Final Project - DSApps

May Ben Hamo

2020-08-14

Contents

Exploratory Data Analysis

Reading & Preparing the data

```
food_train <- read.csv("data/food_train.csv")
food_test <- read.csv("data/food_test.csv")
food_nutrients <- read.csv("data/food_nutrients.csv")
nutrients <- read.csv("data/nutrients.csv")
```

For convenient, I'll change the names of the categories only for part A:

```
levels(food_train$category) <- c("cakes","candy", "chips", "chocolate", "cookies", "popcorn" )
```

Images

I created a df contain all the img_path and them merge it to the original data (the entire code available at the notebook)

```
food_train_images <- food_train %>% inner_join(img_df)
food_test_images <- food_test %>% inner_join(img_df_test)
```

Nutrrients

```
#  create a data frame that unifies all 4 data sets relevant to the train/test set:
food_train_all <- food_nutrients %>% inner_join(food_train) %>%
  inner_join(nutrients) %>% inner_join(img_df)

food_test_all <- food_nutrients %>% inner_join(food_test) %>%
  inner_join(nutrients) %>% inner_join(img_df_test)
```

Add a columne for each of the nutrients, impute 0 if the product doesn't contain the nutrient:

```
food_train_with_nutrients =  food_nutrients %>%  left_join(food_train , by = "idx") %>%
  pivot_wider(names_from = nutrient_id, values_from = amount, names_prefix = "nutr_") %>%
  mutate_at(vars(starts_with("nutr_")), ~{if_else(is.na(.), 0, .)}) %>% inner_join(food_train)

food_test_with_nutrients <- food_nutrients %>%  left_join(food_test , by = "idx") %>%
  pivot_wider(names_from = nutrient_id, values_from = amount, names_prefix = "nutr_") %>%
  mutate_at(vars(starts_with("nutr_")), ~{if_else(is.na(.), 0, .)}) %>% inner_join(food_test)
```

Quick View

Main Data

```
food_train %>% select_if(is.numeric) %>% skim_without_charts() %>% as_tibble() %>% select(-skim_type) %>%
  rename_at(vars(starts_with("numeric")), ~(str_remove(.,"numeric.")))%>%
  knitr::kable(digits = 3) %>%
  kable_styling(latex_options = c("striped"), font_size =8)
```

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
idx	0	1	17633.62	10176.708	1.000	8840.5	17615	26458.5	35276
serving_size	0	1	37.64	20.942	0.225	28.0	30	40.0	480

```
food_train %>% select_if(is.factor) %>% skim() %>% select(-factor.ordered, -skim_type)  %>% as_tibble() %>%
  rename_at(vars(starts_with("factor")), ~(str_remove(.,"factor."))) %>% knitr::kable(digits =1) %>%
  kable_styling(latex_options = c("striped"), font_size = 8)
```

skim_variable	n_missing	complete_rate	n_unique	top_counts
brand	0	1	4783	wal: 579, tar: 540, fer: 506, not: 467
description	0	1	26200	coo: 100, mil: 92, pot: 82, dar: 79
ingredients	40	1	26872	alm: 186, wal: 120, pec: 115, cas: 42
serving_size_unit	0	1	2	g: 31743, ml: 8
household_serving_fulltext	11	1	2206	1 o: 5424, 0.2: 2116, 3 p: 1114, 1 c: 1074
category	0	1	6	pop: 7645, can: 7584, coo: 5284, cak: 3786

A general summary of the train set can be seen in the table above. How many snacks in each category, how many companies there are, how many unique values for each factorial variable, etc. Also, it can be seen that in the train set there are a total of 11 + 40 missing values for two factorial variables. This is a very small number :)

Also, Almost no NA's in the test set:

```
miss_var_summary(food_test_images %>% select_if((~(sum(is.na(.)) > 0))) )
```

```
## # A tibble: 2 x 3
##   variable    n_miss pct_miss
##   <chr>        <int>    <dbl>
## 1 ingredients      5   0.142
## 2 brand            1   0.0284
```

Food nutrients + Nutrients

```
glimpse(food_train_all %>% select(nutrient_id,name,unit_name, amount )  )
```

```
## Observations: 443,970
## Variables: 4
## $ nutrient_id <int> 1087, 1089, 1104, 1162, 1003, 1004, 1005, 1008, 2000, 1...
## $ name        <fct> "Calcium, Ca", "Iron, Fe", "Vitamin A, IU", "Vitamin C,...
## $ unit_name   <fct> MG, MG, IU, MG, G, G, G, KCAL, G, G, MG, MG, G, G, G, M...
## $ amount      <dbl> 143.00, 5.14, 0.00, 0.00, 7.14, 35.71, 53.57, 536.00, 4...
```

```
food_train_all %>% bind_rows(food_test_all %>% mutate("category" == "test")) %>% distinct(idx) %>% nrow()
```

```
## [1] 35276
```

We have all the observations (train&test) in the databases of the nutrients. (31751 + 3525)

Also, no missing values (train & test):

```
food_train_all %>% bind_rows(food_test_all %>% mutate("category" == "test")) %>%
        select(unit_name, name, nutrient_id, amount) %>% select_if((~(sum(is.na(.)) > 0)))  %>% ncol()
```

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

In-Depth Analysis of the Fatures

My main goal: investigate in depth the relationship of each of the features to the snack's category.

Household Serving

As we saw in the quick view, we have 11 NA's in the train set, and there are 2206 distict values(train). There are a lot of levels, so I will now try to create a new feature using `household_serving_fulltext`:

```
top_words <- function(data, col, by_category = T) { # find top words of a col
  data <- data %>% mutate_at({{col}}, as.character)
  if(by_category) data <- data %>% group_by(category)
  data %>% unnest_tokens(word, {{col}}) %>%
  anti_join(stop_words) %>%
  filter(
    !str_detect(word, pattern = "[[:digit:]]"), # removes any words with numeric digits
    !str_detect(word, pattern = "[[:punct:]]"), # removes any remaining punctuations
    !str_detect(word, pattern = "(.)\\1{2,}"),  # removes any words with 3 or more repeated letters
    !str_detect(word, pattern = "\\b(.)\\b")    # removes any remaining single letter words
  ) %>%  count(word, sort = T)  }

glimpse( top_words(food_train, "household_serving_fulltext", by_category = F) %>% arrange(-n) )
```

```
## Observations: 412
## Variables: 2
## $ word <chr> "onz", "pieces", "cup", "cookies", "cookie", "grm", "piece", "...
## $ n    <int> 7770, 5671, 3483, 2052, 1152, 1110, 852, 821, 748, 732, 693, 5...
```

Using the above data, I identified keywords for creating `household_serving_update` as follows:

```
# a list with key and values
key_words_list <- list("cookie" = c("cookie", "cookies", "coookie", "brownie", "brookie", "macaroon","macarons
                         "truffle") ,
                   "cake" = c("cake", "cupcake", "donut", "danish", "loaf", "pastry"),
                   "pieces" = c("piece", "pieces", "pcs", "pcs.", "pc", "pc.", "psc", "portion"),
                   "pie" = "pie",  "onz" = c("onz","oz"),
                   "cup" = c("cup", "cups"),  "bar" = "bar" , "slice" = "slice",
                   "package" = c("package", "pack", "pkg" , "pk", "box", "pkg.","kit" ),
                   "lollipop" =  c("lol", "pop"), "bag" = "bag", "grm" ="grm" ,
                   "container" = "container",  "pretzel" ="pretzel", "squares" ="square" ,
                   "tbsp" =c("tbsp", "tsp"),  "chips" =c("chip","crisps"), "balls" = "ball",
                   "section" = c("section", "block") , "candy" = "cand",    "egg"= "egg",
                   "sticks"="stick" ,   "kernels" ="kernel" , "pouch" = "pouch",   "roll" = "roll")
keywords_fun <- function(words) str_c(words, collapse = "|")
key_words_list <- lapply(key_words_list, keywords_fun)
```

I created a function: `household_serving_update_fun` that classify each of the words in `household_serving_fulltext` according to `key_words_list` (The code is long so you can see it in the notebook)

```
datafulltext <- household_serving_update_fun(food_train,household_serving_fulltext, key_words_list )
```

```
cat("There are only", datafulltext %>% select(household_serving_update) %>% n_distinct(), "levels in the new f
```

```
## There are only 26 levels in the new feature
```

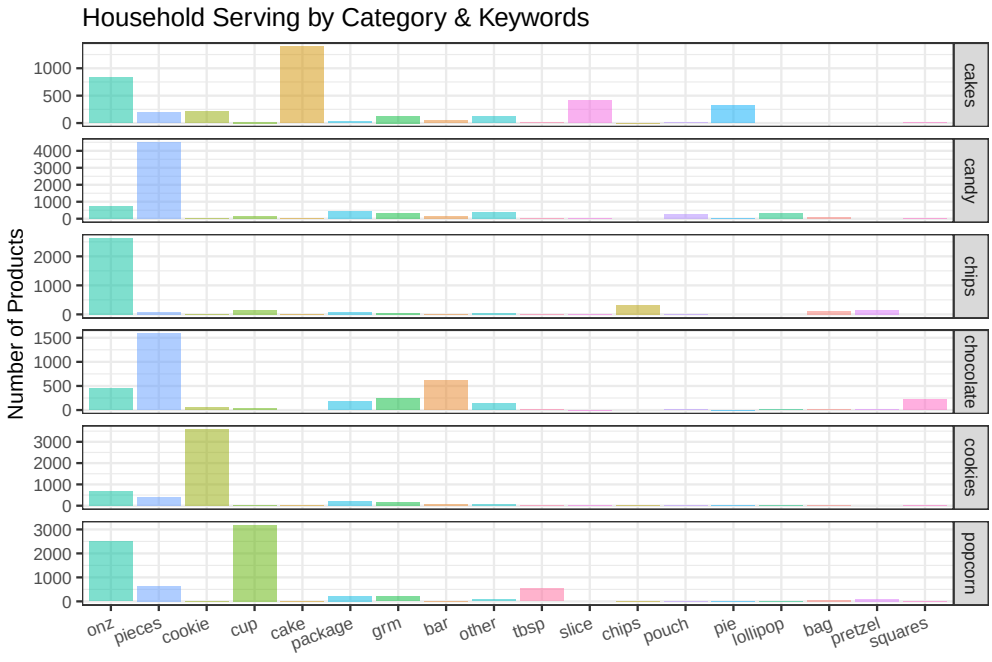
How many products do we have in each of the keys I created? (I will show the 15 top levels)

```
# d_cat -  no. of distinct categories contains products with this key
data2_fulltext <- datafulltext %>% group_by(household_serving_update) %>% summarise(d_cat = n_distinct(categor
kable(t( data2_fulltext %>% rename("key"=household_serving_update )), "latex", booktabs = T) %>%
   kable_styling(latex_options = c("striped"), font_size = 8)
```

key	onz	pieces	cookie	cup	cake	package	grm	bar	other	tbsp	slice	chips	pouch	pie	lollipop
d_cat	6	6	6	6	5	6	6	6	6	6	5	4	6	5	4
n	7788	7381	3908	3548	1404	1184	1110	892	831	622	418	343	336	326	314

Let's look at the distribution of the keywords by each of the categories (filter: keywords that appear at least 200 times):

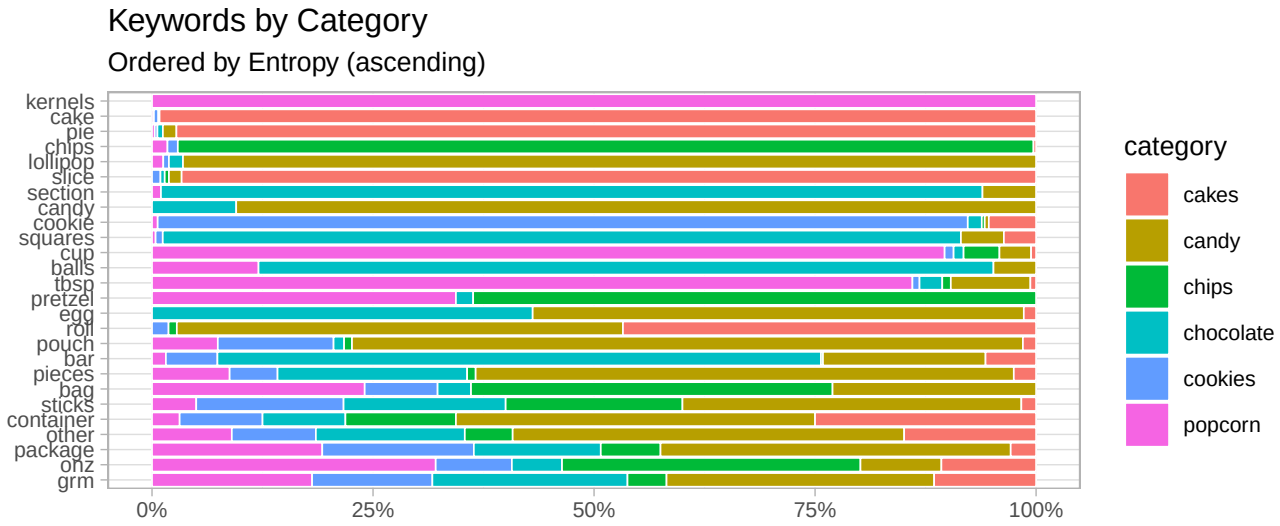
```
entropy_fun <- function(vec) -sum(vec[vec>0]*log(vec[vec>0]) )
data_plot_fulltext <- datafulltext %>% group_by(household_serving_update, category) %>% count(sort=T) %>% grou
  mutate(n= n, sum_n = sum(n), pct = n/sum(n), entropy = entropy_fun(pct) )

data_plot_fulltext %>% filter(sum_n>200) %>%
  ggplot(aes(x =reorder(household_serving_update, -sum_n), y = n , fill = household_serving_update ))  +
  geom_bar(stat = "identity", alpha = 0.5) +  facet_grid(category~., scales = "free_y") + theme_bw() +
  theme(axis.text.x = element_text(angle=20, hjust=1),
        text = element_text(size=10),legend.position = "") +
  labs(x= "", y = "Number of Products", title = "Household Serving by Category & Keywords")
```



Some keys are very popular in each of the categories, while some categories have one key that is very popular - like candy (pieces) ,chips (onz) and cookies (cookie). Also, there are categories that have many popular key like popcorn (onz, cup).

