Assignment 9 solution

EMATM0061: Statistical Computing and Empirical Methods, TB1, 2022

Introduction

This is the 9th assignment for Statistical Computing and Empirical Methods (Unit EMATM0061) on the MSc in Data Science & MSc in Financial Technology with Data Science. This assignment is mainly based on Lectures 22, 23 and 24 (see the Blackboard).

The submission deadline for this assignment is 23:59, 5 December 2022. Note that this assignment will not count towards your final grade. However, it is recommended that you try to answer the questions to gain a better understanding of the concepts.

Create an R Markdown for the assignment

It is a good practice to use R Markdown to organize your code and results.

If you are considering submitting your solutions, please generate a PDF file. For example, you can choose the "PDF" option when creating the R Markdown file (note that this option may require Tex to be installed on your computer), or use R Markdown to output an HTML and print it as a PDF file in a browser, or use your own way of creating a PDF file that contains your solutions.

Only a PDF file will be accepted in the submission of this assignment. To submit the assignment, please visit the "Assignment" tab on the Blackboard page, where you downloaded the assignment.

Load packages

Some of the questions in this assignment require the tidyverse package. If it hasn't been installed on your computer, please use `install.packages()` to install them first.

To road the tidyverse package:

```
library(tidyverse)
```

1. Basic concepts in classification

In lecture 24, we introduced some concepts in classification.

(Q1) Write down your explanation of each of the following concepts. Give an example where appropriate.

1. A classification rule
2. A learning algorithm
3. Training data
4. Feature vector
5. Label
6. Expected test error
7. Train error
8. The train test split

Answer

1. A classification rule

A classification rule, also known as a classifier, is a mapping $\phi : \mathcal{X} \to \mathcal{Y}$ where X is a space of feature vectors and Y is a set of discrete class labels, typically finite in number. As an example, consider a classifier which maps images of cats and dogs, represented as a vector, into a single label corresponding to whether the image is of a dog or a cat.

2. A learning algorithm

In the context of classification, a learning algorithm is an algorithm which takes as input a set of training data and outputs a classification rule. This classification rule may then be applied to previously unseen data. Examples include linear discriminant analysis and logistic regression.

3. Training data

Training data is a sequence of ordered pairs of the form $((X_1, Y_1), \cdots, (X_n, Y_n))$ where X_i is a feature vector and Y_i is an associated class label. For example, when trying to learn a cat/dog image classifier, X_i corresponds to an image of a dog or cat and Y_i is the label - "cat" or "dog".

4. Feature vector

A feature vector X is a vector containing one or more variables (also known as features) which we shall use to predict the class label, in the context of a classification rule. For example, in the context of the cat/dog image classifier, the feature vector is a vector corresponding to a grey-scale image where each element corresponds to the numerical value of a particular pixel. As another example, in the case of penguin classification, the feature vector is a vector containing several morphological properties such as the bill length, the flipper length, the weight etc.

5. Label

The label Y is a unique identifier that signifies that the corresponding feature vector X is associated with an item belonging to a particular class. For example Y could be 1 when associated with an image X of a dog and 0 when associated with an image X of a cat. The set of labels is in one-to-one correspondence with the number of classes eg. if there are three classes then there must be three labels.

6. Expected test error

The test error is the average number of mistakes a classification rule makes on unseen data. More precisely, suppose we have a classification rule $\phi : \mathcal{X} \to \mathcal{Y}$ then given a distribution P which describes the distribution over random pairs (X, Y) where X is a feature vector and Y is a label, the test error is

$$\mathcal{R}(\phi) = \mathbb{P}_{(X,Y)}[\phi(X) \neq Y].$$

This is typically estimated by the average error on a particular test data set.

