Final Group project

GUIDO BOZZANO, IGNACIO GAING AND JUAN ARANGUREN

2023-06-09

Table of Contents

# The problem: predicting credit card fraud

The goal of the project is to predict fraudulent credit card transactions.

We will be using a dataset with credit card transactions containing legitimate and fraud transactions. Fraud is typically well below 1% of all transactions, so a naive model that predicts that all transactions are legitimate and not fraudulent would have an accuracy of well over 99%– pretty good, no?

You can read more on credit card fraud on [Credit Card Fraud Detection Using Weighted Support Vector Machine](https://www.scirp.org/journal/paperinformation.aspx?paperid=105944)

The dataset we will use consists of credit card transactions and it includes information about each transaction including customer details, the merchant and category of purchase, and whether or not the transaction was a fraud.

## Obtain the data

The dataset is too large to be hosted on Canvas or Github, so please download it from dropbox <https://www.dropbox.com/sh/q1yk8mmnbbrzavl/AAAxzRtIhag9Nc_hODafGV2ka?dl=0> and save it in your dsb repo, under the data folder.

As we will be building a classifier model using tidymodels, there’s two things we need to do:

1. Define the outcome variable is\_fraud as a factor, or categorical, variable, instead of the numerical 0-1 varaibles.
2. In tidymodels, the first level is the event of interest. If we leave our data as is, 0 is the first level, but we want to find out when we actually did (1) have a fraudulent transaction

## Rows: 671,028  
## Columns: 14  
## $ trans\_date\_trans\_time <dttm> 2019-02-22 07:32:58, 2019-02-16 15:07:20, 2019-…  
## $ trans\_year <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2020, …  
## $ category <chr> "entertainment", "kids\_pets", "personal\_care", "…  
## $ amt <dbl> 7.79, 3.89, 8.43, 40.00, 54.04, 95.61, 64.95, 3.…  
## $ city <chr> "Veedersburg", "Holloway", "Arnold", "Apison", "…  
## $ state <chr> "IN", "OH", "MO", "TN", "CO", "GA", "MN", "AL", …  
## $ lat <dbl> 40.1186, 40.0113, 38.4305, 35.0149, 39.4584, 32.…  
## $ long <dbl> -87.2602, -80.9701, -90.3870, -85.0164, -106.385…  
## $ city\_pop <dbl> 4049, 128, 35439, 3730, 277, 1841, 136, 190178, …  
## $ job <chr> "Development worker, community", "Child psychoth…  
## $ dob <date> 1959-10-19, 1946-04-03, 1985-03-31, 1991-01-28,…  
## $ merch\_lat <dbl> 39.41679, 39.74585, 37.73078, 34.53277, 39.95244…  
## $ merch\_long <dbl> -87.52619, -81.52477, -91.36875, -84.10676, -106…  
## $ is\_fraud <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …

The data dictionary is as follows

| column(variable) | description |
| --- | --- |
| trans\_date\_trans\_time | Transaction DateTime |
| trans\_year | Transaction year |
| category | category of merchant |
| amt | amount of transaction |
| city | City of card holder |
| state | State of card holder |
| lat | Latitude location of purchase |
| long | Longitude location of purchase |
| city\_pop | card holder’s city population |
| job | job of card holder |
| dob | date of birth of card holder |
| merch\_lat | Latitude Location of Merchant |
| merch\_long | Longitude Location of Merchant |
| is\_fraud | Whether Transaction is Fraud (1) or Not (0) |

We also add some of the variables we considered in our EDA for this dataset during homework 2.

card\_fraud <- card\_fraud %>%   
 mutate( hour = hour(trans\_date\_trans\_time),  
 wday = wday(trans\_date\_trans\_time, label = TRUE),  
 month\_name = month(trans\_date\_trans\_time, label = TRUE),  
 age = interval(dob, trans\_date\_trans\_time) / years(1)  
) %>%   
 rename(year = trans\_year) %>%   
   
 mutate(  
   
 # convert latitude/longitude to radians  
 lat1\_radians = lat / 57.29577951,  
 lat2\_radians = merch\_lat / 57.29577951,  
 long1\_radians = long / 57.29577951,  
 long2\_radians = merch\_long / 57.29577951,  
   
