**Customer churn**

Customer churn describes the problem that customers may leave a business. The reasons for leaving can be manifold but some of the most common are:

dissatisfaction with the product, service, etc. too expensive

don’t need the service any longer

…

Customer churn models generally fall into two categories:

1. Aim to identify customers who are on the brink of leaving, so that appropriate measures can be taken to try and retain them (as a function over time).

A simple model as shown below takes a snapshot of data into account but as is quite obvious: customer churn likelihoods change over time. So, real models will most likely take into account features that describe changes over time, like customer engagement with the site or product, number of calls to customer service, etc.

1. Aim to identify customer segments who are more likely to churn after a certain period of time (e.g. men being more likely to churn than women, or younger people changing contracts more often than older people, etc.).

Simple customer churn models use the customer’s information (see below for example data: demographic information, services that customers have signed up for and account information) to try to predict the likelihood of churn for each customer.

More advanced models will also try to classify the reason for potential churn (see above). Depending on the reason, specific actions can be taken to try and prevent the customer from churning (e.g. offering them a better deal when you think they might leave due to finding the service too costly).

**Setup**

All analyses are done in R using RStudio. For detailed session information including R version, operating system and package versions, see the sessionInfo() output at the end of this

document.

All figures are produced with ggplot2.

Libraries

# Load libraries

library(tidyverse) # for tidy data analysis library(readr) # for fast reading of input files library(caret) # for convenient splitting

library(mice) # mice package for Multivariate Imputation by Chained Equations (MICE)

library(keras) # for neural nets library(lime) # for explaining neural nets

library(rsample) # for splitting training and test data library(recipes) # for preprocessing

library(yardstick) # for evaluation library(ggthemes) # for additional plotting themes library(corrplot) # for correlation

theme\_set(theme\_minimal())

# Install Keras if you have not installed it before

# follow instructions if you haven't installed TensorFlow install\_keras()

**Data preparation**

# The dataset

UPDATE: The old link doesn’t seem to exist any longer but the dataset is still available from Kaggle.

The Telco Customer Churn data set is the same one that Matt Dancho used in his post (see above). It was downloaded from Kaggle.

churn\_data\_raw <- read\_csv("WA\_Fn-UseC\_-Telco-Customer-Churn.csv") glimpse(churn\_data\_raw)

## Rows: 7,043

## Columns: 21

## $ customerID "7590-VHVEG", "5575-GNVDE", "3668-QPYBK", "7795-

CFOCW…

## $ gender "Female", "Male", "Male", "Male", "Female", "Female",…

## $ SeniorCitizen 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0,…

## $ Partner "Yes", "No", "No", "No", "No", "No", "No", "No", "Yes…

## $ Dependents "No", "No", "No", "No", "No", "No", "Yes", "No", "No"…

## $ tenure 1, 34, 2, 45, 2, 8, 22, 10, 28, 62, 13, 16, 58,

49, 2…

## $ PhoneService "No", "Yes", "Yes", "No", "Yes", "Yes", "Yes",

"No", …

## $ MultipleLines "No phone service", "No", "No", "No phone service", "…

## $ InternetService "DSL", "DSL", "DSL", "DSL", "Fiber optic", "Fiber opt…

## $ OnlineSecurity "No", "Yes", "Yes", "Yes", "No", "No", "No", "Yes", "…

## $ OnlineBackup "Yes", "No", "Yes", "No", "No", "No", "Yes", "No", "N…

## $ DeviceProtection "No", "Yes", "No", "Yes", "No", "Yes", "No", "No", "Y…

## $ TechSupport "No", "No", "No", "Yes", "No", "No", "No", "No", "Yes…

## $ StreamingTV "No", "No", "No", "No", "No", "Yes", "Yes", "No", "Ye…

## $ StreamingMovies "No", "No", "No", "No", "No", "Yes", "No", "No", "Yes…

## $ Contract "Month-to-month", "One year", "Month-to-month", "One …

## $ PaperlessBilling "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "No", …

## $ PaymentMethod "Electronic check", "Mailed check", "Mailed check", "…

## $ MonthlyCharges 29.85, 56.95, 53.85, 42.30, 70.70, 99.65, 89.10, 29.7…

## $ TotalCharges 29.85, 1889.50, 108.15, 1840.75, 151.65, 820.50, 1949…

## $ Churn "No", "No", "Yes", "No", "Yes", "Yes", "No", "No", "Y…

## EDA

Proportion of churn (customers who left within the last month):

churn\_data\_raw %>% count(Churn)

## # A tibble: 2 x 2 ## Churn n

##

|  |  |  |  |
| --- | --- | --- | --- |
| ## | 1 | No | 5174 |
| ## | 2 | Yes | 1869 |

Plot categorical features (demographic information, services that customers have signed up for and account information):

churn\_data\_raw %>%

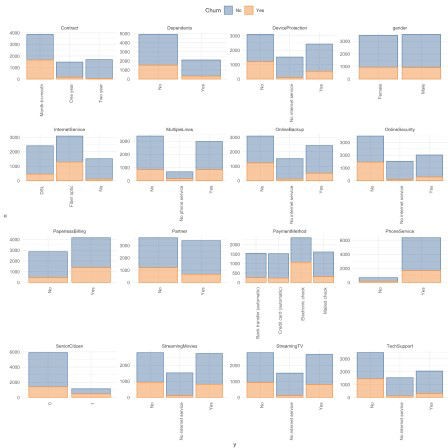
mutate(SeniorCitizen = as.character(SeniorCitizen)) %>% select(-customerID) %>%

select\_if(is.character) %>% select(Churn, everything()) %>% gather(x, y, gender:PaymentMethod) %>% count(Churn, x, y) %>%

ggplot(aes(x = y, y = n, fill = Churn, color = Churn)) +

facet\_wrap(~ x, ncol = 4, scales = "free") + geom\_bar(stat = "identity", alpha = 0.5) + theme(axis.text.x = element\_text(angle = 90, hjust = 1),

legend.position = "top") + scale\_color\_tableau() + scale\_fill\_tableau()



