Iris ML classification

Ajay A

2024-10-21

Table of contents

# 1. Predicting the Species of Iris with Machine Learning

## 1.1 Introduction

This is the “Iris” dataset. Originally published at UCI Machine Learning Repository: Iris Data Set, this small dataset from 1936 is often used for testing out machine learning algorithms and visualizations (for example, Scatter Plot). Each row of the table represents an iris flower, including its species and dimensions of its botanical parts, sepal and petal, in centimeters.

Iris data set is used widely as an example in field of data sciences and widely available to both python and R users.

## 1.2 Importing Libraries

Let’s load the required packages

library(tidyverse)
library(patchwork)

## 1.3 Importing data

iris\_data <- read\_csv("F:/Data\_Sci/Internship Projects/Iris\_ML/Iris.csv")

Rows: 150 Columns: 5
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (1): Species
dbl (4): SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm

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.

iris\_data <- iris\_data %>% janitor::clean\_names() %>%
 mutate(species = str\_replace\_all(species,"Iris-", "")) %>%
 mutate(species = as.factor(species))

glimpse(iris\_data)

Rows: 150
Columns: 5
$ sepal\_length\_cm <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4,…
$ sepal\_width\_cm <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7,…
$ petal\_length\_cm <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5,…
$ petal\_width\_cm <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2,…
$ species <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa…

We have features like sepal\_length\_cm, sepal\_width\_cm, petal\_length\_cm and petal\_width\_cm and species necessary for the classification of the species.

First, check for NA in the data.

iris\_data %>% map(~sum(is.na(.)))

$sepal\_length\_cm
[1] 0

$sepal\_width\_cm
[1] 0

$petal\_length\_cm
[1] 0

$petal\_width\_cm
[1] 0

$species
[1] 0

There no NA which is really good for the data.

## 1.4 Analysisng data

Let’s visualize the data with above parameters

plot\_iris <- function(param){
 iris\_data %>% ggplot(aes(species, {{param}})) +
 geom\_boxplot(aes(color = species)) +
 theme(legend.position = "none")
}

(plot\_iris(sepal\_length\_cm) + plot\_iris(petal\_length\_cm))/
(plot\_iris(sepal\_width\_cm) + plot\_iris(petal\_width\_cm))

|  |
| --- |
| Figure 1: Difference in observed parameters between species |

There is quite a difference between the species in all parameters “setosa” < “versicolor” < “virginica” except for sepal\_width\_cm where “versicolor” < “virginica” < “setosa”

## 1.5 Building a model

Let’s start by loading the tidymodels package and splitting our data into training and testing sets.

library(tidymodels)
set.seed(2024)

iris\_split <- initial\_split(iris\_data, prop = 0.8)

iris\_train <- training(iris\_split)
iris\_test <- testing(iris\_split)

Data is not large enough to build a model so creating resamples of the data to evaluate the model

set.seed(2025)

iris\_boot <- bootstraps(iris\_train, times = 5)

iris\_boot

# Bootstrap sampling
# A tibble: 5 × 2
 splits id
 <list> <chr>
1 <split [120/45]> Bootstrap1
2 <split [120/40]> Bootstrap2
3 <split [120/38]> Bootstrap3
4 <split [120/39]> Bootstrap4
5 <split [120/46]> Bootstrap5

Let’s build 2 models and check which is better for the data.

### 1.5.1 Random Forest Model

# random forest model

rf\_spec <- rand\_forest() %>%
 set\_mode("classification") %>%
 set\_engine("ranger")

rf\_spec

Random Forest Model Specification (classification)

Computational engine: ranger

### 1.5.2 **K - nearest neighbors** model

knn\_spec <- nearest\_neighbor(neighbors = 5) %>%
 set\_mode("classification") %>%
 set\_engine("kknn")

knn\_spec

K-Nearest Neighbor Model Specification (classification)

Main Arguments:
 neighbors = 5

Computational engine: kknn

### 1.5.3 Setting workflow()

Next let’s start putting together a tidymodels workflow(), a helper object to help manage modeling pipelines with pieces that fit together like Lego blocks. Notice that there is no model yet: Model: None

iris\_wf <- workflow() %>%
 add\_formula(species ~ .)

iris\_wf

══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Formula
Model: None

── Preprocessor ────────────────────────────────────────────────────────────────
species ~ .