Let's have a second look and focus on the distribution of the category for each keyword - I'll compute the pct of each word for each of the categories and also the **entropy**. (Entropy is a measure of the impurity of the distribution of the categories for each word. As the entropy gets smaller, it means that the word is more "pure", and thus it might be good for prediction. (I'll explain it later - in the modeling stage - page **22**)

```
fulltext_plotly <- data_plot_fulltext  %>% mutate( pct = paste0((round(n/sum(n)*100, 2))," %")) %>%
  ggplot(aes(x =reorder(household_serving_update, -entropy), y = n , fill = category ,
             text = paste( "Houshold serving:", household_serving_update,"\n Category:", category,"\n Count",
                           "\n Total n", sum_n , "\n Entropy" , round(entropy,2) )))  +
 geom_col(position = "fill", color = "white", size = .3)  + coord_flip() +  theme_light() + theme(text = eleme
  labs(x="", y="", title = "Keywords by Category", subtitle = "Ordered by Entropy (ascending)")  +
  scale_y_continuous(labels = percent)
fulltext_plotly
```



For interactive plot - See html file

There are keywords that are very common only in one category (kernels, cakes, pie, etc), while there are keywords that are common in many categories like onz, grm. We even have one key (kernels) that appears only in the one category(popcorn),

but it contains only 25 obs and I may drop it later. **In conclusion, it seems like a very good feature to work with.**

Ingredients

As we saw in the quick view, we have 40 NA's in the train set and 5 in the test set, and there are 26872 distict values(train).

Let's find out which are the most common ingredients in each of the categories.

```
# flat_fun -  input: list of words,  output: one string contains all the words, separate by ","
flat_fun <- function(list)  str_flatten(unlist(list) , collapse = ",")
# a function to split & clean the ingredients. output: ingredients split by ","
split_by_ingredient <- function(data) {
  data %>%
    mutate(new_ingredient = str_replace_all(ingredients,  "\\s*[.,]\\s*$", "")) %>% # if the last char is , or
    mutate(new_ingredient = str_replace_all(new_ingredient,  "\\s*[$+*]\\s*", "")) %>%
    mutate(new_ingredient = str_replace_all(new_ingredient,  "\\s*and/or\\s*", ",")) %>% # drop and/or
    mutate(new_ingredient = str_replace_all(new_ingredient,  ",\\s*and\\s*", ",")) %>% # drop and if it is in
    mutate(new_ingredient = str_replace_all(new_ingredient,  "\\s+[,.()\\[\\]{}]+an\\s+", ",")) %>% # replace
    mutate(new_ingredient= str_split(new_ingredient, "\\s*[&,.:()\\[\\]{}]+\\s*") ) %>% # split by ,.() and re
    mutate(new_ingredient =  map(new_ingredient, function(vec) {stri_remove_empty(unique(vec))}) ) %>%
    mutate(new_ingredient = map_chr(.x = new_ingredient ,.f = flat_fun))
}
ingredients_data_train <- split_by_ingredient(food_train) ; ingredients_data_test <- split_by_ingredient(food_
```

```
# a function to find n top ingredients (by category)
n_top_ingredient <- function(data,num, by_category = TRUE) {
  data <- data %>% select(new_ingredient, category) %>%
  unnest_tokens(word, new_ingredient, token = stringr::str_split , pattern = ",") # split words by ","
  if(by_category)  data <- data %>%  group_by(category) # optional to fint top ingredients it by category
  data %>% count(word, sort = TRUE) %>% rename(ingredient = word) %>% dplyr::slice(1:num)          }
top_12_ing <- n_top_ingredient(ingredients_data_train, 12)
```

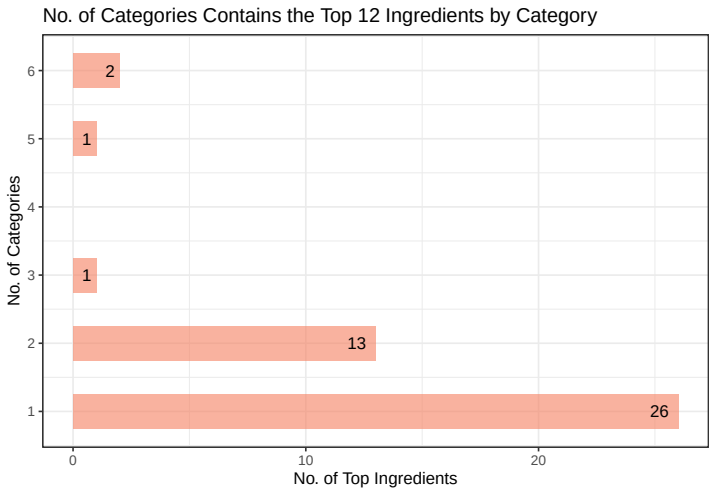
Let's take a look at top 10 ingredients in general(NOT by category):

```
kable(t( n_top_ingredient(ingredients_data_train, 12, by_category = F) ), "latex", booktabs = T) %>%
  kable_styling(latex_options = c("striped",  "scale_down"))
```

ingredient	sugar	salt	soy lecithin	corn syrup	citric acid	water	folic acid	niacin	riboflavin	cocoa butter	wheat flour	reduced iron
n	21464	17315	11055	9977	7877	6905	6884	6832	6787	6719	6705	5658

Now, I want to look at the top 12 ingredients by category. For each ingredient out of top 12 of each category - How many distinct categories contain it in the top 12?

```
data_plot_12_ing <- top_12_ing %>% group_by(ingredient) %>%
  summarize(total_category = n_distinct(category), n=n(), categories = str_flatten(unique(category), collapse
data_plot_12_ing %>%  ggplot( aes(x= total_category)) +
  geom_bar( stat="count", fill="#f68060", alpha=.6, width=.5) +
  geom_text(aes(label = ..count..), stat= "count", hjust = +1.5) +
  coord_flip() + theme_bw()+  scale_x_continuous( breaks = 1:6 ) +
  labs(x = "No. of Categories", y= "No. of Top Ingredients", title = "No. of Categories Contains the Top 12 In
```



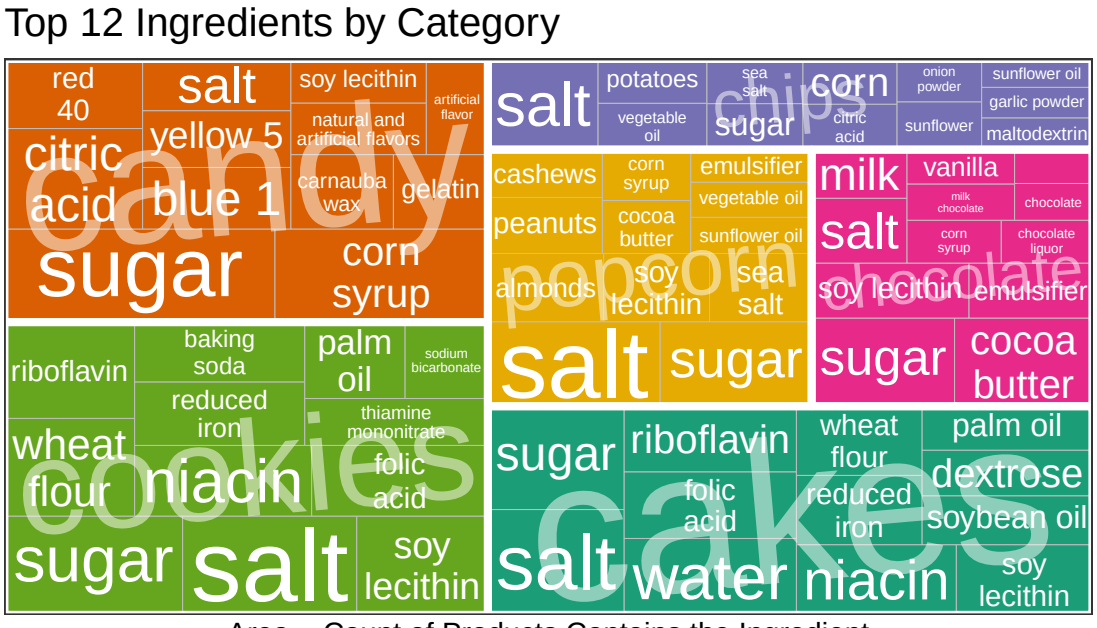
Most of the top ingredients by category appears only in 1 category, while just 3 of them appears in more than 3 categories:

```
data_plot_12_ing %>% filter(n>3) %>% select(ingredient,n,categories) %>% arrange(-n)
```

```
## # A tibble: 3 x 3
##   ingredient       n categories
##   <chr>        <int> <chr>
## 1 salt             6 cakes, candy, chips, chocolate, cookies, popcorn
## 2 sugar            6 cakes, candy, chips, chocolate, cookies, popcorn
## 3 soy lecithin     5 cakes, candy, chocolate, cookies, popcorn
```

Let's look at all the top 12 ingredient by category:

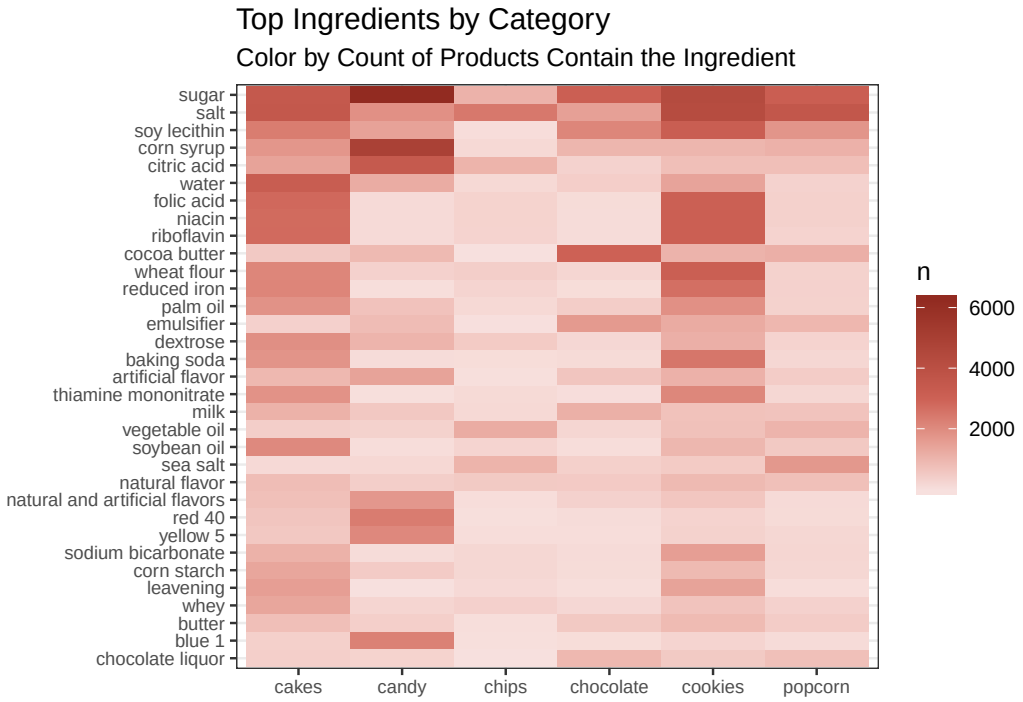
```
top_12_ing %>% ggplot( aes(area = n , fill = category , label =  ingredient, subgroup = category)) +
  scale_fill_brewer(palette = "Dark2") +
  geom_treemap() + geom_treemap_subgroup_border(colour = "white") +
  geom_treemap_subgroup_text(place = "centre", grow = T, reflow = T, alpha = 0.5, colour = "white",family="san
  geom_treemap_text(family = "sans", colour = "white", place = "centre", reflow = T, grow = T) +
  labs(x = "Area - Count of Products Contains the Ingredient", y = "", title="Top 12 Ingredients by Category")
  theme_bw() +  theme(text = element_text(family = "sans"), axis.text = element_text(size = 12),
        plot.title = element_text(size = 15), legend.position = "")
```



makes sense.

Let's take a look at the heatmap of the top ingredients in general appear at least 3000 times:

```
ingredients_plotly <- n_top_ingredient(ingredients_data_train, 1000)  %>% group_by(ingredient) %>%
  mutate(pct = paste0((round(n/sum(n)*100, 2))," %") ,sum_n = sum(n)) %>%  filter(sum_n>3000)  %>%
  ggplot(aes(category,reorder(ingredient, sum_n)  ,fill= n,
             text = paste("Ingredient: " , ingredient , "\n Category: " ,category, "\n Count: ", n, "\n Pct: "
  scale_fill_gradient2(low = "#F9EBEA", mid = "#CD6155", high = "#922B21", midpoint = 3000) +
  geom_tile( ) + labs(x="", y="", title = "Top Ingredients by Category",
       subtitle = "Color by Count of Products Contain the Ingredient") +
  theme_bw() +  theme(axis.text = element_text(size = 8))
ingredients_plotly
```



For interactive plot - See html file

Ingredient Score - I want to take it one step further and try to think of a nice feature that can be taken out of the ingerdient variable. For example - calculate for each observation - how many of the 12 popular ingredients by category it contains, and according to this calculate a score for each category. Thus, I will receive for each obs. - a score to belong to each of the categories.

```
# a function to compute the score by category for a product
# I did a change in score function later, But I leave it that way
# so that it will be reproducible. (I will discuss this later)
score_fixed <- function(vec1, ings) sum(str_detect(ings, vec1)) / length(vec1)

score_by_cat_fixed <- function(top_by_category, product) {
  if(is.na(product)) return(rep(NA,6)) #I will ignore NA's.
  apply(X = top_by_category , MARGIN = 2, FUN = score_fixed, ings = product )
}
# input: df and n , output: df with the scores for each of the categories
create_scores <- function(data,n) {
  top_n_ing <- n_top_ingredient(data, n)
  top_n_ing_by_category <-  data.frame( tapply(top_n_ing$ingredient, top_n_ing$category, list))
  as_tibble(t(map_dfc( data$new_ingredient, .f = score_by_cat_fixed, top_by_category =top_n_ing_by_category ))
  set_names(names(top_n_ing_by_category))  %>% rename_all(~(paste0("score_", {{n}}, "_" , .))) %>%
    mutate(category = data$category, idx = data$idx) }
ingredient_score_12 <-  create_scores(ingredients_data_train,12)
```

