ggplot2)
```

Dataset

The starting (expanded) dataset:

```
<dataframe> <- read.csv("<filename>",
+ stringsAsFactors = FALSE)
str(<dataframe>)                      # structure of <dataframe>, all variables/columns
summary(<dataframe>)                  # summarizing column values in the <dataframe>
```

```
the.beatles.songs <-
  read.csv("The Beatles songs dataset, v3.csv", stringsAsFactors = FALSE)
str(the.beatles.songs)
```

```
## 'data.frame':    310 obs. of  15 variables:
##  $ Title               : chr  "12-Bar Original" "A Day in the Life" "A Hard Day's Night" "A Shot of Rhythm
and Blues" ...
##  $ Year                : chr  "1965" "1967" "1964" "1963" ...
##  $ Duration            : int  174 335 152 104 163 230 139 NA 124 124 ...
##  $ Other.releases      : int  NA 12 35 NA 29 19 14 9 9 32 ...
##  $ Single.A.side       : chr  "" "Sgt. Pepper's Lonely Hearts Club Band / With a Little Help from My Friend
s" "A Hard Day's Night; A Hard Day's Night" "" ...
##  $ Single.B.side       : chr  "" "A Day in the Life" "Things We Said Today; I Should Have Known Better" ""
...
##  $ Single.certification: chr  "" "" "RIAA Gold" "" ...
##  $ Genre               : chr  "Blues" "Psychedelic Rock, Art Rock, Pop/Rock" "Rock, Electronic, Pop/Rock"
"R&B, Pop/Rock" ...
##  $ Songwriter          : chr  "Lennon, McCartney, Harrison and Starkey" "Lennon and McCartney" "Lennon" "Th
ompson" ...
##  $ Lead.vocal          : chr  "" "Lennon and McCartney" "Lennon, with McCartney" "Lennon" ...
##  $ Cover               : chr  "" "" "" "Y" ...
##  $ Covered.by          : int  NA 27 35 NA NA 32 NA NA 2 20 ...
##  $ Top.50.Billboard    : int  NA NA 8 NA NA NA 50 41 NA NA ...
##  $ Top.50.Rolling.Stone: int  NA 1 11 NA NA NA NA NA NA 44 ...
##  $ Top.50.NME          : int  NA 2 19 NA NA 7 NA NA NA 35 ...
```

```
summary(the.beatles.songs)
```

```
##     Title              Year              Duration      Other.releases
##  Length:310         Length:310         Min.   : 23.0   Min.   : 2.00
##  Class :character   Class :character   1st Qu.:130.0   1st Qu.: 8.00
##  Mode  :character   Mode  :character   Median :149.0   Median :12.50
##                                        Mean   :160.6   Mean   :14.96
##                                        3rd Qu.:176.0   3rd Qu.:19.25
##                                        Max.   :502.0   Max.   :56.00
##                                        NA's   :29      NA's   :94
##  Single.A.side      Single.B.side      Single.certification
##  Length:310         Length:310         Length:310
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##     Genre             Songwriter          Lead.vocal
##  Length:310         Length:310         Length:310
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##     Cover             Covered.by     Top.50.Billboard Top.50.Rolling.Stone
##  Length:310         Min.   : 1.00   Min.   : 1.00    Min.   : 1.00
##  Class :character   1st Qu.: 3.00   1st Qu.:13.00    1st Qu.:13.00
##  Mode  :character   Median : 7.00   Median :25.00    Median :26.00
##                     Mean   :11.50   Mean   :25.31    Mean   :25.55
##                     3rd Qu.:15.75   3rd Qu.:38.00    3rd Qu.:38.00
##                     Max.   :70.00   Max.   :50.00    Max.   :50.00
##                     NA's   :128     NA's   :261      NA's   :261
##    Top.50.NME
##  Min.   : 1.00
##  1st Qu.:13.00
##  Median :26.00
##  Mean   :25.69
##  3rd Qu.:38.00
##  Max.   :50.00
##  NA's   :261
```

Decision trees

Reading the dataset

A modified version of the dataset has been saved earlier using:

```
saveRDS(object = <dataframe or another R object>, file = "<filename>")  # save R object for the next session
```

Restoring the dataset from the corresponding RData file:

```
<dataframe or another R object> <- readRDS(file = "<filename>")         # restore R object in the next session
```

The Beatles songs dataset has been saved earlier using:

```
saveRDS(object = the.beatles.songs, file = "The Beatles songs dataset, v3.2.RData")
```

```
the.beatles.songs <- readRDS("The Beatles songs dataset, v3.2.RData")
summary(the.beatles.songs)
```

```
##     Title                 Year       Duration     Other.releases
##  Length:310         1963   :66   Min.   : 23.0   Min.   : 0.00
##  Class :character   1968   :45   1st Qu.:133.0   1st Qu.: 0.00
##  Mode  :character   1969   :43   Median :150.0   Median : 9.00
##                     1964   :41   Mean   :159.6   Mean   :10.42
##                     1965   :37   3rd Qu.:172.8   3rd Qu.:16.00
##                     1967   :27   Max.   :502.0   Max.   :56.00
##                     (Other):51
##            Single.certification Cover     Covered.by     Top.50.Billboard
##  No                   :259      N:239   Min.   : 0.000   Min.   : 0.000
##  RIAA 2xPlatinum      :  6      Y: 71   1st Qu.: 0.000   1st Qu.: 0.000
##  RIAA 4xPlatinum      :  2              Median : 2.000   Median : 0.000
##  RIAA Gold            : 33              Mean   : 6.752   Mean   : 4.061
##  RIAA Gold, BPI Silver:  2              3rd Qu.: 8.000   3rd Qu.: 0.000
##  RIAA Platinum        :  8              Max.   :70.000   Max.   :50.000
##
##  Top.50.Rolling.Stone   Top.50.NME
##  Min.   : 0.000       Min.   : 0
##  1st Qu.: 0.000       1st Qu.: 0
##  Median : 0.000       Median : 0
##  Mean   : 4.023       Mean   : 4
##  3rd Qu.: 0.000       3rd Qu.: 0
##  Max.   :50.000       Max.   :50
##
```

Binary classification

Decide on the output variable and make appropriate adjustments.

Examine the Top.50.Billboard variable more closely:

```
the.beatles.songs$Top.50.Billboard
```

```
##   [1]  0  0 43  0  0  0  1 10  0  0  0  0 36 14  0  0  0  0  0  0  0  3  0
##  [24]  0  0  0  0  0  0  0  0  0  0  0 41  0  0  0  0  0  0  0  0  0  0 45
##  [47]  0  0  0 22  0  0  0  0  0 25  0  0  0  0  0  0 30 13  0  0  0  0  0
##  [70]  0  0  0  0 12  0  0 47  0  0  0  0  0  0  0  0  0 28  0  0  0  0 44
##  [93] 37  0  0  0  0  0 50  0  0  0  0  0  0  0  2 40  0  0  0  0  0  0  0
## [116] 15  0  0 49  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0
## [139]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 29  0  0 46  0
## [162]  0  0  0  0  0 35  0  0  0  0  0  0  0  0  0  0 11  0  0  0  0  0  0
## [185]  0  0  0  6  0  0  0  0  0 24  0  0  0  0  0  0  0  0 17 32 27  0  0
## [208] 33  0  9  4  0  0 19  0  0  0  0  0  0  0  0  0  0  0  0  0 48  0 20
## [231]  0  0  0  7  0  0  0 21  0  0 18  0  0  0  0  0  0  0  0  0  5  0  0
## [254]  0 23  0  0  0  0  0 31  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0
## [277] 38  0  0  0 42  0  0  0  0  0  0  0  0  0  0  0  0  0 26  0  0 39  0
## [300]  0  0  0  0  0  0  0  0  0  0  0
```

Output variable: a song is among Billboard's top 50 Beatles songs or it isn't (Yes/No, TRUE/FALSE).

Create a new varible to represent the output variable:

```
the.beatles.songs$Top.50.BB <- "No"
the.beatles.songs$Top.50.BB
```

```
##   [1] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [15] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [29] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [43] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [57] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [71] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [85] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
##  [99] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [113] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [127] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [141] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [155] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [169] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [183] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [197] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [211] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [225] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [239] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [253] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [267] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [281] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [295] "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No" "No"
## [309] "No" "No"
```

```
the.beatles.songs$Top.50.BB[the.beatles.songs$Top.50.Billboard > 0] <- "Yes"
the.beatles.songs$Top.50.BB
```

```
##   [1] "No"  "No"  "Yes" "No"  "No"  "No"  "Yes" "Yes" "No"  "No"  "No" 
##  [12] "No"  "Yes" "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes"
##  [23] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
##  [34] "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
##  [45] "No"  "Yes" "No"  "No"  "No"  "Yes" "No"  "No"  "No"  "No"  "No" 
##  [56] "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "Yes" "No"  "No" 
##  [67] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "Yes"
##  [78] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No" 
##  [89] "No"  "No"  "No"  "Yes" "Yes" "No"  "No"  "No"  "No"  "No"  "Yes"
## [100] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "Yes" "No"  "No" 
## [111] "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "Yes" "No"  "No" 
## [122] "No"  "No"  "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No" 
## [133] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
## [144] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
## [155] "No"  "No"  "Yes" "No"  "No"  "Yes" "No"  "No"  "No"  "No"  "No" 
## [166] "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
## [177] "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
## [188] "Yes" "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "No"  "No" 
## [199] "No"  "No"  "No"  "No"  "Yes" "Yes" "Yes" "No"  "No"  "Yes" "No" 
## [210] "Yes" "Yes" "No"  "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "No" 
## [221] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "Yes" "No" 
## [232] "No"  "No"  "Yes" "No"  "No"  "No"  "Yes" "No"  "No"  "Yes" "No" 
## [243] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No" 
## [254] "No"  "Yes" "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "No" 
## [265] "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No"  "No"  "No" 
## [276] "No"  "Yes" "No"  "No"  "No"  "Yes" "No"  "No"  "No"  "No"  "No" 
## [287] "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "Yes" "No"  "No" 
## [298] "Yes" "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No"  "No" 
## [309] "No"  "No"
```

Turn the new variable into a factor:

