Midterm II Study Guide and Review

Deepak Bastola

2024-05-20

Midterm II Study Guide

Format: In Class with open-ended questions.

One-sided Cheat-sheet allowed (A4 paper) and a basic calculator allowed.

- You may use a calculator
- You are not permitted to use a laptop or classroom computer.

Topics

- The exam covers iterations, functionals, web scraping, shiny kNN, linear regression and logistic regression (through Mon. 05/20)
- You will be tested on your conceptual understanding of the tools and algorithms, shiny function logic, the accuracy metrics, and the associated construction of the workflow in R that we have discussed in the class. I will not make you write extremely complicated code from scratch, but be prepared to write small chunks of code. Additional ways I could assess your understanding of R include (but are not limited to):
  - Identifying the error in written code.
  - Putting lines of code in order to complete a specified task.
  - Describing the output resulting from a code/code-chunk.

Your name:

Questions

Q1

Given below are the monthly deaths from bronchitis, emphysema and asthma in the UK from 1974 to 1979.

```r
knitr::kable(mydata)
```

year	jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	dec
1974	3035	1721	2933	1607	2787	1489	3102	1498	2815	1529	3084	1461
1975	2552	1524	2889	1545	3891	1300	2294	1361	3137	1366	2605	1354
1976	2704	1596	2938	1396	3179	1356	2385	1346	2679	1357	2573	1333
1977	2554	2074	2497	1787	2011	1653	2444	1564	1969	1570	2143	1492
1978	2014	2199	1870	2076	1636	2013	1748	1640	1870	1535	1693	1781
1979	1655	2512	1726	2837	1580	2823	1554	2293	1633	2491	1504	1915

a. Describe what is returned by the code below, including the type of R object produced, the length or dimension of the object, and the information contained in the object.

```
map_dbl(mydata %>% select(-1), mean) %>% mean()
## [1] 2056.625
```

The code calculates the average of the mean of all 12 columns (months) of `mydata`, excluding the first column (Year). The result is a single numeric value, representing the average of monthly mean deaths from bronchitis, emphysema, and asthma.

b. Describe what is returned by the code below, including the type of R object produced, the length or dimension of the object, and the information contained in the object.

```
ratio_fun <- function(x) quantile(x, probs= c(0.25, 0.5, 0.75))
map_dfc(mydata %>% select(-1), ratio_fun)
## # A tibble: 3 x 12
##     jan   feb   mar   apr   may   jun   jul   aug   sep   oct   nov   dec
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 2148. 1627. 2027. 1560. 1730. 1389. 1884. 1395. 1895. 1407. 1806. 1381.
## 2 2553  1898. 2693  1697  2399  1571  2340. 1531  2324  1532  2358  1476.
## 3 2666. 2168. 2922  2004. 3081  1923  2429. 1621  2781  1561. 2597  1709.
```

This code creates a data frame where each column represents a different month (`dfc` in `map_dfc` stands for `data frame - columns`). The result is a data frame with 3 rows (quartiles) and 12 columns (months).

c. Describe what is returned by the code below, including the type of R object produced, the length or dimension of the object, and the information contained in the object.

```
lapply(mydata %>% select(-1), ratio_fun) %>%
  unlist()
## jan.25% jan.50% jan.75% feb.25% feb.50% feb.75% mar.25% mar.50% mar.75% apr.25%
## 2148.50 2553.00 2666.50 1627.25 1897.50 2167.75 2026.75 2693.00 2922.00 1560.50
## apr.50% apr.75% may.25% may.50% may.75% jun.25% jun.50% jun.75% jul.25% jul.50%
## 1697.00 2003.75 1729.75 2399.00 3081.00 1389.25 1571.00 1923.00 1884.50 2339.50
## jul.75% aug.25% aug.50% aug.75% sep.25% sep.50% sep.75% oct.25% oct.50% oct.75%
## 2429.25 1395.25 1531.00 1621.00 1894.75 2324.00 2781.00 1406.75 1532.00 1561.25
## nov.25% nov.50% nov.75% dec.25% dec.50% dec.75%
## 1805.50 2358.00 2597.00 1380.75 1476.50 1708.75
```

The code calculates the quartiles for each of the 12 columns (months) in `mydata`, excluding the first column (Year), then combines the results into a single vector. The output is a numeric vector of length 36 (3 quartiles for each of 12 months).

Q2 Shiny UI and Server Logic

a. Describe the relationship between `ui` and `server` in a Shiny application. How do they interact?

