1. Introduction

Instacart is an American company that operates as a same-day grocery delivery service. Customers select groceries through a web application from various retailers and delivered by a personal shopper. Instacart's service is mainly provided through a smartphone app, available on iOS and Android platforms, apart from its website.

In 2017 Instacart organised a Kaggle competition and provided to the community a sample of over 3 million grocery orders from more than 200,000 Instacart users. The orders include 32 million basket items and 50,000 unique products. The objective of the competition was to predict which previously purchased products will be in a user’s next order.

1.1 Objective

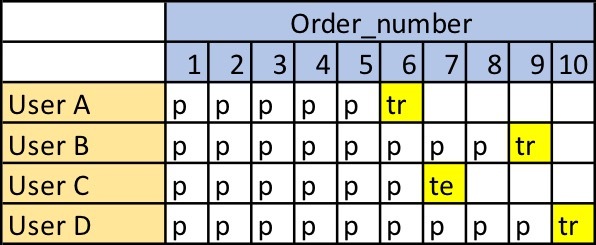
The objective of this Kernel is to introduce graduate students to predictive business analytics with R through the Instacart case.

By the time you finish this example, you will be able to:

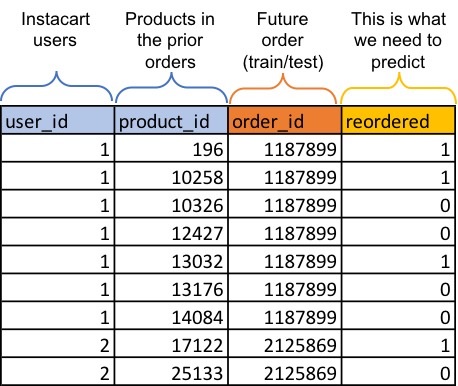
* Describe the steps of creating a predictive analytics model
* Use R to manipulate data
* Use R to create, combine, and delete data tables
* Use XGBoost to create a predictive model
* Apply the predictive model in order to make a prediction

1.2 Problem definition

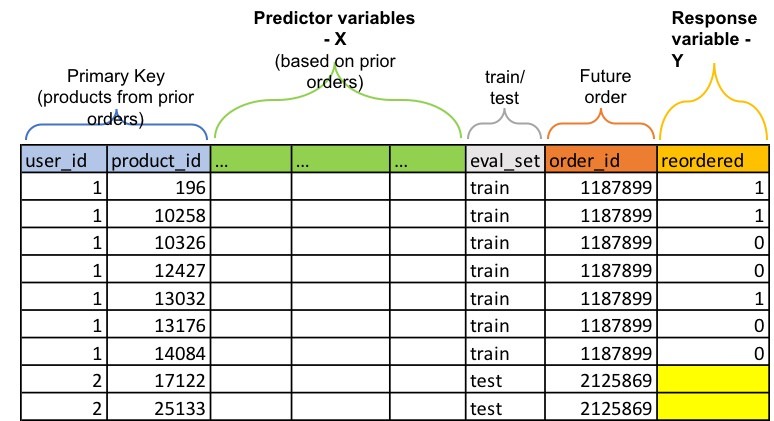
The data that Instacart opened up include orders of 200,000 Instacart users with each user having between 4 and 100 orders. Instacart indicates each order in the data as prior, train or test. Prior orders describe the past behaviour of a user while train and test orders regard the future behaviour that we need to predict. As a result, we want to predict which previously purchased products (prior orders) will be in a user’s next order (train and test orders). For the train orders Instacart reveals the results (i.e. the ordered products) while for the test orders we do not have this piece of information. Moreover, the future order of each user can be either train or test meaning that each user will be either a train or a test user. The setting of the Instacart problem is described in the figure below.



Each user has purchased various products during their prior orders. Moreover, for each user we know the order\_id of their future order. The goal is to predict which of these products will be in a user's future order. This is a classification problem because we need to predict whether each pair of user and product is a reorder or not. This is indicated by the value of the reordered variable, i.e. reordered=1 or reordered=0 (see figure below).



