Train and Evaluate Classification Models using Tidymodels

Eric Wanjau

Classification Challenge

Wine experts can identify wines from specific vineyards through smell and taste, but the factors that give different wines their individual characteristics are actually based on their chemical composition.

In this challenge, you must train a classification model to analyze the chemical and visual features of wine samples and classify them based on their cultivar (grape variety).

> **Citation**: The data used in this exercise was originally collected by Forina, M. et al.
>
> PARVUS - An Extendible Package for Data Exploration, Classification and Correlation. Institute of Pharmaceutical and Food Analysis and Technologies, Via Brigata Salerno, 16147 Genoa, Italy.
>
> It can be downloaded from the UCI dataset repository (Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science).

Explore the data

Let's hit the ground running by importing a CSV file of wine data, which consists of 12 numeric features and a classification label with the following classes:

- **0** (*variety A*)
- **1** (*variety B*)
- **2** (*variety C*)

```
# Load the required packages into your current R session
suppressPackageStartupMessages({
  library(tidyverse)
  library(tidymodels)
  library(here)
  library(janitor)
})

# Read the csv file into a tibble
wine_data <- read_csv(file = "https://raw.githubusercontent.com/MicrosoftDocs/ml-basics/master/challenge

# Print the first 10 rows of the data
wine_data %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 14
##    Alcohol Malic_acid   Ash Alcalinity Magnesium Phenols Flavanoids
##      <dbl>      <dbl> <dbl>      <dbl>     <dbl>   <dbl>      <dbl>
##  1    14.2       1.71  2.43       15.6       127    2.8        3.06
##  2    13.2       1.78  2.14       11.2       100    2.65       2.76
##  3    13.2       2.36  2.67       18.6       101    2.8        3.24
##  4    14.4       1.95  2.5        16.8       113    3.85       3.49
##  5    13.2       2.59  2.87       21         118    2.8        2.69
##  6    14.2       1.76  2.45       15.2       112    3.27       3.39
##  7    14.4       1.87  2.45       14.6        96    2.5        2.52
##  8    14.1       2.15  2.61       17.6       121    2.6        2.51
##  9    14.8       1.64  2.17       14          97    2.8        2.98
## 10    13.9       1.35  2.27       16          98    2.98       3.15
## # ... with 7 more variables: Nonflavanoids <dbl>, Proanthocyanins <dbl>,
## #   Color_intensity <dbl>, Hue <dbl>, OD280_315_of_diluted_wines <dbl>,
## #   Proline <dbl>, WineVariety <dbl>
```

Your challenge is to explore the data and train a classification model that achieves an overall *Recall* metric of over 0.95 (95%).

Let's kick off our adventure by reformatting the data to make it easier for a model to use effectively.

Question 1.

