CAL Project - Knowledge Transfer Documentation - Data Engineering 4.1 - Member Retention

4/28/2020

This document outlines the data engineering steps that the CAL team performed in order to prepare the data for **Member Retention predictive modeling**. This file primarily covers Membership and Engagement data engineering while pre-engineered Events and Email files are simply loaded and joined. Please see file: "4.0 - Predictive Model Emails and Events" for the Email and Events file data engineering.

Running this R script in its entirety will produce two files:

- **Data_member_retention_future_training_final.csv**
  - This file will be used to train the predictive model. It will leverage data from last year and two years ago.
  - Ex. If it is Jul, 1 2020 this file will use FY19 to train the model and final membership information from FY20 to measure these predictions.
- **Data_member_retention_future_prediction_final.csv**
  - This file will be used in predictive modeling scripts, leveraging the most recent year's membership data as inputs to make predictions for the following year.
  - Ex. If it is Jul 1, 2020 this file will use final data from FY20 to make prediction for FY21.

We set the working directory

```
## set working directory
setwd('D:/Group Folder/SEQUENTIAL FILES')
```

Load required packages and initialize global FY variable.

```
## Libraries used
library(tidyverse)
library(lubridate)
library(data.table)
library(dplyr)

## GLOBAL VARIABLES

# current fiscal year
current_fy = ifelse( month(Sys.Date()) > 6, year(Sys.Date()) + 1,
                     year(Sys.Date()))
```

Membership

Read the data

```
indy_info <- read_csv('2 Data Cleaned Files/individual_info_cleaned.csv')
membership_old <- read_csv('2 Data Cleaned Files/membership_cleaned.csv')
```

A - Initial Member engineering to expand data across years

First we filter the membership file to contain only the past five years of data. This will be referred to as "the five year window" or "the window", in subsequent explanation.

```
# Create membership df to capture last 5 year window
membership = membership_old %>%
  filter((FISCAL_YEAR > current_fy - 6 & FISCAL_YEAR < current_fy)) %>%
  select(ID_DEMO, APPEAL_CODE, AMOUNT_PAID, FISCAL_YEAR, MEMBERSHIP_LEVEL,
ID_OF_RECORD_WITH_RELATIONSHIP) %>%
  mutate(life_before_window = 0)
```

The below code is used to ensure that each ID retained in the membership data frame contains a row for each year in the five year window.

```
# Spreading per person
per_indy <-
  reshape2::dcast(data = membership,
                  ID_DEMO ~ FISCAL_YEAR,
                  value.var = 'AMOUNT_PAID',
                  fun=sum)

# Gathering back
new_membership <-
  per_indy %>%
  gather(key = "FISCAL_YEAR",
         value = 'AMOUNT_PAID',
         -ID_DEMO) %>%
  mutate(FISCAL_YEAR = as.numeric(FISCAL_YEAR))
```

```
# Person + year combo
membership_join <- new_membership %>% left_join(membership, by = c('ID_DEMO',
'FISCAL_YEAR', 'AMOUNT_PAID'))
```

ID_DEMO	FISCAL_YEAR	AMOUNT_PAID	APPEAL_CODE	MEMBERSHIP_LEVEL
FFFF328D53DC4859E074AECCBC89E0ED	2015	50.0	UM15AAMR15K6	UMAA Annual Membership (Single)
FFFF328D53DC4859E074AECCBC89E0ED	2016	50.0	UW16AAMR16WB	UMAA Annual Membership (Single)
FFFF328D53DC4859E074AECCBC89E0ED	2017	50.0	UM17AAMR17A3	UMAA Annual Membership (Single)
FFFF328D53DC4859E074AECCBC89E0ED	2018	50.0	UM17AAMR17A3	UMAA Annual Membership (Single)
FFFF328D53DC4859E074AECCBC89E0ED	2019	50.0	UMXXAAMR30DP	Annual Membership 1 Year (Single)
FFFEB1F33CF0BF0A34F6CA04B8CC6ED3	2015	55.0	UM15AAMR15C6	UMAA Annual Membership (Joint)
FFFEB1F33CF0BF0A34F6CA04B8CC6ED3	2016	0.0	UM16AAMR16C3	UMAA Annual Membership (Joint Secondary)
FFFEB1F33CF0BF0A34F6CA04B8CC6ED3	2017	0.0	*NA*	*NA*
FFFEB1F33CF0BF0A34F6CA04B8CC6ED3	2018	0.0	*NA*	*NA*
FFFEB1F33CF0BF0A34F6CA04B8CC6ED3	2019	0.0	*NA*	*NA*
FFFE767EFA22DAAA6C2FC8EA685F5ACF	2015	50.0	UM15AAMR15K6	UMAA Annual Membership (Single)
FFFE767EFA22DAAA6C2FC8EA685F5ACF	2016	0.0	*NA*	*NA*
FFFE767EFA22DAAA6C2FC8EA685F5ACF	2017	0.0	*NA*	*NA*
FFFE767EFA22DAAA6C2FC8EA685F5ACF	2018	0.0	*NA*	*NA*
FFFE767EFA22DAAA6C2FC8EA685F5ACF	2019	50.0	UM19AAMF19DL	UMAA Annual Membership (Single)

