Harvard Data Science Capstone: Elo Merchant Category Recommendation

James Hope

1/24/2019

Right now, Elo, one of the largest payment brands in Brazil, has built partnerships with merchants in order to offer promotions or discounts to cardholders. But do these promotions work for either the consumer or the merchant? Do customers enjoy their experience? Do merchants see repeat business? Personalization is key.

Elo has built machine learning models to understand the most important aspects and preferences in their customers' lifecycle, from food to shopping. But so far none of them is specifically tailored for an individual or profile.

In this task I am developing an algorithm to identify and serve the most relevant opportunities to individuals, by uncovering signal in customer loyalty. This will enable Elo reduce unwanted campaigns, to create the right experience for customers.

Data files I will be using for this task

I will need, at a minimum, the train.csv and test.csv files. These contain the card_ids that we'll be using for training and prediction.

The historical_transactions.csv and new_merchant_transactions.csv files contain information about each card's transactions. historical_transactions.csv contains up to 3 months' worth of transactions for every card at any of the provided merchant_ids. new_merchant_transactions.csv contains the transactions at new merchants (merchant_ids that this particular card_id has not yet visited) over a period of two months.

merchants.csv contains aggregate information for each merchant_id represented in the data set.

How is the data formatted

The data is formatted as follows:

train.csv and test.csv contain card_ids and information about the card itself - the first month the card was active, etc. train.csv also contains the target.

historical_transactions.csv and new_merchant_transactions.csv are designed to be joined with train.csv, test.csv, and merchants.csv. They contain information about transactions for each card, as described above.

merchants can be joined with the transaction sets to provide additional merchant-level information.

What I am predicting

I will be building a model that predicts a loyalty score for each card_id. I will report the best RMSE.

File descriptions

The following files have been provided.

- train.csv - the training set
- test.csv - the test set
- sample_submission.csv - a sample submission file in the correct format - contains all card_ids that I will be predicting for.
- historical_transactions.csv - up to 3 months' worth of historical transactions for each card_id
- merchants.csv - additional information about all merchants / merchant_ids in the dataset.
- new_merchant_transactions.csv - two months' worth of data for each card_id containing ALL purchases that card_id made at merchant_ids that were not visited in the historical data.

Data Import

First we'll set up our environment, import the data and install a number of packages that we'll be using.

```
## Warning: unable to access index for repository http://nbcgib.uesc.br/mirrors/cran/src/contrib:
##   cannot open URL 'http://nbcgib.uesc.br/mirrors/cran/src/contrib/PACKAGES'
```

```
## Warning: packages 'corrplot', 'xgboost', 'mlbench', 'caret', 'lubridate',
## 'magrittr', 'tidyverse', 'dplyr', 'ggplot2', 'Matrix', 'Ckmeans.1d.dp' are
## not available (for R version 3.4.1)
```

```
## Warning: unable to access index for repository http://nbcgib.uesc.br/mirrors/cran/bin/macosx/el-capi
##   cannot open URL 'http://nbcgib.uesc.br/mirrors/cran/bin/macosx/el-capitan/contrib/3.4/PACKAGES'
```

```
## Loading required package: corrplot
```

```
## Warning: package 'corrplot' was built under R version 3.4.2
```

```
## corrplot 0.84 loaded
```

```
## Loading required package: xgboost
```

```
## Loading required package: mlbench
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
## Loading required package: lubridate
```

```
## Warning: package 'lubridate' was built under R version 3.4.4
```

```
##
## Attaching package: 'lubridate'
## The following object is masked from 'package:base':
##
##     date

## Loading required package: magrittr

## Loading required package: tidyverse

## Warning: package 'tidyverse' was built under R version 3.4.2

## -- Attaching packages ------------------------------------------------------------------- tidyverse

## v tibble  2.0.0     v purrr   0.2.5
## v tidyr   0.8.2     v dplyr   0.7.8
## v readr   1.3.1     v stringr 1.3.1
## v tibble  2.0.0     v forcats 0.3.0

## Warning: package 'tidyr' was built under R version 3.4.4

## Warning: package 'readr' was built under R version 3.4.4

## Warning: package 'purrr' was built under R version 3.4.4

## Warning: package 'dplyr' was built under R version 3.4.4

## Warning: package 'stringr' was built under R version 3.4.4

## Warning: package 'forcats' was built under R version 3.4.3

## -- Conflicts ------------------------------------------------------------------ tidyverse_confli
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x tidyr::extract()         masks magrittr::extract()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x purrr::lift()            masks caret::lift()
## x purrr::set_names()       masks magrittr::set_names()
## x lubridate::setdiff()     masks base::setdiff()
## x dplyr::slice()           masks xgboost::slice()
## x lubridate::union()       masks base::union()

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 3.4.4

##
## Attaching package: 'Matrix'
## The following object is masked from 'package:tidyr':
##
##     expand

## Loading required package: Ckmeans.1d.dp

## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'Ckmeans.1d.dp'

## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] TRUE
##
## [[5]]
## [1] TRUE
##
## [[6]]
## [1] TRUE
##
## [[7]]
## [1] TRUE
##
## [[8]]
## [1] TRUE
##
## [[9]]
## [1] TRUE
##
## [[10]]
## [1] TRUE
##
## [[11]]
## [1] FALSE

## Warning: package 'data.table' was built under R version 3.4.4

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year
```

```
## Warning: package 'scales' was built under R version 3.4.4
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## Parsed with column specification:
## cols(
##   first_active_month = col_character(),
##   card_id = col_character(),
##   feature_1 = col_double(),
##   feature_2 = col_double(),
##   feature_3 = col_double()
## )
```

```
## Parsed with column specification:
## cols(
##   first_active_month = col_character(),
##   card_id = col_character(),
##   feature_1 = col_double(),
##   feature_2 = col_double(),
##   feature_3 = col_double(),
##   target = col_double()
## )
```

```r
head(test)
```

```
## # A tibble: 6 x 5
##   first_active_month card_id         feature_1 feature_2 feature_3
##   <chr>              <chr>               <dbl>     <dbl>     <dbl>
## 1 2017-04            C_ID_0ab67a22ab         3         3         1
## 2 2017-01            C_ID_130fd0cbdd         2         3         0
## 3 2017-08            C_ID_b709037bc5         5         1         1
## 4 2017-12            C_ID_d27d835a9f         2         1         0
## 5 2015-12            C_ID_2b5e3df5c2         5         1         1
## 6 2017-07            C_ID_5814b4f13c         5         1         1
```

```
head(train)
```

```
## # A tibble: 6 x 6
##   first_active_month card_id         feature_1 feature_2 feature_3 target
##   <chr>              <chr>               <dbl>     <dbl>     <dbl>  <dbl>
## 1 2017-06            C_ID_92a2005557         5         2         1 -0.820
## 2 2017-01            C_ID_3d0044924f         4         1         0  0.393
## 3 2016-08            C_ID_d639edf6cd         2         2         0  0.688
## 4 2017-09            C_ID_186d6a6901         4         3         0  0.142
## 5 2017-11            C_ID_cdbd2c0db2         1         3         0 -0.160
## 6 2016-09            C_ID_0894217f2f         4         2         0  0.872
```

