**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**

The Telco Customer Churn data set is the same one that Matt Dancho used in his post (see above). It was downloaded from [IBM Watson](https://www.ibm.com/communities/analytics/watson-analytics-blog/predictive-insights-in-the-telco-customer-churn-data-set/)

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

glimpse(churn\_data\_raw)

## Observations: 7,043

## Variables: 21

## $ customerID "7590-VHVEG", "5575-GNVDE", "3668-QPYBK", "77...

## $ gender "Female", "Male", "Male", "Male", "Female", "...

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

## $ Partner "Yes", "No", "No", "No", "No", "No", "No", "N...

## $ Dependents "No", "No", "No", "No", "No", "No", "Yes", "N...

## $ tenure 1, 34, 2, 45, 2, 8, 22, 10, 28, 62, 13, 16, 5...

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

## $ MultipleLines "No phone service", "No", "No", "No phone ser...

## $ InternetService "DSL", "DSL", "DSL", "DSL", "Fiber optic", "F...

## $ OnlineSecurity "No", "Yes", "Yes", "Yes", "No", "No", "No", ...

## $ OnlineBackup "Yes", "No", "Yes", "No", "No", "No", "Yes", ...

## $ DeviceProtection "No", "Yes", "No", "Yes", "No", "Yes", "No", ...

## $ TechSupport "No", "No", "No", "Yes", "No", "No", "No", "N...

## $ StreamingTV "No", "No", "No", "No", "No", "Yes", "Yes", "...

## $ StreamingMovies "No", "No", "No", "No", "No", "Yes", "No", "N...

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

## $ PaperlessBilling "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes"...

## $ PaymentMethod "Electronic check", "Mailed check", "Mailed c...

## $ MonthlyCharges 29.85, 56.95, 53.85, 42.30, 70.70, 99.65, 89....

## $ TotalCharges 29.85, 1889.50, 108.15, 1840.75, 151.65, 820....

## $ Churn "No", "No", "Yes", "No", "Yes", "Yes", "No", ...

**EDA**

* Proportion of churn

churn\_data\_raw %>%

count(Churn)

## # A tibble: 2 x 2

## Churn n

##

## 1 No 5174

## 2 Yes 1869

* Plot categorical features

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") +

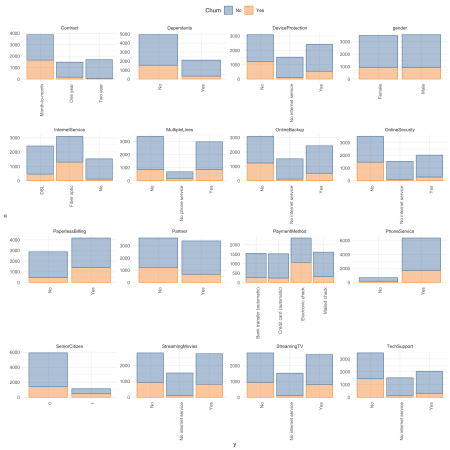
geom\_bar(stat = "identity", alpha = 0.5) +

theme(axis.text.x = element\_text(angle = 90, hjust = 1),

legend.position = "top") +

scale\_color\_tableau() +

scale\_fill\_tableau()



* Plot numerical features

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") +

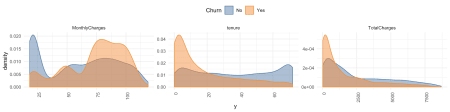
geom\_density(alpha = 0.5) +

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

## 7032 1 1 1 1 1 1

## 11 1 1 1 1 1 1

## 0 0 0 0 0 0

## MultipleLines InternetService OnlineSecurity OnlineBackup

## 7032 1 1 1 1

## 11 1 1 1 1

## 0 0 0 0

## DeviceProtection TechSupport StreamingTV StreamingMovies Contract

## 7032 1 1 1 1 1

## 11 1 1 1 1 1

## 0 0 0 0 0

## PaperlessBilling PaymentMethod MonthlyCharges Churn TotalCharges

## 7032 1 1 1 1 1 0

## 11 1 1 1 1 0 1

## 0 0 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 <- to\_categorical(as.integer(as.factor(train\_data$Churn)) - 1, 2)

