# Run MOMAM - short ver.

Michael Wiley

2023-12-03

### The short version removes the data extraction and database building steps (~10 mins)

### as well as greatly reducing the number of models trained (~1 hour saved)

### Step 0: Setup

Load all helper functions located in `lib`.

```
## [[1]]
## [[1]]$value
## function ()
## {
##     POST("https://id.twitch.tv/oauth2/token", query = list(client_id = get_client_id(),
##         client_secret = get_client_secret(), grant_type = "client_credentials")) %>%
##         use_series("content") %>% rawToChar() %>% fromJSON() %>%
##         use_series("access_token") %>% return()
## }
##
## [[1]]$visible
## [1] FALSE
##
##
## [[2]]
## [[2]]$value
## function (db)
## {
##     games_all <- dbGetQuery(db, "SELECT DISTINCT game_id FROM game_data")
##     games_count <- nrow(games_all)
##     QUERY_LIMIT <- 500L
##     split_num <- 1L
##     if (games_count > QUERY_LIMIT) {
##         max_splits <- ceiling(games_count/QUERY_LIMIT)
##         while (split_num < max_splits) {
##             games_current <- rep(NA, QUERY_LIMIT)
##             games_current <- games_all$game_id[seq((split_num -
##                 1) * QUERY_LIMIT + 1, split_num * QUERY_LIMIT)]
##             get_query(games_current) %>% post_query() %>% process_query() %>%
##                 save_results(db, .)
##             split_num <- split_num + 1
##         }
##     }
##     games_current <- ifelse(games_count%%QUERY_LIMIT == 0, rep(NA,
##         QUERY_LIMIT), rep(NA, games_count%%QUERY_LIMIT))
```

```
##     games_current <- games_all$game_id[seq((split_num - 1) *
##         QUERY_LIMIT + 1, games_count)]
##     get_query(games_current) %>% post_query() %>% process_query() %>%
##         save_results(db, .)
## }
##
## [[2]]$visible
## [1] FALSE
##
##
## [[3]]
## [[3]]$value
## function (db)
## {
##     past_results <- read_csv(str_c(here("data/momam7"), "/momam7_base_data.csv"))
##     games_list <- dbGetQuery(db, "SELECT DISTINCT game\n          FROM game_data;")
##     to_add_names <- past_results$game[which(!past_results$game %in%
##         unlist(games_list))]
##     to_add_urls <- c("https://www.igdb.com/games/super-mario-land-2-6-golden-coins",
##         "https://www.igdb.com/games/mega-man-3", "https://www.igdb.com/games/mega-man-7",
##         "https://www.igdb.com/games/pokemon-card-gb2-great-rocket-dan-sanjou",
##         "https://www.igdb.com/games/pokemon-puzzle-league", "https://www.igdb.com/games/wave-race-64
##         "https://www.igdb.com/games/f-zero-x", "https://www.igdb.com/games/contra-iii-the-alien-wars
##         "https://www.igdb.com/games/the-simpsons-hit-run", "https://www.igdb.com/games/kirby-and-the
##         "https://www.igdb.com/games/ninja-gaiden-ii-the-dark-sword-of-chaos",
##         "https://www.igdb.com/games/sonic-and-the-secret-rings",
##         "https://www.igdb.com/games/sonic-forces", "https://www.igdb.com/games/monopoly-for-nintendo
##         "https://www.igdb.com/games/spongebob-squarepants-the-cosmic-shake",
##         "https://www.igdb.com/games/tony-hawks-pro-skater-3--1",
##         "https://www.igdb.com/games/bubsy-in-claws-encounters-of-the-furred-kind")
##     to_add_ids <- sapply(to_add_urls, search_game_id)
##     to_add_names <- "Bubsy in Claws Encounters of the Furred Kind"
##     to_add_urls <- "https://www.igdb.com/games/bubsy-in-claws-encounters-of-the-furred-kind"
##     to_add_ids <- sapply(to_add_urls, search_game_id)
##     to_add_names <- "The Lord of the Rings: The Two Towers"
##     to_add_urls <- "https://www.igdb.com/games/the-lord-of-the-rings-the-two-towers"
##     to_add_ids <- sapply(to_add_urls, search_game_id)
##     to_add_names <- "Monopoly Plus"
##     to_add_urls <- "https://www.igdb.com/games/monopoly-plus"
##     to_add_ids <- sapply(to_add_urls, search_game_id)
##     insert <- dbSendStatement(db, "INSERT INTO game_data (game_id, game, url) VALUES (?, ?, ?) ON CO
##     dbBind(insert, list(to_add_ids, to_add_names, to_add_urls))
##     dbClearResult(insert)
## }
##
## [[3]]$visible
## [1] FALSE
##
##
## [[4]]
## NULL
##
## [[5]]
## [[5]]$value
```

```
## function (dbname, csv)
## {
##     message(paste("\nNow reading:", csv))
##     v_streamer <- tolower(str_extract(csv, "(pie|Spike)"))
##     v_year <- str_extract(csv, "20[:number:]+") %>% as.numeric()
##     yearly_data <- read_csv(csv) %>% select(2, 3) %>% `colnames<-`(c("game",
##         "minutes")) %>% mutate(streamer = v_streamer, year = v_year,
##         game = str_extract(game, "[^|]+"), game = str_replace(game,
##             "é", "e"), game = sapply(game, pokemon_checker),
##         url = str_to_lower(game), url = str_replace_all(url,
##             "&", "and"), url = str_replace_all(url, "\\+", "plus"),
##         url = str_remove_all(url, "[^([:alnum:]-?|[:space:]|'|$)]"),
##         url = str_replace_all(url, "([:space:]+|')", "-"), url = str_replace(url,
##             "chip-s", "chips"), url = str_replace(url, "freddy-s",
##             "freddys"), url = str_replace(url, "igi-s-man", "igis-man"),
##         url = str_replace(url, "known-s-bat", "knowns-bat"),
##         url = str_replace(url, "let-s-go-pika", "lets-go-pika"),
##         url = str_replace(url, "pooh-s-home", "poohs-home"),
##         url = str_replace(url, "ter-s-aren", "ters-aren"), url = str_replace(url,
##             "shi-s-wool", "shis-wool"), url = str_replace(url,
##             "super-mario-3d-world-plus-bowser-s-fury", "super-mario-3d-world-plus-bowsers-fury"),
##         url = str_replace(url, "shi-s-saf", "shis-saf"), url = str_replace(url,
##             "mafiagg", "mafia-dot-gg"), url = str_replace(url,
##             "o-hare", "ohare"), url = str_replace(url, "sid-meier-s-civilization-vi",
##             "sid-meiers-civilization-vi"), url = str_replace(url,
##             "4-it-s-abo", "4-its-abo"), url = str_replace(url,
##             "evil-director-s-cut", "evil-directors-cut"), url = str_replace(url,
##             "link-s-awakening", "links-awakening"), url = str_replace(url,
##             "links-awakening-dx", "link-s-awakening-dx"), url = str_replace(url,
##             "duelists-of-the-roses", "duelists-of-the-roses-fa82a70b-8784-4e43-babc-9ba06b3ba75d"),
##         url = str_replace(url, "--", "-"), url = str_replace(url,
##             "ing-harmony", "ing-harmony--1"), url = str_replace(url,
##             "fate-of-atlantis", "fate-of-atlantis--1"), url = str_replace(url,
##             "mike-tyson-s-punch-out", "punch-out--2"), url = str_replace(url,
##             "hd-15", "hd-1-dot-5"), url = str_replace(url, "speedy-gonzales-in-los-gatos-banditos",
##             "speedy-gonzales-los-gatos-bandidos"), url = str_replace(url,
##             "mega-man-zerozx-legacy-collection", "mega-man-zero-slash-zx-legacy-collection"),
##         url = str_replace(url, "dai-gyakuten-saiban-naruhodou-ryuunosuke-no-bouken",
##             "dai-gyakuten-saiban-naruhodou-ryuunosuke-no-bouken-1-and-2-best-price"),
##         url = str_replace(url, "mario-s-time-machine", "marios-time-machine"),
##         url = str_replace(url, "sesame-street-elmo-s-number-journey",
##             "sesame-street-elmos-number-journey"), url = str_replace(url,
##             "sesame-street-elmo-s-letter-adventure", "sesame-street-elmos-letter-adventure--1"),
##         url = str_replace(url, "25-remix", "2-dot-5-remix"),
##         url = str_replace(url, "sonic-adventure-dx-director-s-cut",
##             "sonic-adventure-dx-directors-cut"), url = str_replace(url,
##             "shantae-and-the-pirate-s-curse", "shantae-and-the-pirates-curse"),
##         url = str_replace(url, "^(disney-s-)?goof-troop$", "disneys-goof-troop"),
##         url = str_replace(url, "marvel-s-spider-man", "marvels-spider-man"),
##         url = str_replace(url, "spyro-2-ripto-s-rage--reignited",
##             "spyro-2-riptos-rage-reignited"), url = str_replace(url,
##             "clan-o-conall-and-the-crown-of-the-stag", "clan-oconall-and-the-crown-of-the-stag"),
##         url = str_replace(url, "ratchet-and-clank-up-your-arsenal",
##             "ratchet-clank-up-your-arsenal"), url = str_replace(url,
```

```
##             "the-legend-of-zelda-ocarina-of-time--master-quest",
##             "the-legend-of-zelda-ocarina-of-time-master-quest"),
##         url = str_replace(url, "mario-and-luigi", "mario-luigi"),
##         url = str_replace(url, "mario-luigi-superstar-saga-plus",
##             "mario-and-luigi-superstar-saga-plus"), url = str_replace(url,
##             "mario-luigi-paper-jam", "mario-and-luigi-paper-jam"),
##         url = str_replace(url, "are-you-afraid-of-the-dark\\?-the-tale-of-orpheo-s-curse",
##             "are-you-afraid-of-the-dark-the-tale-of-orpheo-s-curse"),
##         url = str_replace(url, "phoenix-wright-ace-attorney--dual-destinies",
##             "phoenix-wright-ace-attorney-dual-destinies"), url = str_replace(url,
##             "kingdom-hearts-hd-28-final-chapter-prologue", "kingdom-hearts-hd-2-dot-8-final-chapter-p
##         url = str_replace(url, "phoenix-wright-ace-attorney--spirit-of-justice",
##             "phoenix-wright-ace-attorney-spirit-of-justice"),
##         url = str_replace(url, "^[:graph:]{1}kami$", "okami"),
##         url = str_replace(url, "star-wars-jedi-knight--jedi-academy",
##             "star-wars-jedi-knight-jedi-academy"), url = str_replace(url,
##             "man-and-bass", "man-bass"), url = str_c("https://www.igdb.com/games/",
##             url), game_id = sapply(url, search_game_id)) %>%
##         dplyr::filter(game_id >= 0) %>% select(streamer, year,
##         game_id, game, minutes, url)
##     insert <- dbSendStatement(db, "INSERT INTO game_data (game_id, game, url) VALUES (?, ?, ?) ON CO
##     dbBind(insert, list(as.vector(yearly_data$game_id), as.vector(yearly_data$game),
##         as.vector(yearly_data$url)))
##     dbClearResult(insert)
##     insert <- dbSendStatement(db, "INSERT INTO yearly_playtime (streamer, year, game_id, minutes) VA
##     dbBind(insert, list(as.vector(yearly_data$streamer), as.vector(yearly_data$year),
##         as.vector(yearly_data$game_id), as.vector(yearly_data$minutes)))
##     dbClearResult(insert)
## }
##
## [[5]]$visible
## [1] FALSE
```