Data Exploration & Visualisation

Train / Test

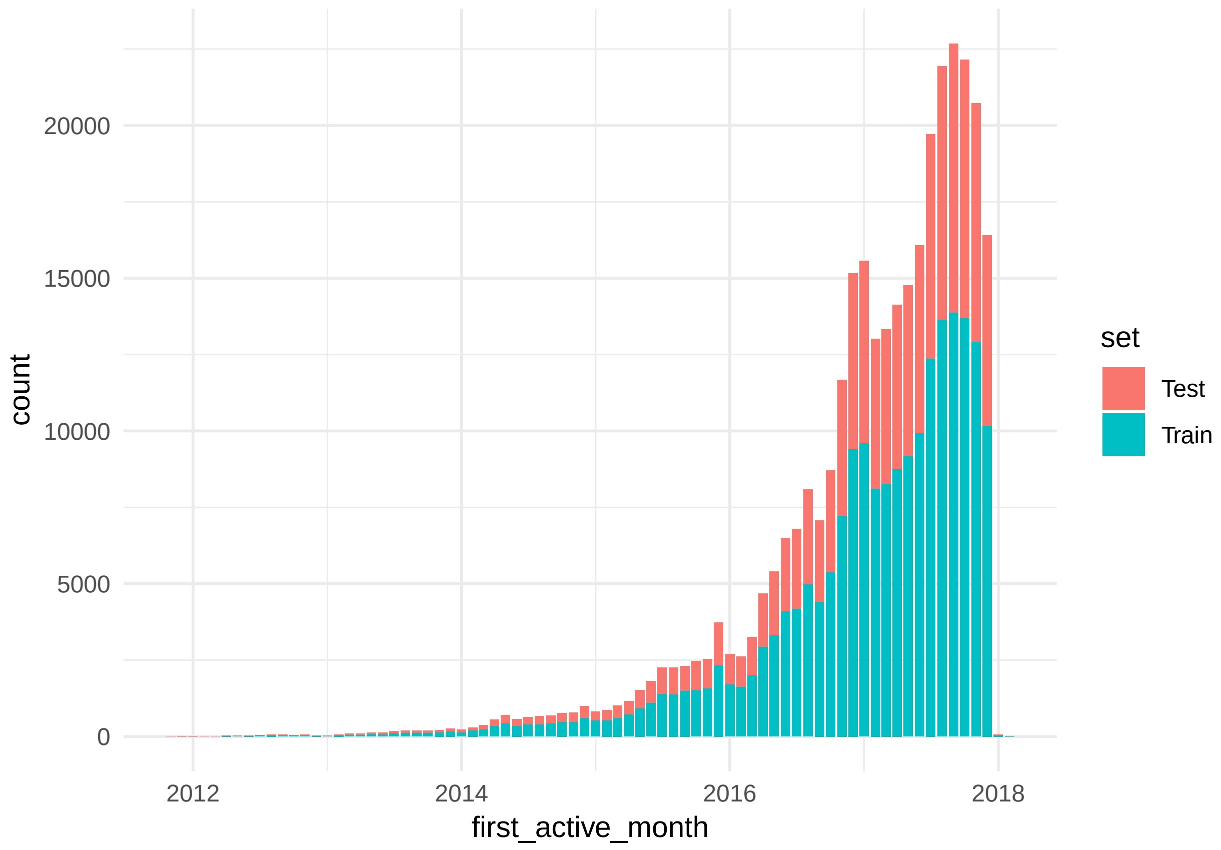
There is a total of 5 features:

- **card_id** - unique card identifier
- **first_active_month** - 'YYYY-MM', month of first purchase
- **feature_1** - anonymized card categorical feature
- **feature_2** - anonymized card categorical feature
- **feature_3** - anonymized card categorical feature