Plot numerical features (demographic information, services that customers have signed up for and account information):

churn\_data\_raw %>% select(-customerID) %>%

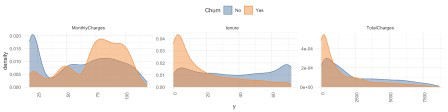
#select\_if(is.numeric) %>%

select(Churn, MonthlyCharges, tenure, TotalCharges) %>% gather(x, y, MonthlyCharges:TotalCharges) %>% ggplot(aes(x = y, fill = Churn, color = Churn)) +

facet\_wrap(~ x, ncol = 3, scales = "free") + geom\_density(alpha = 0.5) +

theme(axis.text.x = element\_text(angle = 90, hjust = 1), legend.position = "top") +

scale\_color\_tableau() + scale\_fill\_tableau()



Remove customer ID as it doesn’t provide information

churn\_data <- churn\_data\_raw %>% select(-customerID)

## Dealing with missing values

Pattern of missing data

md.pattern(churn\_data, plot = FALSE

## gender SeniorCitizen Partner Dependents tenure PhoneService MultipleLines

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ## | 7032 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 |  |  |  |  |  |  |  |
| ## | 11 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 |  |  |  |  |  |  |  |
| ## |  | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 |  |  |  |  |  |  |  |

## InternetService OnlineSecurity OnlineBackup DeviceProtection TechSupport

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## | 7032 | 1 | 1 | 1 | 1 |
| 1 |  |  |  |  |  |
| ## | 11 | 1 | 1 | 1 | 1 |
| 1 |  |  |  |  |  |
| ## |  | 0 | 0 | 0 | 0 |
| 0 |  |  |  |  |  |

## StreamingTV PaymentMethod

|  |  |  |
| --- | --- | --- |
| StreamingMovies | Contract | PaperlessBilling |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

1

|  |  |
| --- | --- |
| ## | 7032 |
| 1 |  |
| ## | 11 |
| 1 |  |
| ## |  |

1

0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0  ## | MonthlyCharges | Churn | TotalCharges |  |
| ## | 7032 1 | 1 | 1 | 0 |
| ## | 11 1 | 1 | 0 | 1 |
| ## | 0 | 0 | 11 | 11 |

Option 1: impute missing data => NOT done here!

imp <- mice(data = churn\_data, print = FALSE) train\_data\_impute <- complete(imp, "long")

Option 2: drop missing data => done here because not too much information is lost by removing it

churn\_data <- churn\_data %>% drop\_na()

# Training and test split

Partition data into training and test set

set.seed(42)

index <- createDataPartition(churn\_data$Churn, p = 0.7, list = FALSE)

Partition test set again into validation and test set

train\_data <- churn\_data[index, ] test\_data <- churn\_data[-index, ]

index2 <- createDataPartition(test\_data$Churn, p = 0.5, list = FALSE)

valid\_data <- test\_data[-index2, ] test\_data <- test\_data[index2, ] nrow(train\_data)

## [1] 4924

nrow(valid\_data) ## [1] 1054

nrow(test\_data) ## [1] 1054

# Pre-Processing

Create recipe for preprocessing

A recipe is a description of what steps should be applied to a data set in order to get it ready for data analysis.

recipe\_churn <- recipe(Churn ~ ., train\_data) %>% step\_dummy(all\_nominal(), -all\_outcomes()) %>% step\_center(all\_predictors(), -all\_outcomes()) %>% step\_scale(all\_predictors(), -all\_outcomes()) %>% prep(data = train\_data)

Apply recipe to three datasets

train\_data <- bake(recipe\_churn, new\_data = train\_data) %>% select(Churn, everything())

valid\_data <- bake(recipe\_churn, new\_data = valid\_data) %>% select(Churn, everything())

test\_data <- bake(recipe\_churn, new\_data = test\_data) %>% select(Churn, everything())

For Keras create response variable as one-hot encoded matrix

train\_y\_drop <- keras::to\_categorical(as.integer(as.factor(train\_data$ Churn)) - 1, 2)

colnames(train\_y\_drop) <- c("No", "Yes")

valid\_y\_drop <- keras::to\_categorical(as.integer(as.factor(valid\_data$ Churn)) - 1, 2)

colnames(valid\_y\_drop) <- c("No", "Yes")

test\_y\_drop <- keras::to\_categorical(as.integer(as.factor(test\_data$

Churn)) - 1, 2)

colnames(test\_y\_drop) <- c("No", "Yes")

Because we want to train on a binary outcome, we can delete the “No” column

# if training with binary crossentropy train\_y\_drop <- train\_y\_drop[, 2, drop = FALSE] head(train\_y\_drop)

|  |  |  |
| --- | --- | --- |
| ##  ## | [1,] | Yes  0 |
| ## | [2,] | 0 |
| ## | [3,] | 1 |
| ## | [4,] | 0 |
| ## | [5,] | 1 |
| ## | [6,] | 0 |

valid\_y\_drop <- valid\_y\_drop[, 2, drop = FALSE] test\_y\_drop <- test\_y\_drop[, 2, drop = FALSE]

Remove response variable from preprocessed data (for Keras)

train\_data\_bk <- select(train\_data, -Churn) head(train\_data\_bk)

## # A tibble: 6 x 30

## SeniorCitizen tenure MonthlyCharges TotalCharges gender\_Male Partner\_Yes

##

## 1 -0.439 -1.28 -1.17 -1.00 -1.02

1.05

|  |  |
| --- | --- |
| ## 2 | -0.439 |
| -0.956 |  |
| ## 3 | -0.439 |
| -0.956 |  |
| ## 4 | -0.439 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0.0678 -0.262 | | | | -0.173 | 0.983 |
| -1.24 -0.366 | | | | -0.965 | 0.983 |
| 0.518 -0.751 | | | | -0.195 | 0.983 |
| -0.956 |  |  |  |  | |
| ## 5 | -0.439 | -1.24 | 0.197 | -0.946 -1.02 | |
| -0.956 |  |  |  |  | |
| ## 6 | -0.439 | -0.423 | 0.811 | -0.146 0.983 | |
| -0.956 |  |  |  |  | |