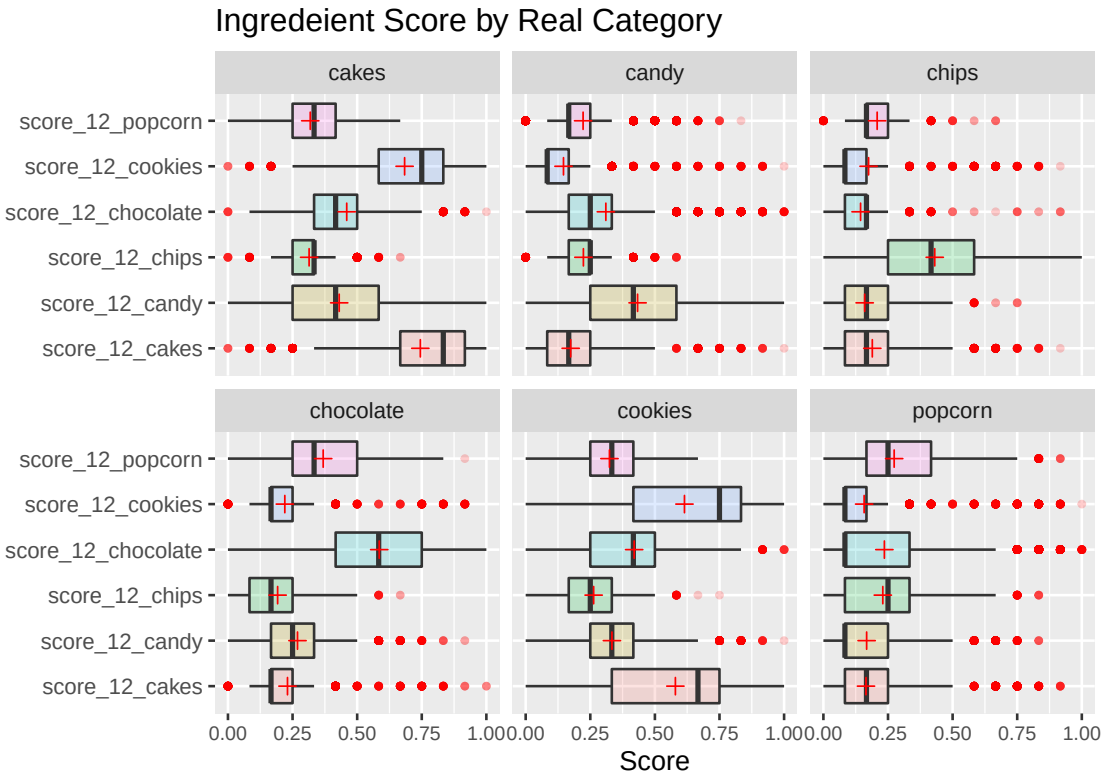
```
kable(ingredient_score_12 %>% group_by(category) %>% dplyr::slice(1) %>%  head(2), "latex", booktabs = T,  dig
  kable_styling(latex_options = c("striped"), , font_size = 8)
```

score_12_cakes	score_12_candy	score_12_chips	score_12_chocolate	score_12_cookies	score_12_popcorn	category	idx
0.7500	0.3333	0.3333	0.4167	0.7500	0.2500	cakes	9
0.3333	0.6667	0.2500	0.4167	0.3333	0.3333	candy	10

For example - the first obs. belongs to the cakes category, and its highest scores is for cookies/cakes. The second obs. is from the candy category, and indeed its highest score is for candy.

Let's take a deep look at the box plots for each of the scores by real category:

```
ingredient_score_12 %>% select(-idx) %>%
  pivot_longer(-category, names_to = "category_score", values_to = "score")  %>% drop_na() %>%
  group_by(category) %>% ggplot(aes(x=category_score, y= score, fill = category_score )) +
  geom_boxplot(alpha = 0.2, outlier.colour = 'red', outlier.size = 1) +
  stat_summary(fun.y=mean, geom="point", shape=3, size=2, color="red", fill="red") +
  facet_wrap(.~ category , nrow = 2) + coord_flip() +
  theme(legend.position = "", axis.text.x = element_text(size=8)) + labs(x= "", y = "Score", title = "Ingredei
```



As you can see, for any category - it seems that the relevant score does tend to have the highest distribution. A significant difference can be seen in the categories: candy and chips. There it seems that the score does indeed predict well the real category. **Looks like a nice feature to try when building the predictive model :)**

Description

I do not intend to explore in depth this variable, I will just examine which words are most common (top5) in each category.

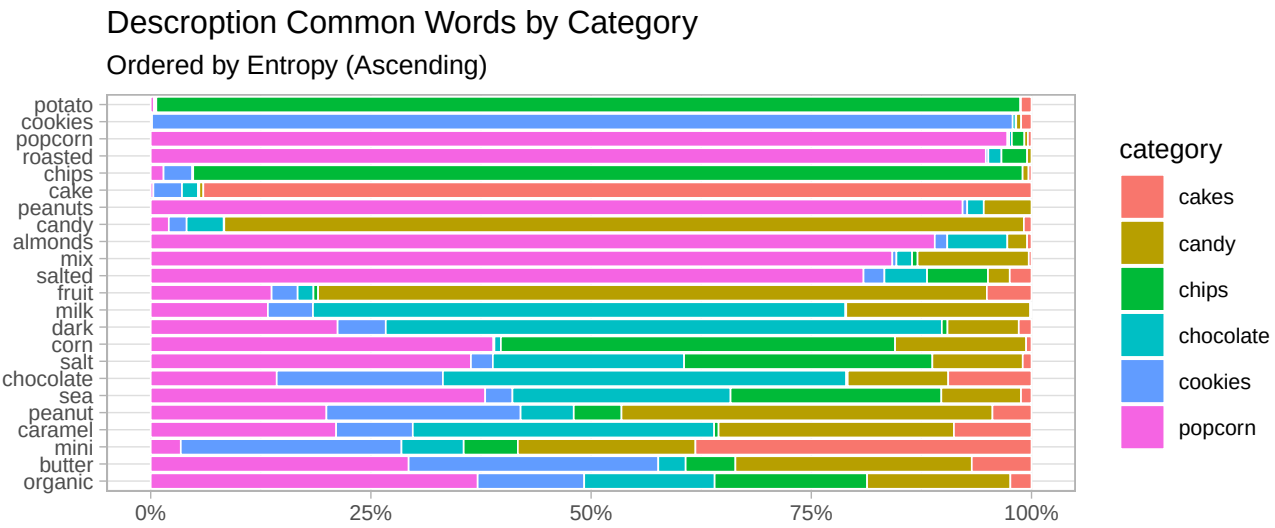
```
descr_top_5 <- top_words(food_train, "description", by_category = T) %>% dplyr::slice(1:5)  %>%
  unite(word_n, c("word", "n"), sep = ",n=") %>% group_by(category) %>%
  group_map(~.) %>% bind_cols() %>% setNames(levels(food_train$category))
```

```
kable(descr_top_5, "latex", booktabs = T) %>% kable_styling(latex_options = c("striped", "scale down"), font_s
```

cakes	candy	chips	chocolate	cookies	popcorn
cake,n=1243	candy,n=2825	chips,n=2516	chocolate,n=3142	cookies,n=3152	roasted,n=1334
chocolate,n=649	chocolate,n=783	potato,n=1387	milk,n=1204	chocolate,n=1295	mix,n=1087
pie,n=518	fruit,n=719	tortilla,n=664	dark,n=1178	cookie,n=557	almonds,n=1070
cupcakes,n=368	sour,n=513	kettle,n=428	caramel,n=338	chip,n=488	chocolate,n=982
mini,n=335	gummi,n=469	corn,n=367	bar,n=332	sandwich,n=450	popcorn,n=959

makes sense.

Let's look at a similar plot to the one we saw for `household serving fulltext`. I will look at words that appear at least 750 times. The words arranged by their value of entropy.



For interactive plot - See html file

We do have pretty strong words for prediction!

Brand

5 top brands:

```
kable(food_train %>% group_by(brand) %>% count(brand, sort = T) %>% head(5), "latex", booktabs = T) %>%
  kable_styling(latex_options = c("striped"), font_size =8)
```

brand	n
wal-mart stores, inc.	579
target stores	540
ferrara candy company	506
not a branded item	467
meijer, inc.	463

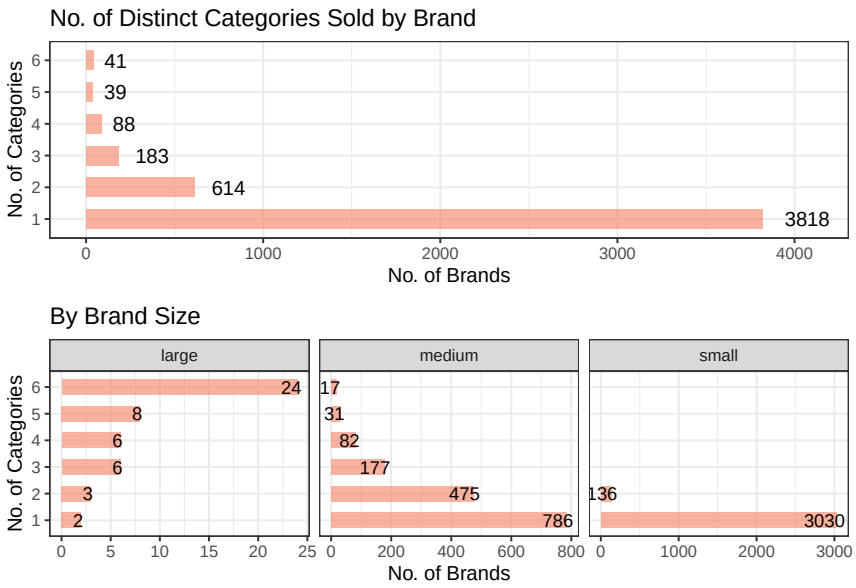
```
brand_data <- food_train %>% group_by(brand) %>% summarize( n=n() , total_category = n_distinct(category))
kable(t(quantile(brand_data$n , probs = c(0.1,0.25,0.5,seq(0.7,1,0.025) ))) ,"latex", booktabs = T) %>%
  kable_styling(latex_options = c("striped"), font_size = 8)
```

10%	25%	50%	70%	72.5%	75%	77.5%	80%	82.5%	85%	87.5%	90%	92.5%	95%	97.5%	100%
1	1	1	3	3	4	4	5	6	7	8	11	15	23	42.45	579

As you can see: 50% of the brands sell just one product, 85% of what brands sell no more than 7 products. Only 2.5% of brands sell between 42 and 579 products.
Let's look at the distribution of the number of categories sold by brands.

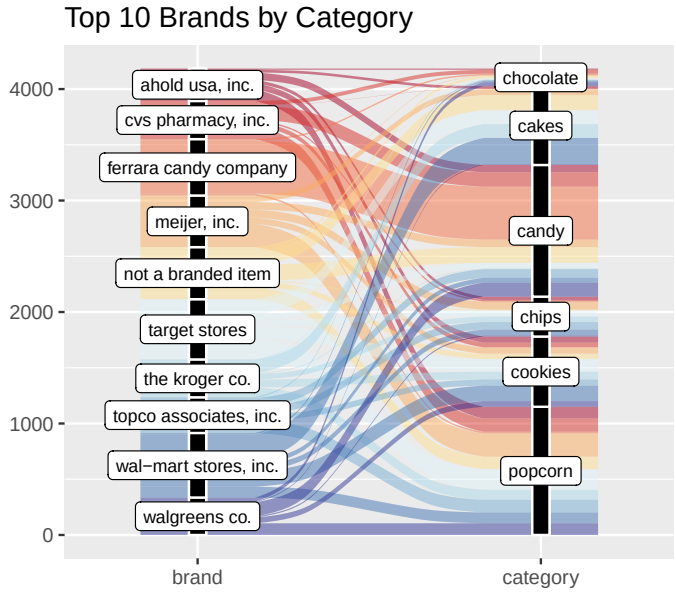
```
a1 <- brand_data %>% ggplot(aes(x= factor(total_category))) +
  geom_bar(stat="count", fill="#f68060", alpha=.6, width=.6) + ylim(c(0,4100)) +
  geom_text(aes(label = ..count..), stat= "count", hjust = -.5, size = 4) +
  coord_flip() +  theme_bw() +
  labs(x = "No. of Categories", y= "No. of Brands", title = "No. of Distinct Categories Sold by Brand")
```

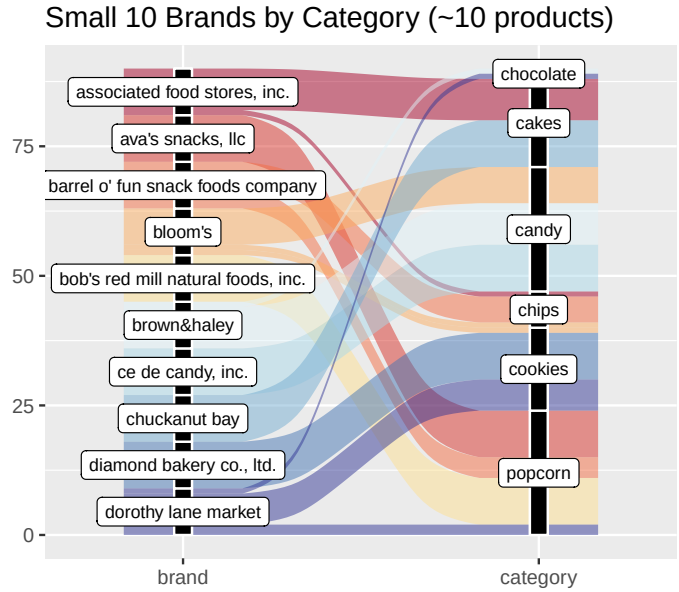
```
a2 <- brand_data %>%  mutate(brand_size = case_when(n >=100  ~ "large", (n > 2 & n < 100) ~ "medium" ,TRUE~ "s
  ggplot( aes(x= factor(total_category))) +
    geom_bar(stat="count",  fill="#f68060", alpha=.6, width=.6)  +
  facet_wrap(.~brand_size, scales = "free_x")+
    geom_text(aes(label = ..count..),size =3.5  ,  stat= "count", hjust = +.8) +
     coord_flip() + theme_bw()+ labs(x = "No. of Categories", y= "No. of Brands", title = "By Brand Size")
grid.arrange(a1,a2, nrow=2)
```



It can be seen that most of the brands have products from only one snack category, while a very small number of brands have products from all snack categories. I have divided the brands according to their size (no. of products each brand sells) - and it seems thatfor small brands - there are usually products from only one category, and for large brands the opposite. This trend can also be seen in the following plot:

```
brand_data_plot <- food_train %>% group_by(category, brand) %>% summarise(n=n()) %>% arrange(-n)
top_brands <- food_train %>% group_by(brand) %>% count(sort=TRUE) %>% head(10) %>% pull(brand)
small_brands <- food_train %>% group_by(brand) %>% count(sort=TRUE) %>% filter(n<10) %>%  pull(brand)
plot_size_of_brand <- function(data, brands, is_small = F) {
  p <- data %>% filter(brand %in% brands) %>% ggplot(aes(y = n, axis1 = brand, axis2 = category)) +
     geom_alluvium(aes(fill = brand)) +  geom_stratum(width = 1/20, fill = "black", color = "white") +
     geom_stratum(width = 1/20, fill = "black", color = "white") +
     geom_label(stat = "stratum", infer.label = TRUE, size = 2.8) +
     scale_x_discrete(limits = c("brand", "category"), expand = c(.05, .31)) +
     scale_fill_brewer(palette = "RdYlBu") +  guides(fill = FALSE) +
     labs(y = "", title = "Top 10 Brands by Category")
  if(is_small) p <- p + labs(title = "Small 10 Brands by Category (~10 products)")
  p
}
p1 <- plot_size_of_brand(brand_data_plot, top_brands[1:10])
p3 <- plot_size_of_brand(brand_data_plot, small_brands[1:10], T)
grid.arrange(p1,p3, ncol = 2)
```





It therefore appears that brand size may be an important predictor variable.

