Homework Unit 10: Neural Networks

Nicole Friedl

2025-08-07

Introduction

To begin, download the following from the course web book (Unit 10):

- `hw_unit_10_neural_nets.qmd` (notebook for this assignment)
- `wine_quality_trn.csv` and `wine_quality_test.csv` (data for this assignment)

The data for this week's assignment include wine quality ratings for various white wines. The outcome is `quality` - a categorical ("high_quality" or "low_quality") outcome. There are 11 numeric predictors that describe attributes of the wine (e.g., acidity) that may relate to the overall quality rating.

We will be doing another competition this week. You will fit a series of neural network model configurations and select the best configuration among them. You will use the `wine_quality_test.csv` file **only once** at the end of the assignment to generate your best model's predictions in the held-out test set (the test set will not include outcome labels). We will assess everyone's predictions on the held-out set to determine the best fitting model in the class. **The winner again gets a free lunch from John!**

Note that this week's assignment is less structured than previous assignments, allowing you to make more independent decisions about how to approach all aspects of the modeling process. Try to use the knowledge that you have developed from the work in previous assignments to explore your data, generate features, and examine different model configurations.

Let's get started!

Setup

Set up your notebook in this section. You will want to be sure to set your `path_data` and initiate parallel processing here!

```
options(conflicts.policy = "depends.ok")
devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_ml.R?raw=true")
```

```
i SHA-1 hash of file is "32a0bc8ced92c79756b56ddcdc9a06e639795da6"
```

```
tidymodels_conflictRules()

library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(keras, exclude = "get_weights")
library(magrittr, exclude = c("set_names", "extract"))
library(rsample)
library(parsnip)
library(recipes)
```

```
Attaching package: 'recipes'

The following object is masked from 'package:stringr':

    fixed

The following object is masked from 'package:stats':

    step
```

```
library(workflows)


devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_plots.R?raw=true")
```

```
i SHA-1 hash of file is "def6ce26ed7b2493931fde811adff9287ee8d874"
```

```
devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_eda.R?raw=true")
```

```
i SHA-1 hash of file is "c045eee2655a18dc85e715b78182f176327358a7"
```

```
options(tibble.width = Inf, dplyr.print_max=Inf)
rerun_setting <- FALSE

cl <- parallel::makePSOCKcluster(parallel::detectCores(logical = FALSE))
doParallel::registerDoParallel(cl)

path_data <- "homework/data"
```

Expectations for minimum model requirements

Across the model configurations you compare, you must fit models that vary with respect to:

- Hidden layer architecture: Consider **at least two** different numbers of units within the hidden layer.
- Controlling overfitting: Consider **at least one** method to control overfitting (either L2 regularization or drop-out) with **at least two** associated hyperparameter values.
- Hidden layer activation function: Consider **at least two** activation functions for the hidden layer.

Some things to consider

About configurations

Of course you can't choose between model configurations based on training set performance, so you'll need to use some resampling technique. There are different costs and benefits associated with different resampling techniques (e.g., bias, variance, computing time), so you'll need to decide which technique is the best for your needs for this assignment. Specifically, you should choose among a validation split, k-fold cross-validation, or bootstrapping for cross-validation.

Resampling and tuning

Given what you've learned about resampling/tuning, think about how you might move systematically through model configurations rather than haphazardly changing model configuration characteristics.

Consider compute time

As you weigh computing costs, think about what that might look like given your current context. For example, imagine that each model configuration takes 2 minutes to fit. If you want to use 100 bootstraps for CV, that means ~200 minutes (just over 3 hours) per model configuration. Now imagine you're comparing 8 different model configurations. That multiplies your 200 minutes by 8, which starts to get pretty long. If you're using 10-fold CV, that means it only takes 20 minutes per model configuration, so you might be able to compare more configurations. A validation split would be even simpler. Think about the costs and benefits of each approach, pick one and motivate it with commentary in your submission.

Performance metric: accuracy

Regardless of the resampling technique you choose, please compare models using **accuracy** as your performance metric.