 # calculate distance in miles  
 distance\_miles = 3963.0 \* acos((sin(lat1\_radians) \* sin(lat2\_radians)) + cos(lat1\_radians) \* cos(lat2\_radians) \* cos(long2\_radians - long1\_radians)),  
  
 # calculate distance in km  
 distance\_km = 6377.830272 \* acos((sin(lat1\_radians) \* sin(lat2\_radians)) + cos(lat1\_radians) \* cos(lat2\_radians) \* cos(long2\_radians - long1\_radians))  
  
 )

## Exploratory Data Analysis (EDA)

You have done some EDA and you can pool together your group’s expertise in which variables to use as features. You can reuse your EDA from earlier, but we expect at least a few visualisations and/or tables to explore teh dataset and identify any useful features.

Group all variables by type and examine each variable class by class. The dataset has the following types of variables:

1. Strings
2. Geospatial Data
3. Dates
4. Date/Times
5. Numerical

Strings are usually not a useful format for classification problems. The strings should be converted to factors, dropped, or otherwise transformed.

***Strings to Factors***

* category, Category of Merchant
* job, Job of Credit Card Holder

***Strings to Geospatial Data***

We have plenty of geospatial data as lat/long pairs, so I want to convert city/state to lat/long so I can compare to the other geospatial variables. This will also make it easier to compute new variables like the distance the transaction is from the home location.

* city, City of Credit Card Holder
* state, State of Credit Card Holder

## Exploring factors: how is the compactness of categories?

* Do we have excessive number of categories? Do we want to combine some?

card\_fraud %>%   
 count(category, sort=TRUE)%>%   
 mutate(perc = n/sum(n))

## # A tibble: 14 × 3  
## category n perc  
## <chr> <int> <dbl>  
## 1 gas\_transport 68046 0.101   
## 2 grocery\_pos 63791 0.0951  
## 3 home 63597 0.0948  
## 4 shopping\_pos 60416 0.0900  
## 5 kids\_pets 58772 0.0876  
## 6 shopping\_net 50743 0.0756  
## 7 entertainment 48521 0.0723  
## 8 food\_dining 47527 0.0708  
## 9 personal\_care 46843 0.0698  
## 10 health\_fitness 44341 0.0661  
## 11 misc\_pos 41244 0.0615  
## 12 misc\_net 32829 0.0489  
## 13 grocery\_net 23485 0.0350  
## 14 travel 20873 0.0311

card\_fraud %>%   
 count(job, sort=TRUE) %>%   
 mutate(perc = n/sum(n))

## # A tibble: 494 × 3  
## job n perc  
## <chr> <int> <dbl>  
## 1 Film/video editor 5106 0.00761  
## 2 Exhibition designer 4728 0.00705  
## 3 Naval architect 4546 0.00677  
## 4 Surveyor, land/geomatics 4448 0.00663  
## 5 Materials engineer 4292 0.00640  
## 6 Designer, ceramics/pottery 4262 0.00635  
## 7 IT trainer 4014 0.00598  
## 8 Financial adviser 3959 0.00590  
## 9 Systems developer 3948 0.00588  
## 10 Environmental consultant 3831 0.00571  
## # ℹ 484 more rows

The predictors category and job are transformed into factors.

card\_fraud <- card\_fraud %>%   
 mutate(category = factor(category),  
 job = factor(job))

category has 14 unique values, and job has 494 unique values. The dataset is quite large, with over 670K records, so these variables don’t have an excessive number of levels at first glance. However, it is worth seeing if we can compact the levels to a smaller number.

### Why do we care about the number of categories and whether they are “excessive”?

Consider the extreme case where a dataset had categories that only contained one record each. There is simply insufficient data to make correct predictions using category as a predictor on new data with that category label. Additionally, if your modeling uses dummy variables, having an extremely large number of categories will lead to the production of a huge number of predictors, which can slow down the fitting. This is fine if all the predictors are useful, but if they aren’t useful (as in the case of having only one record for a category), trimming them will improve the speed and quality of the data fitting.

If I had subject matter expertise, I could manually combine categories. If you don’t have subject matter expertise, or if performing this task would be too labor intensive, then you can use cutoffs based on the amount of data in a category. If the majority of the data exists in only a few categories, then it might be reasonable to keep those categories and lump everything else in an “other” category or perhaps even drop the data points in smaller categories.