7. Train error

The train error is the average number of mistakes made by a classifier on the training data. More precisely, suppose we have set of training data $((X_1, Y_1), \cdots, (X_n, Y_n))$ where X_i and a classifier $\phi : \mathcal{X} \to \mathcal{Y}$, the train error is given by

$$\hat{\mathcal{R}}(\phi) = \frac{1}{n} \sum_{i=1}^{n} [\phi(X) \neq Y].$$

8. The train test split

The train test split refers to a split of your data set into two pieces - containing two groups of examples: The training data and the testing data. The training data is used as input into the learning algorithm. Based on the training data (and not the test data), the learning algorithm generates a classification rule. The test data plays no role in the learning of the classification rule. Instead, the test data is used to assess the classification rule's performance on previously unseen data, not used within the training process. Hence, we compute the number of mistakes on the test data, which gives rise to an estimate for the expected test data.

2. A chi-squared test of population variance

Suppose we have an i.i.d. sample $X_1, \cdots, X_n \sim \mathcal{N}(\mu, \sigma^2)$ and a conjectured value for the population variance σ_0^2. We wish to test the null hypothesis that σ_0^2.

(Q1)

Implement a function called `chi_square_test_one_sample_var` which takes as input a sample called "`sample`" and a null value for the variance called `sigma_square_null`.

Answer

```
chi_square_test_one_sample_var <- function(sample, sigma_square_null){
  sample <- sample[!is.nan(sample)] # remove any missing values
  n <- length(sample) # sample length
  #. compute test statistic
  chi_squared_statistic <- (n-1)*var(sample)/sigma_square_null
  # compute p-valu e
  p_value <- 2*min(pchisq(chi_squared_statistic, df=n-1),
                   1-pchisq(chi_squared_statistic, df=n-1))
  return (p_value)
}
```

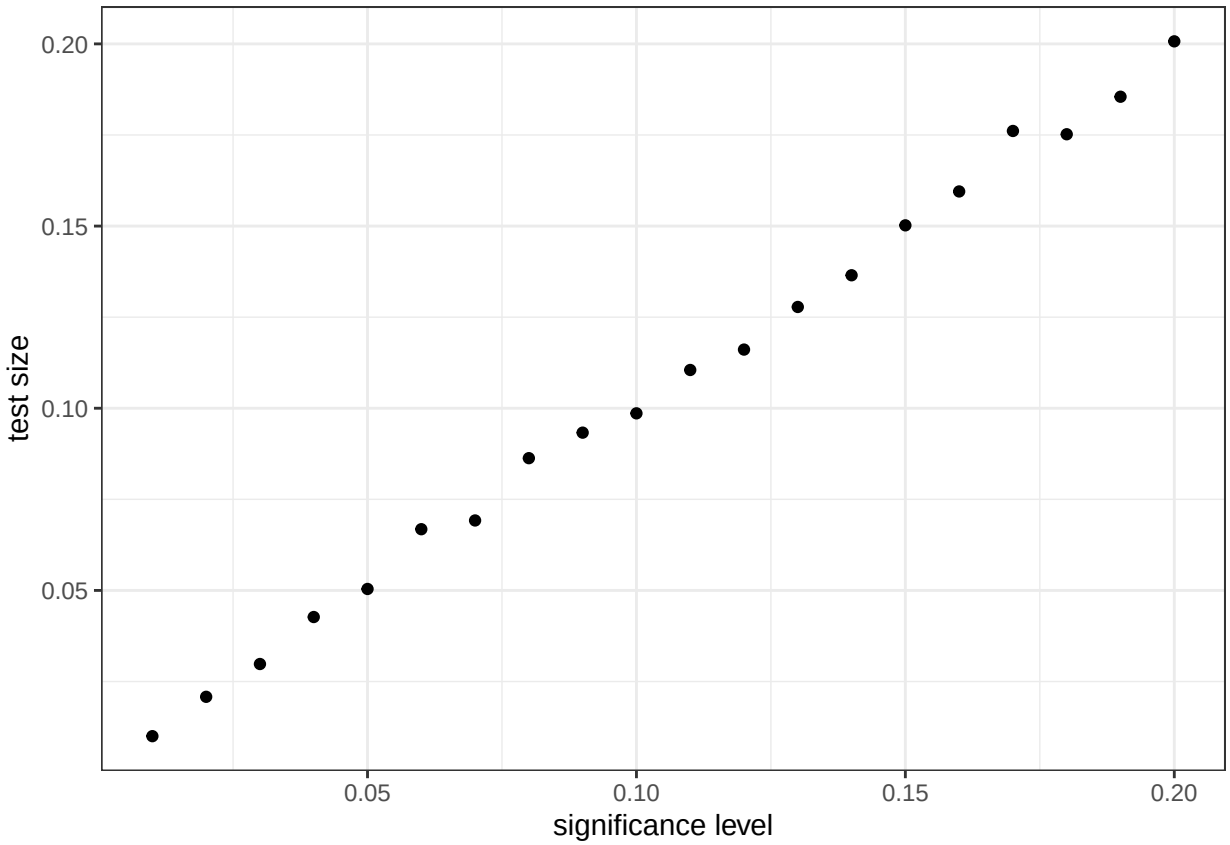
(Q2)