Let's plot how the dates of the first purchases are distributed:

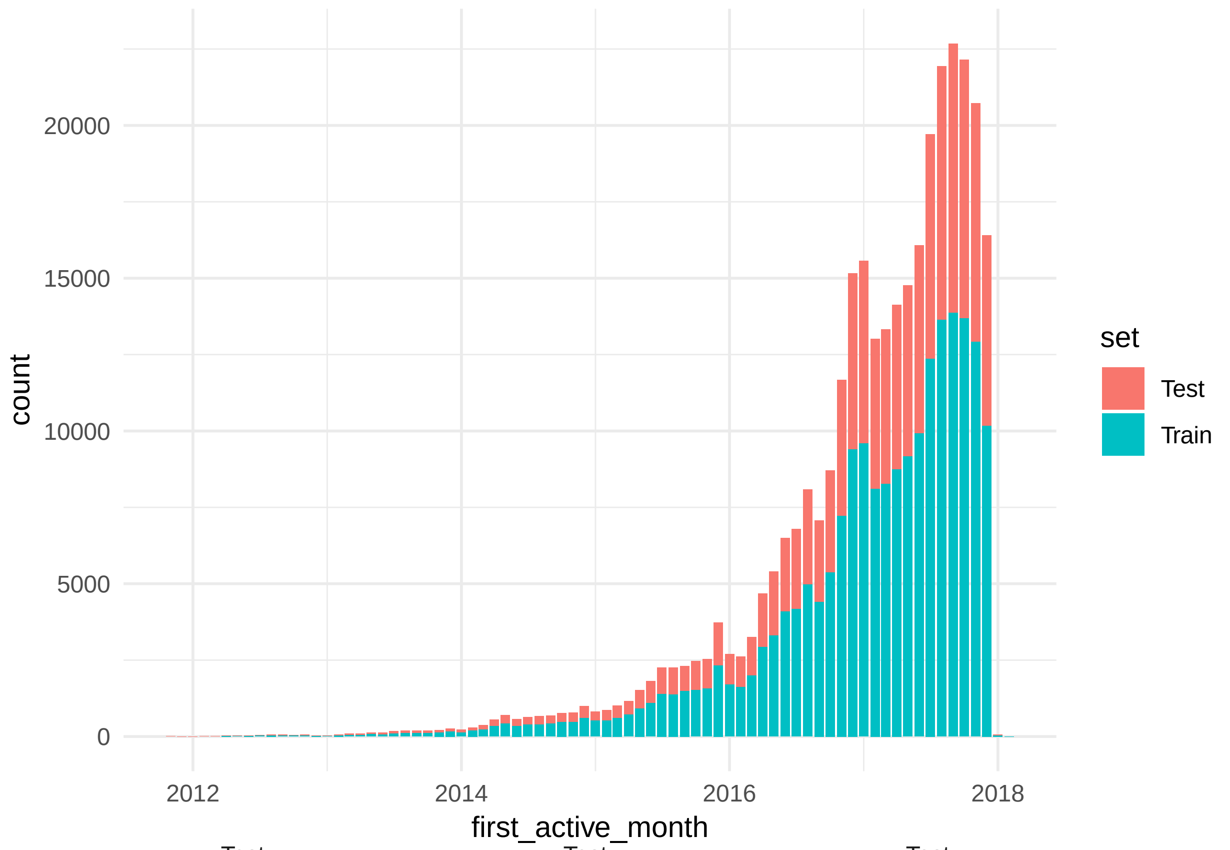
```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
## Warning: Removed 1 rows containing non-finite values (stat_count).
```

