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.

# Begin the process of mutating our card\_fraud dataframe.  
card\_fraud <- card\_fraud %>%   
  
 # Create a series of new variables related to the transaction date/time.  
 mutate(   
 # Extract the hour from the transaction datetime  
 hour = hour(trans\_date\_trans\_time),  
  
 # Extract the weekday from the transaction datetime, labels provide the actual weekday name  
 wday = wday(trans\_date\_trans\_time, label = TRUE),  
  
 # Extract the month name from the transaction datetime  
 month\_name = month(trans\_date\_trans\_time, label = TRUE),  
  
 # Calculate the age of the customer at the time of the transaction  
 age = interval(dob, trans\_date\_trans\_time) / years(1)  
 ) %>%   
   
 # Rename the transaction year variable  
 rename(year = trans\_year) %>%   
   
 # Continue mutating by adding geographic calculations  
 mutate(  
   
 # Convert latitude/longitude to radians for both customer and merchant  
 lat1\_radians = lat / 57.29577951,  
 lat2\_radians = merch\_lat / 57.29577951,  
 long1\_radians = long / 57.29577951,  
 long2\_radians = merch\_long / 57.29577951,  
   
 # Calculate the geographical distance between customer and merchant in miles using the Haversine formula  
 distance\_miles = 3963.0 \* acos((sin(lat1\_radians) \* sin(lat2\_radians)) +   
 cos(lat1\_radians) \* cos(lat2\_radians) \*   
 cos(long2\_radians - long1\_radians)),  
  
 # Calculate the geographical distance between customer and merchant in kilometers using the Haversine formula  
 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?

# Count the number of occurrences of each category  
# Calculate the proportion each category represents of the total  
card\_fraud\_category <- card\_fraud %>%   
 count(category, sort=TRUE) %>% # Count occurrences for each category, sort in descending order  
 mutate(perc = n/sum(n)) # Compute proportion of each category  
  
# Display results  
print(head(card\_fraud\_category, 5))

## # A tibble: 5 × 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

# Count the number of occurrences of each job  
# Calculate the proportion each job represents of the total  
card\_fraud\_job <- card\_fraud %>%   
 count(job, sort=TRUE) %>% # Count occurrences for each job, sort in descending order  
 mutate(perc = n/sum(n)) # Compute proportion of each job  
  
# Display first 5 rows of the results  
print(head(card\_fraud\_job, 5))

## # A tibble: 5 × 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

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”?

In some cases, we may encounter a dataset where certain categories contain only one record each. This situation poses a challenge because having such limited data makes it difficult to make accurate predictions using those categories as predictors for new data with the same labels. Additionally, if our modeling process involves generating dummy variables, a large number of categories can lead to an overwhelming number of predictors. This abundance of predictors can slow down the model fitting process. While it’s acceptable if all the predictors are useful, it becomes problematic when we have categories with only one record since those predictors are not informative. Trimming them can significantly improve the speed and quality of the data fitting process.

If we possess subject matter expertise, we can manually combine categories based on our knowledge. This allows us to merge related categories and create a more meaningful representation of the data. However, if we lack the necessary expertise or performing manual category combination is time-consuming, we can use cutoffs based on the amount of data within each category as an alternative approach.

By setting cutoffs, we can determine if a category has enough data to be considered meaningful. If the majority of the data is concentrated in only a few categories, it may be reasonable for us to retain those categories and group the remaining ones into an “other” category. Another option could be to exclude the data points belonging to smaller categories altogether.

This strategy of using cutoffs and consolidating categories allows us to simplify the dataset and make the modeling process more efficient. By doing so, we improve our ability to derive meaningful insights from the data while maintaining the overall integrity of the analysis.

## Do all variables have sensible types?

In this project, we need to carefully consider each variable and decide what to do with it: should we keep it as is, transform it in some way, or drop it altogether? This process involves a mix of Exploratory Data Analysis and Feature Engineering. While exploring the data, it’s often helpful to perform some simple feature engineering techniques to enhance our understanding and extract valuable insights.

