NPFL142.C4DHI – tutorial #4
Machine learning with *Titanic*

Exercises with *Titanic* dataset 2
Exercise 1.1 – Loading the data set 2
Exercise 1.2 – Preprocessing the data sets 3
Exercise 1.3 – Machine learning - Naive Bayes algorithm 5
Exercise 1.4 – Machine learning - Decision Tree algorithm 5

Exercises with *Titanic* dataset

A task is to predict whether a passenger on the Titanic would have survived or not. We will use the data from Kaggle.

Exercise 1.1 – Loading the data set

In RStudio create a blank R script

- Move to Files&Plots desktop
- In Files manager
  - move to the home directory
  - create a new directory New folder > `4`
  - move to the directory `4` and create a new directory New folder > `titanic`
  - move to the directory `4/titanic`
  - use More in the menu to run Set as working directory
  - use New Blank File in the menu to create a blank R Script and name it `titanic.t.R`. Then the script is open in the Code editor window (upper-left window) and you can add the `commands` listed below to the script.

We suppose using these packages

```
library(tidyverse)
library(e1071)
library(caret)
library(rpart)
library(rpart.plot)
```

We will download the data from Kaggle, which previously organized a competition with *Titanic* dataset. Let's load the training data first.

```
train.url <-
"https://raw.githubusercontent.com/minsuk-heo/kaggle-titanic/master/input/train.csv"
train.data <- read_csv(url(train.url))
```

The test examples and their true target values are stored in two separate files, namely the feature vectors are in `test.csv` and their true target values in `gender_submission.csv`.

```
test.url <-
"https://raw.githubusercontent.com/minsuk-heo/kaggle-titanic/master/input/test.csv"
test.vectors <- read_csv(url(test.url))
true.url <-
"https://raw.githubusercontent.com/minsuk-heo/kaggle-titanic/master/input/gender_submission.csv"
true <- read_csv(url(true.url))
```

Exercise 1.2 – Preprocessing the data sets

We will merge the test feature vectors with their true target values so that the table (tibble) has the same number of columns as the training data (incl. their order).

```
test.data <- test.vectors %>%
  left_join(true, by = "PassengerId") %>%
  relocate(Survived, .after=PassengerId) # re-arrange the attributes
```

Further, we need to check if there are missing values in the data and consider how to handle them.

```
colSums(is.na(train.data))
```

There are 177 missing values of `Age`, 687 values of `Cabin` and 2 values of `Embarked` in the training data. In our experiments, we will focus on `Age` only. Let's replace `Age` missing values with the median of `Age`.

```
train.wo.na <- train.data %>%
  mutate(Age = if_else(is.na(Age), median(Age, na.rm = TRUE), Age)) # condition, true, false
```

In the training data, we didn't remove the examples with `Age` missing values because there are relatively many (177/891 = 19,9 %). However, we can remove the examples with `Age` missing values from the test data (86 in total).

```
colSums(is.na(test.data))
test.wo.na <- test.data %>%
  filter(!is.na(Age))
```

We won't be using the features `PassengerId`, `Name`, `Ticket`, `Cabin` in the experiments. Therefore we will exclude them (see `select` below). In the data, there are four categorical features, namely `Name`, `Sex`, `Cabin`, `Embarked`, while the rest are numeric. For our experiments, we need the features `Survived`, `Pclass`, `Sex`, `Embarked` to be factors. In R, a "factor" is a data type used to represent categorical attributes. Factors are used to store data that can take on a limited number of distinct values, such as categories/classes/levels. Factors are particularly useful because they allow for efficient handling of categorical data and enable the use of factor-specific methods and functions. When an attribute is converted into a factor, it assigns a level to each unique value in the attribute, which helps in organizing and analyzing the data effectively.

```
train <- train.wo.na %>%
  select(-c(PassengerId, Name, Ticket, Cabin)) %>%
  mutate(Survived = as.factor(Survived),
         Pclass = as.factor(Pclass),
         Sex = as.factor(Sex),
         Embarked = as.factor(Embarked))

test <- test.wo.na %>%
  select(-c(PassengerId, Name, Ticket, Cabin)) %>%
  mutate(Survived = as.factor(Survived),
         Pclass = as.factor(Pclass),
         Sex = as.factor(Sex),
         Embarked = as.factor(Embarked))
```

