**R Packages Covered in this article**:

* parsnip – NEW Machine Learning API in R, similar to scikit learn in Python
* rsample – 10-Fold Cross Validation
* recipes – Data preprocessing
* yardstick – Model scoring and metrics
* skimr – Quickly skim data
* ranger – Random Forest Library used for churn modeling

**Churn Modeling Using Machine Learning**

Although there are a number of packages at different stages in their development, I have decided to take tidymodels “for a spin”, and create and execute a “tidy” modelling workflow to tackle a **classification problem**. My aim is to show how easy it is to fit a simple ***logistic regression*** in R’s glm and quickly switch to a ***cross-validated random forest*** using the ranger engine by changing only a few lines of code.

For this post in particular I’m focusing on four different libraries from the tidymodels suite:

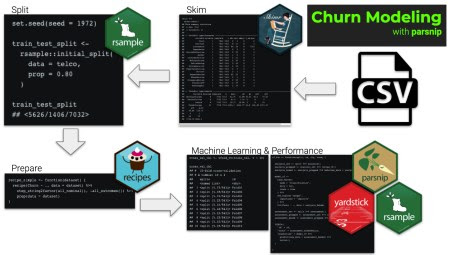
* parsnip for machine learning and modeling
* rsample for data sampling and 10-fold cross-validation
* recipes for data preprocessing
* yardstick for model assessment.

Note that the focus is on modelling workflow and libraries interaction. For that reason, I am keeping data exploration and feature engineering to a minimum. **Data exploration, data wrangling, visualization, and business understanding are *CRITICAL* to your ability to perform machine learning**. If you want to learn the end-to-end process for completing business projects with data science with H2O and parsnip and Shiny web applications using AWS,

**My Workflow**

Here’s a diagram of the workflow I used to web scrape the Specialized Data and create an application:

1. Start with raw data in CSV format
2. Use skimr to *quickly* understand the features
3. Use rsample to split into training/testing sets
4. Use recipes to create data preprocessing pipeline
5. Use parsnip, rsample and yardstick to build models and assess machine learning performance

[](https://www.business-science.io/code-tools/2019/11/18/parsnip-churn-classification-machine-learning.html#workflow)

**Tutorial – Churn Classification using Machine Learning**

This is an ***intermediate tutorial to expose business analysts and data scientists to churn modeling*** with the new parsnip Machine Learning API.

**1.0 Setup and Data**

First, I load the packages I need for this analysis.

library(tidyverse) # Loads dplyr, ggplot2, purrr, and other useful packages

library(tidymodels) # Loads parsnip, rsample, recipes, yardstick

library(skimr) # Quickly get a sense of data

library(knitr) # Pretty HTML Tables

**Churn** is the ***Dependent Variable*** and shows the *customers who left within the last month*. The dataset also includes details on the **Services** that each customer has signed up for, along with **Customer Account** and **Demographic** information.

telco <- read\_csv("<https://raw.githubusercontent.com/DiegoUsaiUK/Classification_Churn_with_Parsnip/master/00_Data/WA_Fn-UseC_-Telco-Customer-Churn.csv>")

telco %>% head() %>% kable()

| **customerID** | **gender** | **SeniorCitizen** | **Partner** | **Dependents** | **tenure** | **PhoneService** | **MultipleLines** | **InternetService** | **OnlineSecurity** | **OnlineBackup** | **DeviceProtection** | **TechSupport** | **StreamingTV** | **StreamingMovies** | **Contract** | **PaperlessBilling** | **PaymentMethod** | **MonthlyCharges** | **TotalCharges** | **Churn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | Yes | No | No | No | No | Month-to-month | Yes | Electronic check | 29.85 | 29.85 | No |
| 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | No | Yes | No | No | No | One year | No | Mailed check | 56.95 | 1889.50 | No |
| 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | Yes | No | No | No | No | Month-to-month | Yes | Mailed check | 53.85 | 108.15 | Yes |
| 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | No | Yes | Yes | No | No | One year | No | Bank transfer (automatic) | 42.30 | 1840.75 | No |
| 9237-HQITU | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | No | No | No | No | No | Month-to-month | Yes | Electronic check | 70.70 | 151.65 | Yes |
| 9305-CDSKC | Female | 0 | No | No | 8 | Yes | Yes | Fiber optic | No | No | Yes | No | Yes | Yes | Month-to-month | Yes | Electronic check | 99.65 | 820.50 | Yes |