Now, in this particular project, we have the luxury of having all the data available right from the beginning. This means that any transformations we apply can be performed on the entire dataset. It’s worth noting that it’s ideal to carry out these transformations using a recipe\_step() within the tidymodels framework.

By encapsulating the transformations within a recipe\_step(), we ensure that the same transformations are consistently applied to any new data that the recipe is used on as part of the modeling workflow. This approach helps prevent data leakage and reduces the chances of missing any crucial steps during the feature engineering process.

So, to summarize, the key idea is to carefully assess each variable, determine whether it needs to be kept, transformed, or dropped, and then perform the transformations using a recipe\_step() within the tidymodels framework. This way, we maintain consistency, minimize the risk of data leakage, and ensure that the feature engineering process is properly executed throughout the modeling workflow.

## 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?

From our exploratory data analysis (EDA), we found that not all variables in the dataset are necessary for creating an effective fraud prediction model. We’ve decided to focus on a select group of variables that provide significant insight based on our EDA.

The variables related to time (hour, wday, and month\_name) have been selected because we identified patterns in fraudulent activities. We noted that fraudulent transactions tend to occur late at night, more frequently in the first half of the year, and are spread evenly across all days of the week.

We also chose the ‘age’ variable as our EDA indicated that younger people are more frequently affected by fraud than older people. The reasons for this could be multifaceted, such as the limited use of credit cards by older people or a higher propensity for younger people to fall victim to fraud schemes.

The dataset contains multiple distance-related variables, but to avoid redundancy in our model, we’ll only use one - ‘distance\_km’. This is because multiple variables representing the same underlying information can lead to model overfitting and decreased performance.

We’ve decided to exclude the ‘job’ variable due to its high cardinality (494 unique values) and its limited predictive relevance for fraudulent transactions. However, the ‘category’ variable will be included, as our previous analysis showed significant differences in fraud rates between categories (e.g., online vs offline transactions).

Lastly, we will include the ‘amount’ and ‘is\_fraud’ variables. The ‘amount’ variable was selected as fraudulent transactions often involve small sums, according to our EDA. The ‘is\_fraud’ variable will serve as our target for prediction in the model.

Our choices are rooted in the objective to develop an effective and accurate fraud prediction model, based on relevant and impactful variables.

## 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.

# The objective is to select a smaller subset from the original 'card\_fraud' dataframe.  
# This can be useful for initial data exploration or model prototyping   
# where a full dataset might be computationally heavy.  
  
# We'll use the 'slice\_sample()' function from the 'dplyr' package   
# which allows us to randomly select a proportion of the rows.  
  
my\_card\_fraud <- card\_fraud %>%   
 # The 'prop = 0.10' argument specifies that we want 10% of the rows.  
 # The selected rows are chosen randomly, and thus the sample should be representative if the data is randomly distributed.  
 # Note: The actual number of rows in 'my\_card\_fraud' can be less than exact 10% of the original dataframe's rows  
 # if 'card\_fraud' does not have enough unique rows to fulfill the request.  
 slice\_sample(prop = 0.10)   
  
# Now 'my\_card\_fraud' contains a random subset that is approximately 10% the size of the original 'card\_fraud' dataframe.

# The objective here is to select specific columns from 'my\_card\_fraud' dataframe.   
# We use the 'select()' function from the 'dplyr' package for this task.  
  
my\_card\_fraud <- my\_card\_fraud %>%   
 # The columns to be selected are 'is\_fraud', 'amt', 'category', 'hour', 'wday', 'month\_name', 'age', and 'distance\_km'.  
 # These columns have been chosen based on their potential relevance to the task at hand, which is presumably fraud prediction.  
 select(is\_fraud, amt, category, hour, wday, month\_name, age, distance\_km)  
  
