model01

Eyal Grinberg & Yam Rozen

2023-08-20

Libraries

```r
library(tidyverse)
library(dplyr)
library(tidytext)
library(tm)
library(stringdist)
library(imager)
library(textstem)
library(reticulate)
library(tidymodels)
library(textrecipes)
```

Reading Data

```r
data_food_train <- read_csv("data/food_train.csv")
data_nutrients <- read_csv("data/nutrients.csv")
data_food_nutrients <- read_csv("data/food_nutrients.csv")

# TEST - use for predictions later
data_food_test <- read_csv("data/food_test.csv")
```

NA values

```r
# After visualizing NA values (see section 1.1 in additional_material) we see that only
# data_food_train and data_food_test contain NA values, let's replace them with "NA".

data_food_train$ingredients <- data_food_train$ingredients %>% replace_na("NA ingredients")
data_food_train$household_serving_fulltext <- data_food_train$household_serving_fulltext %>%
  replace_na("NA household")

# TEST - use for predictions later
data_food_test$ingredients <- data_food_test$ingredients %>%
  replace_na("NA ingredients")
data_food_test$brand <- data_food_test$brand %>% replace_na("NA brand")
data_food_test$household_serving_fulltext <- data_food_test$household_serving_fulltext %>%
  replace_na("NA household")
```

Clean all textual data

```r
text_clean <- function(x)
{
  x  =  gsub("<.*?>", " ", x)
  x  =  iconv(x, "latin1", "ASCII", sub="")
  x  =  gsub("[^[:alnum:]]", " ", x)
  x  =  tolower(x)
  x  =  stripWhitespace(x)
  x  =  gsub("^\\s+|\\s+$", "", x)
  return(x)
}
```

```r
df_cleaned_text <- data_food_train %>% map_df(text_clean) %>%
  select(brand, description, ingredients, household_serving_fulltext)

# TEST - use for predictions later
test_df_cleaned_text <- data_food_test %>% map_df(text_clean) %>%
  select(brand, description, ingredients, household_serving_fulltext)
```

Household_serving

```r
# for the household_serving_fulltext variable we want to drop the digits

df_cleaned_text$household_serving_fulltext <- str_replace_all(
  df_cleaned_text$household_serving_fulltext, "\\d", "")

# TEST - use for predictions later
test_df_cleaned_text$household_serving_fulltext <- str_replace_all(
  test_df_cleaned_text$household_serving_fulltext, "\\d", "")
```

```r
# lemmatizing the household_serving_fulltext variable

length(unique(data_food_train$household_serving_fulltext)) # 2207 before cleaning
```

```
## [1] 2207
```

```r
df_cleaned_text$household_lemma <- lemmatize_strings(
  df_cleaned_text$household_serving_fulltext)
length(unique(df_cleaned_text$household_lemma)) #600 after cleaning
```

```
## [1] 600
```

```r
# TEST - use for predictions later
test_df_cleaned_text$household_lemma <- lemmatize_strings(
  test_df_cleaned_text$household_serving_fulltext)
```

