DSApps 2022 @ TAU: Final Project

Part 2 - Prediction

Sergei Dakov

Load packages required for tibble processing, recipes and model construction (code in notebook)

```
## For binary classification, the first factor level is assumed to be the event.
## Use the argument ‘event_level = "second"‘ to alter this as needed.

## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v dplyr   1.0.9
## v tibble  3.1.8     v stringr 1.4.0
## v tidyr   1.2.0     v forcats 0.5.1
## v readr   2.1.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x readr::spec()   masks yardstick::spec()
##
## Attaching package: 'recipes'
##
##
## The following object is masked from 'package:stringr':
##
##     fixed
##
##
## The following object is masked from 'package:stats':
##
##     step
##
##
##
## Attaching package: 'xgboost'
##
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

read in data

```
nutrients <-read.csv(file="data/nutrients.csv")
food_nutrients <- read.csv(file="data/food_nutrients.csv")
test <- read.csv(file="data/food_test.csv")
train <- read.csv(file="data/food_train.csv")
```

add the nutrient information into the data

```
food_nutrients_wider <- food_nutrients %>% pivot_wider(names_from = nutrient_id,values_from = amount)

train_big <- food_nutrients_wider %>% filter(idx %in% train$idx) %>% left_join(train,by = "idx")
test_big <- food_nutrients_wider %>% filter(idx %in% test$idx) %>% left_join(test,by = "idx")
```

add keywords from description (similarly to the example in the instructions) (code in notebook)

```
train_big <- train_big %>% mutate(has_popcorn = ifelse(str_detect(description, popcorn_peanuts_keywords)
  mutate(has_candy = ifelse(str_detect(description, candy_keywords ),1,0)) %>%
  mutate(has_cookies = ifelse(str_detect(description, coockies_keywords ),1,0)) %>%
  mutate(has_chips = ifelse(str_detect(description, chips_keywords ),1,0)) %>%
  mutate(has_chocolate = ifelse(str_detect(description, chcolate_keywords ),1,0)) %>%
  mutate(has_cake = ifelse(str_detect(description, cakes_keywords ),1,0))



test_big <- test_big %>% mutate(has_popcorn = ifelse(str_detect(description, popcorn_peanuts_keywords),
  mutate(has_candy = ifelse(str_detect(description, candy_keywords ),1,0)) %>%
  mutate(has_cookies = ifelse(str_detect(description, coockies_keywords ),1,0)) %>%
  mutate(has_chips = ifelse(str_detect(description, chips_keywords ),1,0)) %>%
  mutate(has_chocolate = ifelse(str_detect(description, chcolate_keywords ),1,0)) %>%
  mutate(has_cake = ifelse(str_detect(description, cakes_keywords ),1,0))
```

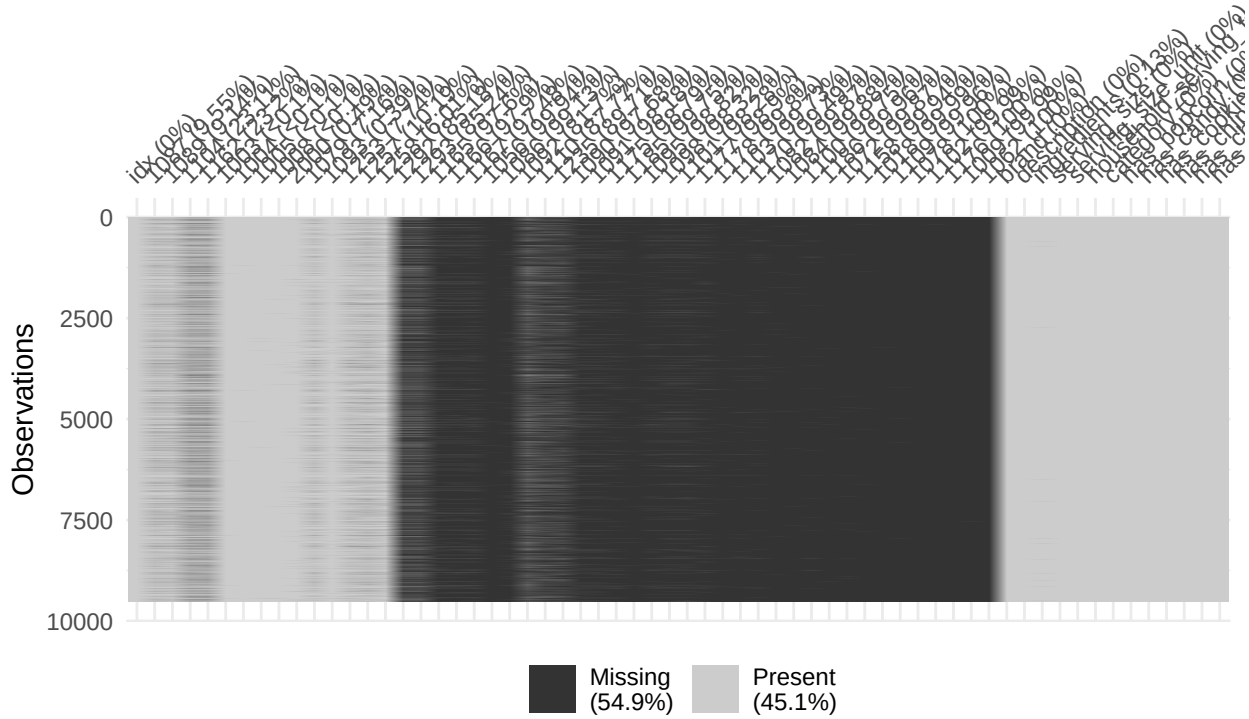
create train and test split for model tuning