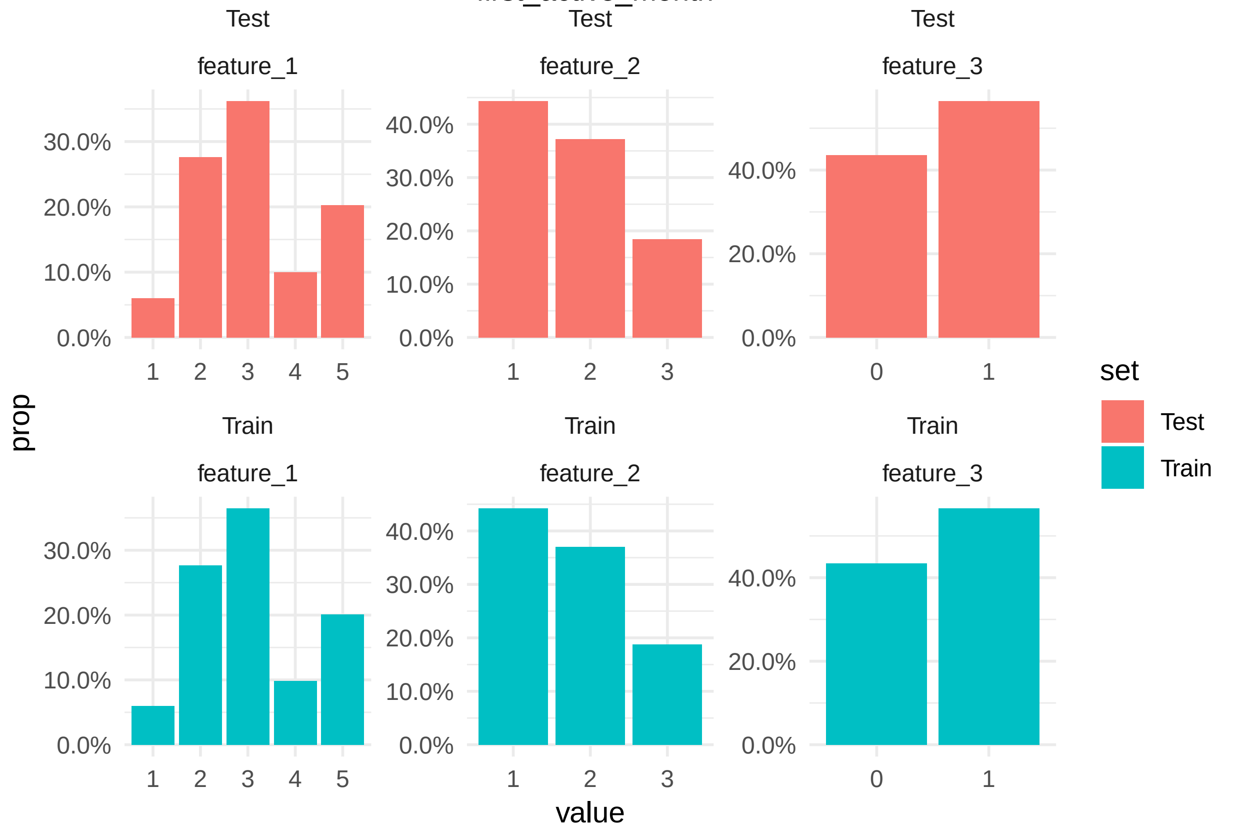


We note the distribution of features across the train and test data.

```
## Warning: Removed 1 rows containing non-finite values (stat_count).
```



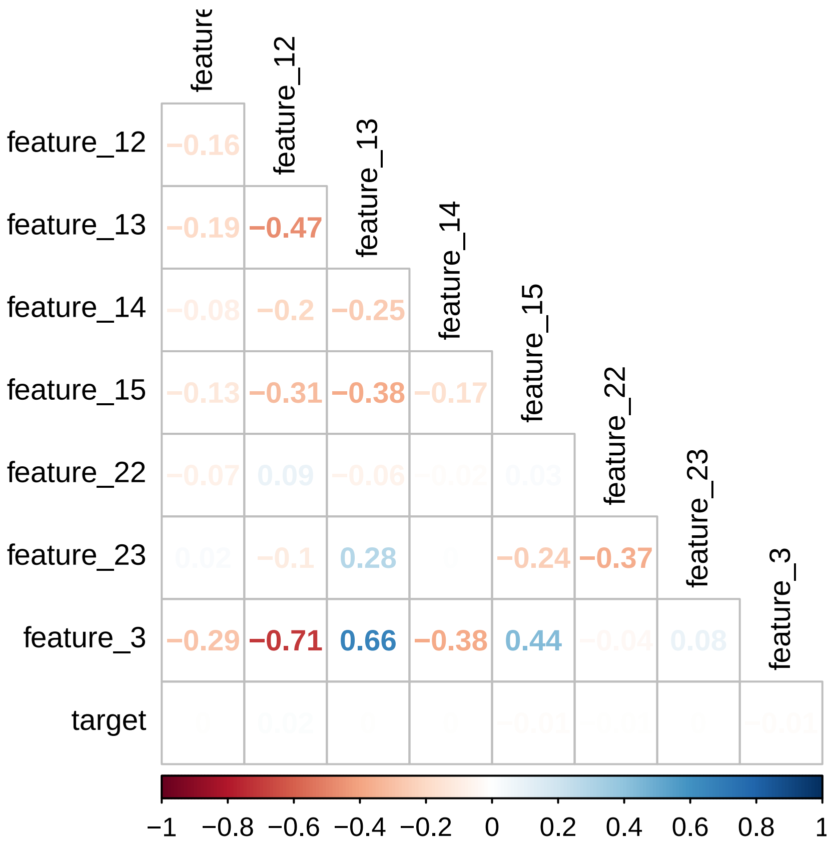
count
20000
15000
10000
5000
0
2012
2014
2016
2018
first_active_month
set
Test
Train



Test
feature_1
feature_2
feature_3
Train
prop
30.0%
20.0%
10.0%
0.0%
40.0%
1
2
3
4
5
0
value
set
Test
Train

Correlation

Let’s convert categorical features to dummy variables and check correlations:



We notice that Feature 1 and Feature 3 are correlated, however the correlation coefficient is relatively low, and typically too low to remove a feature.

Transaction Data

Let’s read in the historical and new merchant transaction data. There are a total of 14 features in each dataset.