Serving Size & Serving Size Unit

In the quick view of the data we saw summary of this feature.
There are only 8 obs. in which their serving size measured in "ml":

description	serving_size	category
sour & fruity candy	9.0	candy
bubbles candy	20.0	candy
icee, squeeze candy, blue raspberry	62.0	candy
toxic waste, slime licker sour rolling liquid candy	5.0	candy
markets of meijer, fresg cin-a-butter pound cake	480.0	cakes
spray candy	4.5	candy
candylicious bubbles	20.0	candy
candy spray	90.0	candy

But in the test we have no observations in which unit_size = ml:

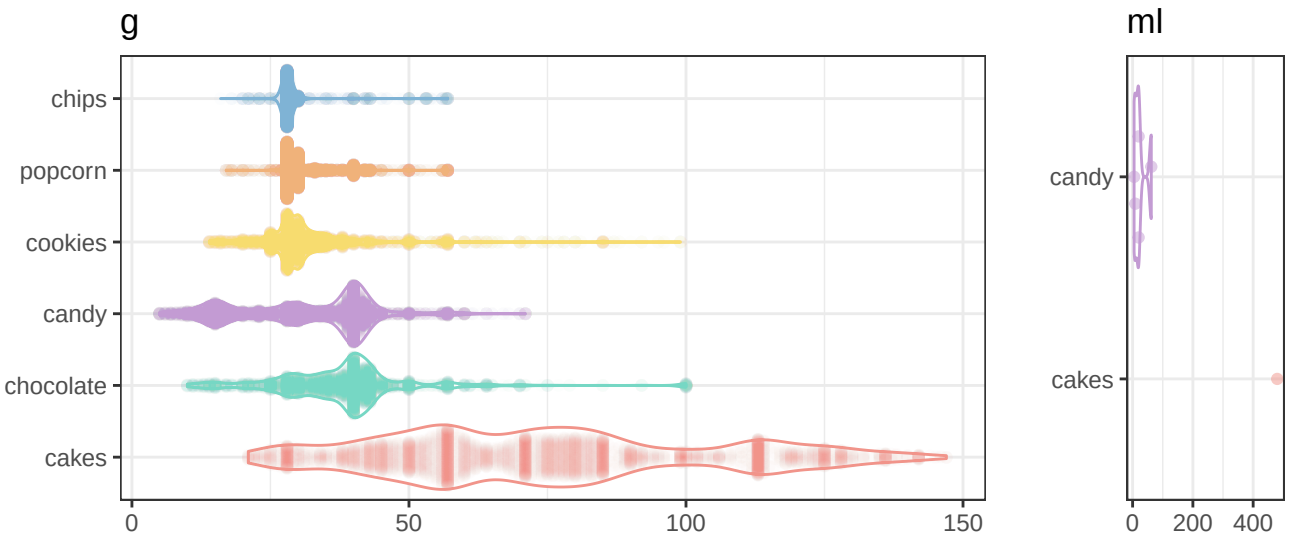
```
sum(food_test$serving_size_unit == "ml")
```

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

Therefore, I am going to ignore the feature `serving_size_unit` in the modeling part since I don't belive I can learn something from only 8 obs.

Let's take a look at the density of serving size by category & size unit (I trimmed extreme values)

```
color_set <- c("#F1948A","#C39BD3", "#7FB3D5", "#76D7C4", "#F7DC6F", "#F0B27A")
g_fun <- function(data, alpha ,unit_size) {
  p <- data %>% filter(serving_size_unit == unit_size) %>%  group_by(category) %>%
    filter(serving_size >= quantile(serving_size, 0.01) & serving_size <=  quantile(serving_size, 0.99) )  %>%
    mutate(median_servig_size = median(serving_size)) %>%
    ggplot(aes(x = reorder(category, -median_servig_size) , y = serving_size, color = category  )) +
    geom_violin(scale="width", size=.5)  +  geom_quasirandom(alpha = alpha) +
    scale_color_manual(values = color_set) +   coord_flip() +
    theme_bw() +labs(title = unit_size, x ="",y="") +  theme(legend.position = "")
  if(unit_size == "ml") p <- p + scale_y_continuous(breaks = c(0,200,400)) + scale_color_manual(values = c("#F
  p
}
g1 <- g_fun(food_train, 0.02, "g") ; g2 <- g_fun(food_train, 0.5, "ml")
grid.arrange(g1,g2, ncol=2, widths = c(3.5,1),  top = "Serving Size Density By Category & Size Unit")
```



It seems that the size of cakes tends to be larger and has the largest variance. The distributions of chips and Popcorn are quite similar. The distributions of candy and chocolate are also relatively similar.

Food Nutrition & Nutrients

```
nutr_data <- food_train_all %>% select(idx, nutrient_id, amount, category, name, unit_name, description, brand
```

```
cat("There are", nutr_data %>% group_by(name) %>% distinct(name) %>% nrow(), "food nutrients")
```

```
## There are 47 food nutrients
```

What are the most common nutrients and their average amount? (top 5)

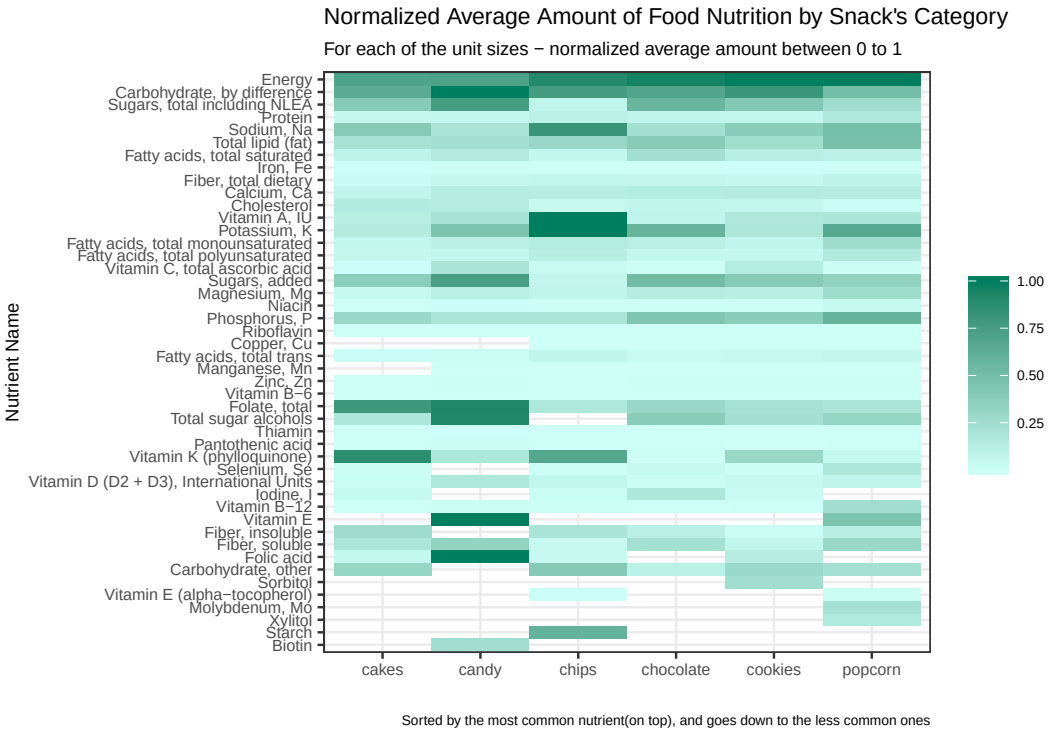
```
kable(nutr_data %>% filter(amount > 0) %>% group_by(name) %>%
        summarise(n=n(), mean_amount = mean(amount)) %>% arrange(-n) %>% head(5), "latex", booktabs = T) %>%
  kable_styling(latex_options = c("striped") , font_size = 8)
```

name	n	mean_amount
Energy	31732	460.350214
Carbohydrate, by difference	31725	59.707241
Sugars, total including NLEA	29156	36.819905
Protein	27558	8.029765
Sodium, Na	27192	339.265152

```
nutr_data2 <- nutr_data %>% group_by(category ,unit_name, name) %>% filter(amount!=0) %>% summarise(mean_amoun
order_nutrient <- nutr_data %>% filter(amount !=0) %>% count(name, sort = T) %>% arrange(n) %>% pull(name)
```

Let's take a look at heatmap of nutrients by category. For each of the units of measurement, I normalized the average amount to be between 0 and 1, so that they would be comparable.

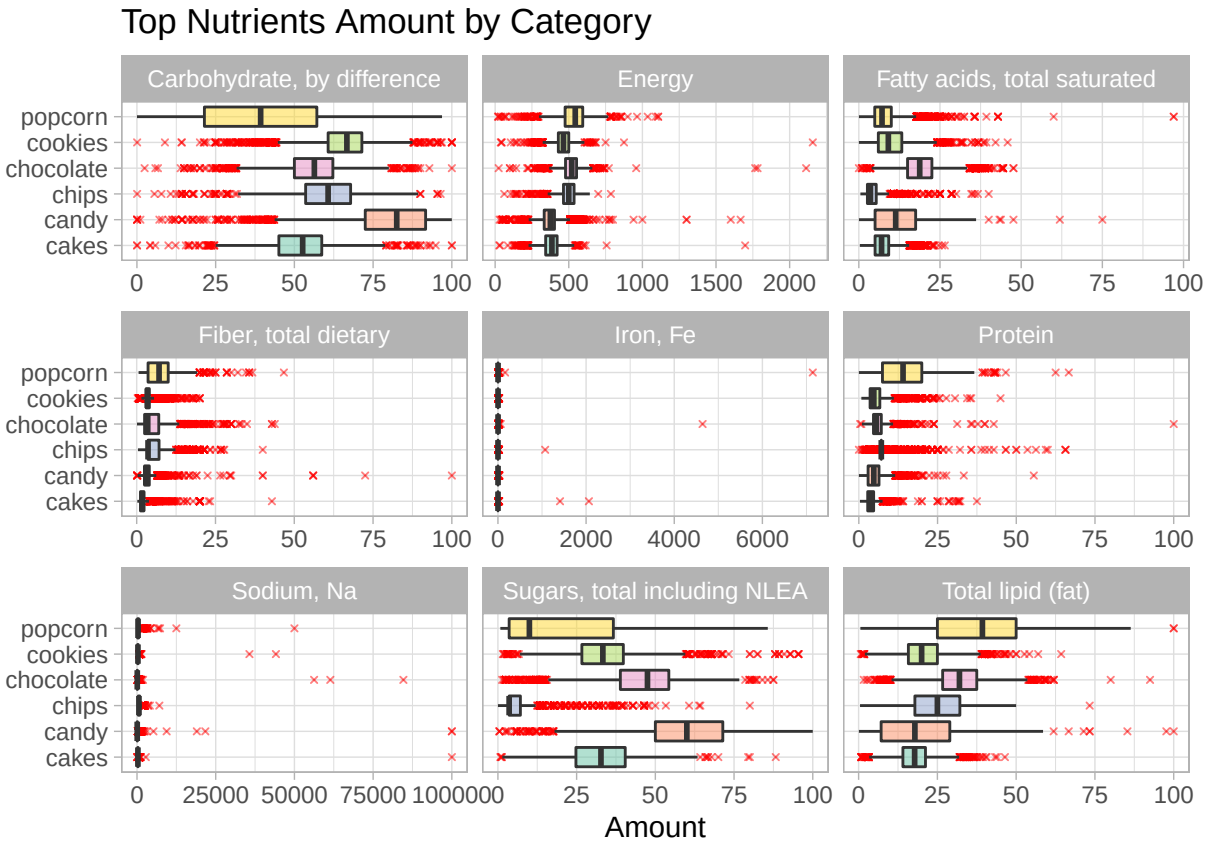
```
plotly_nutr <- nutr_data2 %>% group_by(unit_name) %>%
  mutate(normalize_mean_amount = mean_amount/max(mean_amount),name = factor(name, levels = order_nutrient)) %>
  ggplot(aes(x = category,y = name , fill= normalize_mean_amount,
                 text = paste("Nutrient", name, "\n Category", category, "\n Average Amount", round(mean_amou
                              "\n Normalizes mean", normalize_mean_amount , "\n Count", n))) +
  geom_tile() + scale_fill_gradient(low="#CDFFF7", high="#007F5F", name = "")  +
  theme_bw() +  theme(axis.text = element_text(size = 6.5), text = element_text(size=8)) +
  labs(title = "Normalized Average Amount of Food Nutrition by Snack's Category",
       y = "Nutrient Name", x="", subtitle = "For each of the unit sizes - normalized average amount between 0
plotly_nutr
```



For interactive plot - See html file

Let's take a deep look at the 9 most popular nutrients:

```
nutr_data %>% filter(amount !=0, name %in% rev(order_nutrient)[1:9]) %>% group_by(category,name) %>%
  ggplot(mapping = aes(x = category, y = amount, fill = category)) +
  geom_boxplot(outlier.colour = 'red', outlier.size = .8, outlier.shape = 4, alpha=.5, show.legend = F) +
  facet_wrap(name~., scales = "free_x", nrow = 3) + scale_fill_brewer(palette = "Set2") +
  theme_light() + coord_flip() + labs(x="", y= "Amount", title = "Top Nutrients Amount by Category")
```



There are products with very extreme values. Maybe it's typos, and maybe it's true. Let's explore some of them:

Total lipid(fat)

```
df2 <- nutr_data %>% filter(name == "Total lipid (fat)", amount>90) %>% select(idx, name, amount,unit_name, ca
```

idx	name	amount	unit_name	category	brand	description
10968	Total lipid (fat)	100.00	G	candy	rb. confections, lc.	dierbergs markets, soft cherry candy balls
10986	Total lipid (fat)	97.67	G	candy	jerry kelly deli, inc.	the candy tree, gum drop candy
16201	Total lipid (fat)	92.50	G	chocolate	chocolate holdings, inc.	charles chocolates, orange twigs chocolate
17911	Total lipid (fat)	100.00	G	popcorn	so nut and confections	praline pecans
23556	Total lipid (fat)	100.00	G	popcorn	kara chocolates	happy easter caramel and bunny corn mix

Let's take a look on them: (*the order is according to the table from left to right*)

```
knitr::include_graphics(df2 %>% select(img_path)%>% pull())
```

Be careful with those snacks!! Very NOT healthy!