Starting with the `wine_data`, encode the `WineVariety` column to a categorical variable (factor) with levels set as: 0, 1 ,2.

```
BEGIN QUESTION
name: Question 1
manual: false
```

```
# Load the janitor package for cleaning data
library(janitor)

# Clean data a bit
wine_data <- wine_data %>%
  # Encode WineVariety as category
  mutate(WineVariety = factor(WineVariety, levels = c(0, 1, 2))) %>%
  # Clean column names
  clean_names()


# View data set
wine_data %>%
  glimpse()
```

```
## Rows: 178
## Columns: 14
## $ alcohol                      <dbl> 14.23, 13.20, 13.16, 14.37, 13.24, 14.20, 1~
## $ malic_acid                   <dbl> 1.71, 1.78, 2.36, 1.95, 2.59, 1.76, 1.87, 2~
## $ ash                          <dbl> 2.43, 2.14, 2.67, 2.50, 2.87, 2.45, 2.45, 2~
## $ alcalinity                   <dbl> 15.6, 11.2, 18.6, 16.8, 21.0, 15.2, 14.6, 1~
## $ magnesium                    <dbl> 127, 100, 101, 113, 118, 112, 96, 121, 97, ~
## $ phenols                      <dbl> 2.80, 2.65, 2.80, 3.85, 2.80, 3.27, 2.50, 2~
## $ flavanoids                   <dbl> 3.06, 2.76, 3.24, 3.49, 2.69, 3.39, 2.52, 2~
## $ nonflavanoids                <dbl> 0.28, 0.26, 0.30, 0.24, 0.39, 0.34, 0.30, 0~
## $ proanthocyanins              <dbl> 2.29, 1.28, 2.81, 2.18, 1.82, 1.97, 1.98, 1~
## $ color_intensity              <dbl> 5.64, 4.38, 5.68, 7.80, 4.32, 6.75, 5.25, 5~
## $ hue                          <dbl> 1.04, 1.05, 1.03, 0.86, 1.04, 1.05, 1.02, 1~
## $ od280_315_of_diluted_wines <dbl> 3.92, 3.40, 3.17, 3.45, 2.93, 2.85, 3.58, 3~
## $ proline                      <dbl> 1065, 1050, 1185, 1480, 735, 1450, 1290, 12~
## $ wine_variety                 <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
. = " # BEGIN TEST CONFIG

success_message: Great start! Your tibble dimensions are correct.

failure_message: Almost there! Let's take a look at this again. Expected dimensions [178 14]
" # END TEST CONFIG

suppressPackageStartupMessages({
  library(testthat)
  library(ottr)
})

## Test ##
test_that('data dimensions correct', {
  expect_equal(dim(wine_data), c(178, 14))

})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

success_message: Excellent. You have successfully encoded the **wine_variety** column as a factor varia

failure_message: Almost there! Ensure the **wine_variety column is encoded as a factor with levels 0, 1
" # END TEST CONFIG




## Test ##
test_that('wine_variety is encoded as categorical', {
    expect_equal(class(wine_data$wine_variety), "factor")
  expect_equal(levels(wine_data$wine_variety), c("0", "1", "2"))


})
```

```
## Test passed
```

The goal of data exploration is to try to understand the `relationships` between its attributes; in particular, any apparent correlation between the *features* and the *label* your model will try to predict. One way of doing this is by using data visualization.

Now let's compare the feature distributions for each label value.

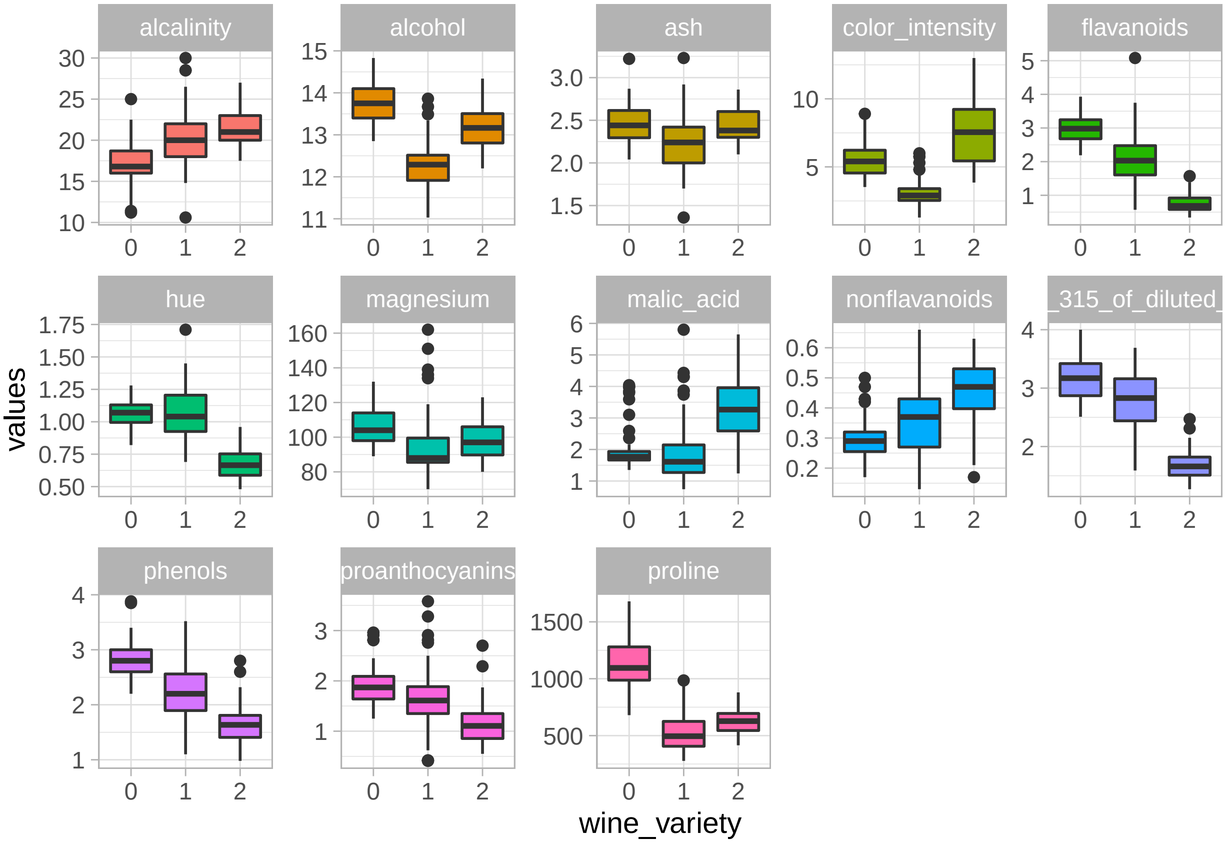
Question 2.