Check the results of data type conversion

```
str(train)
str(test)
```

Exercise 1.3 – Machine learning - Naive Bayes algorithm

A task is to predict whether the Titanic passenger survives. It is a binary classification task with a target attribute `Survived = {0, 1}`. Let's use the following features: `Pclass, Sex, Age, SibSp, Parch, Fare, Embarked` to predict a target value.

First, we use the Naive Bayes algorithm. The Naive Bayes algorithm is a probabilistic classification technique based on Bayes' theorem with the "naive" assumption of independence between features given a target attribute. It works by calculating the probability of each class given the input features and selecting the class with the highest probability as the predicted target value.

To run training, we will use **`naiveBayes`** function from **`e1071`** library. The training feature vectors are its first argument and the second argument is a vector of true target values of the training data.

```
# train
model.nb <- naiveBayes(x = train %>% select(-Survived), y = train$Survived)
```

The output of **`naiveBayes`** function is an object representing a trained Naive Bayes classifier model (**`model.nb`**). This model encapsulates the statistical information learned from the training data, including the probability distributions of the features given each target value, which are used to make classification on new data – **`test data`** in our case.

```
# test
test.nb <- predict(model.nb, test, type = "class")
```

The output of testing is a vector of 0s and 1s. The length of the vector corresponds to the number of test examples (332 in our case), and for each example, we know its predicted target value, either 0 or 1. For the classifier, we will create a confusion matrix and measure its performance using metrics such as Accuracy, Precision, Recall, F1 measure. We will create the confusion matrix using **`confusionMatrix`** function from **`caret`** library.

```
# evaluate
cm.nb <- confusionMatrix(data = test.nb, reference = test$Survived, positive = "1")
cm.nb$table
```

```
          Reference
Prediction   0   1
         0 178  36
         1  27  91
```

```
A.nb  <- round((cm.nb$overall["Accuracy"]), 2)
R.nb  <- round((cm.nb$byClass["Recall"]), 2)
P.nb  <- round((cm.nb$byClass["Precision"]), 2)
F.nb  <- round(2*P.nb*R.nb/(P.nb+R.nb),2)
tibble(Model="Naive Bayes", A.nb, R.nb, P.nb, F.nb)         # Naive Bayes performance
```

```
Model          A.nb  R.nb  P.nb  F.nb
  <chr>       <dbl> <dbl> <dbl> <dbl>
1 Naive Bayes  0.81  0.72  0.77  0.94
```

Exercise 1.4 – Machine learning - Decision Tree algorithm

Second, we use the Decision Tree algorithm. The Decision Tree algorithm is a supervised learning method used for both classification and regression tasks. It works by recursively splitting the dataset into subsets based on the value of features. At each step of the tree-building process, the algorithm selects the best feature and split point that maximizes some measure. This process continues until a stopping criterion is met. The resulting tree structure can be easily interpreted and visualized, making it a popular choice for understanding decision-making processes in data analysis.

