Final Project - Classification

Etay Nahum - 313163735

Strategy

preproccessing

- First i removed the variables (nutrients) with large number of NA (cols_with_NA table from before), removed the red,blue,yellow featured from the images, and finally removed the serving size unit feature.
- The main goal is to make the data all numeric and normalize so i can use different models.
- I used the information i got from the EDA in order to make mostly dummy variables based on the nominal variables.
- For ingredients i made dummy variable based on unique ingredient for each category, plus made length and number of word based on the ingredients variable and applied log transformation.
- For brand i did the same, mostly for the most appearance brands.
- For description i made dummy variable if the description contains the name of the category.
- For household_serving_fulltext i made dummy variable based on the type of serving, as i presented in the first graph.
- For all the other nutrients variables i applied normalization.
- replaced NA with 0, if nutrients was with NA the logical assumption that is zero, and for the new dummy variables the same logic is applied.

Models

- I split the training data into train_train and train_val.
- For the train_train data, i used CV with 5 folds to tune hyper parameters for the next 4 models: 1 - multinom regression: tuned penalty and mixture. 2 - knn model: tuned number of neighbors and the distance power. 3 - xgboost: tuned learning rate, number of tree and trees depth. 4 - random forest: tuned number of trees and number of feature to be taken each tree.
- After tuning and taking the best hyper parameters for each model, i fit them with those parameters on the entire train_train data and compare their accuracy on the train_val data, and takes the best performing model.

Splitting training data

```
set.seed(77)
split_obj <- initial_split(final_train , 0.8  , strata = category)
train_train <- training(split_obj)
train_val <- testing(split_obj)
```