Description

```r
data_food_train <- data_food_train %>% mutate_at(vars(
  c(brand, description, ingredients, household_serving_fulltext)), text_clean)
data_train_tokenized_description <- data_food_train %>%
  unnest_tokens(word, description) %>% count(category, word, sort = TRUE) %>%
  group_by(category) %>% mutate(total = sum(n))
data_train_tokenized_description <- data_train_tokenized_description %>%
  bind_tf_idf(word, category, n)
head(data_train_tokenized_description)
```

```
## # A tibble: 6 x 7
## # Groups:   category [5]
##   category                          word         n total     tf   idf  tf_idf
##   <chr>                             <chr>    <int> <int>  <dbl> <dbl>   <dbl>
## 1 cookies_biscuits                  cookies   3155 24331 0.130  0     0
## 2 chocolate                         chocola~  3142 19337 0.162  0     0
## 3 candy                             candy     2825 33870 0.0834 0     0
## 4 chips_pretzels_snacks             chips     2516 18213 0.138  0     0
## 5 chips_pretzels_snacks             potato    1387 18213 0.0762 0     0
## 6 popcorn_peanuts_seeds_related_snacks roasted   1334 34110 0.0391 0.182 0.00713
```

```r
# We want to focus on the distinguishing words (those with high tf-idf values)
threshold <- quantile(data_train_tokenized_description$tf_idf, probs = 0.9)

# Subset the dataframe for the top 10% values
df_top_10_percent_tf_idf <- data_train_tokenized_description %>%
  filter(tf_idf >= threshold)

distinguishing_words_by_category <- split(df_top_10_percent_tf_idf,
                                          df_top_10_percent_tf_idf$category)

distinguishing_words_cakes <-
  distinguishing_words_by_category$cakes_cupcakes_snack_cakes$word
distinguishing_words_candy <- distinguishing_words_by_category$candy$word
distinguishing_words_chips <-
  distinguishing_words_by_category$chips_pretzels_snacks$word
distinguishing_words_chocolate <- distinguishing_words_by_category$chocolate$word
distinguishing_words_cookies <-
  distinguishing_words_by_category$cookies_biscuits$word
distinguishing_words_popcorn <-
  distinguishing_words_by_category$popcorn_peanuts_seeds_related_snacks$word
head(distinguishing_words_candy)
```

```
## [1] "gummi"    "jelly"    "gummy"    "chewy"    "bears"    "assorted"
```

```r
# a function that receives a textual feature and a vector of distinguishing
# words of a category and calculates for each string in the feature how many
# appearances of distinguishing words exist in the string.

terms_score_func <- function(feature, terms_vec){
  tmp <- lapply(feature, PlainTextDocument)
  res <- sapply(tmp, tm_term_score, terms_vec)
  return(res)
}

# we will add the following variables to the features matrix
```

```r
description_candy_score <- terms_score_func(
  data_food_train$description, distinguishing_words_candy)
description_cakes_score <- terms_score_func(
  data_food_train$description, distinguishing_words_cakes)
description_chips_score <- terms_score_func(
  data_food_train$description, distinguishing_words_chips)
description_chocolate_score <- terms_score_func(
  data_food_train$description, distinguishing_words_chocolate)
description_popcorn_score <- terms_score_func(
  data_food_train$description, distinguishing_words_popcorn)
description_cookies_score <- terms_score_func(
  data_food_train$description, distinguishing_words_cookies)
head(description_cookies_score)
```

```
## [1] 0 1 1 3 1 1
```

```r
# TEST - use for predictions later
test_description_candy_score <- terms_score_func(
  test_df_cleaned_text$description, distinguishing_words_candy)
test_description_cakes_score <- terms_score_func(
  test_df_cleaned_text$description, distinguishing_words_cakes)
test_description_chips_score <- terms_score_func(
  test_df_cleaned_text$description, distinguishing_words_chips)
test_description_chocolate_score <- terms_score_func(
  test_df_cleaned_text$description, distinguishing_words_chocolate)
test_description_popcorn_score <- terms_score_func(
  test_df_cleaned_text$description, distinguishing_words_popcorn)
test_description_cookies_score <- terms_score_func(
  test_df_cleaned_text$description, distinguishing_words_cookies)
```