```
the.beatles.songs$Top.50.BB <- as.factor(the.beatles.songs$Top.50.BB)
head(the.beatles.songs$Top.50.BB)
```

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

Save this version of the dataset as well, for future reuse:

```
saveRDS(object = <dataframe or another R object>, file = "<filename>")  # save R object for the next session
```

```
saveRDS(object = the.beatles.songs, file = "The Beatles songs dataset, v3.3.RData")
```

Examine the distribution of the two factor values more precisely:

```
table(<dataset>$<output variable>)
prop.table(table(<dataset>$<output variable>))
round(prop.table(table(<dataset>$<output variable>)), digits = 2)
```

```
table(the.beatles.songs$Top.50.BB)
```

```
##
##  No Yes
## 261  49
```

```
prop.table(table(the.beatles.songs$Top.50.BB))
```

```
##
##        No       Yes
## 0.8419355 0.1580645
```

```
round(prop.table(table(the.beatles.songs$Top.50.BB)), digits = 2)
```

```
##
##   No  Yes
## 0.84 0.16
```

Train and test datasets

Split the dataset into train and test sets:

```
# install.packages("caret")
library(caret)
set.seed(<n>)
<train dataset indices> <-                         # stratified partitioning:
+ createDataPartition(<dataset>$<output variable>,  # the same distribution of the output variable in both sets
+                     p = .80,                      # 80/20% of data in train/test sets
+                     list = FALSE)                 # don't make a list of results, make a matrix
<train dataset> <- <dataset>[<train dataset indices>, ]
<test dataset>  <- <dataset>[-<train dataset indices>, ]
```

```
library(caret)
set.seed(444)
# set.seed(333) - results in a different split, and different tree and evaluation metrics
train.data.indices <- createDataPartition(the.beatles.songs$Top.50.BB, p = 0.80, list = FALSE)
train.data <- the.beatles.songs[train.data.indices, ]
test.data <- the.beatles.songs[-train.data.indices, ]
```

Model

Build the model / decision tree:

```
install.packages("rpart")
library(rpart)
<decision tree> <- rpart(<output variable> ~                                     # build the tree
+                        <predictor variable 1> + <predictor variable 2> + ...,  # . to include all  variables
+                        data = <train dataset>,
+                        method = "class")                                        # build classification tree
print(<decision tree>)                                                            # default textual representation
```

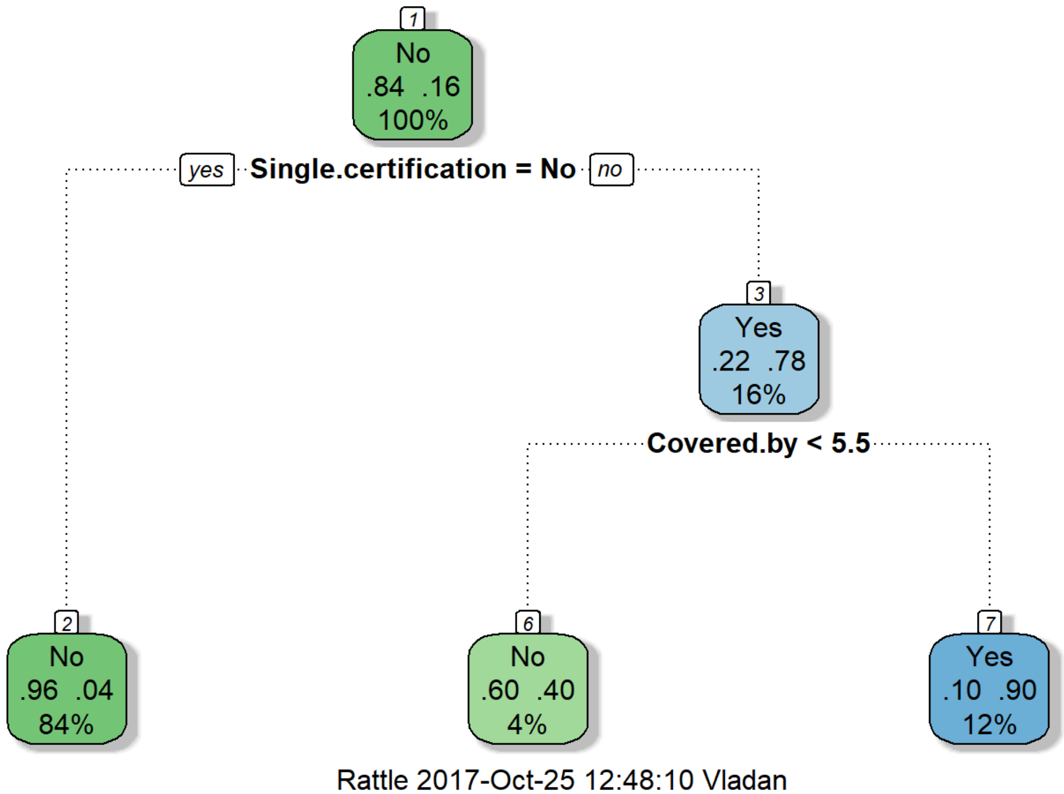
```
library(rpart)
top.50.tree.1 <- rpart(Top.50.BB ~ Single.certification + Covered.by + Year,
                       data = train.data,
                       method = "class")
print(top.50.tree.1)
```

```
## n= 249
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 249 40 No (0.83935743 0.16064257)
##   2) Single.certification=No 208  8 No (0.96153846 0.03846154) *
##   3) Single.certification=RIAA 2xPlatinum,RIAA 4xPlatinum,RIAA Gold,RIAA Gold, BPI Silver,RIAA Platinum 41
9 Yes (0.21951220 0.78048780)
##     6) Covered.by< 5.5 10  4 No (0.60000000 0.40000000) *
##     7) Covered.by>=5.5 31  3 Yes (0.09677419 0.90322581) *
```

Depict the model:

```
# install.packages("rattle")
# install.packages("rpart.plot")
# install.packages("RColorBrewer")
library(rattle)
library(rpart.plot)
library(RColorBrewer)
fancyRpartPlot(<decision tree>)
```

```
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
fancyRpartPlot(top.50.tree.1)
```



Predictions

Make predictions using the model built in the previous step:

```
<predictions> <- predict(object = <decision tree>,
+                        newdata = <test dataset>,
+                        type = "class")
<predictions>[<i1>:<ik>]                           # examine some of the predictions
<predictions dataframe> <-
+ data.frame(<observation ID> = <test dataset>$<observation ID column>,
+            <another relevant feature> = <test dataset>$<another relevant feature column>,
+            ...,
+            <output feature> = <test dataset>$<output variable>,
+            <predictions feature> = <predictions>)
```

```
top.50.predictions.1 <- predict(top.50.tree.1, newdata = test.data, type = "class")
top.50.predictions.1[1:20]
```

```
##   1   2   3   5  11  15  19  22  25  27  42  60  62  71  72  78  85  99
##  No  No Yes  No  No  No  No  No  No  No  No Yes  No  No  No  No  No Yes
## 102 107
##  No  No
## Levels: No Yes
```

```
top.50.predictions.1.dataframe <- data.frame(Song = test.data$Title,
                                             Top.50.Billboard = test.data$Top.50.Billboard,
                                             Top.50.BB = test.data$Top.50.BB,
                                             Prediction = top.50.predictions.1)
```

Evaluation

How good are the predictions?

Compute confusion matrix:

```
<cm> <- table(True = <predictions dataframe>$<output variable>,
+             Predicted = <predictions dataframe>$<predictions>)
```

Compute evaluation metrics:

- accuracy = (TP + TN) / N
- precision = TP / (TP + FP)
- recall = TP / (TP + FN)
- F1 = (2 * precision * recall) / (precision + recall)

Note: precision and recall are inversely proportional to each other.

```
<evaluation metrics vector> <- <user-specified function>(<cm>)
# accuracy = sum(diag(cm)) / sum(cm)
# precision <- TP / (TP + FP)
# recall <- TP / (TP + FN)
# F1 <- (2 * precision * recall) / (precision + recall)
```

```
cm.1 <- table(True = test.data$Top.50.BB, Predicted = top.50.predictions.1)
cm.1
```

```
##      Predicted
## True  No Yes
##   No  51   1
##   Yes  5   4
```

```
# alternatively:
# cm.1 <- table(True = top.50.predictions.1.dataframe$Top.50.BB,
#               Predicted = top.50.predictions.1.dataframe$Prediction)
# cm.1
```

```
source("Evaluation metrics.R")
eval.1 <- getEvaluationMetrics(cm.1)
eval.1
```