Conduct a simulation study to see how the size of the test varies as a function of the significance level. You can consider a sample size of 100, $\mu = 1$, $\sigma^2 = 4$.

Answer

```
trials <- seq(10000)
sample_size <- 100
significance_levels <- seq(0.01, 0.2, 0.01)

mu_0 <- 1
sigma_0 <- 2

set.seed(0)

df_simulated_var_test <- crossing(trials=trials,
                                  significance_levels=significance_levels) %>%
  mutate(sample=map(trials,~rnorm(sample_size, mean=mu_0, sd=sigma_0)) ) %>%
  mutate(p_value = map_dbl(sample, ~chi_square_test_one_sample_var(.x, sigma_0^2) ) ) %>%
  mutate(reject=(p_value<significance_levels))


df_test_size <- df_simulated_var_test %>%
  group_by(significance_levels) %>%
  summarise(test_size=mean(reject))
```

```
df_test_size %>% ggplot( aes(x=significance_levels, y=test_size) ) + geom_point() +
  theme_bw() + xlab('significance level') + ylab('test size')
```



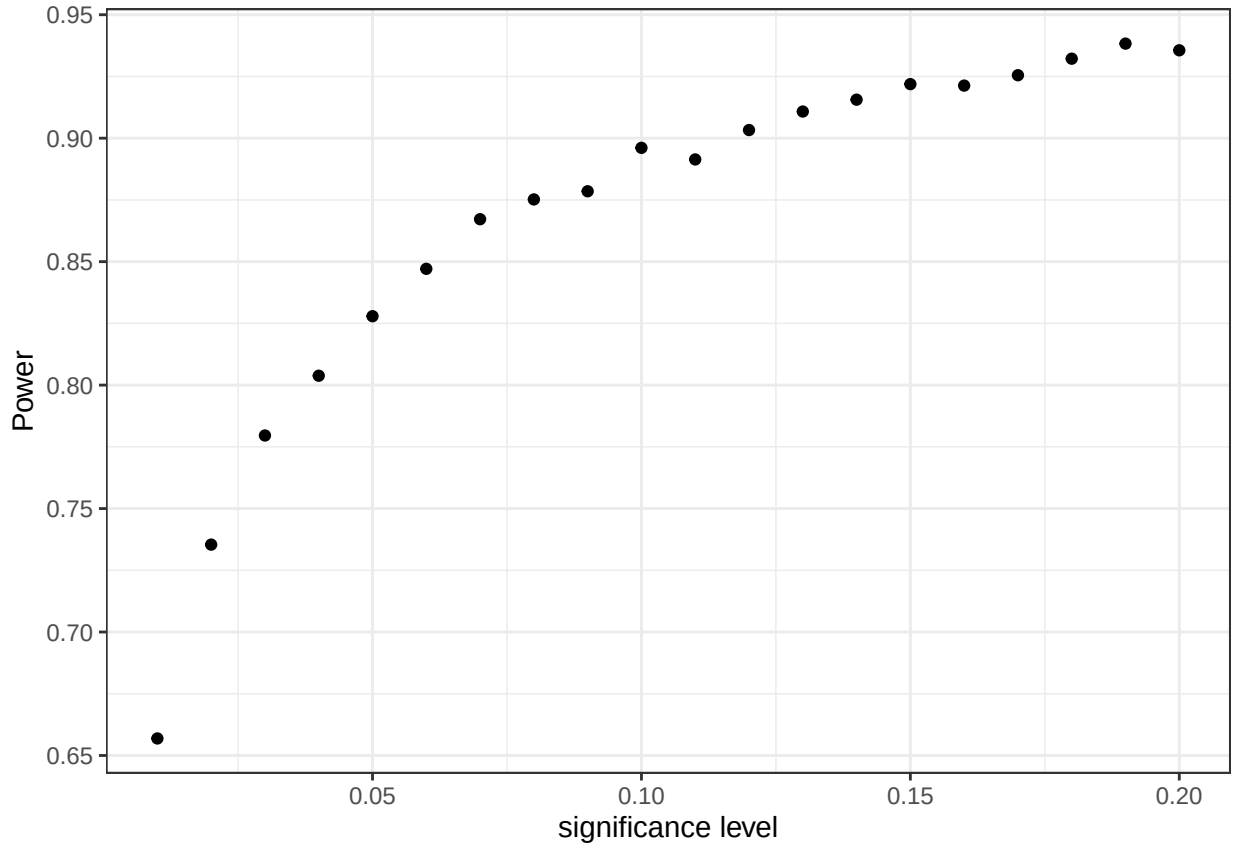
(Q4)