As a result we need to come up and calculate various predictor variables (X) that will describe the characteristics of a product and the behaviour of a user regarding one or multiple products. We will do so by analysing the prior orders of the dataset. We will then use the train users to create a predictive model and the test users to make our actual prediction. As a result we create a table as the following one and we train an algorithm based on predictor variables (X) and response variable (Y).



1.3 Method

Our method includes the following steps:

1. **Import and reshape data**: This step includes loading CSV files into R tables, tranform character variables to categorical variables, and create a supportive table.
2. **Calculate predictor variables**: This step includes identifying and calculating predictor variables (aka features) from the initial datasets provided by Instacart.
3. **Create the test and train datasets**: In this step we create two distinct datasets that will be used in the creation and the use of the predictive model.
4. **Create the preditive model**: In this step we employ XGBoost algorithm to create the predictive model through the train dataset.
5. **Apply the model**: This step includes applying the model to predict the 'reordered' variable for the test dataset.

2. Import and Reshape Data

We first load the necessary R packages using the library() function.

Output

In [1]:

library(data.table)

library(dplyr)

library(tidyr)

2.1 Load data from the CSV files

Instacart provides 6 CSV files, which we have to load into R. Towards this end, we use the fread() function, which is included in the data.table package. This function facilitates importing large datasets into R. Reading in data with the fread() function returns a data table.

Output

In [2]:

path <- "../input"

orderp <- fread(file.path(path, "order\_products\_\_prior.csv"))

ordert <- fread(file.path(path, "order\_products\_\_train.csv"))

orders <- fread(file.path(path, "orders.csv"))

products <- fread(file.path(path, "products.csv"))

aisles <- fread(file.path(path, "aisles.csv"))

departments <- fread(file.path(path, "departments.csv"))

This step results in the following tables:

* **orders**: This table includes all orders, namely prior, train, and test. It has single primary key (**order\_id**).
* **ordert**: This table includes training orders. It has a composite primary key (**order\_id and product\_id**) and indicates whether a product in an order is a reorder or not (through the reordered variable).
* **orderp**: This table includes prior orders. It has a composite primary key (**order\_id and product\_id**) and indicates whether a product in an order is a reorder or not (through the reordered variable).
* **products**: This table includes all products. It has a single primary key (**product\_id**)
* **aisles**: This table includes all aisles. It has a single primary key (**aisle\_id**)
* **departments**: This table includes all departments. It has a single primary key (**department\_id**)

We now use the head() function in order to visualise the first 10 rows of these tables. Click the Output button below to see the tables.

Output

In [3]:

head(orders,10)

head(ordert,10)

head(orderp,10)

head(products,10)

head(aisles,10)

head(departments)

2.2 Reshape data

We transform the data in order to facilitate their further analysis. First, we convert character variables into factors so we can use them in the creation of the model. In R, a categorical variable is called factor and has a fixed number of different values.

Output

In [4]:

*# We convert character variables into factor. In R, a categorical variable is called factor and has a fixed number of different values*

aisles$aisle <- as.factor(aisles$aisle)

departments$department <- as.factor(departments$department)

orders$eval\_set <- as.factor(orders$eval\_set)

products$product\_name <- as.factor(products$product\_name)

Moreover, in the **products** table we replace aisle\_id and department\_id with aisle name and department name.

In [5]:

*# In the products table we replace aisle\_id and department\_id with aisle name and department name*

products <- products %>%

inner\_join(aisles) %>% inner\_join(departments) %>%

select(-aisle\_id, -department\_id)

*# We delete the tables "aisles" and "departments" as we don't need them*

rm(aisles, departments)

*# Let's see the new products table*

head(products,10)

Joining, by = "aisle\_id"

Joining, by = "department\_id"

Finally, we add the column "user\_id" at the table **ordert** after matching the "order\_id" of this table with the "order\_id" of the table **orders**.

