Neural Network Hyperparameter Tuning for Flight Delay Classification

Pratik Ganguli

Title: Classification Analysis of Late Flight Arrivals

This example is adapted from https://www.tidymodels.org/start/recipes/

Load required packages

```r
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.2.0 --

v broom        1.0.7     v recipes      1.1.0
v dials        1.3.0     v rsample      1.2.1
v dplyr        1.1.4     v tibble       3.2.1
v ggplot2      3.5.1     v tidyr        1.3.1
v infer        1.0.7     v tune         1.2.1
v modeldata    1.4.0     v workflows    1.1.4
v parsnip      1.2.1     v workflowsets 1.1.0
v purrr        1.0.2     v yardstick    1.3.2

-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x purrr::discard() masks scales::discard()
x dplyr::filter()  masks stats::filter()
x dplyr::lag()     masks stats::lag()
x recipes::step()  masks stats::step()
* Learn how to get started at https://www.tidymodels.org/start/
```

```r
library(themis)
library(nycflights13)
#install.packages("nnet")
#install.packages("NeuralNetTools")
library(nnet)
library(NeuralNetTools)

tidymodels_prefer()
```

Question of interest: Will a plane departing from a New York City airport arrive more than 30 minutes late?

Data: all flights from New York City airports in 2013

Data Preparation

Data Preparation: Data Transformation 1

```r
set.seed(123)
flight_data <-
  flights |>
  mutate(
    # Convert the arrival delay to a factor
    arr_delay = ifelse(arr_delay >= 30, "late", "on_time"),
    arr_delay = factor(arr_delay),
    # We will use the date (not date-time) in the recipe below
    date = lubridate::as_date(time_hour)
  ) |>
  # Include the weather data at origin
  inner_join(weather, by = c("origin", "time_hour")) |>
  # Take a random sample of flights (original dataset is too big)
  # Only do this for demo purposes.
  # On real data use the whole dataset
  slice_sample(n = 10000)
```

Data Transformation 2

```
flight_data <- flight_data |>
  # Only retain the specific columns we will use
  select(flight, time_hour, arr_delay,
         date,  dep_time,
         #air_time,  distance,
         #temp, carrier, origin, dest,
         )  |>
  # Exclude flights with missing arrival delay
  filter(!is.na(arr_delay)) |>
  # Encode qualitative columns as factors (instead of character strings)
  mutate_if(is.character, as.factor)
```

Note: We won't use variables `flight` and `time_hour` in the model, but keeping them in the dataset is useful for identification.

Modelling

Modelling: Pre-processing: Partition the data

80% training, 20% test

```
# Fix the random numbers by setting the seed
# This enables the analysis to be reproducible
# when random numbers are used
set.seed(222)

# Split 80% of the data into the training set
data_split <- initial_split(flight_data, prop = 0.8, strata = arr_delay)

# Create data frames for the three sets:
train_data <- training(data_split)
test_data  <- testing(data_split)
```

Modelling: Pre-processing: Recipe

```
flights_rec <-
  # create recipe and specify formula
  recipe(arr_delay ~ ., data = train_data)  |>
  # update role of ID variables
```

```r
  update_role(flight, time_hour, new_role = "ID") |>
  # pre-process dates - extract day of week and month
  step_date(date, features = c("dow", "month"),
            keep_original_cols = FALSE) |>
  # normalize variables (required for knn)
  step_normalize(all_numeric_predictors()) |>
  # create dummy variables for nominal predictors
  step_dummy(all_nominal_predictors())|>
  # remove zero variance predictors
  step_zv(all_predictors())  |>
  # use upsampling to address class imbalance
  step_upsample(arr_delay, over_ratio = 1)
```