B - Merging Individual Info

The goal of the below code is to obtain a data frame of all lifetime members who became life members before the beginning of the 5 year window that this file is using. After getting their IDs we create records for all five years of the window for each ID.

```
# Find all current lifetime members that purchased memberships before the five
year window
ind_info_lt = indy_info %>%
    filter(MEMBERSHIP_TYPE_CODE == "L") %>%
    select(ID_DEMO) %>%
    filter(!(ID_DEMO %in% membership$ID_DEMO))

# Place them in a DF
records_minus5 = ind_info_lt %>% mutate(FISCAL_YEAR = current_fy-5)
records_minus4 = ind_info_lt %>% mutate(FISCAL_YEAR = current_fy-4)
records_minus3 = ind_info_lt %>% mutate(FISCAL_YEAR = current_fy-3)
records_minus2 = ind_info_lt %>% mutate(FISCAL_YEAR = current_fy-2)
records_LastYear = ind_info_lt %>% mutate(FISCAL_YEAR = current_fy-1)

records_new =
rbind(records_minus5,records_minus4,records_minus3,records_minus2,records_lastYe
ar) %>%
  mutate(APPEAL_CODE = NA,
         AMOUNT_PAID = NA,
         MEMBERSHIP_LEVEL = NA,
         ID_OF_RECORD_WITH_RELATIONSHIP = NA,
         life_before_window = 1)
```

Finally we combine these members who became life members before the window with those who had membership data in the window.

```
# Add these members who became life members before the 5 year window to the data
master data frame
membership_join = rbind(membership_join, records_new)
```

C - Feature Engineering

We begin by categorizing the member status using simple text matching on the MEMBERSHIP_LEVEL column. This provides us the new column: member_status. It can contain three values:

- non-member
- annual-member
- life-member

```
# Annual or lifetime or non-member (accounting for multi-year memberships and
lifetime purchases in later code chunks)
membership_join <- membership_join %>%
  mutate(member_status = ifelse(is.na(MEMBERSHIP_LEVEL), 'non-member',
                                ifelse(str_detect(MEMBERSHIP_LEVEL, 'Life'),
'life-member', 'annual-member')))
```

We also included a joint membership flag. This feature was not actively used in our analysis.

```
# Joint memberships
membership_join <- membership_join %>% mutate(joint =
ifelse(is.na(ID_OF_RECORD_WITH_RELATIONSHIP), 0, 1))
```

Next we use text matching here to determine the length of the annual membership that was purchased, assigning it to ann_num_years.

```
# How many years of Annual?
regexp <- "[[:digit:]]+"

# First we assign ann_num_years the numeric value we have extracted from the
MEMBERSHIP_LEVEL column if and only if there is a numeric value present
membership_join <- membership_join %>% mutate(ann_num_years =
str_extract(MEMBERSHIP_LEVEL, regexp))

# Next we use the ann_num_years column to determine the numeric value to assign.
# a: If ann_num_years is NA (meaning no digit was found in MEMBERHSIP_STATUS)
and the
#    word "Annual" was found in MEMEBRSHIP_STATUS then we classify the purchase
as a
#    one year annual membership
# b: If "Life" is found in membership status then we classify as 0 annual years
# c: If the word "Annual" was found and ann_num_years is not NA then we know
there was
#    a multi-year purchase made and we assign that value to ann_num_years
membership_join <- membership_join %>%
mutate(ann_num_years = ifelse(is.na(ann_num_years) &
grepl("Annual",MEMBERSHIP_LEVEL), 1,
ifelse(grepl("Life",MEMBERSHIP_LEVEL) | is.na(MEMBERSHIP_LEVEL), 0,
ann_num_years)))
```

```
membership_join$ann_num_years = as.numeric(membership_join$ann_num_years)
```