In [6]:

*# We add the column "user\_id" at the table ordert*

ordert$user\_id <- orders$user\_id[match(ordert$order\_id, orders$order\_id)]

*#Let's see ordert*

head(ordert,10)

## 2.3 Create an orders\_products table

We create a new table **orders\_products** which contains the tables **orders** and **orderp**. Bear in mind that **orderp** table includes only prior orders, so the new table **orders\_products** will also contain only these observations. Towards this end, we use inner\_join() function, which returns records that have matching values in both tables.

In [7]:

*# We create a new table "orders\_products" which contains the tables "orders" and orderp*

orders\_products <- orders %>% inner\_join(orderp, by = "order\_id")

*# We delete the table "orderp"*

rm(orderp)

gc()

*# Let's see the new table orders\_products*

head(orders\_products,10)

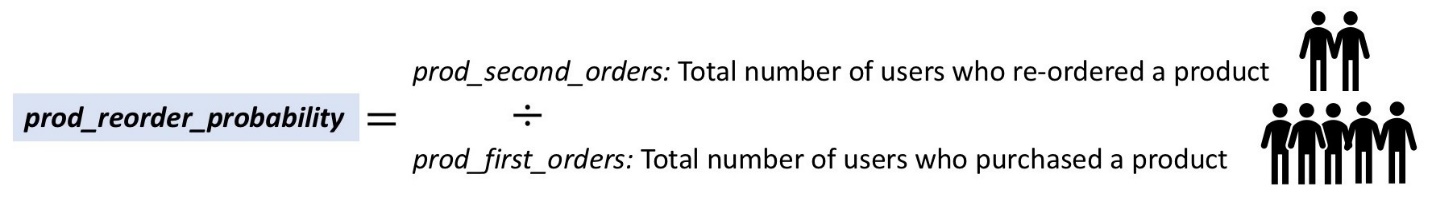
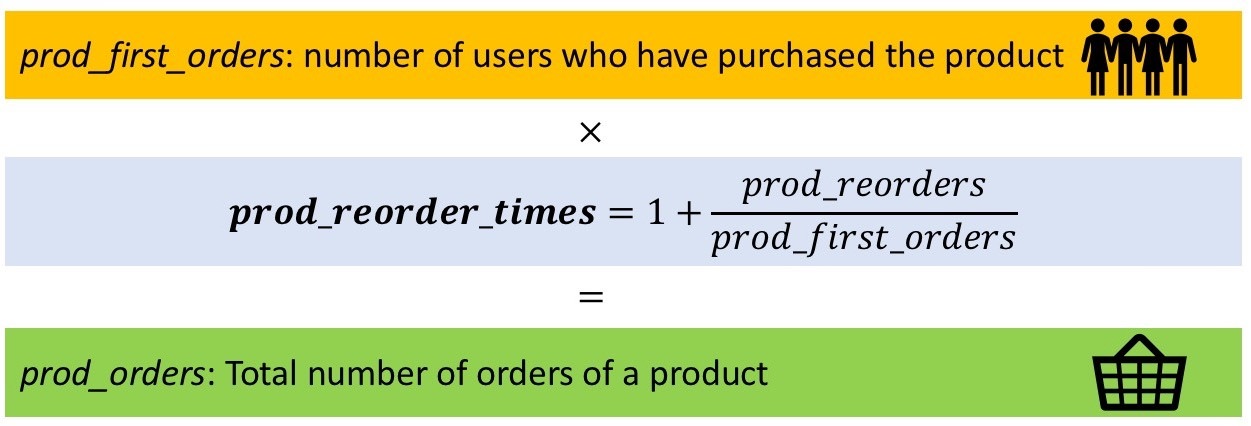
3. Create Predictor Variables

We are now ready to identify and calculate predictor variables based on the provided data. We can create various types of predictors such as:

* **Product predictors** describing characteristics of a product e.g. total number of times a product has been purchased.
* **User predictors** describing the behavior of a user e.g. total number of orders of a user.
* **User & product predictors** describing the behavior of a user towards a specific product e.g. the total times a user ordered a product.
* **Datetime predictors** describing temporal characteristics of the orders.