Protein - We can see one product contain **100%** of protein! It is clearly NOT true...

```
df3 <- nutr_data %>% filter(name == "Protein", amount==100) %>% select(idx, name, amount,unit_name, category,
```

idx	name	amount	unit_name	category	brand	description
13676	Protein	100	G	chocolate	tcho	tcho, tcho-a-day dark chocolate

And we can keep going like that but due to lack of space I will stop here :)

Images

I created in python histograms of the 3 colors of the "average image" by category (I calculated the average value for each of the pixels across all images of each of the categories). The entire code avalilabe at the notebook.

```
mean_img_by_category <- category_hist_by_rgb %>%
  pivot_longer(everything(),names_to = "category_color", values_to = "mean_pixel") %>%
  separate(category_color, c("category", "color"), sep = "\\." ) %>% mutate_at("category", as.factor)
levels(mean_img_by_category$category) <-  c("cakes", "candy", "chips", "chocolate", "cookies", "popcorn" )

mean_img_by_category %>% ggplot(aes(x= mean_pixel, fill = color)) +
  geom_histogram(alpha=0.4, color = "white", bins = 30,show.legend = FALSE) +
  facet_wrap(.~category, nrow =2) + xlim(0,255) +
  scale_fill_manual(values = c("blue", "green", "red")) + theme(legend.position = "none") + theme_bw() +
  labs(title ="Colors Histograms of the Average Image By Category", x="" )
```

Indeed there are differences, some larger and some less, between the color histograms of the "average image" in each of the categories. Although it doesn't seem that on average there is any noticeable trend of difference between the images of different categories.

Note: Page numbering starts from 2, So I'm still ok :)

Modeling

Strategy

I'll split the data into six parts:

1. **Class Imbalance**
2. **Numerical Features** - Nutrients, Serving Size, Ingredient Score, Household Service fulltext num
3. **Categorical Features** - Brand, Unit Size - *as I said before, I'll ignore this feature.*, Serving Fulltext
4. **Text Features** - Ingredients, Description
5. **Model Tunning**
6. **Model Selection** - Select the best model out of all the models.

Missing data- As we saw in part A, we have a small no. of NA's both in the train set and in the test set. Also, all the NA's are in categorical features. I decided anyway to keep these observations and to use `step_unknown` or mode/mean impute. Therefore, I do not intend to allocate data for the treatment of missing values since even if there's a trend regarding NA's since it's such a small number, you can not learn anything from it.

Intercations - Most of the models that I am considering is taking care of intercations (neural networks, tree based models, svm), so I don't allocate data to consider possible interactions.

Nutrients Data set - I decided to insert each nutrient individually using the columns I created ("nutr_"), and I intend to normalize them all. Therefore I do not want to include `unit_name` in my models since if all the nutrients are normalized and inserted separately - it has no meaning.

Models

I'll consider the following models: Multinomial regression lasso, Multinomial regression ridge, SVM rbf, SVM poly, Random Forest, Xgboost, KNN, Nueral Networks.

Due to lots of problems running `keras` and `tensorflow` using `reticulate`, I decided that I would run the network models after making all the decisions (ie in the final stage of model selection) in Python.

Initial Data + NA's

The data (train + test) I will start with will also include the columns I created for each nutrient, the clean ingredients column, and another column I created from the numbers in the "household_serving_fulltext" column.

```
food_train_initial <- food_train_with_nutrients %>% mutate(
  new_ingredient = ingredients_data_train$new_ingredient,
  household_serving_fulltext_num = as.numeric(str_extract(household_serving_fulltext, pattern = "^\\d+\\.?\\d*
food_test_initial <- food_test_with_nutrients %>% mutate(
  new_ingredient = ingredients_data_test$new_ingredient,
  household_serving_fulltext_num = as.numeric(str_extract(household_serving_fulltext, pattern = "^\\d+\\.?\\d*
```

As we saw in Part A, there are very few NAs. We saw that in the `brand` column in the test set there is one NA - I will replace it with "not a branded item":

```
food_test_initial <- food_test_initial %>% mutate("brand"=  fct_explicit_na(brand, "not a branded item"))
```

As I said, I'll deal with the rest of the NA's already through the use of the recipes.

Spliting the data

The split was done as follows (code in notebook):

name	imbalance	numerical	categorical	txt	tunning	selection	test
n_rows	2500	4000	5001	5001	4000	4901	6348

In addition, I kept the indexes for the train & test so that I could split the data in the same way in Python, when I run the neural network model with images.

Main Functions

Create functions for running the models:

```
fit_model <- function(rec_obj, model) { # get: recipe and model, returns: fit object
  fit(model, category ~ ., data = juice(rec_obj, all_predictors(), all_outcomes())) }
# a function that creates data frame with all the possible combinations between model & recipe
all_comb_mod_rec <- function(name_rec , names_models) {
  n_mods <- length(names_models)
  tibble( "name_rec" = rep(name_rec, n_mods), "model_name" = names_models)  }
# a function that returns the model and recipe predictions for a particular split.
pred_mod <- function(split_obj, rec_obj, model_obj) {
  mod_data <- bake(rec_obj,
                   new_data = assessment(split_obj),
                   all_predictors(), all_outcomes())
  out <- mod_data %>% select(category)
  out$predicted <- predict(model_obj, mod_data)$.pred_class
  out
}
# a function to train the recipe
train_recipe <- function(rec_name, model_name,splits, lst_recs, lst_mods) {
  cat("start working:" ,rec_name, model_name, paste(Sys.time()), "\n")
  splits_prepped <- map(splits, prepper, recipe = lst_recs[[rec_name]])
  splits_fit <- map(splits_prepped, fit_model , model = lst_mods[[model_name]] )
  splits_pred <- pmap(
    lst(split_obj = splits, rec_obj = splits_prepped,  model_obj = splits_fit),
    pred_mod
  )
  res <- map_dfr(splits_pred, accuracy, category, predicted)$.estimate
  name_res <-  paste0(rec_name,".", model_name)
  cat("mean cv accuracy", mean(res) , "\n")
  tibble(res) %>% setNames(name_res)
}
# a function to compute acc for each of the models & recipes
acc_fun <- function( df_mods_recs, lst_recs ,lst_mods ,  cv_df) {
  cv_df <- cv_df %>%  bind_cols(
      map2_dfc(df_mods_recs$name_rec, df_mods_recs$model_name, train_recipe,
               splits = cv_df$splits, lst_recs = lst_recs, lst_mods = lst_mods))
  cv_df <- cv_df %>% pivot_longer(cols = cv_df %>% select(-c(id,splits)) %>% names() ,
                                  names_to = "recipe", values_to = "Accuracy") %>%
    select(id, recipe, Accuracy) %>% separate(recipe, c("rec", "model"), sep ="\\.")
}
```

Class Imbalance

category	n
cakes_cupcakes_snack_cakes	3786
candy	7584
chips_pretzels_snacks	3680
chocolate	3772
cookies_biscuits	5284
popcorn_peanuts_seeds_related_snacks	7645

As you can see the classes are not balanced (at a light level), and if I want to use accuracy metric for assessing the performance of the models - I should handle this.

Strategies: Upsample, Downsample, Smot

I will start by creating some features just to run the models at this stage, later I'll examine each of the strategies in depth.

```
# For nutrients cols: function to check if we have more then 5% values above 0
is_remove <- function(col) (sum(col>0) / length(col)) < 0.05
nutr_to_rm <- map_lgl(snacks_tr_imbalance %>% select(starts_with("nutr_")), is_remove)
nutr_to_rm <- names(nutr_to_rm[nutr_to_rm==TRUE]) # keep the names of the nutrients to drop
# let's create brand_size col:
data_for_brand <- snacks_tr_imbalance %>% group_by(brand) %>% count(sort=T)
kable (t( quantile( data_for_brand %>% pull(n) , c(0.1,0.25, 0.5, seq(0.7,1,.025) ) ) ))
```

10%	25%	50%	70%	72.5%	75%	77.5%	80%	82.5%	85%	87.5%	90%	92.5%	95%	97.5%	100%
1	1	1	2	2	2	2	2	3	3	4	4	6	7	13	43

```
small_brands <- data_for_brand %>% filter(n ==1) %>% pull(brand)
medium_brands <- data_for_brand %>% filter(n < 11 && n>=2) %>% pull(brand)
# I'll choose 10 top ingredients (excluding sugar and salt) and create dummy vars for them
n_top_ingredient(data = as_tibble(snacks_tr_imbalance), num = 12,  by_category = FALSE) %>%
  filter(! (ingredient %in% c("sugar","salt"))) %>% pull(ingredient) %>% t() %>% kable()
```

soy lecithin	corn syrup	citric acid	water	cocoa butter	niacin	riboflavin	folic acid	wheat flour	reduced iron

I added dummies for these top words and also added `brand_size` col according to what I defined above (See code)

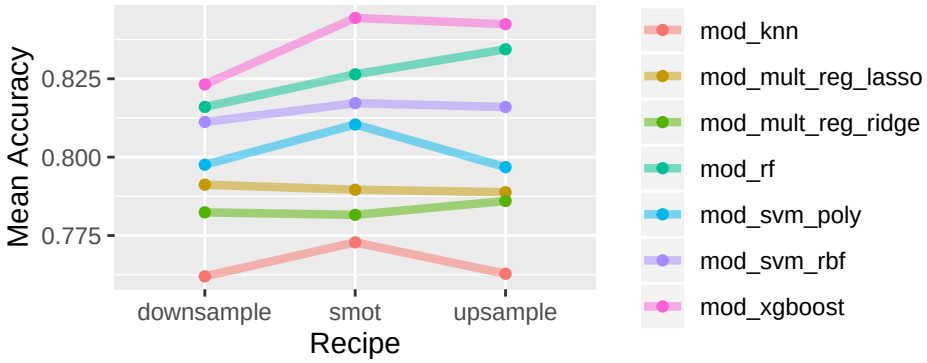
Recipes:

```
rec_imb_upsample <- recipe(category~. , data = snacks_tr_imbalance2) %>%
  # We'll remove columns we now don't need:
  step_rm("idx","ingredients","new_ingredient", "description", "serving_size_unit", "household_serving_fulltex
  step_rm(nutr_to_rm) %>%
  step_meanimpute(household_serving_fulltext_num) %>% # Impute average instead of NA's:
  step_normalize(all_numeric(), -starts_with("ing_")) %>% # normalize
  step_unknown(all_nominal(), -all_outcomes()) %>% # unknown instead of NA's, other for uncomoon values:
  step_other(all_nominal(), other = "other1", threshold = 0.01) %>%
  step_novel(all_nominal(), -all_outcomes()) %>% # step novel for possible new categories in val set:
  step_dummy(all_nominal(), -all_outcomes(), one_hot = FALSE) %>% # step dummy for categorical features:
  step_upsample(category, over_ratio = 1, seed= 123) # upsample
rec_imb_downsample <- recipe(category~. , data = snacks_tr_imbalance2) %>%
  step_rm("idx","ingredients","new_ingredient", "description", "serving_size_unit", "household_serving_fulltex
  step_rm(nutr_to_rm) %>%
  step_meanimpute(household_serving_fulltext_num) %>%
  step_normalize(all_numeric(), -starts_with("ing_")) %>%
  step_unknown(all_nominal(), -all_outcomes()) %>%
  step_other(all_nominal(), other = "other1", threshold = 0.01) %>%
  step_novel(all_nominal(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = FALSE) %>%
  step_downsample(category, under_ratio = 1, seed= 123)
rec_imb_smote <- recipe(category~. , data = snacks_tr_imbalance2) %>%
  step_rm("idx","ingredients","new_ingredient", "description", "serving_size_unit", "household_serving_fulltex
  step_rm(nutr_to_rm) %>%
  step_meanimpute(household_serving_fulltext_num) %>%
  step_normalize(all_numeric(), -starts_with("ing_")) %>%
  step_unknown(all_nominal(), -all_outcomes()) %>%
  step_other(all_nominal(), other = "other1", threshold = 0.01) %>%
  step_novel(all_nominal(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = FALSE) %>%
  step_downsample(category, under_ratio = 1.5) %>%
  step_smote(category, over_ratio = 1, seed = 123)
lst_rec_imb <- list("upsample" = rec_imb_upsample, "downsample" = rec_imb_downsample,"smot" =rec_imb_smote )
```

```
# model list for imbalance stage
mod_lst <- list("mod_mult_reg_ridge" =mod_mult_reg_ridge, "mod_mult_reg_lasso" = mod_mult_reg_lasso,
                "mod_rf" = mod_rf ,  "mod_xgboost" = mod_xgboost,
                "mod_svm_rbf" = mod_svm_rbf, "mod_svm_poly" = mod_svm_poly,  "mod_knn" =mod_knn)
```

```
set.seed(123) # cv splits for imbalance
cv_splits_imb <- vfold_cv(snacks_tr_imbalance2,v=10, strata  = category)
# all combinations to consider
all_comb_mods_recs <- map_dfr(names(lst_rec_imb), .f =all_comb_mod_rec  , names_models = names(mod_lst)  )
```

```
acc_all_mod_rec_imb <- acc_fun( df_mods_recs = all_comb_mods_recs, lst_recs = lst_rec_imb  ,
                                lst_mods = mod_lst , cv_df =cv_splits_imb )
```



The differences between the upsample and the smot seem pretty negligible. In addition, it seems that downsample is less preferable.

Top 5 Recipes

model	rec	mean_acc
mod_xgboost	smot	0.8444045
mod_xgboost	upsample	0.8423966
mod_rf	upsample	0.8344075
mod_rf	smot	0.8264138
mod_xgboost	downsample	0.8232025

Mean Acc by Recipe

rec	mean_acc
smot	0.8060638
upsample	0.8038871
downsample	0.7976513

Let's compare for each model between upsample to smot: (upsample- smot)

model	pvalue	conf_int_5%
mod_mult_reg_ridge	0.3410	[-0.0055 , 0.0144]
mod_mult_reg_lasso	0.8524	[-0.0102 , 0.0086]
mod_rf	0.0317	[9e-04 , 0.0151]
mod_xgboost	0.6838	[-0.0128 , 0.0088]
mod_svm_rbf	0.7121	[-0.0085 , 0.006]
mod_svm_poly	0.0047	[-0.0219 , -0.0054]
mod_knn	0.0605	[-0.0206 , 5e-04]

It can be seen that the differences between "upsample" to "smot" are quite small, and there are only 2 models for which the difference between the two strategies is significant - mod_svm_poly for which smot is better and mod_rf in which upsample is better .
My Decision: I'll proceed to the next step only with upsample. Also, I want at this point to get rid of KNN model since I don't think it will get better in the next stages and I want to save computation time.