Inspect the impact of the recipe

```r
flights_prepped <-
flights_rec |>
  prep() |>
  bake(new_data = NULL)

flights_prepped |> glimpse()
```

```
Rows: 13,064
Columns: 21
$ flight         <int> 1109, 5699, 1532, 2454, 2189, 371, 471, 5378, 3069, 352~
$ time_hour      <dttm> 2013-02-07 17:00:00, 2013-05-27 14:00:00, 2013-08-15 2~
$ dep_time       <dbl> 0.831601469, 1.145951129, 1.986938534, 1.221476697, 1.7~
$ arr_delay      <fct> late, late, late, late, late, late, late, late, late, l~
$ date_dow_Mon   <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0~
$ date_dow_Tue   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1~
$ date_dow_Wed   <dbl> 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ date_dow_Thu   <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0~
$ date_dow_Fri   <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0~
$ date_dow_Sat   <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0~
$ date_month_Feb <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0~
$ date_month_Mar <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ date_month_Apr <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0~
$ date_month_May <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ date_month_Jun <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0~
$ date_month_Jul <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0~
$ date_month_Aug <dbl> 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
$ date_month_Sep <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ date_month_Oct <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
$ date_month_Nov <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0~
$ date_month_Dec <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1~
```

```r
flights_prepped |>  count(arr_delay)
```

```
# A tibble: 2 x 2
  arr_delay     n
  <fct>     <int>
1 late       6532
2 on_time    6532
```

Tuning: Specify model and workflow

Does the number of hidden nodes and number of iterations make a difference? `tidymodels` has some functions facilitate trying (tuning) different parameters.

Notes:

- The `mlp` has 1 hidden layer, so `hidden_units` specifies the number nodes in the hidden layer.
- epochs are the number of "passes" through the training data

Specify model

```r
nn_model_tune <-
  mlp(hidden_units = tune(),  # number of nodes in hidden layer
      epochs = tune() # number of iterations
      ) |>
  set_engine("nnet") |>
  set_mode("classification")
```

Create workflow = recipe + model

```r
nn_tune_wkflow <- workflow() |>
                  add_model(nn_model_tune) |>
                  add_recipe(flights_rec)
```

Tuning: Define values to test

Create set of values to test

```
# grid_regular chooses sensible values for the parameters
nn_grid <- grid_regular(hidden_units(),
                            epochs(),
                          levels = c(5, 3))
print(nn_grid, n = 16)
```

```
# A tibble: 15 x 2
   hidden_units epochs
          <int>  <int>
 1            1     10
 2            3     10
 3            5     10
 4            7     10
 5           10     10
 6            1    505
 7            3    505
 8            5    505
 9            7    505
10           10    505
11            1   1000
12            3   1000
13            5   1000
14            7   1000
15           10   1000
```

Tuning: Use cross validation to compare hyperparameter values

Create 5 cross-validation folds using training data

```
set.seed(747)
flights_folds <- vfold_cv(train_data, v = 5, strata = arr_delay)
flights_folds
```

```
#  5-fold cross-validation using stratification
# A tibble: 5 x 2
  splits              id
  <list>              <chr>
```

```
1 <split [6217/1555]> Fold1
2 <split [6217/1555]> Fold2
3 <split [6218/1554]> Fold3
4 <split [6218/1554]> Fold4
5 <split [6218/1554]> Fold5
```

Fit model on each fold

```r
flights_metric <- metric_set(
  accuracy,
  roc_auc,
  sens, spec, bal_accuracy)

Sys.time()
```

```
[1] "2025-03-24 03:47:25 NZDT"
```

```r
flights_res <- nn_tune_wkflow |>
  tune_grid(
    # select object containing folds
    resamples = flights_folds,
    # specify grid of values to evaluate
    grid = nn_grid,
    # specify set of metric (optional)
    metrics = flights_metric,
    # save predictions
    control = control_grid(save_pred = TRUE)
    )
Sys.time()
```

```
[1] "2025-03-24 03:48:36 NZDT"
```