3.1. Product Predictors (prd table)

In this step, we create four new predictor variables that define **products** and we store them in a new table that is called **prd**. We have selected to include the following:

* **prod\_orders**: Total number of orders per product
* **prod\_reorder\_probability**: Probability a product is reordered after the first order
* **prod\_reorder\_times**: In average how many times a product has been purchased by the users who purchased it at least once
* **prod\_reorder\_ratio**: Reorders per total number of orders of the product

Although in our analysis we have selected to include only these four product predictors, one could come up with many other.

**What other product variables we could have included in our analysis?** Examples include:

* How often the product is purchased
* Position in the cart
* How many users buy it as "one shot" item
* Stats on the number of items that co-occur with this item
* Stats on the order streak
* Probability of being reordered within N orders
* Distribution of the day of week it is ordered
* Statistics around the time between orders

Going back to the selected predictors, we first need to calculate the following **supportive variables**:

* **prod\_reorders**: Total number of reorders per product
* **prod\_first\_orders**: Total number of first orders per product or total number of customers who bought a product
* **prod\_second\_orders**: Total number of second orders per product or total number of customers who bougth a product at least twice

Inline code comments explain how we handle the data. We start with the **orders\_products** and after executing the code below we get a **temporary version of the prd table** in which a new variable "product\_number" has been added to denote how many times a user purchased a product.

In [8]:

*# We create the prd and we start with the data inside the orders\_products table*

prd <- orders\_products %>%

*# We arrange() the three variables "user\_id", "order\_number", "product\_id" to sort them in descending order*

arrange(user\_id, order\_number, product\_id) %>%

*# We group\_by() to group together users and products*

group\_by(user\_id, product\_id) %>%

*# Mutate() creates the new variable "product time" through row\_number() which returns a sequential number starting at 1*

mutate(product\_time = row\_number()) %>%

*# We have now identified how many times a user bought a product*

ungroup()

*# let's see the temporary prd table*

head(prd,10)

By executing the following code we get a new temporary version of the prd table with the supportive variables we described above. This version will be used to create the final product variables.

In [9]:

prd <- prd %>%

*# We group data per product\_id*

group\_by(product\_id) %>%

*# We create the four new variables based on the groups we have created i.e. per product\_id*

summarise(

*# n() counts the objects inside the different groups of product\_id i.e. the total number of orders per product*

prod\_orders = n(),

*# Summarise the reordered variable per product\_id (recall that reordered is 1 or 0) i.e. the total number of reorders per product*

prod\_reorders = sum(reordered),

prod\_first\_orders = sum(product\_time == 1),

prod\_second\_orders = sum(product\_time == 2)

)

*# let's see the temporary prd table*

head(prd,10)

We are now ready to calcuate the final product predictors. After running the following code we get the **final prd table**.

In [10]:

*# we calculate the prod\_reorder\_probability variable*

prd$prod\_reorder\_probability <- prd$prod\_second\_orders / prd$prod\_first\_orders

*# we caclculate the prod\_reorder\_times variable*

prd$prod\_reorder\_times <- 1 + prd$prod\_reorders / prd$prod\_first\_orders

*# we caclculate the prod\_reorder\_ratio variable*

prd$prod\_reorder\_ratio <- prd$prod\_reorders / prd$prod\_orders

*# we remove the prod\_reorders, prod\_first\_orders, and prod\_second\_orders variables*

prd <- prd %>% select(-prod\_reorders, -prod\_first\_orders, -prod\_second\_orders)

*# we delete the products table*

rm(products)

gc()

*# Let's see the final prd table*

head(prd,20)

3.2 User Predictors (users table)

The next set of predictor variables that we calculate are related to **users** and we store them in the new **users table**. To calculate these variables we take into account only the **prior** orders.

We start by calculating three new variables:

* **user\_orders**: Total number of orders per user
* **user\_period**: The time period (in days) between the first and last order of a user
* **user\_mean\_days\_since\_prior**: Mean time period (in days) between two consequtive orders of a user

Inline code comments explain how we calculate these variables. We start by creating a temporary version of the **users** table from the **orders** table.