The ui (User Interface) component of a Shiny app defines the layout and appearance of the app, which includes elements like sliders, buttons, and plots that the user interacts with. The server function contains the logic that responds to user inputs, performing calculations or manipulating data based on these inputs. It communicates with the UI via reactive expressions, which automatically update outputs when inputs change.

b. Given the code snippet below, identify and explain any errors that would prevent the app from displaying a histogram of the generated data.

```r
library(shiny)
ui <- fluidPage(
  titlePanel("Data Histogram"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs", "Number of observations:", min = 0, max = 1000, value = 500)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)

server <- function(input, output) {
  output$distPlot <- renderPlot({
    data <- rnorm(input$obs, input = 100)
    hist(data)
  })
}

shinyApp(ui = ui, server = server)
```

The error in the code snippet is in the rnorm function call: input = 100 is not a valid argument. It should likely be mean = 100 if the intention is to set the mean of the normal distribution to 100. The corrected line should be: `data <- rnorm(input$obs, mean = 100)`

c. Consider a Shiny app that allows users to select a dataset from a dropdown menu and displays a summary of that dataset. Identify potential problems in this hypothetical Shiny code snippet and suggest improvements.

```r
library(shiny)
ui <- fluidPage(
  selectInput("dataset", "Choose a dataset:", choices = c("mtcars", "iris")),
  tableOutput("summaryTable")
)

server <- function(input, output) {
  data <- reactive({
    switch(input$dataset,
           "mtcars" = mtcars,
           "iris" = iris)
  })

  output$summaryTable <- renderTable({
    summary(data())
  })
}

shinyApp(ui = ui, server = server)
```

The code provided generally works, but it's important to ensure that the reactive expression efficiently handles the switching between datasets only when the input changes. A potential problem is if the datasets are large, switching could cause performance issues due to reactivity. To improve efficiency, consider using `req()` to ensure that the dataset is only processed when valid:

```r
data <- reactive({
  req(input$dataset)  # Ensures input$dataset is not NULL
  switch(input$dataset,
         "mtcars" = mtcars,
         "iris" = iris)
})
```

d. Given a Shiny application that uses actionButton to trigger calculations, explain the purpose of this input type and describe a scenario where it might be useful.

```r
ui <- fluidPage(
  actionButton("calc", "Calculate"),
  numericInput("num", "Enter a number:", 10),
  textOutput("result")
)

server <- function(input, output) {
  observeEvent(input$calc, {
    result <- sqrt(input$num)
    output$result <- renderText({
      paste("The square root is:", result)
    })
  })
}

shinyApp(ui = ui, server = server)
```

The `actionButton` in Shiny does not directly modify any input or output until it is explicitly used in server logic. It's useful for triggering reactive events only when a user clicks it, rather than automatically reacting to input changes. This is particularly useful for operations that require heavy computations or need to be controlled manually, like data submissions or initiating a calculation:

Q3 Web scraping

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Sample Web Page</title>
</head>
<body>
    <h1>Welcome to My Website</h1>
    <p>This is a paragraph of text on the main page.</p>
    <a href="/about">About Us</a>
    <a href="/contact">Contact Us</a>
</body>
</html>
```

Assume the HTML above is served from http://mywebpage.com.

a. Write the R code to extract and print the text inside the `<h1>` tag using `rvest` and/or `polite`.

```
library(rvest)
html_code <- read_html('http://mywebpage.com')
h1_text <- html_code %>% html_node('h1') %>% html_text()
h1_text
```

b. Write the R code to extract the href attributes of all links in the document.

```
links <- html_code %>% html_nodes('a') %>% html_attr('href')
links
```

c. Given the base URL `http://mywebpage.com`, write the R code to convert the relative URLs found in the href attributes into absolute URLs.

```
absolute_links <- url_absolute(links, 'http://example.com')
absolute_links
```

d. Write the R code to extract and print the text of each link (tag).

```
link_texts <- html_code %>% html_nodes('a') %>% html_text()
link_texts
```

Q4 : K-nearest neighbor

Consider the `Smarket` dataset, which provides daily percentage returns for the S&P 500 stock index from 2001 to 2005. Your task is to fit a K-nearest neighbor model to predict the direction of stock returns.