Inspect results

```r
flights_res
```

```
# Tuning results
# 5-fold cross-validation using stratification
# A tibble: 5 x 5
  splits              id    .metrics          .notes           .predictions
```

```
  <list>               <chr> <list>            <list>           <list>
1 <split [6217/1555]> Fold1 <tibble [75 x 6]> <tibble [0 x 3]> <tibble>
2 <split [6217/1555]> Fold2 <tibble [75 x 6]> <tibble [0 x 3]> <tibble>
3 <split [6218/1554]> Fold3 <tibble [75 x 6]> <tibble [0 x 3]> <tibble>
4 <split [6218/1554]> Fold4 <tibble [75 x 6]> <tibble [0 x 3]> <tibble>
5 <split [6218/1554]> Fold5 <tibble [75 x 6]> <tibble [0 x 3]> <tibble>
```

```r
# inspect metrics for first fold
flights_res$.metrics[[1]]
```

```
# A tibble: 75 x 6
   hidden_units epochs .metric      .estimator .estimate .config
          <int>  <int> <chr>        <chr>          <dbl> <chr>
 1            1     10 accuracy     binary         0.648 Preprocessor1_Model01
 2            1     10 sens         binary         0.706 Preprocessor1_Model01
 3            1     10 spec         binary         0.637 Preprocessor1_Model01
 4            1     10 bal_accuracy binary         0.671 Preprocessor1_Model01
 5            1     10 roc_auc      binary         0.732 Preprocessor1_Model01
 6            3     10 accuracy     binary         0.668 Preprocessor1_Model02
 7            3     10 sens         binary         0.718 Preprocessor1_Model02
 8            3     10 spec         binary         0.658 Preprocessor1_Model02
 9            3     10 bal_accuracy binary         0.688 Preprocessor1_Model02
10            3     10 roc_auc      binary         0.743 Preprocessor1_Model02
# i 65 more rows
```

Tuning: Evaluate performance of each k

Inspect results for each fold

```r
flights_res |> collect_metrics(summarize = FALSE) |> print(n=7)
```

```
# A tibble: 375 x 7
  id    hidden_units epochs .metric      .estimator .estimate .config
  <chr>        <int>  <int> <chr>        <chr>          <dbl> <chr>
1 Fold1            1     10 accuracy     binary         0.648 Preprocessor1_Mod~
2 Fold1            1     10 sens         binary         0.706 Preprocessor1_Mod~
3 Fold1            1     10 spec         binary         0.637 Preprocessor1_Mod~
4 Fold1            1     10 bal_accuracy binary         0.671 Preprocessor1_Mod~
5 Fold1            1     10 roc_auc      binary         0.732 Preprocessor1_Mod~
6 Fold2            1     10 accuracy     binary         0.719 Preprocessor1_Mod~
7 Fold2            1     10 sens         binary         0.560 Preprocessor1_Mod~
# i 368 more rows
```

Inspect summarised results

```r
flights_res |> collect_metrics() |> print(n=7)
```

```
# A tibble: 75 x 8
  hidden_units epochs .metric      .estimator  mean     n std_err .config
         <int>  <int> <chr>        <chr>      <dbl> <int>   <dbl> <chr>
1            1     10 accuracy     binary     0.669     5 0.0157  Preprocessor1~
2            1     10 bal_accuracy binary     0.655     5 0.00610 Preprocessor1~
3            1     10 roc_auc      binary     0.707     5 0.00901 Preprocessor1~
4            1     10 sens         binary     0.636     5 0.0276  Preprocessor1~
5            1     10 spec         binary     0.675     5 0.0232  Preprocessor1~
6            3     10 accuracy     binary     0.686     5 0.0151  Preprocessor1~
7            3     10 bal_accuracy binary     0.655     5 0.00968 Preprocessor1~
# i 68 more rows
```