## # … with 24 more variables: Dependents\_Yes , PhoneService\_Yes , ## # MultipleLines\_No.phone.service , MultipleLines\_Yes ,

## # InternetService\_Fiber.optic , InternetService\_No ,

## # OnlineSecurity\_No.internet.service , OnlineSecurity\_Yes , ## # OnlineBackup\_No.internet.service , OnlineBackup\_Yes ,

## # DeviceProtection\_No.internet.service , DeviceProtection\_Yes , ## # TechSupport\_No.internet.service , TechSupport\_Yes ,

## # StreamingTV\_No.internet.service , StreamingTV\_Yes ,

## # StreamingMovies\_No.internet.service , StreamingMovies\_Yes , ## # Contract\_One.year , Contract\_Two.year ,

## # PaperlessBilling\_Yes , PaymentMethod\_Credit.card..automatic. , ## # PaymentMethod\_Electronic.check , PaymentMethod\_Mailed.check valid\_data\_bk <- select(valid\_data, -Churn)

test\_data\_bk <- select(test\_data, -Churn)

Alternative to above, to convert response variable into numeric format where 1 = Yes and 0 = No

train\_data$Churn <- ifelse(train\_data$Churn == "Yes", 1, 0) valid\_data$Churn <- ifelse(valid\_data$Churn == "Yes", 1, 0) test\_data$Churn <- ifelse(test\_data$Churn == "Yes", 1, 0)

# Modeling with Keras

Define a simple MLP

model\_keras <- keras\_model\_sequential(

model\_keras %>%

layer\_dense(units = 32, kernel\_initializer = "uniform", activation = "relu",

input\_shape = ncol(train\_data\_bk)) %>% layer\_dropout(rate = 0.2) %>%

layer\_dense(units = 16, kernel\_initializer = "uniform", activation = "relu") %>%

layer\_dropout(rate = 0.2) %>%

layer\_dense(units = 8, kernel\_initializer = "uniform", activation = "relu") %>%

layer\_dropout(rate = 0.2) %>%

layer\_dense(units = 1,

kernel\_initializer = "uniform", activation = "sigmoid")

%>%

compile(

optimizer = 'adamax',

loss = 'binary\_crossentropy', metrics = c("binary\_accuracy", "mse")

)

Fit model (we could have used validation split on the trainings data instead of creating a validation set => see #)

fit\_keras <- fit(model\_keras,

x = as.matrix(train\_data\_bk), y = train\_y\_drop,

batch\_size = 32,

epochs = 20,

#validation\_split = 0.30,

validation\_data = list(as.matrix(valid\_data\_bk), valid\_y\_drop), verbose = 2

)

Epoch 1/20

154/154 - 0s - loss: 0.6728 - binary\_accuracy: 0.7342 - mse: 0.2399 154/154 - 1s - loss: 0.6728 - binary\_accuracy: 0.7342 - mse: 0.2399 -

154/154 - 0s - loss: 0.5564 - binary\_accuracy: 0.7342 - mse: 0.1877 154/154 - 0s - loss: 0.5564 - binary\_accuracy: 0.7342 - mse: 0.1877 - val\_loss: 0.4816 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1586 Epoch 3/20

154/154 - 0s - loss: 0.4920 - binary\_accuracy: 0.7342 - mse: 0.1630 154/154 - 0s - loss: 0.4920 - binary\_accuracy: 0.7342 - mse: 0.1630 - val\_loss: 0.4569 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1515 Epoch 4/20

154/154 - 0s - loss: 0.4851 - binary\_accuracy: 0.7342 - mse: 0.1603 154/154 - 0s - loss: 0.4851 - binary\_accuracy: 0.7342 - mse: 0.1603 - val\_loss: 0.4466 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1486 Epoch 5/20

154/154 - 0s - loss: 0.4779 - binary\_accuracy: 0.7342 - mse: 0.1582 154/154 - 0s - loss: 0.4779 - binary\_accuracy: 0.7342 - mse: 0.1582 - val\_loss: 0.4409 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1465 Epoch 6/20

154/154 - 0s - loss: 0.4705 - binary\_accuracy: 0.7342 - mse: 0.1557 154/154 - 0s - loss: 0.4705 - binary\_accuracy: 0.7342 - mse: 0.1557 - val\_loss: 0.4368 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1452 Epoch 7/20

154/154 - 0s - loss: 0.4721 - binary\_accuracy: 0.7342 - mse: 0.1553 154/154 - 0s - loss: 0.4721 - binary\_accuracy: 0.7342 - mse: 0.1553 - val\_loss: 0.4334 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1438 Epoch 8/20

154/154 - 0s - loss: 0.4609 - binary\_accuracy: 0.7342 - mse: 0.1535 154/154 - 0s - loss: 0.4609 - binary\_accuracy: 0.7342 - mse: 0.1535 - val\_loss: 0.4306 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1429 Epoch 9/20

154/154 - 0s - loss: 0.4674 - binary\_accuracy: 0.7342 - mse: 0.1540 154/154 - 0s - loss: 0.4674 - binary\_accuracy: 0.7342 - mse: 0.1540 - val\_loss: 0.4298 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1426 Epoch 10/20

154/154 - 0s - loss: 0.4671 - binary\_accuracy: 0.7342 - mse: 0.1540 154/154 - 0s - loss: 0.4671 - binary\_accuracy: 0.7342 - mse: 0.1540 - val\_loss: 0.4286 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1422 Epoch 11/20

154/154 - 0s - loss: 0.4623 - binary\_accuracy: 0.7342 - mse: 0.1524 154/154 - 0s - loss: 0.4623 - binary\_accuracy: 0.7342 - mse: 0.1524 - val\_loss: 0.4281 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1420 Epoch 12/20

