Using `tidymodels`

Let's first split our data into training and testing datasets:

```
set.seed(1)
split <- initial_split(data = gss_subset, prop = 3/4)
gss_train <- training(split)
gss_test <- testing(split)
```

Next, let's use 5-fold cross validation:

```
folds <- rsample::vfold_cv(gss_train, v = 5)
```

Now, let's make our `recipe()` and `workflow()` that will be used for each of our models:

```
# Create the recipe
gss_recipe <- recipe(partyid ~ ., data = gss_train) %>%
  step_rm(year) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors())

# Create the workflow
gss_workflow <- workflow() %>%
  add_recipe(gss_recipe)

# View the workflow
gss_workflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: None
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_rm()
## * step_dummy()
## * step_zv()
```

Now, we can begin to specify our models for our model stack:

```
# Basic Logistic regression specification
basic_logreg_spec <- logistic_reg() %>%
  set_engine("glm")

# Add logistic regression to workflow
basic_logreg_workflow <- gss_workflow %>%
  add_model(basic_logreg_spec)

# Cross validation
set.seed(13)
basic_logreg_resamples <- fit_resamples(
```

```r
  basic_logreg_workflow,
  resamples = folds,
  control = control_stack_resamples()
)
```

```
##
## Attaching package: 'rlang'

## The following objects are masked from 'package:purrr':
##
##     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
##     splice

##
## Attaching package: 'vctrs'

## The following object is masked from 'package:dplyr':
##
##     data_frame

## The following object is masked from 'package:tibble':
##
##     data_frame

## The workflow being saved contains a recipe, which is 0.4 Mb in memory. If this was not intentional, p
```

```r
# Penalized Logistic regression specification
logreg_spec <- logistic_reg(penalty = tune(),
                            mixture = tune()) %>%
  set_engine("glmnet")

# add grid
lr_reg_grid <- tidyr::crossing(
  penalty = 10 ^ seq(-6, -1, length.out = 20),
  mixture = c(0.05, 0.2, 0.4, 0.6, 0.8, 1)
)

# Add logistic regression to workflow
logreg_workflow <- gss_workflow %>%
  add_model(logreg_spec)

# Tuning hyperparameters
set.seed(13)
logreg_resamples <- tune_grid(
  logreg_workflow,
  resamples = folds,
  grid = lr_reg_grid,
  control = control_stack_grid()
)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.0-2

## The workflow being saved contains a recipe, which is 0.4 Mb in memory. If this was not intentional, p
```

```r
# LDA specification
lda_spec <- discrim_linear(penalty = tune()) %>%
  set_engine("mda")

# Add LDA to workflow
lda_workflow <- gss_workflow %>%
  add_model(lda_spec)

# Fit with our cross validation
set.seed(13)
lda_resamples <- tune_grid(
  lda_workflow,
  resamples = folds,
  control = control_stack_grid()
)
```

```
## Loading required package: class

## Loaded mda 0.5-2

##
## Attaching package: 'mda'

## The following object is masked from 'package:parsnip':
##
##     mars

## The workflow being saved contains a recipe, which is 0.4 Mb in memory. If this was not intentional, p
```

```r
# SVM specification (ooh fancy)
svm_spec <- svm_rbf(
    cost = tune(),
    rbf_sigma = tune()
  ) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

# workflow
svm_workflow <-
  gss_workflow %>%
```

```r
  add_model(svm_spec)

# tuning
set.seed(13)
svm_res <-
  tune_grid(
    svm_workflow,
    resamples = folds,
    grid = 3,
    control = control_stack_grid()
  )
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:scales':
##
##     alpha

## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     alpha

## The workflow being saved contains a recipe, which is 0.4 Mb in memory. If this was not intentional, p
```

```r
# Specify random forest
rf_spec <- rand_forest(mtry = tune(),
                       min_n = tune(),
                       trees = 1000) %>%
  set_mode("classification") %>%
  set_engine("ranger")

# Workflow
rf_workflow <-
  gss_workflow %>%
  add_model(rf_spec)

# tuning
set.seed(13)
rf_res <-
  tune_grid(
    rf_workflow,
    resamples = folds,
    grid = 3,
    control = control_stack_grid()
  )
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
## The workflow being saved contains a recipe, which is 0.4 Mb in memory. If this was not intentional, p
```

```
knn_spec <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("classification") %>%
  set_engine("kknn")

knn_wf <- gss_workflow %>%
  add_model(knn_spec)

# tuning
set.seed(13)
knn_res <-
  tune_grid(
    knn_wf,
    resamples = folds,
    control = control_stack_grid()
  )
```

```
## The workflow being saved contains a recipe, which is 0.4 Mb in memory. If this was not intentional, p
```