There is a total of 14 features in each dataset:

- **card_id** - card identifier
- **month_lag** - month lag to reference date
- **purchase_date** - purchase date
- **authorized_flag** - ‘Y’ if approved, ‘N’ if denied
- **category_3** - anonymized category
- **installments** - number of installments of purchase
- **category_1** - anonymized category
- **merchant_category_id** - merchant category identifier (anonymized)
- **subsector_id** - merchant category group identifier (anonymized)
- **merchant_id** - merchant identifier (anonymized)
- **purchase_amount** - normalized purchase amount
- **city_id** - city identifier (anonymized)
- **state_id** - state identifier (anonymized)

- **category__2** - anonymized category

```
## Parsed with column specification:
## cols(
##   card_id = col_character(),
##   target = col_double()
## )
```

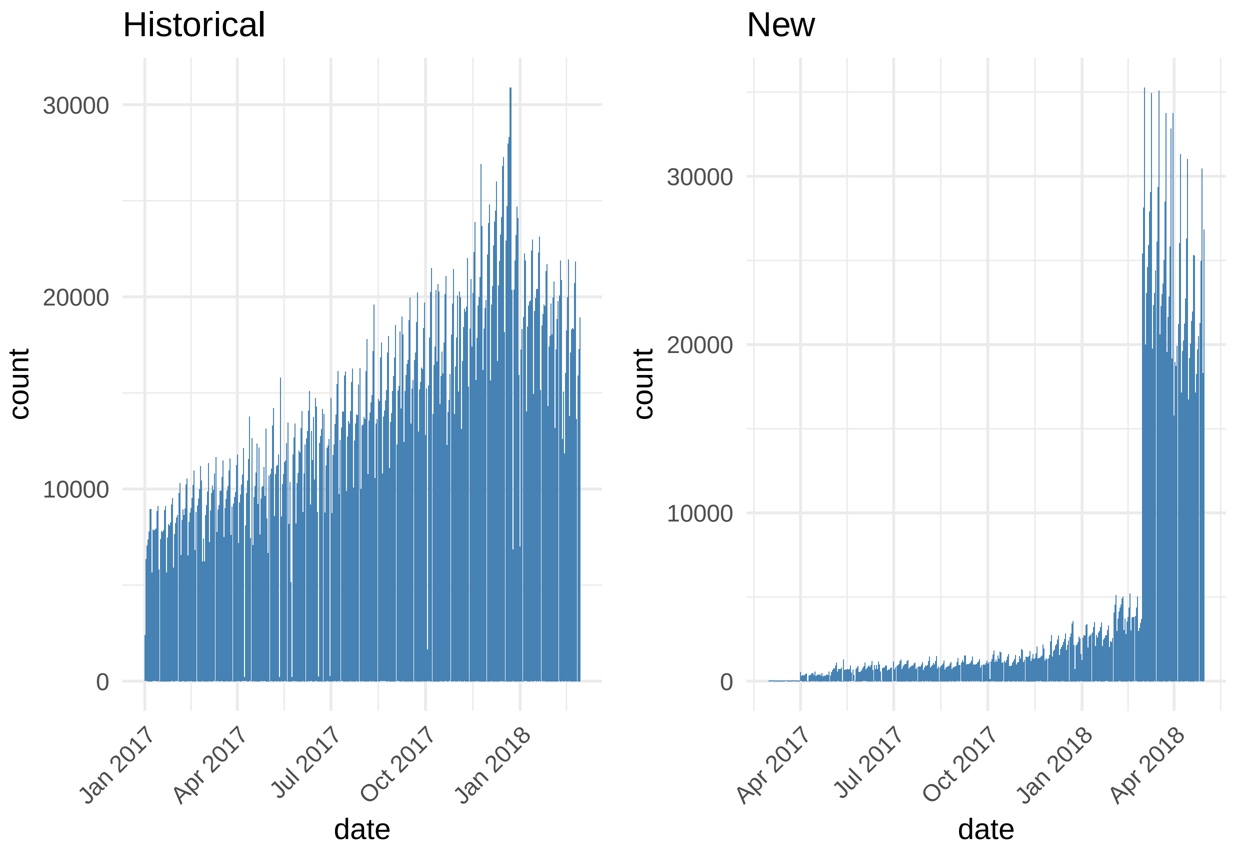
```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   merchant_id = col_character(),
##   category_1 = col_character(),
##   most_recent_sales_range = col_character(),
##   most_recent_purchases_range = col_character(),
##   category_4 = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
## Parsed with column specification:
## cols(
##   authorized_flag = col_character(),
##   card_id = col_character(),
##   city_id = col_double(),
##   category_1 = col_character(),
##   installments = col_double(),
##   category_3 = col_character(),
##   merchant_category_id = col_double(),
##   merchant_id = col_character(),
##   month_lag = col_double(),
##   purchase_amount = col_double(),
##   purchase_date = col_datetime(format = ""),
##   category_2 = col_double(),
##   state_id = col_double(),
##   subsector_id = col_double()
## )
## Parsed with column specification:
## cols(
##   authorized_flag = col_character(),
##   card_id = col_character(),
##   city_id = col_double(),
##   category_1 = col_character(),
##   installments = col_double(),
##   category_3 = col_character(),
##   merchant_category_id = col_double(),
##   merchant_id = col_character(),
##   month_lag = col_double(),
##   purchase_amount = col_double(),
##   purchase_date = col_datetime(format = ""),
##   category_2 = col_double(),
##   state_id = col_double(),
##   subsector_id = col_double()
## )
## # A tibble: 6 x 2
##   card_id         target
##   <chr>            <dbl>
## 1 C_ID_0ab67a22ab      0
## 2 C_ID_130fd0cbdd      0
## 3 C_ID_b709037bc5      0
## 4 C_ID_d27d835a9f      0
## 5 C_ID_2b5e3df5c2      0
## 6 C_ID_5814b4f13c      0

## # A tibble: 6 x 14
##   authorized_flag card_id city_id category_1 installments category_3
##   <chr>           <chr>     <dbl> <chr>             <dbl> <chr>
## 1 Y               C_ID_4~      88 N                     0 A
## 2 Y               C_ID_4~      88 N                     0 A
## 3 Y               C_ID_4~      88 N                     0 A
## 4 Y               C_ID_4~      88 N                     0 A
## 5 Y               C_ID_4~      88 N                     0 A
## 6 Y               C_ID_4~     333 N                     0 A
## # ... with 8 more variables: merchant_category_id <dbl>,
## #   merchant_id <chr>, month_lag <dbl>, purchase_amount <dbl>,
## #   purchase_date <dttm>, category_2 <dbl>, state_id <dbl>,
## #   subsector_id <dbl>

## # A tibble: 6 x 22
##   merchant_id merchant_group_~ merchant_catego~ subsector_id numerical_1
##   <chr>                  <dbl>            <dbl>        <dbl>       <dbl>
## 1 M_ID_83806~             8353              792            9     -0.0575
## 2 M_ID_9339d~             3184              840           20     -0.0575
## 3 M_ID_e726b~              447              690            1     -0.0575
## 4 M_ID_a70e9~             5026              792            9     -0.0575
## 5 M_ID_64456~             2228              222           21     -0.0575
## 6 M_ID_a0915~            20201               87           27     -0.0575
## # ... with 17 more variables: numerical_2 <dbl>, category_1 <chr>,
## #   most_recent_sales_range <chr>, most_recent_purchases_range <chr>,
## #   avg_sales_lag3 <dbl>, avg_purchases_lag3 <dbl>,
## #   active_months_lag3 <dbl>, avg_sales_lag6 <dbl>,
## #   avg_purchases_lag6 <dbl>, active_months_lag6 <dbl>,
## #   avg_sales_lag12 <dbl>, avg_purchases_lag12 <dbl>,
## #   active_months_lag12 <dbl>, category_4 <chr>, city_id <dbl>,
## #   state_id <dbl>, category_2 <dbl>

## # A tibble: 6 x 14
##   authorized_flag card_id city_id category_1 installments category_3
##   <chr>           <chr>     <dbl> <chr>             <dbl> <chr>
## 1 Y               C_ID_4~     107 N                     1 B
## 2 Y               C_ID_4~     140 N                     1 B
## 3 Y               C_ID_4~     330 N                     1 B
## 4 Y               C_ID_4~      -1 Y                     1 B
## 5 Y               C_ID_e~      -1 Y                     1 B
## 6 Y               C_ID_e~     231 N                     1 B
## # ... with 8 more variables: merchant_category_id <dbl>,
## #   merchant_id <chr>, month_lag <dbl>, purchase_amount <dbl>,
## #   purchase_date <dttm>, category_2 <dbl>, state_id <dbl>,
## #   subsector_id <dbl>
```