We'll begin by restructuring the data such that we can easily plot our data as facets, subplots that each display one subset of the data. This is done using `facet_wrap`

- `pivot_longer` the data (increase the number of rows and decrease the number of columns) such that all the existing column names except **wine__variety** now fall under a new column name called **`features`** and their corresponding values under a new column name **`values`**.

```
BEGIN QUESTION
name: Question 2
manual: false
```

```
theme_set(theme_light())
# Pivot data to a long format
wine_data_long <- wine_data %>%
    pivot_longer(!wine_variety, names_to = "features", values_to = "values")


# Make a box plot for each predictor feature
wine_data_long %>%
  ggplot(mapping = aes(x = wine_variety, y = values, fill = features)) +
  geom_boxplot() +
  facet_wrap(~ features, scales = "free", ncol = 5) +
  scale_color_viridis_d(option = "plasma", end = .7) +
  theme(legend.position = "none")
```



What insights about the wine varieties have you derived from the distribution of the different features?

```
. = " # BEGIN TEST CONFIG

success_message: Congratulations! You have successfully pivoted the data to a longer format suitable fo

failure_message: Almost there! Ensure you pivoted the exsiting columns except wine_variety to obtain tw
" # END TEST CONFIG


## Test ##
test_that('data dimensions correct', {
  expect_equal(dim(wine_data_long), c(2314, 3))
  expect_equal(sort(colnames(wine_data_long)), c("features", "values", "wine_variety"))

  expect_equal(sort(unique(wine_data_long$features)) %>% paste(collapse = ", "), "alcalinity, alcohol, 

})
```

```
## Test passed
```

Split the data for training and validation

It is best practice to hold out some of your data for **testing** in order to get a better estimate of how your models will perform on new data by comparing the predicted labels with the already known labels in the test set.

Question 3.

In this section:

- Make a split specification of `wine_data` such that `70%` goes to training and the rest goes to testing. Save this to a variable name `wine_split`
- Extract the training and testing sets from `wine_split` and save them in `wine_train` and `wine_test` variable names respectively.

```
BEGIN QUESTION
name: Question 3
manual: false
```

```r
# Load the Tidymodels packages
library(tidymodels)



# Split data into 70% for training and 30% for testing
set.seed(2056)
wine_split <- wine_data %>%
  initial_split(prop = 0.70)


# Extract the data in each split
wine_train <- training(wine_split)
wine_test <- testing(wine_split)


# Print the number of cases in each split
cat("Training cases: ", nrow(wine_train), "\n",
    "Test cases: ", nrow(wine_test), sep = "")
```

```
## Training cases: 124
## Test cases: 54
```

```r
. = " # BEGIN TEST CONFIG

success_message: Fantastic! You have successfully split the data and extracted the training (70%) and te

failure_message: Almost there. Let's have a look at this again.Ensure that the splitting specification o
" # END TEST CONFIG


## Test ##
test_that('data dimensions correct', {
  expect_equal(dim(wine_train), c(124, 14))
  expect_equal(dim(wine_test), c(54, 14))

})
```

```
## Test passed
```

Data preprocessing with recipes

Now that we have a set of training features and corresponding training label (`wine_variety`), we can fit a multiclass classification algorithm to the data to create a model.