# Now 'my\_card\_fraud' only contains the selected columns.  
  
# Display the first 5 rows of the 'my\_card\_fraud' dataframe using the 'head()' function.  
print(head(my\_card\_fraud, 5))

## # A tibble: 5 × 8  
## is\_fraud amt category hour wday month\_name age distance\_km  
## <fct> <dbl> <fct> <int> <ord> <ord> <dbl> <dbl>  
## 1 0 50.1 gas\_transport 5 Fri Aug 90.2 97.3  
## 2 0 43.2 gas\_transport 3 Sat Feb 83.8 114.   
## 3 0 9.77 shopping\_net 20 Fri Jan 47.3 46.5  
## 4 0 62.3 food\_dining 23 Wed Aug 30.1 72.3  
## 5 0 55.2 gas\_transport 7 Mon Jan 23.1 105.

## 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?

# Prepare a recipe for preprocessing the 'card\_fraud\_train' dataframe.  
fraud\_rec <- recipe(is\_fraud ~ ., data = card\_fraud\_train) %>%  
   
 # Apply log transformation to 'amt' to handle skewness.  
 step\_log(amt) %>%   
   
 # Handle new factor levels in the nominal variables before dummyfication.  
 step\_novel(all\_nominal(), -all\_outcomes()) %>%   
   
 # Convert all nominal variables into binary dummy variables.  
 step\_dummy(all\_nominal(), -all\_outcomes())  
  
# Display the defined preprocessing recipe.  
print(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

# Apply the defined preprocessing recipe to the training data.  
prepped\_data <-   
 fraud\_rec %>% # Use the preprocessing recipe object defined earlier  
 prep() %>% # Execute the preprocessing steps defined in the recipe on the training data  
 juice() # Extract the preprocessed dataframe from the recipe  
  
# Use the 'glimpse()' function to get a concise overview of the preprocessed dataframe  
glimpse(prepped\_data)

## Rows: 53,681  
## Columns: 38  
## $ amt <dbl> 3.766303, 2.279316, 4.010782, 4.315353, 2.1329…  
## $ hour <int> 3, 20, 7, 22, 14, 11, 23, 20, 11, 21, 9, 7, 15…  
## $ age <dbl> 83.81186, 47.30831, 23.05002, 35.92865, 55.654…  
## $ distance\_km <dbl> 114.19502, 46.45455, 105.42949, 112.78996, 118…  
## $ 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, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0…  
## $ category\_gas\_transport <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0…  
## $ category\_health\_fitness <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1…  
## $ category\_home <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_kids\_pets <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…  
## $ category\_misc\_net <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ category\_shopping\_net <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 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> 0.38575837, 0.23145502, -0.38575837, 0.0771516…  
## $ wday\_2 <dbl> 0.07715167, -0.23145502, 0.07715167, -0.385758…  
## $ wday\_3 <dbl> -0.3077287, -0.4308202, 0.3077287, -0.1846372,…  
## $ wday\_4 <dbl> -0.5237849, -0.1208734, -0.5237849, 0.3626203,…  
## $ wday\_5 <dbl> -0.4921546, 0.3637664, 0.4921546, 0.3209704, -…  
## $ wday\_6 <dbl> -0.30772873, 0.55391171, -0.30772873, -0.30772…  
## $ wday\_7 <dbl> -0.11948803, 0.35846409, 0.11948803, -0.597440…  
## $ month\_name\_01 <dbl> -3.706247e-01, -4.447496e-01, -4.447496e-01, -…  
## $ month\_name\_02 <dbl> 0.24584459, 0.49168917, 0.49168917, 0.24584459…  
## $ month\_name\_03 <dbl> -2.266693e-16, -4.599331e-01, -4.599331e-01, -…  
## $ month\_name\_04 <dbl> -0.2529720, 0.3794580, 0.3794580, -0.2529720, …  
## $ month\_name\_05 <dbl> 4.195066e-01, -2.796711e-01, -2.796711e-01, 4.…  
## $ month\_name\_06 <dbl> -0.46135485, 0.18454194, 0.18454194, -0.461354…  
## $ month\_name\_07 <dbl> 3.981082e-01, -1.085750e-01, -1.085750e-01, 3.…  
## $ month\_name\_08 <dbl> -0.28200308, 0.05640062, 0.05640062, -0.282003…  
## $ month\_name\_09 <dbl> 1.652602e-01, -2.542464e-02, -2.542464e-02, 1.…  
## $ month\_name\_10 <dbl> -0.078836844, 0.009653491, 0.009653491, -0.078…  
## $ month\_name\_11 <dbl> 2.916406e-02, -2.916406e-03, -2.916406e-03, 2.…  
## $ month\_name\_12 <dbl> -0.0072973533, 0.0006081128, 0.0006081128, -0.…