```
set.seed(800)
first_split <- initial_split(train_big,strata = category)
train_train <- training(first_split)
train_test <- testing(first_split)
```

view missing variables in expanded data

```
vis_miss(train_big %>% sample_frac(0.3))
```

```
## Warning: `gather_()` was deprecated in tidyr 1.2.0.
## Please use `gather()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```



several of the nutrients appear very rarely in the overall data

define helper functions for multiple fitting

```
mod_tree <- decision_tree(mode="classification",cost_complexity = 0.01)

fit_tree <- function(rec) {
  fit(mod_tree,category~.,data = bake(rec,NULL,all_predictors(),all_outcomes()))
}

pred_tree_acc <- function(spl,rec,mod) {
  mod_baked <- bake(rec,new_data=assessment(spl),all_predictors(),all_outcomes())
  out <- mod_baked %>% select(category)
  predicted <- predict(mod,mod_baked)
  out <- out %>% cbind(predicted)
  names(out) <- c("truth","prediction")
  out
}


train_recipe_acc <- function(rec,rec_name,spl) {
  spl_prep <- map(spl,prepper,recipe=rec)
  spl_fit <- map(spl_prep,fit_tree)
  spl_pred <- pmap(lst(spl=spl,rec=spl_prep,mod=spl_fit),pred_tree_acc)
  out <- c()
  for (i in 1:length(spl_pred)) {
    current_split <- spl_pred[[i]]
```

```
    new_val <- mean(current_split$truth==current_split$prediction)
    out <- c(out,new_val)
  }
  out
}
```

create resamping splits

```
nrow(train_train)
```

```
## [1] 23812
```

```
# define sizes for resampling
split_sizes= c("imbalance"=5812,"missing"=4000, "text_features" =8000,"tuning"=6000)


set.seed(600)
missing_split <- initial_split(train_train,strata = category, prop = (split_sizes["missing"]+3)/nrow(tra
train_missing <- training(missing_split)
further_split <- testing(missing_split)


set.seed(300)
imbalance_split <- initial_split(further_split,strata = category, prop = (split_sizes["imbalance"]+3)/n
train_imbalance <- training(imbalance_split)
further_split <- testing(imbalance_split)

set.seed(200)
engineering_split <- initial_split(further_split,strata = category, prop = (split_sizes["text_features"]
train_engineering <- training(engineering_split)
train_tuning <- testing(engineering_split)
```