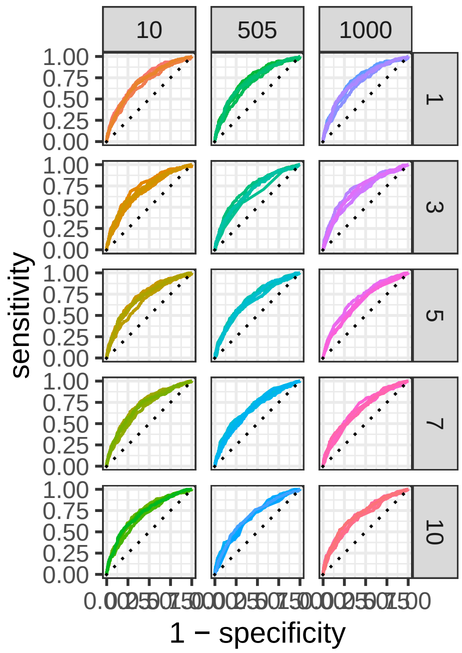
Collect predictions

```r
flights_pred <- flights_res |>  collect_predictions()
flights_pred
```

```
# A tibble: 116,580 x 9
   .pred_class .pred_late .pred_on_time id     .row hidden_units epochs
   <fct>            <dbl>         <dbl> <chr> <int>        <int>  <int>
 1 on_time          0.474         0.526 Fold1     4            1     10
 2 late             0.584         0.416 Fold1     6            1     10
 3 on_time          0.464         0.536 Fold1     9            1     10
 4 late             0.535         0.465 Fold1    12            1     10
 5 late             0.564         0.436 Fold1    20            1     10
 6 late             0.580         0.420 Fold1    23            1     10
 7 late             0.571         0.429 Fold1    24            1     10
 8 late             0.523         0.477 Fold1    32            1     10
 9 late             0.584         0.416 Fold1    33            1     10
10 late             0.578         0.422 Fold1    34            1     10
# i 116,570 more rows
# i 2 more variables: arr_delay <fct>, .config <chr>
```

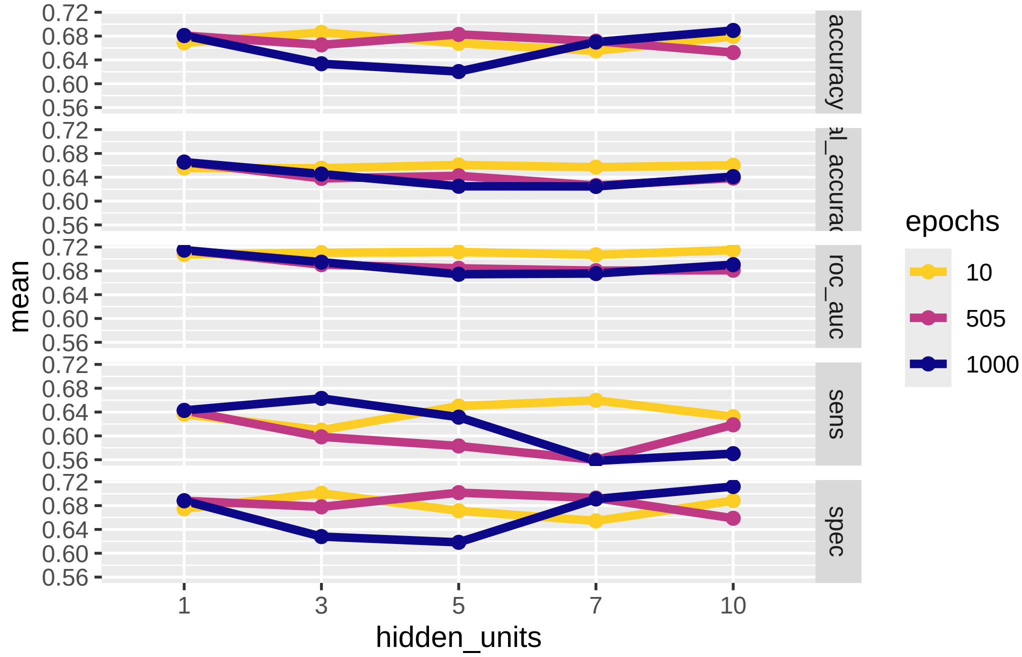
Plot ROC curves

```
flights_pred |>
  group_by(id, hidden_units, epochs) |># id contains folds
  roc_curve(truth = arr_delay, .pred_late) |>
  autoplot() +
  facet_grid(rows = vars(hidden_units), cols = vars(epochs))+ theme(legend.position = "none")
```