154/154 - 0s - loss: 0.4631 - binary\_accuracy: 0.7342 - mse: 0.1533 154/154 - 0s - loss: 0.4631 - binary\_accuracy: 0.7342 - mse: 0.1533 - val\_loss: 0.4278 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1419 Epoch 13/20

154/154 - 0s - loss: 0.4609 - binary\_accuracy: 0.7342 - mse: 0.1520 154/154 - 0s - loss: 0.4609 - binary\_accuracy: 0.7342 - mse: 0.1520 - val\_loss: 0.4267 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1414 Epoch 14/20

154/154 - 0s - loss: 0.4636 - binary\_accuracy: 0.7342 - mse: 0.1529 154/154 - 0s - loss: 0.4636 - binary\_accuracy: 0.7342 - mse: 0.1529 -

154/154 - 0s - loss: 0.4603 - binary\_accuracy: 0.7342 - mse: 0.1510 154/154 - 0s - loss: 0.4603 - binary\_accuracy: 0.7342 - mse: 0.1510 - val\_loss: 0.4264 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1414 Epoch 16/20

154/154 - 0s - loss: 0.4611 - binary\_accuracy: 0.7342 - mse: 0.1519 154/154 - 0s - loss: 0.4611 - binary\_accuracy: 0.7342 - mse: 0.1519 - val\_loss: 0.4258 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1410 Epoch 17/20

154/154 - 0s - loss: 0.4638 - binary\_accuracy: 0.7342 - mse: 0.1522 154/154 - 0s - loss: 0.4638 - binary\_accuracy: 0.7342 - mse: 0.1522 - val\_loss: 0.4261 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1411 Epoch 18/20

154/154 - 0s - loss: 0.4571 - binary\_accuracy: 0.7342 - mse: 0.1507 154/154 - 0s - loss: 0.4571 - binary\_accuracy: 0.7342 - mse: 0.1507 - val\_loss: 0.4259 - val\_binary\_accuracy: 0.7343 - val\_mse: 0.1410 Epoch 19/20

154/154 - 0s - loss: 0.4603 - binary\_accuracy: 0.7498 - mse: 0.1513 154/154 - 0s - loss: 0.4603 - binary\_accuracy: 0.7498 - mse: 0.1513 - val\_loss: 0.4256 - val\_binary\_accuracy: 0.7875 - val\_mse: 0.1408 Epoch 20/20

154/154 - 0s - loss: 0.4606 - binary\_accuracy: 0.7738 - mse: 0.1512 154/154 - 0s - loss: 0.4606 - binary\_accuracy: 0.7738 - mse: 0.1512 - val\_loss: 0.4258 - val\_binary\_accuracy: 0.7856 - val\_mse: 0.1409

# Evaluation

Predict classes and probabilities

pred\_classes\_test <- predict\_classes(object = model\_keras, x = as.matrix(test\_data\_bk))

pred\_proba\_test <- predict\_proba(object = model\_keras, x = as.matrix(test\_data\_bk))

Create results table

test\_results <- tibble(

actual\_yes = as.factor(as.vector(test\_y\_drop)), pred\_classes\_test = as.factor(as.vector(pred\_classes\_test)), Yes = as.vector(pred\_proba\_test),

No = 1 - as.vector(pred\_proba\_test)) head(test\_results)

## # A tibble: 6 x 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ##  ## | actual\_yes | pred\_classes\_test | Yes | No |
| ## | 1 0 | 0 | 0.481 | 0.519 |
| ## | 2 0 | 0 | 0.0256 | 0.974 |
| ## | 3 0 | 0 | 0.383 | 0.617 |
| ## | 4 1 | 1 | 0.502 | 0.498 |
| ## | 5 0 | 0 | 0.0527 | 0.947 |
| ## | 6 1 | 1 | 0.502 | 0.498 |

Calculate confusion matrix

test\_results %>%

conf\_mat(actual\_yes, pred\_classes\_test)

|  |  |  |
| --- | --- | --- |
| ##  ## | Truth  Prediction 0 | 1 |
| ## | 0 733 | 164 |
| ## | 1 41 | 116 |

Calculate metrics

test\_results %>%

metrics(actual\_yes, pred\_classes\_test) ## # A tibble: 2 x 3

## .metric .estimator .estimate ##

## 1 accuracy binary 0.806

## 2 kap binary 0.420

Area under the ROC curve

test\_results %>% roc\_auc(actual\_yes, Yes)

## # A tibble: 1 x 3

## .metric .estimator .estimate ##

## 1 roc\_auc binary 0.150

Precision and recall

tibble(

precision = test\_results %>% yardstick::precision(actual\_yes, pred\_classes\_test) %>% select(.estimate) %>% as.numeric(),

recall = test\_results %>% yardstick::recall(actual\_yes, pred\_classes\_test) %>% select(.estimate) %>% as.numeric()

)

## # A tibble: 1 x 2 ## precision recall ##

## 1 0.817 0.947

F1-Statistic

test\_results %>% yardstick::f\_meas(actual\_yes, pred\_classes\_test, beta

= 1)

## # A tibble: 1 x 3

## .metric .estimator .estimate ##

## 1 f\_meas binary 0.877

# H2O

Shows an alternative to Keras!

Initialise H2O instance and convert data to h2o frame

library(h2o)

h2o.init(nthreads = -1) ## Connection successful! ##

## R is connected to the H2O cluster:

## H2O cluster uptime: 2 hours 34 minutes ## H2O cluster timezone: Europe/Berlin

## H2O data parsing timezone: UTC

## H2O cluster version: 3.32.0.1

## H2O cluster version age: 5 months and 9 days !!! ## H2O cluster name: H2O\_started\_from\_R\_ shiringlander\_jfz275

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## | H2O | cluster | total | nodes: | 1 |
| ## | H2O | cluster | total | memory: | 6.25 GB |
| ## | H2O | cluster | total | cores: | 16 |