Conduct a simulation study to see how the statistical power of the test varies as a function of the significance level. You can consider a sample size of 100, $\mu = 1$, $\sigma^2 = 6$ and $\sigma_0^2 = 4$.

Answer

```
trials <- seq(10000)
sample_size <- 100
significance_levels <- seq(0.01, 0.2, 0.01)

mu_0 <- 1
sigma_0 <- 2
sigma <- sqrt(6)

set.seed(0)

df_simulated_var_test <- crossing(trials=trials,
                                  significance_levels=significance_levels) %>%
  mutate(sample=map(trials,~rnorm(sample_size, mean=mu_0, sd=sigma)) ) %>%
  mutate(p_value = map_dbl(sample, ~chi_square_test_one_sample_var(.x, sigma_0^2) ) ) %>%
  mutate(reject=(p_value<significance_levels))


df_power <- df_simulated_var_test %>%
  group_by(significance_levels) %>%
  summarise(power=mean(reject))
```

```
df_power %>% ggplot( aes(x=significance_levels, y=power) ) + geom_point() +
  theme_bw() + xlab('significance level') + ylab('Power')
```



(Q5)

Load the "Palmer penguins" library and extract a vector called "`bill_adelie`" consisting of the bill lengths of the Adelie penguins belonging to the Adelie species.

Suppose we model the sequence of bill lengths as a sample of independent and identically distributed Gaussian random variables $X_1, \cdots, X_n \sim \mathcal{N}(\mu, \sigma^2)$ with a population mean μ and population standard deviation σ.

Now apply your function "`chi_square_test_one_sample_var`" to test the null hypothesis that the population standard deviation is 3 mm at a significance level of $\alpha = 0.1$.

Answer

```
library(palmerpenguins)
bill_adelie_df <- penguins %>% filter(species=='Adelie') %>%
  select(bill_length_mm) %>% drop_na()

bill_adelie <- pull(bill_adelie_df)
# you can also use: bill_adelie <- bill_adelie_df$bill_length_mm

chi_square_test_one_sample_var(bill_adelie, 3^2)
```

```
## [1] 0.05196711
```

Therefore, given significance level 0.1, we should reject the null hypothesis.

