

6.5: R Assignment

Molecular Similarity

S. Kim, J. Cuadros

November 23rd, 2019

Objectives

- Generate molecular fingerprints for a given molecule.
- Evaluate structural similarity between molecules using different molecular fingerprints and similarity metrics.

Many useful documents/papers describe various aspects of molecular similarity, including molecular fingerprints and similarity measures. Please read these if you need more details.

- Getting Started with the RDKit in Python (<https://www.rdkit.org/docs/GettingStartedInPython.html#fingerprinting-and-molecular-similarity>)
- Fingerprint Generation, GraphSim Toolkit 2.4.2 (<https://docs.eyesopen.com/toolkits/python/graphsimtk/fingerprint.html>)
- Chemical Fingerprints (<https://docs.chemaxon.com/display/docs/Chemical+Fingerprints>)
- Extended-Connectivity Fingerprints (<https://doi.org/10.1021/ci100050t>)

1. Fingerprint Generation

```
if (!require("rcdk", quietly=TRUE)) {  
  install.packages("rcdk", repos="https://cloud.r-project.org/",  
    quiet=TRUE, type="binary")  
  library("rcdk")  
}  
if (!require("fingerprint", quietly=TRUE)) {  
  install.packages("fingerprint", repos="https://cloud.r-project.org/",  
    quiet=TRUE, type="binary")  
  library("fingerprint")  
}
```

```
smi <- "CC(C)C1=C(C(=C(N1CC[C@H](C[C@H](CC(=O)O)O)O)C2=CC=C(C=C2)F)C3=CC=CC=C3)C(=O)N"  
mol <- parse.smiles(smi)
```

1-(1) MACCS keys

The MACCS key is a binary fingerprint (a string of 0's and 1's). Each bit position represents the presence (=1) or absence (=0) of a pre-defined structural feature. The feature definitions for the MACCS keys are available at:

- <https://github.com/rdkit/rdkit/blob/master/rdkit/Chem/MACCSKeys.py>
- <http://www.mayachemtools.org/docs/modules/html/MACCSKeys.html>

```
fp <- get.fingerprint(mol[[1]], type = 'maccs',  
  fp.mode = 'bit', verbose=FALSE)  
str(fp)
```

```
as.character(fp)
```

```
nchar(as.character(fp))
```

These are some methods that allow you to get some additional information on the fingerprint.

```
print(paste("Number of bits:", length(fp)))
```

```
print(paste("Number of ON bits:", length(fp@bits)))
```

```
print(paste("Number of OFF bits:", fp@nbit-length(fp@bits)))
```

```
as.character(fp)
```

```
(fp_bin <- unlist(strsplit(as.character(fp), "")))
```

```
paste(fp_bin, collapse="")
```

```
fp_bytes <- substring(paste("00", as.character(fp), sep=""),
  seq(1, length(fp)+2, 8), seq(8, length(fp)+2, 8))
# bits in bytes are read right to left, https://code.google.com/archive/p/chem-finger/
fp_bytes <- sapply(fp_bytes, function(x)
  paste(rev(strsplit(x, "")[[1]]), collapse=""))

(fp_bytes <- strtoi(fp_bytes, base=2))
```

```
(fp_hex <- as.raw(fp_bytes))
```

```
paste(fp_hex, collapse = "")
```

```
fp_bin2 <- unlist(lapply(paste("0x", fp_hex, sep=""), function(x) rawToBits(as.raw(x))))
fp_bin2 <- as.numeric(fp_bin2)
substring(paste(fp_bin2, collapse=""), 3, 168)
```

Exercise 1a: Generate the MACCS keys for the molecules represented by the following SMILES, and get the positions of the bits set to ON in each of the three fingerprints. What fragments do these bit positions correspond to?

```
smiles <- c('C1=CC=CC=C1', # Benzene (Kekule)
  'c1ccccc1', # Benzene ("Aromatized" carbons)
  'C1CCCCC1') # Cyclohexene
```

```
# Write your code here
```

Write the fragment definition of the bits ON (one is already provided for you as an example). - 118: ACH2CH2A > 1

1-(2) Circular Fingerprints

Circular fingerprints are hashed fingerprints. They are generated by exhaustively enumerating “circular” fragments (containing all atoms within a given radius from each heavy atom of the molecule) and then hashing these fragments into a fixed-length bitstring. (Here, the “radius” from an atom is measured by the number of bonds that separates two atoms).

Examples of circular fingerprints are the extended-connectivity fingerprint (ECFPs) and their variant called FCFPs (Functional-Class Fingerprints), originally described in a paper by Rogers and Hahn (<https://doi.org/10.1021/ci100050t>). Sometimes, for instance in RDKit, these fingerprints are called “Morgan Fingerprints” (<https://www.rdkit.org/docs/GettingStartedInPython.html#morganfingerprints-circular-fingerprints>).