missing value decision: keep/drop rare columns, drop or impute: mean/median for missing values (code in notebook)

cross validation splits

```
set.seed(100)
cv_splits <- vfold_cv(train_missing,v=10,strata = category)
```

decision for missing values and rare columns

```
lst_recs <- list("drop_drop"=rec_drop_drop,
                 "drop_zero"=rec_drop_zero,
                 "drop_mean"=rec_drop_mean,
                 "drop_median"=rec_drop_med,
                 "keep_zero"=rec_keep_zero,
                 "keep_mean"=rec_keep_mean,
                 "keep_median"=rec_keep_med)


calculate_splits_acc <- function(x) {
```

```
  temp_split <- x %>% select(id)
  for(i in 1:length(lst_recs)) {
    y <- train_recipe_acc(lst_recs[[i]],names(lst_recs)[i],x$splits )
    nm <- names(lst_recs)[i]
    temp_split <- temp_split %>% mutate(!!nm := y)
  }
  temp_split
}
cv_splits <- calculate_splits_acc(cv_splits)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(missing_flag)‘ instead of ‘missing_flag‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1109‘.
## Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1186‘ and
## ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1109‘.
## Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1056‘,
## ‘1186‘ and ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before
## normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1109‘. Consider using ‘step_zv()‘ t
## Column(s) have zero variance so scaling cannot be used: ‘1109‘. Consider using ‘step_zv()‘ to remove
## Column(s) have zero variance so scaling cannot be used: ‘1109‘. Consider using ‘step_zv()‘ to remove

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1056‘ and
## ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1158‘ and
## ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1109‘.
## Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1186‘ and
## ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1109‘.
## Consider using ‘step_zv()‘ to remove those columns before normalizing

## Warning: Column(s) have zero variance so scaling cannot be used: ‘1056‘,
## ‘1186‘ and ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before
## normalizing
```

```
## Warning: Column(s) have zero variance so scaling cannot be used: ‘1109‘. Consider using ‘step_zv()‘ t
## Column(s) have zero variance so scaling cannot be used: ‘1109‘. Consider using ‘step_zv()‘ to remove
## Column(s) have zero variance so scaling cannot be used: ‘1109‘. Consider using ‘step_zv()‘ to remove
```

```
## Warning: Column(s) have zero variance so scaling cannot be used: ‘1056‘ and
## ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before normalizing
```

```
## Warning: Column(s) have zero variance so scaling cannot be used: ‘1158‘ and
## ‘1109‘. Consider using ‘step_zv()‘ to remove those columns before normalizing
```

```
cv_res_missing <- cv_splits %>% pivot_longer(cols=names(lst_recs),names_to = "recipe",values_to = "ACC"
  select(id,recipe,ACC) %>% separate(recipe,c("rare","missing"))%>%
  group_by(rare,missing) %>% summarise (ACC = mean(ACC)) %>% arrange(-ACC)
```

```
## ‘summarise()‘ has grouped output by 'rare'. You can override using the
## ‘.groups‘ argument.
```

```
cv_res_missing
```

```
## # A tibble: 7 x 3
## # Groups:   rare [2]
##   rare  missing   ACC
##   <chr> <chr>   <dbl>
## 1 drop  median  0.815
## 2 keep  median  0.815
## 3 drop  mean    0.815
## 4 keep  mean    0.815
## 5 drop  zero    0.815
## 6 keep  zero    0.815
## 7 drop  drop    0.810
```

the selected recipe is dropping the rare columns and imputing the median for missing values decision regarding group balance: upsample, downsample or do nothing (code in notebook)

crossvalidation splits

```
set.seed(100)
cv_splits <- vfold_cv(train_imbalance,v=10,strata = category)
```

decide best recipe

```
lst_recs <- list("upsample" = rec_upsample,
                 "downsample"= rec_downsample,
                 "nothing"=rec_nothing)
cv_splits <- calculate_splits_acc(cv_splits)
cv_res_balance <- cv_splits %>% pivot_longer(cols=names(lst_recs),names_to = "recipe",values_to = "ACC"
  select(id,recipe,ACC) %>%
  group_by(recipe) %>% summarise (ACC = mean(ACC)) %>% arrange(-ACC)
cv_res_balance
```

```
## # A tibble: 3 x 2
##   recipe       ACC
##   <chr>      <dbl>
## 1 upsample   0.829
## 2 downsample 0.825
## 3 nothing    0.825
```