Numerical Features

Strategies:

Serving_size & nutrients & Household serving fulltext num:

- Regular (no transformation)
- Log trandformation
- Trim top and/or bottom percentile (nutrients - only top)

Nutrients - I'll not trim the bottom edge, because I created these features by setting 0 for each product that did not contain the nutrient. (I want to know which nutrients have amount 0)

Ingredient Score: As we saw in Part A, I created a variable that counts for each product - how many ingredients it contained from the list of the top n ingredients for each category.

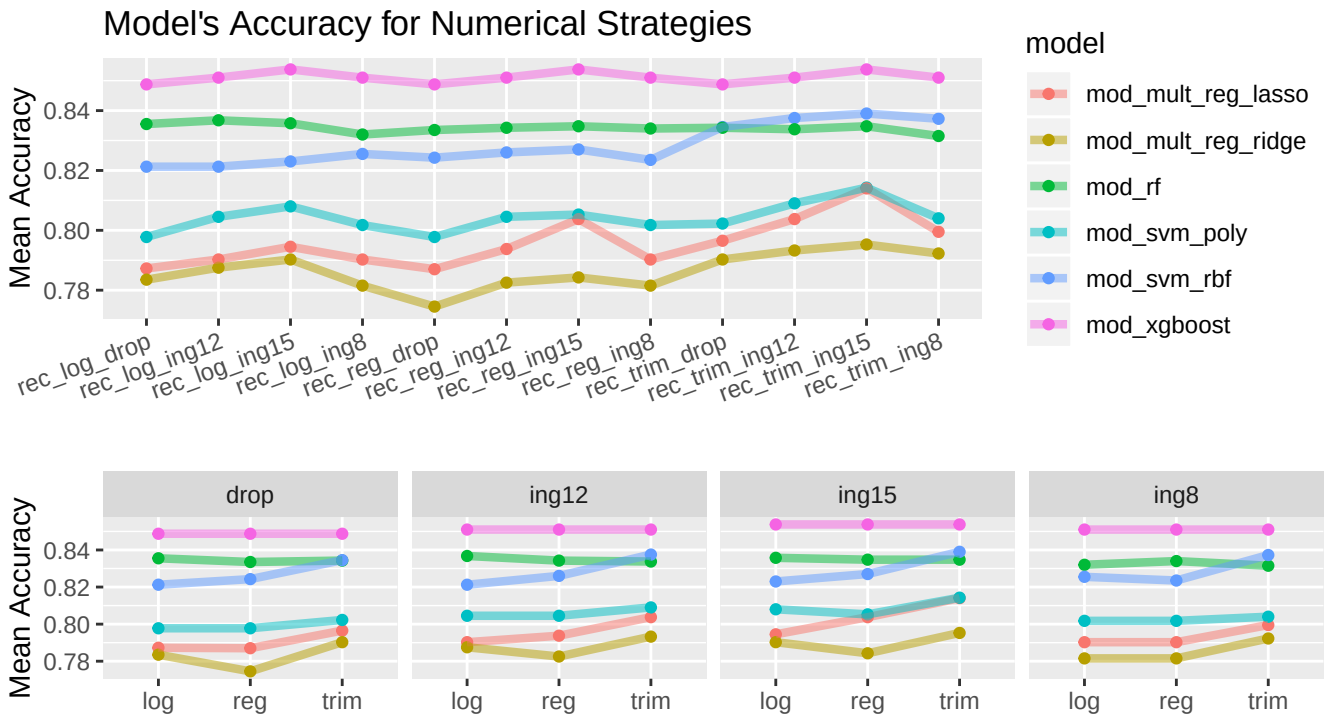
- Include with 8,12,15 top ingredients by category
- drop

Note: at the end I will normalize all the numeric cols (except for the binary cols).
NA's: numerical - mean impute , Categorical - mode ipute (*There are a very small number*)

I added again the variables I added in the previous step (see code)

```
# add ingredient scores using n = 8,12,15  (using functions from part a)
snacks_tr_numeric2 <- snacks_tr_numeric2 %>% inner_join(create_scores(snacks_tr_numeric2,8) %>% select(-catego
  inner_join(create_scores(snacks_tr_numeric2,12) %>% select(-category), by = "idx" ) %>%
  inner_join(create_scores(snacks_tr_numeric2,15) %>% select(-category), by = "idx" )
trim_fun <- function(col, bottom = F) {  # trim function: trim the highest and/or lowest values
  q_99 <- quantile(col, probs = .99) ; q_1 <- quantile(col, probs = .01)
  col <- ifelse(col >  q_99, q_99, col)
  if(bottom)  col <- ifelse(col<q_1, q_1, col)
  col  }
```

All the code for the recipes can be seen in the notebook.



Left: Top 5 Recipes, **Right:** for the leading models- Xgb, RF, SVM: (see code - these two tables are relevant only for these 3 models)

Top 5 Recipes

rec	mean_acc
rec_trim_ing15	0.8251814
rec_trim_ing12	0.8213909
rec_trim_ing8	0.8192615
rec_reg_ing15	0.8181424
rec_trim_drop	0.8177644

Mean Acc Ing Score

ing_score	mean_acc
ing15	0.8395179
ing12	0.8380716
ing8	0.8374316
drop	0.8366240

Mean Acc Numeric Strategy

numeric	mean_acc
trim	0.8405996
reg	0.8368292
log	0.8363051

Conclusions: It can be seen that there is no such significant difference between the strategies. It always seems better to include the score feature, but it is not entirely clear whether 8,12 or 15 is better. For leading models (rf, xgboost, svm_rbf) it seems better to take a score of 15. Also, it seems that there is not much difference between the log, trim, regular strategy. trim is slightly higer then the others, so I decided to go with trim. I will also say goodbye to ridge and lasso models - since in the previous two stages they were not good and and I don't believe they will rise above xgboost or random forest.

Categorical Features

I'll create new features:

- `brand_size` - classiy to "small", "medium" , "large", "huge" brand by the number of products.
- `n_distinct_cat` - classify brand by the number of (disticnt) categories it sells:
- Brand keywrods - classify by keywords of brand name (droping words appears less then 50 times)
- Serving fulltext keywords - classify using keywords I created at part A + step_other .01
- Serving fulltext Feature Hashing (num trems - 10)

Strategies::

Brand 1 - `brand_size`	Brand 2 - `n_distinct_cat`	Brand 3 - keywords	Serving Fulltext
brand size	numerical	keywords	keywords
drop	categorical	drop	hashing
-	drop	-	-

Total of 24 strategies but due to considerations of time , I will run all the recipes with brand size and then I'll choose top recipes and I'll run them without brand_size and check whther there is any difference.

My strategy for choosing the best words to include in the model:
Since we have more then 2 categories - I decided to use *entropy* metric as a impurity measurment. For each of the words:
- I'll calculate the number of appearance in each of the categories
- Then I'll compute the entropy: $\sum_{i=1}^{6} -log(\hat{p}_i) \cdot \hat{p}_i$ where Σ denotes the sum over the categories, $\hat{p}$ denotes the estimated frequency of the word in category i. This metric expresses the impurity (of the categories) for each word. - I'll get "purity score" for each of the words. The smaller it is - that means there is more impurity - meaning fewer categories or one very large category. Therefore, I would like to take all the words for which cross enrtopy is the smallest ($\sim$ <1.5).
In addition, I will look at words that appear at least 50-100 times in the data. *(I used that* $0 \cdot \log(0) = 0$*)*

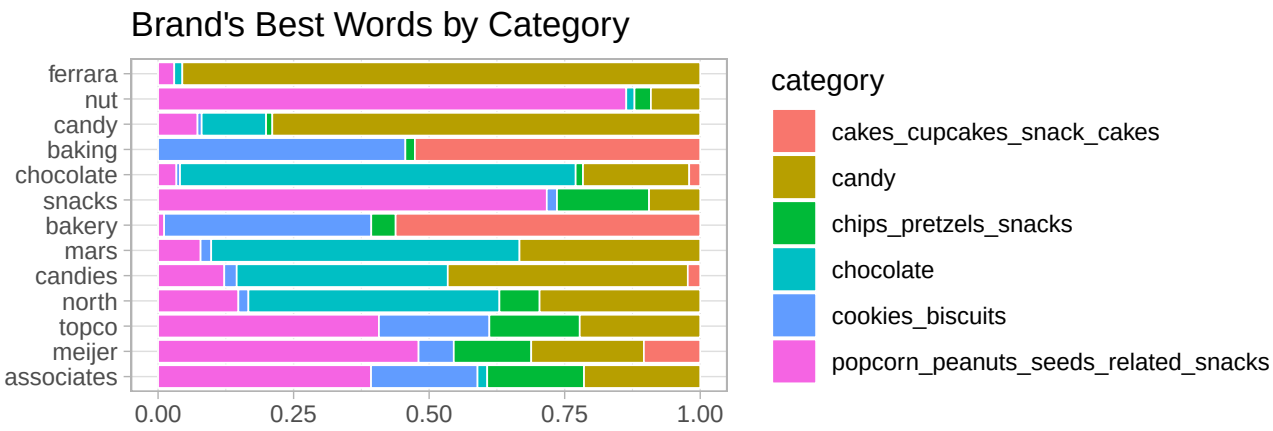
```
# add the score for ingredient
snacks_tr_categorical2 <- snacks_tr_categorical %>% inner_join(create_scores(snacks_tr_categorical,15) %>% sel
# find the nutrients to remove: all the nutrients in which less then 5% are greater then 0
nutr_to_rm <- map_lgl(snacks_tr_categorical2 %>% select(starts_with("nutr_")), is_remove)
nutr_to_rm <- names(nutr_to_rm[nutr_to_rm==TRUE])
```

```
# a function that classify brand size according to the no. of products it sells
classify_brand_size <- function(data) {
  data_brand <- data %>% group_by(brand) %>% count(sort=T)
  quantiles <- quantile( data_brand %>% pull(n) , c(.5, .85,.99 ) )
  small_brands <- data_brand %>% filter(n <= quantiles[1]) %>% pull(brand)
  medium_brands <- data_brand %>% filter(n > quantiles[1] && n <=quantiles[2] ) %>% pull(brand)
  large_brands <- data_brand %>% filter(n > quantiles[2] && n <= quantiles[3]) %>% pull(brand)
  data %>% mutate(brand_size = case_when(
    brand %in% small_brands ~ "small", brand %in% medium_brands ~ "medium",
    brand %in% large_brands ~ "large", TRUE ~ "huge") )
}
snacks_tr_categorical2 <- classify_brand_size(snacks_tr_categorical2)
# create 'n_distinct_cat' - for each brand: compute the number of
# distinct categories and merge it to the data
snacks_tr_categorical2 <-  snacks_tr_categorical2 %>%  group_by(brand) %>%
  summarise(n_distinct_cat = n_distinct(category)) %>% inner_join(snacks_tr_categorical2)
```

Let's create some help functions for handling text features:

```
# a function to count words by each category and compute entropy
top_words_by_category_entropy <- function(data, col) { # a function to
  data %>% mutate_at({{col}}, as.character) %>% group_by(category) %>%
  unnest_tokens(word, {{col}}) %>%
  anti_join(stop_words) %>%
  filter(!str_detect(word, pattern = "[[:digit:]]"), # removes any words with numeric digits
    !str_detect(word, pattern = "[[:punct:]]"),      # removes any remaining punctuations
    !str_detect(word, pattern = "(.)\\1{2,}"),       # removes any words with 3 or more repeated letters
    !str_detect(word, pattern = "\\b(.)\\b")        # removes any remaining single letter words
  ) %>%  count(word, sort = T) %>% group_by(word) %>% mutate(pct = n/sum(n), entropy = entropy_fun(pct) , sum_
}
# a function that returns the best words  that appears at least n times and with entropy < entropy_max
order_words_by_entropy <- function(data, col,  n_words, entrop_max) {
  data %>% filter(sum_n>n_words,entropy<entrop_max) %>% arrange(entropy) %>%
    distinct({{col}}) %>% pull({{col}})  }
# plot the chosen words
plot_chosen_words <-  function(data, col, order_words) {
  data  %>% filter({{col}} %in% order_words ) %>%
    ggplot(aes(x = reorder({{col}}, -entropy),  y = n , fill = category)) +
    geom_bar(stat = "identity",position = "fill", color = "white", size = .3) + coord_flip() + theme_light() +
    labs(x="", y="")
}
# a function to create columns for the chosen words and merge it to the data
merge_chosen_words <- function(data, col, order_words, starter) {
  df_to_mrege <- data %>% mutate_at({{col}}, as.character) %>%
  unnest_tokens(word, {{col}}) %>%
  anti_join(stop_words) %>%
  filter(word %in% order_words) %>%     # filter for only words in the wordlist
  count(idx, word) %>%                  # count word useage by  ID
  spread(word, n) %>%                   # convert to wide format
  map_df(replace_na, 0)   %>%  rename_at(vars(-idx), ~(paste0(starter, .))  )
  data %>% left_join(df_to_mrege) %>%
  mutate_at(vars(starts_with(starter)), ~(ifelse(is.na(.),0,.)))
}
```

```
# create data with words & counts of brand + entropy
brnd_words_dat <- top_words_by_category_entropy(snacks_tr_categorical2, "brand")
chosen_words <- order_words_by_entropy(brnd_words_dat,word, 50,entrop_max =  1.4)
# let's take a look at the chosen words of brand:
plot_chosen_words(brnd_words_dat, word, chosen_words) + ggtitle("Brand's Best Words by Category")
```

