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.

2. 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).

Inspiration & Sources

Thank you to the following people for providing excellent code examples about customer churn:

- Matt Dancho: http://www.business-science.io/business/2017/11/28/customer_churn_analysis_keras.html
- JJ Allaire: https://github.com/rstudio/keras-customer-churn
- Susan Li: https://towardsdatascience.com/predict-customer-churn-with-r-9e62357d47b4
- John Sullivan: https://jtsulliv.github.io/churn-eda/

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

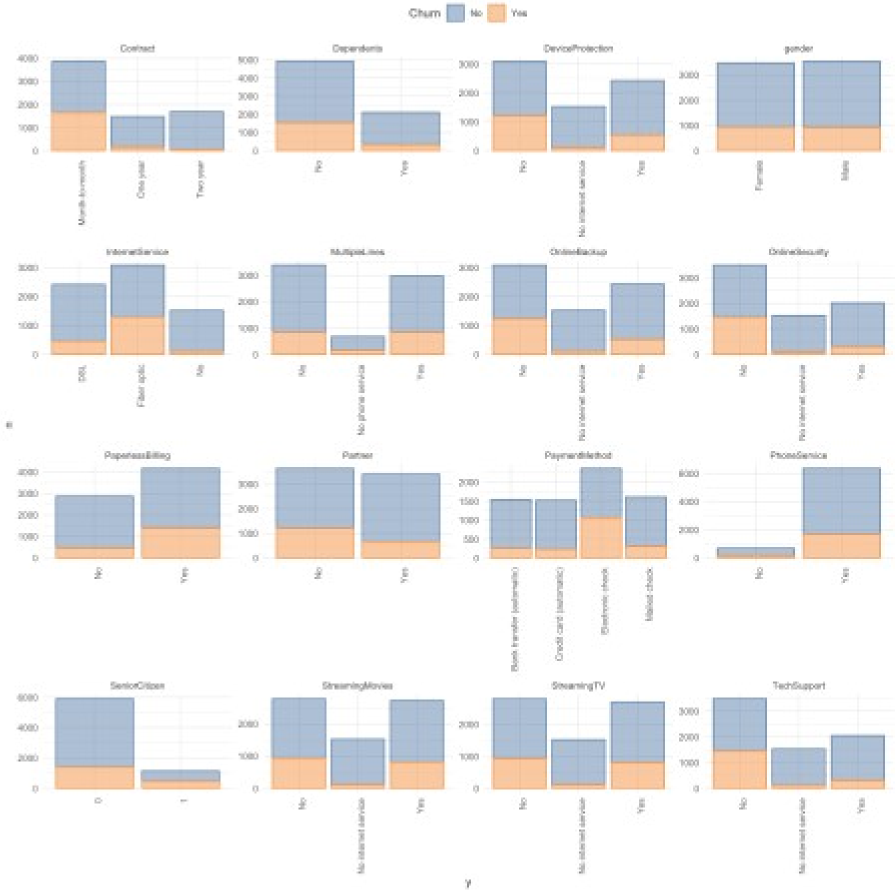
- Proportion of churn (customers who left within the last month):

```
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):

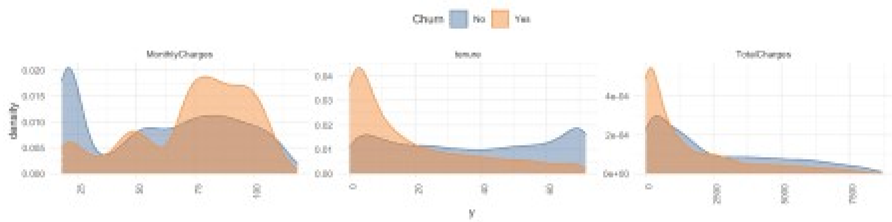
```
churn_data_raw %>%
  mutate(SeniorCitizen = as.character(SeniorCitizen)) %>%
  select(-customerID) %>%
  select_if(is.character) %>%
  select(Churn, everything()) %>%
  gather(x, y, gender:PaymentMethod) %>%
  count(Churn, x, y) %>%
  ggplot(aes(x = y, y = n, fill = Churn, color = Churn)) +
```