## H2O cluster allowed cores: 16

## H2O cluster healthy: TRUE

## H2O Connection ip: localhost

## H2O Connection port: 54321

## H2O Connection proxy: NA

## H2O Internal Security: FALSE

## H2O API Extensions: Amazon S3, XGBoost, Algos, AutoML,

Core V3, TargetEncoder, Core V4

## R Version: R version 4.0.4 (2021-02-15) h2o.no\_progress()

train\_hf <- as.h2o(train\_data) valid\_hf <- as.h2o(valid\_data) test\_hf <- as.h2o(test\_data)

response <- "Churn"

features <- setdiff(colnames(train\_hf), response)

# For binary classification, response should be a factor train\_hf[, response] <- as.factor(train\_hf[, response]) valid\_hf[, response] <- as.factor(valid\_hf[, response]) test\_hf[, response] <- as.factor(test\_hf[, response]) summary(train\_hf$Churn, exact\_quantiles = TRUE)

## Churn

## 0:3615

## 1:1309

summary(valid\_hf$Churn, exact\_quantiles = TRUE) ## Churn

## 0:774

## 1:280

summary(test\_hf$Churn, exact\_quantiles = TRUE) ## Churn

## 0:774

## 1:280

Train model with AutoML.

“During model training, you might find that the majority of your data belongs in a single class. For example, consider a binary classification model that has 100 rows,

with 80 rows labeled as class 1 and the remaining 20 rows labeled as class 2. This is a common scenario, given that machine learning attempts to predict class 1 with the highest accuracy. It can also be an example of an imbalanced dataset, in this case, with a ratio of 4:1. The balance\_classes option can be used to balance the class distribution. When enabled, H2O will either undersample the majority classes or oversample the minority classes. Note that the resulting model will also correct the final probabilities (“undo the sampling”) using a monotonic transform, so the predicted probabilities of the first model will differ from a second model. However, because AUC only cares about ordering, it won’t be affected. If this option is enabled, then you can also specify a value for the class\_sampling\_factors and max\_after\_balance\_size options." <http://docs.h2o.ai/h2o/latest-stable/h2o-> docs/data-science/algo-params/balance\_classes.html

aml <- h2o.automl(x = features,

y = response, training\_frame = train\_hf,

validation\_frame = valid\_hf, balance\_classes = TRUE, max\_runtime\_secs = 3600)

# View the AutoML Leaderboard lb <- aml@leaderboard

best\_model <- aml@leader

h2o.saveModel(best\_model, "/Users/shiringlander/Documents/Github/Data")

Prediction

pred <- h2o.predict(best\_model, test\_hf[, -1])

Mean per class error

h2o.mean\_per\_class\_error(best\_model, train = TRUE, valid = TRUE, xval = TRUE)

## train valid xval ## 0.2125416 0.2268365 0.2387021

Confusion matrix on validation data

h2o.confusionMatrix(best\_model, valid = TRUE)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.336658768072945:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | 0 | 1 | Error | Rate |
| ## | 0 | 644 | 130 | 0.167959 | =130/774 |
| ## | 1 | 80 | 200 | 0.285714 | =80/280 |
| ## | Totals | 724 | 330 | 0.199241 | =210/1054 |

h2o.auc(best\_model, train = TRUE) ## [1] 0.8707153

h2o.auc(best\_model, valid = TRUE) ## [1] 0.8531331

h2o.auc(best\_model, xval = TRUE) ## [1] 0.8447738

Performance and confusion matrix on test data

perf <- h2o.performance(best\_model, test\_hf) h2o.confusionMatrix(perf)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.362814272627094:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | 0 | 1 | Error | Rate |
| ## | 0 | 642 | 132 | 0.170543 | =132/774 |
| ## | 1 | 76 | 204 | 0.271429 | =76/280 |
| ## | Totals | 718 | 336 | 0.197343 | =208/1054 |

Plot performance

plot(perf)

More performance metrics extracted

h2o.logloss(perf) ## [1] 0.3997853

h2o.mse(perf) ## [1] 0.129026

h2o.auc(perf)

## [1] 0.8608204

metrics <- as.data.frame(h2o.metric(perf ) head(metrics)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| f1 | f2 | f0point5 | accuracy | precision |
| 0.007117438 | 0.004460303 | 0.01760563 | 0.7352941 | 1.0000000 |
| 0.014184397 | 0.008912656 | 0.03472222 | 0.7362429 | 1.0000000 |
| 0.021201413 | 0.013357079 | 0.05136986 | 0.7371917 | 1.0000000 |
| 0.028169014 | 0.017793594 | 0.06756757 | 0.7381404 | 1.0000000 |
| 0.035087719 | 0.022222222 | 0.08333333 | 0.7390892 | 1.0000000 |
| 0.034965035 | 0.022202487 | 0.08223684 | 0.7381404 | 0.8333333 |