`parsnip::multinom_reg()` defines a model that uses linear predictors to predict multiclass data using the multinomial distribution.

Question 4.

In this section:

- Create a multinomial model specification `mr_spec` which uses `nnet` package as its engine and whose mode is set to `classifcation`. This model has 1 tuning parameters: `penalty`: Amount of Regularization. The best way to estimate this value is to `tune` it, but for now, set it to an arbitrary value, say `1`.
- Create a recipe, `wine_recipe`, with `wine_variety` as the outcome and the rest as predictors, and a step that specifies that all the numeric predictors should be normalized.
- Bundle the model specification and recipe into a workflow, `mr_wflow`

```
BEGIN QUESTION
name: Question 4
manual: false
```

```
# Specify a multinomial regression via nnet
mr_spec <- multinom_reg(
  penalty = 1,
  engine = "nnet",
  mode = "classification"
)


# Create a recipe that specifies that predictors should be on the same scale
wine_recipe <- recipe(wine_variety ~ ., data = wine_train) %>%
  step_normalize(all_numeric_predictors())

# Bundle recipe and model specification into a workflow
mr_wflow <- workflow(preprocessor = wine_recipe, spec = mr_spec)

# Print out workflow
mr_wflow
```

```
## == Workflow ==================================================================
## Preprocessor: Recipe
## Model: multinom_reg()
##
## -- Preprocessor --------------------------------------------------------------
## 1 Recipe Step
##
## * step_normalize()
##
## -- Model ---------------------------------------------------------------------
## Multinomial Regression Model Specification (classification)
```

```
##
## Main Arguments:
##   penalty = 1
##
## Computational engine: nnet
```

```
. = " # BEGIN TEST CONFIG

success_message: Excellent! Your model specification is looking great!

failure_message: Let's have a look at this again. Ensure you have set your engine to **nnet** and the mo
" # END TEST CONFIG

## Test ##
test_that('the model specification is correct', {
  expect_equal(mr_spec$mode, "classification")
  expect_equal(mr_spec$engine, "nnet")


})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

success_message: Excellent! Your recipe is looking great!

failure_message: Let's have a look at this again. Ensure you have set the **wine_variety** variable as t
" # END TEST CONFIG

## Test ##
test_that('recipe specification is correct', {
  # expect_output(invisible(print(wine_recipe)), "Recipe\n\nInputs:\n\n\nOperations:\n\nCentering and s

  expect_equal(summary(wine_recipe) %>% filter(variable == "wine_variety") %>% pull(role), "outcome")
})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

success_message: Excellent! Your workflow is all set up!

failure_message: Let's have a look at this again. Perhaps we forgot to add something e.g add_model(...)'
" # END TEST CONFIG

## Test ##
test_that('workflow specification is correct', {
  expect_output(print(mr_wflow), "== Workflow =========================================================

})
```

```
## Test passed
```

Good job!

After a workflow has been *specified*, a model can be `trained` using the `fit()` function.

```
# Fit a workflow object
mr_wflow_fit <- mr_wflow %>%
  fit(data = wine_train)

# Print wf object
mr_wflow_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: multinom_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Call:
## nnet::multinom(formula = ..y ~ ., data = data, decay = ~1, trace = FALSE)
##
## Coefficients:
##   (Intercept)     alcohol malic_acid          ash alcalinity   magnesium
## 1  0.05907133 -1.35508237 -0.4155301 -0.782465012  0.7455790 -0.04196882
## 2 -0.45955408 -0.08191744  0.2386987 -0.001106485  0.3265549 -0.14266632
##       phenols  flavanoids nonflavanoids proanthocyanins color_intensity
## 1 -0.05577945 -0.05082982     0.3114386    -0.006529624      -1.0029004
## 2 -0.57773942 -0.82887210     0.2716521    -0.405656682       0.6359159
##          hue od280_315_of_diluted_wines   proline
## 1  0.3271264                 -0.3363225 -1.104231
## 2 -0.7331035                 -0.9215869 -0.484650
##
## Residual Deviance: 36.03699
## AIC: 92.03699
```

Evaluate model performance

Good job! We now have a trained workflow. The workflow print out shows the coefficients learned during training.