```
set.seed(1234)

data_Smarket <- as_tibble(Smarket)
split <- initial_split(data_Smarket, strata = Direction, prop = 4/5)
Smarket_train <- training(split)
Smarket_test <- testing(split)

# glimpses of data
glimpse(Smarket_train)
## Rows: 999
## Columns: 9
## $ Year      <dbl> 2001, 2001, 2001, 2001, 2001, 2001, 2001, 2001, 2001, 2001, ~
## $ Lag1      <dbl> 1.032, 1.392, -0.498, 0.546, 0.359, -0.623, 1.183, -0.865, 0~
## $ Lag2      <dbl> 0.959, 0.213, 0.287, -0.562, -1.747, -0.841, -1.334, 1.183, ~
## $ Lag3      <dbl> 0.381, 0.614, 1.303, 0.701, 0.546, -0.151, -0.623, -1.334, -~
## $ Lag4      <dbl> -0.192, -0.623, 0.027, 0.680, -0.562, 0.359, -0.841, -0.623,~
## $ Lag5      <dbl> -2.624, 1.032, -0.403, -0.189, 0.701, -1.747, -0.151, -0.841~
## $ Volume    <dbl> 1.4112, 1.4450, 1.2580, 1.1188, 1.0130, 1.1072, 1.0391, 1.07~
## $ Today     <dbl> -0.623, -0.403, -0.189, -1.747, -0.151, -1.334, -0.865, -0.2~
## $ Direction <fct> Down, Down, Down, Down, Down, Down, Down, Down, Down, Down, ~
glimpse(Smarket_test)
## Rows: 251
## Columns: 9
## $ Year      <dbl> 2001, 2001, 2001, 2001, 2001, 2001, 2001, 2001, 2001, 2001, ~
## $ Lag1      <dbl> 0.381, 0.614, 0.213, 0.287, 0.701, -0.562, -0.151, -0.841, -~
## $ Lag2      <dbl> -0.192, -0.623, 0.614, 1.303, 0.680, 0.701, 0.359, -0.151, -~
## $ Lag3      <dbl> -2.624, 1.032, -0.623, 0.027, -0.189, 0.680, -1.747, 0.359, ~
## $ Lag4      <dbl> -1.055, 0.959, 1.032, -0.403, -0.498, -0.189, 0.546, -1.747,~
## $ Lag5      <dbl> 5.010, 0.381, 0.959, 1.392, 0.287, -0.498, -0.562, 0.546, 0.~
## $ Volume    <dbl> 1.19130, 1.20570, 1.34910, 1.30900, 1.14980, 1.29530, 1.0596~
## $ Today     <dbl> 0.959, 0.213, 1.392, -0.498, -0.562, 0.546, -0.841, -0.623, ~
## $ Direction <fct> Up, Up, Up, Down, Down, Up, Down, Down, Up, Up, Up, Up, Up, ~
```

a. . The following code splits the Smarket dataset into a training set and a test set. Explain the purpose of splitting the data and what stratification accomplishes in this context. Splitting the dataset into training and test sets is essential for assessing the model's performance on unseen data, simulating how it would perform in a real-world scenario. Stratification ensures that both training and test datasets have a similar proportion of categories in the Direction variable. This is crucial for maintaining a balanced dataset, especially when the target variable categories are imbalanced, to prevent the model from being biased towards the majority class.

```r
Smarket_recipe <- recipe(Direction ~ Lag1 + Lag2 + Lag3 + Year + Volume,
                         data = Smarket_train) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

Smarket_knn_spec <- nearest_neighbor(mode = "classification",
                             engine = "kknn",
                             weight_func = "rectangular",
                             neighbors = 5)

Smarket_workflow <- workflow() %>%
  add_recipe(Smarket_recipe) %>%
  add_model(Smarket_knn_spec)

Smarket_fit <- fit(Smarket_workflow, data = Smarket_train)

test_features <- Smarket_test %>% select(Direction, Lag1, Lag2, Lag3, Year, Volume)
nn1_pred3 <- predict(Smarket_fit, test_features, type = "raw")
Smarket_results <- Smarket_test %>%
  select(Direction) %>%
  bind_cols(predicted = nn1_pred3) %>% mutate(Direction = as.factor(Direction))
```

b. The following trained model is used to produce a data-frame of the actual and predicted `Direction` in the test dataset. Call this data-frame `Smarket_results`. What information does `Smarket_results` contain? What is the dimension of this dataset? Explain. Smarket_results contains the actual and predicted Direction of stock returns. The dimension of this dataset would be the same as the number of rows in the test dataset, which is 251 rows, each corresponding to an observation. Each row includes the actual Direction as observed in the dataset and the Direction predicted by the K-nearest neighbor model.

c. Calculate the sensitivity, specificity, accuracy, and positive predictive value of the classifier based on the confusion matrix below. Discuss the interpretation of these metrics in the context of this problem.

- Sensitivity (True Positive Rate): Measures the proportion of actual positives correctly identified as such.
- Specificity (True Negative Rate): Measures the proportion of actual negatives correctly identified.
- Accuracy: The ratio of correctly predicted observations to the total observations.
- Positive Predictive Value (PPV): Probability that subjects with a positive screening test truly have the disease.