### 1.5.4 Fitting the model

Now we can add a model and fit the model to each of the resamples. First, we can fit the randomforest model

rf\_rs <- iris\_wf %>%
 add\_model(rf\_spec) %>%
 fit\_resamples(
 resamples = iris\_boot,
 control = control\_resamples(save\_pred = TRUE)
 )

rf\_rs

# Resampling results
# Bootstrap sampling
# A tibble: 5 × 5
 splits id .metrics .notes .predictions
 <list> <chr> <list> <list> <list>
1 <split [120/45]> Bootstrap1 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
2 <split [120/40]> Bootstrap2 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
3 <split [120/38]> Bootstrap3 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
4 <split [120/39]> Bootstrap4 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
5 <split [120/46]> Bootstrap5 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>

Now we can add a model and fit the model to each of the resamples. First, we can fit the knn model

knn\_rs <- iris\_wf %>%
 add\_model(knn\_spec) %>%
 fit\_resamples(
 resamples = iris\_boot,
 control = control\_resamples(save\_pred = TRUE)
 )

knn\_rs

# Resampling results
# Bootstrap sampling
# A tibble: 5 × 5
 splits id .metrics .notes .predictions
 <list> <chr> <list> <list> <list>
1 <split [120/45]> Bootstrap1 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
2 <split [120/40]> Bootstrap2 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
3 <split [120/38]> Bootstrap3 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
4 <split [120/39]> Bootstrap4 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>
5 <split [120/46]> Bootstrap5 <tibble [3 × 4]> <tibble [0 × 3]> <tibble>

## 1.6 Evaluating the model

### 1.6.1 Evaluating Random Forest model

collect\_metrics function collect the necessary parameters for evaluation

collect\_metrics(rf\_rs)

# A tibble: 3 × 6
 .metric .estimator mean n std\_err .config
 <chr> <chr> <dbl> <int> <dbl> <chr>
1 accuracy multiclass 0.947 5 0.0144 Preprocessor1\_Model1
2 brier\_class multiclass 0.0332 5 0.0106 Preprocessor1\_Model1
3 roc\_auc hand\_till 0.998 5 0.000846 Preprocessor1\_Model1

collect\_predictions(rf\_rs) %>% glimpse()

Rows: 208
Columns: 8
$ .pred\_class <fct> versicolor, virginica, virginica, setosa, versicolor,…
$ .pred\_setosa <dbl> 0.0020, 0.0000, 0.0000, 1.0000, 0.0000, 1.0000, 0.000…
$ .pred\_versicolor <dbl> 0.93323254, 0.00000000, 0.33701825, 0.00000000, 0.998…
$ .pred\_virginica <dbl> 0.0647674603, 1.0000000000, 0.6629817460, 0.000000000…
$ id <chr> "Bootstrap1", "Bootstrap1", "Bootstrap1", "Bootstrap1…
$ .row <int> 1, 6, 7, 9, 18, 19, 21, 22, 28, 30, 34, 35, 39, 40, 4…
$ species <fct> versicolor, virginica, virginica, setosa, versicolor,…
$ .config <chr> "Preprocessor1\_Model1", "Preprocessor1\_Model1", "Prep…

### 1.6.2 Evaluating K-nearest neighbor model

collect\_metrics(knn\_rs)

# A tibble: 3 × 6
 .metric .estimator mean n std\_err .config
 <chr> <chr> <dbl> <int> <dbl> <chr>
1 accuracy multiclass 0.922 5 0.00981 Preprocessor1\_Model1
2 brier\_class multiclass 0.0660 5 0.00783 Preprocessor1\_Model1
3 roc\_auc hand\_till 0.979 5 0.00687 Preprocessor1\_Model1

collect\_predictions(rf\_rs) %>% glimpse()

