Exploratory Data Analysis: Golub Case Study

Chapter 1, Lab 3: Solutions

OpenIntro Biostatistics

This lab presents the details of how to conduct the analysis discussed in Section 1.7.2 of *OpenIntro Biostatistics*. We recommend that a reader carefully review the material in the text prior to working through the lab, as the lab will focus on demonstrating techniques in R rather than reviewing the conceptual strategy behind the analysis.

Background information

The 1999 Golub leukemia study represents one of the earliest applications of microarray technology for diagnostic purposes. At the time of the Golub study, no single diagnostic test was sufficient for distinguishing between acute myeloid leukemia (AML) and acute lymphoblastic leukemia (ALL). To investigate whether gene expression profiling could be a tool for classifying acute leukemia type, Golub and co-authors used Affymetrix DNA microarrays to measure the expression level of 7,129 genes from children known to have either AML or ALL.

The goal of the study was to develop a procedure for distinguishing between AML and ALL based only on the gene expression levels of a patient. There are two major issues to be addressed:

1. *Which genes are the most informative for making a prediction?* If a gene is differentially expressed between individuals with AML versus ALL, then measuring the expression level of that gene may be informative for diagnosing leukemia type. For example, if a gene tends to be highly expressed in AML individuals, but only expressed at low levels in ALL individuals, it is more likely to be a reliable predictor of leukemia type than a gene that is expressed at similar levels in both AML and ALL patients.
2. *How can leukemia type be predicted from expression data?* Suppose that a patient's expression profile is measured for a group of genes. In an ideal scenario, all the genes measured would express AML-like expression, or ALL-like expression, making a prediction obvious. In reality, however, a patient's expression profile will not follow an idealized pattern. Some of the genes may have expression levels more typical of AML, while others may suggest ALL. It is necessary to clearly define a strategy for translating raw expression data into a prediction of leukemia type.

All datasets used in this lab are available from the `oibiostat` package. Phenotypic and expression data have been collected for 72 patients. The expression data from the 62 patients in `golub.train` will be used to identify informative genes for making a prediction. The prediction strategy will then be tested on the remaining 10 patients in `golub.test`.

Identifying informative genes

The discussion in the text begins by illustrating concepts using a simplified version of the dataset (`golub.small`) that contains only data from the 10 patients and 10 genes. Here, instead of starting with `golub.small`, we will examine a random sample of 100 genes for all patients in `golub.train`. The methods from the initial analysis can then be applied to the data from all 7,129 genes.

1. Run the following code to load `golub.train` and create `gene.matrix`, which contains only the expression data and not the phenotype information in the first 6 columns.

```
#load the data
library(oibiostat)
data(golub.train)

gene.matrix = as.matrix(golub.train[ ,-(1:6)])
```

   By using the - in front of the column numbers, the matrix notation specifies that columns 1 through 6 should *not* be included. The same matrix could be created by specifying that columns 7 through 7,135 should be included, with `[,7:7135]`.

2. Draw a random sample of 100 genes from the dataset.

```
#create a vector of integers from 1 to the total number of genes
gene.columns = 1:ncol(gene.matrix)

#set the seed for a pseudo-random sample
set.seed(2401)

#sample 100 numbers from gene.columns, without replacement
gene.index.set = sample(gene.columns, size = 100, replace = FALSE)

#creeate a matrix with expression data from the rows specified by gene.index.set
gene.matrix.sample = gene.matrix[ ,gene.index.set]
```

   a) What are the first five values of `gene.index.set`? How were the numbers in `gene.index.set` chosen?

   The first five values of `gene.index.set` are 4841, 6731, 6535, 6743, 4804. The numbers in `gene.index.set` were randomly sampled from the integers between 1 and 7,129.

```
#view gene.index.set
gene.index.set[1:5]
```

```
## [1] 4841 6731 6535 6743 4804
```

   b) Why is it important to sample without replacement?

   Sampling without replacement is necessary to ensure that a number does not get chosen more than once. Since the numbers are used to identify the column numbers of particular genes, sampling without replacement allows for 100 distinct genes to be drawn from the pool of 7,129 genes.

   c) View `gene.matrix.sample`; what does it contain? How is `gene.matrix.sample` related

to `gene.index.set`?

The matrix contains gene expression data for the 100 genes corresponding to the columns specified by the numbers in `gene.index.set`, for each of the 62 patients.

d) What are the first five gene names of the 100 genes sampled?