**2.0 Skim the Data**

We can get a quick sense of the data using the skim() function from the skimr package.

telco %>% skim()

## Skim summary statistics

## n obs: 7043

## n variables: 21

##

## ── Variable type:character ───────────────────────────────────────────────────────────────────────────────────────────────────

## variable missing complete n min max empty n\_unique

## Churn 0 7043 7043 2 3 0 2

## Contract 0 7043 7043 8 14 0 3

## customerID 0 7043 7043 10 10 0 7043

## Dependents 0 7043 7043 2 3 0 2

## DeviceProtection 0 7043 7043 2 19 0 3

## gender 0 7043 7043 4 6 0 2

## InternetService 0 7043 7043 2 11 0 3

## MultipleLines 0 7043 7043 2 16 0 3

## OnlineBackup 0 7043 7043 2 19 0 3

## OnlineSecurity 0 7043 7043 2 19 0 3

## PaperlessBilling 0 7043 7043 2 3 0 2

## Partner 0 7043 7043 2 3 0 2

## PaymentMethod 0 7043 7043 12 25 0 4

## PhoneService 0 7043 7043 2 3 0 2

## StreamingMovies 0 7043 7043 2 19 0 3

## StreamingTV 0 7043 7043 2 19 0 3

## TechSupport 0 7043 7043 2 19 0 3

##

## ── Variable type:numeric ─────────────────────────────────────────────────────────────────────────────────────────────────────

## variable missing complete n mean sd p0 p25

## MonthlyCharges 0 7043 7043 64.76 30.09 18.25 35.5

## SeniorCitizen 0 7043 7043 0.16 0.37 0 0

## tenure 0 7043 7043 32.37 24.56 0 9

## TotalCharges 11 7032 7043 2283.3 2266.77 18.8 401.45

## p50 p75 p100 hist

## 70.35 89.85 118.75 ▇▁▃▂▆▅▅▂

## 0 0 1 ▇▁▁▁▁▁▁▂

## 29 55 72 ▇▃▃▂▂▃▃▅

## 1397.47 3794.74 8684.8 ▇▃▂▂▁▁▁▁

There are a couple of things to notice here:

* **customerID** is a unique identifier for each row. As such it has no descriptive or predictive power and it needs to be removed.
* Given the relative small number of missing values in **TotalCharges** (only 11 of them) I am dropping them from the dataset.

telco <- telco %>%

select(-customerID) %>%

drop\_na()

**3.0 Tidymodels Workflow – Generalized Linear Model (Baseline)**

To show the basic steps in the tidymodels framework I am fitting and evaluating a simple logistic regression model as a baseline.

**3.1 Train/Test Split**

rsample provides a streamlined way to create a randomised training and test split of the original data.

set.seed(seed = 1972)

train\_test\_split <-

rsample::initial\_split(

data = telco,

prop = 0.80

)

train\_test\_split

## <5626/1406/7032>

Of the 7,043 total customers, 5,626 have been assigned to the training set and 1,406 to the test set. I save them as train\_tbl and test\_tbl.

train\_tbl <- train\_test\_split %>% training()

test\_tbl <- train\_test\_split %>% testing()

**3.2 Prepare**

The recipes package uses a **cooking metaphor** to handle all the data preprocessing, like missing values imputation, removing predictors, centring and scaling, one-hot-encoding, and more.

First, I create a recipe where I define the transformations I want to apply to my data. In this case I create a simple recipe to change all character variables to factors.

Then, I *“prep the recipe”* by mixing the ingredients with prep. Here I have included the prep bit in the recipe function for brevity.

recipe\_simple <- function(dataset) {

recipe(Churn ~ ., data = dataset) %>%

step\_string2factor(all\_nominal(), -all\_outcomes()) %>%

prep(data = dataset)

}

**Note** – In order to avoid ***Data Leakage*** (e.g: transferring information from the train set into the test set), data should be “prepped” using the train\_tbl only.

recipe\_prepped <- recipe\_simple(dataset = train\_tbl)

Finally, to continue with the cooking metaphor, I *“bake the recipe”* to apply all preprocessing to the data sets.

train\_baked <- bake(recipe\_prepped, new\_data = train\_tbl)

test\_baked <- bake(recipe\_prepped, new\_data = test\_tbl)

**3.3 Machine Learning and Performance**

**Fit the Model**

parsnip is a recent addition to the tidymodels suite and is probably the one I like best. This package offers a unified API that allows access to several machine learning packages without the need to learn the syntax of each individual one.