## Do all variables have sensible types?

Consider each variable and decide whether to keep, transform, or drop it. This is a mixture of Exploratory Data Analysis and Feature Engineering, but it’s helpful to do some simple feature engineering as you explore the data. In this project, we have all data to begin with, so any transformations will be performed on the entire dataset. Ideally, do the transformations as a recipe\_step() in the tidymodels framework. Then the transformations would be applied to any data the recipe was used on as part of the modeling workflow. There is less chance of data leakage or missing a step when you perform the feature engineering in the recipe.

## Which variables to keep in your model?

You have a number of variables and you have to decide which ones to use in your model. For instance, you have the latitude/lognitude of the customer, that of the merchant, the same data in radians, as well as the distance\_km and distance\_miles. Do you need them all?

Based on the EDA Analysis, we believe we do not need every variable for the model. Having said that we have decided to use these first set of variables to explore a fraud prediction model. Time variables such as hour, wday and month\_name because we analyzed that fraud tends to happen late and night, during the first half of the year and approximately every day of the week. Another variable would be age because based on the EDA analysis we realized that younger people was more affected by fraud than older people. This could be due by multiple reasons such as lack of credit card use for older people, or easier to fall for fraud as a young person, etc. Also, since we have multiple distance variables in the dataset we decided to focus on 1 only because using several variables that essentially do the same in the model does not make sense. Therefore, we plan to use distance\_km for distance. Furthermore, we decided to exclude the job variable because of the high unique values (494) and because we believe it does not necessarily explain whether a transaction is fraud. On the other hand, we’ve decided to include the category variable since based on the EDA we did for homework2 we saw that there are multiple categories that are fraud more frequent than others (e.g: online vs offline) Last, we included the amount of the fraud since the amount usually tends to be small based on the EDA and the is\_fraud variable that will be our prediction

## Fit your workflows in smaller sample

You will be running a series of different models, along the lines of the California housing example we have seen in class. However, this dataset has 670K rows and if you try various models and run cross validation on them, your computer may slow down or crash.

Thus, we will work with a smaller sample of 10% of the values the original dataset to identify the best model, and once we have the best model we can use the full dataset to train- test our best model.

# select a smaller subset  
my\_card\_fraud <- card\_fraud %>%   
 # select a smaller subset, 10% of the entire dataframe   
 slice\_sample(prop = 0.10)

my\_card\_fraud <- my\_card\_fraud %>%   
 select(is\_fraud, amt, category, hour, wday, month\_name, age, distance\_km)  
  
my\_card\_fraud

## # A tibble: 67,102 × 8  
## is\_fraud amt category hour wday month\_name age distance\_km  
## <fct> <dbl> <fct> <int> <ord> <ord> <dbl> <dbl>  
## 1 0 35.1 gas\_transport 2 Mon Aug 46.1 135.   
## 2 0 78.6 gas\_transport 9 Fri Jan 17.5 87.7  
## 3 0 6.43 home 22 Mon Jan 47.1 86.4  
## 4 0 79.8 grocery\_pos 8 Mon Mar 37.1 123.   
## 5 0 13.7 food\_dining 13 Wed Jan 32.3 103.   
## 6 0 5.33 travel 14 Mon Mar 61.6 42.9  
## 7 0 60.3 gas\_transport 2 Tue Apr 70.4 43.4  
## 8 0 50.8 gas\_transport 9 Mon Nov 47.6 68.8  
## 9 0 64.0 kids\_pets 16 Thu Nov 30.0 45.8  
## 10 0 77.3 home 21 Sat Nov 57.1 88.0  
## # ℹ 67,092 more rows

## Split the data in training - testing

# \*\*Split the data\*\*  
  
set.seed(123)  
  
data\_split <- initial\_split(my\_card\_fraud, # updated data  
 prop = 0.8,   
 strata = is\_fraud)  
  
card\_fraud\_train <- training(data\_split)   
card\_fraud\_test <- testing(data\_split)

## Cross Validation

Start with 3 CV folds to quickly get an estimate for the best model and you can increase the number of folds to 5 or 10 later.

set.seed(123)  
cv\_folds <- vfold\_cv(data = card\_fraud\_train,   
 v = 3,   
 strata = is\_fraud)  
cv\_folds