Each column contains expression information for agene; the first five gene names are given at the top of the first five columns:

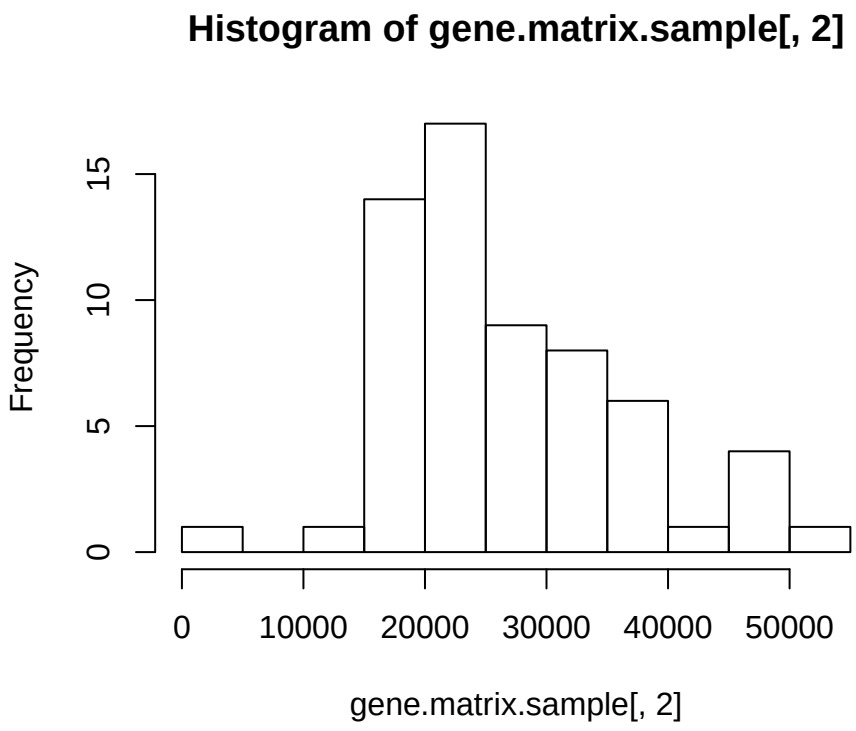
```
colnames(gene.matrix.sample)[1:5]
```

```
## [1] "X95406_at"        "S82297_at"        "X82850_s_at"      "HG2239-HT2324_at"
## [5] "X92475_at"
```

e) Plot a histogram showing the distribution of the expression levels of the second gene across patients. Describe the distribution.

The histogram is approximately symmetric, although there is some slight skewing to the right. The values are all positive, and centered around 25,000.

```
#plot a histogram
hist(gene.matrix.sample[ ,2])
```



3. Create a logical variable, `leuk.type`, that has value `TRUE` for AML and value `FALSE` for anything that is not AML (i.e., `allT` and `allB`). For a logical variable, R interprets `TRUE` as `1` and `FALSE` as `0`.

```
#create logical variable
leuk.type = (golub.train$cancer == "aml")

#view table of leukemia types
table(leuk.type)
```

```
## leuk.type
## FALSE  TRUE
##    42    20
```

```
#calculate sum of leuk.type
sum(leuk.type)
```

```
## [1] 20
```

   a) When creating the logical variable, why not write `"allT"` or `"allB"` instead of `"aml"`?

      The purpose of creating the logical variable is to separate patients based on whether they have AML or ALL. For this analysis, both types of ALL patients need to be combined in a single group. To do this, the logical variable queries whether or not a patient is in the AML category; if so, `TRUE` is assigned. All other patients (i.e., both ALL-T and ALL-B patients) are assigned a `FALSE`. Basing the logical variable on `allT` or `allB` would result in AML and ALL-B/ALL-T patients being grouped together.

   b) How many patients have AML? How many have ALL?

      There are 42 patients with ALL and 20 patients with AML.

4. Summarize the data separately for AML patients and for ALL patients.

   a) The following code calculates the mean expression level for each sampled gene across AML patients, storing it in the variable `aml.mean.expression`. The `apply()` function executes a function across a matrix—in this case, the function is `mean`, and the `2` in the argument indicates that the function should be applied on each column (replacing the `2` with a `1` would result in the mean being calculated across the rows).

```
#calculate mean expression level for each sampled gene across AML patients
aml.mean.expression = apply(gene.matrix.sample[leuk.type == TRUE, ], 2, mean)

#calculate mean expression level for each sampled gene across ALL patients
all.mean.expression = apply(gene.matrix.sample[leuk.type == FALSE, ], 2, mean)
```

   Run the code to create `aml.mean.expression`, then create `all.mean.expression`, a vector containing the mean expression levels for each gene in ALL patients.

   b) Explain the logic behind the code to generate `aml.mean.expression` and `all.mean.expression`. In other words, what do the separate components instruct R to do?