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

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

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

test\_y\_drop <- 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,] 1

## [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 0.0765 -0.256 -0.163 0.979 -0.966

## 2 -0.439 -1.23 -0.359 -0.949 0.979 -0.966

## 3 -0.439 -0.983 1.17 -0.635 -1.02 -0.966

## 4 -0.439 -0.901 -1.16 -0.863 -1.02 -0.966

## 5 -0.439 -0.168 1.34 0.347 -1.02 1.03

## 6 -0.439 1.22 -0.282 0.542 0.979 -0.966

## # ... 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")

)

summary(model\_keras)

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Layer (type) Output Shape Param #

## ===========================================================================

## dense\_1 (Dense) (None, 32) 992

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dropout\_1 (Dropout) (None, 32) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dense\_2 (Dense) (None, 16) 528

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dropout\_2 (Dropout) (None, 16) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dense\_3 (Dense) (None, 8) 136

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dropout\_3 (Dropout) (None, 8) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dense\_4 (Dense) (None, 1) 9

## ===========================================================================

## Total params: 1,665

## Trainable params: 1,665

## Non-trainable params: 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 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

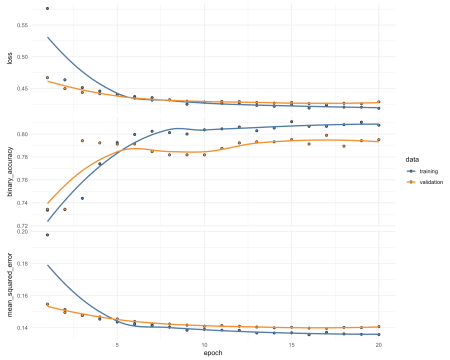
)

* Plot Keras training results

plot(fit\_keras) +

scale\_color\_tableau() +

scale\_fill\_tableau()



**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 1 0.515 0.485

## 2 0 0 0.0142 0.986

## 3 1 1 0.575 0.425

## 4 1 0 0.436 0.564

## 5 0 0 0.00362 0.996

## 6 0 0 0.0309 0.969

* Calculate confusion matrix

test\_results %>%

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

## Truth

## Prediction 0 1

## 0 690 124

## 1 84 156

* Calculate metrics

test\_results %>%

metrics(actual\_yes, pred\_classes\_test)

## # A tibble: 2 x 3

## .metric .estimator .estimate

##

## 1 accuracy binary 0.803

## 2 kap binary 0.470

* Are under the ROC curve

test\_results %>%

roc\_auc(actual\_yes, Yes)

## # A tibble: 1 x 3

## .metric .estimator .estimate

##

## 1 roc\_auc binary 0.837

* 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 0.848 0.891

* 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 0.869

**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: 1 hours 16 minutes

## H2O cluster timezone: Europe/Berlin

## H2O data parsing timezone: UTC

## H2O cluster version: 3.20.0.8

## H2O cluster version age: 2 months and 20 days

## H2O cluster name: H2O\_started\_from\_R\_shiringlander\_wcd642

## H2O cluster total nodes: 1

## H2O cluster total memory: 3.03 GB

## H2O cluster total cores: 8

## H2O cluster allowed cores: 8

## H2O cluster healthy: TRUE

## H2O Connection ip: localhost

## H2O Connection port: 54321

## H2O Connection proxy: NA

## H2O Internal Security: FALSE

## H2O API Extensions: XGBoost, Algos, AutoML, Core V3, Core V4

## R Version: R version 3.5.1 (2018-07-02)

h2o.no\_progress()

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

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

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

response <- "Churn"

# 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

“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.