selected option is upsample

apply text features

```
train_recycled <- train_engineering %>% rbind(train_imbalance) %>% rbind(train_missing)
train_recycled_words <- train_recycled %>% select(idx,category,ingredients) %>%
  mutate(ingredients = str_replace_all(ingredients,",|//*","")) %>% unnest_tokens(word,ingredients) %>%

top_words <- train_recycled %>% select(idx,ingredients) %>% mutate(ingredients = str_replace_all(ingredi
  filter(!word %in% stop_words$word,str_detect(word,"[a-z]")) %>% count(idx,word) %>% count(word) %>% fi
group_sizes <- train_recycled %>% group_by(category) %>% summarize(n = n()) %>% pull(n)

fisher_combinatoric <- function(x1,x2,x3,x4,x5,x6) {
  var_vec <- c(x1,x2,x3,x4,x5,x6)
  min_OR <- 1
  for (i in 1:6){
    j = i+1
    while (j<=6) {
      new_OR <- ((var_vec[i]+1)/(group_sizes[i]-var_vec[i]+1))/((var_vec[j]+1)/(group_sizes[j]-var_vec[j
      if ((new_OR<min_OR)|((1/new_OR)<min_OR)) {
        min_OR <- new_OR
        var1 <- i
        var2 <- j
      }
      j <- j+1
    }
  }
  final_table <- matrix(c(var_vec[var1],group_sizes[var1]-var_vec[var1],var_vec[var2],group_sizes[var2]-
  ft <- fisher.test(final_table)
  list(OR = ft$estimate, Pval = ft$p.value)

}

categories <- unique(train_engineering$category)
rec_fisher <- train_recycled_words %>% filter(word %in% top_words) %>% count(idx,word,category) %>% coun
  mutate(fisher = map2(cakes_cupcakes_snack_cakes,candy,chips_pretzels_snacks,chocolate,cookies_biscuits

ORs <- c()
Pvals <- c()
for (i in 1:nrow(rec_fisher)) {
  new_results <-fisher_combinatoric(rec_fisher[[i,2]],rec_fisher[[i,3]],rec_fisher[[i,4]],rec_fisher[[i
  ORs <- c(ORs,new_results$OR)
  Pvals <- c(Pvals,new_results$Pval)
}
rec_fisher <- rec_fisher %>% cbind(Pval = Pvals,OR = ORs)

rec_fisher <- rec_fisher %>% mutate(Padjust = p.adjust(Pval,method="BH"))
```

```
important_words <- rec_fisher %>% filter(Padjust < 10e-4) %>% pull(word)

count_words <- function(word_vec,data_vec) {
  counts <- map(word_vec,~str_count(data_vec,.x))
  names(counts)= word_vec
  counts
}

train_recycled_words <- train_recycled %>% bind_cols(count_words(important_words,train_recycled$ingredi
```

recipes for text features (code in notebook)

cross validation split

```
set.seed(100)
cv_splits <- vfold_cv(train_recycled_words,v=10,strata = category)

lst_recs <- list("word features" = rec_top_words,
                 "nothing"=rec_nothing)
cv_splits <- calculate_splits_acc(cv_splits)
cv_res_balance <- cv_splits %>% pivot_longer(cols=names(lst_recs),names_to = "recipe",values_to = "ACC"
  select(id,recipe,ACC) %>%
  group_by(recipe) %>% summarise (ACC = mean(ACC)) %>% arrange(-ACC)
cv_res_balance
```

```
## # A tibble: 2 x 2
##   recipe          ACC
##   <chr>         <dbl>
## 1 word features 0.831
## 2 nothing       0.829
```