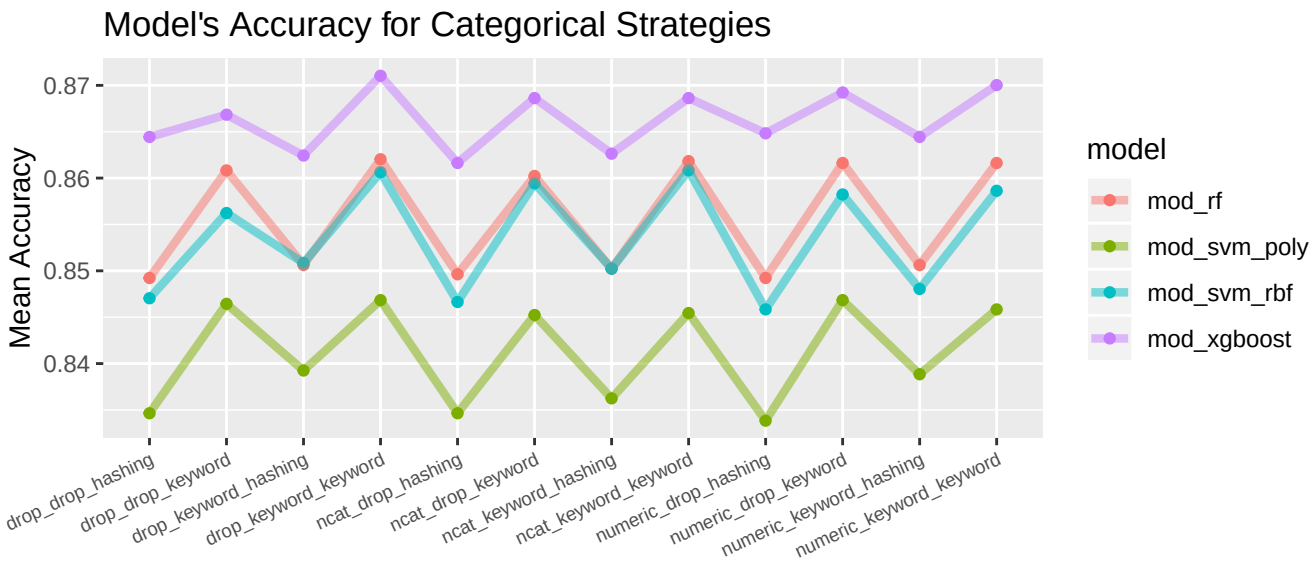



```
# add the best words to the data
snacks_tr_categorical3 <- merge_chosen_words(data =snacks_tr_categorical2, col = "brand",
                                             order_words =chosen_words, starter = "brnd_"  )
```

```
# fulltext - create "household serving update" using the keyword from part A
snacks_tr_categorical4 <- household_serving_update_fun(snacks_tr_categorical3,
                                                       household_serving_fulltext, key_words_list )
# prepare the data in order to find out the best words
fulltext_keys_data <- snacks_tr_categorical4 %>% group_by(category, household_serving_update) %>%
  count(sort = T) %>% group_by(household_serving_update) %>%
  mutate(pct = n/sum(n), entropy = entropy_fun(pct) , sum_n = sum(n))
# best words: entropy < 1.6 , appears more then 50 times.
order_fulltext_keys_by_entropy <- order_words_by_entropy(fulltext_keys_data,"household_serving_update", 50,ent
# make everything that desn't belong to the selected word list "other"
snacks_tr_categorical5 <- snacks_tr_categorical4 %>%
  mutate_at("household_serving_update" ,~(ifelse(.%in%order_fulltext_keys_by_entropy, . , "other")))
```

The code for the recipes is available in the nootbook.

```
set.seed(123)
cv_splits_categorical <- vfold_cv(snacks_tr_categorical5,v=10, strata  = category)
```



Top 5 Recipes

rec	mean_acc
rec_brndsize_drop_keyword_keyword	0.8601339
rec_brndsize_ncat_keyword_keyword	0.8591839
rec_brndsize_numeric_keyword_keyword	0.8590317
rec_brndsize_numeric_drop_keyword	0.8589823
rec_brndsize_ncat_drop_keyword	0.8583820

Mean Acc Brand2 Strategy

brand2	mean_acc
drop	0.8543359
numeric	0.8542348
ncat	0.8538854

Mean Acc Fulltext Strategy

fulltext	mean_acc
keyword	0.8588829
hashing	0.8494211

Mean Acc Brand3 Strategy

brand3	mean_acc
keyword	0.8549109
drop	0.8533931

I would like to run the 3 top recipes **with and without** brand_size, only with RF and Xgboost models: (See code)

brandsize	n_cat	mean_acc
drop	ncat	0.8667283
brndsize	drop	0.8665321
drop	drop	0.8659273
brndsize	numeric	0.8658277
brndsize	ncat	0.8652275
drop	numeric	0.8651283

Conclusions:

- Hashing on description seems to be very bad.
- I want to continue with keyword for brand and fulltext.
- You can see that there is not much difference if we include brand_size & n_distinct_cat or drop one or both of them. Anyway I want to leave `brand_size` for the next step when I add more features. `n_distinct_cat` - did not bring so much improvement, so I will throw it away.

Text Features

Strategies
Ingredients + Description:

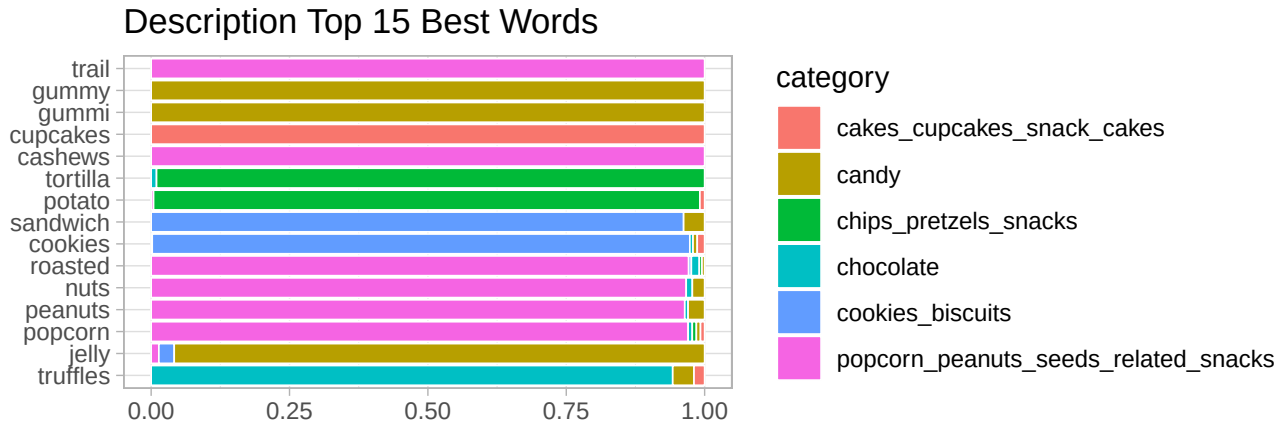
- keywords
- hashing (description - num terms 2^5 , ingredient - num terms 2^8)
- drop

Also, I want to reconsider at this point that I have most of the features the strategies: **upsample** or **smot**. Moreover, since I'm using now keyword from ingredient columne, I will consider whether or not to drop the **ingredient score** I created.

In addition - I decided to finally choose the keywords for fulltext, brand, description ingredient in this stage.

I'll add the selected features from previots steps (same code - see notebook)

```
# Let's see what words to take for description:  (same functions as on brand col)
description_words_dat2 <- top_words_by_category_entropy(snacks_tr_txt5, "description")
chosen_words_description <- order_words_by_entropy(description_words_dat2 ,word, 50,entrop_max =  1.6)
# let's take a look at the top 15 chosen words for description
plot_chosen_words(description_words_dat2, word, chosen_words_description[1:15]) + ggtitle("Description Top 15
```



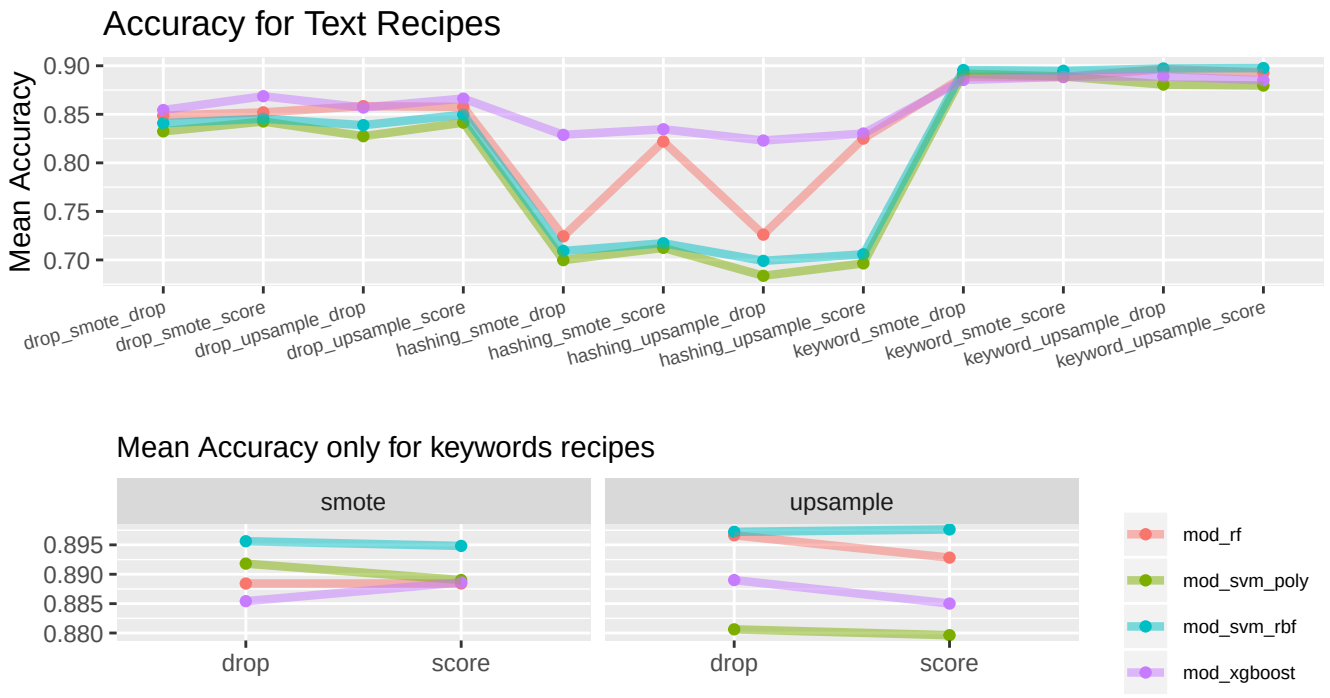
Wow! We found very good words!

```
# add the best words to the data
snacks_tr_txt6 <- merge_chosen_words(data =snacks_tr_txt5, col = "description",
                                     order_words =chosen_words_description, starter = "descr_"  )
```

```
# ingredient
ingred_dat <- n_top_ingredient(snacks_tr_txt6, 2000)  %>% group_by(ingredient) %>%
  mutate(pct = n/sum(n), entropy = entropy_fun(pct) , sum_n = sum(n)) %>% arrange(-sum_n)
chosen_ingred <- order_words_by_entropy(ingred_dat,"ingredient", 100 ,entrop_max =  1.5)
```

```r
# we have expressions and not words, so i can't ust merge_chosen_words:
# I will use these two functions in order to create cols with the counts of thw chosen words:
str_count_na <- function(string, pattern = "") {
  n <- str_count(string, pattern)
  ifelse(is.na(n), 0, n)  }
count_words <- function(essays, chosen_words) {
  counts <- map(chosen_words, ~str_count_na(essays, .x))
  names(counts) = janitor::make_clean_names(str_c("ingrd_", chosen_words))
  counts  }
snacks_tr_txt7 <- snacks_tr_txt6 %>%
  bind_cols( map_dfr(snacks_tr_txt6$new_ingredient, count_words, chosen_words =chosen_ingred ) )
```

The recipe code can be seen in the notebook.



Hashing seems to be pretty bad, and drop less is better too. The four models are quite similar. svm and rf have greatly improved and even sometimes exceed xgboost.

Conclusions: from looking at the **average** and **minimum** accuracy over the folds by recipe & model I decided to continue with:

- SVM rbf: keywords, upsample, ingredient score
- Random Forest: keywords, upsample, drop ingredient score
- Xgboost - keywords, upsample, drop ingredient score

More Desicions: (1) I decided to drop svm poly model because on average (including previous steps) it is less good than the others (consideration of running time).
(2) I'll save the selected words in this section for: barnd, fulltext, description, ingredient. I'll also use them in the final model. Here are the no. of words:

chosen_word_fulltext	chosen_word_brnd	chosen_word_descr	chosen_word_ingrd
13	18	78	138

I created `full_features()` function that take care of adding all the relevant features I decided to add in the previous steps to the final model. (See code)

Tuning

From now on I will get rid of near zero variance nutrients using `step_nzv()` instead of using `nutr_to_rm`

```
snacks_tr_tunning2 <- full_features(snacks_tr_tunning)
```

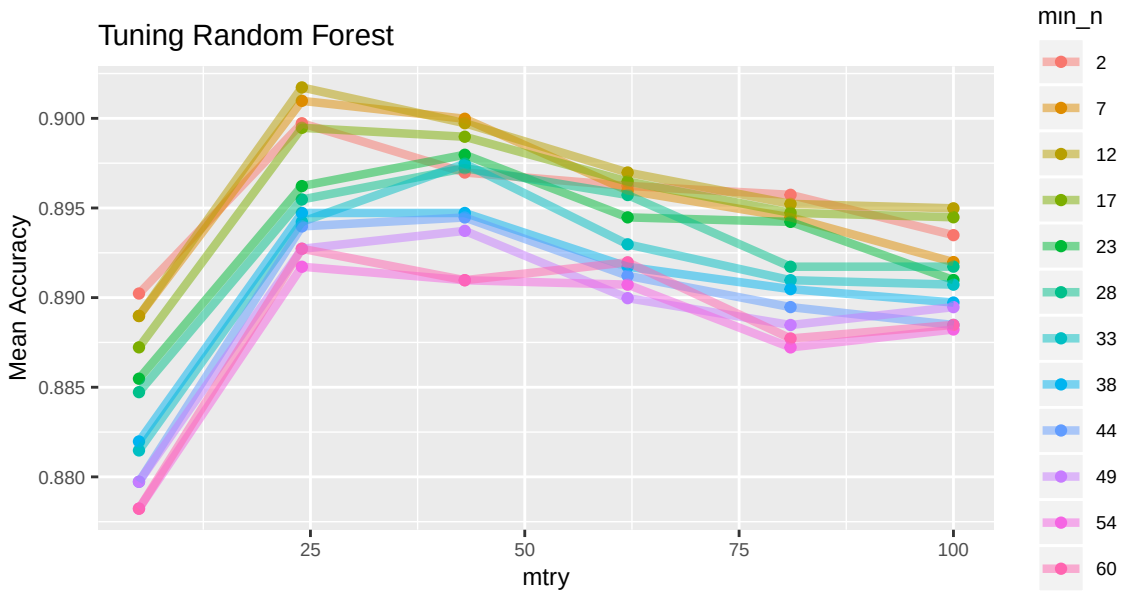
Tuning Random Forest

Final recipe for Random Forest:

```
final_rec_random_forest <-  recipe(category~. , data = snacks_tr_tunning2 %>% select(-idx) ) %>%
  step_nzv(starts_with("nutr_"), freq_cut = 99/1)  %>%
  step_rm(starts_with("score_")) %>%
  step_meanimpute(household_serving_fulltext_num) %>%
  step_mutate(serving_size =trim_fun(serving_size,bottom = T)) %>%
  step_mutate(household_serving_fulltext_num =trim_fun(household_serving_fulltext_num,bottom = T)) %>%
  step_mutate_at(starts_with("nutr_"),fn =  trim_fun) %>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric(), -starts_with("score"), -starts_with("brnd_"),
                 -starts_with("descr"), -starts_with("ingrd_")) %>%
  step_unknown(all_nominal(), -all_outcomes()) %>%
  step_other(all_nominal(), other = "other1", threshold = 0.01) %>%
  step_novel(all_nominal(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = FALSE) %>%
  step_upsample(category, over_ratio = 1, seed= 123)
```

I want to set the number of the tree to be 1000 since more trees are better for random forest (average of trees). Thus I will have to tune `mtry` and `min_n`. I first tried a grid created by R to get an idea of which combination of parameters is good. Using the results, I chose values intelligently for another grid to test(see code). Here are the top 5 combinations:

mtry	min_n	.metric	mean	n	std_err
24	12	accuracy	0.9017271	10	0.0055661
24	7	accuracy	0.9009784	10	0.0052872
43	7	accuracy	0.8999778	10	0.0053750
43	12	accuracy	0.8997303	10	0.0054394
24	2	accuracy	0.8997284	10	0.0052255



I'll continue with the 2 best combinations for the model selection stage.