## threshold recall

## 1 0.8798453

0.003571429

## 2 0.8719808

0.007142857

## 3 0.8698321

0.010714286

## 4 0.8658536

0.014285714

## 5 0.8600962

0.017857143

## 6 0.8515710

0.017857143

## specificity absolute\_mcc min\_per\_class\_accuracy mean\_per\_class\_accuracy tns

## 1 1.000000 0.05123624 0.003571429

0.5017857 774

## 2 1.000000 0.07249342 0.007142857

0.5035714 774

## 3 1.000000 0.08882817 0.010714286

0.5053571 774

## 4 1.000000 0.10261877 0.014285714

0.5071429 774

## 5 1.000000 0.11478595 0.017857143

0.5089286 774

## 6 0.998708 0.09724979 0.017857143

0.5082826 773

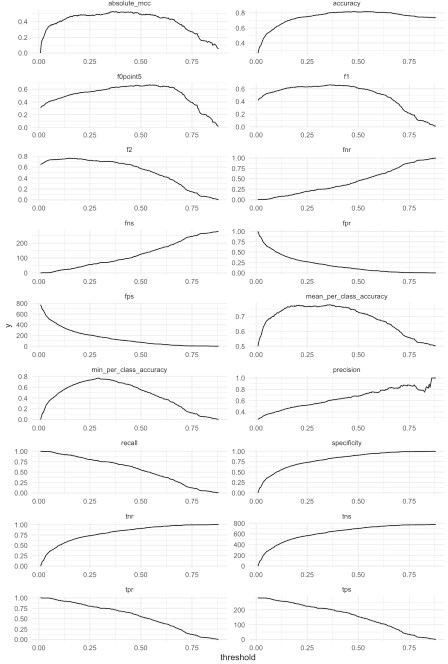
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | fns | fps | tps | tnr | fnr | fpr | tpr | idx |
| ## | 1 279 | 0 | 1 | 1.000000 | 0.9964286 | 0.00000000 | 0.003571429 | 0 |
| ## | 2 278 | 0 | 2 | 1.000000 | 0.9928571 | 0.00000000 | 0.007142857 | 1 |
| ## | 3 277 | 0 | 3 | 1.000000 | 0.9892857 | 0.00000000 | 0.010714286 | 2 |
| ## | 4 276 | 0 | 4 | 1.000000 | 0.9857143 | 0.00000000 | 0.014285714 | 3 |
| ## | 5 275 | 0 | 5 | 1.000000 | 0.9821429 | 0.00000000 | 0.017857143 | 4 |
| ## | 6 275 | 1 | 5 | 0.998708 | 0.9821429 | 0.00129199 | 0.017857143 | 5 |

Plot performance metrics

metrics %>

gather(x, y, f1:tpr) %>%

ggplot(aes(x = threshold, y = y, group = x)) + facet\_wrap(~ x, ncol = 2, scales = "free") + geom\_line()



Examine prediction thresholds

# optimal threshold:

threshold <- metrics[order(-metrics$accuracy), "threshold"][1] print(threshold)

## [1] 0.5436611

finalRf\_predictions <- data.frame(actual = as.vector(test\_hf$Churn),

as.data.frame(h2o.predict(object =

best\_model,

test\_hf))) head(finalRf\_predictions)

newdata =

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | actual | predict | p0 | p1 |
| ## | 1 0 | 1 | 0.5300039 | 0.46999608 |
| ## | 2 0 | 0 | 0.9695586 | 0.03044143 |
| ## | 3 0 | 1 | 0.4360508 | 0.56394924 |
| ## | 4 1 | 1 | 0.4275426 | 0.57245736 |
| ## | 5 0 | 0 | 0.9369931 | 0.06300689 |
| ## | 6 1 | 1 | 0.3377812 | 0.66221884 |

finalRf\_predictions$accurate <- ifelse(finalRf\_predictions$actual ==

finalRf\_predictions$predict,

"ja", "nein")

finalRf\_predictions$predict\_stringent <- ifelse(finalRf\_predictions$p1

> threshold, 1,

ifelse(finalRf\_predictions$p0 > threshold, 0, "unsicher")) finalRf\_predictions$accurate\_stringent <- ifelse(finalRf\_predictions$ actual ==

finalRf\_predictions$predict\_stringent, "ja",

ifelse(finalRf\_predictions$

predict\_stringent == "nein"))

finalRf\_predictions %>% group\_by(actual, predict) %>% dplyr::summarise(n = n())

## # A tibble: 4 x 3

## # Groups: actual [2] ## actual predict n ##

## 1 0 0 620

## 2 0 1 154

## 3 1 0 71

## 4 1 1 209

finalRf\_predictions %>%

group\_by(actual, predict\_stringent) %>% dplyr::summarise(n = n())

## # A tibble: 6 x 3

## # Groups: actual [2]

## actual predict\_stringent n ##

"unsicher", "unsicher",

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | 1 | 0 | 0 | 681 |
| ## | 2 | 0 | 1 | 51 |
| ## | 3 | 0 | unsicher | 42 |
| ## | 4 | 1 | 0 | 103 |
| ## | 5 | 1 | 1 | 138 |
| ## | 6 | 1 | unsicher | 39 |

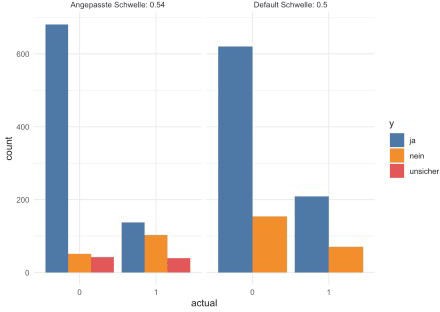
finalRf\_predictions %>%

gather(x, y, accurate, accurate\_stringent) %>%

mutate(x = ifelse(x == "accurate", "Default Schwelle: 0.5",

paste("Angepasste Schwelle:", round(threshold,

digits = 2)))) %>%

ggplot(aes(x = actual, fill = y)) + facet\_grid(~ x) + geom\_bar(position = "dodge") + scale\_fill\_tableau()

df <- finalRf\_predictions[, c(1, 3, 4)] thresholds <- seq(from = 0, to = 1, by = 0.1) prop\_table <- data.frame(threshold = thresholds,

prop\_p0\_correct\_pred = NA, prop\_p0\_wrong\_pred

= NA,

= NA)

prop\_p1\_correct\_pred = NA, prop\_p1\_wrong\_pred

for (threshold in thresholds) {

# if prediction probability for churn (1) > threshold, predict 1, else 0 (no churn)

pred\_1 <- ifelse(df$p1 > threshold, 1, 0) # if prediction == actual, TRUE

pred\_1\_t <- ifelse(pred\_1 == df$actual, TRUE, FALSE) group <- data.frame(df,

"pred\_true" = pred\_1\_t) %>% group\_by(actual, pred\_true) %>% dplyr::summarise(n = n())

# actual no churns

group\_p0 <- filter(group, actual == "0")

prop\_p0\_t <- sum(filter(group\_p0, pred\_true == TRUE)$n) / sum(group\_p0$n)

prop\_p0\_f <- sum(filter(group\_p0, pred\_true == FALSE)$n) / sum(group\_p0$n)

prop\_table[prop\_table$threshold == threshold, "prop\_p0\_correct\_pred"]

<- prop\_p0\_t

prop\_table[prop\_table$threshold == threshold, "prop\_p0\_wrong\_pred"]