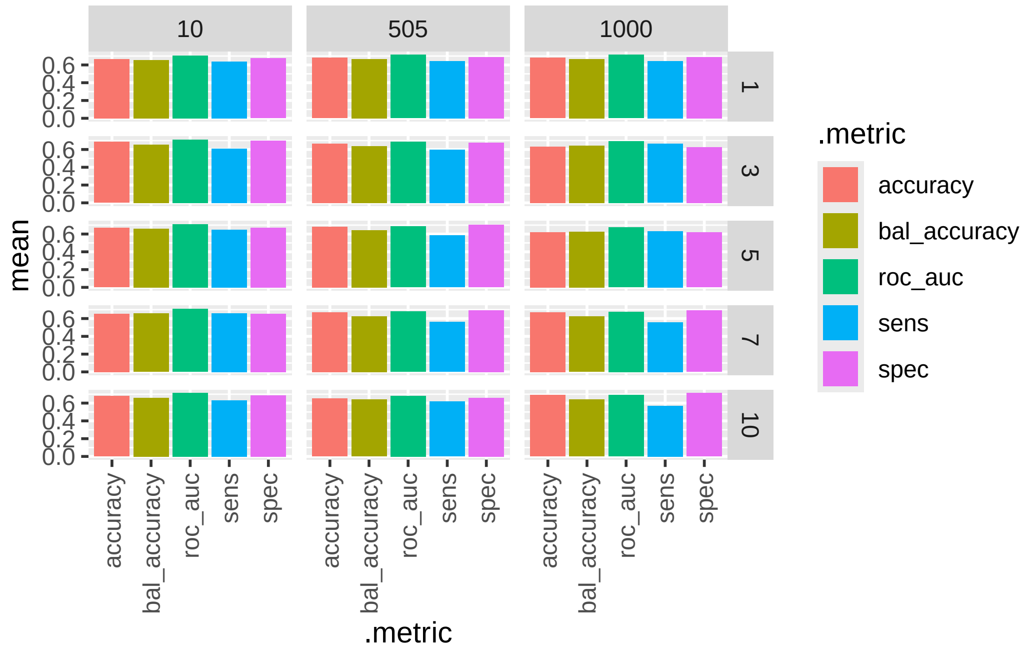
Plot metrics

```
g <-
flights_res |>
  collect_metrics() |>
  mutate(hidden_units = factor(hidden_units),
         epochs = factor(epochs)) |>
  ggplot(aes(hidden_units, mean, color = epochs, group = epochs)) +
  geom_line(linewidth = 1.5) +
  geom_point(size = 2) +
  scale_color_viridis_d(option = "plasma", begin = .9, end = 0)+
  facet_grid(rows =  vars(.metric))
g
```



Alternative representation

```
g <-
flights_res |>
  collect_metrics() |>
  mutate(hidden_units = factor(hidden_units),
         epochs = factor(epochs)) |>
  ggplot(aes(x = .metric, y=mean, fill = .metric, group = .metric)) +
  geom_col(linewidth = 1.5) +
  #geom_point(size = 2) +
  scale_color_viridis_d(option = "plasma", begin = .9, end = 0)+
  facet_grid(rows =  vars(hidden_units ), cols = vars(epochs))+ theme(axis.text.x = element_t
g
```



Tuning: Select the best model

In some applications, it is clear which metric should be used. In this case, use `select_best` with the specified metric.

However sometimes, it is worth making a trade-off to sacrifice the score on one metric, to avoid very poor score on another metric, or to reduce computational complexity (e.g. run-time or number of parameters). These types of trade-offs usually have to be performed manually. The following code demonstrates how to inspect the metrics.

```
flights_res |> select_best(metric ="roc_auc")
```

```
# A tibble: 1 x 3
  hidden_units epochs .config
         <int>  <int> <chr>
1           10     10 Preprocessor1_Model05
```

```
flights_res |> select_best(metric ="sens")
```

```
# A tibble: 1 x 3
  hidden_units epochs .config
```

```
        <int>  <int> <chr>
1           3   1000 Preprocessor1_Model12
```