```
    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 StreamingMovies Contract PaperlessBilling
PaymentMethod
## 7032           1               1        1                1
1
## 11             1               1        1                1
1
##                0               0        0                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)
##      Yes
## [1,]   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  0.0678         -0.262       -0.173       0.983
-0.956
## 3        -0.439 -1.24           -0.366       -0.965       0.983
-0.956
## 4        -0.439  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 -
val_loss: 0.6291 - val_binary_accuracy: 0.7343 - val_mse: 0.2183
```

```
Epoch 2/20
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 -
val_loss: 0.4262 - val_binary_accuracy: 0.7343 - val_mse: 0.1412
```

```
Epoch 15/20
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)
```

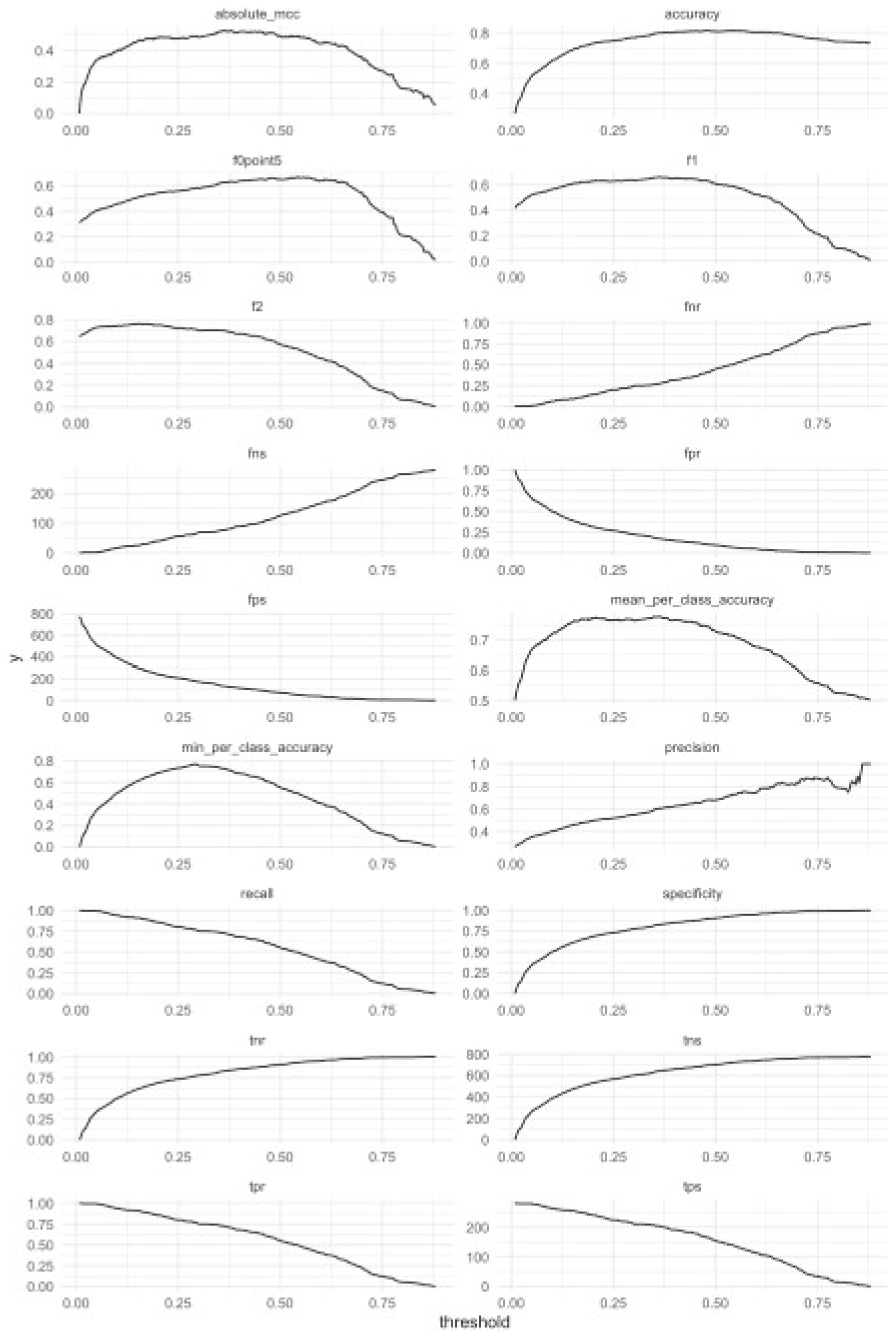
- More performance metrics extracted