## 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. Choose 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 the 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) %>% # Adjust the number of neighbors   
 set\_engine("kknn") %>%   
 set\_mode("classification")

## Bundle recipe and model with workflows

# Bundle recipe and model with `workflows`  
  
# Logistic Regression workflow  
log\_wflow <-   
 workflow() %>% # Create new workflow  
 add\_recipe(fraud\_rec) %>% # Add preprocessing recipe  
 add\_model(log\_spec) # Add logistic regression model  
  
# Decision Tree workflow  
tree\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(tree\_spec)   
  
# Random Forest workflow  
rf\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(rf\_spec)   
  
# XGBoost workflow  
xgb\_wflow <-  
 workflow() %>%  
 add\_recipe(fraud\_rec) %>%   
 add\_model(xgb\_spec)  
  
# k-NN workflow  
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

# List of workflows  
wflows <- list(log\_wflow, tree\_wflow, rf\_wflow, xgb\_wflow, knn\_wflow)  
# Corresponding names  
wflow\_names <- c("logistic regression", "decision tree", "random forest", "xgboost", "k-NN")  
  
# Empty lists to store results and metrics  
res\_list <- list()  
metrics\_list <- list()  
time\_list <- list()  
  
# For each workflow  
for (i in 1:length(wflows)) {  
 tic()  
 wflow\_res <- wflows[[i]] %>%   
 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()  
 time\_list[[i]] <- time[[4]]  
 cat(paste(wflow\_names[i], "takes", time[[4]], "seconds to run.\n"))  
   
 # Save results and metrics  
 res\_list[[i]] <- wflow\_res  
 metrics\_list[[i]] <- wflow\_res %>% collect\_metrics(summarize = 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

## 2.647 sec elapsed  
## logistic regression takes 2.647 sec elapsed seconds to run.  
## 9.889 sec elapsed  
## decision tree takes 9.889 sec elapsed seconds to run.  
## 21.034 sec elapsed  
## random forest takes 21.034 sec elapsed seconds to run.  
## 2.174 sec elapsed  
## xgboost takes 2.174 sec elapsed seconds to run.  
## 170.937 sec elapsed  
## k-NN takes 170.937 sec elapsed seconds to run.

## Compare models

# Combine metrics  
model\_compare <- bind\_rows(metrics\_list) %>%   
 mutate(model = rep(wflow\_names, each = 8), # Update '8' if you change the number of metrics  
 time = rep(time\_list, each = 8),   
 time = str\_sub(time, end = -13) %>% as.double())  
  
# Filter and arrange the accuracy rates  
accuracy\_rates <- model\_compare %>%  
 filter(.metric == "accuracy") %>%  
 arrange(desc(mean))  
  
# Print the top two models with their accuracy rates  
cat("The top two models based on accuracy are:\n")

## The top two models based on accuracy are:

cat(paste(accuracy\_rates$model[1], "with an accuracy of", scales::percent(accuracy\_rates$mean[1], accuracy = 0.01), "\n"))

## xgboost with an accuracy of 99.77%

cat(paste(accuracy\_rates$model[2], "with an accuracy of", scales::percent(accuracy\_rates$mean[2], accuracy = 0.01), "\n"))