3. The train test split

Suppose you want to build a classifier to predict whether a hawk belongs to either the "Sharp-shinned" or the "Cooper's" species of hawks. The feature vector will be a four-dimensional row vector containing the weight, and the lengths of the wing, the tail and the hallux. The labels will be binary: 1 if the hawk is "Sharp-shinned" and 0 if the hawk belongs to "Cooper's" species.

(Q1)

Begin by loading the "Hawks" data frame from the "Stat2Data" library. Now extract a subset of the data frame called "`hawks_total`" with five columns - "`Weight`", "`Wing`", "`Hallux`", "`Tail`" and "`Species`". The data frame should only include rows corresponding to hawks from either the "Sharp-shinned" (SS) or the "Cooper's" (CH) species, and not the "Red-tailed" (RT) species. Convert the Species column to a binary variable with a 1 if the hawk belongs to the sharp-shinned species and 0 if the hawk belongs to Cooper's species. Finally, remove any rows with missing values from one of the relevant columns.

Answer

```
library(Stat2Data)
data(Hawks)

hawks_total <- Hawks %>% select( Weight, Wing, Hallux, Tail, Species) %>%
  filter(Species=='SS' | Species =='CH') %>% drop_na() %>%
  mutate(Species=as.numeric(Species=='SS'))
```

(Q2)

Now implement a train test split for your "`hawks_total`" data frame. You should use 60% of your data within your training data and 40% in your test data. You should create a data frame consisting of training data called "`hawks_train`" and a data frame consisting of test data called "`hawks_test`". Display the number of rows in each data frame.

Answer

```
num_total <- hawks_total %>% nrow() # number of penguin data
num_train <- floor(num_total*0.6) # number of train examples
num_test <- num_total-num_train # number of test samples
set.seed(123) # set random seed for reproducibility

test_inds <- sample(seq(num_total),num_test) # random sample of test indicies
train_inds <- setdiff(seq(num_total),test_inds) # training data indicies

hawks_train <- hawks_total %>% filter(row_number() %in% train_inds) # train data
hawks_test <- hawks_total %>% filter(row_number() %in% test_inds) # test data
hawks_train %>% nrow()
```

```
## [1] 194
```

```
hawks_test%>%nrow()
```

```
## [1] 130
```

(Q3)

Next extract a data frame called "`hawks_train_x`" from your training data (from "`hawks_train`") containing the feature vectors and no labels. In addition, extract a vector called "`hawks_train_y`" consisting of labels from your training data. Similarly, create data frames called "`hawks_test_x`" and "`hawks_test_y`" corresponding to the feature vectors and labels within the test set, respectively.

Answer

```
hawks_train_x <- hawks_train %>% select(-Species) # train feature vectors
hawks_train_y <- hawks_train %>% pull(Species) # train labels
hawks_test_x <- hawks_test %>% select(-Species) # test feature vectors
hawks_test_y <- hawks_test %>% pull(Species) # test labels
```

(Q4)

Now let's consider a very simple (and not very effective) classifier which entirely ignores the feature vectors. Instead the classifier simply predicts a single fixed value $\hat{y} \in \{0, 1\}$. Hence, your classifier is of the form $\phi_{\hat{y}}(x) \equiv y$ for all $x \in \mathbb{R}^4$. Begin by choosing a value $\hat{y} \in \{0, 1\}$ based on your training data - choose the value which minimises the training error.

Answer

```
# only two possible phi in this case: one for y_hat=0, the other for y_hat=1
train_error_phi_0 <- mean(abs(hawks_train_y-0))
train_error_phi_1 <- mean(abs(hawks_train_y-1))

# finding y-hat with minimum training error
if(train_error_phi_0<train_error_phi_1){
  y_hat<-0
  }else{
    y_hat<-1
  }

# alternatively, you can also compute y_hat by (why?):
# y_hat<-as.numeric(mean(hawks_train_y)>=0.5)

y_hat
```

```
## [1] 1
```

(Q5)

Next compute the train and test error of $\phi_{\hat{y}}$

In general, $\phi_{\hat{y}}$ performs poorly, as it does not use any information of the feature vector. However, in the example, the train error and test error seems relatively low (much less than 50%). Try to explain why the errors are relatively low. In which cases we might have a error of $\phi_{\hat{y}}$ close to 50%?