Tuning Xgboost

Final recipe for xgboost same as for random forest.

This model has many parameters to tune. I decided to stick with a number of 1000 trees, and do tuning on `tree_depth`, `min_n`, `lose_reduction`, `mtry`, `learn_rate`. For some of the parameters I selected a range of values that I belive the optimal

values will be in. Since these are a lot of parameters, I will use `grid_latin_hypercube` function which will create a grid for me that covers pretty much all of the options space.

5 best combinations: (I'll continue with best two)

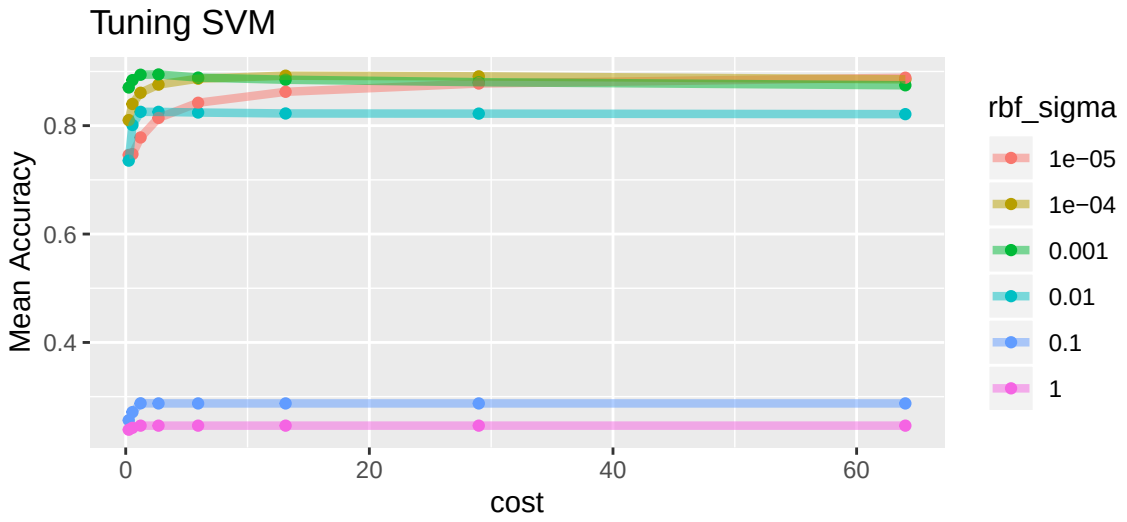
mtry	min_n	tree_depth	learn_rate	loss_reduction	.metric	mean	n	std_err
102	5	7	0.0318189	0.0000167	accuracy	0.9054847	10	0.0048402
27	11	11	0.0182188	1.1146334	accuracy	0.9037297	10	0.0055170
81	13	8	0.0083149	0.0000008	accuracy	0.9009772	10	0.0062956
23	6	12	0.0116911	0.0002708	accuracy	0.8997309	10	0.0053302
96	21	11	0.0442620	0.0000440	accuracy	0.8934877	10	0.0035938

Tuning SVM

The recipe is the same as the xgboost recipe, except that I now normalize all numeric variables (since it is SVM model) and used score ingredient. We have to tune `cost`,`rbf_sigma`. I chose range of values that I belive the optimal parameters will be in.

‘

The results from the two grids I chose:



5 best combinations: (I'll continue with best two)

cost	rbf_sigma	mean_acc
2.691800	1e-03	0.8944759
1.219014	1e-03	0.8939765
13.125366	1e-04	0.8922271
28.983157	1e-04	0.8909790
5.943977	1e-03	0.8887246

I got from default parameters mean accuracy of 0.896 so I'll continue with default parameters and with the best combination from the grid.

MODEL SELECTION

Now, I will run each of the models on the 2 best combinations from the tuning step, then I will choose the best.

```
snacks_tr_selection2 <- full_features(snacks_tr_selection) # data preparation
```

```
# definition of final models:
final_mod_rf_1 <- rand_forest( mtry = 24, trees = 1000, min_n = 12) %>%
  set_mode("classification") %>% set_engine("ranger")
```

```
final_mod_rf_2 <- rand_forest( mtry =24, trees = 1000, min_n = 7) %>%
  set_mode("classification") %>% set_engine("ranger")
final_mod_xgb_1 <- boost_tree(
  trees = 1000, tree_depth = 7, min_n = 5 , loss_reduction = 0.0000167, mtry = 102 , learn_rate = 0.0318) %>%
  set_engine("xgboost") %>% set_mode("classification")
final_mod_xgb_2 <- boost_tree(
  trees = 1000, tree_depth = 11, min_n = 11 , loss_reduction =1.11, mtry = 27 , learn_rate =0.0182)%>%
  set_engine("xgboost") %>% set_mode("classification")
final_mod_svm_1 <- svm_rbf() %>% set_engine("kernlab") %>% set_mode("classification") #default
final_mod_svm_2 <- svm_rbf(cost = 2.6918, rbf_sigma = 0.001) %>%
  set_engine("kernlab") %>% set_mode("classification")
list_selection_models <- list( # final list of models
  "final_mod_rf_1" = final_mod_rf_1, "final_mod_rf_2" = final_mod_rf_2,
  "final_mod_xgb_1" = final_mod_xgb_1, "final_mod_xgb_2" = final_mod_xgb_2,
  "final_mod_svm_1" = final_mod_svm_1, "final_mod_svm_2" = final_mod_svm_2 )
```

```
list_selection_rec <- list( # the final list of recipes:
 "final_rec_rf_xgb" = final_rec_rf_xgb,
  "final_rec_rf_xgb" = final_rec_rf_xgb ,
  "final_rec_svm_rbf" = final_rec_svm_rbf )
```

model	rec	mean_acc
final_mod_xgb_1	final_rec_rf_xgb	0.9171569
final_mod_xgb_2	final_rec_rf_xgb	0.9128704
final_mod_svm_1	final_rec_svm_rbf	0.9120611
final_mod_svm_2	final_rec_svm_rbf	0.9118562
final_mod_rf_1	final_rec_rf_xgb	0.9087896
final_mod_rf_2	final_rec_rf_xgb	0.9085871

I decided to continue to assessment with the best combination of each model, ie with `final_mod_rf_1`, `final_mod_xgb_1`, `final_mod_svm_1`.

Assessment of The Final Models on the Test Set

```
train_combained <- full_features(snacks_tr)
test_comb <- full_features(snacks_te)
```

I made a recipe to get data with hashing to put into the neural network in Python(without upsample). See code.

Let's compute the test error on the 3 chosen models (of xgboost and of random forest). See code.

rf1	xgb1	svm1
0.9387209	0.9428166	0.9352552

Nice :) I'll continute with xgboost and random forest for prediction.

Nueral Network

I'll **briefly** summarize what I did with the networks, and you can see **everything** in the jupyter notebook.

I built:

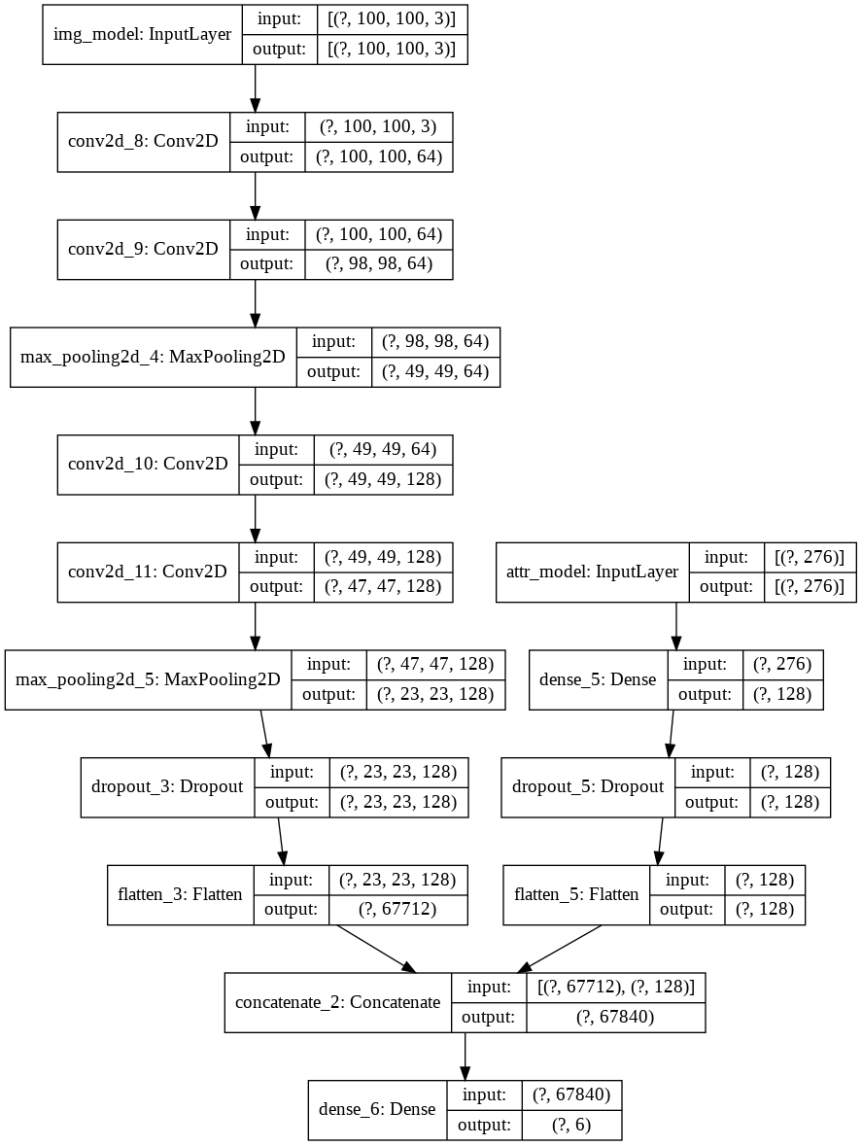
- CNN neural with only the snack's images.
- Combined Neural Network - images and final numerical data (with the features I created at all stages - including hashwords)
- Neural Network only with the final numerical data (no images)

Summarise Results:

- The images alone yielded an accuracy of ~ 60 percent on the test set.
- The images + data yielded an accuracy of ~ 93 percent.
- The data alone yielded an accuracy of about ~ 93.

Conclusion: - using images **probably** don't yield higher accuracy, and don't contribute much to the prediction. I'll note that perhaps smarter network construction and proper handling of both types of data would probably have yielded better results.

The architecture of the combined neural network I built:



I highly recommend looking at the jupyter notebook to see the complete code with which I built the network.

Notes: For building the networks, the validation I used was the same data as `snacks_te`. A **better way** was to take 0.8 from the rest of the train (ie our `snacks_tr`) and leave another 0.2 for validation. Then, **after** building the network - check its performance on `snacks_te` (as I did in the models in R). I didn't have time to make changes due to a very very long run time. **However**, the train error and the validation error have been pretty close in the last epochs so I am less afraid of overfitting.

However, I decided to try and submit the predictions from the model that combines the images and other features.

Building the Final Models for Prediction

Like I said, I'll submit predictions from my 3 best models - both xgboost combinations and the random forest model.

```
food_test_final_model <- full_features(food_test_initial, is_test = T)
food_train_final_model <- full_features(food_train_initial)
```

```
build_final_model <- function(rec, model)  {
  final_prep = prep(rec, food_train_final_model) # prep
  train_baked = bake(final_prep, food_train_final_model) # bake
  test_baked = bake(final_prep, food_test_final_model)

  fit_final_mod = model %>%  fit (category ~., data = train_baked) # fit
  pred_test = fit_final_mod %>%
    predict(new_data = test_baked, type = "class") %>%
    mutate(idx = food_test_final_model$idx)

  return(list( "results" = pred_test, fit = fit_final_mod ))
}
```

First Model: XGboost 1

```
fit_xgb1  <- build_final_model(final_rec_rf_xgb ,final_mod_xgb_1)
fit_xgb1$results %>%select(idx, .pred_class) %>%  write_csv("model101.csv")
```

Second Model:Random Forest 1

```
fit_rf1 <- build_final_model(final_rec_rf_xgb ,final_mod_rf_1)
fit_rf1$results %>% select(idx, .pred_class) %>%  write_csv("model102.csv")
```

Third Model: Neural Network

```
results_nn  <- read.csv("final_nn_preds_03.csv")
results_nn %>% arrange(idx) %>%
  select(idx, Prediction) %>%  rename(".pred_class" = Prediction) %>%
  write_csv("model103.csv")
```

General Comments:

- I could at any stage move forward with the ideal recipe for each model, but due to considerations of running time, I decided not to do so.
- It is clear to me that it is not necessarily correct to make the decisions editively. So I tried to combine previous steps even after that.
- I could add another stage to take care of feature selection for the final model - I didn't do that because anyway the models I chose already know how to give more weights to better features and fewer weights for not good features.