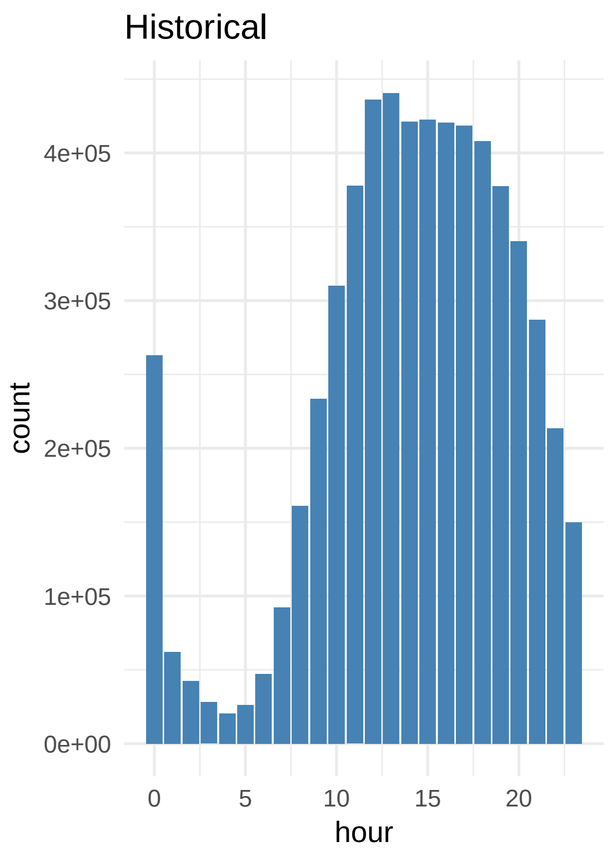
Purchase date

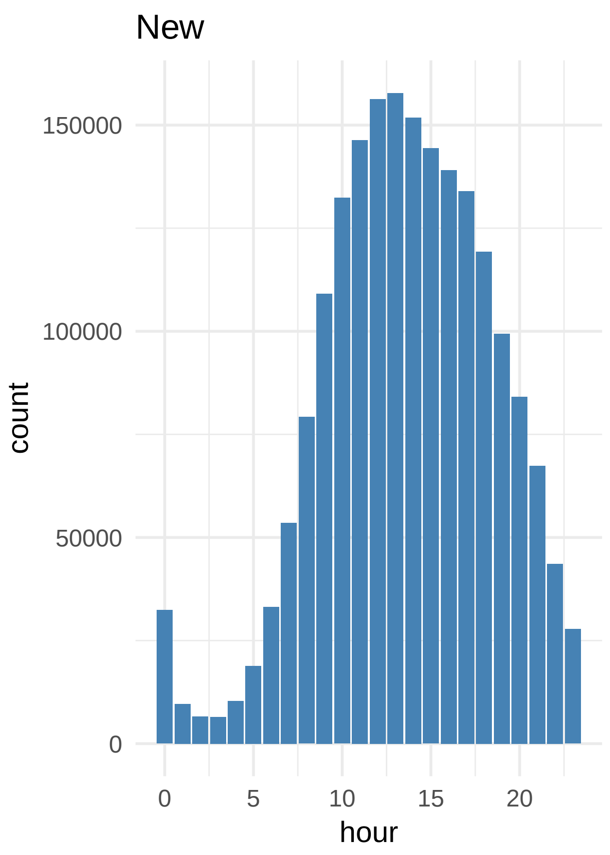
We note a bias in the distribution by purchase date.