D - Accounting for Multi-year purchase types in member_status

The following code uses the above ann_num_years feature to ensure the multi-year memberships are factored in. More specifically, it labels the member who purchased the multi-year membership as an "annual-member" (member_status column) for the duration of their purchase even though the only record they appear in in the membership table is the year of their multi-year purchase.

```
## Annual:
# Add new columns indicating members' 2,3,4,5 year purchases carying forward
membership_join = membership_join %>% group_by(ID_DEMO) %>% arrange(FISCAL_YEAR)
%>%
  mutate(prev1 = lag(ann_num_years)) %>%
  mutate(prev2 = lag(ann_num_years,2)) %>%
  mutate(prev3 = lag(ann_num_years,3)) %>%
  mutate(prev4 = lag(ann_num_years,4))

# Replace NA's with zero for mutation step to work
membership_join$prev1 = membership_join$prev1 %>% replace_na(0)
membership_join$prev2 = membership_join$prev2 %>% replace_na(0)
membership_join$prev3 = membership_join$prev3 %>% replace_na(0)
membership_join$prev4 = membership_join$prev4 %>% replace_na(0)

# Fill the member_status column based on multi-year purchases
membership_join = membership_join %>% mutate(member_status = ifelse((prev1 == 2
| prev1 == 3 | prev1 == 4 | prev1 == 5), "annual-member", member_status))
membership_join = membership_join %>% mutate(member_status = ifelse((prev2 == 3
| prev2 == 4 | prev2 == 5), "annual-member", member_status ))
membership_join = membership_join %>% mutate(member_status = ifelse((prev3 == 4
| prev3 == 5), "annual-member", member_status ))
membership_join = membership_join %>% mutate(member_status = ifelse((prev4 ==
5), "annual-member", member_status ))

# Remove columns used in this engineering
membership_join = membership_join %>% select(-prev1, -prev2, -prev3, -prev4)
```

Similarly we must ensure that life members who made a purchase are labeled as "life-members" (member_status column) for the duration of the window after they make their membership purchase.

```
## Life:
# Fill Lifetime members' member_status with "life-member" for years after they
made their lifetime purchase
membership_join = membership_join %>% group_by(ID_DEMO) %>% arrange(FISCAL_YEAR)
%>%
  mutate(prev1 = lag(member_status)) %>%
  mutate(prev2 = lag(member_status,2)) %>%
  mutate(prev3 = lag(member_status,3)) %>%
  mutate(prev4 = lag(member_status,4))
```

```
# Replace NA's with zero for mutation step to work
membership_join$prev1 = membership_join$prev1 %>% replace_na(0)
membership_join$prev2 = membership_join$prev2 %>% replace_na(0)
membership_join$prev3 = membership_join$prev3 %>% replace_na(0)
membership_join$prev4 = membership_join$prev4 %>% replace_na(0)

# Fill the member_status column based on lifetime purchases
membership_join = membership_join %>%
  mutate(member_status = ifelse((prev1 == "life-member" |
                                   prev2 == "life-member" |
                                   prev3 == "life-member" |
                                   prev4 == "life-member"), "life-member",
member_status))

# Remove columns used in this engineering
membership_join = membership_join %>% select(-prev1, -prev2, -prev3, -prev4)
```

E - Rollup Prep

This section prepares the data for rolling up. We first remove any duplicate rows that may exist, ensuring that each ID has five rows reflecting their membership status in each year of the five year window.

```
# Remove duplicates
membership_join = membership_join %>% distinct(ID_DEMO, FISCAL_YEAR, .keep_all =
TRUE)
```

Next we go back to our life members who purchased their membership before the five year window. This section of code ensures that their member_status is "life-member" for the duration of the window.

```
# Life members before window
membership_join$life_before_window = membership_join$life_before_window %>%
replace_na(0)
membership_join = membership_join %>% mutate(member_status =
ifelse(life_before_window == 1, "life-member", member_status))
```

Finally, we add a life_year column for rollup to see how many of the five years they have been life-members.

```
# Add a life_year column for rollup
membership_join <- membership_join %>% mutate(life_yr =
ifelse(grepl("Life",MEMBERSHIP_LEVEL), FISCAL_YEAR, 0))
```

Finally we make sure that rows reflecting life installment payments are not incurring annual years even though they contain the word Annual in their title.

```
# Handle installments
membership_join = membership_join %>%
  mutate(ann_num_years = ifelse(grepl("Install",MEMBERSHIP_LEVEL), 0,
ann_num_years))
```

F - Rollup

In this section we roll up the membership_join data frame.