Okay, now onto some EDA and then modeling…

Feature engineering

Read in `wine_quality_trn.csv`

```
data_trn <- data_trn <- read_csv("/Users/nicolefriedl/Desktop/Psych752/homework/data/wine_qua
```

```
Rows: 3674 Columns: 12
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (1): quality
dbl (11): fixed_acidity, volatile_acidity, citric_acid, residual_sugar, chlo...

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.
```

```
glimpse(data_trn)
```

```
Rows: 3,674
Columns: 12
$ fixed_acidity        <dbl> 7.0, 6.3, 7.2, 7.2, 8.1, 7.0, 6.3, 8.1, 7.9, 6.6,~
$ volatile_acidity     <dbl> 0.27, 0.30, 0.23, 0.23, 0.28, 0.27, 0.30, 0.27, 0~
$ citric_acid          <dbl> 0.36, 0.34, 0.32, 0.32, 0.40, 0.36, 0.34, 0.41, 0~
$ residual_sugar       <dbl> 20.70, 1.60, 8.50, 8.50, 6.90, 20.70, 1.60, 1.45,~
$ chlorides            <dbl> 0.045, 0.049, 0.058, 0.058, 0.050, 0.045, 0.049, ~
$ free_sulfur_dioxide  <dbl> 45, 14, 47, 47, 30, 45, 14, 11, 16, 48, 41, 28, 3~
$ total_sulfur_dioxide <dbl> 170, 132, 186, 186, 97, 170, 132, 63, 75, 143, 17~
$ density              <dbl> 1.0010, 0.9940, 0.9956, 0.9956, 0.9951, 1.0010, 0~
$ ph                   <dbl> 3.00, 3.30, 3.19, 3.19, 3.26, 3.00, 3.30, 2.99, 3~
$ sulphates            <dbl> 0.45, 0.49, 0.40, 0.40, 0.44, 0.45, 0.49, 0.56, 0~
$ alcohol              <dbl> 8.8, 9.5, 9.9, 9.9, 10.1, 8.8, 9.5, 12.0, 10.8, 1~
$ quality              <chr> "high_quality", "high_quality", "high_quality", "~
```

Perform some modeling EDA. How will you scale your features? Should your features be normalized, scaled, or transformed in some other way? Decorrelated? Provide an explanation for your decisions here along with as much EDA as you find necessary.

To ensure our neural network performs optimally, I will standardize most features using Z-score scaling, as many follow a roughly normal distribution. However, highly skewed features like chlorides, residual_sugar, free_sulfur_dioxide, and total_sulfur_dioxide will be log-transformed to reduce skewness and improve model stability. Standardization ensures all features contribute equally, preventing those with larger magnitudes from dominating the model. I also checked feature correlations and may consider removing or combining highly correlated variables to reduce redundancy and improve generalization.

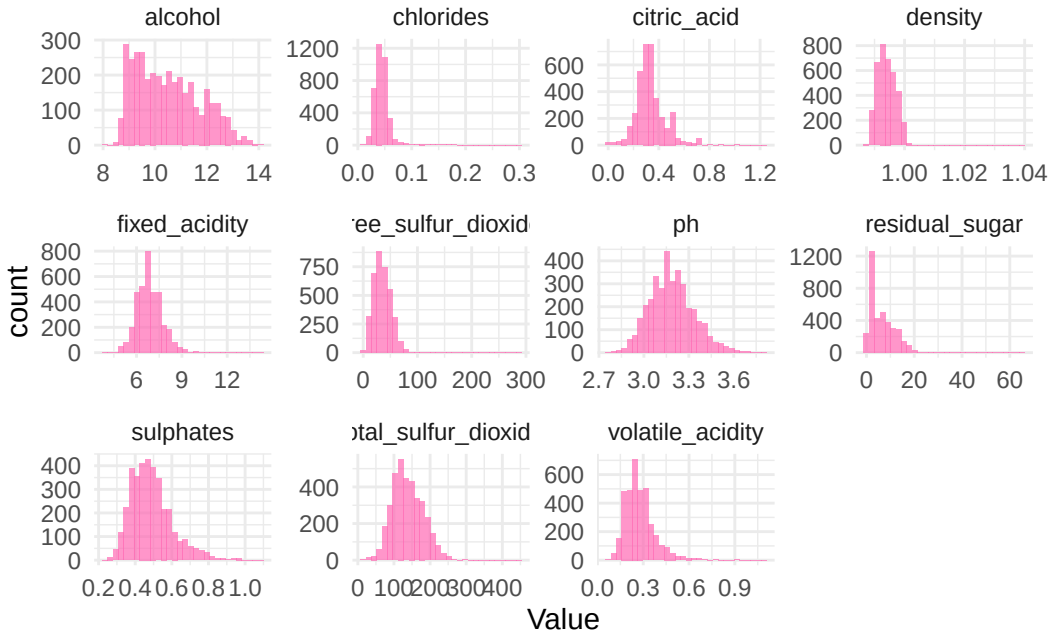
```
data_trn %>%
  count(quality) %>%
  mutate(prop = n / sum(n))
```

```
# A tibble: 2 x 3
  quality          n  prop
  <chr>        <int> <dbl>
1 high_quality  2444 0.665
2 low_quality   1230 0.335
```