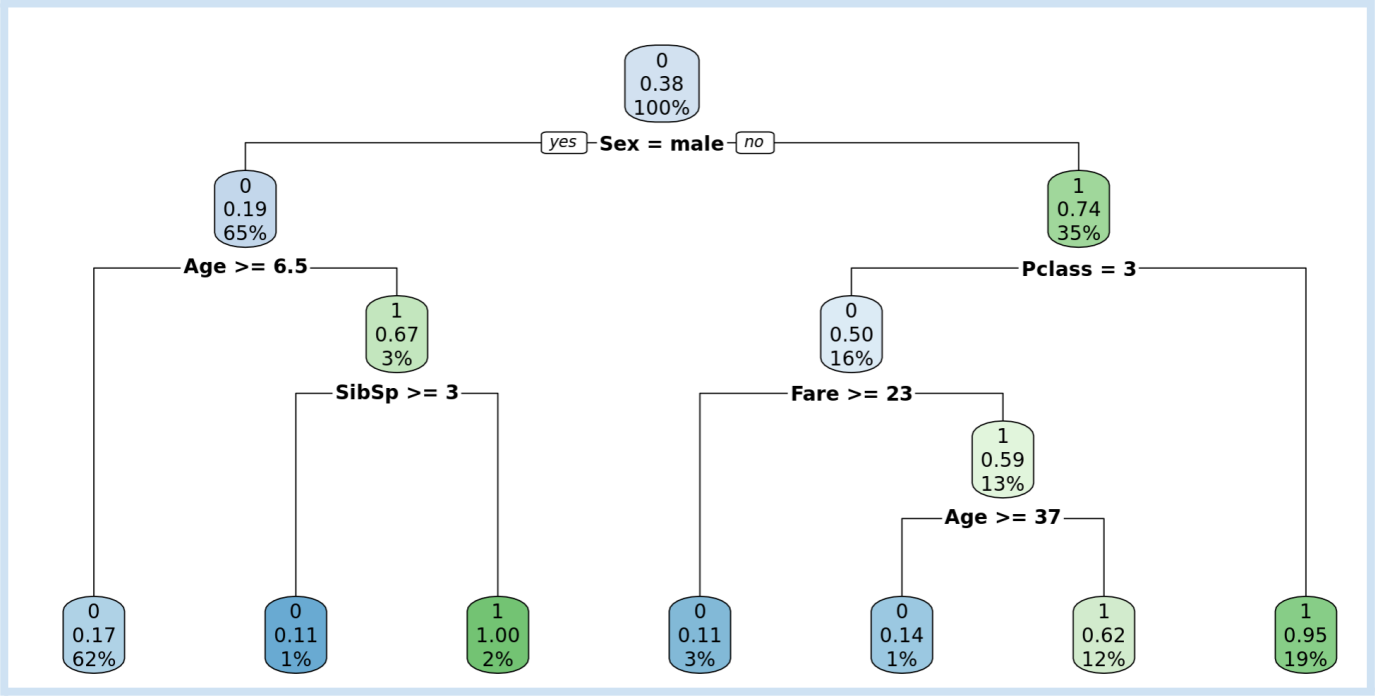
To run training, we will use `rpart` function from `rpart` library. The first argument of `rpart` function is a formula that includes a target attribute (`Survived`) and features used for predicting the target value. Formulas in R provide a flexible and powerful way to specify relationships between target attributes and features. The general form of a formula is: `target.attribute ~ features`. We use the symbol "." in a formula (see below) - it means that we use all the features in our dataset for predicting the target value.

```
# train
model.dt <- rpart(Survived~ .,train)
```

Plot the model using `rpart.plot` function from `rpart.plot` library.

```
rpart.plot(model.dt)
```

Each node shows the predicted target attribute value, the predicted probability of `Survived` and the percentage of examples in the node.



Test **model.dt** on **test** data. The output of the **predict** function is a vector of predicted target values (**test.dt**) for each example in **test** data.

```
# test
test.dt <- predict(model.dt, newdata = test, type = "class")
```

```
# evaluate
cm.dt <- confusionMatrix(data = test.dt, reference = test$Survived, positive = "1")

A.dt  <- round((cm.nb$overall["Accuracy"]), 2)
R.dt  <- round((cm.nb$byClass["Recall"]), 2)
P.dt  <- round((cm.nb$byClass["Precision"]), 2)
F.dt  <- round(2*P.dt*R.dt/(P.dt+R.dt),2)

tibble(Model="Decision Trees", A.dt, R.dt, P.dt, F.dt)      # Decision Trees performance
```

	Model	A.dt	R.dt	P.dt	F.dt
	<chr>	*<dbl>*	*<dbl>*	*<dbl>*	*<dbl>*
1	Decision trees	0.96	0.94	0.95	0.94