```
##  Accuracy Precision    Recall        F1
## 0.9016393 0.8000000 0.4444444 0.5714286
```

Try another tree, using fewer and/or different predictors (e.g., Duration + Covered.by, Duration + Year, etc.). # In practice, strong predictors from the previous tree are kept in the new tree as well.

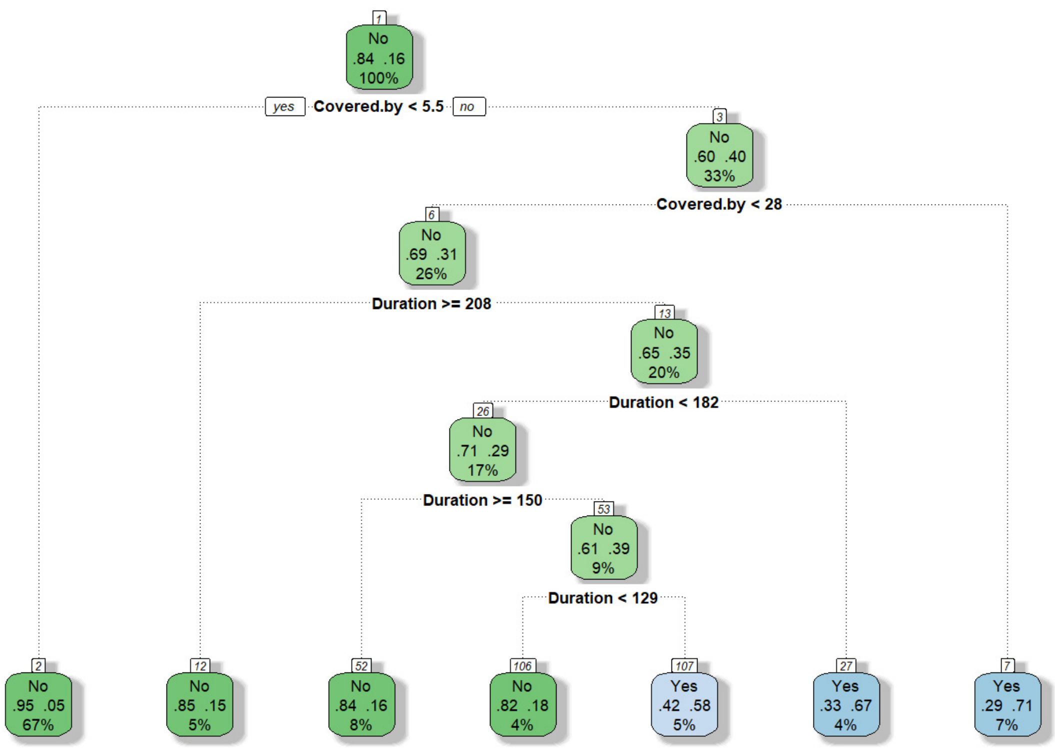
Also try changing the seed in splitting the dataset into # train and test sets.

```
top.50.tree.2 <- rpart(Top.50.BB ~ Duration + Covered.by,
                       data = train.data,
                       method = "class")
# top.50.tree.2 <- rpart(Top.50.BB ~ .,       # use almost all variables, excluding some specific ones
#                         data = subset(train.data, select = -c(Title, Top.50.Billboard)),
#                         method = "class")
```

```
print(top.50.tree.2)
```

```
## n= 249
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 249 40 No (0.83935743 0.16064257)
##     2) Covered.by< 5.5 168  8 No (0.95238095 0.04761905) *
##     3) Covered.by>=5.5 81 32 No (0.60493827 0.39506173)
##       6) Covered.by< 28 64 20 No (0.68750000 0.31250000)
##        12) Duration>=207.5 13  2 No (0.84615385 0.15384615) *
##        13) Duration< 207.5 51 18 No (0.64705882 0.35294118)
##          26) Duration< 182.5 42 12 No (0.71428571 0.28571429)
##            52) Duration>=150 19  3 No (0.84210526 0.15789474) *
##            53) Duration< 150 23  9 No (0.60869565 0.39130435)
##             106) Duration< 129 11  2 No (0.81818182 0.18181818) *
##             107) Duration>=129 12  5 Yes (0.41666667 0.58333333) *
##          27) Duration>=182.5 9  3 Yes (0.33333333 0.66666667) *
##       7) Covered.by>=28 17  5 Yes (0.29411765 0.70588235) *
```

```
fancyRpartPlot(top.50.tree.2)
```



Rattle 2017-Oct-25 12:48:12 Vladan

```
top.50.predictions.2 <- predict(top.50.tree.2, newdata = test.data, type = "class")
top.50.predictions.2[1:20]
```

```
##   1   2   3   5  11  15  19  22  25  27  42  60  62  71  72  78  85  99
##  No  No Yes  No  No  No  No  No  No  No  No  No Yes  No  No  No Yes Yes
## 102 107
##  No  No
## Levels: No Yes
```

```
top.50.predictions.2.dataframe <- data.frame(Song = test.data$Title,
                                             Top.50.Billboard = test.data$Top.50.Billboard,
                                             Top.50.BB = test.data$Top.50.BB,
                                             Prediction = top.50.predictions.2)
cm.2 <- table(True = test.data$Top.50.BB, Predicted = top.50.predictions.2)
cm.2
```

```
##      Predicted
## True  No Yes
##   No  42  10
##   Yes  6   3
```

```
eval.2 <- getEvaluationMetrics(cm.2)
eval.2
```

```
##  Accuracy Precision    Recall        F1
## 0.7377049 0.2307692 0.3333333 0.2727273
```

Optimization

Controlling rpart parameters:

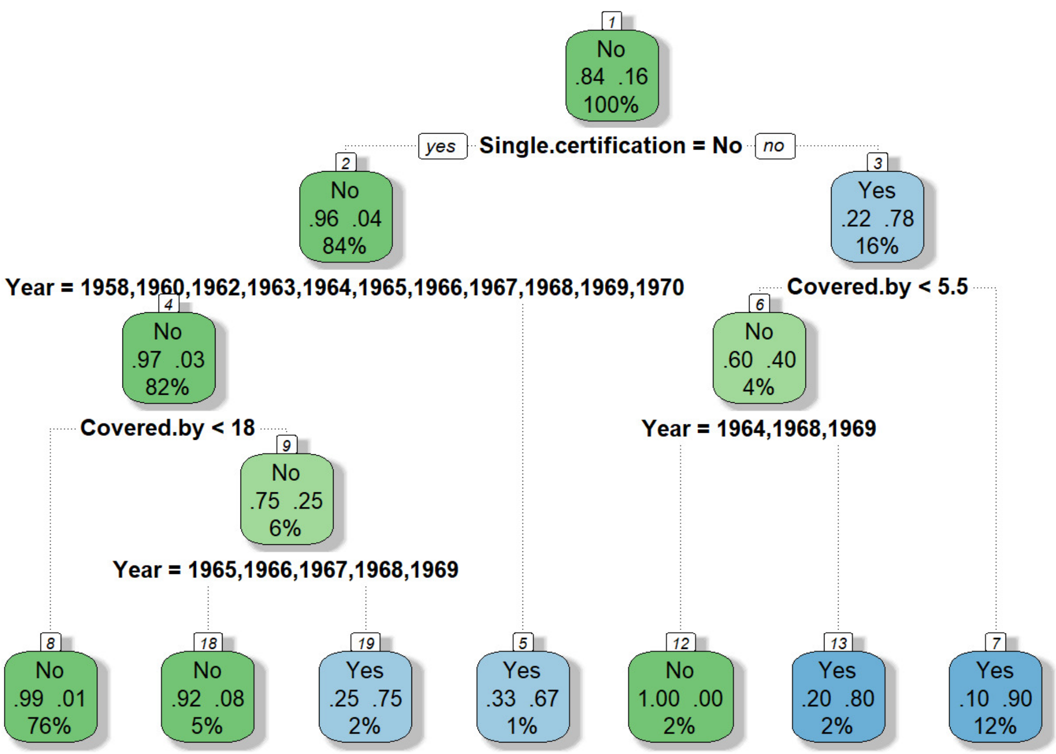
- cp (complexity parameter) - don't split at a node if the split does not improve the model by at least cp (default: 0.01)
- minsplit - don't attempt a split at a node if the number of observations is not higher than minsplit (default: 20)

```
# install.packages("rpart")
library(rpart)
<decision tree> <- rpart(<output variable> ~                                   # build the tree
+                        <predictor variable 1> + <predictor variable 2> + ...,  # . to include all variables
+                        data = <train dataset>,
+                        method = "class",                                     # build classification tree
+                        control = rpart.control(minsplit = <n>, cp = <q>))      # decrease both for larger tree
print(<decision tree>)                                                         # default textual  representation
```

```
top.50.tree.3 <- rpart(Top.50.BB ~ Single.certification + Covered.by + Year,
                       data = train.data,
                       method = "class",
                       control = rpart.control(minsplit = 10, cp = 0.001))
print(top.50.tree.3)
```

```
## n= 249
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 249 40 No (0.83935743 0.16064257)
##    2) Single.certification=No 208  8 No (0.96153846 0.03846154)
##      4) Year=1958,1960,1962,1963,1964,1965,1966,1967,1968,1969,1970 205  6 No (0.97073171 0.02926829)
##        8) Covered.by< 17.5 189  2 No (0.98941799 0.01058201) *
##        9) Covered.by>=17.5 16  4 No (0.75000000 0.25000000)
##         18) Year=1965,1966,1967,1968,1969 12  1 No (0.91666667 0.08333333) *
##         19) Year=1963,1964 4  1 Yes (0.25000000 0.75000000) *
##      5) Year=1961 3  1 Yes (0.33333333 0.66666667) *
##    3) Single.certification=RIAA 2xPlatinum,RIAA 4xPlatinum,RIAA Gold,RIAA Gold, BPI Silver,RIAA Platinum 41
9 Yes (0.21951220 0.78048780)
##      6) Covered.by< 5.5 10  4 No (0.60000000 0.40000000)
##       12) Year=1964,1968,1969 5  0 No (1.00000000 0.00000000) *
##       13) Year=1962,1963,1965,1980 5  1 Yes (0.20000000 0.80000000) *
##      7) Covered.by>=5.5 31  3 Yes (0.09677419 0.90322581) *
```

```
fancyRpartPlot(top.50.tree.3)
```



Rattle 2017-Oct-25 12:48:14 Vladan

```
top.50.predictions.3 <- predict(top.50.tree.3, newdata = test.data, type = "class")
top.50.predictions.3[1:20]
```

```
##   1   2   3   5  11  15  19  22  25  27  42  60  62  71  72  78  85  99
##  No  No Yes  No  No  No  No  No  No  No Yes Yes  No  No  No  No  No Yes
## 102 107
##  No  No
## Levels: No Yes
```

```
top.50.predictions.3.dataframe <- data.frame(Song = test.data$Title,
                                             Top.50.Billboard = test.data$Top.50.Billboard,
                                             Top.50.BB = test.data$Top.50.BB,
                                             Prediction = top.50.predictions.3)
cm.3 <- table(True = test.data$Top.50.BB, Predicted = top.50.predictions.3)
cm.3
```