Ingredients

```r
data_train_tokenized_ingredients <- data_food_train %>%
  unnest_tokens(word, ingredients) %>% count(category, word, sort = TRUE) %>%
  group_by(category) %>% mutate(total = sum(n))
data_train_tokenized_ingredients <- data_train_tokenized_ingredients %>%
  bind_tf_idf(word, category, n)
head(data_train_tokenized_ingredients)
```

```
## # A tibble: 6 x 7
## # Groups:   category [3]
##   category                            word      n  total     tf   idf tf_idf
##   <chr>                               <chr> <int>  <int>  <dbl> <dbl>  <dbl>
## 1 cakes_cupcakes_snack_cakes          and   11004 385760 0.0285     0      0
## 2 cakes_cupcakes_snack_cakes          oil   10557 385760 0.0274     0      0
## 3 cookies_biscuits                    flour 10425 296895 0.0351     0      0
## 4 popcorn_peanuts_seeds_related_snacks oil    9788 206374 0.0474     0      0
## 5 cakes_cupcakes_snack_cakes          flour  9737 385760 0.0252     0      0
## 6 cookies_biscuits                    sugar  9643 296895 0.0325     0      0
```

```r
# Same as we did for the description

threshold <- quantile(data_train_tokenized_ingredients$tf_idf, probs = 0.9)

df_top_10_percent_tf_idf <- data_train_tokenized_ingredients %>%
  filter(tf_idf >= threshold)

distinguishing_words_by_category <- split(
  df_top_10_percent_tf_idf, df_top_10_percent_tf_idf$category)

distinguishing_words_cakes <-
  distinguishing_words_by_category$cakes_cupcakes_snack_cakes$word
distinguishing_words_candy <- distinguishing_words_by_category$candy$word
distinguishing_words_chips <-
  distinguishing_words_by_category$chips_pretzels_snacks$word
distinguishing_words_chocolate <- distinguishing_words_by_category$chocolate$word
distinguishing_words_cookies <-
  distinguishing_words_by_category$cookies_biscuits$word
distinguishing_words_popcorn <-
  distinguishing_words_by_category$popcorn_peanuts_seeds_related_snacks$word

# we will add the following variables to the features matrix

ingredients_candy_score <- terms_score_func(
  data_food_train$ingredients, distinguishing_words_candy)
ingredients_cakes_score <- terms_score_func(
  data_food_train$ingredients, distinguishing_words_cakes)
ingredients_chips_score <- terms_score_func(
  data_food_train$ingredients, distinguishing_words_chips)
ingredients_chocolate_score <- terms_score_func(
  data_food_train$ingredients, distinguishing_words_chocolate)
ingredients_popcorn_score <- terms_score_func(
  data_food_train$ingredients, distinguishing_words_popcorn)
ingredients_cookies_score <- terms_score_func(
  data_food_train$ingredients, distinguishing_words_cookies)

# TEST - use for predictions later
test_ingredients_candy_score <- terms_score_func(
  test_df_cleaned_text$ingredients, distinguishing_words_candy)
test_ingredients_cakes_score <- terms_score_func(
  test_df_cleaned_text$ingredients, distinguishing_words_cakes)
test_ingredients_chips_score <- terms_score_func(
  test_df_cleaned_text$ingredients, distinguishing_words_chips)
test_ingredients_chocolate_score <- terms_score_func(
  test_df_cleaned_text$ingredients, distinguishing_words_chocolate)
test_ingredients_popcorn_score <- terms_score_func(
  test_df_cleaned_text$ingredients, distinguishing_words_popcorn)
test_ingredients_cookies_score <- terms_score_func(
  test_df_cleaned_text$ingredients, distinguishing_words_cookies)
```

Brand

```
# let's transfer brands that appear less than 5 times to another brand

length(unique(df_cleaned_text$brand))
```

```
## [1] 4714
```

```
rec <- recipe(~ brand, data = df_cleaned_text)
rec <- rec %>% step_other(
  brand, threshold = 6, other = "less than 5 appearances") %>% step_novel(brand)
rec <- prep(rec, training = df_cleaned_text)
rec_b <- rec %>% bake(df_cleaned_text)
df_cleaned_text$brand_reduced <- rec_b$brand
length(unique(df_cleaned_text$brand_reduced))
```

```
## [1] 854
```

```
# TEST - use for predictions later
test_rec <- rec %>% bake(test_df_cleaned_text)
test_df_cleaned_text$brand_reduced <- test_rec$brand
```