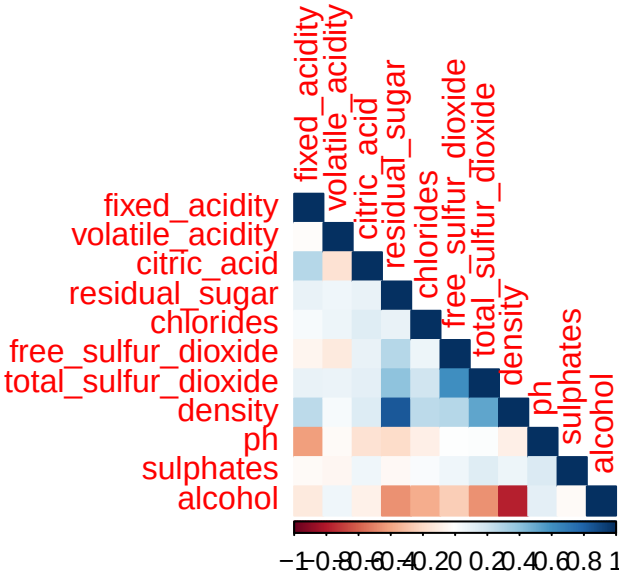
```
data_trn %>%
  select(-quality) %>%
  summary()
```

```
 fixed_acidity    volatile_acidity  citric_acid     residual_sugar
 Min.   : 3.900   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.100
 Mean   : 6.864   Mean   :0.2775   Mean   :0.3352   Mean   : 6.327
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.800
 Max.   :14.200   Max.   :1.1000   Max.   :1.2300   Max.   :65.800
   chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
 Min.   :0.01200   Min.   :  2.00      Min.   :  9.0        Min.   :0.9871
 1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:107.0        1st Qu.:0.9917
 Median :0.04300   Median : 34.00      Median :133.0        Median :0.9937
 Mean   :0.04541   Mean   : 35.29      Mean   :138.1        Mean   :0.9940
 3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:168.0        3rd Qu.:0.9960
 Max.   :0.30100   Max.   :289.00      Max.   :440.0        Max.   :1.0390
       ph           sulphates         alcohol
 Min.   :2.770   Min.   :0.2300   Min.   : 8.00
 1st Qu.:3.080   1st Qu.:0.4100   1st Qu.: 9.50
 Median :3.180   Median :0.4700   Median :10.40
 Mean   :3.188   Mean   :0.4899   Mean   :10.52
 3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40
 Max.   :3.810   Max.   :1.0800   Max.   :14.00
```

```r
data_trn %>%
  pivot_longer(-quality, names_to = "Feature", values_to = "Value") %>%
  ggplot(aes(x = Value)) +
  geom_histogram(bins = 30, fill = "hotpink", alpha = 0.7) +
  facet_wrap(~Feature, scales = "free") +
  theme_minimal()
```




```
cor_matrix <- cor(data_trn %>% select(-quality))
corrplot::corrplot(cor_matrix, method = "color", type = "lower")
```



Fit models

Depending on the resampling technique you choose and how you plan to examine various model configurations, your code set-up is going to look different. Create as many code chunks

as needed for whatever your approach.