The recipe

```
cols_NA_for_rec <- cols_with_NA %>%
  select(-category , -sum_of_rows) %>%
  colnames()

rec <- recipe(category~. , train_train) %>%
  update_role(idx , new_role = "ID") %>%
  step_rm(all_of(cols_NA_for_rec) ,serving_size_unit , red , blue , green) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_mutate(len_ing = str_length(ingredients)) %>%
  step_mutate(n_word_ing = str_count(ingredients , "\\w+")) %>%
  step_mutate(brand_ferrara = str_detect(brand , "ferrara")) %>%
  step_mutate(brand_walmart = str_detect(brand , "wal-mart")) %>%
  step_mutate(brand_meijer = str_detect(brand , "meijer"))%>%
  step_mutate(brand_target = str_detect(brand , "target stores"))%>%
  step_mutate(brand_lindt = str_detect(brand , "lindt")) %>%
  step_mutate(brand_russel = str_detect(brand , "russell stores")) %>%
  step_mutate(brand_quality = str_detect(brand , "quality foods")) %>%
  step_mutate(brand_frankford = str_detect(brand , "frankford")) %>%
  step_mutate(brand_not = str_detect(brand , "not a")) %>%
  step_mutate(brand_nabisco = str_detect(brand , "nabisco")) %>%
  step_mutate(description_chocolate = str_detect(description , "chocolate")) %>%
  step_mutate(description_cookie = str_detect(description , "cookie|biscuit")) %>%
  step_mutate(description_cake = str_detect(description , "cake|cupcake")) %>%
  step_mutate(description_candy = str_detect(description , "candy"))%>%
  step_mutate(description_chips = str_detect(description , "chips|pretzel")) %>%
  step_mutate(description_pop_peanut = str_detect(description , "popcorn|peanut|seed")) %>%
  step_mutate(ingredients_milk = str_detect(ingredients , "milk"))%>%
  step_mutate(ingredients_cocoa = str_detect(ingredients , "cocoa")) %>%
  step_mutate(ingredients_chocolate = str_detect(ingredients , "chocolate")) %>%
  step_mutate(ingredients_butter = str_detect(ingredients , "butter")) %>%
  step_mutate(ingredients_emuls = str_detect(ingredients , "emulsifier")) %>%
  step_mutate(ingredients_vanilla = str_detect(ingredients , "vanilla")) %>%
  step_mutate(ingredients_wheat = str_detect(ingredients , "wheat")) %>%
  step_mutate(ingredients_palm = str_detect(ingredients , "palm")) %>%
  step_mutate(ingredients_syrup = str_detect(ingredients , "syrup")) %>%
  step_mutate(ingredients_soduim = str_detect(ingredients , "sodium")) %>%
  step_mutate(ingredients_gum = str_detect(ingredients , "gum")) %>%
  step_mutate(ingredients_starch = str_detect(ingredients , "starch")) %>%
  step_mutate(ingredients_water = str_detect(ingredients , "water")) %>%
  step_mutate(ingredients_yellow = str_detect(ingredients , "yellow")) %>%
  step_mutate(ingredients_red = str_detect(ingredients , "red"))%>%
  step_mutate(ingredients_blue = str_detect(ingredients , "blue")) %>%
  step_mutate(ingredients_citric = str_detect(ingredients , "citric")) %>%
  step_mutate(ingredients_sunflower = str_detect(ingredients , "sunflower")) %>%
  step_mutate(ingredients_canola = str_detect(ingredients , "canola")) %>%
  step_mutate(ingredients_organic = str_detect(ingredients , "organic")) %>%
  step_mutate(ingredients_veg = str_detect(ingredients , "vegetable")) %>%
  step_mutate(ingredients_almond = str_detect(ingredients , "almond")) %>%
  step_mutate(household_serving_fulltext_bag = str_detect(household_serving_fulltext,"bag"))%>%
  step_mutate(household_serving_fulltext_bar = str_detect(household_serving_fulltext,
                                                          "bar")) %>%
```

```
  step_mutate(household_serving_fulltext_brownie = str_detect(household_serving_fulltext,
                                                              "brownie")) %>%
  step_mutate(household_serving_fulltext_chip = str_detect(household_serving_fulltext,
                                                              "chip"))%>%
  step_mutate(household_serving_fulltext_cookie = str_detect(household_serving_fulltext,

  step_mutate(household_serving_fulltext_cracker = str_detect(household_serving_fulltext,
                                                              "cracker")) %>%
  step_mutate(household_serving_fulltext_cup = str_detect(household_serving_fulltext,
                                                              "cup"))%>%
  step_mutate(household_serving_fulltext_donut = str_detect(household_serving_fulltext,
                                                              "donut")) %>%
  step_mutate(household_serving_fulltext_grm = str_detect(household_serving_fulltext,
                                                              "grm")) %>%
  step_mutate(household_serving_fulltext_onz = str_detect(household_serving_fulltext,
                                                              "onz")) %>%
  step_mutate(household_serving_fulltext_package = str_detect(household_serving_fulltext,
                                                              "package|pkg|pack")) %>%
  step_mutate(household_serving_fulltext_piece = str_detect(household_serving_fulltext,
                                                              "piece|pcs")) %>%
  step_mutate(household_serving_fulltext_pop = str_detect(household_serving_fulltext,
                                                              "pop")) %>%
  step_mutate(household_serving_fulltext_pouch = str_detect(household_serving_fulltext,
                                                              "pouch")) %>%
  step_mutate(household_serving_fulltext_pretz = str_detect(household_serving_fulltext,
                                                              "pretzel|petzels")) %>%
  step_mutate(household_serving_fulltext_slice = str_detect(household_serving_fulltext,
                                                              "slice")) %>%
  step_mutate(household_serving_fulltext_square = str_detect(household_serving_fulltext,
                                                              "square")) %>%
  step_mutate(household_serving_fulltext_tbsp = str_detect(household_serving_fulltext,
                                                              "tbsp")) %>%
  step_mutate(across(is.logical ,  ~replace_na(.x , 0))) %>%
  step_mutate(across(is.numeric ,  ~replace_na(.x , 0))) %>%
  step_log(n_word_ing , len_ing , offset = 1) %>%
  step_rm(ingredients, brand , description , household_serving_fulltext) %>%
  step_mutate_at(has_type(match = "logical") , fn = as.numeric)
```