## # 3-fold cross-validation using stratification   
## # A tibble: 3 × 2  
## splits id   
## <list> <chr>  
## 1 <split [35787/17894]> Fold1  
## 2 <split [35787/17894]> Fold2  
## 3 <split [35788/17893]> Fold3

## Define a tidymodels recipe

What steps are you going to add to your recipe? Do you need to do any log transformations?

fraud\_rec <- recipe(is\_fraud ~ ., data = card\_fraud\_train) %>%  
 step\_log(amt) %>% #We are transforming the amount variable because it's highly skewed to the lower amounts. Therefore applying a log transformation makes the amt variable normal  
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes())  
  
fraud\_rec

##

## ── Recipe ──────────────────────────────────────────────────────────────────────

##

## ── Inputs

## Number of variables by role

## outcome: 1  
## predictor: 7

##

## ── Operations

## • Log transformation on: amt

## • Novel factor level assignment for: all\_nominal(), -all\_outcomes()

## • Dummy variables from: all\_nominal(), -all\_outcomes()

Once you have your recipe, you can check the pre-processed dataframe

prepped\_data <-   
 fraud\_rec %>% # use the recipe object  
 prep() %>% # perform the recipe on training data  
 juice() # extract only the preprocessed dataframe   
  
glimpse(prepped\_data)

## Rows: 53,681  
## Columns: 38  
## $ amt <dbl> 4.3646261, 1.8609745, 4.3801499, 2.6173958, 1.…  
## $ hour <int> 9, 22, 8, 13, 14, 2, 16, 21, 13, 10, 21, 19, 1…  
## $ age <dbl> 17.50250, 47.07106, 37.10512, 32.29659, 61.589…  
## $ distance\_km <dbl> 87.67117, 86.39159, 122.80446, 102.99097, 42.8…  
## $ is\_fraud <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_food\_dining <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_gas\_transport <dbl> 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0…  
## $ category\_grocery\_net <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_grocery\_pos <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1…  
## $ category\_health\_fitness <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0…  
## $ category\_home <dbl> 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_kids\_pets <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0…  
## $ category\_misc\_net <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0…  
## $ category\_misc\_pos <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_personal\_care <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_shopping\_net <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_shopping\_pos <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_travel <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0…  
## $ category\_new <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ wday\_1 <dbl> 0.23145502, -0.38575837, -0.38575837, -0.07715…  
## $ wday\_2 <dbl> -0.23145502, 0.07715167, 0.07715167, -0.385758…  
## $ wday\_3 <dbl> -0.4308202, 0.3077287, 0.3077287, 0.1846372, 0…  
## $ wday\_4 <dbl> -0.1208734, -0.5237849, -0.5237849, 0.3626203,…  
## $ wday\_5 <dbl> 0.3637664, 0.4921546, 0.4921546, -0.3209704, 0…  
## $ wday\_6 <dbl> 0.55391171, -0.30772873, -0.30772873, -0.30772…  
## $ wday\_7 <dbl> 0.35846409, 0.11948803, 0.11948803, 0.59744015…  
## $ month\_name\_01 <dbl> -4.447496e-01, -4.447496e-01, -2.964997e-01, -…  
## $ month\_name\_02 <dbl> 0.49168917, 0.49168917, 0.04469902, 0.49168917…  
## $ month\_name\_03 <dbl> -4.599331e-01, -4.599331e-01, 2.508726e-01, -4…  
## $ month\_name\_04 <dbl> 0.3794580, 0.3794580, -0.3679593, 0.3794580, -…  
## $ month\_name\_05 <dbl> -2.796711e-01, -2.796711e-01, 2.288218e-01, -2…  
## $ month\_name\_06 <dbl> 0.18454194, 0.18454194, 0.06710616, 0.18454194…  
## $ month\_name\_07 <dbl> -1.085750e-01, -1.085750e-01, -3.388855e-01, -…  
## $ month\_name\_08 <dbl> 0.05640062, 0.05640062, 0.45633226, 0.05640062…  
## $ month\_name\_09 <dbl> -2.542464e-02, -2.542464e-02, -4.067943e-01, -…  
## $ month\_name\_10 <dbl> 0.009653491, 0.009653491, 0.267079920, 0.00965…  
## $ month\_name\_11 <dbl> -2.916406e-03, -2.916406e-03, -1.283219e-01, -…  
## $ month\_name\_12 <dbl> 0.0006081128, 0.0006081128, 0.0401354433, 0.00…