Where needed, please include some annotation (in text outside of code chunks and/or in commented lines within code chunks) to help us review your assignment. You don't need to tell us everything you're doing because that should be relatively clear from the code! A good rule of thumb is to use annotation to tell us **why** you're doing something (e.g., if you had several choices for how to proceed, why did you choose the one you did?) but you don't have to describe **what** you're doing (e.g., you don't need to tell us you're building a recipe - we will see that in your code).

For my neural network modeling approach, I decided to use a validation split strategy that would allow me to efficiently tune hyperparameters while maintaining a test set for final evaluation. I set up a structured workflow that includes data preprocessing, model specification, hyperparameter tuning, and final model fitting. I began by creating a reproducible environment with a fixed random seed and splitting my data into training and validation sets with an 80-20 split. For my recipe, I applied log transformations to variables with potential skewness (residual_sugar, free_sulfur_dioxide, and total_sulfur_dioxide) to improve the neural network's performance on these typically right-skewed features. I also normalized all numeric predictors to ensure they're on the same scale, which is crucial for neural network convergence.

I chose to create a multilayer perceptron with tunable hyperparameters including: Hidden units (network width), dropout rate (regularization), activation function. I fixed the epochs at 30 to prevent overfitting while still allowing sufficient training, and set the batch size to 32 to balance computation efficiency and gradient accuracy.