In [11]:

users <- orders %>%

*# We keep only the prior orders*

filter(eval\_set == "prior") %>%

*# We group orders by user\_id*

group\_by(user\_id) %>%

*# We calculate the variables based on different user\_id*

summarise(

*# We calculate the total number of orders per user using the order\_number variable.*

*# What other variable we could have used in order to calculate total number of orders?*

user\_orders = max(order\_number),

*# Using the na.rm = T we omit the missing values and calculate the sum and mean only for the values that we have*

user\_period = sum(days\_since\_prior\_order, na.rm = T),

user\_mean\_days\_since\_prior = mean(days\_since\_prior\_order, na.rm = T)

)

*# Let's see the temporary users table*

head(users,10)

We also create a supportive table, namely **us** in order to calculate three more new variables:

* **user\_total\_products**: Total numbers of basket items included in user's orders
* **user\_reorder\_ratio**: Reorder ratio (as defined above) per user
* **user\_distinct\_products**: Total number of distinct products ordered by a user

Towards this end, we start from the **orders\_products** table and we group observations using **user\_id** variable.

In [12]:

us <- orders\_products %>%

group\_by(user\_id) %>%

summarise(

user\_total\_products = n(),

user\_reorder\_ratio = sum(reordered == 1) / sum(order\_number > 1),

*# The n\_distinct() function counts the number of unique values in a set*

user\_distinct\_products = n\_distinct(product\_id)

)

*# Let's see the us table*

head(us,10)

Then we combine the users and us tables ussing **inner\_join()** function and we calculate the final variable:

* **user\_average\_basket**: Average number of basket items per order per user

In [13]:

*# We combine users and us tables and store the results into users table*

users <- users %>% inner\_join(us)

*# We calculate the user\_average\_basket variable*

users$user\_average\_basket <- users$user\_total\_products / users$user\_orders

*# let's see the users table*

head(users,10)

We now identify the future order per user and add them in the users table. The future orders are indicated as **train** and **test** in the eval\_set variable. As a result, we will know what is the order\_id of the future order per user, whether this order belongs in the train or test set, and the time in days since the last order.

In [14]:

us <- orders %>%

*# we exclude prior orders and thus we keep only train and test orders*

filter(eval\_set != "prior") %>%

select(user\_id, order\_id, eval\_set,

time\_since\_last\_order = days\_since\_prior\_order)

*# We combine users and us tables and store the results into the users table*

users <- users %>% inner\_join(us)

*# We delete the us table*

rm(us)

gc()

*# let's see the final users table*

head(users,10)

3.3 User x Product Predictors (data table)

We now create predictors that indicate how a user behaves towards a specific product. We store these predictors in the **data** table, which is also the final table that we create. Towards this end, we use both **prd** and **users** tables. We create the following predictors:

* **up\_orders**: The total times a user ordered a product
* **up\_first\_order**: What was the first time a user purchased a product
* **up\_last\_order**: What was the last time a user purchased a product
* **up\_average\_cart\_position**: The average position in a user's cart of a product
* **up\_order\_rate**: Percentage of user’s orders that include a specific product
* **up\_orders\_since\_last\_order**: Number of orders since user’s last order of a product
* **up\_order\_rate\_since\_first\_order**: Pecentage of orders since first order of a product in which a user purchased this product

Again we could have selected more predictors of this type. Examples include:

* Days since the user last purchased the product
* Number of orders in a row the user has purchased a product
* The longest that a user has ever gone without ordering a product

We create the **data** table starting from the **orders\_products** table and summarising using both **user\_id** and **product\_id**. We then create the four first new variables. After running the following code we get a temporary version of the **data** table.

data <- orders\_products %>%

group\_by(user\_id, product\_id) %>%

summarise(

up\_orders = n(),

up\_first\_order = min(order\_number),

up\_last\_order = max(order\_number),

up\_average\_cart\_position = mean(add\_to\_cart\_order))

*# We delete the tables "orders\_products" and "orders"*

rm(orders\_products, orders)

*# Let's see the temporary data table*

head(data, 10)

We combine the **data** table with the **prd** and **users** tables and we calculate the final three variables.