<- prop\_p0\_f

# actual churns

group\_p1 <- filter(group, actual == "1")

prop\_p1\_t <- sum(filter(group\_p1, pred\_true == TRUE)$n) / sum(group\_p1$n)

prop\_p1\_f <- sum(filter(group\_p1, pred\_true == FALSE)$n) / sum(group\_p1$n)

prop\_table[prop\_table$threshold == threshold, "prop\_p1\_correct\_pred"]

<- prop\_p1\_t

prop\_table[prop\_table$threshold == threshold, "prop\_p1\_wrong\_pred"]

<- prop\_p1\_f

}

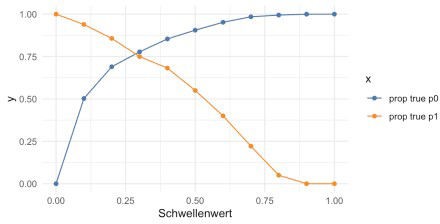
prop\_table %>%

gather(x, y, prop\_p0\_correct\_pred, prop\_p1\_correct\_pred) %>% rename(Schwellenwert = threshold) %>%

mutate(x = ifelse(x == "prop\_p0\_correct\_pred", "prop true p0", "prop true p1")) %>%

ggplot(aes(x = Schwellenwert, y = y, color = x)) + geom\_point() +

geom\_line() + scale\_color\_tableau()



## Cost/revenue calculation per year

Let’s assume that

1. a marketing campaign + employee time will cost the company 1000€ per year for every customer that is included in the campaign.
2. the annual average revenue per customer is 2000€ (in more complex scenarios customers could be further divided into revenue groups to calculate how “valuable” they are and how harmful loosing them would be)
3. investing into unnecessary marketing doesn’t cause churn by itself (i.e. a customer who isn’t going to churn isn’t reacting negatively to the add campaign - which could happen in more complex scenarios).
4. without a customer churn model the company would target half of their customer (by chance) for ad-campaigns
5. without a customer churn model the company would lose about 25% of their customers to churn

This would mean that compared to no intervention we would have

prop\_p0\_correct\_pred == customers who were correctly predicted to not churn did not cost anything (no marketing money was spent): +/-0€

prop\_p0\_wrong\_pred == customers that did not churn but were predicted to churn will be an empty investment: +/-0€ - 1500€

prop\_p1\_wrong\_pred == customer that were predicted to stay but churned: -2000€ prop\_p1\_correct\_pred == customers that were correctly predicted to churn:

let’s say 100% of those could be kept by investing into marketing: +2000€ -1500€ let’s say 50% could be kept by investing into marketing: +2000€ \* 0.5 -1500€

Let’s play around with some values:

# Baseline revenue <- 2000

cost <- 1000

## number of customers who churn

customers\_churn <- filter(test\_data, Churn == 1) customers\_churn\_n <- nrow(customers\_churn)

## number of customers who don't churn

customers\_no\_churn <- filter(filter(test\_data, Churn == 0)) customers\_no\_churn\_n <- nrow(customers\_no\_churn)

## number of customers

customers <- customers\_churn\_n + customers\_no\_churn\_n

# percentage of customers randomly targeted for ad campaign ad\_target\_rate <- 0.5

ad\_cost\_default <- customers \* ad\_target\_rate \* cost

churn\_rate\_default <- customers\_churn\_n / customers\_no\_churn\_n ann\_revenue\_default <- customers\_no\_churn\_n \* revenue

# net win per year: revenue from non-churn customers - ad costs

net\_win\_default <- ann\_revenue\_default - ad\_cost\_default net\_win\_default

## [1] 1021000

How much revenue can we gain from predicting customer churn with our model (assuming a conversion rate after ad campaign of 0.7):

# all customers predicted to churn will be targeted by ad campaign # of those, 70% can be convinced to stay

conversion <- 0.7

net\_win\_table <- prop\_table %>% mutate(

# proportion of correctly predicted no-churns: make normal revenue at no cost

prop\_p0\_correct\_pred\_X = prop\_p0\_correct\_pred \* customers\_no\_churn\_n \* revenue,

# proportion of no-churns predicted to churn: make revenue but also cost ad money

prop\_p0\_wrong\_pred\_X = prop\_p0\_wrong\_pred \* customers\_no\_churn\_n \* (revenue - cost),

# proportion of churns predicted to not churn: revenue lost prop\_p1\_wrong\_pred\_X = prop\_p1\_wrong\_pred \* customers\_churn\_n \* 0, # proportion of correctly predicted churns: 70% stay and make

revenue but all of them cost ad money

prop\_p1\_correct\_pred\_X = prop\_p1\_correct\_pred \* customers\_churn\_n \* ((revenue \* conversion) - cost)) %>%

group\_by(threshold) %>%

summarise(net\_win = sum(prop\_p0\_correct\_pred\_X + prop\_p0\_wrong\_pred\_X

+ prop\_p1\_wrong\_pred\_X + prop\_p1\_correct\_pred\_X), net\_win\_compared = net\_win - net\_win\_default) %>%

arrange(-net\_win\_compared)

net\_win\_table

## # A tibble: 11 x 3

## threshold net\_win net\_win\_compared

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## |  | | | |
| ## | 1 | 0.7 | 1560800 | 539800 |
| ## | 2 | 0.6 | 1555800 | 534800 |
| ## | 3 | 0.8 | 1549600 | 528600 |
| ## | 4 | 0.9 | 1548000 | 527000 |
| ## | 5 | 1 | 1548000 | 527000 |
| ## | 6 | 0.5 | 1536600 | 515600 |
| ## | 7 | 0.4 | 1511400 | 490400 |
| ## | 8 | 0.3 | 1460000 | 439000 |
| ## | 9 | 0.2 | 1404000 | 383000 |
| ## | 10 | 0.1 | 1268200 | 247200 |
| ## | 11 | 0 | 886000 | -135000 |

# LIME

Explaining predictions

Xtrain <- as.data.frame(train\_hf) Xtest <- as.data.frame(test\_hf)