This transforms our data frame:

- FROM: One row per ID per year indicating their member_status in that year (five records per ID total)
- TO: One row per ID with rollup columns indicating the number of years they were each member_status (one row per ID)

We begin by setting section specific variables.

```
TARGET_YR_GLOBAL = current_fy - 1
TRAIN_DATA_YRS = 4
```

Next we obtain the values we will need to filter and train the predictive models. Knowing their membership status in, for example, 2018 and 2019 will allow us to understand exactly which records made membership changes. i.e. if membership_TwoYearsAgo = "non-member" and membership_LastYear = "annual-member" we know that this person purchased a membership in 2019.

```
# Get the target values
membership_LastYear = membership_join %>% filter(FISCAL_YEAR == current_fy - 1)
%>%
  rename(membership_LastYear = member_status) %>%
  select(ID_DEMO, membership_LastYear)

membership_TwoYearsAgo = membership_join %>% filter(FISCAL_YEAR == current_fy -
2) %>%
  rename(membership_TwoYearsAgo = member_status) %>%
  select(ID_DEMO, membership_TwoYearsAgo)
```

Next we make some adjustments to not include the target column.

```
# Remove membership_LastYear to not include it in the counts
membership_join_rollup = membership_join %>% filter(FISCAL_YEAR != current_fy -
1)

# Add non-mem years
membership_join_rollup = membership_join_rollup %>% mutate(nonmem_num_years =
ifelse(member_status == "non-member", 1, 0))
```

We then aggregate the to obtain the number of years each member was annual, life and non-member.

```
# aggregate
membership_rollup = membership_join_rollup %>% group_by(ID_DEMO) %>%
  summarise(annual_years = sum(as.numeric(ann_num_years)),
            life_years = TARGET_YR_GLOBAL - sum(life_yr),
            nonmem_years = sum(nonmem_num_years),
            life_before_window_flag = sum(life_before_window))

membership_rollup = membership_rollup %>%
    mutate(annual_years = ifelse(annual_years > TRAIN_DATA_YRS, TRAIN_DATA_YRS,
 annual_years),
           life_years = ifelse(life_years == TARGET_YR_GLOBAL, 0, life_years),
           life_years = ifelse(life_before_window_flag == TRAIN_DATA_YRS,
4,life_years))
```

We then join the two columns back to the rolled up data frame

```
# Join the LastYear and TwoYearsAgo member_status columns
membership_rollup = membership_rollup %>% left_join(membership_LastYear, by =
"ID_DEMO")
membership_rollup = membership_rollup %>% left_join(membership_TwoYearsAgo, by =
"ID_DEMO")
```

Next, we perform clean up on the membership_rollup data frame.

```
# Confirm anyone with life years is classified as a life member in target
variable
membership_rollup = membership_rollup %>% mutate(membership_LastYear =
ifelse(life_years == 0, membership_LastYear, "life-member"))

# Adjust rows that are not calculating life_years correctly (due to
installments)
membership_rollup = membership_rollup %>%
  mutate(life_years = ifelse(life_years < 0 | life_years > 4,
                             (TRAIN_DATA_YRS - nonmem_years - annual_years),
                             life_years))

# Remove columns used in this engineering
membership_rollup = membership_rollup %>% select(-life_before_window_flag)
```

We then create a separate data frame for the prediction and training files. In the prediction file we make an adjustment to include the most recent year's membership status in the "number of years" rollups.

```
# Create separate data frame for prediction. This now ill include the most
#   recent year in rollups based on their most recent status
membership_rollup_prediction = membership_rollup %>%
  mutate(annual_years = ifelse(membership_LastYear == "annual-member",
                               annual_years + 1 ,
                               annual_years)) %>%
  mutate(life_years = ifelse(membership_LastYear == "life-member",
                               life_years + 1 ,
                               life_years)) %>%
  mutate(nonmem_years = ifelse(membership_LastYear == "non-member",
                               nonmem_years + 1 ,
                               nonmem_years))
```

Finally, we use prepare these data frames for use in the following sections.

```
# write to this DF that will be used in join operations below
member = membership_rollup
member_prediction = membership_rollup_prediction
```

Emails and Events

This section first reads in the output of the "4.0 - Predictive Model Emails and Events" script placing it in the emails_events_member data frame.