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

## 0.1717911 0.2172683 0.2350682

* Confusion matrix on validation data

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

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.258586059604449:

## 0 1 Error Rate

## 0 478 130 0.213816 =130/608

## 1 49 173 0.220721 =49/222

## Totals 527 303 0.215663 =179/830

h2o.auc(best\_model, train = TRUE)

## [1] 0.9039696

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

## [1] 0.8509068

h2o.auc(best\_model, xval = TRUE)

## [1] 0.8397085

* 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 = 0.40790847476229:

## 0 1 Error Rate

## 0 667 107 0.138243 =107/774

## 1 86 194 0.307143 =86/280

## Totals 753 301 0.183112 =193/1054

* Plot performance

plot(perf)

* More performance metrics extracted

h2o.logloss(perf)

## [1] 0.4008041

h2o.mse(perf)

## [1] 0.1278505

h2o.auc(perf)

## [1] 0.8622301

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

head(metrics)

## threshold f1 f2 f0point5 accuracy precision

## 1 0.8278177 0.01418440 0.008912656 0.03472222 0.7362429 1

## 2 0.8203439 0.02816901 0.017793594 0.06756757 0.7381404 1

## 3 0.8173635 0.04195804 0.026642984 0.09868421 0.7400380 1

## 4 0.8160146 0.04878049 0.031055901 0.11363636 0.7409867 1

## 5 0.8139018 0.05555556 0.035460993 0.12820513 0.7419355 1

## 6 0.8112067 0.07560137 0.048629531 0.16975309 0.7447818 1

## recall specificity absolute\_mcc min\_per\_class\_accuracy

## 1 0.007142857 1 0.07249342 0.007142857

## 2 0.014285714 1 0.10261877 0.014285714

## 3 0.021428571 1 0.12580168 0.021428571

## 4 0.025000000 1 0.13594622 0.025000000

## 5 0.028571429 1 0.14540208 0.028571429

## 6 0.039285714 1 0.17074408 0.039285714

## mean\_per\_class\_accuracy tns fns fps tps tnr fnr fpr tpr

## 1 0.5035714 774 278 0 2 1 0.9928571 0 0.007142857

## 2 0.5071429 774 276 0 4 1 0.9857143 0 0.014285714

## 3 0.5107143 774 274 0 6 1 0.9785714 0 0.021428571

## 4 0.5125000 774 273 0 7 1 0.9750000 0 0.025000000

## 5 0.5142857 774 272 0 8 1 0.9714286 0 0.028571429

## 6 0.5196429 774 269 0 11 1 0.9607143 0 0.039285714

## idx

## 1 0

## 2 1

## 3 2

## 4 3

## 5 4

## 6 5

* Plot performance metrics

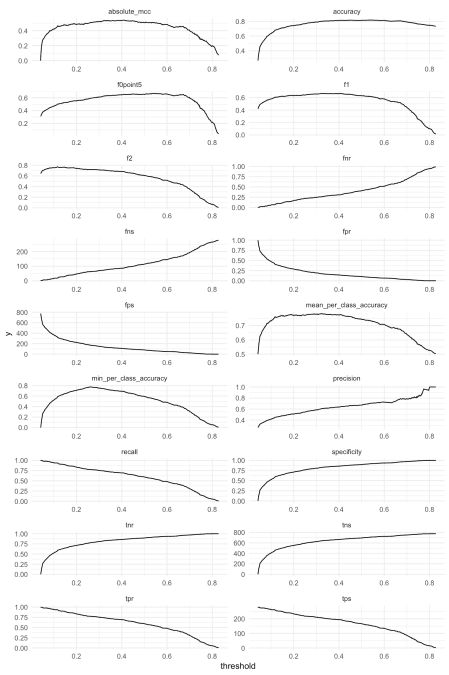
metrics %>%

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

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

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

as.data.frame(h2o.predict(object = best\_model,

newdata = test\_hf)))

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 [?]