# run lime() on training set explainer <- lime::lime(x = Xtrain,

model = best\_model)

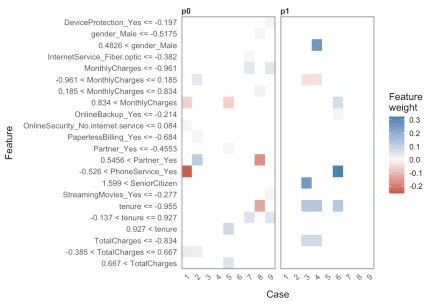
# run explain() on the explainer

explanation <- lime::explain(x = Xtest[1:9, ],

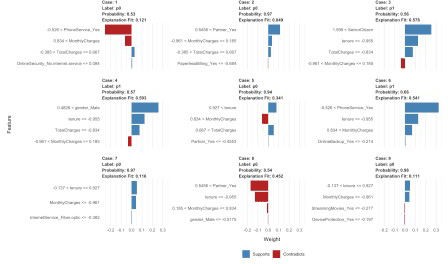
explainer = explainer, n\_labels = 1,

n\_features = 4,

kernel\_width = 0.5) plot\_explanations(explanation)



explanation %>% plot\_features(ncol = 3)



sessionInfo()

## R version 4.0.4 (2021-02-15)

## Platform: x86\_64-apple-darwin17.0 (64-bit) ## Running under: macOS Catalina 10.15.7

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib

/libRblas.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib

/libRlapack.dylib ##

## locale:

## [1] en\_US.UTF-8/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8 ##

## attached base packages:

## [1] stats graphics grDevices utils datasets methods base ##

## other attached packages:

## [1] h2o\_3.32.0.1 corrplot\_0.84 ggthemes\_4.2.4 yardstick\_0.0.7

## [5] recipes\_0.1.15 rsample\_0.0.9 lime\_0.5.2 keras\_2.3.0.0.9000

## [9] mice\_3.13.0 caret\_6.0-86 lattice\_0.20-41 forcats\_0.5.1

## [13] stringr\_1.4.0 dplyr\_1.0.5 purrr\_0.3.4 readr\_1.4.0

## [17] tidyr\_1.1.3 tibble\_3.1.0 ggplot2\_3.3.3 tidyverse\_1.3.0

##

## loaded via a namespace (and not attached):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | [1] | colorspace\_2.0-0 | ellipsis\_0.3.1 | class\_7.3-18 |
| ## | [4] | base64enc\_0.1-3 | fs\_1.5.0 | rstudioapi\_0.13 |
| ## | [7] | listenv\_0.8.0 | furrr\_0.2.2 | farver\_2.1.0 |
| ## | [10] | bit64\_4.0.5 | prodlim\_2019.11.13 | fansi\_0.4.2 |
| ## | [13] | lubridate\_1.7.10 | xml2\_1.3.2 | codetools\_0.2-18 |
| ## | [16] | splines\_4.0.4 | knitr\_1.31 | zeallot\_0.1.0 |
| ## | [19] | jsonlite\_1.7.2 | pROC\_1.17.0.1 | broom\_0.7.5 |
| ## | [22] | dbplyr\_2.1.0 | tfruns\_1.5.0 | compiler\_4.0.4 |
| ## | [25] | httr\_1.4.2 | backports\_1.2.1 | assertthat\_0.2.1 |
| ## | [28] | Matrix\_1.3-2 | cli\_2.3.1 | htmltools\_0.5.1.1 |
| ## | [31] | tools\_4.0.4 | gtable\_0.3.0 | glue\_1.4.2 |
| ## | [34] | reshape2\_1.4.4 | Rcpp\_1.0.6 | cellranger\_1.1.0 |
| ## | [37] | jquerylib\_0.1.3 | vctrs\_0.3.6 | nlme\_3.1-152 |
| ## | [40] | blogdown\_1.2 | iterators\_1.0.13 | timeDate\_3043.102 |
| ## | [43] | gower\_0.2.2 | xfun\_0.22 | globals\_0.14.0 |
| ## | [46] | rvest\_1.0.0 | lifecycle\_1.0.0 | future\_1.21.0 |
| ## | [49] | MASS\_7.3-53.1 | scales\_1.1.1 | ipred\_0.9-11 |
| ## | [52] | hms\_1.0.0 | parallel\_4.0.4 | yaml\_2.2.1 |
| ## | [55] | reticulate\_1.18 | sass\_0.3.1 | rpart\_4.1-15 |
| ## | [58] | stringi\_1.5.3 | highr\_0.8 | tensorflow\_2.2.0 |
| ## | [61] | foreach\_1.5.1 | lava\_1.6.9 | shape\_1.4.5 |
| ## | [64] | bitops\_1.0-6 | rlang\_0.4.10 | pkgconfig\_2.0.3 |
| ## | [67] | evaluate\_0.14 | labeling\_0.4.2 | bit\_4.0.4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | [70] | tidyselect\_1.1.0 | parallelly\_1.23.0 | plyr\_1.8.6 |
| ## | [73] | magrittr\_2.0.1 | bookdown\_0.21 | R6\_2.5.0 |
| ## | [76] | generics\_0.1.0 | DBI\_1.1.1 | pillar\_1.5.1 |
| ## | [79] | haven\_2.3.1 | whisker\_0.4 | withr\_2.4.1 |
| ## | [82] | RCurl\_1.98-1.2 | survival\_3.2-7 | nnet\_7.3-15 |
| ## | [85] | modelr\_0.1.8 | crayon\_1.4.1 | utf8\_1.2.1 |
| ## | [88] | rmarkdown\_2.7 | grid\_4.0.4 | readxl\_1.3.1 |
| ## | [91] | data.table\_1.14.0 | ModelMetrics\_1.2.2.2 | reprex\_1.0.0 |
| ## | [94] | digest\_0.6.27 | stats4\_4.0.4 | munsell\_0.5.0 |
| ## | [97] | glmnet\_4.1-1 | bslib\_0.2.4 |  |