- After prepping the recipe on train_train, baking train_train and train_val, and creating 5 folds for the CV i define the models and tuned them:

Multinom regression

```
set.seed(77)
multi_mod <- multinom_reg( mode = "classification" , penalty =  tune() ,mixture =   tune() , engine = "g
multi_grid <- grid_regular(penalty() , mixture() , levels = 5)
multi_wf <- workflow() %>%
  add_model(multi_mod) %>%
  add_formula(category~ . )
multi_mod_tuning <- multi_wf %>%
  tune_grid(
```

```
    resamples = folds,
    grid = multi_grid
  )
```

Knn model

```
set.seed(77)
knn_mod <- nearest_neighbor( mode = "classification" , neighbors = tune() ,dist_power =  tune() , engin
knn_grid <- grid_regular(neighbors() , dist_power() , levels = c(10 , 2))
knn_wf <- workflow() %>%
  add_model(knn_mod) %>%
  add_formula(category~ . )
knn_mod_tuning <- knn_wf %>%
  tune_grid(
    resamples = folds,
    grid = knn_grid
  )
```

xgboost model

```
set.seed(77)
boost_mod <- boost_tree( mode = "classification" , learn_rate = tune() ,trees = tune()  ,
                         tree_depth = tune()  , engine = "xgboost")
boost_grid <- grid_regular(learn_rate() , trees(range = c(1,100)) ,
                           tree_depth() , levels = c(5,5,4))
boost_wf <- workflow() %>%
  add_model(boost_mod) %>%
  add_formula(category~ . )
boost_mod_tuning <- boost_wf %>%
  tune_grid(
    resamples = folds,
    grid = boost_grid
  )
```

random forest

```
set.seed(77)
rand_mod <- rand_forest( mode = "classification" ,trees = tune() , mtry = tune(),
                         engine = "randomForest")
rand_grid <- grid_regular(trees(range = c(500,1000)) , mtry(range = c(12,60)) ,
                          levels = c(3,5))
rand_wf <- workflow() %>%
  add_model(rand_mod) %>%
  add_formula(category~ . )
rand_mod_tuning <- rand_wf %>%
  tune_grid(
    resamples = folds,
```

```
    grid = rand_grid
  )
```

best hyper parameters results

```
## # A tibble: 5 x 8
##        penalty mixture .metric  .estimator  mean     n std_err .config
##          <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.0000000001     1   accuracy multiclass 0.892     5 0.00201 Preprocessor1_Mo~
## 2 0.0000000316     1   accuracy multiclass 0.892     5 0.00201 Preprocessor1_Mo~
## 3 0.00001          1   accuracy multiclass 0.892     5 0.00201 Preprocessor1_Mo~
## 4 0.0000000001     0.5 accuracy multiclass 0.892     5 0.00189 Preprocessor1_Mo~
## 5 0.0000000316     0.5 accuracy multiclass 0.892     5 0.00189 Preprocessor1_Mo~
```

```
## # A tibble: 5 x 8
##   neighbors dist_power .metric  .estimator  mean     n std_err .config
##       <int>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1        10          1 accuracy multiclass 0.917     5 0.00119 Preprocessor1_Mo~
## 2         9          1 accuracy multiclass 0.916     5 0.00142 Preprocessor1_Mo~
## 3         8          1 accuracy multiclass 0.916     5 0.00166 Preprocessor1_Mo~
## 4         7          1 accuracy multiclass 0.915     5 0.00171 Preprocessor1_Mo~
## 5         6          1 accuracy multiclass 0.914     5 0.00189 Preprocessor1_Mo~
```

```
## # A tibble: 5 x 9
##   trees tree_depth learn_rate .metric  .estimator  mean     n std_err .config
##   <int>      <int>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1   100         10        0.1 accuracy multiclass 0.925     5 0.00303 Preproces~
## 2   100         15        0.1 accuracy multiclass 0.924     5 0.00255 Preproces~
## 3    75         10        0.1 accuracy multiclass 0.924     5 0.00224 Preproces~
## 4    75         15        0.1 accuracy multiclass 0.924     5 0.00262 Preproces~
## 5    50         15        0.1 accuracy multiclass 0.923     5 0.00275 Preproces~
```

```
## # A tibble: 5 x 8
##    mtry trees .metric  .estimator  mean     n std_err .config
##   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1    12  1000 accuracy multiclass 0.924     5 0.00337 Preprocessor1_Model03
## 2    12   500 accuracy multiclass 0.924     5 0.00295 Preprocessor1_Model01
## 3    12   750 accuracy multiclass 0.924     5 0.00295 Preprocessor1_Model02
## 4    24   500 accuracy multiclass 0.923     5 0.00309 Preprocessor1_Model04
## 5    24  1000 accuracy multiclass 0.923     5 0.00287 Preprocessor1_Model06
```