Answer

```
train_error_simple <- mean(abs(y_hat-hawks_train_y)) # train error
test_error_simple <- mean(abs(y_hat-hawks_test_y)) # test error
train_error_simple
```

```
## [1] 0.2371134
```

```
test_error_simple
```

```
## [1] 0.1769231
```

4. Multivariate distributions and parameter estimation

Suppose that we have a sample of red-tailed hawks and we want to investigate the distribution of several features (Wing, Weight and Tail) of red-tailed hawks. We model the Wing, Weight and Tail with a multivariate Gaussian distribution.

(Q1)

Again, load the "Hawks" data frame from the "Stat2Data" library. Now extract a subset of the data frame called "`hawks_rt`" that contains only the rows corresponding to hawks from the "Red-tailed" (RT) species and three columns - "`Wing`", "`Weight`", "`Tail`", "`Tail`". Remove any rows of "`hawks_rt`" with missing values from one of the relevant columns.

Answer

```
hawks_rt <- Hawks %>% filter(Species=='RT') %>% select(Wing, Weight, Tail)
```

(Q2)

Now, lets model the three features "`Wing`", "`Weight`" and "`Tail`" with a multivariate Gaussian distribution. Suppose that your data frame `hawks_rt` consists of a i.i.d. sample $X_1, \cdots, X_n \sim \mathcal{N}(\mu, \Sigma)$. Here we model the three features "`Wing`", "`Weight`" and "`Tail`" with a multivariate Gaussian distribution with population mean μ and population covariance matrix Σ. Compute the minimum variance unbiased estimates (MVUE) of the μ and Σ.

Answer

```
mu_mvue <- map_dbl(hawks_rt, ~mean(.x, na.rm=TRUE))
mu_mvue
```

```
##      Wing    Weight      Tail
##  383.3036 1094.4301  222.1490
```

```
Sigma_mvue <- cov(hawks_rt, use="complete.obs")
Sigma_mvue
```

```
##             Wing     Weight     Tail
## Wing    970.2672  1983.0489 148.8996
## Weight 1983.0489 35800.5187 705.7409
## Tail    148.8996   705.7409 211.4762
```

5. Investigate hypotheis testing where the Gaussian model assumptions are violated

Many hypothesis testing approaches assume that the underlying distribution is Gaussian or approximately Gaussian. For example, when we apply one-sample t-test to test a value of the population mean for a population, we often assume that the distribution of our data is Gaussian or approximately Gaussian. In this question, we will carry out a simulation study to investigate the behaviour of the hypothesis testing when this assumption is violated. Let's focus on one sample t-test here.

Consider a continuous random variable X with the following probability density function

$$f(x) = \begin{cases} 2(x-a) & \text{if } x \in (a, a+1) \\ 0 & \text{othewise.} \end{cases}$$

where a is a parameter. We will perform a one-sample t-test in a setting where the sample are i.i.d. copies of X (which is non-Gaussian).

(Q1)

Let the population mean of X be denoted by μ. Express a as a function of μ.

Then write down a formula for the cumulative distribution F_X of X and write down a formula for the quantile function F_X^{-1} of X.

Answer

$$\mu := \mathbb{E}(X) = \int_a^{a+1} x \cdot 2(x-a)dx = \frac{2}{3} + a$$

So $a = \mu - \frac{2}{3}$

The cumulative distribution function is

$$F_X(x) = \begin{cases} (x-a)^2 & \text{if } x \in (a, a+1) \\ 1 & \text{if } x \geq a+1 \\ 0 & \text{if } x \leq a. \end{cases}$$

The quantile function is

$$F_X^{-1}(p) = \begin{cases} -\infty & \text{if } p = 0 \\ \sqrt{p} + a & \text{if } p \in (0, 1] \end{cases}$$

(Q2)

Write an R function called `generate_sample_X` for generating samples of X. The function should take as input two arguments the sample size called `sample_size` and the population mean called `mu` and output a vector. The output vector is computed as follows:

1. First, generate a sample call `sample_U` (the size of `sample_U` should be `sample_size`) from the uniform distribution between 0 and 1. You can use the function `runif`.
2. Second, apply the function F_X^{-1} to each element of `sample_U` to get a new vector (and this is the output of `generate_sample_X`).