```
##      Predicted
## True  No Yes
##   No  50   2
##   Yes  5   4
```

```
eval.3 <- getEvaluationMetrics(cm.3)
eval.3
```

```
##  Accuracy Precision    Recall        F1
## 0.8852459 0.6666667 0.4444444 0.5333333
```

Compare the results (the corresponding models/trees):

```
data.frame(rbind(<evaluation metrics 1>, <evaluation metrics 2>),
+          row.names = c("<tree 1>", "<tree 2>"))
```

```
data.frame(rbind(eval.1, eval.3),
           row.names = c("top.50.tree.1", "top.50.tree.3"))
```

```
##                Accuracy Precision    Recall        F1
## top.50.tree.1 0.9016393 0.8000000 0.4444444 0.5714286
## top.50.tree.3 0.8852459 0.6666667 0.4444444 0.5333333
```

Model 3 exhibits overfitting. It is a frequent case with large trees.

Cross-validation

Cross-validate the model - find the optimal value for cp (the most important parameter), in order to avoid overfitting the model to the training data:

```
# install.packages("e1071")                                # relevant caret functions need e1071
# install.packages("caret")
library(e1071)
library(caret)
<folds> = trainControl(method = "cv", number = <k>)          # define <k>-fold cross-validation  parameters
<cpGrid> = expand.grid(.cp =                                 # specify the range of the cp values to examine
+                      seq(from = <start value>, to = <end value>, by = <step>))
set.seed(<seed>)
train(<output variable> ~                                    # find the optimal value for cp
+     <predictor variable 1> + <predictor variable 2> + ...,  # . to include all variables
+     data = <train dataset>,
+     method = "rpart",                                      # use rpart() to build multiple classification trees
+     control = rpart.control(minsplit = 10),
+     trControl = <folds>, tuneGrid = <cpGrid>)              # <folds> and <cpGrid> from above
```

```r
library(e1071)
library(caret)
folds = trainControl(method = "cv", number = 10)               # 10-fold cross-validation
cpGrid = expand.grid(.cp = seq(from = 0.001, to = 0.05, by = 0.001))
set.seed(11)
train(Top.50.BB ~ Single.certification + Covered.by + Year,
      data = train.data,
      method = "rpart",
      control = rpart.control(minsplit = 10),
      trControl = folds, tuneGrid = cpGrid)
```

```
## CART
##
## 249 samples
##   3 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 224, 224, 225, 224, 224, 224, ...
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.001  0.9113333  0.6777534
##   0.002  0.9113333  0.6777534
##   0.003  0.9113333  0.6777534
##   0.004  0.9113333  0.6777534
##   0.005  0.9113333  0.6777534
##   0.006  0.9113333  0.6777534
##   0.007  0.9113333  0.6777534
##   0.008  0.9113333  0.6777534
##   0.009  0.9113333  0.6777534
##   0.010  0.9113333  0.6777534
##   0.011  0.9113333  0.6777534
##   0.012  0.9113333  0.6777534
##   0.013  0.9113333  0.6777534
##   0.014  0.9028333  0.6741848
##   0.015  0.9028333  0.6741848
##   0.016  0.9028333  0.6741848
##   0.017  0.9028333  0.6741848
##   0.018  0.9028333  0.6741848
##   0.019  0.9028333  0.6741848
##   0.020  0.9028333  0.6741848
##   0.021  0.9028333  0.6741848
##   0.022  0.9028333  0.6741848
##   0.023  0.9028333  0.6741848
##   0.024  0.9028333  0.6741848
##   0.025  0.9028333  0.6741848
##   0.026  0.9028333  0.6741848
##   0.027  0.9028333  0.6741848
##   0.028  0.8828333  0.5697261
##   0.029  0.8828333  0.5697261
##   0.030  0.8828333  0.5697261
##   0.031  0.8828333  0.5697261
##   0.032  0.8828333  0.5697261
##   0.033  0.8828333  0.5697261
##   0.034  0.8828333  0.5697261
##   0.035  0.8828333  0.5697261
##   0.036  0.8828333  0.5697261
##   0.037  0.8828333  0.5697261
##   0.038  0.8828333  0.5697261
##   0.039  0.8828333  0.5697261
##   0.040  0.8828333  0.5697261
##   0.041  0.8828333  0.5697261
##   0.042  0.8748333  0.5248611
##   0.043  0.8748333  0.5248611
##   0.044  0.8748333  0.5248611
##   0.045  0.8748333  0.5248611
##   0.046  0.8748333  0.5248611
##   0.047  0.8748333  0.5248611
##   0.048  0.8748333  0.5248611
##   0.049  0.8748333  0.5248611
```

```
##   0.050  0.8748333  0.5248611
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.013.
```

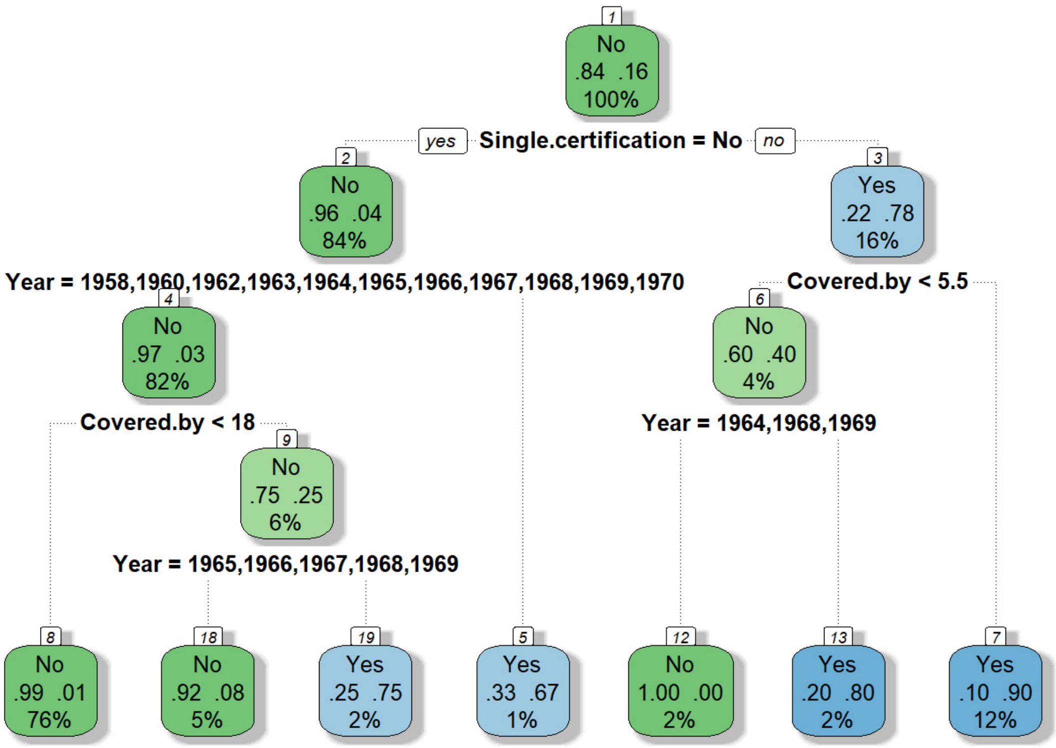
Prunning a complex tree using the optimal cp value:

```
<prunned decision tree> <- prune(<decision tree>, cp = <optimal cp value>)
print(<prunned decision tree>)
fancyRpartPlot(<prunned decision tree>)
```

```
top.50.tree.5 <- prune(top.50.tree.3, cp = 0.013)              # cp value found in the previous step (train
())
print(top.50.tree.5)
```

```
## n= 249
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 249 40 No (0.83935743 0.16064257)
##    2) Single.certification=No 208  8 No (0.96153846 0.03846154)
##      4) Year=1958,1960,1962,1963,1964,1965,1966,1967,1968,1969,1970 205  6 No (0.97073171 0.02926829)
##        8) Covered.by< 17.5 189  2 No (0.98941799 0.01058201) *
##        9) Covered.by>=17.5 16  4 No (0.75000000 0.25000000)
##         18) Year=1965,1966,1967,1968,1969 12  1 No (0.91666667 0.08333333) *
##         19) Year=1963,1964 4  1 Yes (0.25000000 0.75000000) *
##      5) Year=1961 3  1 Yes (0.33333333 0.66666667) *
##    3) Single.certification=RIAA 2xPlatinum,RIAA 4xPlatinum,RIAA Gold,RIAA Gold, BPI Silver,RIAA Platinum 41
9 Yes (0.21951220 0.78048780)
##      6) Covered.by< 5.5 10  4 No (0.60000000 0.40000000)
##       12) Year=1964,1968,1969 5  0 No (1.00000000 0.00000000) *
##       13) Year=1962,1963,1965,1980 5  1 Yes (0.20000000 0.80000000) *
##      7) Covered.by>=5.5 31  3 Yes (0.09677419 0.90322581) *
```

```
fancyRpartPlot(top.50.tree.5)
```



Rattle 2017-Oct-25 12:48:18 Vladan

Make predictions with the prunned tree:

```
top.50.predictions.5 <- predict(top.50.tree.5, newdata = test.data, type = "class")
top.50.predictions.5[1:20]
```

```
##   1   2   3   5  11  15  19  22  25  27  42  60  62  71  72  78  85  99
##  No  No Yes  No  No  No  No  No  No  No Yes Yes  No  No  No  No  No Yes
## 102 107
##  No  No
## Levels: No Yes
```

```
top.50.predictions.5.dataframe <- data.frame(Song = test.data$Title,
                                             Top.50.Billboard = test.data$Top.50.Billboard,
                                             Top.50.BB = test.data$Top.50.BB,
                                             Prediction = top.50.predictions.5)
cm.5 <- table(True = test.data$Top.50.BB, Predicted = top.50.predictions.5)
cm.5
```

```
##      Predicted
## True  No Yes
##   No  50   2
##   Yes  5   4
```

```
eval.5 <- getEvaluationMetrics(cm.5)
eval.5
```

```
##  Accuracy Precision    Recall        F1
## 0.8852459 0.6666667 0.4444444 0.5333333
```

Compare all relevant models: `data.frame(rbind(<evaluation metrics 1>, <evaluation metrics 2>, ...),`
`+ row.names = c("<tree 1>", "<tree 2>", ...))`

```
data.frame(rbind(eval.1, eval.3, eval.5),
           row.names = c("top.50.tree.1", "top.50.tree.3", "top.50.tree.5"))
```

```
##                Accuracy Precision    Recall        F1
## top.50.tree.1 0.9016393 0.8000000 0.4444444 0.5714286
## top.50.tree.3 0.8852459 0.6666667 0.4444444 0.5333333
## top.50.tree.5 0.8852459 0.6666667 0.4444444 0.5333333
```

KNN (K-nearest neighbors)

Reading the dataset

Restoring the dataset from the corresponding RData file:
`<dataframe or another R object> <- readRDS(file = "<filename>")        # restore R object in the next session`
The Beatles songs dataset has been saved earlier using:
`saveRDS(object = the.beatles.songs, file = "The Beatles songs dataset, v3.4.RData")`

```
the.beatles.songs <- readRDS("The Beatles songs dataset, v3.4.RData")
```

Adapting the dataset

Eliminate Top.50.Billboard from the dataset. Top.50.BB has been created from Top.50.Billboard, hence Top.50.Billboard predicts Top.50.BB 100%. Still, store Top.50.Billboard in a separate vector for possible later use.

