**Exercise**

**Imbalanced class distribution**

The dataset transfers contains credit transfers and some of them were recorded as fraud. The column fraud\_flag indicates whether the transaction is fraudulent (fraud\_flag = 1) or not (fraud\_flag = 0).

Since fraud is typically very rare, it is important to take the large imbalance between the number of fraudulent cases and regular cases into account. Let's check the fraction of legitimate and fraudulent cases and visualize the imbalance with a pie chart.

The dataset transfers is loaded in your workspace. The visualization part has been defined for you, as data visualization in general is out of the scope of this course.

**Question 1: Print the first 6 rows of the transfers to familiarize yourself with the data.**

# Print the first 6 rows of the dataset

head(transfers)

**Question 2: Print the structure of transfers to better understand each feature.**

# Display the structure of the dataset

str(transfers)

**Question 3: Use table() on the column fraud\_flag, then check determine the fraction of legitimate and fraudulent cases in the dataset using prop.table() on the obtained result. Store the result in a variable called class\_distribution.**

# Determine fraction of legitimate and fraudulent cases

class\_distribution <- prop.table(table(transfers$fraud\_flag))

print(class\_distribution)

Question 4: # Make pie chart of column fraud\_flag

# Make pie chart of column fraud\_flag

df <- data.frame(class = c("no fraud", "fraud"),

pct = as.numeric(class\_distribution)) %>%

mutate(class = factor(class, levels = c("no fraud", "fraud")),

cumulative = cumsum(pct), midpoint = cumulative - pct / 2,

label = paste0(class, " ", round(pct\*100, 2), "%"))

ggplot(df, aes(x = 1, weight = pct, fill = class)) +

scale\_fill\_manual(values = c("dodgerblue", "red")) +

geom\_bar(width = 1, position = "stack") +

coord\_polar(theta = "y") +

geom\_text(aes(x = 1.3, y = midpoint, label = label)) +

theme\_nothing()

# Final Result:

# # Print the first 6 rows of the dataset

# head(transfers)

# # Display the structure of the dataset

# str(transfers)

# # Determine fraction of legitimate and fraudulent cases

# class\_distribution <- prop.table(table(transfers$fraud\_flag))

# print(class\_distribution)

# # Make pie chart of column fraud\_flag

# df <- data.frame(class = c("no fraud", "fraud"),

# pct = as.numeric(class\_distribution)) %>%

# mutate(class = factor(class, levels = c("no fraud", "fraud")),

# cumulative = cumsum(pct), midpoint = cumulative - pct / 2,

# label = paste0(class, " ", round(pct\*100, 2), "%"))

# ggplot(df, aes(x = 1, weight = pct, fill = class)) +

# scale\_fill\_manual(values = c("dodgerblue", "red")) +

# geom\_bar(width = 1, position = "stack") +

# coord\_polar(theta = "y") +

# geom\_text(aes(x = 1.3, y = midpoint, label = label)) +

# theme\_nothing()

# <https://campus.datacamp.com/courses/fraud-detection-in-r/introduction-motivation?ex=3#skiponboarding>

# Cost of not detecting fraud

When no detection model is used, then all transactions in the transfers dataset are considered legitimate. You will determine the corresponding confusion matrix. Despite fraud being rare, the resulting financial losses can be huge. You will compute the total cost of not detecting the fraudulent transfers.

The caret package is already loaded for you to construct the confusionMatrix(). The transfers dataset is loaded in your workspace, don't hesitate to explore it in the Console.

**Instructions**

**100 XP**

* Create a vector called predictions in which all transfers are predicted as legitimate (class 0). at the slides on the right of the Console to see how this function was used in the video.
* Use the function confusionMatrix() from the caret package to compute the confusion matrix of predictions and the fraud\_flag column from transfers.
* Compute the total cost of not detecting fraud as the sum of fraudulent transferred amounts.

# # Create vector predictions containing 0 for every transfer

# predictions <- factor(rep.int(0, nrow(transfers)), levels = c(0, 1))

# # Compute confusion matrix

# confusionMatrix(data = predictions, reference = transfers$fraud\_flag)

# # Compute cost of not detecting fraud

# cost <- sum(transfers$amount[transfers$fraud\_flag == 1])

# print(cost)

# Circular histogram

The circular histogram is a visual representation of the timestamps of events. It can be used to analyze a person's typical behavior. The dataset timestamps contains 25 timestamps of transactions made by Mr. Cooper. You're going to get the periodic mean and plot the histogram. Keep an eye on the visualizations to spot anything out of the ordinary!

The dataset timestamps has been loaded in your workspace, as well as the lubridate, circular and ggplot packages.

# Question1: The timestamps are recorded as text. Convert the data to hours on a 24 hour clock using hms() from the lubridate package.

# # Convert the plain text to hours

# ts <- as.numeric(hms(timestamps)) / 3600

# Question 2: Create a Von Mises distribution: use circular() with hourly units on a 24 hour clock to convert the data as circular.