## decision tree with an accuracy of 99.71%

## 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()

# Perform last fit on XGBoost model using the test set  
last\_fit\_xgb <- last\_fit(xgb\_wflow,   
 split = data\_split,  
 metrics = metric\_set(  
 accuracy, f\_meas, kap, precision,  
 recall, roc\_auc, sens, spec))  
  
# Collect and summarize metrics from last fit on test set  
last\_fit\_xgb\_metrics <- last\_fit\_xgb %>% collect\_metrics(summarize = TRUE)  
  
# Print metrics for last fit on test set  
cat("Metrics for last fit on test set:\n")

## Metrics for last fit on test set:

print(last\_fit\_xgb\_metrics)

## # A tibble: 8 × 4  
## .metric .estimator .estimate .config   
## <chr> <chr> <dbl> <chr>   
## 1 accuracy binary 0.997 Preprocessor1\_Model1  
## 2 f\_meas binary 0.793 Preprocessor1\_Model1  
## 3 kap binary 0.791 Preprocessor1\_Model1  
## 4 precision binary 0.915 Preprocessor1\_Model1  
## 5 recall binary 0.699 Preprocessor1\_Model1  
## 6 sens binary 0.699 Preprocessor1\_Model1  
## 7 spec binary 1.00 Preprocessor1\_Model1  
## 8 roc\_auc binary 0.982 Preprocessor1\_Model1

# Fit XGBoost model on the training set using fit\_resamples()  
xgb\_fit\_res <- xgb\_wflow %>%   
 fit\_resamples(  
 resamples = cv\_folds,  
 metrics = metric\_set(  
 accuracy, f\_meas, kap, precision,  
 recall, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE)  
 )  
  
# Collect and summarize metrics for XGBoost model on the training set  
xgb\_fit\_metrics <- xgb\_fit\_res %>% collect\_metrics(summarize = TRUE)  
  
# Print metrics for XGBoost model on training set  
cat("Metrics for XGBoost model on training set:\n")

## Metrics for XGBoost model on training set:

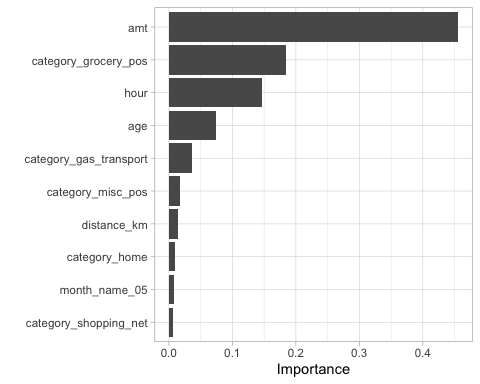
print(xgb\_fit\_metrics)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.998 3 0.000183 Preprocessor1\_Model1  
## 2 f\_meas binary 0.759 3 0.0188 Preprocessor1\_Model1  
## 3 kap binary 0.758 3 0.0189 Preprocessor1\_Model1  
## 4 precision binary 0.902 3 0.0118 Preprocessor1\_Model1  
## 5 recall binary 0.656 3 0.0242 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.985 3 0.00518 Preprocessor1\_Model1  
## 7 sens binary 0.656 3 0.0242 Preprocessor1\_Model1  
## 8 spec binary 1.00 3 0.0000324 Preprocessor1\_Model1

## Get variable importance using vip package

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

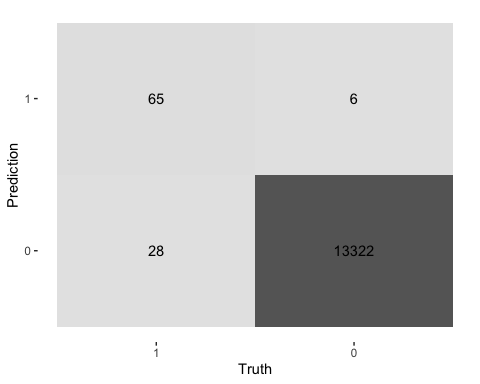
## Warning: `pull\_workflow\_fit()` was deprecated in workflows 0.2.3.  
## ℹ Please use `extract\_fit\_parsnip()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