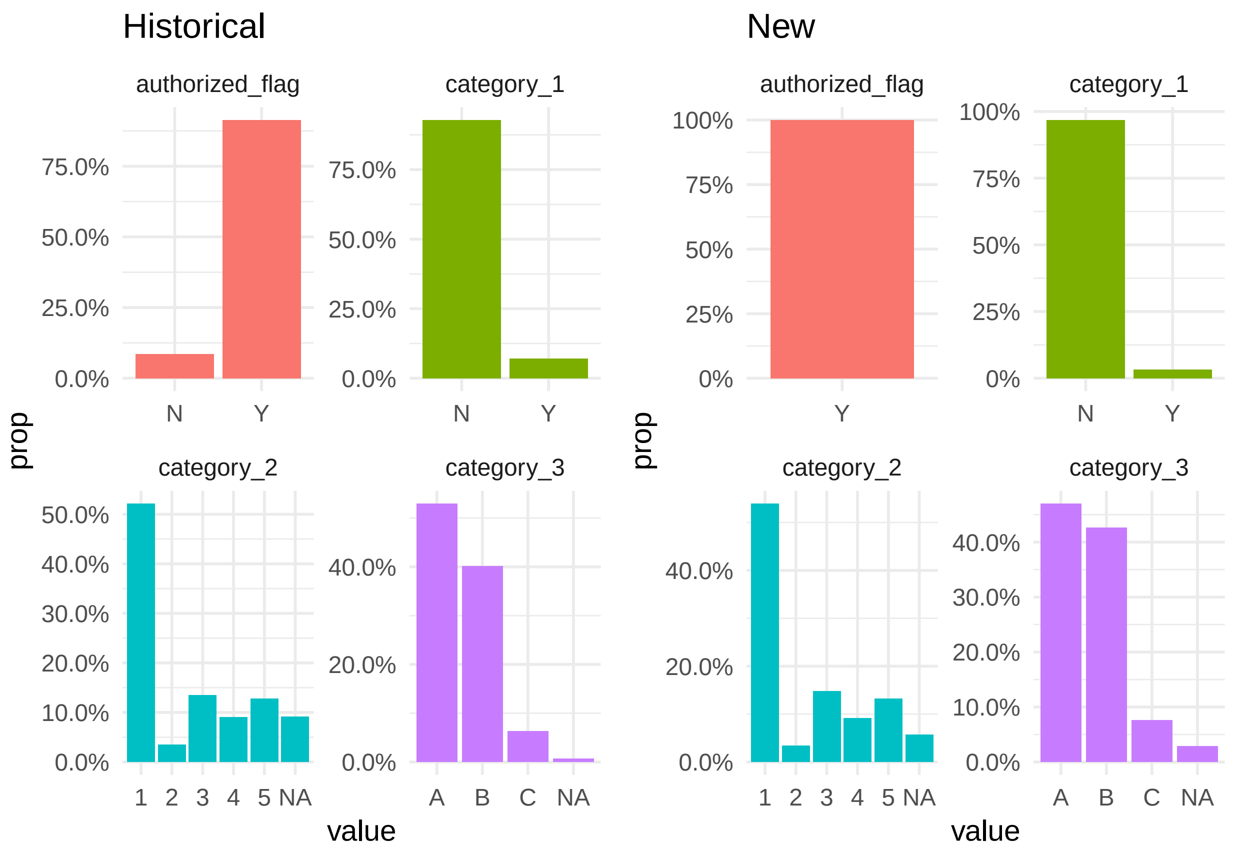
Purchase Data by Hour

We note a bias in the distribution by hour.


Historical
count
4e+05
3e+05
2e+05
1e+05
0e+00
0
5
10
15
20
hour


New
count
150000
100000
50000
0
0
5
10
15
20
hour

Other features



Merchants

In this additional dataset, there is a total of 22 features:

- **merchant_id** - unique merchant identifier
- **merchant_group_id** - merchant group (anonymized)
- **merchant_category_id** - unique identifier for merchant category (anonymized)
- **subsector_id** - merchant category group (anonymized)
- **numerical_1** - anonymized measure
- **numerical_2** - anonymized measure
- **category_1** - anonymized category
- **most_recent_sales_range** - range of revenue (monetary units) in last active month –> A > B > C > D > E
- **most_recent_purchases_range** - range of quantity of transactions in last active month –> A > B > C > D > E
- **avg_sales_lag3** - monthly average of revenue in last 3 months divided by revenue in last active month
- **avg_purchases_lag3** - monthly average of transactions in last 3 months divided by transactions in last active month
- **active_months_lag3** - quantity of active months within last 3 months
- **avg_sales_lag6** - monthly average of revenue in last 6 months divided by revenue in last active month
- **avg_purchases_lag6** - monthly average of transactions in last 6 months divided by transactions in last active month

- **active__months__lag6** - quantity of active months within last 6 months
- **avg__sales__lag12** - monthly average of revenue in last 12 months divided by revenue in last active month
- **avg__purchases__lag12** - monthly average of transactions in last 12 months divided by transactions in last active month
- **active__months__lag12** - quantity of active months within last 12 months category_4 - anonymized category
- **city__id** - city identifier (anonymized)
- **state__id** - sState identifier (anonymized)
- **category__2** - anonymized category

This additional dataset contains many numerical features.