```
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)
##   threshold          f1          f2   f0point5  accuracy precision
recall
## 1 0.8798453 0.007117438 0.004460303 0.01760563 0.7352941 1.0000000
0.003571429
## 2 0.8719808 0.014184397 0.008912656 0.03472222 0.7362429 1.0000000
0.007142857
## 3 0.8698321 0.021201413 0.013357079 0.05136986 0.7371917 1.0000000
0.010714286
## 4 0.8658536 0.028169014 0.017793594 0.06756757 0.7381404 1.0000000
0.014285714
## 5 0.8600962 0.035087719 0.022222222 0.08333333 0.7390892 1.0000000
0.017857143
## 6 0.8515710 0.034965035 0.022202487 0.08223684 0.7381404 0.8333333
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

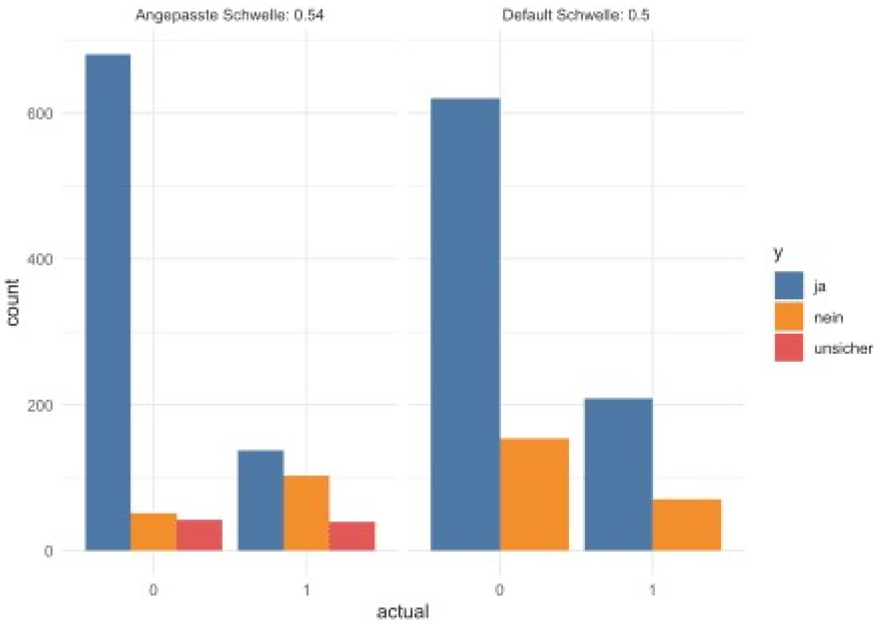
```
metrics %>%
  gather(x, y, f1:tpr) %>%
  ggplot(aes(x = threshold, y = y, group = x)) +
    facet_wrap(~ x, ncol = 2, scales = "free") +
    geom_line()
```



- Examine prediction thresholds

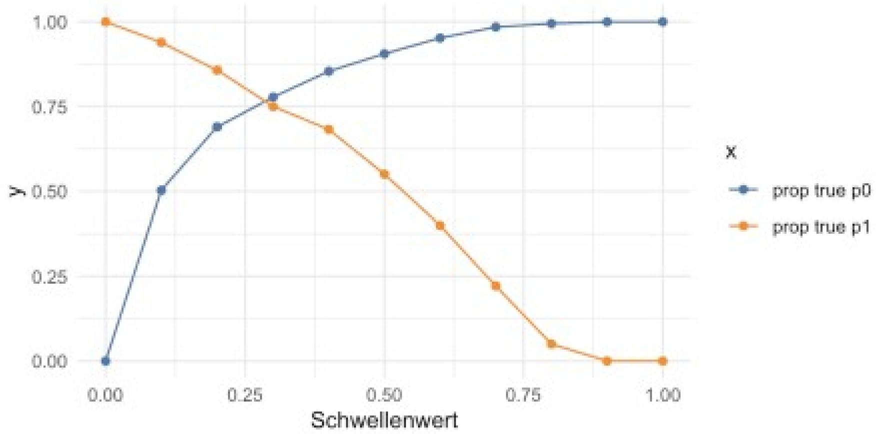
```
# optimal threshold:
threshold <- metrics[order(-metrics$accuracy), "threshold"][1]
print(threshold)
## [1] 0.5436611
finalRf_predictions <- data.frame(actual = as.vector(test_hf$Churn),
                                  as.data.frame(h2o.predict(object =
best_model,
                                                             newdata =
test_hf)))
head(finalRf_predictions)
##   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 ==
                                                "unsicher", "unsicher",
"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
##
```

```
## 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,
                         prop_p1_correct_pred = NA, prop_p1_wrong_pred
= NA)

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
```