Various global environment variable setup:

Establish database connection Load all csv files Creates tables name Loads API information

```r
#db <- dbConnect(SQLite(), str_c(here("output"), "/", "MOMAM.sqlite"))
db <- connect_db(here("output"), "MOMAM.sqlite")
csvs <- load_all_csvs_from_folder(here("data/channel_data/"))

tablenames <- str_c(tolower(str_extract(csvs, "(pie|Spike)")),
                    str_extract(csvs, "20[:number:]+"))


# End points with necessary information from the API.
# TO DO: Organize this into a single data structure.
api_url <- "https://api.igdb.com/v4"
end_games <- "/games"
end_categories <- "/categories"
end_genres <- "/genres"
end_age <- "/age_ratings"
end_company <- "/involved_companies"
end_engines <- "/game_engines"
end_platforms <- "/platforms"
```

```r
end_franchises <- "/franchises"
end_themes <- "/themes"
end_keywords <- "/keywords"
end_perspectives <- "/player_perspectives"

client_id <- get_client_id()
access_token <- get_access_token()
```

## Step 2: Prepare Data for Analysis

The above step steps collected the necessary data and placed it into normalized SQL tables

Now it's time to move that data into a analysis-ready format.

This cell here will connect to the database containing our game metadata, as well the csv file holding the results of all previous MOMAMs.

game_list is our master list of games that the streamers have played cumulative_playtime is our list of how long each streamer spent playing each game in total imputed_yearly_metadata lists pertinent metadata for each game, such as genre and release data.

```r
#db <- connect_db(here("output"), "MOMAM.sqlite")
past_results <- read_csv(str_c(here("data/momam7"), "/momam7_base_data.csv"))
```

```
## Rows: 273 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): game, category, winner
## dbl (2): MOMAM, grouping
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
game_list <- dbGetQuery(db, 'SELECT game_id, game FROM game_data;') %>%
  as_tibble()

cumulative_playtime <- dbGetQuery(db, 'SELECT * FROM cumulative_playtime;') %>%
  as_tibble()

imputed_yearly_metadata <- dbGetQuery(db, 'SELECT * FROM imputed_yearly_metadata') %>%
  as_tibble()
```

The following cells will all focus around taking our db normalized data and preparing it for analysis. The first thing to do is create dummy variables columns for our categorical variables, but not our numeric ones.

```r
past_results_with_id <- past_results %>%
  left_join(game_list, by = join_by(game)) %>%
  left_join(imputed_yearly_metadata, by = join_by(game_id == id), relationship = "many-to-many")

past_results_categorical <- past_results_with_id %>%
  dplyr::filter(!(name %in% c("first_release_date", "total_rating"))) %>%
  distinct() %>%
  pivot_wider(names_from = c(name, value), values_from = value,
              names_glue = "{name}_{value}") %>%
  mutate(across(-c(MOMAM, grouping, game, category, winner, game_id),
                ~ ifelse(is.na(.), 0, 1)))
```

```r
past_results_numeric <- past_results_with_id %>%
  dplyr::filter((name %in% c("first_release_date", "total_rating"))) %>%
  distinct() %>%
  pivot_wider(names_from = name, values_from = value, values_fn = mean)

# For reordering columns with the pivoted columns in the back.
desired_order <- c("winner", "MOMAM", "grouping", "game", "category", "game_id")

# Combines the categorical and numeric dfs.
past_results_pivoted_metadata <- past_results_categorical %>%
  left_join(past_results_numeric, by = join_by(MOMAM, grouping, game, category, game_id, winner)) %>%
  dplyr::select(sort(names(.))) %>%
  dplyr::select(all_of(desired_order), everything()) %>%
  dplyr::select(-category) # category is handled by the cumulative time function below.
```

Next is to deal with multi-game challenges. Sometimes one day will consist of a race of 2 or more games and we need to treat these days one prediction. To achieve this, each game that's played on the same day as the data in all of their columns averaged out. The comments in the code should have more detail.

```r
# Counts how many games are in each group.
# A group is a set of games that either are being played or can be played on the same day.
# For example, if there's a bid war between Ocarina of Time and Majora's Mask,
# they would share the same grouping for that entry.
counts <- past_results %>%
  group_by(grouping) %>%
  count(grouping)

# Splits games with multiple categories and assigns weights to each entry depending
# on their grouping. All of the joins should go here.
past_results_pivoted <- past_results %>%
  left_join(counts, by = join_by(grouping)) %>%
  mutate(weight = 1/n) %>%
  dplyr::select(-n) %>%
  left_join(game_list, by = join_by(game)) %>%
  left_join(cumulative_playtime, by = join_by(game_id), multiple = "all", relationship = "many-to-many"
  dummy_cols(select_columns = c("category"),
             remove_selected_columns = T,
             split = ",") %>%
  pivot_wider(names_from = streamer, values_from = minutes, values_fill = 0,
              names_glue = "cum_playtime_{streamer}") %>%
  left_join(past_results_pivoted_metadata, by = join_by(MOMAM, grouping, game, winner, game_id))


# Sets the data up to be weighted. Will lose game id with this, so need to have all the data
# ready to go by this point.
past_results_weighted <- past_results_pivoted %>%
  pivot_longer(cols = c(starts_with("category"), starts_with("cum"),
                        starts_with("first"), starts_with("franchise"),
                        starts_with("genre"), starts_with("involved"),
                        starts_with("platform"), starts_with("player"),
                        starts_with("theme"), starts_with("total")), # Brings all of the columns togeth
               names_to = "placeholder") %>%                        # that need to be weighted.
  mutate(weighted_val = value * weight) %>% # Applies the weight
  group_by(MOMAM, grouping, winner, placeholder) %>%
```

```
  summarise(weighted_group = sum(weighted_val)) %>%
  pivot_wider(names_from = placeholder, values_from = weighted_group, values_fill = 0) # Puts the data
```

```
## `summarise()` has grouped output by 'MOMAM', 'grouping', 'winner'. You can
## override using the `.groups` argument.
```

```
head(past_results_weighted)
```

```
## # A tibble: 6 x 703
## # Groups:   MOMAM, grouping, winner [6]
##   MOMAM grouping winner `category_100%` `category_Any%` `category_CPU Battle`
##   <dbl>    <dbl> <chr>            <dbl>           <dbl>                 <dbl>
## 1     1        1 spike                0               1                     0
## 2     1        2 pie                  0               1                     0
## 3     1        3 pie                  0               1                     0
## 4     1        4 pie                  0               1                     0
## 5     1        5 spike                0               1                     0
## 6     1        6 spike                0               1                     0
## # i 697 more variables: category_Challenge <dbl>, category_Community <dbl>,
## #   category_Hard <dbl>, `category_Low%` <dbl>, category_Rando <dbl>,
## #   `category_Rom hack` <dbl>, cum_playtime_NA <dbl>, cum_playtime_pie <dbl>,
## #   cum_playtime_spike <dbl>, first_release_date <dbl>, franchises_1 <dbl>,
## #   franchises_100 <dbl>, franchises_1068 <dbl>, franchises_1071 <dbl>,
## #   franchises_11 <dbl>, franchises_1228 <dbl>, franchises_123 <dbl>,
## #   franchises_1351 <dbl>, franchises_145 <dbl>, franchises_147 <dbl>, ...
```

