Making optimal use of ChatGPT and other chatbots for data science

Ronald (Ryy) Glenn Thomas

2025-01-02

Table of contents

1 Introduction

"ChatGPT stunned the world on its launch in November 2022. Powered by a large language model (LLM) and trained on much of the text published on the Internet, the artificial intelligence (AI) chatbot, created by OpenAI in San Francisco, California, makes the latest advances in natural-language processing broadly accessible by providing a dialogue-based interface capable of answering complex questions, composing sophisticated essays and generating source code. One obvious question was: how could this tool improve science?"

Nature: Chatbots in science: What can ChatGPT do for you?

Start with a couple of examples of having ## Example work up of a regression analysis for the iris data set.

1. Prompt: "I'm doing a logistic regression analysis and want to K-fold cross-validation"
2. Prompt: "I want to do the analysis in R"
3. Prompt: "I want to use an ROC curve to visualize the results"
4. Prompt: "list all the R analysis code in one block"
5. Prompt: "list all my R analysis related prompts for just now in one block"
6. Prompt: "should I split my data set into train and test parts if I'm using k-fold cross-validation?"
7. Prompt: "extend the code for the R logistic regression analysis to split the iris data into test and training sets before applying k-fold cross-validation"
8. Prompt: "add a 95% confidence interval calculation to the AUC annotation in the ROC plot"
9. Prompt: "do the same analysis but use lasso regression instead of logistic regression"

2 R code

```r
install.packages("caret")  # Install if not already installed
install.packages("pROC")   # Install if not already installed
install.packages("glmnet") # Install if not already installed
library(caret)
library(pROC)
library(glmnet)

# Prepare binary dataset
data(iris)
iris_binary <- iris[iris$Species != "setosa", ]
iris_binary$Species <- factor(iris_binary$Species)

# Split the data into training and test sets
set.seed(42)
trainIndex <- createDataPartition(iris_binary$Species, p = 0.8, list = FALSE)
train_data <- iris_binary[trainIndex, ]
test_data <- iris_binary[-trainIndex, ]

# Convert the data into matrix format as required by glmnet
x_train <- model.matrix(Species ~ ., train_data)[,-1]
y_train <- train_data$Species

x_test <- model.matrix(Species ~ ., test_data)[,-1]
y_test <- test_data$Species

# Set up cross-validation for Lasso regression on the training data
train_control <- trainControl(method = "cv", number = 10, classProbs = TRUE, summaryFunction =

# Define the lasso model using glmnet (alpha=1 for Lasso)
lasso_model <- train(x = x_train, y = y_train, method = "glmnet", trControl = train_control, tu

# Make predictions on the test set
test_predictions <- predict(lasso_model, newdata = x_test, type = "prob")[,2]

# Plot ROC curve for the test set
roc_curve_test <- roc(y_test, test_predictions, levels = rev(levels(y_test)))
plot(roc_curve_test, col = "blue", lwd = 2, main = "ROC Curve for Lasso Regression on Test Data
v
# Calculate AUC and its 95% confidence interval
auc_value <- auc(roc_curve_test)
```

```r
ci <- ci.auc(roc_curve_test, conf.level = 0.95)

# Add AUC with 95% confidence interval to the plot
legend_text <- paste("AUC =", round(auc_value, 2), "\n95% CI:", round(ci[1], 2), "-", round(ci
legend("bottomright", legend = legend_text, col = "blue", lwd = 2)
```

3 Complete R Code for Lasso Regression with LOOCV on the Entire Dataset

```r
# Prepare binary dataset
data(iris)
iris_binary <- iris[iris$Species != "setosa", ]
iris_binary$Species <- factor(iris_binary$Species)

# Convert the data into matrix format as required by glmnet
x <- model.matrix(Species ~ ., iris_binary)[,-1]
y <- iris_binary$Species

# Set up LOOCV for Lasso regression on the entire dataset
train_control <- trainControl(method = "LOOCV", classProbs = TRUE, summaryFunction = twoClassSu

# Define the lasso model using glmnet (alpha=1 for Lasso)
lasso_model <- train(x = x, y = y, method = "glmnet", trControl = train_control, tuneGrid = exp

# Make predictions using LOOCV
predictions <- predict(lasso_model, newdata = x, type = "prob")[,2]

# Plot ROC curve for the entire dataset
roc_curve <- roc(y, predictions, levels = rev(levels(y)))
plot(roc_curve, col = "blue", lwd = 2, main = "ROC Curve for Lasso Regression with LOOCV")

# Calculate AUC and its 95% confidence interval
auc_value <- auc(roc_curve)
ci <- ci.auc(roc_curve, conf.level = 0.95)

# Add AUC with 95% confidence interval to the plot
```

```r
legend_text <- paste("AUC =", round(auc_value, 2), "\n95% CI:", round(ci[1], 2), "-", round(ci
legend("bottomright", legend = legend_text, col = "blue", lwd = 2)
```

4 today's date is 2024-11-10

Trying to get the vim plugin for openai to work. having trouble vim can't find the python module OpenAI

5 useful prompts:

Have chatGPT provide the README.md file as a download. "provide the README.md file as a download."