Now, we can stack these models:

```
gss_stack <- stacks() %>%
  add_candidates(basic_logreg_resamples) %>%
  add_candidates(logreg_resamples) %>%
  add_candidates(lda_resamples) %>%
  add_candidates(svm_res) %>%
  add_candidates(rf_res) %>%
  add_candidates(knn_res)
```

```
## Warning: Predictions from the candidates c(".pred_dem_logreg_resamples_1_002", ".pred_dem_logreg_resa
## ".pred_dem_logreg_resamples_1_044", ".pred_dem_logreg_resamples_1_064", ".pred_dem_logreg_resamples_
## ".pred_dem_logreg_resamples_1_086", ".pred_dem_logreg_resamples_1_106", ".pred_dem_logreg_resamples_
## ".pred_dem_logreg_resamples_1_009", ".pred_dem_logreg_resamples_1_029", ".pred_dem_logreg_resamples_
## ".pred_rep_logreg_resamples_1_022", ".pred_rep_logreg_resamples_1_042", ".pred_rep_logreg_resamples_
## ".pred_rep_logreg_resamples_1_064", ".pred_rep_logreg_resamples_1_084", ".pred_rep_logreg_resamples_
## ".pred_rep_logreg_resamples_1_106", ".pred_rep_logreg_resamples_1_007", ".pred_rep_logreg_resamples_
## ".pred_rep_logreg_resamples_1_029", ".pred_rep_logreg_resamples_1_049", ".pred_rep_logreg_resamples_
```

```
gss_stack
```

```
## # A data stack with 6 model definitions and 91 candidate members:
## #   basic_logreg_resamples: 1 model configuration
## #   logreg_resamples: 65 model configurations
## #   lda_resamples: 10 model configurations
## #   svm_res: 3 model configurations
## #   rf_res: 3 model configurations
## #   knn_res: 9 model configurations
## # Outcome: partyid (factor)
```

```r
gss_stack <- gss_stack %>%
  blend_predictions()
```

```r
gss_stack <- gss_stack %>%
  fit_members()

gss_stack
```

```
## -- A stacked ensemble model -------------------------------------
```

```
##
## Out of 91 possible candidate members, the ensemble retained 11.
## Lasso penalty: 0.01.
```

```
##
## The 10 highest weighted member classes are:
```

```
## # A tibble: 10 x 3
##    member                            type           weight
##    <chr>                             <chr>           <dbl>
##  1 .pred_rep_logreg_resamples_1_100 logistic_reg   1.57
##  2 .pred_rep_logreg_resamples_1_097 logistic_reg   1.36
##  3 .pred_rep_logreg_resamples_1_073 logistic_reg   0.956
##  4 .pred_rep_rf_res_1_2             rand_forest    0.911
##  5 .pred_rep_rf_res_1_3             rand_forest    0.701
##  6 .pred_rep_logreg_resamples_1_112 logistic_reg   0.417
##  7 .pred_rep_lda_resamples_1_07     discrim_linear 0.272
##  8 .pred_rep_logreg_resamples_1_118 logistic_reg   0.203
##  9 .pred_rep_logreg_resamples_1_015 logistic_reg   0.0441
## 10 .pred_rep_lda_resamples_1_04     discrim_linear 0.00112
```

```r
gss_preds <-
  gss_test %>%
  dplyr::select(partyid) %>%
  bind_cols(
    predict(
      gss_stack,
      gss_test,
      members = TRUE
    )
  )

colnames(gss_preds) %>%
  map_dfr(
    .f = accuracy,
    truth = partyid,
    data = gss_preds
  ) %>%
  mutate(member = colnames(gss_preds))
```

```
## # A tibble: 13 x 4
```

```
##    .metric  .estimator .estimate member
##    <chr>    <chr>          <dbl> <chr>
##  1 accuracy binary         1     partyid
##  2 accuracy binary         0.648 .pred_class
##  3 accuracy binary         0.639 .pred_class_logreg_resamples_1_112
##  4 accuracy binary         0.641 .pred_class_logreg_resamples_1_073
##  5 accuracy binary         0.639 .pred_class_logreg_resamples_1_034
##  6 accuracy binary         0.639 .pred_class_logreg_resamples_1_015
##  7 accuracy binary         0.643 .pred_class_logreg_resamples_1_097
##  8 accuracy binary         0.636 .pred_class_logreg_resamples_1_118
##  9 accuracy binary         0.570 .pred_class_logreg_resamples_1_100
## 10 accuracy binary         0.637 .pred_class_lda_resamples_1_04
## 11 accuracy binary         0.637 .pred_class_lda_resamples_1_07
## 12 accuracy binary         0.64  .pred_class_rf_res_1_2
## 13 accuracy binary         0.652 .pred_class_rf_res_1_3
```

For a base-line, let's fit a simple logistic regression:

```r
glm_logreg <- glm(partyid ~ ., data = gss_train, family = "binomial")
probs_logreg <- predict(glm_logreg, gss_test, type = "response")
preds_logreg <- ifelse(probs_logreg >=.5, 1, 0)
confusion_logreg <- table(preds_logreg, gss_test$partyid)
confusion_logreg
```

```
##
## preds_logreg dem rep
##            0 511 213
##            1 316 410
```

```r
accuracy_logreg <- 1 - (confusion_logreg[1,2] + confusion_logreg[2,1]) / nrow(gss_test)
accuracy_logreg
```

```
## [1] 0.6351724
```