With 3 simple steps you can:

1. Set the type of model you want to fit (here is a logistic regression) and its mode (classification)
2. Decide which computational engine to use (glm in this case)
3. Spell out the exact model specification to fit (I’m using all variables here) and what data to use (the baked train dataset)

logistic\_glm <- logistic\_reg(mode = "classification") %>%

set\_engine("glm") %>%

fit(Churn ~ ., data = train\_baked)

If you want to use another engine, you can simply switch the set\_engine argument (for logistic regression you can choose from glm, glmnet, stan, spark, and keras) and parsnip will take care of changing everything else for you behind the scenes.

**Assess Performance**

predictions\_glm <- logistic\_glm %>%

predict(new\_data = test\_baked) %>%

bind\_cols(test\_baked %>% select(Churn))

predictions\_glm %>% head() %>% kable()

| **.pred\_class** | **Churn** |
| --- | --- |
| Yes | No |
| No | No |
| No | No |
| No | No |
| No | No |
| No | No |

There are several metrics that can be used to investigate the performance of a classification model but for simplicity I’m only focusing on a selection of them: ***accuracy, precision, recall and F1\_Score***.

All of these measures (and many more) can be derived by the ***Confusion Matrix***, a table used to describe the performance of a classification model on a set of test data for which the true values are known.

In and of itself, the confusion matrix is a relatively easy concept to get your head around as is shows the number of *false positives*, *false negatives*, *true positives*, and *true negatives*. However some of the measures that are derived from it may take some reasoning with to fully understand their meaning and use.

predictions\_glm %>%

conf\_mat(Churn, .pred\_class) %>%

pluck(1) %>%

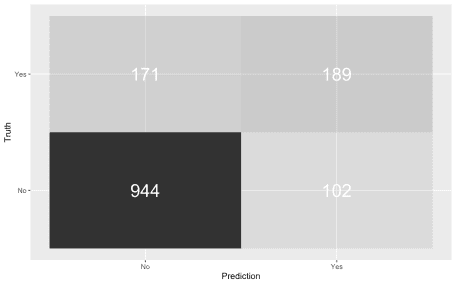
as\_tibble() %>%

# Visualize with ggplot

ggplot(aes(Prediction, Truth, alpha = n)) +

geom\_tile(show.legend = FALSE) +

geom\_text(aes(label = n), colour = "white", alpha = 1, size = 8)



**Accuracy**

The model’s ***Accuracy*** is the fraction of predictions the model got right and can be easily calculated by passing the predictions\_glm to the metrics function. However, accuracy is not a very reliable metric as it will provide misleading results if the data set is unbalanced.

With only basic data manipulation and feature engineering the simple logistic model has achieved 80% accuracy.

predictions\_glm %>%

metrics(Churn, .pred\_class) %>%

select(-.estimator) %>%

filter(.metric == "accuracy") %>%

kable()

| **.metric** | **.estimate** |
| --- | --- |
| accuracy | 0.8058321 |

**Precision and Recall**

***Precision*** shows how sensitive models are to False Positives (i.e. predicting a customer is leaving when he-she is actually staying) whereas ***Recall*** looks at how sensitive models are to False Negatives (i.e. forecasting that a customer is staying whilst he-she is in fact leaving).

**These are very relevant business metrics** because organisations are particularly interested in accurately predicting which customers are truly at risk of leaving so that they can target them with retention strategies. At the same time they want to minimising efforts of retaining customers incorrectly classified as leaving who are instead staying.

tibble(

"precision" =

precision(predictions\_glm, Churn, .pred\_class) %>%

select(.estimate),

"recall" =

recall(predictions\_glm, Churn, .pred\_class) %>%

select(.estimate)

) %>%

unnest(cols = c(precision, recall)) %>%

kable()

| **precision** | **recall** |
| --- | --- |
| 0.8466368 | 0.9024857 |

**F1 Score**

Another popular performance assessment metric is the ***F1 Score***, which is the harmonic average of the precision and recall. An F1 score reaches its best value at 1 with perfect precision and recall.

predictions\_glm %>%

f\_meas(Churn, .pred\_class) %>%

select(-.estimator) %>%

kable()

| **.metric** | **.estimate** |
| --- | --- |
| f\_meas | 0.8736696 |

**4.0 Random Forest – Machine Learning Modeling and Cross Validation**

This is where the real beauty of tidymodels comes into play. Now I can use this tidy modelling framework to fit a **Random Forest** model with the ranger engine.