In [16]:

*# We use inner\_join() to combine the table "data" with the tables "prd" and "users"*

data <- data %>%

inner\_join(prd, by = "product\_id") %>%

inner\_join(users, by = "user\_id")

*# We calculate the variables "up\_order\_rate", "up\_orders\_since\_last\_order", "up\_order\_rate\_since\_first\_order"*

data$up\_order\_rate <- data$up\_orders / data$user\_orders

data$up\_orders\_since\_last\_order <- data$user\_orders - data$up\_last\_order

data$up\_order\_rate\_since\_first\_order <- data$up\_orders / (data$user\_orders - data$up\_first\_order + 1)

*# Let's see the temporary data table*

head(data, 10)

We now combine the **data** table with the **ordert** table in order to see which products that a user has already bought (prior) has been reordered (train).

If a product from **ordert** table has been reordered (reordered=1) by a user then this combination of product and user will exist in **data** table and thus the reordered value will be added in the **data** table through leftjoin(). If not then the specific combination of product and user will not exist in **data** table and thus the reordered value for this observation will remain empty (reordered=NA).

In [17]:

data <- data %>%

left\_join(ordert %>% select(user\_id, product\_id, reordered),

by = c("user\_id", "product\_id"))

*# We delete the tables "ordert", "prd" and "users"*

rm(ordert, prd, users)

gc()

*# Let's see the final data table*

head(data, 10)

4. Create the Train & Test tables

Before we are ready to run the XGBoost algorithm, it is necessary to create the final train and test tables that will feed the algorithm and the evaluation respectively.

* We split the data based on the eval\_set variable into train and test.
* We remove all the columns that are not predictors variables.
* In the train set we tranform the missing values (NA) of reordered to 0 in order to indicate that these products have not been reordered in the future order.

Output

In [18]:

train <- as.data.frame(data[data$eval\_set == "train",])

train$eval\_set <- NULL

train$user\_id <- NULL

*#train$product\_id <- NULL*

*#train$order\_id <- NULL*

*# below we transform missing values of 'reordered' variable to 0*

train$reordered[is.na(train$reordered)] <- 0

head(train,10)

In [19]:

test <- as.data.frame(data[data$eval\_set == "test",])

test$eval\_set <- NULL

test$user\_id <- NULL

test$reordered <- NULL

head(test,10)

rm(data)

gc()

5. Create the Model

We use the train data to create the model. Each variable is a list containing two things, label (or outcome) and data (predictors). In our example the label we want to predict is the column "reordered".

Because xgboost manages only numeric vectors we have to transform the categorical data to dummy variables while building the model we will need. Towards this end we use xgb.DMatrix.

As you can see there is a list of parameters which are necessary to train our decision tree model These are the following:

* **objective:** You need to specify the type of learner you want which includes linear regression, logistic regression, poisson regression etc
* **eval\_metric:** You need to specify the evaluation metrics for validation data, a default metric will be assigned according to objective (rmse for regression, and error for classification, mean average precision for ranking)
* **eta:** The default value is set to 0.3. You need to specify step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features and eta actually shrinks the feature weights to make the boosting process more conservative. The range is 0 to 1. Low eta value means model is more robust to overfitting
* **max\_depth:** The default value is set to 6. You need to specify the maximum depth or splits of a tree. The range is 1 to ∞
* **min\_child\_weight:** The default value is set to 1. You need to specify the minimum sum of instance weight(hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min\_child\_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. The range is 0 to ∞
* **gamma:** The default value is set to 0. You need to specify minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be. The range is 0 to ∞
* **subsample:** The default value is set to 1. You need to specify the subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting. The range is 0 to 1
* **colsample\_bytree:** The default value is set to 1. You need to specify the subsample ratio of columns when constructing each tree. The range is 0 to 1
* **alpha:** These are regularization term on weights. Αlpha default value assumed is 0
* **lambda:** These are regularization term on weights. Lambda default value assumed is 1
* **nround:** The number of trees to the model

A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model. Feature importance in Xgboost can be done through xgb.importance(). A table having all the variables and their gain, cover and frequency is the result of this feature. Gain is the improvement in accuracy brought by a feature to the branches it is on. Cover measures the relative quantity of observations concerned by a feature. Frequency is a simpler way to measure the Gain. It just counts the number of times a feature is used in all generated trees. You should not use it (unless you know why you want to use it). The column Gain provide the information we are looking for.