      The command for calculating `aml.mean.expression` instructs R to calculate the mean down each column of `gene.matrix.sample`, selectively including only values from rows

for which `leuk.type` is `TRUE`; that is, only including expression values from patients with AML. To calculate the mean expression levels for ALL patients, the command skips the rows with data from AML patients.

c) View the contents of `aml.mean.expression`. What is the average expression level of the first sampled gene across AML patients?

The average expression level of the first sampled gene is -483.96.

```
as.matrix(aml.mean.expression)[1, ]
```

```
## X95406_at
## -483.9617
```

5. For each gene, compare the mean expression value among AML patients to the mean among ALL patients; calculate the differences in mean expression levels between AML and ALL patients.

```
#calculate the differences
diff.mean.expression = (aml.mean.expression - all.mean.expression)

#view list as a matrix
diff.mean.expression.matrix = as.matrix(diff.mean.expression)
diff.mean.expression.matrix[1, ]
```

```
## X95406_at
##  129.5117
```

a) What is the difference in mean expression level between AML and ALL for the first gene on the list; on average, is this gene more highly expressed in AML patients or ALL patients? Does it seem like this gene could be a good predictor of leukemia type? Why or why not?

The difference in mean expression level between AML and ALL for the first gene on the list is 129.51; on average, the gene is more highly expressed in AML patients. It is not possible to tell whether this gene could be a good predictor, without knowing the distribution of differences. A difference of 129.51 could very well be a typical mean expression difference between the AML and ALL, or it might be extreme.

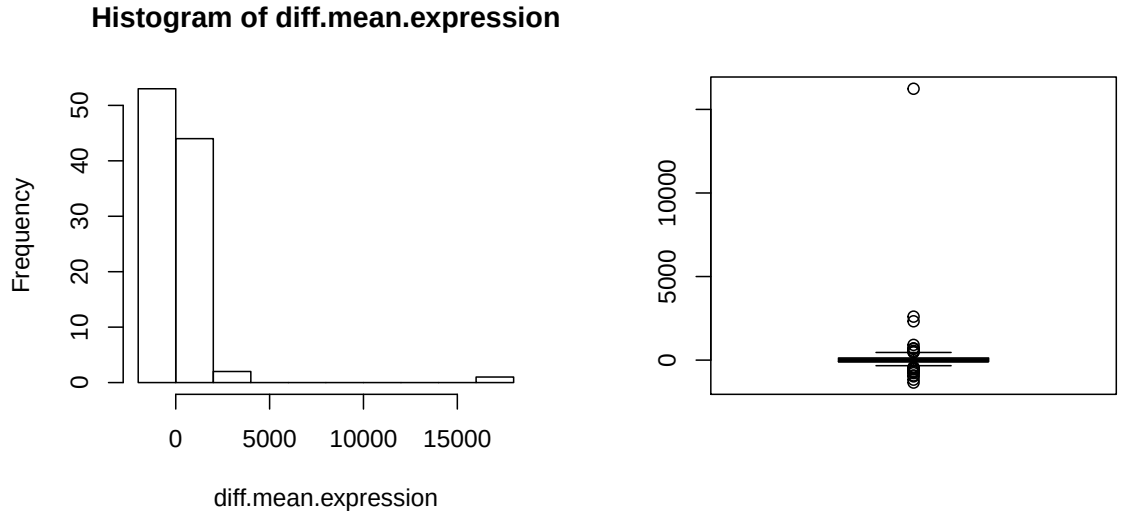
b) Using numerical and graphical summaries, describe the distribution of differences in mean expression levels.

From the histogram, a vast majority of the observations are between -2500 and 2500; more precisely, the middle 50% of the data are between -90.44 and 129.34. However, the boxplot indicates the presence of both small and large outliers; these genes may be ones useful for differentiating between AML and ALL.

```
#numerical summaries
summary(diff.mean.expression)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1344.34   -90.44   -14.90   183.59   129.34 16235.83
```

```r
#graphical summaries
par(mfrow = c(1, 2))
hist(diff.mean.expression)
boxplot(diff.mean.expression)
```



6. Identify the outliers. Run the following code to set up the definition of outliers as specified in Chapter 1 of *OpenIntro Biostatistics*:

```r
#define 3rd and 1st quartiles
quart.3 = quantile(diff.mean.expression.matrix[ ,1], 0.75, na.rm = TRUE)
quart.1 = quantile(diff.mean.expression.matrix[ ,1], 0.25, na.rm = TRUE)

#define interquartile range
iqr = quart.3 - quart.1

#define upper and lower bound for outliers
lb.outlier = quart.1 - 1.5*iqr
ub.outlier = quart.3 + 1.5*iqr
```