- For multinom regression: penalty = 1.000000e-10 , mixture = 1
- For knn model: neighbors = 10 , dist_power = 1
- For xgboost model: trees = 100 , tree_depth = 10 , learn rate = 0.1
- For random forest: mtry = 12 , trees = 500.

Redefining the models with new best HP

```
final_multi_mod <- multinom_reg(mode = "classification" , penalty = 1.000000e-10    ,
                                mixture = 1 , engine = "glmnet")
final_knn_mod <- nearest_neighbor(mode = "classification" ,neighbors = 10 ,
                                  dist_power = 1 , engine = "kknn")
final_boost_mode <- boost_tree(mode = "classification" , trees = 100 ,
                               tree_depth = 10 , learn_rate = 0.1 ,
                               engine = "xgboost")
final_rand_mode <- rand_forest(mode = "classification" , trees = 500 ,
                               mtry = 12 , engine = "randomForest")
```

Fitting them on the entire train_train data

```
set.seed(77)
fit_multi <- fit(final_multi_mod , category~. , data = train_train)
fit_knn <- fit(final_knn_mod , category~. , data = train_train)
fit_boost <- fit(final_boost_mode , category~. , data = train_train)
fit_rand <- fit(final_rand_mode , category ~. , data = train_train)
```

Prediction on train_val data

```
multi_pred <- predict(fit_multi , new_data = train_val)$.pred_class
knn_pred <- predict(fit_knn , new_data = train_val )$.pred_class
boost_pred <- predict(fit_boost , new_data = train_val)$.pred_class
rand_pred <- predict(fit_rand , new_data = train_val)$.pred_class
```

Results

```
## # A tibble: 4 x 2
##   model               accuracy_of_model
##   <chr>                           <dbl>
## 1 xgboost                         0.926
## 2 random forest                   0.925
## 3 knn                             0.917
## 4 multinom regression             0.889
```

- As we can see xgboost and random forest both were better than the other two, but because i doesn't have a clear best model i will make final predictions
  with the two of them.
- After redefining the recipe for the entire training data, prepping on it and baking the training data and test data into final_train_pro and
  final_test_pro, i fit xgboost and random forest model on the final_train_pro.

```
fit_boost <- fit(final_boost_mode , category~. , data = final_train_pro)
fit_rand <- fit(final_rand_mode , category ~. , data = final_train_pro)
```

Final predictions

```
pred_cat_xg <- predict(fit_boost , new_data = final_test_pro)$.pred_class
cate_df_xg <- final_test_pro %>%
  select(idx)
cate_df_xg$pred_cat <- pred_cat_xg

write_csv(cate_df_xg , "model01.csv")
```

```
pred_cat_rf <- predict(fit_rand , new_data = final_test_pro)$.pred_class
cate_df_rf <- final_test_pro %>%
  select(idx)
cate_df_rf$pred_cat <- pred_cat_rf

write_csv(cate_df_rf , "model02.csv")
```

Notes on DL

- Unfortunately CNN models and DNN models on the image data and tabular data yielded worst results than the previous models, and thats including mixed inputs models with concatenation layers, and model for each data individually, and includes many many tries :(.