

Hands-on: Build Your k-NN R Package

Overview

In this 45 min session you will:

1. Create a new R package to wrap your custom k-NN code.
 2. Organize R scripts and C++ code in the package structure.
 3. Document functions using **Roxygen2**.
 4. Install and test your package.
 5. Run predictions and visualize results from the package.
-

Setup

1. Create a new R package project in RStudio:

```
1 # Replace "knnPackage" with your chosen package name
2 usethis::create_package("knnPackage")
```

2. Copy the prepared files into your package:

```
1 knnPackage/
2   R/          # R scripts with roxygen2 (knn_s3.R, knn_s3_formula.R)
3   src/
```

- R/knn_s3.R: S3 class for k-NN
- R/knn_s3_formula.R: Formula interface and predict methods
- src/knn_pred.cpp: Rcpp implementation for fast predictions

All three files already contain Roxygen2 comments, so your functions will be automatically exported.

1. Document your package

Generate documentation for all exported functions:

```
1 devtools::document("knnPackage")
```

Check that .Rd files are created in `man/`.

2. Install and load your package

```
1 # Install locally
2 devtools::install("knnPackage")
3
4 # Load package
5 library(knnPackage)
```

3. Train/test split

Use `mtcars` to test your package:

```
1 set.seed(42)
2 n <- nrow(mtcars)
3 train_idx <- sample(seq_len(n), size = round(0.7 * n))
4 train <- mtcars[train_idx, ]
5 test  <- mtcars[-train_idx, ]
```

4. Fit models using package functions

```

1 # Fit k-NN models with k=5 and k=10
2 mod5 <- knn_s3(mpg ~ disp + hp + wt, train, k = 5)
3 mod10 <- knn_s3(mpg ~ disp + hp + wt, train, k = 10)
4
5 # Predict using Rcpp
6 pred5 <- predict(mod5, newdata = test, method = "cpp")
7 pred10 <- predict(mod10, newdata = test, method = "cpp")

```

5. Evaluate performance

```

1 rmse <- function(y, yhat) sqrt(mean((y - yhat)^2))
2 mae <- function(y, yhat) mean(abs(y - yhat))
3
4 perf <- data.frame(
5   Model = c("k=5", "k=10"),
6   RMSE = c(rmse(test$mpg, pred5), rmse(test$mpg, pred10)),
7   MAE = c(mae(test$mpg, pred5), mae(test$mpg, pred10))
8 )
9 print(perf)

```

6. Visualize predictions

```

1 library(tidyr)
2 library(ggplot2)
3
4 test$pred5 <- pred5
5 test$pred10 <- pred10
6
7 test_long <- test %>%
8   pivot_longer(cols = c(pred5, pred10),
9               names_to = "Model", values_to = "Prediction")
10
11 ggplot(test_long, aes(x = mpg, y = Prediction, color = Model)) +

```

```
12 geom_point(size = 3, alpha = 0.6) +
13 geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
14 labs(title = "Predicted vs Actual MPG (from Package)",
15       x = "Actual MPG", y = "Predicted MPG") +
16 scale_color_manual(values = c("pred5" = "blue", "pred10" = "red"),
17                    labels = c("k = 5", "k = 10")) +
18 theme_minimal()
```

Discussion

- Verify that the package functions work as expected.
- Inspect the help pages generated from Roxygen2.
- Discuss the benefits of wrapping your code into a package:
 - Reusability
 - Documentation
 - Easy sharing

Next steps

- Add more utility functions (e.g., cross-validation) to your package.
- Include unit tests with testthat for your main functions.
- Share your package via GitHub for collaborative development.