The following code creates a list of the large outliers, genes with expression differences larger than `ub.outlier`:

```r
#creates list of large outliers
which.large.out = diff.mean.expression > ub.outlier
large.out = as.matrix(diff.mean.expression.matrix[which.large.out, ])
large.out
```

```
##                          [,1]
## M11147_at           16235.8346
## J02923_at             558.3751
## U47931_at             683.0611
## HG2279-HT2375_at     2600.9531
```

```
## M72885_rna1_s_at  2323.1771
## D78134_at          683.1454
## X55079_rna1_at     471.4683
## X13839_at          479.9994
## X62055_at          910.3332
## U37690_at          701.0644
```

```
#creates ordered list of large outliers, from largest to smallest
order.large.out = order(large.out[ ,1], decreasing = TRUE) #assigns ordering to rows
ordered.large.out = as.matrix(large.out[order.large.out, ]) #sorts outlier list
ordered.large.out
```

```
##                         [,1]
## M11147_at        16235.8346
## HG2279-HT2375_at  2600.9531
## M72885_rna1_s_at  2323.1771
## X62055_at          910.3332
## U37690_at          701.0644
## D78134_at          683.1454
## U47931_at          683.0611
## J02923_at          558.3751
## X13839_at          479.9994
## X55079_rna1_at     471.4683
```

a) What are the upper and lower outlier bounds?

The upper bound is 459.01 and the lower bound is -420.11.

b) How many large outliers are present in the sample?

There are ten large outliers present in the sample.

```
table(which.large.out)
```

```
## which.large.out
## FALSE  TRUE
##    90    10
```

```
length(large.out) #alternatively, use length()
```

```
## [1] 10
```

c) How many rows and columns does `large.out` have? Explain why.

The matrix `large.out` has two columns and ten rows. It is constructed from all two columns of `diff.mean.expression.matrix`, and only the rows for which the expression value is greater than `ub.outlier`

d) View `order.large.out`. What do these numbers represent?

These numbers represent the row ordering if the rows were to be sorted in decreasing order, from largest to smallest. For example, the 1 that is the first number in the vector indicates that the first row contains the largest value, and should be first in an ordered list.

```
order.large.out
```

```
##  [1]  1  4  5  9 10  6  3  2  8  7
```

e) Modify the code to find small outliers. How many small outliers are present in the sample?

```
#creates list of small outliers
which.small.out = diff.mean.expression.matrix < lb.outlier
small.out = as.matrix(diff.mean.expression.matrix[which.small.out,])

#creates ordered list of small outliers, from smallest to largest
order.small.out = order(small.out[,1], decreasing = FALSE)
ordered.small.out = as.matrix(small.out[order.small.out,])
```

\textcolor{NavyBlue}{There are twelve small outliers present in the sample.}

```
table(which.small.out)
```

```
## which.small.out
## FALSE  TRUE
##    88    12
```

f) Which gene has the largest positive difference in mean expression between AML and ALL samples? Which gene has the largest negative difference in mean expression between AML and ALL samples?

These genes are indicated by the largest large outlier and the smallest small outlier:

```
#largest large outlier
ordered.large.out[1, ]
```

```
## M11147_at
##  16235.83
```

```
#smallest small outlier
ordered.small.out[1, ]
```

```
## D86970_at
## -1344.335
```

g) In a research setting, it can also be useful to inspect the entire list and examine genes that are close to the outlier cutoff. Run the following code to order the entire list of expression differences in decreasing order:

```
order.decreasing = order(diff.mean.expression.matrix[ ,1], decreasing = TRUE)
ordered.outliers = as.matrix(diff.mean.expression.matrix[order.decreasing, ])
```

Which gene just missed the cutoff to qualify as a large outlier? Which gene is closest to the cutoff for qualifying as a small outlier?