# # Convert the data to class circular

# ts <- circular(ts, units = "hours", template = "clock24")

# # Plot a circular histogram

# clock <- ggplot(data.frame(ts), aes(x = ts)) +

# geom\_histogram(breaks = seq(0, 24), colour = "blue", fill = "lightblue") +

# coord\_polar() + scale\_x\_continuous("", limits = c(0, 24), breaks = seq(0, 24))

# plot(clock)

# Question 3: Most of the transactions are apparently around 10 AM. Create the von Mises distribution estimates

# # Create the von Mises distribution estimates

# estimates <- mle.vonmises(ts)

# Question 4: Extract the periodic mean mu. Click Submit Answer to add the periodic mean as a vertical line to the circular histogram. Can you spot an anomalous timestamp?

# # Extract the periodic mean from the estimates

# p\_mean <- estimates$mu %% 24

# # Add the periodic mean to the circular histogram

# clock <- ggplot(data.frame(ts), aes(x = ts)) +

# geom\_histogram(breaks = seq(0, 24), colour = "blue", fill = "lightblue") +

# coord\_polar() + scale\_x\_continuous("", limits = c(0, 24), breaks = seq(0, 24)) +

# geom\_vline(xintercept = as.numeric(p\_mean), color = "red", linetype = 2, size = 1.5)

# plot(clock)

# Final Result:

# # Convert the plain text to hours

# ts <- as.numeric(hms(timestamps)) / 3600

# # Convert the data to class circular

# ts <- circular(ts, units = "hours", template = "clock24")

# # Estimate the periodic mean from the von Mises distribution

# estimates <- mle.vonmises(ts)

# # Extract the periodic mean from the estimates

# p\_mean <- estimates$mu %% 24

# # Add the periodic mean to the circular histogram

# clock <- ggplot(data.frame(ts), aes(x = ts)) +

# geom\_histogram(breaks = seq(0, 24), colour = "blue", fill = "lightblue") +

# coord\_polar() + scale\_x\_continuous("", limits = c(0, 24), breaks = seq(0, 24)) +

# geom\_vline(xintercept = as.numeric(p\_mean), color = "red", linetype = 2, size = 1.5)

# plot(clock)

**Suspicious timestamps**

**A confidence interval (CI) for the time of a transaction can indicate a suspicious timestamp. By estimating the parameters mu and kappa of the von Mises distribution on previous timestamps, you can calculate the density (or likelihood) of a new timestamp.**

**The dataset ts containing all timestamps and the circular package are already loaded. The estimates of the first 24 timestamps are available in your workspace, as well as the probability level alpha set to 95%.**

**Instructions**

**100 XP**

* Get the periodic mean (mu) and the concentration (kappa) of the first 24 estimates.
* Use dvonmises() to estimate the densities of all timestamps in ts.
* Use dvonmises() and qvonmises() to determine the 95% cutoff value for (1 - alpha)/2). Refer to the slides if necessary!
* Define the variable time\_feature: it should be true if densities are greater than or equal to the cutoff and false otherwise. Submit answer to see which timestamps lie outside the 95% confidence interval

# # Estimate the periodic mean and concentration on the first 24 timestamps

# p\_mean <- estimates$mu %% 24

# concentration <- estimates$kappa

# # Estimate densities of all 25 timestamps

# densities <- dvonmises(ts, mu = p\_mean, kappa = concentration)

# # Check if the densities are larger than the cutoff of 95%-CI

# cutoff <- dvonmises(qvonmises((1 - alpha)/2, mu = p\_mean, kappa = concentration), mu = p\_mean, kappa = concentration)

# # Define the variable time\_feature

# time\_feature <- densities >= cutoff

# print(cbind.data.frame(ts, time\_feature))

# Frequency feature for one account

A frequency feature counts how frequently a certain event has happened in the past. Creating such features helps detecting anomalous behavior. In the video, you learned how to create a frequency feature based on a categorical feature.

You're now provided with transactional data from Bob. One of the columns is called channel\_cd which indicates the payment channel that Bob used to book each of his transactions. You'll be creating a frequency feature called freq\_channel based on the column channel\_cd using the function rollapply(). You can use ?rollaply in the console to see the function documentation.

The dataset trans\_Bob, the zoo and dplyr packages are loaded in your workspace.

* Write a function frequency\_fun() which takes steps and channel as inputs, counts the number of steps, and sums how often the latest channel has been used in the past.
* Create the feature freq\_channel by using the function rollapply on the transfer\_id column. The feature should count how often a particular channel\_cd has been used before.
* Print the features channel\_cd, freq\_channel and fraud\_flag. Inspect the newly created feature.

# # Frequency feature based on channel\_cd

# frequency\_fun <- function(steps, channel) {

# n <- length(steps)

# frequency <- sum(channel[1:n] == channel[n + 1])

# return(frequency)

# }

# # Create freq\_channel feature

# freq\_channel <- rollapply(trans\_Bob$transfer\_id, width = list(-1:-length(trans\_Bob$transfer\_id)), partial = TRUE, FUN = frequency\_fun, trans\_Bob$channel\_cd)

# # Print the features channel\_cd, freq\_channel and fraud\_flag next to each other

# freq\_channel <- c(0, freq\_channel)

# cbind.data.frame(trans\_Bob$channel\_cd, freq\_channel, trans\_Bob$fraud\_flag)

# Frequency feature for multiple accounts

# Now that you know how to create a frequency feature for one account or person, let's do the same for multiple accounts at once. The dataset trans contains transactions made by Alice and Bob. Create the frequency feature freq\_channel based on the same column channel\_cd as the previous exercise.