I chose to test: Two network sizes (10 and 20 hidden units) to compare simpler vs. more complex architectures Dropout vs. no dropout (0 vs. 0.3) to evaluate regularization benefits ReLU vs. sigmoid activation functions to compare modern vs. traditional activation approaches. For the validation strategy, I implemented another validation split within the training data to avoid touching my final test set during tuning. This approach balances computational efficiency with thorough exploration of key neural network parameters, focusing on accuracy as the primary metric for wine quality classification.

```
set.seed(123)

data_split <- initial_split(data_trn, prop = 0.8)
train_data <- training(data_split)
val_data <- testing(data_split)

recipe_nn <- recipe(quality ~ ., data = train_data) %>%
  step_log(residual_sugar, free_sulfur_dioxide, total_sulfur_dioxide, offset = 1) %>%
  step_normalize(all_numeric_predictors())
```

```r
nn_spec <- mlp(
  hidden_units = tune(),
  dropout = tune(),
  activation = tune(),
  epochs = 30
) %>%
  set_engine("keras", batch_size = 32) %>%
  set_mode("classification")
```

```r
nn_wf <- workflow() %>%
  add_recipe(recipe_nn) %>%
  add_model(nn_spec)
```

```r
library(dials)
```

```
Loading required package: scales
```

```
Attaching package: 'scales'
```

```
The following object is masked from 'package:purrr':

    discard
```

```r
library(tune)
nn_grid <- expand_grid(
  hidden_units = c(10, 20),
  dropout = c(0, 0.3),
  activation = c("relu", "sigmoid")
)

nn_results <- nn_wf %>%
  tune_grid(
    resamples = validation_split(train_data, prop = 0.8),
    grid = nn_grid,
    metrics = yardstick::metric_set(yardstick::accuracy),
    control = control_grid(verbose = TRUE)
  )
```

```
Warning: `validation_split()` was deprecated in rsample 1.2.0.
i Please use `initial_validation_split()` instead.
```

```
! Some required packages prohibit parallel processing:  'keras'

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 1/8

v validation: preprocessor 1/1, model 1/8

i validation: preprocessor 1/1, model 1/8 (extracts)

i validation: preprocessor 1/1, model 1/8 (predictions)

19/19 - 0s - 81ms/epoch - 4ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 2/8

v validation: preprocessor 1/1, model 2/8

i validation: preprocessor 1/1, model 2/8 (extracts)

i validation: preprocessor 1/1, model 2/8 (predictions)

19/19 - 0s - 54ms/epoch - 3ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 3/8

v validation: preprocessor 1/1, model 3/8
```

```
i validation: preprocessor 1/1, model 3/8 (extracts)

i validation: preprocessor 1/1, model 3/8 (predictions)

19/19 - 0s - 63ms/epoch - 3ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 4/8

v validation: preprocessor 1/1, model 4/8

i validation: preprocessor 1/1, model 4/8 (extracts)

i validation: preprocessor 1/1, model 4/8 (predictions)

19/19 - 0s - 60ms/epoch - 3ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 5/8

v validation: preprocessor 1/1, model 5/8

i validation: preprocessor 1/1, model 5/8 (extracts)

i validation: preprocessor 1/1, model 5/8 (predictions)

19/19 - 0s - 56ms/epoch - 3ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1
```

```
i validation: preprocessor 1/1, model 6/8

v validation: preprocessor 1/1, model 6/8

i validation: preprocessor 1/1, model 6/8 (extracts)

i validation: preprocessor 1/1, model 6/8 (predictions)