Note that this is problematic here - there are significantly more columns than rows at the moment.

```
dim(past_results_weighted)
```

```
## [1] 230 703
```

### Step 3: Analysis

Now that the data has been created, it's time to analyze it.

First up, a preprocessing workflow is created. Since we have so many columns - many of which have near-zero variance, we will filter those out.

```
past_results_weighted$winner <- as.factor(past_results_weighted$winner)

set.seed(323)

# Ideally the size of the test set will be the size of the last MOMAM.
target_split <- 1 -
  nrow(past_results_weighted[past_results_weighted$MOMAM == 7, ]) /
  nrow(past_results_weighted)

initial_split <- initial_split(past_results_weighted,
              prop = target_split,
              strata = MOMAM)

momam_train <- training(initial_split)
momam_test <- testing(initial_split)

momam_folds <- vfold_cv(momam_train)
```

```r
base_recipe <- recipe(winner ~ ., data = momam_train) %>%
  step_rm(grouping) %>%
  step_mutate(cum_playtime_pie = log(cum_playtime_pie + 1)) %>%
  step_mutate(cum_playtime_spike = log(cum_playtime_spike + 1)) %>%
  step_naomit(c(cum_playtime_pie, cum_playtime_spike)) %>%
  step_nzv(all_predictors(), freq_cut = 0.01, unique_cut = 0) %>%
  step_zv(all_predictors()) %>%
  step_lincomb(all_numeric_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_normalize(all_predictors())

# An opinionated way of reducing the dimensionality. This will create different
# PCs for each group of variables.
# However, different thresholds are used for each of categories because domain
# knowledge tells me that some of these items are significantly more important
# than others.
# For example, enough PCs are taken from genre columns in order to explain
# at least 90% of the variance because genre is one of the most important factors
# for who will end up winning.
grouped_pca <- recipe(winner ~ ., data = momam_train) %>%
  step_rm(grouping) %>%
  step_mutate(cum_playtime_pie = log(cum_playtime_pie + 1)) %>%
  step_mutate(cum_playtime_spike = log(cum_playtime_spike + 1)) %>%
  step_naomit(c(cum_playtime_pie, cum_playtime_spike)) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_pca(starts_with("category"), threshold = 0.5, prefix = "category_") %>%
  step_pca(starts_with("cum"), threshold = 0.8, prefix = "cum") %>%
  step_pca(starts_with("franchises"), threshold = 0.2, prefix = "franchises") %>%
  step_pca(starts_with("genres"), threshold = 0.9, prefix = "genres") %>%
  step_pca(starts_with("platforms"), threshold = 0.80, prefix = "platforms") %>%
  step_pca(starts_with("player"), threshold = 0.8, prefix = "player") %>%
  step_pca(starts_with("themes"), threshold = 0.8, prefix = "themes") %>%
  step_pca(starts_with("involved"), threshold = 0.2, prefix = "involved") %>%
  step_corr(all_predictors(), threshold = 0.9)

# Preforms an additional PCA on the grouped PCA since the dimensionality produced
# by that approach is still rather high.
# this processor ends up being one of the most effective after training.
hierarchical_pca <- grouped_pca %>%
  step_pca(-c("MOMAM", "first_release_date", "total_rating", "winner"),
           threshold = 0.8, prefix = "hier_")

pca_strict_recipe <- base_recipe %>%
  step_pca(all_predictors(), threshold = 0.8)
```