Inspect training performance for selected parameters

```
flights_res |>
  collect_metrics() |> filter(.config %in% c("Preprocessor1_Model02",
                                            "Preprocessor1_Model11",
                                            "Preprocessor1_Model12"))
```

```
# A tibble: 15 x 8
   hidden_units epochs .metric      .estimator  mean     n std_err .config
          <int>  <int> <chr>        <chr>      <dbl> <int>   <dbl> <chr>
 1            3     10 accuracy     binary     0.686     5 0.0151  Preprocessor~
 2            3     10 bal_accuracy binary     0.655     5 0.00968 Preprocessor~
 3            3     10 roc_auc      binary     0.710     5 0.0109  Preprocessor~
 4            3     10 sens         binary     0.610     5 0.0352  Preprocessor~
 5            3     10 spec         binary     0.701     5 0.0234  Preprocessor~
 6            1   1000 accuracy     binary     0.681     5 0.00641 Preprocessor~
 7            1   1000 bal_accuracy binary     0.665     5 0.00999 Preprocessor~
 8            1   1000 roc_auc      binary     0.715     5 0.0104  Preprocessor~
 9            1   1000 sens         binary     0.643     5 0.0199  Preprocessor~
10            1   1000 spec         binary     0.688     5 0.00730 Preprocessor~
11            3   1000 accuracy     binary     0.634     5 0.0312  Preprocessor~
12            3   1000 bal_accuracy binary     0.645     5 0.0135  Preprocessor~
13            3   1000 roc_auc      binary     0.695     5 0.0125  Preprocessor~
14            3   1000 sens         binary     0.663     5 0.0304  Preprocessor~
15            3   1000 spec         binary     0.628     5 0.0413  Preprocessor~
```

Select the best model

```
(selected_nn_model <-
  flights_res |>
   select_best(metric ="roc_auc")
   #select_best(metric ="sens")
   )
```

```
# A tibble: 1 x 3
  hidden_units epochs .config
         <int>  <int> <chr>
1           10     10 Preprocessor1_Model05
```

```r
## alternatively, manually specify selected model
# selected_nn_model <- tibble(
#   hidden_units = 5,
#   epochs = 10,
#   rowNumber = 2, #obtain from nn_grid
#   .config = paste0("Preprocessor1_Model", ifelse(rowNumber<10, "0", ""), rowNumber)
# )

flights_res |> collect_metrics()|> filter(.config == selected_nn_model$.config)
```

```
# A tibble: 5 x 8
  hidden_units epochs .metric      .estimator  mean     n std_err .config
         <int>  <int> <chr>        <chr>      <dbl> <int>   <dbl> <chr>
1           10     10 accuracy     binary     0.679     5 0.0103  Preprocessor1~
2           10     10 bal_accuracy binary     0.660     5 0.00749 Preprocessor1~
3           10     10 roc_auc      binary     0.715     5 0.00946 Preprocessor1~
4           10     10 sens         binary     0.632     5 0.0183  Preprocessor1~
5           10     10 spec         binary     0.688     5 0.0139  Preprocessor1~
```

Tuning: Finalise the workflow

```r
final_nn_tune_wkflow <-
  nn_tune_wkflow |>
  finalize_workflow(selected_nn_model)
```

Tuning: Final fit

Do a final fit (train on all training data and test on testing data)

```r
final_fit <-
  final_nn_tune_wkflow |>
  last_fit(data_split,
             metrics = flights_metric)
```

Note: choice of metrics will vary for different applications.

Tuning: Evaluate final fit

```
final_fit |>
  collect_metrics()
```

```
# A tibble: 5 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.701 Preprocessor1_Model1
2 sens        binary         0.610 Preprocessor1_Model1
3 spec        binary         0.719 Preprocessor1_Model1
4 bal_accuracy binary        0.664 Preprocessor1_Model1
5 roc_auc     binary         0.705 Preprocessor1_Model1
```

Tuning: Collect predictions and plot ROC curve

```
final_fit |>
  collect_predictions() |>
  roc_curve(truth = arr_delay, .pred_late) |>
  autoplot()
```