data nutrients + data food nutrients

```
merged_df_nutrients <- merge(data_food_nutrients, data_nutrients,
                             by = "nutrient_id", all.x = TRUE) %>% arrange(idx)

# We also can drop the nutrient_id variable because we have the 'name' variable

merged_df_nutrients <- merged_df_nutrients[-1]
glimpse(merged_df_nutrients)
```

```
## Rows: 493,054
## Columns: 4
## $ idx       <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, ~
## $ amount    <dbl> 7.14, 35.71, 53.57, 536.00, 3.60, 143.00, 5.14, 89.00, 0.00,~
## $ name      <chr> "Protein", "Total lipid (fat)", "Carbohydrate, by difference~
## $ unit_name <chr> "G", "G", "G", "KCAL", "G", "MG", "MG", "MG", "IU", "MG", "M~
```

```
# add test

data_food_test$category <- "unknown"
data_food <- rbind(data_food_train, data_food_test)
merged_df_nutrients <- merge(merged_df_nutrients, data_food[, c(1,8)],
                             by = "idx", all.x = TRUE)

merged_df_nutrients <- merged_df_nutrients %>% group_by(idx) %>%
  mutate(num_nutrients = length(name))
```

```
# for model use later, take 'idx' and 'num_nutrients' for train and test

df_idx_num_nuts <- merged_df_nutrients %>% select(idx, num_nutrients) %>% distinct()
```

```r
df_idx_num_nuts_train <- df_idx_num_nuts[data_food_train$idx , ]

# TEST - use for predictions later
df_idx_num_nuts_test <- df_idx_num_nuts[data_food_test$idx , ]
```

```r
# Pivot Wider

# check duplicates - because we received a warning related to it

merged_df_nutrients %>% dplyr::group_by(idx, name) %>%
  dplyr::summarise(n = dplyr::n(), .groups = "drop") %>%
  dplyr::filter(n > 1L)
```

```
## # A tibble: 1 x 3
##     idx name       n
##   <dbl> <chr>  <int>
## 1 29244 Energy     2
```

```r
# found one case, drop it

merged_df_nutrients <- merged_df_nutrients[-407232,]

# 407232 is the row of idx 29244, that observation had two unit names (KCAL, KJ)
# for the nutrient Energy, we dropped the one with KJ and left the calories one.

nut_wide <- merged_df_nutrients %>% select(idx, name, amount)
nut_wide <- pivot_wider(data = nut_wide, names_from = name,
                        values_from = amount, values_fill = 0)

# Split for train and test for model usage later

nut_wide_train <- nut_wide[data_food_train$idx , ]

# TEST - use for predictions later
nut_wide_test <- nut_wide[data_food_test$idx , ]
```

Model 01

```r
# Collecting the features

data_model_train <- data.frame(data_food_train$idx, data_food_train$serving_size,
  df_cleaned_text$household_lemma, df_cleaned_text$brand_reduced,
  description_candy_score, description_cakes_score, description_chips_score,
  description_chocolate_score, description_popcorn_score, description_cookies_score,
  ingredients_candy_score, ingredients_cakes_score, ingredients_chips_score,
  ingredients_chocolate_score, ingredients_popcorn_score, ingredients_cookies_score,
  data_food_train$category)

data_model_train$num_nutrients <- df_idx_num_nuts_train$num_nutrients

colnames(data_model_train) <- c("idx", "serving_size", "household", "brand", "desc_candy", "desc_cakes"

data_model_train <- merge(data_model_train, nut_wide_train, by = "idx")
```

Write csv

```r
write.csv(data_model_train, "data/data_model_train.csv")
```

Split data

```r
split <- initial_split(data = data_model_train, strata = idx)
train <- training(split)
val <- testing(split)
```

Recipe

```r
recip <- recipe(category ~ . , train) %>% update_role(idx, new_role = "Id")
recip <- recip %>%
  step_novel(household) %>%
  step_dummy(all_nominal_predictors())
recip <- prep(recip, training = train)
train_b <- recip %>% bake(train)
val_b <- recip %>% bake(val)
```

Random Forest model

Tuning for mtry and min_n parameters

```r
# We are using 100 trees for the tuning from running time considerations.

mod_rf <- rand_forest(mode = "classification", mtry = tune(), trees = 100 , min_n = tune()) %>%
  set_engine("ranger")

rf_grid <- grid_regular(mtry(range(10, 70)), min_n(range(10, 30)),
                        levels = c(4, 3))
rf_grid

cv_splits <- vfold_cv(train_b, v = 5)
cv_splits
```

```r
tune_res <- tune_grid(mod_rf,
                      recipe(category ~ . , data = train_b) %>%
                        update_role(idx, new_role = "Id") ,
                      resamples = cv_splits,
                      grid = rf_grid,
                      metrics = metric_set(roc_auc))
tune_res

tune_res$.metrics[[1]]
estimates <- collect_metrics(tune_res)
estimates
```

```r
estimates %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(aes(x = mtry, y = mean, color = min_n)) +
  geom_point() +
  geom_line() +
  labs(y = "roc auc") +
  theme_bw()

best_roc_auc <- tune_res %>% select_best(metric = "roc_auc")
best_roc_auc
mod_rf_final <- finalize_model(mod_rf, best_roc_auc)
mod_rf_final
```

```r
# The final RF model will have the parameters: mtry=70, trees=1000 , min_n=10

mod_rf <- rand_forest(mode = "classification", mtry = 70, trees = 1000 , min_n = 10) %>%
  set_engine("ranger") %>%
  fit_xy(x = train_b %>% select(-category),
         y = train_b$category)
```

```r
final_model_RF_res <- mod_rf %>% predict(new_data = val_b)
mean(final_model_RF_res$.pred_class == val_b$category)
```

```
## [1] 0.9170025
```

```r
write_rds(mod_rf, "data/RF_01.rds")
```