Question 5.

This allows us to use the model trained by this workflow to predict labels and their corresponding class probabilities for the test set.

In this section:

- Obtain a tibble containing the actual wine varieties of the test set, the hard class wine variety predictions and their corresponding probability predictions.

  ```
  BEGIN QUESTION
  name: Question 5
  manual: false
  ```

```r
# Obtain predictions on the test set
results <- mr_wflow_fit %>%
  augment(wine_test)

# Print the results
results %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 18
##    alcohol malic_acid   ash alcalinity magnesium phenols flavanoids
##      <dbl>      <dbl> <dbl>      <dbl>     <dbl>   <dbl>      <dbl>
##  1    14.4       1.95  2.5        16.8       113    3.85       3.49
##  2    13.2       2.59  2.87       21         118    2.8        2.69
##  3    13.9       1.35  2.27       16          98    2.98       3.15
##  4    13.6       1.81  2.7        17.2       112    2.85       2.91
##  5    13.8       1.57  2.62       20         115    2.95       3.4
##  6    14.2       1.59  2.48       16.5       108    3.3        3.93
##  7    14.1       1.63  2.28       16         126    3          3.17
##  8    13.4       1.77  2.62       16.1        93    2.85       2.94
##  9    13.7       1.83  2.36       17.2       104    2.42       2.69
## 10    13.5       1.8   2.65       19         110    2.35       2.53
## # ... with 11 more variables: nonflavanoids <dbl>, proanthocyanins <dbl>,
## #   color_intensity <dbl>, hue <dbl>, od280_315_of_diluted_wines <dbl>,
## #   proline <dbl>, wine_variety <fct>, .pred_class <fct>, .pred_0 <dbl>,
## #   .pred_1 <dbl>, .pred_2 <dbl>
```

```r
. = " # BEGIN TEST CONFIG

success_message: Fantastic! You have successfully used the trained model to make hard class and probabil

failure_message: Let's have a look at this again! Ensure you have used your trained model to make hard c
Hints: augment or predict + bind_cols functions.
" # END TEST CONFIG

## Test ##
test_that('results contain true values, hard class predictions and probabilities', {
  expect_equal(sum(c("wine_variety", ".pred_class", ".pred_0", ".pred_1", ".pred_2") %in% names(results)

})
```

```
## Test passed
```

Awesome! Time to evaluate how our model performed. Let's look at the confusion matrix for our model