```
top.50.billboard <- the.beatles.songs$Top.50.Billboard
the.beatles.songs$Top.50.Billboard <- NULL
```

Save this version of the dataset as well, for future reuse:

```
saveRDS(object = the.beatles.songs, file = "The Beatles songs dataset, v3.5.RData")
```

Rescaling

Is rescaling of numeric variables needed? Are their ranges different?
`summary(<dataframe>)                    # examine the ranges of numeric variables`

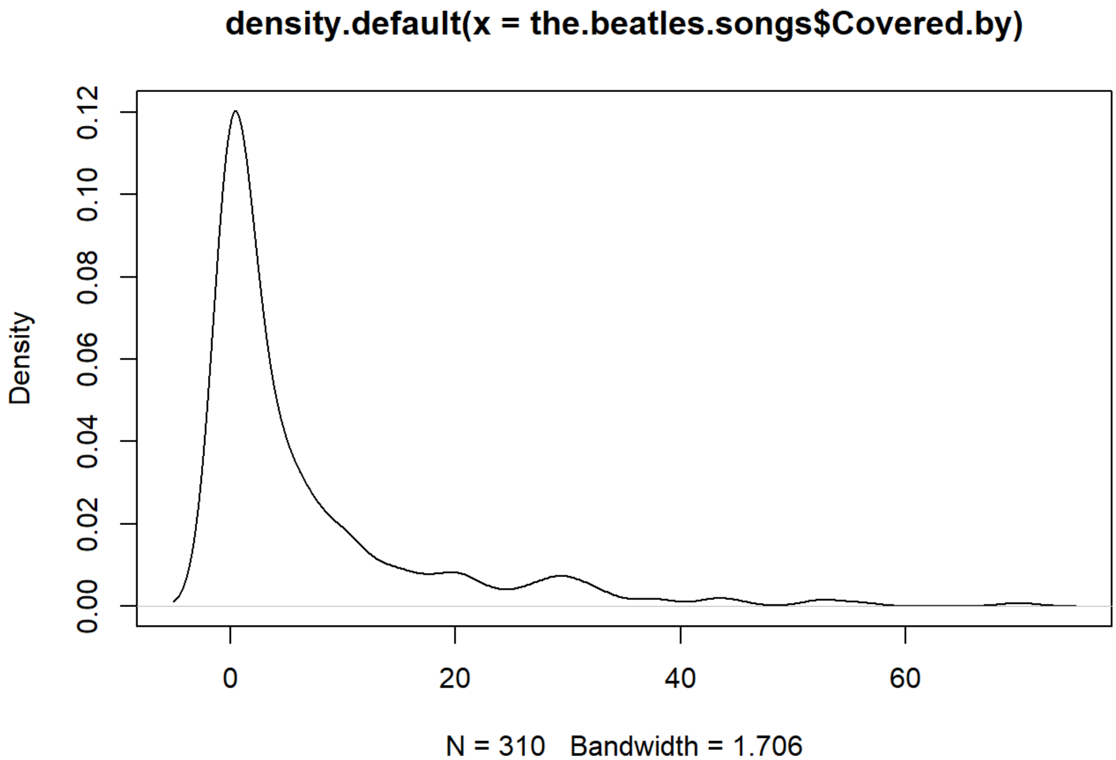
```
summary(the.beatles.songs)
```

```
##     Title               Year       Duration     Other.releases 
##  Length:310         1963   :66   Min.   : 23.0   Min.   : 0.00  
##  Class :character   1968   :45   1st Qu.:133.0   1st Qu.: 0.00  
##  Mode  :character   1969   :43   Median :150.0   Median : 9.00  
##                     1964   :41   Mean   :159.6   Mean   :10.42  
##                     1965   :37   3rd Qu.:172.8   3rd Qu.:16.00  
##                     1967   :27   Max.   :502.0   Max.   :56.00  
##                     (Other):51                                  
##            Single.certification Cover     Covered.by    
##  No                  :259       N:239   Min.   : 0.000  
##  RIAA 2xPlatinum      :  6      Y: 71   1st Qu.: 0.000  
##  RIAA 4xPlatinum      :  2              Median : 2.000  
##  RIAA Gold            : 33              Mean   : 6.752  
##  RIAA Gold, BPI Silver:  2              3rd Qu.: 8.000  
##  RIAA Platinum        :  8              Max.   :70.000  
##                                                         
##  Top.50.Rolling.Stone   Top.50.NME Top.50.BB 
##  Min.   : 0.000       Min.   : 0   No :261   
##  1st Qu.: 0.000       1st Qu.: 0   Yes: 49   
##  Median : 0.000       Median : 0             
##  Mean   : 4.023       Mean   : 4             
##  3rd Qu.: 0.000       3rd Qu.: 0             
##  Max.   :50.000       Max.   :50             
## 
```

Check if numeric variables follow normal distribution:

```
summary(<numeric variable>)           # the mean and the median values similar: probably normal distribution
plot(density((<numeric variable>))    # visual inspection
hist(<numeric variable>)              # visual inspection
qqnorm(<numeric variable>)            # values lie more or less along the diagonal (straight line)
shapiro.test(<numeric variable>)      # good for small sample sizes, e.g. n < ~2000; H0: normal distribution
```

```
plot(density(the.beatles.songs$Covered.by))
```




```
# ...                                     # check the distributions of other variables as well
```

Since the distribution of numeric variables shows that rescaling is needed:

```
source("Rescale numeric variables.R")
the.beatles.songs.rescaled <- rescaleNumericVariables(the.beatles.songs)
```

Train and test datasets

Split the dataset into train and test sets:

```
# install.packages("caret")
library(caret)
set.seed(<n>)
<train dataset indices> <-                            # stratified partitioning:
+ createDataPartition(<dataset>$<output variable>,  # the same distribution of the output variable in both sets
+                     p = .80,                        # 80/20% of data in train/test sets
+                     list = FALSE)                   # don't make a list of results, make a matrix
<train dataset> <- <dataset>[<train dataset indices>, ]
<test dataset>  <- <dataset>[-<train dataset indices>, ]
```

```
library(caret)
set.seed(444)
# set.seed(333) - results in a different split, and different results and eval. metrics
train.data.indices <- createDataPartition(the.beatles.songs.rescaled$Top.50.BB, p = 0.80, list = FALSE)
train.data <- the.beatles.songs.rescaled[train.data.indices, ]
test.data <- the.beatles.songs.rescaled[-train.data.indices, ]
```

Model

Build the model:

```
library(class)
<knn model> <- knn(train = <training dataset>,        # training data without the output (class) variable
+                  test = <test dataset>,             # test data without the output (class) variable
+                  cl = <class values for training>,  # output (class) variable is specified here
+                  k = <n>)                           # <n>: random guess, or obtained from  cross-validation
head(<knn model>)
```

```
library(class)
top.50.knn.1 <- knn(train = train.data[, -c(1, 10)],    # eliminate Title (non-numeric) and output/class (Top.
50.BB)
                    test = test.data[, -c(1, 10)],      # eliminate Title (non-numeric) and output/class (Top.
50.BB)
                    cl = train.data$Top.50.BB,          # output (class) variable
                    k = 5)                              # k = 5: random value to start with
head(top.50.knn.1)                                      # these are already the predictions, i.e. no predict()
etc.
```

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

```
which(test.data$Top.50.BB != top.50.knn.1)
```

```
## [1] 11 29 41 48
```

Evaluation

Compute confusion matrix:

```
<cm> <- table(True = <test dataset>$<output variable>,
+             Predicted = <test dataset>$<predictions>)
```

```
knn.cm.1 <- table(True = test.data$Top.50.BB, Predicted = top.50.knn.1)
knn.cm.1
```

```
##      Predicted
## True  No Yes
##   No  51   1
##   Yes  3   6
```

Compute evaluation metrics:

- accuracy = (TP + TN) / N
- precision = TP / (TP + FP)
- recall = TP / (TP + FN)
- F1 = (2 * precision * recall) / (precision + recall)

Note: precision and recall are inversely proportional to each other.