Define models:

These are the models is trained on. Currently a few simple models are here as a baseline, but ideally more will be used.

```r
model_dt <- decision_tree(tree_depth = tune(), min_n = tune(), cost_complexity = tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  translate()
```

```r
model_rf <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("randomForest") %>%
  set_mode("classification") %>%
  translate()

model_xgb <- boost_tree(tree_depth = tune(),
                        trees = tune(),
                        learn_rate = tune(),
                        mtry = tune(),
                        min_n = tune(),
                        loss_reduction = tune(),
                        sample_size = tune(),
                        stop_iter = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification") %>%
  translate()
```

List of models to train. Each model will be trained with each preprocessing technique.

```r
momam_workflowset <- workflow_set(
  preproc = list(base = base_recipe,
                 strict = pca_strict_recipe,
                 hierarchical_pca = hierarchical_pca),
  models = list(
    decision_tree = model_dt,
    rand_forest = model_rf,
    xgboost = model_xgb
  ),
  cross = TRUE
)
```

Trains and tunes each model with each pre-processor.

```r
race_ctrl <-
  control_race(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )

library(doParallel)
doParallel::registerDoParallel(cores = 10)

grid_results <-
  momam_workflowset %>%
  workflow_map(
    "tune_race_anova",
    seed = 0323,
    resamples = momam_folds,
    grid = 15,
    control = race_ctrl,
    verbose = TRUE
  )
```

```
## i 1 of 9 tuning:     base_decision_tree
```

```
## v 1 of 9 tuning:     base_decision_tree (56.6s)
## i 2 of 9 tuning:     base_rand_forest
## i Creating pre-processing data to finalize unknown parameter: mtry
## v 2 of 9 tuning:     base_rand_forest (54.6s)
## i 3 of 9 tuning:     base_xgboost
## i Creating pre-processing data to finalize unknown parameter: mtry
## v 3 of 9 tuning:     base_xgboost (30.6s)
## i 4 of 9 tuning:     strict_decision_tree
## v 4 of 9 tuning:     strict_decision_tree (26.5s)
## i 5 of 9 tuning:     strict_rand_forest
## i Creating pre-processing data to finalize unknown parameter: mtry
## v 5 of 9 tuning:     strict_rand_forest (48.3s)
## i 6 of 9 tuning:     strict_xgboost
## i Creating pre-processing data to finalize unknown parameter: mtry
## v 6 of 9 tuning:     strict_xgboost (36.2s)
## i 7 of 9 tuning:     hierarchical_pca_decision_tree
## v 7 of 9 tuning:     hierarchical_pca_decision_tree (32.9s)
## i 8 of 9 tuning:     hierarchical_pca_rand_forest
## i Creating pre-processing data to finalize unknown parameter: mtry
## v 8 of 9 tuning:     hierarchical_pca_rand_forest (45.1s)
## i 9 of 9 tuning:     hierarchical_pca_xgboost
## i Creating pre-processing data to finalize unknown parameter: mtry
## v 9 of 9 tuning:     hierarchical_pca_xgboost (44s)
```

```r
doParallel::stopImplicitCluster()
```

Displays how the models performed. Accuracy was chosen as the metric since we care more about predicting the winner correctly by any means necessary moreso than which predictions the model is getting right or wrong.

```r
autoplot(
  grid_results,
  rank_metric = "accuracy",  # <- how to order models
  #metric = "accuracy",       # <- which metric to visualize
  select_best = TRUE     # <- one point per workflow
)
```