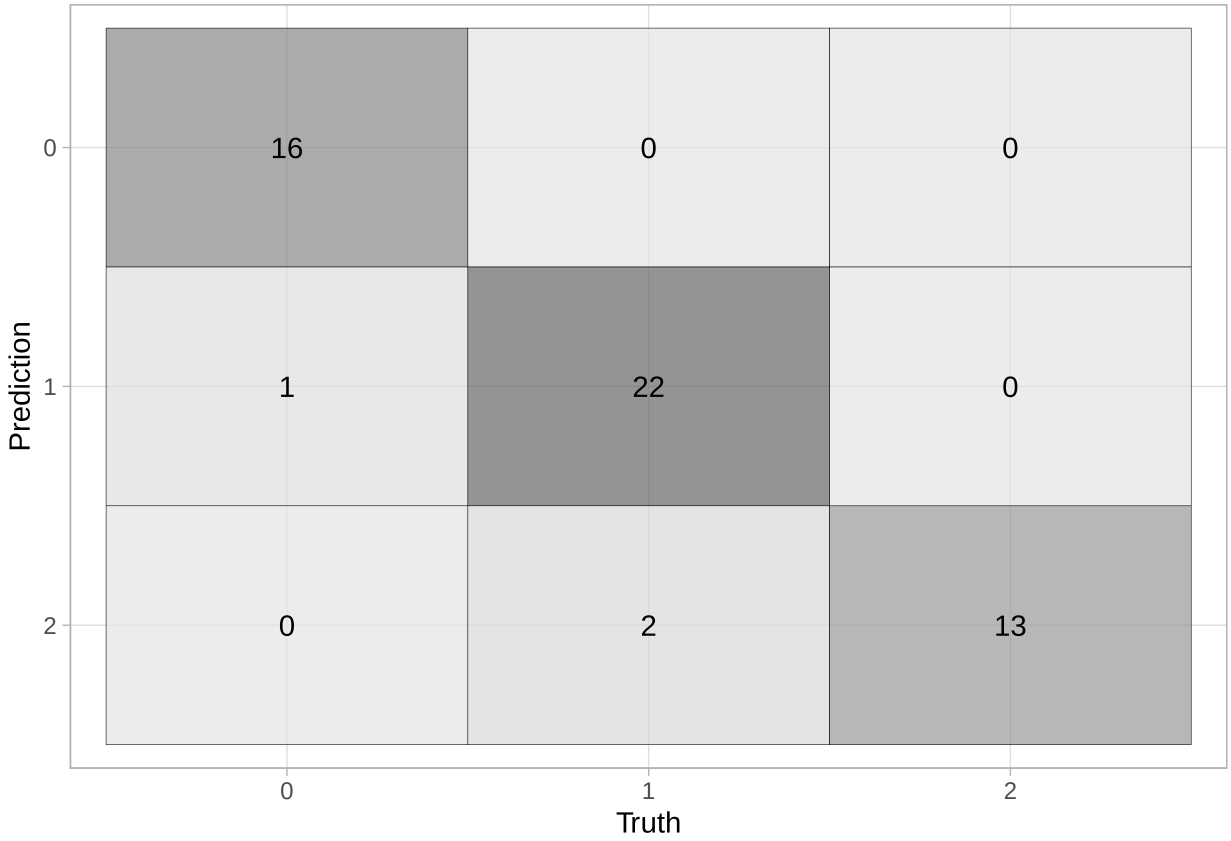
```r
# Confusion matrix
results %>%
  conf_mat(truth = wine_variety, estimate = .pred_class)
```

```
##           Truth
## Prediction  0  1  2
##          0 16  0  0
##          1  1 22  0
##          2  0  2 13
```

The confusion matrix shows the intersection of predicted and actual label values for each class - in simple terms, the diagonal intersections from top-left to bottom-right indicate the number of correct predictions.

When dealing with multiple classes, it's generally more intuitive to visualize this as a heat map, like this:

```
update_geom_defaults(geom = "tile", new = list(color = "black", alpha = 0.7))
# Visualize confusion matrix
results %>%
  conf_mat(wine_variety, .pred_class) %>%
  autoplot(type = "heatmap")
```



The darker squares in the confusion matrix plot indicate high numbers of cases, and you can hopefully see a diagonal line of darker squares indicating cases where the predicted and actual label are the same.

Question 6.

The confusion matrix is helpful since it gives rise to other metrics that can help us better evaluate the performance of a classification model. Let's go through some of them:

`Precision: TP/(TP + FP)` defined as the proportion of predicted positives that are actually positive.

`Recall: TP/(TP + FN)` defined as the proportion of positive results out of the number of samples which were actually positive.

`Accuracy: TP + TN/(TP + TN + FP + FN)` The percentage of labels predicted accurately for a sample.

In this section, using the `yardstick` package, obtain the following metrics:

- Accuracy

- Precision
- Recall

```
BEGIN QUESTION
name: Question 6
manual: false
```

```
# Statistical summaries for the confusion matrix
accuracy = accuracy(data = results, truth = wine_variety, estimate = .pred_class)

precision = precision(data = results, truth = wine_variety, estimate = .pred_class)

recall = recall(data = results, truth = wine_variety, estimate = .pred_class)

# Simpler way
conf_mat(data = results, truth = wine_variety, estimate = .pred_class) %>%
  summary() %>%
  filter(.metric %in% c("accuracy", "recall", "precision"))
```

```
## # A tibble: 3 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  multiclass     0.944
## 2 precision macro          0.941
## 3 recall    macro          0.953
```

```
. = " # BEGIN TEST CONFIG

" # END TEST CONFIG

## Test ##
test_that('accuracy is correct', {
  expect_equal(accuracy$.metric, "accuracy")
  expect_equal(accuracy$.estimator, "multiclass")
  expect_true(is.numeric(accuracy$.estimate))

})
```

```
## Test passed
```

```
. = " # BEGIN TEST CONFIG

" # END TEST CONFIG

## Test ##
test_that('precision is correct', {
  expect_equal(precision$.metric, "precision")
  expect_equal(precision$.estimator, "macro")
  expect_true(is.numeric(precision$.estimate))

})
```

```
## Test passed
```