```
<evaluation metrics vector> <- <user-specified function>(<cm>)
# accuracy = sum(diag(cm)) / sum(cm)
# precision <- TP / (TP + FP)
# recall <- TP / (TP + FN)
# F1 <- (2 * precision * recall) / (precision + recall)
```

```
source("Evaluation metrics.R")
eval.knn.1 <- getEvaluationMetrics(knn.cm.1)
eval.knn.1
```

```
##  Accuracy Precision    Recall        F1
## 0.9344262 0.8571429 0.6666667 0.7500000
```

Cross-validation

What if k had a different value?

Cross-validate the model - find the optimal value for k (the most important parameter), in order to avoid overfitting the model to the training data:

```
# install.packages("e1071")                                  # relevant caret functions need e1071
# install.packages("caret")
library(e1071)
library(caret)
<folds> = trainControl(method = "cv", number = <k>)          # define <k>-fold cross-validation parameters
<cpGrid> = expand.grid(.k =                                  # specify the range of the (odd) values to examine
seq(from = <start value>, to = <end value>, by = <step>))
set.seed(<seed>)
<knn cv> <- train(<output variable> ~                        # find the optimal value for k
+    <predictor variable 1> + <predictor variable 2> + ...,  # . to include all variables
+    data = <train dataset>,
+    method = "knn",                                         # use knn() to build multiple classification models
+    trControl = <folds>, tuneGrid = <cpGrid>)               # <folds> and <cpGrid> from above
```

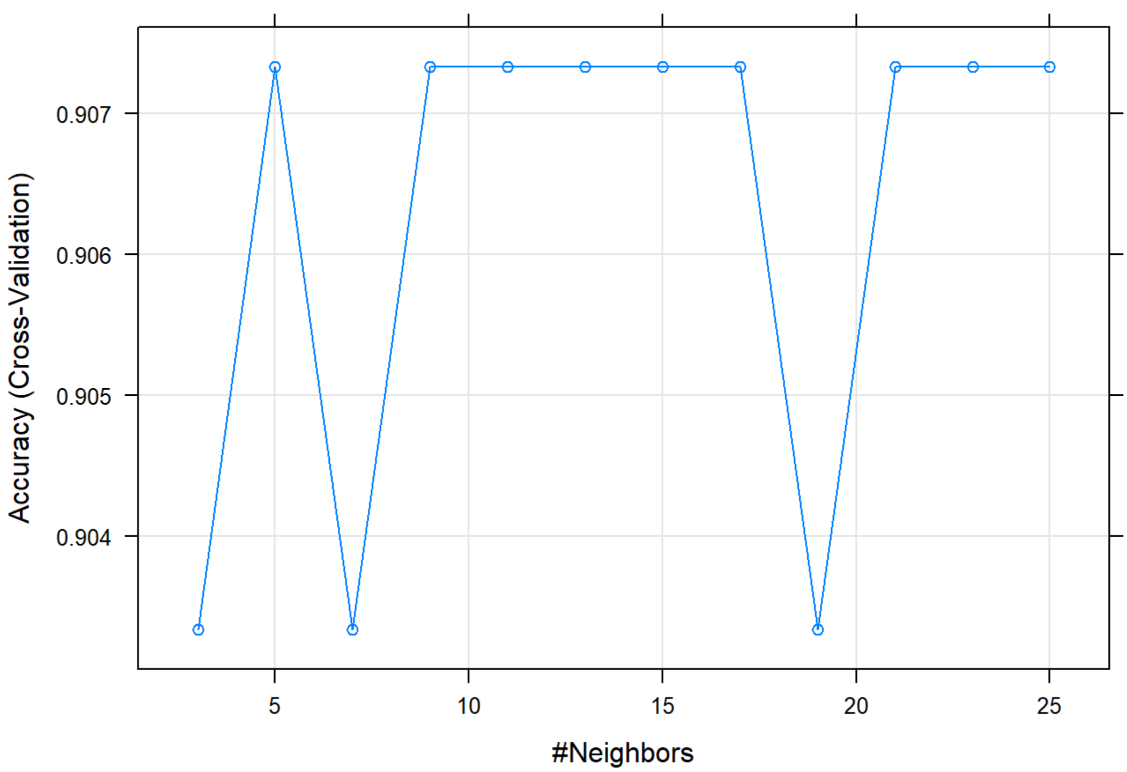
```
library(e1071)
library(caret)
knn.folds = trainControl(method = "cv", number = 10)            # 10-fold cross-validation
knn.cpGrid = expand.grid(.k = seq(from = 3, to = 25, by = 2))
set.seed(11)
knn.cv <- train(Top.50.BB ~ . - Title,
                data = train.data,
                method = "knn",
                trControl = knn.folds, tuneGrid = knn.cpGrid)
knn.cv
```

```
## k-Nearest Neighbors
##
## 249 samples
##   9 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 224, 224, 225, 224, 224, 224, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    3  0.9033333  0.5982314
##    5  0.9073333  0.6323177
##    7  0.9033333  0.6221013
##    9  0.9073333  0.6314675
##   11  0.9073333  0.6323177
##   13  0.9073333  0.6323177
##   15  0.9073333  0.6323177
##   17  0.9073333  0.6323177
##   19  0.9033333  0.6124107
##   21  0.9073333  0.6259243
##   23  0.9073333  0.6259243
##   25  0.9073333  0.6259243
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was k = 25.
```

Plot the cross-validation results (accuracy for different values of k):

```
# plot(<knn model>)   # the model obtained by <knn model> <- train(...)
```

```
plot(knn.cv)
```



Build the model and compute confusion matrix and evaluation metrics for another value of k:

```
top.50.knn.2 <- knn(train = train.data[, -c(1, 10)],    # eliminate Title (non-numeric) and output/class (Top.
50.BB)
                    test = test.data[, -c(1, 10)],      # eliminate Title (non-numeric) and output/class (Top.
50.BB)
                    cl = train.data$Top.50.BB,          # output (class) variable
                    k = 7)                              # k = 7: another random value to test and compare
knn.cm.2 <- table(True = test.data$Top.50.BB, Predicted = top.50.knn.2)
knn.cm.2
```

```
##      Predicted
## True  No Yes
##   No  51   1
##   Yes  2   7
```

```
eval.knn.2 <- getEvaluationMetrics(knn.cm.2)
eval.knn.2
```

```
##  Accuracy Precision    Recall        F1
## 0.9508197 0.8750000 0.7777778 0.8235294
```

Compare the evaluation metrics of the two classifiers:

```
data.frame(rbind(<evaluation metrics 1>, <evaluation metrics 2>, ...),
+          row.names = c("<model 1>", "<model 2>", ...))
```

```
data.frame(rbind(eval.knn.1, eval.knn.2),
           row.names = c("eval.knn.1", "eval.knn.2"))
```

```
##             Accuracy Precision    Recall        F1
## eval.knn.1 0.9344262 0.8571429 0.6666667 0.7500000
## eval.knn.2 0.9508197 0.8750000 0.7777778 0.8235294
```

Naive Bayes

Reading the dataset

Read an appropriate version of the dataset:

```
<dataframe or another R object> <- readRDS(file = "<filename>")        # restore R object from another session
```

The Beatles songs dataset, v3.5.RData (without duplicated song names), has been saved in the session on KNN, assuming that the.beatles.songs$Top.50.BB (a factor variable) is the output variable:

```
saveRDS(object = the.beatles.songs, file = "The Beatles songs dataset, v3.5.RData")
```

The same output variable is assumed in this session.

```
the.beatles.songs <- readRDS("The Beatles songs dataset, v3.5.RData")
str(the.beatles.songs)
```

```
## 'data.frame':	310 obs. of  10 variables:
##  $ Title               : chr  "12-Bar Original" "A Day in the Life" "A Hard Day's Night" "A Shot of Rhythm and Blues" ...
##  $ Year                : Factor w/ 14 levels "1958","1960",..: 7 9 6 5 5 10 7 3 5 5 ...
##  $ Duration            : num  174 335 152 104 163 230 139 150 124 124 ...
##  $ Other.releases      : num  0 12 35 0 29 19 14 9 9 32 ...
##  $ Single.certification: Factor w/ 6 levels "No","RIAA 2xPlatinum",..: 1 1 4 1 1 1 4 1 1 1 ...
##  $ Cover               : Factor w/ 2 levels "N","Y": 1 1 1 2 2 1 2 2 1 1 ...
##  $ Covered.by          : num  0 27 35 0 0 32 0 0 2 20 ...
##  $ Top.50.Rolling.Stone: num  0 50 40 0 0 0 0 0 0 7 ...
##  $ Top.50.NME          : num  0 49 32 0 0 44 0 0 0 16 ...
##  $ Top.50.BB           : Factor w/ 2 levels "No","Yes": 1 1 2 1 1 1 2 2 1 1 ...
```

NB essentials

Naive Bayes classification is typically used with categorical (factor) variables.

Numeric variables (if any) should be either:

- represented as probabilities, if they follow normal distribution, or
- discretized (split either into equal-length or equal-frequency intervals (the latter is used more often))

Check numeric variables in the dataset for normality assumption:

```
apply(<numeric dataframe>,  # <original dataframe>[, c(<num. col. 1>, <num. col. 1>, ...]
+     MARGIN = 2,           # apply FUN by columns
+     FUN = shapiro.test)   # Shapiro-Wilks' test of normality; good for small no. of observations (< 2000)
```

```
apply(the.beatles.songs[, c(3:4, 7:9)], MARGIN = 2, FUN = shapiro.test)
```

```
## $Duration
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.79255, p-value < 2.2e-16
##
##
## $Other.releases
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.88075, p-value = 8.343e-15
##
##
## $Covered.by
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.66532, p-value < 2.2e-16
##
##
## $Top.50.Rolling.Stone
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.41964, p-value < 2.2e-16
##
##
## $Top.50.NME
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.4198, p-value < 2.2e-16
```

No normally distributed numeric variables in the dataset.

Discretize numeric variables using `bnlearn::discretize()` :

```
library(bnlearn)
?discretize()
<new dataframe with discretized variables> <-
+ discretize(<numeric dataframe>,                   # <original dataframe>[, c(<num. col. 1>, <num. col. 1>, ...]
+            method = "quantile" |                  # use equal-frequency intervals (default)
+            method = "interval" |                  # use equal-length intervals
+            method = "hartemink",                  # use Hartemink's algorithm
+            breaks = c(<n1>, <n2>, ..., <ncol>))   # no. of discrete intervals for each column
```