Finally we can plot the results from our model using xgb.plot.importance()

library(xgboost)

params <- list(

"objective" = "reg:logistic",

"eval\_metric" = "logloss",

"eta" = 0.1,

"max\_depth" = 6,

"min\_child\_weight" = 10,

"gamma" = 0.70,

"subsample" = 0.76,

"colsample\_bytree" = 0.95,

"alpha" = 2e-05,

"lambda" = 10

)

*# We get a sample containing 10% of the train table*

subtrain <- train %>% sample\_frac(0.1)

*# We create an xgb.DMatrix that is named X with predictors from subtrain table and response the reordered variable*

X <- xgb.DMatrix(as.matrix(subtrain %>% select(-reordered, -order\_id, -product\_id)), label = subtrain$reordered)

*# We create the actual model*

model <- xgboost(data = X, params = params, nrounds = 80)

In [21]:

*# We estimate the importance of the predictors*

importance <- xgb.importance(colnames(X), model = model)

*# We plot the importance of the predictors*

xgb.ggplot.importance(importance)

rm(X, importance, subtrain)

gc()

# 6. Apply the model

Because Instacart did not reveal the outcome of the test data we also apply the model in the train data and create a table that enables comparing the predicted to the actural results.

## 6.1 Apply the model in test data

Now we are ready to perform the prediction with the model we have built to classify test data.

In [22]:

*# We use the xgb.DMatrix to group our test data into a matrix*

X <- xgb.DMatrix(as.matrix(test %>% select(-order\_id, -product\_id)))

*# We apply the model and we predict the reordered variable for the test set.*

test$reordered <- predict(model, X)

*# The model estimates a probability. We apply a threshold so every prediction above 0.21 will be considered as a reorder (reordered=1)*

test$reordered <- (test$reordered > 0.21) \* 1

*# We create the final table with reordered products per order*

submission <- test %>%

filter(reordered == 1) %>%

group\_by(order\_id) %>%

summarise(

products = paste(product\_id, collapse = " ")

)

*# Let's see the submission table*

head(submission,10)

*# We create the table "missing" where we have the orders in which none product will be ordered according to our prediction*

missing <- data.frame(

order\_id = unique(test$order\_id[!test$order\_id %in% submission$order\_id]),

products = "None"

)

submission <- submission %>% bind\_rows(missing) %>% arrange(order\_id)

*#Let's see the submission table*

head(submission,10)

Bear in mind that Instacart has not revealed the response value of the test observations. As a result, we cannot evaluate our actual prediction.

In order to be able to compare our prediction to the actual result we re-apply the model in the train data (see below)

## 6.2 Apply the model in train data

Now we are ready to perform the prediction with the model we have built to classify train data.

In [24]:

*# We use the xgb.DMatrix to group our train data into a matrix*

X <- xgb.DMatrix(as.matrix(train %>% select(-order\_id, -product\_id, -reordered)))

*# We apply the model and we predict the reordered variable for the train set.*

train$reordered\_pred <- predict(model, X)

*# The model estimates a probability. We apply a threshold so every prediction above 0.21 will be considered as a reorder (reordered=1)*

train$reordered\_pred <- (train$reordered\_pred > 0.21) \* 1

*# We create the final table with reordered products per order*

submission\_train <- train %>%

filter(reordered\_pred == 1) %>%

group\_by(order\_id) %>%

summarise(

products = paste(product\_id, collapse = " ")

)

real\_reorders <- train %>%

filter(reordered == 1) %>%

group\_by(order\_id) %>%

summarise(

real\_products = paste(product\_id, collapse = " ")

)

submission\_train <- real\_reorders %>%

inner\_join(submission\_train, by = "order\_id")

*# Let's see the submission table*

head(submission\_train,10)