**4.1 Cross Validation – 10-Fold**

To further refine the model’s predictive power, I am implementing a **10-fold cross validation** using vfold\_cv from rsample, which splits again the initial training data.

cross\_val\_tbl <- vfold\_cv(train\_tbl, v = 10)

cross\_val\_tbl

## # 10-fold cross-validation

## # A tibble: 10 x 2

## splits id

##

## 1 Fold01

## 2 Fold02

## 3 Fold03

## 4 Fold04

## 5 Fold05

## 6 Fold06

## 7 Fold07

## 8 Fold08

## 9 Fold09

## 10 Fold10

If we take a further look, we should recognise the 5,626 number, which is the total number of observations in the initial train\_tbl. In each round, 563 observations will in turn be retained from estimation and used to validate the model for that fold.

cross\_val\_tbl %>% pluck("splits", 1)

## <5063/563/5626>

To avoid confusion and distinguish the initial train/test splits from those used for cross validation, the author of rsample Max Kuhn has coined two new terms: the ***analysis*** and the ***assessment***\_ sets. The former is the portion of the train data used to recursively estimate the model, where the latter is the portion used to validate each estimate.

**4.2 Machine Learning**

**Random Forest**

Switching to another model could not be simpler! All I need to do is to change the **type of model** to random\_forest, add its hyper-parameters, change the set\_engine argument to ranger, and I’m ready to go.

I’m bundling all steps into a function that estimates the model across all folds, runs predictions and returns a convenient tibble with all the results. I need to add an extra step before the recipe “prepping” to maps the cross validation splits to the analysis() and assessment() functions. This will guide the iterations through the 10 folds.

rf\_fun <- function(split, id, try, tree) {

analysis\_set <- split %>% analysis()

analysis\_prepped <- analysis\_set %>% recipe\_simple()

analysis\_baked <- analysis\_prepped %>% bake(new\_data = analysis\_set)

model\_rf <-

rand\_forest(

mode = "classification",

mtry = try,

trees = tree

) %>%

set\_engine("ranger",

importance = "impurity"

) %>%

fit(Churn ~ ., data = analysis\_baked)

assessment\_set <- split %>% assessment()

assessment\_prepped <- assessment\_set %>% recipe\_simple()

assessment\_baked <- assessment\_prepped %>% bake(new\_data = assessment\_set)

tibble(

"id" = id,

"truth" = assessment\_baked$Churn,

"prediction" = model\_rf %>%

predict(new\_data = assessment\_baked) %>%

unlist()

)

}

**Modeling with purrr**

I iteratively apply the random forest modeling function, rf\_fun(), to each of the 10 cross validation folds using purrr.

pred\_rf <- map2\_df(

.x = cross\_val\_tbl$splits,

.y = cross\_val\_tbl$id,

~ rf\_fun(split = .x, id = .y, try = 3, tree = 200)

)

head(pred\_rf)

## # A tibble: 6 x 3

## id truth prediction

##

## 1 Fold01 Yes Yes

## 2 Fold01 Yes No

## 3 Fold01 Yes Yes

## 4 Fold01 No No

## 5 Fold01 No No

## 6 Fold01 Yes Yes

**Assess Performance**

I’ve found that yardstick has a very handy confusion matrix summary() function, which returns an array of **13 different confusion matrix metrics** but in this case I want to see the four I used for the glm model.

pred\_rf %>%

conf\_mat(truth, prediction) %>%

summary() %>%

select(-.estimator) %>%

filter(.metric %in% c("accuracy", "precision", "recall", "f\_meas")) %>%

kable()

| **.metric** | **.estimate** |
| --- | --- |
| accuracy | 0.7975471 |
| precision | 0.8328118 |
| recall | 0.9050279 |
| f\_meas | 0.8674194 |

The random forest model is performing in par with the simple logistic regression. Given the very basic feature engineering that I’ve carried out, there is scope to further improve the model but this is beyond the scope of this post.

**Parting Thoughts**

One of the great advantage of tidymodels is the flexibility and ease of access to every phase of the analysis workflow. Creating the modelling pipeline is a breeze and you can easily re-use the initial framework by changing model type with parsnip and data pre-processing with recipes and in no time you’re ready to check your new model’s performance with yardstick.

In any analysis you would typically audit several models and parsnip frees you up from having to learn the unique syntax of every modelling engine so that you can focus on finding the best solution for the problem at hand.