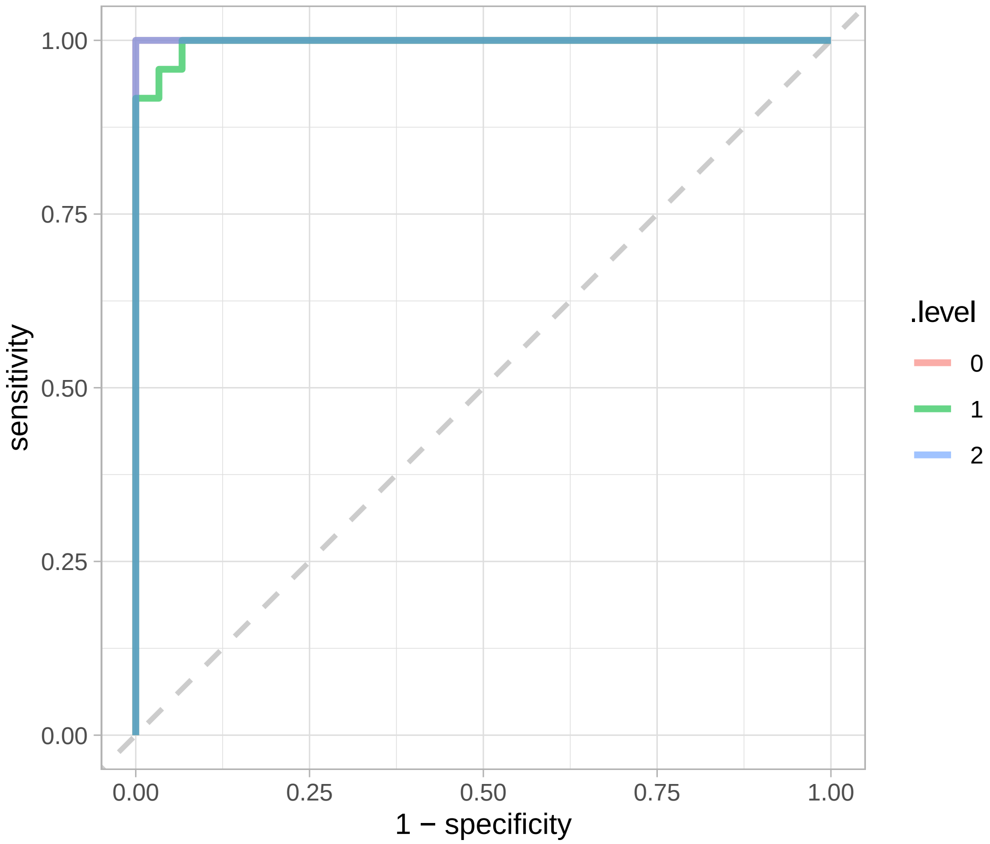
```
. = " # BEGIN TEST CONFIG

" # END TEST CONFIG

## Test ##
test_that('recall is correct', {
  expect_equal(recall$.metric, "recall")
  expect_equal(recall$.estimator, "macro")
  expect_true(is.numeric(recall$.estimate))

})
```

```
## Test passed
```

What do you think of these metrics?

Let's now evaluate the ROC metrics. In the case of a multiclass classification model, a single ROC curve showing true positive rate vs false positive rate is not possible. However, you can use the rates for each class in a One vs Rest (OVR) comparison to create a ROC chart for each class as below:

```
# Make a ROC_CURVE
results %>%
  roc_curve(wine_variety, c(.pred_0, .pred_1, .pred_2)) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, color = .level)) +
  geom_abline(lty = 2, color = "gray80", size = 0.9) +
  geom_path(show.legend = T, alpha = 0.6, size = 1.2) +
  coord_equal()
```



To quantify the ROC performance, you can calculate an aggregate area under the curve score that is averaged across all of the OVR curves.