The gene from the probe U38980_at with a difference of 457.43 just missed the cutoff to qualify as a large outlier, while U47414_at with a difference of -333.81 is closest to the cutoff for

qualifying as a small outlier. This can be read off the ordered list of all outliers; alternatively, the commands used below can be used to extract the relevant rows:

```
#pull the 11th largest value from the list where decreasing = TRUE
ordered.outliers[11, ]
```

```
## U38980_at
##  457.4343
```

```
#pull the 13th largest value from the list where decreasing = FALSE
order.increasing = order(diff.mean.expression.matrix[ ,1], decreasing = FALSE)
ordered.outliers.inc = as.matrix(diff.mean.expression.matrix[order.increasing, ])
ordered.outliers.inc[13, ]
```

```
## U47414_at
## -333.8123
```

7. The genes previously identified as outliers are only outliers of the specific 100 genes chosen in the sample. From the complete set of data in `golub.train`, identify the five largest outliers and five smallest outliers out of all 7,129 genes. (*Hint*: this can be done with only a few modifications to the code run for the initial analysis.)

```
#calculate mean expression level for each gene across AML patients and ALL patients
aml.mean.expression = apply(gene.matrix[leuk.type == TRUE, ], 2, mean)
all.mean.expression = apply(gene.matrix[leuk.type == FALSE, ], 2, mean)

#calculate the differences
diff.mean.expression = (aml.mean.expression - all.mean.expression)
diff.mean.expression.matrix = as.matrix(diff.mean.expression)

#define outlier bounds
quart.3 = quantile(diff.mean.expression.matrix[ ,1], 0.75, na.rm = TRUE)
quart.1 = quantile(diff.mean.expression.matrix[ ,1], 0.25, na.rm = TRUE)
iqr = quart.3 - quart.1
lb.outlier = quart.1 - 1.5*iqr
ub.outlier = quart.3 + 1.5*iqr

#identify large outliers
which.large.out = diff.mean.expression.matrix > ub.outlier
large.out = as.matrix(diff.mean.expression.matrix[which.large.out,])

order.large.out = order(large.out[,1], decreasing = TRUE)
ordered.large.out = as.matrix(large.out[order.large.out,])
ordered.large.out[1:5, ]
```

```
##      M19507_at      M27891_at      M11147_at M96326_rna1_at    Y00787_s_at
##       19820.86       17657.05       16235.83       15914.76       15845.77
```

```
#identify small outliers
which.small.out = diff.mean.expression.matrix < lb.outlier
small.out = as.matrix(diff.mean.expression.matrix[which.small.out,])
```

```
order.small.out = order(small.out[,1], decreasing = FALSE)
ordered.small.out = as.matrix(small.out[order.small.out,])
ordered.small.out[1:5, ]
```

```
## M14483_rna1_s_at   X82240_rna1_at        X58529_at        M33680_at
##       -11293.653       -10415.906        -9629.464        -9296.219
##   U05259_rna1_at
##        -8383.484
```

The only necessary changes are to specify `gene.matrix` instead of `gene.matrix.sample` when calculating the vectors containing mean expression, and to print out the 5 largest/smallest outliers rather than single largest/smallest outlier.

Predicting leukemia type

Figure 1 illustrates the main ideas behind the strategy developed by the Golub team to predict leukemia type from expression data. The vertical orange bars represent the gene expression levels of a patient for each gene, relative to the mean expression for AML patients and ALL patients from the training dataset (vertical blue bars). A gene will "vote" for either AML or ALL, depending on whether the patient's expression level is closer to μ_{AML} or μ_{ALL}. In the example shown, three of the genes are considered to have ALL-like expression, versus the other two that are more AML-like. The votes are also weighted to account for how far an observation is from the midpoint between the two means (horizontal dotted blue line); i.e., the length of the dotted line shows the deviation from the midpoint. For example, the observed expression value for gene 2 is not as strong an indicator of ALL as the expression value for gene 1. The magnitude of the deviations $(v_1, v_2, \ldots)$ are summed to obtain V_{AML} and V_{ALL}, and a higher value indicates a prediction of either AML or ALL, respectively.

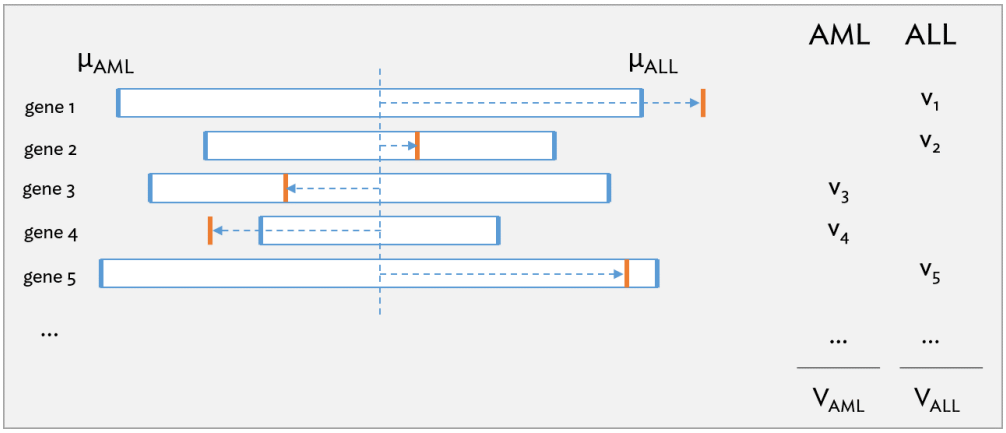


Figure 1: Schematic of the prediction strategy used by the Golub team, reproduced with modifications from Fig. 1B of the original paper.