Answer

```
generate_sample_X <- function(sample_size, mu){
  a <- mu-2/3
  sample_U <- runif(sample_size)
  sample_X <- sqrt(sample_U) + a
}
```

(Q3) (*optional)

Explain why the function `generate_sample_X` generate a sample of X.

Answer

The output of the function `generate_sample_X` is a sample of $Y := F_X^{-1}(U)$ where U is a random variable following a uniform distribution on $[0, 1]$.

For any $p \in (0, 1)$, we have $F_X(F_X^{-1}(p)) = p$, hence for any α,

$$F_Y(\alpha) := \mathbb{P}(Y \leq \alpha) = \mathbb{P}\big(F_X^{-1}(U) \leq \alpha\big) = \mathbb{P}(U \leq F_X(\alpha)) = F_X(\alpha)$$

Therefore, Y has the same distribution as X, and `generate_sample_X` generates a sample of X.

(Q4)

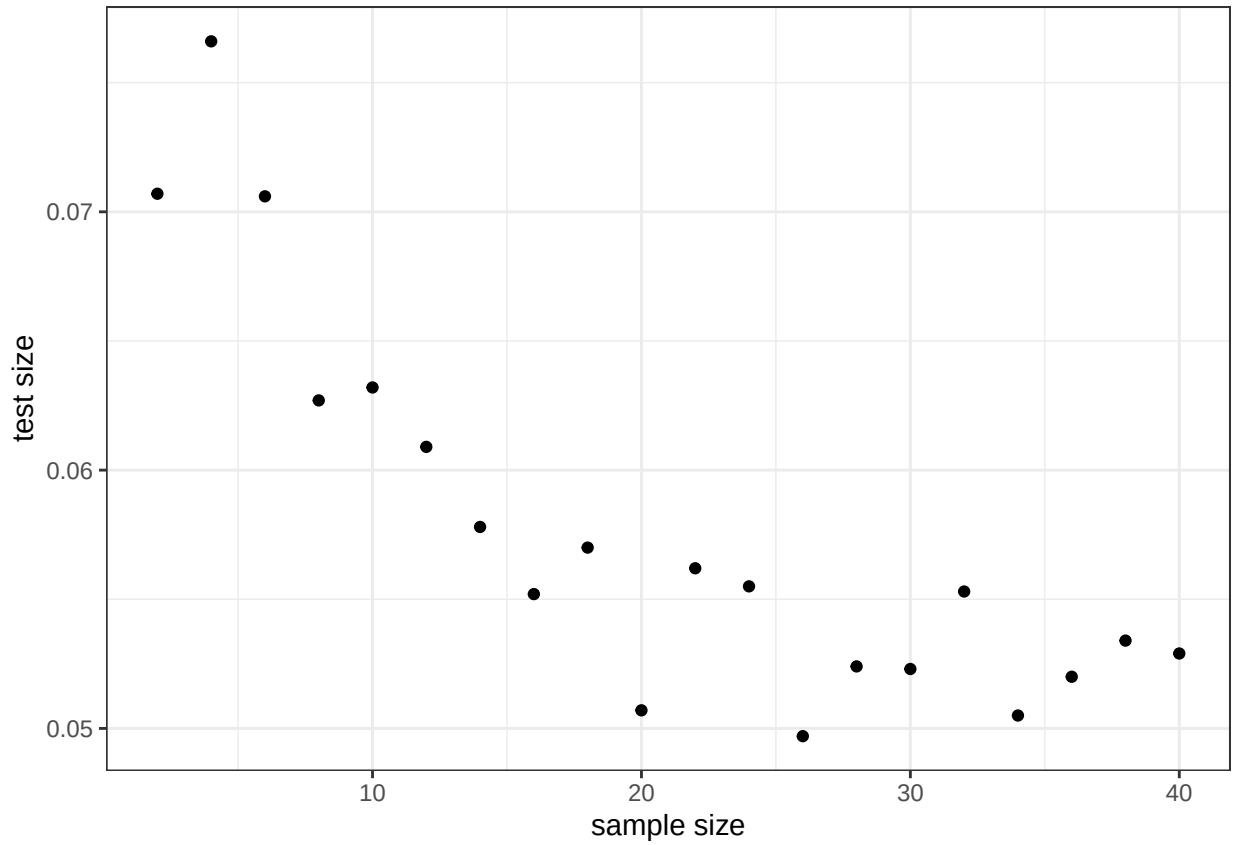
Suppose that, given a sample of X, we want to test if the population mean μ is equal to a given number μ_0.