We then join this data frame to the final data frame produced in the Membership section above.

```
# Emails & Events ----------------------------------------------------------
member_filter <- member %>% select(ID_DEMO)
member_prediction_filter <- member_prediction %>% select(ID_DEMO)

emails_events <- read_csv('3 Data Generated Files/pred_emails_events.csv')

emails_events_member <-
  member %>%
  left_join(emails_events, by = c('ID_DEMO')) %>%
  replace(is.na(.), 0) %>%
  ungroup()

emails_events_member_prediction <-
  member_prediction %>%
  left_join(emails_events, by = c('ID_DEMO')) %>%
  replace(is.na(.), 0) %>%
  ungroup()
```

Engagement

First, load the cleaned engagement file into a data frame.

```
eng_cleaned <- read.csv("2 Data Cleaned Files/engagement_cleaned.csv",
                        header = TRUE)
```

This code calculates **Average UMN Annual sub engagement scores** for last 5 years for each Alumni. **Filter** function selects only the last 5 Fiscal years and excludes a Fiscal year based on the purpose of the data frame. **Select** function selects only the required columns like ID_DEMO, 6 Annual sub engagement scores and total Annual score.

After this calculation each Alumni has one record that has Average UMN Annual sub engagement scores for the last 5 years.

This data frame is used in the data for training Membership retention model. For example, if Current Fiscal year = 2020, average scores from 2014 to 2018 are used to train the model on the outcome of 2019.

```
UMN_last_5_avg_training <- eng_cleaned %>%
           filter( (YEAR_FISCAL >= previous_fy - 5) & (YEAR_FISCAL !=
previous_fy) ) %>%
                          select(c(1,3, 4 ,5, 6,7,8, 9)) %>%
                          group_by(ID_DEMO) %>%
                          summarise(UMN_event = mean(EVENT_ATTEND_ANNUAL),
                                    UMN_member = mean(UMAA_MEMBER_ANNUAL),
                                    UMN_donor = mean(DONOR_ANNUAL),
                                    UMN_volun = mean(VOLUNTEER_ANNUAL),
                                    UMN_inform = mean(STAY_INFORMED_ANNUAL),
                                    UMN_loyalty = mean(LOYALTY_ANNUAL),
                                    UMN_avg_Annual_score_5_years =
mean(ENGAGEMENT_TOTAL_ANNUAL))
```

This data frame is used in the data for future prediction of Membership retention model. For example, if Current Fiscal year = 2020, Since we don't have engagement scores for 2020, average scores from 2015 to 2019 are used in the model to predict the outcome in 2020.

```
UMN_last_5_avg_future_pred <- eng_cleaned %>%
              filter( (YEAR_FISCAL >= current_fy - 5) & (YEAR_FISCAL !=
current_fy) ) %>%
                            select(c(1,3, 4 ,5, 6,7,8, 9)) %>%
                            group_by(ID_DEMO) %>%
                            summarise(UMN_event = mean(EVENT_ATTEND_ANNUAL),
                                      UMN_member = mean(UMAA_MEMBER_ANNUAL),
                                      UMN_donor = mean(DONOR_ANNUAL),
                                      UMN_volun = mean(VOLUNTEER_ANNUAL),
                                      UMN_inform = mean(STAY_INFORMED_ANNUAL),
                                      UMN_loyalty = mean(LOYALTY_ANNUAL),
                                      UMN_avg_Annual_score_5_years =
mean(ENGAGEMENT_TOTAL_ANNUAL))
```

Next step is to join engagement scores with emails_events_membership data frame.

```
# join engagemnet scores with emails_events_membership dataframe - training
training_df <- UMN_last_5_avg_training %>%
                    inner_join(emails_events_member, by = "ID_DEMO")

# join engagemnet scores with emails_events_membership dataframe - future
prediction
future_prediction_df <- UMN_last_5_avg_future_pred %>%
                        inner_join(emails_events_member_prediction, by =
"ID_DEMO")
```

This code saves the **training_df** and **future_prediction_df** data frame into a csv files which are used for Membership Retention predictive models.

```
write.csv(training_df, "3 Data Generated
Files/Data_member_retention_training_final.csv", row.names = FALSE)
write.csv(future_prediction_df, "3 Data Generated
Files/Data_member_retention_future_prediction_final.csv", row.names = FALSE)
```