chosen recipe is using the fisher based text features

tuning the tree parameters

```
test <- train_tuning %>% bind_cols(count_words(important_words,train_tuning$ingredients))
error_count <- rep(0,100)
sequence_of_costs <- seq(0,0.01,length.out=100)
for (i in 1:100){
  mod_tree <- decision_tree(mode="classification",cost_complexity = sequence_of_costs[i])

  train_recycled_words <- train_recycled_words %>% mutate("serving_pieces" = ifelse(str_detect(household
  rec_trial <- rec_top_words %>%
    prep()
  test2 <- bake(rec_trial,test)
  fit_trial <- fit(mod_tree,category~.,data=bake(rec_trial,new_data = NULL))
  pred_trial <-predict(fit_trial,test2)
  trial <- test2 %>% bind_cols(pred_trial)
  errors <- trial %>% filter(category!=.pred_class)
  error_count[i] <- nrow(errors)
  names(error_count)[i]=sequence_of_costs[i]
}
best_option <- which.min(error_count)
```

```
calculated_best <- names(best_option)
```

update the model to the best parameter

```
mod_tree <- decision_tree(mode="classification",cost_complexity = calculated_best)
```

check result on validation set

```
train_final <- train_train %>% bind_cols(count_words(important_words,train_train$ingredients))
train_test <- train_test %>% bind_cols(count_words(important_words,train_test$ingredients))
rec_final <- recipe(category~.,data=train_final) %>%
  step_rm(idx) %>%
  step_rm(c(description,ingredients,household_serving_fulltext)) %>%
  step_rm(missing_flag) %>%
  step_impute_median(all_numeric())%>%
  step_other(all_nominal(),-all_outcomes(),other='other') %>%
  step_normalize(all_numeric()) %>%
  prep()

train_final <- bake(rec_final,NULL)
train_test <- bake(rec_final,train_test)
fit_final <- fit(mod_tree,category~.,data = train_final)
predicted <- predict(fit_final,train_test)
train_test <- train_test %>% bind_cols(predicted)
accuracy(train_test,truth = "category",estimate = ".pred_class")
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.907
```

apply XGBoost on the selected model

```
xgboost_spec <- boost_tree(trees = 40) %>%
  set_mode("classification") %>%
  set_engine("xgboost")
#fit the model
set.seed(1)
xgboost_fit <- xgboost_spec %>% fit(category~.,data=train_final)
#predict the result
xg_pred <-predict(xgboost_fit,new_data =train_test)
trial_xg <- cbind(category = train_test$category,xg_pred)
trial_xg <- trial_xg %>% mutate(across(everything(),~factor(.x)))
#get accuracy for boosted model
print(accuracy(data=trial_xg,truth = category,estimate = .pred_class))
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.941
```

#final prediction model 1: tree

```
train_final <- train_big %>% bind_cols(count_words(important_words,train_big$ingredients))
test_final <- test_big %>% bind_cols(count_words(important_words,test_big$ingredients))

rec_final <- recipe(category~.,data=train_final) %>%
  step_rm(idx) %>%
  step_rm(c(description,ingredients,household_serving_fulltext)) %>%
  step_rm(missing_flag) %>%
  step_impute_median(all_numeric())%>%
  step_other(all_nominal(),-all_outcomes(),other='other') %>%
  step_normalize(all_numeric()) %>%
  prep()

train_final <- bake(rec_final,NULL)
test_final <- bake(rec_final,test_final)
fit_final <- fit(mod_tree,category~.,data = train_final)
predicted <- predict(fit_final,test_final)
test_final_1 <- test_big %>% bind_cols(predicted)
test_final_1 %>% select(idx,.pred_class) %>% rename(pred_cat=.pred_class) %>% write_csv("model101.csv")
```

#final prediction model 2: XGBoost

```
set.seed(1)
xgboost_fit <- xgboost_spec %>% fit(category~.,data=train_final)

xg_pred <-predict(xgboost_fit,new_data =test_final)
test_final_2 <- test_big %>% bind_cols(xg_pred)
test_final_2 %>% select(idx,.pred_class) %>% rename(pred_cat=.pred_class) %>% write_csv("model102.csv")
```