## actual predict n

##

## 1 0 0 602

## 2 0 1 172

## 3 1 0 63

## 4 1 1 217

finalRf\_predictions %>%

group\_by(actual, predict\_stringent) %>%

dplyr::summarise(n = n())

## # A tibble: 6 x 3

## # Groups: actual [?]

## actual predict\_stringent n

##

## 1 0 0 683

## 2 0 1 63

## 3 0 unsicher 28

## 4 1 0 101

## 5 1 1 152

## 6 1 unsicher 27

finalRf\_predictions %>%

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

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

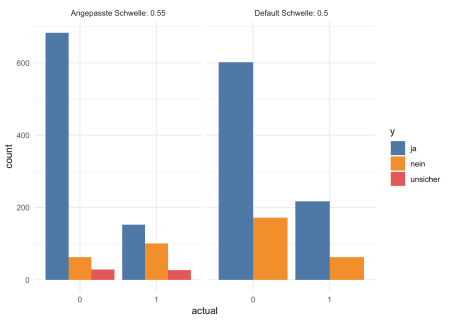
paste("Angepasste Schwelle:", round(threshold, digits = 2)))) %>%

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\_true = NA, prop\_p0\_false = NA,

prop\_p1\_true = NA, prop\_p1\_false = NA)

for (threshold in thresholds) {

pred\_1 <- ifelse(df$p1 > threshold, 1, 0)

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())

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\_true"] <- prop\_p0\_t

prop\_table[prop\_table$threshold == threshold, "prop\_p0\_false"] <- prop\_p0\_f

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\_true"] <- prop\_p1\_t

prop\_table[prop\_table$threshold == threshold, "prop\_p1\_false"] <- prop\_p1\_f

}

prop\_table %>%

gather(x, y, prop\_p0\_true, prop\_p1\_true) %>%

rename(Schwellenwert = threshold) %>%

mutate(x = ifelse(x == "prop\_p0\_true", "prop true p0",

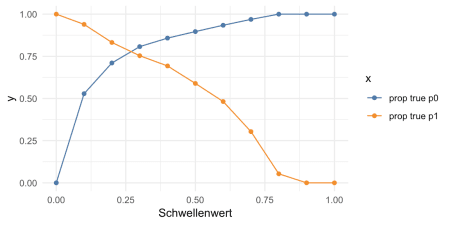
"prop true p1")) %>%

ggplot(aes(x = Schwellenwert, y = y, color = x)) +

geom\_point() +

geom\_line() +

scale\_color\_tableau()



**Cost/revenue calculation**

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\_true == customers who were correctly predicted to not churn did not cost anything (no marketing money was spent): +/-0€
* prop\_p0\_false == customers that did not churn who are predicted to churn will be an empty investment: +/-0€ – 1500€
* prop\_p1\_false == customer that were predicted to stay but churned: -2000€
* prop\_p1\_true == 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

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

customers\_churn\_n <- nrow(customers\_churn)

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

customers\_no\_churn\_n <- nrow(customers\_no\_churn)

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

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

net\_win\_default

## [1] 1021000

* How much revenue can we gain from predicting customer churn (assuming conversionr rate of 0.7):

conversion <- 0.7

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

mutate(prop\_p0\_true\_X = prop\_p0\_true \* customers\_no\_churn\_n \* revenue,

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

prop\_p1\_false\_X = prop\_p1\_false \* customers\_churn\_n \* 0,

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

group\_by(threshold) %>%

summarise(net\_win = sum(prop\_p0\_true\_X + prop\_p0\_false\_X + prop\_p1\_false\_X + prop\_p1\_true\_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 1558000 537000

## 2 0.8 1554000 533000

## 3 0.6 1551000 530000

## 4 0.9 1548000 527000

## 5 1 1548000 527000

## 6 0.5 1534000 513000

## 7 0.4 1515600 494600

## 8 0.3 1483400 462400

## 9 0.2 1417200 396200

## 10 0.1 1288200 267200

## 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, ],

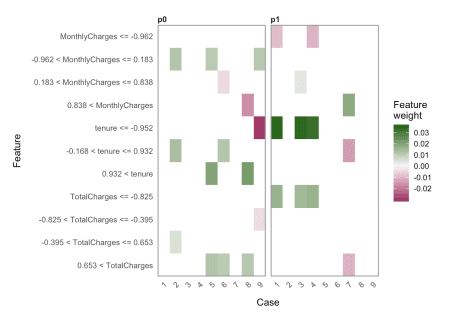
explainer = explainer,

n\_labels = 1,

n\_features = 4,

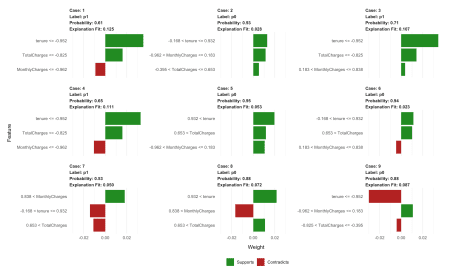
kernel\_width = 0.5)

plot\_explanations(explanation)



explanation %>%

plot\_features(ncol = 3)



sessionInfo()