19/19 - 0s - 54ms/epoch - 3ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 7/8

v validation: preprocessor 1/1, model 7/8

i validation: preprocessor 1/1, model 7/8 (extracts)

i validation: preprocessor 1/1, model 7/8 (predictions)

19/19 - 0s - 62ms/epoch - 3ms/step

i validation: preprocessor 1/1

v validation: preprocessor 1/1

i validation: preprocessor 1/1, model 8/8

v validation: preprocessor 1/1, model 8/8

i validation: preprocessor 1/1, model 8/8 (extracts)

i validation: preprocessor 1/1, model 8/8 (predictions)

19/19 - 0s - 63ms/epoch - 3ms/step
```

```r
show_best(nn_results, metric = "accuracy")
```

```
# A tibble: 5 x 9
  hidden_units dropout activation .metric  .estimator  mean     n std_err
         <dbl>   <dbl> <chr>      <chr>    <chr>      <dbl> <int>   <dbl>
1           20     0   relu       accuracy binary     0.776     1      NA
2           10     0.3 relu       accuracy binary     0.770     1      NA
3           20     0.3 relu       accuracy binary     0.770     1      NA
4           10     0   relu       accuracy binary     0.769     1      NA
5           10     0.3 sigmoid    accuracy binary     0.745     1      NA
  .config
  <chr>
1 Preprocessor1_Model5
2 Preprocessor1_Model3
3 Preprocessor1_Model7
4 Preprocessor1_Model1
5 Preprocessor1_Model4
```

```r
best_nn <- finalize_workflow(nn_wf, select_best(nn_results, metric = "accuracy"))
```

```r
final_nn_fit <- fit(best_nn, data = data_trn)
```

```
Epoch 1/30
115/115 - 1s - loss: 0.6482 - 565ms/epoch - 5ms/step
Epoch 2/30
115/115 - 0s - loss: 0.5753 - 121ms/epoch - 1ms/step
Epoch 3/30
115/115 - 0s - loss: 0.5338 - 142ms/epoch - 1ms/step
Epoch 4/30
115/115 - 0s - loss: 0.5089 - 133ms/epoch - 1ms/step
Epoch 5/30
115/115 - 0s - loss: 0.4948 - 129ms/epoch - 1ms/step
Epoch 6/30
115/115 - 0s - loss: 0.4870 - 165ms/epoch - 1ms/step
Epoch 7/30
115/115 - 0s - loss: 0.4813 - 130ms/epoch - 1ms/step
Epoch 8/30
115/115 - 0s - loss: 0.4779 - 153ms/epoch - 1ms/step
Epoch 9/30
115/115 - 0s - loss: 0.4747 - 125ms/epoch - 1ms/step
```

```
Epoch 10/30
115/115 - 0s - loss: 0.4723 - 131ms/epoch - 1ms/step
Epoch 11/30
115/115 - 0s - loss: 0.4700 - 126ms/epoch - 1ms/step
Epoch 12/30
115/115 - 0s - loss: 0.4675 - 134ms/epoch - 1ms/step
Epoch 13/30
115/115 - 0s - loss: 0.4660 - 124ms/epoch - 1ms/step
Epoch 14/30
115/115 - 0s - loss: 0.4645 - 127ms/epoch - 1ms/step
Epoch 15/30
115/115 - 0s - loss: 0.4630 - 363ms/epoch - 3ms/step
Epoch 16/30
115/115 - 0s - loss: 0.4615 - 133ms/epoch - 1ms/step
Epoch 17/30
115/115 - 0s - loss: 0.4602 - 119ms/epoch - 1ms/step
Epoch 18/30
115/115 - 0s - loss: 0.4597 - 141ms/epoch - 1ms/step
Epoch 19/30
115/115 - 0s - loss: 0.4580 - 132ms/epoch - 1ms/step
Epoch 20/30
115/115 - 0s - loss: 0.4570 - 131ms/epoch - 1ms/step
Epoch 21/30
115/115 - 0s - loss: 0.4562 - 124ms/epoch - 1ms/step
Epoch 22/30
115/115 - 0s - loss: 0.4555 - 130ms/epoch - 1ms/step
Epoch 23/30
115/115 - 0s - loss: 0.4545 - 134ms/epoch - 1ms/step
Epoch 24/30
115/115 - 0s - loss: 0.4539 - 128ms/epoch - 1ms/step
Epoch 25/30
115/115 - 0s - loss: 0.4532 - 130ms/epoch - 1ms/step
Epoch 26/30
115/115 - 0s - loss: 0.4531 - 123ms/epoch - 1ms/step
Epoch 27/30
115/115 - 0s - loss: 0.4521 - 133ms/epoch - 1ms/step
Epoch 28/30
115/115 - 0s - loss: 0.4520 - 124ms/epoch - 1ms/step
Epoch 29/30
115/115 - 0s - loss: 0.4513 - 135ms/epoch - 1ms/step
Epoch 30/30
115/115 - 0s - loss: 0.4509 - 133ms/epoch - 1ms/step
```