The published analysis chose to use 50 informative genes. For simplicity, a smaller number of genes will be used in our version of the analysis. As identified in the previous section, 10 genes will be used as predictors—the 5 largest outliers and 5 smallest outliers for the difference in mean expression between AML and ALL. The expression levels for these 10 genes will be used to make predictions of leukemia type for the patients in `golub.test`.

Steps in the analysis:

- Calculate μ_{AML} and μ_{ALL} for each of the predictor genes.
- Determine the vote direction for each gene; is its expression more AML-like or ALL-like?
- Calculate $v_1, \ldots, v_{10}$, the magnitude of the deviations from the midpoint between the two means..
- Calculate V_{AML} and V_{ALL} for each patient, and determine which leukemia type is predicted.
- Assess prediction accuracy, comparing the prediction to the actual leukemia status of each patient; recall that leukemia status is known for all study patients.

8. Run the following code to identify the column numbers in `golub.test` and `golub.train` corresponding to the 10 predictor genes identified in Question 7. Using `which()` returns the column numbers that correspond to the 10 predictor genes. from `gene.matrix`, the matrix without the 6 columns of phenotypic data. Adding 6 results in the corresponding column numbers in `golub.test` and `golub.train`.

```
#column numbers for large outliers (gene.matrix)
predictors.large = which(which.large.out)[order.large.out[1:5]]

#column numbers for small outliers (gene.matrix)
predictors.small = which(which.small.out)[order.small.out[1:5]]

#combine column numbers for large and small outliers (gene.matrix)
predictors = c(predictors.large, predictors.small)

#predictor column numbers of golub.test and golub.train
predictor.cols = predictors + 6
```

9. Calculate μ_{AML} and μ_{ALL} for each of the predictor genes. Store the results in `predictor.means.aml` and `predictor.means.all`. What are μ_{AML} and μ_{ALL} for the first predictor gene?

   For the first predictor gene, $\mu_{AML} = 20,142.84$ and $\mu_{ALL} = 321.99$.

```
#subset golub.train into aml and all patients
leuk.type = (golub.train$cancer == "aml")
golub.train.aml = golub.train[leuk.type == TRUE,]
golub.train.all = golub.train[leuk.type == FALSE,]

#calculate predictor.means.aml
predictor.means.aml = apply(golub.train.aml[, predictor.cols], 2, mean)

#calculate predictor.means.all
predictor.means.all = apply(golub.train.all[, predictor.cols], 2, mean)

#means for the first predictor gene
predictor.means.aml[1]; predictor.means.all[1]
```

```
## M19507_at
##  20142.84
```

```
## M19507_at
##  321.9871
```

10. Determine the vote direction for each gene. Let `0` represent a vote for AML and `1` represent a vote for ALL.

```
#load golub.test
data("golub.test")

#create matrix with expression data for predictor genes
test.gene.matrix = golub.test[, predictor.cols]
```

```
#create empty matrix to store vote directions
num.genes = ncol(test.gene.matrix); num.patients = nrow(test.gene.matrix)
votes = matrix(nrow = num.patients, ncol = num.genes)

#calculate vote directions
dist.from.aml = vector("numeric", num.genes)
dist.from.all = vector("numeric", num.genes)

for(i in 1:dim(votes)[1]){
  for(j in 1:dim(votes)[2]) {

dist.from.aml[i] = abs(test.gene.matrix[i, j] - predictor.means.aml[j])
dist.from.all[i] = abs(test.gene.matrix[i, j] - predictor.means.all[j])

if(dist.from.aml[i] <= dist.from.all[i]){
  votes[i, j] = 0  #assigns 0 if dist. from AML mean <= dist. from ALL mean
} else { votes[i, j] = 1  #assigns 1 if otherwise
}

  }
}
```

The above code uses `for()` loops and a conditional `if()` statement to assign vote directions for each of the 10 predictor genes, for each patient. A more formal introduction to loops and conditional statements will be provided in the next chapter; for now, focus on understanding the logic of the code, rather than the precise syntax.

a) The loop stores the results in a matrix called `votes`. What are the dimensions of the matrix? Does a single row contain the votes for a single patient, or the votes for a single gene? (*Hint*: refer to the syntax used to create the empty `votes` matrix.)