```
library(bnlearn)
discretized.features <- discretize(the.beatles.songs[, c(3:4, 7:9)],
                                   method = "interval",
                                   breaks = c(5, 5, 5, 5, 5))
summary(discretized.features)
```

```
##       Duration        Other.releases      Covered.by
##  [22.5,119]: 38   [-0.056,11.2]:193    [-0.07,14]:261
##  (119,215] :238   (11.2,22.4]  : 78    (14,28]   : 27
##  (215,310] : 28   (22.4,33.6]  : 27    (28,42]   : 15
##  (310,406] :  3   (33.6,44.8]  : 10    (42,56]   :  6
##  (406,502] :  3   (44.8,56.1]  :  2    (56,70.1] :  1
##  Top.50.Rolling.Stone      Top.50.NME
##  [-0.05,10]:271       [-0.05,10]:271
##  (10,20]   : 10       (10,20]   : 10
##  (20,30]   :  9       (20,30]   : 10
##  (30,40]   : 10       (30,40]   :  9
##  (40,50]   : 10       (40,50]   : 10
```

Re-compose the dataframe with the discretized features:

```
<dataframe> <- cbind(<dataframe 1>[, c(<col11>, <col12>, ...)], <dataframe 2>[, c(<col21>, <col22>, ...)])
<dataframe> <- <dataframe>[, c(<col i>, <col k>, ...)]            # rearrange columns (optional)
```

Alternatively:

```
<dataframe> <- <dataframe>[, names(<original dataframe>)]         # rearrange columns (optional)
```

```
the.beatles.songs.nb <- cbind(the.beatles.songs[, c(1, 2, 5, 6, 10)], discretized.features)
the.beatles.songs.nb <- the.beatles.songs.nb[, names(the.beatles.songs)]
```

Train and test datasets

Split the dataset into train and test sets:

```
# install.packages("caret")
library(caret)
set.seed(<n>)
<train dataset indices> <-                          # stratified partitioning:
+ createDataPartition(<dataset>$<output variable>,  # the same distribution of the output variable in both sets
+                     p = .80,                      # 80/20% of data in train/test sets
+                     list = FALSE)                 # don't make a list of results, make a matrix
<train dataset> <- <dataset>[<train dataset indices>, ]
<test dataset>  <- <dataset>[-<train dataset indices>, ]
```

```
library(caret)
set.seed(4455)
# set.seed(333) - results in a different split, and different results and eval. metrics
train.data.indices <- createDataPartition(the.beatles.songs.nb$Top.50.BB, p = 0.80, list = FALSE)
train.data <- the.beatles.songs.nb[train.data.indices, ]
test.data <- the.beatles.songs.nb[-train.data.indices, ]
```

Model

Build the model:

```
library(e1071)
?naiveBayes
<model> <- naiveBayes(<output variable> ~ .,          # include all predictors from the training set
+                     data = <training dataset>)
<model> <- naiveBayes(<output variable> ~
+                     <var 1> + <var 2> + ...,        # include only selected predictors from the training set
+                     data = <training dataset>)
print<model>                                          # shows P(o/c) for factor vars, mean() and sd() for numeric vars
```

```
library(e1071)
top.50.nb.1 <- naiveBayes(Top.50.BB ~ .,
                          data = train.data[, -1])
print(top.50.nb.1)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        No       Yes
## 0.8393574 0.1606426
##
## Conditional probabilities:
##      Year
## Y            1958        1960        1961        1962        1963
##   No  0.009569378 0.019138756 0.004784689 0.052631579 0.244019139
##   Yes 0.000000000 0.000000000 0.025000000 0.075000000 0.100000000
##      Year
## Y            1964        1965        1966        1967        1968
##   No  0.124401914 0.105263158 0.047846890 0.090909091 0.157894737
##   Yes 0.250000000 0.175000000 0.150000000 0.075000000 0.050000000
##      Year
## Y            1969        1970        1977        1980
##   No  0.138755981 0.004784689 0.000000000 0.000000000
##   Yes 0.100000000 0.000000000 0.000000000 0.000000000
##
##      Duration
## Y      [22.5,119]   (119,215]   (215,310]   (310,406]   (406,502]
##   No  0.148325359 0.755980861 0.081339713 0.009569378 0.004784689
##   Yes 0.050000000 0.825000000 0.100000000 0.000000000 0.025000000
##
##      Other.releases
## Y     [-0.056,11.2] (11.2,22.4] (22.4,33.6] (33.6,44.8] (44.8,56.1]
##   No    0.717703349 0.229665072 0.043062201 0.009569378 0.000000000
##   Yes   0.075000000 0.325000000 0.375000000 0.200000000 0.025000000
##
##      Single.certification
## Y              No RIAA 2xPlatinum RIAA 4xPlatinum   RIAA Gold
##   No  0.961722488     0.009569378     0.000000000 0.014354067
##   Yes 0.175000000     0.075000000     0.025000000 0.575000000
##      Single.certification
## Y     RIAA Gold, BPI Silver RIAA Platinum
##   No            0.004784689   0.009569378
##   Yes           0.000000000   0.150000000
##
##      Cover
## Y             N         Y
##   No  0.7416268 0.2583732
##   Yes 0.8750000 0.1250000
##
##      Covered.by
## Y      [-0.07,14]     (14,28]     (28,42]     (42,56]   (56,70.1]
##   No  0.904306220 0.062200957 0.028708134 0.004784689 0.000000000
##   Yes 0.500000000 0.150000000 0.225000000 0.100000000 0.025000000
##
##      Top.50.Rolling.Stone
## Y      [-0.05,10]     (10,20]     (20,30]     (30,40]     (40,50]
##   No  0.947368421 0.009569378 0.019138756 0.014354067 0.009569378
##   Yes 0.525000000 0.100000000 0.100000000 0.125000000 0.150000000
##
##      Top.50.NME
## Y      [-0.05,10]     (10,20]     (20,30]     (30,40]     (40,50]
```

```
##   No  0.933014354 0.028708134 0.009569378 0.009569378 0.019138756
##   Yes 0.575000000 0.075000000 0.150000000 0.125000000 0.075000000
```

Predictions

Make predictions:

```
<predictions> <- predict(object = <NB model>,
+                        newdata = <test dataset>,
+                        type = "class")
<predictions>[<i1>:<ik>]                           # examine some of the predictions
<predictions dataframe> <-
+ data.frame(<observation ID> = <test dataset>$<observation ID column>,
+            <another relevant feature> = <test dataset>$<another relevant feature column>,
+            ...,
+            <predictions feature> = <predictions>)
```

```
top.50.nb.predictions.1 <- predict(top.50.nb.1, newdata = test.data[, -1], type = "class")
top.50.nb.predictions.1[1:20]
```

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

```
top.50.nb.predictions.1.dataframe <- data.frame(Song = test.data$Title,
                                                Top.50.BB = test.data$Top.50.BB,
                                                Prediction = top.50.nb.predictions.1)
```

Evaluation

Compute confusion matrix:

```
<cm> <- table(True = <test dataset>$<output variable>,
+             Predicted = <test dataset>$<predictions>)
```

```
nb.cm.1 <- table(True = test.data$Top.50.BB, Predicted = top.50.nb.predictions.1)
nb.cm.1
```

```
##       Predicted
## True   No Yes
##   No  46   6
##   Yes  4   5
```

Compute evaluation metrics:

- accuracy = (TP + TN) / N
- precision = TP / (TP + FP)
- recall = TP / (TP + FN)
- F1 = (2 * precision * recall) / (precision + recall)

Note: precision and recall are inversely proportional to each other.

```
<evaluation metrics vector> <- <user-specified function>(<cm>)
# accuracy = sum(diag(cm)) / sum(cm)
# precision <- TP / (TP + FP)
# recall <- TP / (TP + FN)
# F1 <- (2 * precision * recall) / (precision + recall)
```

```
source("Evaluation metrics.R")
eval.nb.1 <- getEvaluationMetrics(nb.cm.1)
eval.nb.1
```

```
##  Accuracy Precision    Recall        F1
## 0.8360656 0.4545455 0.5555556 0.5000000
```

Build another model, using only selected predictors, and compute confusion matrix and evaluation metrics:

```
library(e1071)
top.50.nb.2 <- naiveBayes(Top.50.BB ~ Year + Duration + Other.releases,
                          data = train.data[, -1])
print(top.50.nb.2)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        No       Yes
## 0.8393574 0.1606426
##
## Conditional probabilities:
##      Year
## Y            1958        1960        1961        1962        1963
##   No  0.009569378 0.019138756 0.004784689 0.052631579 0.244019139
##   Yes 0.000000000 0.000000000 0.025000000 0.075000000 0.100000000
##      Year
## Y            1964        1965        1966        1967        1968
##   No  0.124401914 0.105263158 0.047846890 0.090909091 0.157894737
##   Yes 0.250000000 0.175000000 0.150000000 0.075000000 0.050000000
##      Year
## Y            1969        1970        1977        1980
##   No  0.138755981 0.004784689 0.000000000 0.000000000
##   Yes 0.100000000 0.000000000 0.000000000 0.000000000
##
##      Duration
## Y      [22.5,119]   (119,215]   (215,310]   (310,406]   (406,502]
##   No  0.148325359 0.755980861 0.081339713 0.009569378 0.004784689
##   Yes 0.050000000 0.825000000 0.100000000 0.000000000 0.025000000
##
##      Other.releases
## Y     [-0.056,11.2] (11.2,22.4] (22.4,33.6] (33.6,44.8] (44.8,56.1]
##   No    0.717703349 0.229665072 0.043062201 0.009569378 0.000000000
##   Yes   0.075000000 0.325000000 0.375000000 0.200000000 0.025000000
```

```
top.50.nb.predictions.2 <- predict(top.50.nb.2, newdata = test.data[, -1], type = "class")
top.50.nb.predictions.2[1:20]
```

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