Generate predictions

This section is for generating predictions for your best model in the held-out test set. You should only generate predictions for **one model** out of all configurations you examined. We will use these predictions to generate your one estimate of model performance in the held-out data.

Add your last name between the quotation marks to be used in naming your predictions file.

```
last_name <- "Friedl"
```

Read in test data

Read in the `wine_quality_test.csv` file.

```
data_test <- read_csv("/Users/nicolefriedl/Desktop/Psych752/homework/data/wine_quality_test.c
```

```
Rows: 1224 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (11): fixed_acidity, volatile_acidity, citric_acid, residual_sugar, chlo...

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.
```

Make feature matrices

Make training feature matrix using your best recipe.

```
feat_trn_best <- recipe_nn %>%
  prep(training = data_trn) %>%
  bake(new_data = data_trn)
```

Make test feature matrix using your best recipe.

```
feat_test_best <- recipe_nn %>%
  prep(training = data_trn) %>%
  bake(new_data = data_test)
```

Fit best model

Fit your best model in `feat_trn_best.csv` (no resampling).

```
best_params <- select_best(nn_results, metric = "accuracy")

best_model <- mlp(
  hidden_units = best_params$hidden_units,
  dropout = best_params$dropout,
  activation = best_params$activation,
  epochs = 30
) %>%
  set_engine("keras", batch_size = 32) %>%
  set_mode("classification") %>%
  fit(quality ~ ., data = feat_trn_best)
```

```
Epoch 1/30
115/115 - 1s - loss: 0.6391 - 561ms/epoch - 5ms/step
Epoch 2/30
115/115 - 0s - loss: 0.5444 - 137ms/epoch - 1ms/step
Epoch 3/30
115/115 - 0s - loss: 0.5112 - 137ms/epoch - 1ms/step
Epoch 4/30
115/115 - 0s - loss: 0.4967 - 137ms/epoch - 1ms/step
Epoch 5/30
115/115 - 0s - loss: 0.4883 - 144ms/epoch - 1ms/step
Epoch 6/30
115/115 - 0s - loss: 0.4827 - 123ms/epoch - 1ms/step
Epoch 7/30
115/115 - 0s - loss: 0.4785 - 121ms/epoch - 1ms/step
Epoch 8/30
115/115 - 0s - loss: 0.4753 - 124ms/epoch - 1ms/step
Epoch 9/30
115/115 - 0s - loss: 0.4728 - 135ms/epoch - 1ms/step
Epoch 10/30
115/115 - 0s - loss: 0.4704 - 124ms/epoch - 1ms/step
Epoch 11/30
115/115 - 0s - loss: 0.4686 - 126ms/epoch - 1ms/step
Epoch 12/30
115/115 - 0s - loss: 0.4667 - 133ms/epoch - 1ms/step
Epoch 13/30
115/115 - 0s - loss: 0.4655 - 122ms/epoch - 1ms/step
```

```
Epoch 14/30
115/115 - 0s - loss: 0.4640 - 122ms/epoch - 1ms/step
Epoch 15/30
115/115 - 0s - loss: 0.4630 - 128ms/epoch - 1ms/step
Epoch 16/30
115/115 - 0s - loss: 0.4614 - 132ms/epoch - 1ms/step
Epoch 17/30
115/115 - 0s - loss: 0.4612 - 131ms/epoch - 1ms/step
Epoch 18/30
115/115 - 0s - loss: 0.4601 - 124ms/epoch - 1ms/step
Epoch 19/30
115/115 - 0s - loss: 0.4593 - 128ms/epoch - 1ms/step
Epoch 20/30
115/115 - 0s - loss: 0.4585 - 122ms/epoch - 1ms/step
Epoch 21/30
115/115 - 0s - loss: 0.4583 - 132ms/epoch - 1ms/step
Epoch 22/30
115/115 - 0s - loss: 0.4571 - 127ms/epoch - 1ms/step
Epoch 23/30
115/115 - 0s - loss: 0.4568 - 135ms/epoch - 1ms/step
Epoch 24/30
115/115 - 0s - loss: 0.4557 - 132ms/epoch - 1ms/step
Epoch 25/30
115/115 - 0s - loss: 0.4557 - 130ms/epoch - 1ms/step
Epoch 26/30
115/115 - 0s - loss: 0.4550 - 126ms/epoch - 1ms/step
Epoch 27/30
115/115 - 0s - loss: 0.4545 - 126ms/epoch - 1ms/step
Epoch 28/30
115/115 - 0s - loss: 0.4538 - 122ms/epoch - 1ms/step
Epoch 29/30
115/115 - 0s - loss: 0.4533 - 133ms/epoch - 1ms/step
Epoch 30/30
115/115 - 0s - loss: 0.4529 - 130ms/epoch - 1ms/step
```

Generate test predictions

Run this code chunk to save out your best model's predictions in the held-out test set. Look over your predictions to confirm your model generated valid predictions for each test observation. Make sure the file containing your predictions has the form that you think it should. This requires visually inspecting the output csv file after you write it. The `glimpse()` call helps too.

```r
library(here)
```

```
here() starts at /Users/nicolefriedl/Desktop/Psych752
```

```r
feat_test_best %>%
  mutate(quality = predict(best_model, feat_test_best)$.pred_class) %>%
  select(quality) %>%
  glimpse() %>%
  write_csv(here(path_data, str_c("test_preds_", last_name, ".csv")))
```

```
39/39 - 0s - 66ms/epoch - 2ms/step
Rows: 1,224
Columns: 1
$ quality <fct> high_quality, low_quality, high_quality, low_quality, low_qual~
```

Save & render

Render this .qmd file with your last name at the end (e.g., hw_unit_10_neural_nets_name.qmd). Make sure you changed "Your name here" at the top of the file to be your own name. Render the file to html. Upload the rendered file and your saved test_preds_lastname.csv to Canvas.

We will assess everyone's predictions in the test, and the winner gets a free lunch from John!

Way to go!!