Conduct a simulation study to investigate the test size (the average number of rejecting the null hypothesis when $\mu = \mu_0$) of one sample t-test applied to the case where the underlying distribution is the distribution X rather than a Gaussian. To obtain samples of X you can use your function `generate_sample_X`. Plot the test size as a function of the sample size. In your simulation study, you can set the population mean of X to 3 (in this case the μ_0 should be also 3). Consider a set of different sample sizes ranging from 2 to 40, and for each sample size, carry out 10000 trials.

Answer

```
trials <- seq(10000)
sample_sizes <- seq(2, 40, 2)
significance_level <- 0.05

mu_0 <- 3
mu <- 3

set.seed(2)

df_simulated_t_test <- crossing(trials=trials,
                                sample_sizes=sample_sizes) %>%
  mutate(sample=map(sample_sizes,  ~generate_sample_X(.x, mu)  ) ) %>%
  #mutate(sample=map(sample_sizes,  ~rnorm(.x, mean=mu, sd=3)  ) ) %>%
  mutate(p_value = map_dbl(sample, ~t.test(.x, mu=mu_0, alternative='two.sided')$p.value ) ) %>%
  mutate(reject=(p_value<significance_level))


df_test_size <- df_simulated_t_test %>%
  group_by(sample_sizes) %>%
  summarise(test_size=mean(reject))
```

```
df_test_size %>% ggplot( aes(x=sample_sizes, y=test_size) ) + geom_point() +
  theme_bw() + xlab('sample size') + ylab('test size')
```



(Q5)

Conduct a simulation study to investigate the statistical power (the average number of rejecting the null hypothesis when $\mu \neq \mu_0$) of one sample t-test applied to the case where the underlying distribution is the distribution X rather than a Gaussian. To obtain samples of X you can use your function `generate_sample_X`. Plot the power as a function of the sample size. In your simulation study, you can set the population mean of X to 3 and $\mu_0 = 3.2$. Consider sample sizes ranging from 2 to 40, and for each sample size, carry out 10000 trials.

Answer

```
trials <- seq(10000)
sample_sizes <- seq(2, 40, 2)
significance_level <- 0.05

mu_0 <- 3.2
mu <- 3

set.seed(2)

df_simulated_t_test <- crossing(trials=trials,
                                sample_sizes=sample_sizes) %>%
  mutate(sample=map(sample_sizes,  ~generate_sample_X(.x, mu)  ) ) %>%
  #mutate(sample=map(sample_sizes,  ~rnorm(.x, mean=mu, sd=3)  ) ) %>%
  mutate(p_value = map_dbl(sample, ~t.test(.x, mu=mu_0, alternative='two.sided')$p.value ) ) %>%
  mutate(reject=(p_value<significance_level))
```

```r
df_power <- df_simulated_t_test %>%
  group_by(sample_sizes) %>%
  summarise(power=mean(reject))
```

```r
df_power %>% ggplot( aes(x=sample_sizes, y=power) ) + geom_point() +
  theme_bw() + xlab('sample size') + ylab('power')
```