```
top.50.nb.predictions.2.dataframe <- data.frame(Song = test.data$Title,
                                                Top.50.BB = test.data$Top.50.BB,
                                                Prediction = top.50.nb.predictions.2)
nb.cm.2 <- table(True = test.data$Top.50.BB, Predicted = top.50.nb.predictions.2)
nb.cm.2
```

```
##      Predicted
## True  No Yes
##   No  51   1
##   Yes  7   2
```

```
source("Evaluation metrics.R")
eval.nb.2 <- getEvaluationMetrics(nb.cm.2)
eval.nb.2
```

```
##  Accuracy Precision    Recall        F1
## 0.8688525 0.6666667 0.2222222 0.3333333
```

Compare the evaluation metrics of the two classifiers:

```
data.frame(rbind(<evaluation metrics 1>, <evaluation metrics 2>, ...),
+          row.names = c("<model 1>", "<model 2>", ...))
```

```
data.frame(rbind(eval.nb.1, eval.nb.2),
           row.names = c("eval.nb.1", "eval.nb.2"))
```

```
##            Accuracy Precision    Recall        F1
## eval.nb.1 0.8360656 0.4545455 0.5555556 0.5000000
## eval.nb.2 0.8688525 0.6666667 0.2222222 0.3333333
```

ROC curve (Receiver Operating Characteristic curve)

Important concepts:

- sensitivity (True Positive Rate, TPR) - the proportion of correctly identified positive cases (same as recall)
  - TPR = TP / (TP + FN)
- specificity (True Negative Rate, TNR) - the proportion of correctly identified negative cases
  - TNR = TN / (TN + FP)
- False Positive Rate, FPR - the proportion of incorrectly identified negative cases
  - FPR = 1 - TNR = FP / (TN + FP)

ROC curve is the plot representing the function: TPR = f(FPR). It can be used to select a probability threshold for the classifier (it does not have to be 0.5). To do this, making predictions is done by computing probabilities for each class value (suc as ‘Yes’ and ‘No’).

Build yet another model, to be used for plotting the ROC curve:

```
library(e1071)
?naiveBayes
<model> <- naiveBayes(<output variable> ~ .,          # include all predictors from the training set
+                     data = <training dataset>)
<model> <- naiveBayes(<output variable> ~
+                     <var 1> + <var 2> + ...,        # include only selected predictors from the training set
+                     data = <training dataset>)
print<model>                                          # shows P(o/c) for factor vars, mean() and sd() for numeric
```

```
library(e1071)
top.50.nb.3 <- naiveBayes(Top.50.BB ~ Year + Duration + Other.releases,      # can be the same as for top.50.n
b.2
                          data = train.data[, -1])
```

Make predictions as probabilities:

```
<predictions> <- predict(object = <NB model>,
+                        newdata = <test dataset>,
+                        type = "raw")             # the value "raw" means "compute probabilities, not classes"
<predictions>[<i1>:<ik>]                           # examine some of the predictions
<predictions dataframe> <-
+ data.frame(<observation ID> = <test dataset>$<observation ID column>,
+            <another relevant feature> = <test dataset>$<another relevant feature column>,
+            ...,
+            <predictions feature> = <predictions>)
```

```
top.50.nb.predictions.3 <- predict(top.50.nb.3, newdata = test.data[, -1], type = "raw")
top.50.nb.predictions.3[1:20, ]
```

```
##              No         Yes
##  [1,] 0.9649849 0.035015096
##  [2,] 0.9771835 0.022816472
##  [3,] 0.2784095 0.721590502
##  [4,] 0.6273678 0.372632229
##  [5,] 0.9845139 0.015486143
##  [6,] 0.8919640 0.108036045
##  [7,] 0.9951647 0.004835349
##  [8,] 0.9698362 0.030163823
##  [9,] 0.8919640 0.108036045
## [10,] 0.8064744 0.193525575
## [11,] 0.9931359 0.006864109
## [12,] 0.8064744 0.193525575
## [13,] 0.6705253 0.329474707
## [14,] 0.9579813 0.042018747
## [15,] 0.9931359 0.006864109
## [16,] 0.6273678 0.372632229
## [17,] 0.9845139 0.015486143
## [18,] 0.9359577 0.064042299
## [19,] 0.9760018 0.023998163
## [20,] 0.9931359 0.006864109
```

```
top.50.nb.predictions.3.dataframe <- data.frame(Song = test.data$Title,
                                                Top.50.BB = test.data$Top.50.BB,
                                                Prediction.probability.No = top.50.nb.predictions.3[, 1],
                                                Prediction.probability.Yes = top.50.nb.predictions.3[, 2])
```

Compute ROC curve parameters, the area under the curve (AUC), and plot the curve:

```
library(pROC)
<ROC curve parameters> <-                                # compute ROC curve parameters
+ roc(response = <test dataset>$<output variable>,
+     predictor = <predicted probabilities>[, <1 | 2>]) # col. no. of the "positive class" (can be the No class!)
<ROC curve parameters>$auc                               # extract and show AUC
```

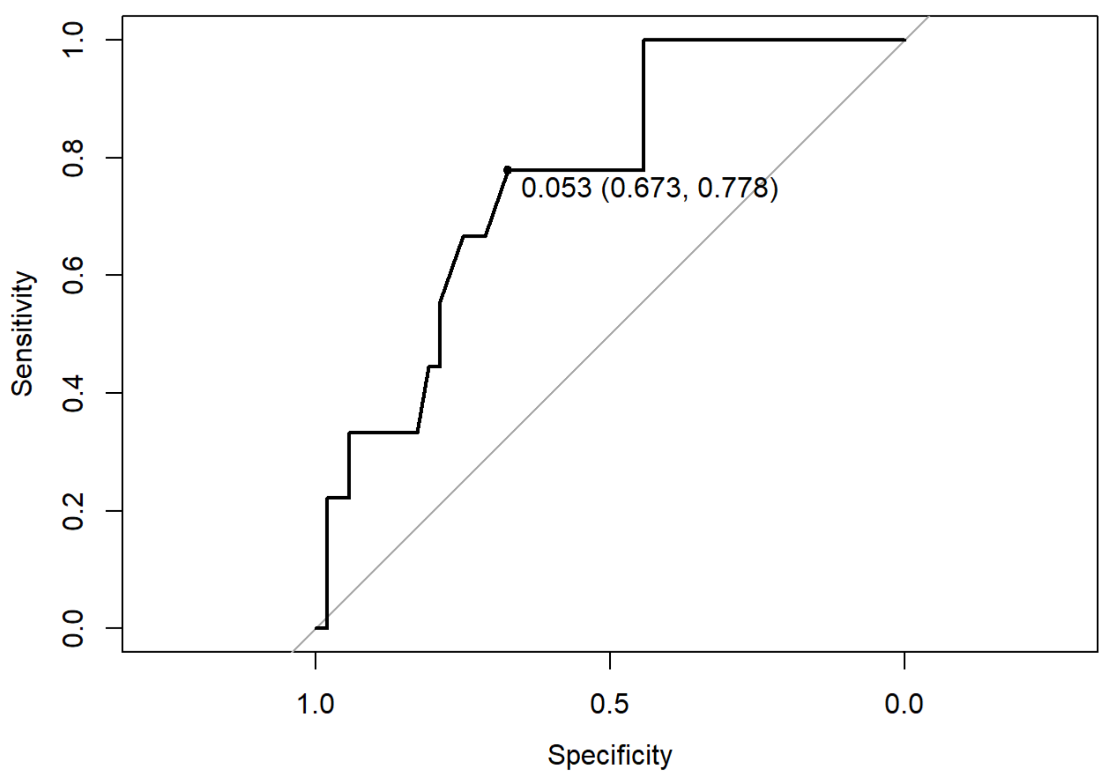
```
library(pROC)
top.50.nb.predictions.3.roc <-
  roc(response = test.data$Top.50.BB,
      predictor = top.50.nb.predictions.3[, 2])
top.50.nb.predictions.3.roc$auc
```

```
## Area under the curve: 0.7618
```

Plot the ROC curve:

```
plot.roc(<ROC curve parameters>,              # computed in the previous step
+        print.thres = TRUE,                  # show the probability threshold (cut-off point) on the plot
+        print.thres.best.method =
+          "youden" |          # maximize the sum of sensitivity and specificity (the distance to the diag. line)
+          "closest.topleft")  # minimize the distance to the top-left point of the plot
```

```
plot.roc(top.50.nb.predictions.3.roc,
         print.thres = TRUE,
         print.thres.best.method = "youden")
```



Getting the probability threshold and other ROC curve parameters using pROC::coords():

```
<ROC coords> <- coords(<ROC curve parameters>,                           # computed in the previous step
+                     ret = c("accuracy", "spec", "sens", "thr", ...),   # ROC curve parameters to return
+                     x =                    # the coordinates to look for:
+                         "local maximas" |  # local maximas of the ROC curve
+                         "best" | ...)      # the point with the best sum of sensitivity and specificity,
+                                            # i.e. the same as the one shown on the ROC curve <ROC coords>
```

```
top.50.nb.predictions.3.coords <-
  coords(top.50.nb.predictions.3.roc,
         ret = c("accuracy", "spec", "sens", "thr"),
         x = "local maximas")
top.50.nb.predictions.3.coords
```

```
##             local maximas local maximas local maximas local maximas
## accuracy       0.52459016    0.68852459     0.7377049     0.7540984
## specificity    0.44230769    0.67307692     0.7500000     0.7884615
## sensitivity    1.00000000    0.77777778     0.6666667     0.5555556
## threshold      0.02340732    0.05303052     0.1017091     0.1418192
##             local maximas local maximas local maximas local maximas
## accuracy        0.7540984     0.8524590     0.8688525      0.852459
## specificity     0.8076923     0.9423077     0.9807692      1.000000
## sensitivity     0.4444444     0.3333333     0.2222222      0.000000
## threshold       0.1903378     0.3474709     0.4897984           Inf
```

Resources, readings, references

10-fold cross-validation: https://www.openml.org/a/estimation-procedures/1 (https://www.openml.org/a/estimation-procedures/1)