6 Guide to Building, Submitting, and Managing the `zzlongplot` R Package

6.1 Step 1: Setting Up the Package Structure

1. **Create a New Package Directory**:
    - Use `usethis` to create a package directory:

      ```r
      usethis::create_package("path/to/zzlongplot")
      ```

    - This sets up the necessary directory structure with folders like `R/` and files like `DESCRIPTION`.
2. **Add the Core Script**:
    - Place the `zzlongplot.R` file in the `R/` directory.
3. **Set Up the `DESCRIPTION` File**:
    - Edit the `DESCRIPTION` file to include metadata about the package. Use `usethis::use_description()` to create and fill this file:

```
usethis::use_description(fields = list(
  Title = "Flexible Longitudinal Plotting in R",
  Description = "Provides tools for generating observed and change plots in longitudi
  Version = "0.1.0",
  Author = "Your Name [aut, cre]",
  Maintainer = "Your Name <your_email@example.com>",
  License = "MIT",
  Encoding = "UTF-8"
))
```

4. **Add Dependencies**:

   - List package dependencies in the `DESCRIPTION` file under `Imports`. For example:

```
Imports:
  dplyr,
  ggplot2,
  patchwork
```

6.2 Step 2: Document the Package

1. **Add Roxygen2 Comments**:

   - Ensure all functions in `zzlongplot.R` have Roxygen2 comments for documentation.

2. **Generate Documentation**:

   - Run:

```
devtools::document()
```

   - This creates help files in the `man/` directory and updates the `NAMESPACE` file.

3. **Create a Vignette**:

   - Add the vignette to introduce the package:

```r
usethis::use_vignette("Introduction_to_zzlongplot")
```

- Place the provided `zzlongplot-vignette.Rmd` file in the `vignettes/` directory and build it:

```r
devtools::build_vignettes()
```

6.3 Step 3: Test the Package

1. **Add Unit Tests**:
   - Use `usethis` to set up a testing framework:

   ```r
   usethis::use_testthat()
   ```

   - Place the `test-zzlongplot.R` file in `tests/testthat/`.
2. **Run Tests**:
   - Run all tests:

   ```r
   devtools::test()
   ```

6.4 Step 4: Check the Package

1. **Build and Check**:
   - Build the package:

   ```r
   devtools::build()
   ```

   - Check the package for CRAN compliance:

   ```r
   devtools::check()
   ```

2. **Fix Issues**:
   - Address any warnings or errors reported by `devtools::check()`.

6.5 Step 5: Submit to CRAN

1. **Prepare for Submission**:
    - Ensure the package passes `R CMD check` with no warnings, errors, or notes.
    - Compress the package into a `.tar.gz` file using:

    ```
    devtools::build()
    ```

2. **Submit to CRAN**:
    - Go to the CRAN submission page.
    - Upload the `.tar.gz` file and fill out the required metadata.

3. **Respond to Feedback**:
    - CRAN maintainers might request changes. Address them promptly and resubmit if needed.

6.6 Step 6: Set Up a GitHub Repository

1. **Initialize a Git Repository**:
    - In the package directory, run:

    ```
    git init
    git add .
    git commit -m "Initial commit"
    ```

2. **Create a Repository on GitHub**:
    - Use the GitHub website or the `gh` CLI tool:

    ```
    gh repo create yourusername/zzlongplot --public --source=.
    ```

3. **Push the Code**:
    - Push the code to GitHub:

```
git branch -M main
git push -u origin main
```

6.7 Step 7: Manage the Development Repository

1. **Add Version Control**:

   - Use Git for version control. For example, create a branch for new features:

   ```
   git checkout -b feature-new-plot
   ```

2. **Tag Releases**:

   - Tag versions for releases:

   ```
   git tag -a v0.1.0 -m "First release"
   git push origin v0.1.0
   ```

3. **Add Continuous Integration**:

   - Set up GitHub Actions for testing:

   ```
   usethis::use_github_action_check_standard()
   ```

4. **Publish Development Versions**:

   - Use GitHub to manage development versions and issues.

6.8 Step 8: Maintain the Package

1. **Address Issues**:

   - Monitor and address issues reported by users.

2. **Update the Package**:

- For updates, increment the version number in `DESCRIPTION` and tag the new version.