## Define various models

You should define the following classification models:

1. Logistic regression, using the glm engine
2. Decision tree, using the C5.0 engine
3. Random Forest, using the ranger engine and setting importance = "impurity")
4. A boosted tree using Extreme Gradient Boosting, and the xgboost engine
5. A k-nearest neighbours, using 4 nearest\_neighbors and the kknn engine

## Model Building   
  
# 1. Pick a `model type`  
# 2. set the `engine`  
# 3. Set the `mode`: classification  
  
# Logistic regression  
log\_spec <- logistic\_reg() %>% # model type  
 set\_engine(engine = "glm") %>% # model engine  
 set\_mode("classification") # model mode  
  
# Show your model specification  
log\_spec

## Logistic Regression Model Specification (classification)  
##   
## Computational engine: glm

# Decision Tree  
tree\_spec <- decision\_tree() %>%  
 set\_engine(engine = "C5.0") %>%  
 set\_mode("classification")  
  
tree\_spec

## Decision Tree Model Specification (classification)  
##   
## Computational engine: C5.0

# Random Forest  
library(ranger)  
  
rf\_spec <-   
 rand\_forest() %>%   
 set\_engine("ranger", importance = "impurity") %>%   
 set\_mode("classification")  
  
  
# Boosted tree (XGBoost)  
library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

xgb\_spec <-   
 boost\_tree() %>%   
 set\_engine("xgboost") %>%   
 set\_mode("classification")   
  
# K-nearest neighbour (k-NN)  
knn\_spec <-   
 nearest\_neighbor(neighbors = 4) %>% # we can adjust the number of neighbors   
 set\_engine("kknn") %>%   
 set\_mode("classification")

## Bundle recipe and model with workflows

## Bundle recipe and model with `workflows`  
  
  
log\_wflow <- # new workflow object  
 workflow() %>% # use workflow function  
 add\_recipe(fraud\_rec) %>% # use the new recipe  
 add\_model(log\_spec) # add your model spec  
  
## A few more workflows  
  
tree\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(tree\_spec)   
  
rf\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(rf\_spec)   
  
xgb\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(xgb\_spec)  
  
knn\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(knn\_spec)

## Fit models

You may want to compare the time it takes to fit each model. tic() starts a simple timer and toc() stops it

tic()  
log\_res <- log\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))

## → A | warning: prediction from a rank-deficient fit may be misleading

## There were issues with some computations A: x1There were issues with some computations A: x2There were issues with some computations A: x3There were issues with some computations A: x3

time <- toc()

## 3.187 sec elapsed

log\_time <- time[[4]]  
  
log\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.995 3 0.000318 Preprocessor1\_Model1  
## 2 f\_meas binary 0.221 3 0.0414 Preprocessor1\_Model1  
## 3 kap binary 0.220 3 0.0414 Preprocessor1\_Model1  
## 4 precision binary 0.789 3 0.0676 Preprocessor1\_Model1  
## 5 recall binary 0.129 3 0.0267 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.845 3 0.0104 Preprocessor1\_Model1  
## 7 sens binary 0.129 3 0.0267 Preprocessor1\_Model1  
## 8 spec binary 1.00 3 0.0000496 Preprocessor1\_Model1

## Decision Tree results  
tic()  
tree\_res <-  
 tree\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
time <- toc()

## 9.899 sec elapsed

tree\_time <- time[[4]]  
  
tree\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.997 3 0.000197 Preprocessor1\_Model1  
## 2 f\_meas binary 0.677 3 0.0297 Preprocessor1\_Model1  
## 3 kap binary 0.675 3 0.0298 Preprocessor1\_Model1  
## 4 precision binary 0.876 3 0.0240 Preprocessor1\_Model1  
## 5 recall binary 0.552 3 0.0298 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.860 3 0.00718 Preprocessor1\_Model1  
## 7 sens binary 0.552 3 0.0298 Preprocessor1\_Model1  
## 8 spec binary 1.00 3 0.0000649 Preprocessor1\_Model1