Run_MOMAM_abridged_files/figure-latex/evaluate-1.pdf

Ranks each of the trained models.

```r
grid_results %>%
  workflowsets::rank_results(rank_metric = "accuracy") %>%
  pivot_wider(names_from = .metric, values_from = mean) %>%
  group_by(.config) %>%
  fill(accuracy, roc_auc, .direction = "downup") %>%
  distinct(rank, .keep_all = TRUE) %>%
  mutate(preprocessor = str_extract(wflow_id, ".*?(?=_)")) %>%
  select(wflow_id, model, preprocessor, accuracy, roc_auc, rank) %>%
  arrange(desc(roc_auc))
```

```
## Adding missing grouping variables: `.config`
```

```
## # A tibble: 71 x 7
## # Groups:   .config [15]
##    .config              wflow_id      model preprocessor accuracy roc_auc  rank
##    <chr>                <chr>         <chr> <chr>           <dbl>   <dbl> <int>
##  1 Preprocessor1_Model10 hierarchical~ rand~ hierarchical    0.577   0.653    45
##  2 Preprocessor1_Model05 base_rand_fo~ rand~ base            0.599   0.651    15
##  3 Preprocessor1_Model05 hierarchical~ deci~ hierarchical    0.587   0.651    39
##  4 Preprocessor1_Model13 hierarchical~ rand~ hierarchical    0.556   0.651    53
##  5 Preprocessor1_Model07 base_rand_fo~ rand~ base            0.599   0.651    16
##  6 Preprocessor1_Model07 hierarchical~ rand~ hierarchical    0.588   0.651    36
##  7 Preprocessor1_Model03 base_rand_fo~ rand~ base            0.609   0.647     7
##  8 Preprocessor1_Model03 hierarchical~ deci~ hierarchical    0.609   0.647    11
##  9 Preprocessor1_Model06 hierarchical~ rand~ hierarchical    0.582   0.646    43
## 10 Preprocessor1_Model04 base_rand_fo~ rand~ base            0.615   0.643     5
## # i 61 more rows
```

View the tuning parameters of the best model.

```r
best_results <-
  grid_results %>%
  extract_workflow_set_result("hierarchical_pca_rand_forest") %>%
  select_best(metric = "accuracy")

best_results
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     4  1921    34 Preprocessor1_Model09
```

Extract the best model and see how it performs on the test set.

```r
best_test_results <-
  grid_results %>%
  extract_workflow("hierarchical_pca_rand_forest") %>%
  finalize_workflow(best_results) %>%
  last_fit(split = initial_split)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2
```

```r
best_test_results$.metrics
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
```

```
## 1 accuracy binary         0.537 Preprocessor1_Model1
## 2 roc_auc  binary         0.617 Preprocessor1_Model1
```

View individual predictions of the best model on the test set.

```
predictions <- best_test_results %>%
  collect_predictions()

predictions
```

```
## # A tibble: 41 x 7
##    id               .pred_pie .pred_spike  .row .pred_class winner .config
##    <chr>                <dbl>       <dbl> <int> <fct>       <fct>  <chr>
##  1 train/test split    0.590       0.410      5 pie         spike  Preprocessor~
##  2 train/test split    0.796       0.204      6 pie         spike  Preprocessor~
##  3 train/test split    0.488       0.512     10 spike       pie    Preprocessor~
##  4 train/test split    0.0755      0.925     11 spike       spike  Preprocessor~
##  5 train/test split    0.412       0.588     17 spike       spike  Preprocessor~
##  6 train/test split    0.367       0.633     18 spike       spike  Preprocessor~
##  7 train/test split    0.370       0.630     20 spike       spike  Preprocessor~
##  8 train/test split    0.919       0.0812    37 pie         pie    Preprocessor~
##  9 train/test split    0.603       0.397     39 pie         spike  Preprocessor~
## 10 train/test split    0.501       0.499     56 pie         spike  Preprocessor~
## # i 31 more rows
```

```
predictions %>%
  mutate(correct = ifelse(.pred_class == winner, 1, 0)) %>%
  mutate(weighted_answer = case_when(winner == "pie" ~ .pred_pie,
                                     T ~ .pred_spike)) %>%
  summarise(test_acc = sum(correct)/nrow(.), test_roc = sum(weighted_answer)/nrow(.))
```

```
## # A tibble: 1 x 2
##   test_acc test_roc
##      <dbl>    <dbl>
## 1    0.537    0.523
```