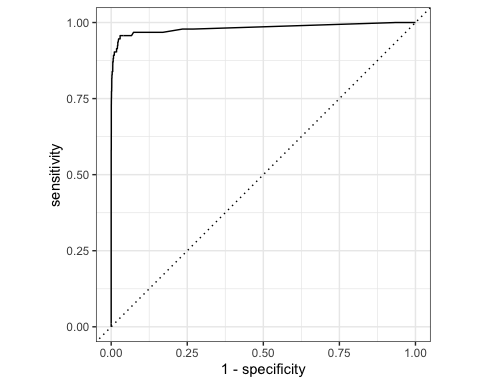
#From this chart we can observe that the amount variable has the greatest level of importance with 38%, followed by the category\_grocery\_pos with 20% and hour with 10%. Last, age with 0.9%

## 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.

# Fit the model and obtain predictions  
best\_model\_preds <- xgb\_wflow %>%   
 fit(data = card\_fraud\_train) %>%   
 augment(new\_data = card\_fraud)  
  
# Compute confusion matrix  
conf\_matrix <- best\_model\_preds %>%   
 conf\_mat(truth = is\_fraud, estimate = .pred\_class)  
  
# Select relevant columns for cost analysis  
cost <- best\_model\_preds %>%  
 select(is\_fraud, amt, pred = .pred\_class)   
  
# Calculate different cost components  
cost <- cost %>%  
 mutate(  
 false\_naives = ifelse(is\_fraud == 1, amt, 0), # Naive false: all transactions considered not fraud  
 false\_negatives = ifelse(pred == 0 & is\_fraud == 1, amt, 0), # False negatives: predicted not fraud, but actually fraud  
 false\_positives = ifelse(pred == 1 & is\_fraud == 0, amt, 0), # False positives: predicted fraud, but actually not fraud  
 true\_positives = ifelse(pred == 1 & is\_fraud == 1, amt, 0), # True positives: predicted fraud and actually fraud  
 true\_negatives = ifelse(pred == 0 & is\_fraud == 0, amt, 0) # True negatives: predicted not fraud and actually not fraud  
 )  
  
# Summarize the cost components  
cost\_summary <- cost %>%   
 summarise(across(starts\_with(c("false", "true", "amt")), ~ sum(.x, na.rm = TRUE)))  
  
cost\_summary <- cost\_summary / 1000000 # Convert cost values to millions  
  
cost\_summary

## false\_naives false\_negatives false\_positives true\_positives true\_negatives  
## 1 2.075089 0.3539973 0.2358012 1.721092 44.87301  
## amt  
## 1 47.1839

# Analyze cost summary  
false\_negatives\_cost <- cost\_summary$false\_negatives  
false\_naives\_cost <- cost\_summary$false\_naives  
  
# Compare cost components  
if (false\_negatives\_cost < false\_naives\_cost) {  
 message("False negatives are costing the company less in terms of money.")  
} else if (false\_naives\_cost < false\_negatives\_cost) {  
 message("False naives are costing the company less in terms of money.")  
} else {  
 message("False negatives and false naives are costing the company the same amount.")  
}

## False negatives are costing the company less in terms of money.

# Calculate the total savings in fraud refunds  
total\_savings <- false\_naives\_cost - false\_negatives\_cost  
message("With our model, we are saving $", total\_savings, " million in fraud refunds.")

## With our model, we are saving $1.72109185 million in fraud refunds.

* If we use a naive classifier thinking that all transactions are legitimate and not fraudulent, the cost to the company is $2.08.
* 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 $0.35.
* Our classifier also has some false positives, $0.24, 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 $0.00
* The $ improvement over the naive policy is $1.72.