Question7 In this section: - Quantify the ROC performance by calculating the aggregate area under the curve score that is averaged across all of the OVR curves.

```
BEGIN QUESTION
name: Question 7
manual: false
```

```
# Calculate ROC_AUC
auc <- results %>%
  roc_auc(wine_variety, c(.pred_0, .pred_1, .pred_2))

auc
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.999
```

```
. = " # BEGIN TEST CONFIG
success_message: Good job! You have successfully quantified the ROC performance.

failure_message: Almost there. Let's have a look at this again. Ensure you are calculating the roc_auc
" # END TEST CONFIG
```

```
## Test ##
test_that('recall is correct', {
  expect_equal(auc$.metric, "roc_auc")
  expect_equal(auc$.estimator, "hand_till")
  expect_true(is.numeric(auc$.estimate))

})
```

```
## Test passed
```

Overall, did the model do a great job in classifying the wine varieties? Be sure to try out a different classification algorithm.

Use the model with new data observation

When you're happy with your model's predictive performance, save it and then use it to predict classes for the following two new wine samples:

- $$13.72, 1.43, 2.5, 16.7, 108, 3.4, 3.67, 0.19, 2.04, 6.8, 0.89, 2.87, 1285$$
- $$12.37, 0.94, 1.36, 10.6, 88, 1.98, 0.57, 0.28, 0.42, 1.95, 1.05, 1.82, 520$$

```
library(here)
# Save trained workflow
saveRDS(mr_wflow_fit, "wine_mr_model.rds")
```

Now, we can load it whenever we need it, and use it to predict labels for new data. This is often called `scoring` or `inferencing`.

```
# Create a tibble for the new wine samples
new_wines <- c(13.72, 1.43, 2.5, 16.7, 108, 3.4, 3.67, 0.19, 2.04, 6.8, 0.89, 2.87, 1285) %>%
  rbind(c(12.37, 0.94, 1.36, 10.6, 88, 1.98, 0.57, 0.28, 0.42, 1.95, 1.05, 1.82, 520)) %>%
  as_tibble()
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```
names(new_wines) = names(wine_data)[1:13]

new_wines
```

```
## # A tibble: 2 x 13
##   alcohol malic_acid   ash alcalinity magnesium phenols flavanoids nonflavanoids
##     <dbl>      <dbl> <dbl>      <dbl>     <dbl>   <dbl>      <dbl>         <dbl>
## 1    13.7       1.43  2.5        16.7       108    3.4        3.67          0.19
## 2    12.4       0.94  1.36       10.6        88    1.98       0.57          0.28
## # ... with 5 more variables: proanthocyanins <dbl>, color_intensity <dbl>,
## #   hue <dbl>, od280_315_of_diluted_wines <dbl>, proline <dbl>
```

```r
# Load the model into the current R session
loaded_model <- readRDS("wine_mr_model.rds")

# Make predictions
predictions <- loaded_model %>%
  augment(new_data = new_wines)

predictions
```

```
## # A tibble: 2 x 17
##   alcohol malic_acid   ash alcalinity magnesium phenols flavanoids nonflavanoids
##     <dbl>      <dbl> <dbl>      <dbl>     <dbl>   <dbl>      <dbl>         <dbl>
## 1    13.7       1.43  2.5        16.7       108    3.4        3.67          0.19
## 2    12.4       0.94  1.36       10.6        88    1.98       0.57          0.28
## # ... with 9 more variables: proanthocyanins <dbl>, color_intensity <dbl>,
## #   hue <dbl>, od280_315_of_diluted_wines <dbl>, proline <dbl>,
## #   .pred_class <fct>, .pred_0 <dbl>, .pred_1 <dbl>, .pred_2 <dbl>
```

That's it for now. Congratulations on finishing this challenge notebook! The adventure does not stop there, you could try tuning that hyperparameter, or try out a new model entirely.

Happy Learning,

Eric, Gold Microsoft Learn Student Ambassador.