   The syntax used to create the `votes` matrix specifies that the number of rows equals the number of patients, and the number of columns equals the number of genes. The line beforehand specifies that `num.genes` and `num.patients` are defined by the number of columns and rows in the matrix `test.gene.matrix`. Thus, the matrix has 10 rows and 10 columns. Since the number of rows corresponds to the number of patients, each row contains the information for a single patient; a single row will contain the votes for a single patient.

b) Inside the loop, the two vectors `dist.from.aml` and `dist.from.all` store the distance between an expression value and the AML or ALL mean, respectively; the distance is calculated as the absolute value of the expression value in a specific cell minus a predictor mean (either AML or ALL). The loop extends in two directions, where `i` ranges from 1:10 and `j` ranges from 1:10, since the dimensions of the `votes` matrix are 10 (rows) by 10 (columns).

i. In the first step of the loop, `i = 1` and `j = 1`. Based on what you know about bracket notation, describe what is being calculated and stored in `dist.from.aml` and `dist.from.all` for these values of `i` and `j`.

When `i = 1` and `j = 1`, the distance between μ_{AML} and μ_{ALL} is being calculated in patient 1, for the first predictor gene. The element `test.gene.matrix[1, 1]` contains the expression level of gene 1 in patient 1; `predictor.means.aml[1]` and `predictor.means.all[1]` contain μ_{AML} and μ_{ALL} for gene 1. The two numbers calculated will be the first value in the two vectors `dist.from.aml` and `dist.from.all`.

ii. What is being calculated and stored when `i = 2` and `j = 1`?

When `i = 2` and `j = 1`, the distance between μ_{AML} and μ_{ALL} is being calculated in patient 2, for the first predictor gene. The element `test.gene.matrix[2, 1]` contains the expression level of gene 1 in patient 2. The values in `predictor.means.aml[1]` and `predictor.means.all[1]` remain the same, containing μ_{AML} and μ_{ALL} for gene 1.

iii. What is being calculated and stored when `i = 1` and `j = 2`?

When `i = 1` and `j = 2`, the distance between μ_{AML} and μ_{ALL} is being calculated in patient 1, for the second predictor gene. The element `test.gene.matrix[1, 2]` contains the expression level of gene 2 in patient 1. The values in `predictor.means.aml[2]` and `predictor.means.all[2]` contain μ_{AML} and μ_{ALL} for gene 2.

Note: One intuitive way to think through how the loop works is that it first calculates the distances for a particular gene for all patients (`i` from 1 to 10), then moves on to the next gene. Once it moves on to another gene (i.e., next value of `j`), the values in `dist.from.aml` and `dist.from.all` are re-calculated.

iv. Based on the previous answers, why does it make sense for the loop to calculate distance based on `predictor.means.aml[j]` and `predictor.means.all[j]` instead of `predictor.means.aml[i]` and `predictor.means.all[i]`?

In the distance calculation, the value of the predictor mean does not change with values of `i`; it remains constant for different values of `i` and only changes with values of `j`. This makes sense conceptually because there is a single μ_{AML} value (and a single μ_{ALL} value) for each gene, and the loop should subtract that single μ value from the expression level at that gene for each patient in order to calculate distance.

c) View the first row of the matrix `votes`. For which predictor genes does this patient have AML-like expression, and for which does this patient have ALL-like expression?

The patient has AML-like expression for predictor genes 2, 3, 6, 7, 9, and 10; AML votes are denoted with a `0`. The patient has ALL-like expression for predictor genes 1, 4, 5, and 8; ALL votes are denoted with a `1`. The names of the genes can be determined by looking at the column names of `test.gene.matrix`.

```
#view the first row of votes
votes[1, ]
```

```
##  [1] 1 0 0 1 1 0 0 1 0 0
```

```
#return the gene names at particular values
patient.1.votes = votes[1, ]
colnames(test.gene.matrix)[patient.1.votes == 0]  #aml-like expression
```

```
## [1] "M27891_at"       "M11147_at"       "M14483_rna1_s_at" "X82240_rna1_at"
## [5] "M33680_at"       "U05259_rna1_at"
```

```
colnames(test.gene.matrix)[patient.1.votes == 1]  #all-like expression
```

```
## [1] "M19507_at"      "M96326_rna1_at" "Y00787_s_at"    "X58529_at"
```