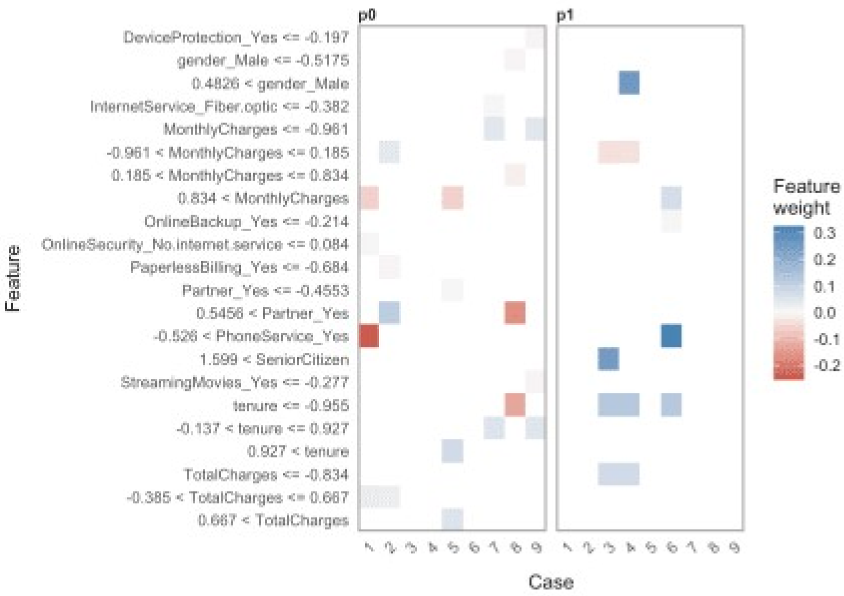
- How much revenue can we gain from predicting customer churn with our model (assuming a conversion rate after ad campaign of 0.7):

```
# 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
```

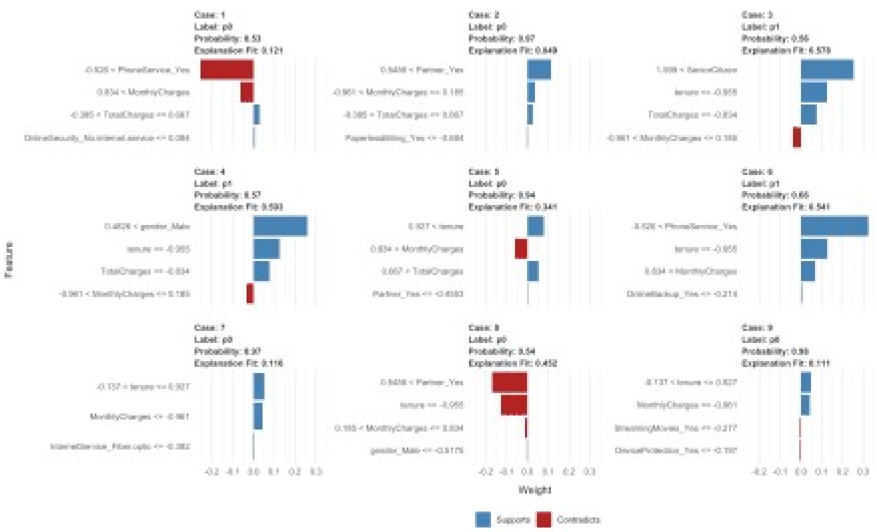
LIME

- 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
```