```
conf_mat(Smarket_results, truth = Direction, estimate = predicted)
##           Truth
## Prediction Down Up
##       Down   50 58
##       Up     71 72
custom_metrics <- metric_set(accuracy, sensitivity, specificity, ppv)
custom_metrics(Smarket_results,
               truth = Direction,
               estimate = predicted)
## # A tibble: 4 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 accuracy    binary         0.486
## 2 sensitivity binary         0.413
## 3 specificity binary         0.554
## 4 ppv         binary         0.463
```

These metrics help in understanding the effectiveness of the classifier in predicting stock directions, particularly how well the classifier avoids false positives and false negatives.

Q4 Miscellaneous

(a) Differentiate between supervised and unsupervised learning by providing examples of situations where each would be most appropriate. Supervised learning involves training a model on a labeled dataset, where the outcomes are known. It's appropriate for predictive tasks such as regression and classification, like predicting housing prices based on features. Unsupervised learning involves finding patterns or structures in data where the outcomes are not known, suitable for clustering and association tasks like market basket analysis or customer segmentation.

(b) Discuss the significance of feature scaling in the K-NN algorithm and illustrate its impact with a hypothetical example. Feature scaling is critical in K-NN because the algorithm uses distance calculations to determine the nearest neighbors. If one feature has a much larger scale than others, it will dominate the distance calculation, potentially skewing results. For example, in a dataset with income (thousands) and age (years), income would disproportionately influence neighbor selection without scaling.

(c) (True/False) Recall is a more relevant metric than precision in scenarios where False Negatives are more detrimental than False Positives. Explain your answer. True. Recall is more relevant than precision in scenarios where the cost of missing a positive instance (false negative) is higher than incorrectly labeling a negative instance as positive (false positive). For instance, in medical diagnosis, missing a disease (false negative) could be life-threatening, making recall a critical measure.

(d) Explain the rationale behind data preprocessing in the KNN algorithm, discussing how the algorithm's performance might be influenced by unprocessed data. Preprocessing in K-NN is essential because the algorithm's performance heavily depends on the distance between examples. Without preprocessing, such as normalizing or standardizing data, variables on larger scales can disproportionately influence the outcome, leading to inaccurate classifications.

(e) (Multiple Choice) Among the following values for K in K-NN, which is most likely to cause underfitting?

1. 30
2. 5
3. 1

Among the given choices, 1) 30 is most likely to cause underfitting in a K-NN model. A large K value can make the model too generalized, smoothing out the predictions too much and potentially ignoring finer details in the data.

(f) Logistic regression is a machine learning algorithm that is typically used to predict the probability of what kind of variable? Explain the reasoning behind your choice.

(A) categorical independent variable
(B) categorical dependent variable.
(C) numerical dependent variable.
(D) numerical independent variable.

B) categorical dependent variable. Logistic regression is used to predict the probability of a binary outcome based on one or more predictor variables. The model outputs probabilities that are then mapped to categories, making it ideal for binary classification tasks like determining whether an email is spam or not spam.

(g) For a K-NN classification model, how would you expect the model's performance to change as K increases? Discuss considering both bias-variance trade-off and the model's complexity. As K increases in K-NN, the model's complexity decreases, leading to higher bias and lower variance. Initially, increasing K might reduce noise in the predictions, but excessively large K values can cause the model to underfit, missing relevant patterns in data. Thus, there's a trade-off to find an optimal K that balances bias and variance effectively.

(h) In logistic regression, which link function is commonly used?

A) Logit
B) Probit
C) Identity
D) Log

The correct answer is A) Logit. In logistic regression, the logit function, also known as the log of odds, is commonly used as the link function. It models the probability of the dependent binary variable as a function of the independent variables.

(i) Which method of data splitting ensures that the training and test sets have approximately the same percentage of samples of each class?

A) Random split
B) Stratified split
C) Sequential split
D) Clustered split

The correct answer is B) Stratified split. This method of data splitting ensures that both the training and test sets have approximately the same percentage of samples of each class. Stratified splitting is particularly useful when dealing with imbalanced datasets, as it helps maintain the class distribution across different subsets of the data.