## Random Forest  
tic()  
rf\_res <-  
 rf\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
time <- toc()

## 22.133 sec elapsed

rf\_time <- time[[4]]  
  
rf\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.997 3 0.000205 Preprocessor1\_Model1  
## 2 f\_meas binary 0.603 3 0.0409 Preprocessor1\_Model1  
## 3 kap binary 0.602 3 0.0409 Preprocessor1\_Model1  
## 4 precision binary 0.974 3 0.0182 Preprocessor1\_Model1  
## 5 recall binary 0.440 3 0.0434 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.964 3 0.00605 Preprocessor1\_Model1  
## 7 sens binary 0.440 3 0.0434 Preprocessor1\_Model1  
## 8 spec binary 1.00 3 0.0000496 Preprocessor1\_Model1

## Boosted tree - XGBoost  
tic()  
xgb\_res <-   
 xgb\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))   
time <- toc()

## 2.156 sec elapsed

xgb\_time <- time[[4]]  
  
xgb\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.998 3 0.0000986 Preprocessor1\_Model1  
## 2 f\_meas binary 0.746 3 0.0200 Preprocessor1\_Model1  
## 3 kap binary 0.745 3 0.0200 Preprocessor1\_Model1  
## 4 precision binary 0.909 3 0.0168 Preprocessor1\_Model1  
## 5 recall binary 0.634 3 0.0277 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.970 3 0.00387 Preprocessor1\_Model1  
## 7 sens binary 0.634 3 0.0277 Preprocessor1\_Model1  
## 8 spec binary 1.00 3 0.0000817 Preprocessor1\_Model1

## K-nearest neighbour  
tic()  
knn\_res <-   
 knn\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))  
time <- toc()

## 173.073 sec elapsed

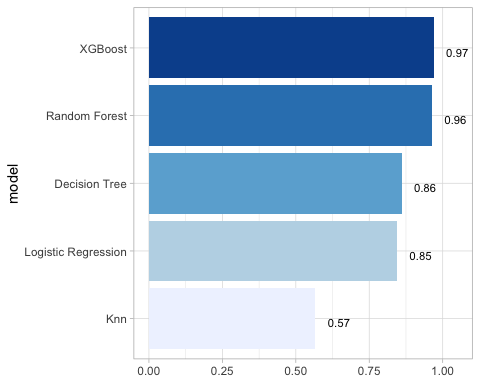
knn\_time <- time[[4]]  
  
knn\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.993 3 0.000130 Preprocessor1\_Model1  
## 2 f\_meas binary 0.121 3 0.0103 Preprocessor1\_Model1  
## 3 kap binary 0.118 3 0.0103 Preprocessor1\_Model1  
## 4 precision binary 0.245 3 0.00691 Preprocessor1\_Model1  
## 5 recall binary 0.0804 3 0.00846 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.567 3 0.00813 Preprocessor1\_Model1  
## 7 sens binary 0.0804 3 0.00846 Preprocessor1\_Model1  
## 8 spec binary 0.999 3 0.0000814 Preprocessor1\_Model1

#The KNN model takes the longer to run with 178.247 seconds whereas the XGBoost takes the shortest amount of time with 2.307 seconds

## Compare models

## Model Comparison  
  
log\_metrics <-   
 log\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 # add the name of the model to every row  
 mutate(model = "Logistic Regression",  
 time = log\_time)  
  
# add mode models here  
tree\_metrics <-   
 tree\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Decision Tree")  
  
rf\_metrics <-   
 rf\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Random Forest")  
  
xgb\_metrics <-   
 xgb\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "XGBoost")  
  
knn\_metrics <-   
 knn\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Knn")  
  
# create dataframe with all models  
model\_compare <- bind\_rows(log\_metrics,  
 tree\_metrics,  
 rf\_metrics,  
 xgb\_metrics,  
 knn\_metrics  
 ) %>%   
 # get rid of 'sec elapsed' and turn it into a number  
 mutate(time = str\_sub(time, end = -13) %>%   
 as.double()  
 )  
  
#Pivot wider to create barplot  
 model\_comp <- model\_compare %>%   
 select(model, .metric, mean, std\_err) %>%   
 pivot\_wider(names\_from = .metric, values\_from = c(mean, std\_err))   
  