11. Calculate $v_1,\ldots,v_{10}$, the magnitude of the deviations from the midpoint between the two means. The following code stores the magnitudes of the deviations in a matrix named deviation.magnitude.

```
deviation.magnitude = matrix(nrow = num.patients, ncol = num.genes)

for(i in 1:dim(deviation.magnitude)[1]){
  for(j in 1:dim(deviation.magnitude)[2]) {

midpoint = (predictor.means.aml - predictor.means.all)/2

deviation.magnitude[i,j] = abs(test.gene.matrix[i, j] - midpoint[j])

  }
}

#adds predictor gene probe names from test.gene.matrix
colnames(deviation.magnitude) <- colnames(test.gene.matrix)
```

    a) View deviation.magnitude. What is the deviation from the midpoint at the M19507_at probe for patient 1? For which patient is the deviation at this gene probe the largest?

    The deviation from the midpoint at the M19507_at probe for patient 1 is 5,429.31. The largest deviation at this gene point occurs with patient number 3, with deviation 11,383.1.

```
#deviation.magnitude for patient 1 at probe M19507_at
deviation.magnitude[1, "M19507_at"]
```

```
## M19507_at
##  5429.313
```

```
#largest deviation at probe M19507_at
which.max(deviation.magnitude[ ,"M19507_at"])
```

```
## [1] 3
```

```
max(deviation.magnitude[ ,"M19507_at"])
```

```
## [1] 11383.1
```

b) For patient 1, which values should be summed to calculate V_{AML}? (*Hint*: Refer to Question 10, part c.)

The values to calculate V_{AML} are the magnitudes corresponding to genes that voted for AML. For patient 1, these values correspond to the predictor genes 2, 3, 6, 7, 9, and 10.

12. Calculate V_{AML} and V_{ALL} for each patient, and determine which leukemia type is predicted.

```
#sum the votes for AML and ALL
V.aml = vector("numeric", num.patients)
V.all = vector("numeric", num.patients)

for(i in 1:num.patients){

  V.aml[i] = sum(deviation.magnitude[i, which(votes[i,] == 0)])
  V.all[i] = sum(deviation.magnitude[i, which(votes[i,] == 1)])

}

#determine the predicted leukemia type
predicted.leuk.type = vector("numeric", num.patients)

for(i in 1:num.patients){
  if (V.aml[i] > V.all[i]){predicted.leuk.type[i] = "aml"}
  if (V.aml[i] < V.all[i]){predicted.leuk.type[i] = "all"}
  if (V.aml[i] == V.all[i]){predicted.leuk.type[i] = "tie"}
}

predicted.leuk.type
```

```
##  [1] "aml" "aml" "aml" "all" "aml" "all" "all" "all" "all" "all"
```

a) Briefly describe how the `for()` loop to create `V.aml` and `V.all` works.

The loop to create `V.aml` and `V.all` sums the deviation magnitudes, including magnitudes based on the values in the votes matrix, to return a value of V_{AML} and V_{ALL} for each patient. For example, when `i = 1`, V_{AML} for patient 1 is calculated by summing the numbers in row 1 of the deviation magnitude for specific columns—only the columns for which there is a `0` in row 1 of the votes matrix.

b) Briefly describe how the `for()` loop and `if` statements to create `predicted.leuk.type` work.

The loop and conditional statements to create `predicted.leuk.type` compares V_{AML} and V_{ALL} for each patient, and returns a specific result based on the comparison. If V_{AML} is less than V_{ALL}, the prediction is AML. If V_{AML} is greater than V_{ALL}, the prediction is ALL. If the two values V_{AML} and V_{ALL} are equal, then the result is a tie.

c) What are the predicted leukemia types for the 10 patients in `golub.test`?

The first three patients and the fifth patient are predicted to have AML; other patients are predicted to have ALL.

13. Assess the prediction accuracy, comparing the prediction to the actual leukemia status of each patient in golub.test. How well do the predictions match patient leukemia status?

The prediction matches actual patient leukemia status for nine out of ten patients.

```
#compare predictions to actual leukemia type

#option 1: visual comparison
golub.test$cancer
```

```
##  [1] aml  aml  aml  aml  aml  allB allB allB allB allB
## Levels: allB allT aml
```

```
predicted.leuk.type
```

```
##  [1] "aml" "aml" "aml" "all" "aml" "all" "all" "all" "all" "all"
```

```
#option 2: use logical variables
actual.leuk.type.ind = (golub.test$cancer == "aml")
predicted.leuk.type.ind = (predicted.leuk.type == "aml")

comparison.results = (actual.leuk.type.ind == predicted.leuk.type.ind)
table(comparison.results)
```

```
## comparison.results
## FALSE  TRUE
##     1     9
```