Rows: 208
Columns: 8
$ .pred\_class <fct> versicolor, virginica, virginica, setosa, versicolor,…
$ .pred\_setosa <dbl> 0.0020, 0.0000, 0.0000, 1.0000, 0.0000, 1.0000, 0.000…
$ .pred\_versicolor <dbl> 0.93323254, 0.00000000, 0.33701825, 0.00000000, 0.998…
$ .pred\_virginica <dbl> 0.0647674603, 1.0000000000, 0.6629817460, 0.000000000…
$ id <chr> "Bootstrap1", "Bootstrap1", "Bootstrap1", "Bootstrap1…
$ .row <int> 1, 6, 7, 9, 18, 19, 21, 22, 28, 30, 34, 35, 39, 40, 4…
$ species <fct> versicolor, virginica, virginica, setosa, versicolor,…
$ .config <chr> "Preprocessor1\_Model1", "Preprocessor1\_Model1", "Prep…

As we can see that random\_forest model has higher accuracy than knn model

Confusion matrix lets us know how accurate the model is predicting the values

rf\_rs %>% conf\_mat\_resampled()

# A tibble: 9 × 3
 Prediction Truth Freq
 <fct> <fct> <dbl>
1 setosa setosa 13.4
2 setosa versicolor 0
3 setosa virginica 0
4 versicolor setosa 0
5 versicolor versicolor 13.2
6 versicolor virginica 1.8
7 virginica setosa 0
8 virginica versicolor 0.4
9 virginica virginica 12.8

Now for the roc curve which shows us how accurate a model is for different species in the data.

rf\_rs %>%
 collect\_predictions() %>%
 roc\_curve(truth = species, .pred\_setosa, .pred\_versicolor,
 .pred\_virginica) %>%
 ggplot(aes(1 - specificity, sensitivity, color = .level)) +
 geom\_abline(lty = 2, color = "gray80", linewidth = 1.5) +
 geom\_path(show.legend = TRUE, alpha = 0.6, linewidth = 1.2) +
 coord\_equal() +
 labs(title = "ROC Curve for Random Forest Classification",
 color = "Class")



ROC curve for Random Forest Model

When we compare the same to the “KNN” model we can see the difference.

knn\_rs %>%
 collect\_predictions() %>%
 roc\_curve(truth = species, .pred\_setosa, .pred\_versicolor,
 .pred\_virginica) %>%
 ggplot(aes(1 - specificity, sensitivity, color = .level)) +
 geom\_abline(lty = 2, color = "gray80", linewidth = 1.5) +
 geom\_path(show.legend = TRUE, alpha = 0.6, linewidth = 1.2) +
 coord\_equal() +
 labs(title = "ROC Curve for KNN Classification",
 color = "Class")



ROC curve for KNN Model

The “1 - Specificity” drops for the “KNN” model when compared to the “Random Forest” model, so we will use the “Random Forest” model to do predictions.

iris\_final <- iris\_wf %>%
 add\_model(rf\_spec) %>%
 last\_fit(iris\_split)

iris\_final %>% collect\_metrics()

# A tibble: 3 × 4
 .metric .estimator .estimate .config
 <chr> <chr> <dbl> <chr>
1 accuracy multiclass 0.933 Preprocessor1\_Model1
2 roc\_auc hand\_till 1 Preprocessor1\_Model1
3 brier\_class multiclass 0.0534 Preprocessor1\_Model1

### 1.6.3 Predicting Outcomes

Based on the “iris\_final” model we can predict the species based on the other parameters.

# Create a new data frame for the measurements
new\_data <- tibble(
 sepal\_length\_cm = 4.6,
 sepal\_width\_cm = 3.8,
 petal\_length\_cm = 1.4,
 petal\_width\_cm = 0.2
)

# Extract the workflow from the last\_fit result
workflow\_fit <- iris\_final %>% extract\_workflow()

# Make predictions using the new data
predictions <- predict(workflow\_fit, new\_data)

predictions

# A tibble: 1 × 1
 .pred\_class
 <fct>
1 setosa