# show mean are under the curve (ROC-AUC) for every model  
model\_comp %>%   
 arrange(mean\_roc\_auc) %>%   
 mutate(model = fct\_reorder(model, mean\_roc\_auc)) %>% # order results  
 ggplot(aes(model, mean\_roc\_auc, fill=model)) +  
 geom\_col() +  
 coord\_flip() +  
 scale\_fill\_brewer(palette = "Blues") +  
 geom\_text(  
 size = 3,  
 aes(label = round(mean\_roc\_auc, 2),   
 y = mean\_roc\_auc + 0.08),  
 vjust = 1  
 )+  
 theme\_light()+  
 theme(legend.position = "none")+  
 labs(y = NULL)



#Based on the model comparison we can conclude that the XGBoost has the greatest level of accuracy with 98%, followed by the Random Forest with 0.97

## Which metric to use

This is a highly imbalanced data set, as roughly 99.5% of all transactions are ok, and it’s only 0.5% of transactions that are fraudulent. A naive model, which classifies everything as ok and not-fraud, would have an accuracy of 99.5%, but what about the sensitivity, specificity, the AUC, etc?

## last\_fit()

## `last\_fit()` on test set  
  
# - `last\_fit()` fits a model to the whole training data and evaluates it on the test set.   
# - provide the workflow object of the best model as well as the data split object (not the training data).   
   
last\_fit\_xgb <- last\_fit(xgb\_wflow,   
 split = data\_split,  
 metrics = metric\_set(  
 accuracy, f\_meas, kap, precision,  
 recall, roc\_auc, sens, spec))  
  
last\_fit\_xgb %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 4  
## .metric .estimator .estimate .config   
## <chr> <chr> <dbl> <chr>   
## 1 accuracy binary 0.998 Preprocessor1\_Model1  
## 2 f\_meas binary 0.781 Preprocessor1\_Model1  
## 3 kap binary 0.780 Preprocessor1\_Model1  
## 4 precision binary 0.962 Preprocessor1\_Model1  
## 5 recall binary 0.658 Preprocessor1\_Model1  
## 6 sens binary 0.658 Preprocessor1\_Model1  
## 7 spec binary 1.00 Preprocessor1\_Model1  
## 8 roc\_auc binary 0.988 Preprocessor1\_Model1

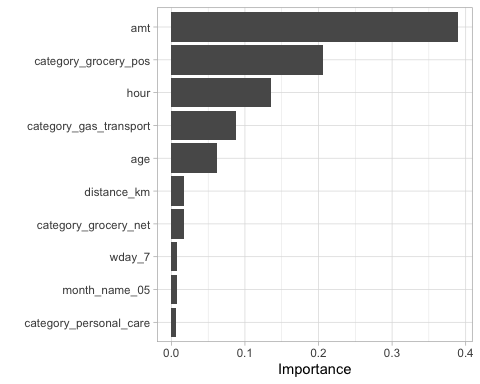
#Compare to training  
xgb\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.998 3 0.0000986 Preprocessor1\_Model1  
## 2 f\_meas binary 0.746 3 0.0200 Preprocessor1\_Model1  
## 3 kap binary 0.745 3 0.0200 Preprocessor1\_Model1  
## 4 precision binary 0.909 3 0.0168 Preprocessor1\_Model1  
## 5 recall binary 0.634 3 0.0277 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.970 3 0.00387 Preprocessor1\_Model1  
## 7 sens binary 0.634 3 0.0277 Preprocessor1\_Model1  
## 8 spec binary 1.00 3 0.0000817 Preprocessor1\_Model1

## Get variable importance using vip package

library(vip)  
  
last\_fit\_xgb %>%   
 pluck(".workflow", 1) %>%   
 pull\_workflow\_fit() %>%   
 vip(num\_features = 10) +  
 theme\_light()

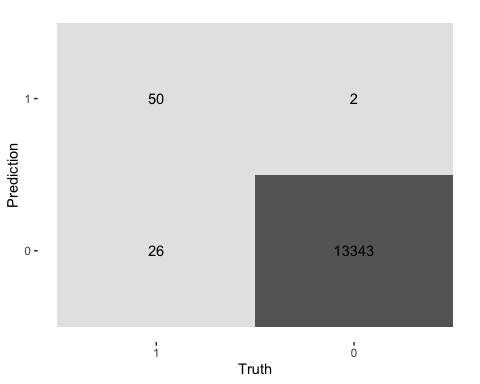
## Warning: `pull\_workflow\_fit()` was deprecated in workflows 0.2.3.  
## ℹ Please use `extract\_fit\_parsnip()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