Model Building

In an ideal world we would want to build and compare the accuracy of models. However, because of the large number of predictive features in the data provided we will use Xgboost (eXtreme Gradient Boosting package) to build a predictive model. Xgboost includes an efficient linear model solver and tree learning algorithms and will automatically do parallel computation on a single machine which can be more than 10 times faster than existing gradient boosting packages. More information on Xgboost can be found here: https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html

Data Processing

```
## Parsed with column specification:
## cols(
##   first_active_month = col_character(),
##   card_id = col_character(),
##   feature_1 = col_double(),
##   feature_2 = col_double(),
##   feature_3 = col_double(),
##   target = col_double()
## )
```

```
## Parsed with column specification:
## cols(
##   first_active_month = col_character(),
##   card_id = col_character(),
##   feature_1 = col_double(),
##   feature_2 = col_double(),
##   feature_3 = col_double()
## )
```

```
## Parsed with column specification:
## cols(
##   authorized_flag = col_character(),
##   card_id = col_character(),
##   city_id = col_double(),
##   category_1 = col_character(),
##   installments = col_double(),
##   category_3 = col_character(),
##   merchant_category_id = col_double(),
##   merchant_id = col_character(),
##   month_lag = col_double(),
##   purchase_amount = col_double(),
##   purchase_date = col_datetime(format = ""),
##   category_2 = col_double(),
##   state_id = col_double(),
##   subsector_id = col_double()
## )

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   merchant_id = col_character(),
##   category_1 = col_character(),
##   most_recent_sales_range = col_character(),
##   most_recent_purchases_range = col_character(),
##   category_4 = col_character()
## )

## See spec(...) for full column specifications.

## Parsed with column specification:
## cols(
##   authorized_flag = col_character(),
##   card_id = col_character(),
##   city_id = col_double(),
##   category_1 = col_character(),
##   installments = col_double(),
##   category_3 = col_character(),
##   merchant_category_id = col_double(),
##   merchant_id = col_character(),
##   month_lag = col_double(),
##   purchase_amount = col_double(),
##   purchase_date = col_datetime(format = ""),
##   category_2 = col_double(),
##   state_id = col_double(),
##   subsector_id = col_double()
## )
```

We'll prepare the data ready for model fitting by performing the following steps: * Separate the historical transaction data into authorised (htrans_auth) and unauthorised (htrans) * Join the merchant information to the new transactions data (ntrans)

For the htrans_auth, htrans and ntrans: * One-Hot-Encode the category information in each using model.matrix.lm() * Add statistical features (mean, standard deviation etc.) for each feature * Separate purchase_date into hours, days etc

Next we can bind train and test together and then join the feature enhanced historical and new transaction data on card_id. We'll create new predictive features for the year, month and date difference for the first_active_month for each card. We will convert the data into a matrix as is required by Xgboost.

Preparing the Data

Next we'll partition the data into a training, validation and test set ready to feed into the Gradient Boost algorithm.

```
## Warning: unable to access index for repository http://nbcgib.uesc.br/mirrors/cran/src/contrib:
##   cannot open URL 'http://nbcgib.uesc.br/mirrors/cran/src/contrib/PACKAGES'
```

```
## Warning: package 'xgboost' is not available (for R version 3.4.1)
```

```
## Warning: unable to access index for repository http://nbcgib.uesc.br/mirrors/cran/bin/macosx/el-capi
##   cannot open URL 'http://nbcgib.uesc.br/mirrors/cran/bin/macosx/el-capitan/contrib/3.4/PACKAGES'
```

```
##             used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   2468081  131.9    7907136   422.3   30163339  1610.9
## Vcells 737538747 5627.0 2424174764 18495.0 2220412767 16940.5
```

Model Training

Defining the Hyperparameters & Cost Function

Now we can train the Xgboost model. We'll use linear regression with the RMSE as the cost function.

```
p <- list(objective = "reg:linear",
          booster = "gbtree",
          eval_metric = "rmse",
          nthread = 4,
          eta = 0.02,
          max_depth = 7,
          min_child_weight = 100,
          gamma = 0,
          subsample = 0.85,
          colsample_bytree = 0.8,
          colsample_bylevel = 0.8,
          alpha = 0,
          lambda = 0.1)
```

Training the Model

We'll train the model using this simple peice of code. We'll train the Xgboost model until the RMSE hasn't improved in 200 rounds. Further information on the training parameters can be found here: https://www.rdocumentation.org/packages/xgboost/versions/0.71.2/topics/xgb.train

```
## [1]  val-rmse:3.963089
## Will train until val_rmse hasn't improved in 200 rounds.
##
## [100]    val-rmse:3.692688
```

Predictions

Finally we'll make predictions with the model. Note that the label provided to Xgboost was the logarithmic RMSE, so the model will output predictions of the logarithmic RMSE. In other words, we won't need to calculate this.

Finally we report the best RSME score reported by the model.

```
## val-rmse
##  3.69269
```