## R version 3.5.1 (2018-07-02)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS 10.14.1

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/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\_3.20.0.8 bindrcpp\_0.2.2 corrplot\_0.84 ggthemes\_4.0.1

## [5] yardstick\_0.0.2 recipes\_0.1.4 rsample\_0.0.3 lime\_0.4.1

## [9] keras\_2.2.4 mice\_3.3.0 caret\_6.0-81 lattice\_0.20-38

## [13] forcats\_0.3.0 stringr\_1.3.1 dplyr\_0.7.8 purrr\_0.2.5

## [17] readr\_1.2.1 tidyr\_0.8.2 tibble\_1.4.2 ggplot2\_3.1.0

## [21] tidyverse\_1.2.1

##

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

## [1] minqa\_1.2.4 colorspace\_1.3-2 class\_7.3-14

## [4] base64enc\_0.1-3 rstudioapi\_0.8 prodlim\_2018.04.18

## [7] fansi\_0.4.0 lubridate\_1.7.4 xml2\_1.2.0

## [10] codetools\_0.2-15 splines\_3.5.1 knitr\_1.21

## [13] shinythemes\_1.1.2 zeallot\_0.1.0 jsonlite\_1.6

## [16] nloptr\_1.2.1 pROC\_1.13.0 broom\_0.5.1

## [19] tfruns\_1.4 shiny\_1.2.0 compiler\_3.5.1

## [22] httr\_1.3.1 backports\_1.1.2 assertthat\_0.2.0

## [25] Matrix\_1.2-15 lazyeval\_0.2.1 cli\_1.0.1

## [28] later\_0.7.5 htmltools\_0.3.6 tools\_3.5.1

## [31] gtable\_0.2.0 glue\_1.3.0 reshape2\_1.4.3

## [34] Rcpp\_1.0.0 cellranger\_1.1.0 nlme\_3.1-137

## [37] blogdown\_0.9 iterators\_1.0.10 timeDate\_3043.102

## [40] gower\_0.1.2 xfun\_0.4 lme4\_1.1-19

## [43] rvest\_0.3.2 mime\_0.6 pan\_1.6

## [46] MASS\_7.3-51.1 scales\_1.0.0 ipred\_0.9-8

## [49] hms\_0.4.2 promises\_1.0.1 parallel\_3.5.1

## [52] yaml\_2.2.0 reticulate\_1.10 rpart\_4.1-13

## [55] stringi\_1.2.4 tensorflow\_1.10 foreach\_1.4.4

## [58] lava\_1.6.4 bitops\_1.0-6 rlang\_0.3.0.1

## [61] pkgconfig\_2.0.2 evaluate\_0.12 bindr\_0.1.1

## [64] labeling\_0.3 htmlwidgets\_1.3 tidyselect\_0.2.5

## [67] plyr\_1.8.4 magrittr\_1.5 bookdown\_0.8

## [70] R6\_2.3.0 generics\_0.0.2 mitml\_0.3-6

## [73] pillar\_1.3.0 haven\_2.0.0 whisker\_0.3-2

## [76] withr\_2.1.2 RCurl\_1.95-4.11 survival\_2.43-3

## [79] nnet\_7.3-12 modelr\_0.1.2 crayon\_1.3.4

## [82] jomo\_2.6-5 utf8\_1.1.4 rmarkdown\_1.11

## [85] grid\_3.5.1 readxl\_1.1.0 data.table\_1.11.8

## [88] ModelMetrics\_1.2.2 digest\_0.6.18 xtable\_1.8-3

## [91] httpuv\_1.4.5 stats4\_3.5.1 munsell\_0.5.0

## [94] glmnet\_2.0-16