Preparing test observations for prediction

```r
data_model_test <- data.frame(data_food_test$idx, data_food_test$serving_size,
  test_df_cleaned_text$household_lemma, test_df_cleaned_text$brand_reduced,
  test_description_candy_score, test_description_cakes_score,
  test_description_chips_score, test_description_chocolate_score,
  test_description_popcorn_score, test_description_cookies_score,
  test_ingredients_candy_score, test_ingredients_cakes_score,
  test_ingredients_chips_score, test_ingredients_chocolate_score,
  test_ingredients_popcorn_score, test_ingredients_cookies_score)

data_model_test$num_nutrients <- df_idx_num_nuts_test$num_nutrients

colnames(data_model_test) <- c("idx", "serving_size", "household", "brand", "desc_candy", "desc_cakes",

data_model_test <- merge(data_model_test, nut_wide_test, by = "idx")
```

Bake test

```r
test_b <- recip %>% bake(data_model_test)
```

Write predictions to csv

```r
rf_predictions <- mod_rf %>% predict(new_data = test_b)
data_food_test <- data_food_test %>% mutate(pred_cat = rf_predictions$.pred_class)
data_food_test %>%
  select(idx, pred_cat) %>%
  write_csv("model01.csv")
```

Model 02

```r
# Let's try to build a model with more intuitive features, that is based on
# our previous knowledge

data_model_train$description <- data_food_train$description
data_model_test$description<- data_food_test$description
rec_1 <- recipe(category ~ ., data = data_model_train) %>%
  update_role(idx ,new_role = "Id") %>%
  step_novel(all_nominal_predictors(),new_level = "still not apper")%>%
  step_mutate(brand_candies = as.numeric(rowSums(sapply(c('candies', 'candy'),
    grepl, description, ignore.case = TRUE)) > 0)) %>%
  step_mutate(description_cake = as.numeric(grepl(
    "cake", description, ignore.case = TRUE))) %>%
  step_mutate(description_candy = as.numeric(grepl(
    "candy", description, ignore.case = TRUE))) %>%
  step_mutate(description_chips = as.numeric(grepl(
    "chips", description, ignore.case = TRUE))) %>%
  step_mutate(description_cookies = as.numeric(grepl(
    "cookies", description, ignore.case = TRUE))) %>%
  step_mutate(description_popcorn = as.numeric(grepl(
    "popcorn", description, ignore.case = TRUE))) %>%
  step_mutate(description_cake = as.numeric(grepl(
    "cake", description, ignore.case = TRUE))) %>%
  step_mutate(description_potato_chips = as.numeric(grepl(
    "potato chips", description, ignore.case = TRUE))) %>%
  step_mutate(description_dark_chocolate = as.numeric(grepl(
    "dark chocolate", description, ignore.case = TRUE))) %>%
  step_mutate(description_milk_chocolate = as.numeric(grepl(
    "milk chocolate", description, ignore.case = TRUE))) %>%
  step_mutate(description_tortilla_chips = as.numeric(grepl(
    "tortilla chips", description, ignore.case = TRUE))) %>%
  step_mutate(description_nuts = as.numeric(rowSums(sapply(c(
    'mixed nuts', 'trail mix'), grepl, description, ignore.case = TRUE)) > 0)) %>%
  step_rm(description) %>%
  step_zv(all_numeric_predictors()) %>%
  prep(data = data_food_train)

food_train_tr_ready1 <- rec_1 %>% bake(data_model_train)
food__te_ready1 <- rec_1 %>% bake(data_model_test)
```

```r
split2 <- initial_split(data = food_train_tr_ready1)
train2 <- training(split2)
val2<- testing(split2)
```

```r
mod_rf2 <- rand_forest(mode = "classification", mtry = 8, trees = 1000 , min_n = 5) %>%
  set_engine("ranger") %>%
  fit_xy(x = train2 %>% select(-category),
         y = factor(train2$category))
```

```r
library(dplyr)
data_food_test$pred_cat_2 <- mod_rf2 %>% predict(new_data = food__te_ready1)
data_food_test %>%
  select(idx, pred_cat_2) %>%
  write.csv("model02.csv")
```