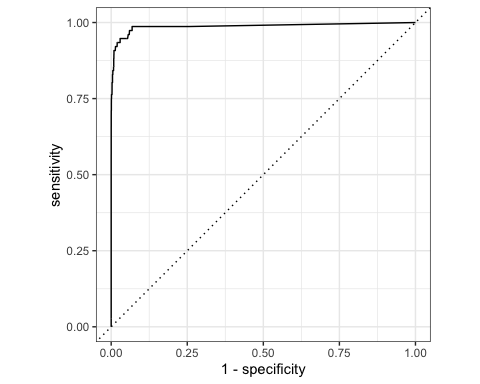
#From this chart we can observe that the amount variable has the greatest level of importance with 50%, followed by the category\_grocery\_pos with 20% and hour with 12.5%. Last, age with 0.9%

## Plot Final Confusion matrix and ROC curve

## Final Confusion Matrix  
  
last\_fit\_xgb %>%  
 collect\_predictions() %>%   
 conf\_mat(is\_fraud, .pred\_class) %>%   
 autoplot(type = "heatmap")



## Final ROC curve  
last\_fit\_xgb %>%   
 collect\_predictions() %>%   
 roc\_curve(is\_fraud, .pred\_1) %>%   
 autoplot()



## Calculating the cost of fraud to the company

* How much money (in US$ terms) are fraudulent transactions costing the company? Generate a table that summarizes the total amount of legitimate and fraudulent transactions per year and calculate the % of fraudulent transactions, in US$ terms. Compare your model vs the naive classification that we do not have any fraudulent transactions.

best\_model\_preds <-   
 xgb\_wflow %>%   
 fit(data = card\_fraud\_train) %>%   
   
 ## Use `augment()` to get predictions for entire data set  
 augment(new\_data = card\_fraud)  
  
best\_model\_preds %>%   
 conf\_mat(truth = is\_fraud, estimate = .pred\_class)

## Truth  
## Prediction 1 0  
## 1 2590 204  
## 0 1346 666888

cost <- best\_model\_preds %>%  
 select(is\_fraud, amt, pred = .pred\_class)   
  
cost <- cost %>%  
 mutate(  
   
  
 # naive false-- we think every single transaction is ok and not fraud  
 false\_naives = ifelse(is\_fraud==1,amt, 0),  
  
 # false negatives-- we thought they were not fraud, but they were  
 false\_negatives = ifelse(pred==0 & is\_fraud ==1, amt, 0),  
   
 # false positives-- we thought they were fraud, but they were not  
 false\_positives = ifelse(pred==1 & is\_fraud ==0, amt, 0),  
   
 # true positives-- we thought they were fraud, and they were   
 true\_positives = ifelse(pred==1 & is\_fraud ==1, amt, 0),  
   
 # true negatives-- we thought they were ok, and they were   
 true\_negatives = ifelse(pred==0 & is\_fraud ==0, amt, 0),  
)  
   
# Summarising  
  
cost\_summary <- cost %>%   
 summarise(across(starts\_with(c("false","true", "amt")),   
 ~ sum(.x, na.rm = TRUE)))  
  
cost\_summary

## # A tibble: 1 × 6  
## false\_naives false\_negatives false\_positives true\_positives true\_negatives  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 2075089. 480363. 179970. 1594726. 44928845.  
## # ℹ 1 more variable: amt <dbl>

#In terms of money we can see that the False Negatives are costing the company around 488k (1.03%) vs the false\_naives that cost around 2M (4.3%). Therefore with our model we are saving more than 1.5M in fraud refunds.

* If we use a naive classifier thinking that all transactions are legitimate and not fraudulent, the cost to the company is $2,075,089.
* With our best model, the total cost of false negatives, namely transactions our classifier thinks are legitimate but which turned out to be fraud, is $480,363.
* Our classifier also has some false positives, $179,970, namely flagging transactions as fraudulent, but which were legitimate. Assuming the card company makes around 2% for each transaction (source: <https://startups.co.uk/payment-processing/credit-card-processing-fees/>), the amount of money lost due to these false positives is $3,599.39
* The $ improvement over the naive policy is $1,591,127.