# The dataset trans, the zoo and dplyr packages are already loaded. The function frequency\_fun() that you have written in the previous exercise is available in your workspace.

# Question 1: Group the data based on account\_name.

# # Group the data by account

# trans <- trans %>% group\_by(account\_name)

# Question 2: Add a new feature to the dataset with mutate(). For each account, freq\_channel has to start with a zero.

# # Mutate the data to add a new feature

# mutate(freq\_channel = c(0))

# Question 3: For each account, freq\_channel has to start with a zero.

# # Rollapply frequency\_fun to the data

# rollapply(transfer\_id,

# width = list(-1:-length(transfer\_id)),

# partial = TRUE,

# FUN = frequency\_fun, channel\_cd)))

# Question 4: Print the features account\_name, channel\_cd, freq\_channel and fraud\_flag as columns next to each other and inspect the newly created feature.

# # Print the features as columns next to each other

# as.data.frame(trans %>% select(account\_name, channel\_cd, freq\_channel, fraud\_flag))

# Recency feature

# A recency feature says how recent a certain event has happened in the past. The more recent an event has occurred, the closer its recency will be to 1. If a new and previously unseen case occurs, its recency will be 0. Such features helps detecting anomalous behavior. In the video, you learned how to create a recency feature based on a categorical feature. You're provided with the dataset trans containing transactions made by Alice and Bob. You're going to create a recency feature called rec\_channel based on the column channel\_cd.

# The zoo and dplyr packages are loaded for you. The frequency feature freq\_channel from the previous exercise was added to the dataset trans. trans$timestamp are converted to hourly format, and gamma has been set for you to -log(0.01)/90.

# Question 1: Write the first part of a function recency\_fun() which takes t, gamma, channel\_cd and freq\_channel as inputs, and computes the recency. If a channel has never been used, it should return 0.

# # Create the recency function

# recency\_fun <- function(t, gamma, channel\_cd, freq\_channel) {

# n\_t <- length(t)

# # If the channel has never been used, return 0

# if (freq\_channel[n\_t] == 0) {

# return(0)

# }

# }

# Question 2: If a channel has already been used, compute the time difference (this has been done for you), set the exponent equal to the negative of gamma multiplied by the time difference, and return its exponential value.

# # Create the recency function

# recency\_fun <- function(t, gamma, channel\_cd, freq\_channel) {

# n\_t <- length(t)

# # If the channel has never been used, return 0

# if (freq\_channel[n\_t] == 0) {

# return(0)

# # Else, return the exponent

# } else {

# time\_diff <- t[1] - max(t[2:n\_t][channel\_cd[(n\_t-1):1] == channel\_cd[n\_t]])

# exponent <- -gamma \* time\_diff

# return(exp(exponent))

# }

# }

# Question 3: Group the data by account name and mutate it to add a new feature rec\_channel. This feature should rollapply the recency\_func you created.

# # Create the recency function

# recency\_fun <- function(t, gamma, channel\_cd, freq\_channel) {

# n\_t <- length(t)

# if (freq\_channel[n\_t] == 0) {

# return(0)

# } else {

# time\_diff <- t[1] - max(t[2:n\_t][channel\_cd[(n\_t-1):1] == channel\_cd[n\_t]])

# exponent <- -gamma \* time\_diff

# return(exp(exponent))

# }

# }

# # Group, mutate and rollapply

# trans <- trans %>% group\_by(account\_name) %>%

# mutate(rec\_channel = rollapply(timestamp, width = list(0:-length(transfer\_id)), partial = TRUE,

# FUN = recency\_fun, gamma, channel\_cd, freq\_channel))

# Question 4: Print a new dataframe containing the account\_name, channel\_cd, timestamp, rec\_channel and fraud\_flag

# # Print a new dataframe

# as.data.frame(trans %>% select(account\_name, channel\_cd, timestamp, rec\_channel, fraud\_flag))

# Comparing frequency & recency

# Now that you've created frequency and recency features, let's compare them between the legitimate transactions and the fraudulent ones. The dataset transfers contains 222 transactions from 4 accounts. The frequency features freq\_channel and freq\_auth, and the recency features rec\_channel and rec\_auth have been added as columns to the dataset.

* Have a look at dataset transfers in the console using functions like head and str.
* Get a summary of the frequency and recency channels for legitimate transactions.
* Get a summary of the frequency and recency channels for fraudulent transactions.

# library(dplyr)

# # Statistics of frequency & recency features of legitimate transactions:

# summary(transfers %>% filter(fraud\_flag == 0) %>% select(freq\_channel, freq\_auth, rec\_channel, rec\_auth))

# # Statistics of frequency & recency features of fraudulent transactions:

# summary(transfers %>% filter(fraud\_flag == 1) %>% select(freq\_channel, freq\_auth, rec\_channel, rec\_auth))