CDK can compute a ECFP6 fingerprint.

```
fp <- get.fingerprint(mol[[1]], type = 'circular',  
  fp.mode = 'bit', verbose=FALSE)  
as.character(fp)
```

Morgan fingerprints can be obtained from the ChEMBL webservice, which is based on RDKit.

```
if(!require("httr")) {  
  install.packages(("httr"), repos="https://cloud.r-project.org/",  
    quiet=TRUE, type="binary")  
  library("httr")  
}
```

```
sdf <- readLines(paste("https://cactus.nci.nih.gov/chemical/structure/",  
  URLencode(smi,reserved = T), "/SDF", sep=""))  
sdf <- paste(sdf, collapse="\n")  
url <- paste("https://www.ebi.ac.uk/chembl/api/utils/sdf2fps",  
  "?n_bits=1024&radius=2", sep="")  
res <- POST(url,  
  body=sdf)  
response <- rawToChar(res$content)  
fp_hex <- strsplit(response, "\n")[[1]][4]  
(fp_hex <- strsplit(fp_hex, "\t")[[1]][1])
```

```
fp_hex <- substring(fp_hex, seq(1,nchar(fp_hex),2), seq(2,nchar(fp_hex),2))  
fp_bin <- unlist(lapply(paste("0x",fp_hex,sep=""),function(x) rawToBits(as.raw(x))))  
(fp_bin <- as.numeric(fp_bin))
```

```
(fp <- paste(fp_bin, collapse=""))
```

```
# Using the tools of the fingerprint package  
fp_obj <- fps.lf(strsplit(response, "\n")[[1]][4])  
fp <- new("fingerprint", nbit = 1024,  
  bits = as.numeric(fp_obj[[2]]),  
  folded = FALSE,  
  provider = gsub("#software=", "",  
    strsplit(response, "\n")[[1]][3], fixed=T),
```

```
name = fp_obj[[1]],  
misc = list()  
str(fp)
```

```
as.character(fp)
```

When comparing the RDKit's Morgan fingerprints with the ECFP/FCFP fingerprints, it is important to remember that the name of ECFP/FCFP fingerprints are suffixed with the diameter of the atom environments considered, while the Morgan Fingerprints take a radius parameter (e.g., the second argument "2" of `GetMorganFingerprintAsBitVect()` in the above code cell). The Morgan fingerprint generated above (with a radius of 2) is comparable to the ECFP4 fingerprint (with a diameter of 4).

Exercise 1b: For the molecules below, generate the 512-bit-long Morgan Fingerprint.

- Search for the compounds by name and get their SMILES strings.
- Generate the molecular fingerprints from the SMILES strings.
- Print the generated fingerprints.

```
synonyms <- c('diphenhydramine', 'cetirizine', 'fexofenadine', 'loratadine')
```

```
# Write your code here
```

1-(3) Path-Based Fingerprints

Path-based fingerprints are also hashed fingerprints. They are generated by enumerating linear fragments of a given length and hashing them into a fixed-length bitstring. An example of this, is the standard fingerprint in CDK. Another example is the RDKit's topological fingerprint.

In CDK, size and depth allow specifying fingerprint and maximum path size, used for constructing the fingerprint. They default to 1024 bits and depth 6.

```
fp <- get.fingerprint(mol[[1]], type = 'standard',  
  fp.mode = 'bit', size = 128, depth= 4, verbose=FALSE)  
str(fp)
```

```
as.character(fp)
```

```
nchar(as.character(fp))
```

```
length(fp@bits)/length(fp)
```

```
fp <- get.fingerprint(mol[[1]], type = 'standard',  
  fp.mode = 'bit', size = 2048, depth= 7, verbose=FALSE)  
str(fp)
```

```
as.character(fp)
```

```
nchar(as.character(fp))
```

The PubChem Fingerprint is a 881-bit-long binary fingerprint (ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf). Similar to the MACCS keys, it uses a pre-defined fragment dictionary. The PubChem fingerprint for each compound in PubChem can be downloaded from PubChem. However, because they are base64-encoded, they should be decoded into binary bitstrings or bitvectors.

```
if(!require("jsonlite")) {
  install.packages(("jsonlite"), repos="https://cloud.r-project.org/",
    quiet=TRUE, type="binary")
  library("jsonlite")
}
```

```
pcfps <- 'AADcYBgAAAAAAAAAAAAAAAAAAAAAAwAAAAAAAAABAAAAGAAAAAAAAACACAEEAwAIAAAI  
CAAAGAAAIiAAAAIgIIckAERCAIAaggAAIiAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=='  
pcfps_raw <- base64_dec(pcfps)  
rawToBits(base64_dec("AAD"))
```

```
sapply(base64_dec("AAAD"),function(x) paste(as.numeric(rawToBits(x)),collapse=""))
```

```
pcfps_bin <- sapply(pcfps_raw,
  function(x) paste(as.numeric(rev(rawToBits(x))), collapse=""))
pcfps_bin <- substring(paste(pcfps_bin, collapse=""), 33, 913)
nchar(pcfps_bin)
```

pcfps_bin

The generated bitstring can be converted to a bitvector that can be used for molecular similarity computations (to be discussed in the next section).

```
(binvect <- as.numeric(unlist(strsplit(pcfps_bin, ""))))
```

```
cids <- c(54454, # Simvastatin (Zocor)
          54687, # Pravastatin (Pravachol)
          60823, # Atorvastatin (Lipitor)
          446155, # Fluvastatin (Lescol)
          446157, # Rosuvastatin (Crestor)
          5282452, # Pitavastatin (Livalo)
          97938126) # Lovastatin (Altoprev)
```

Let's get the SMILES strings from PubChem, generate Mol objects from them, and draw their chemical structures.

```
prolog <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
str_cid <- paste(as.character(cids), collapse=",")
url <- paste(prolog, "/compound/cid/", str_cid, "/property/isomericsmiles/txt", sep="")
smiles <- readLines(url)
```

```
if(!require("httr")) {
  install.packages(("httr"), repos="https://cloud.r-project.org/",
    quiet=TRUE, type="binary")
  library("httr")
}
if(!require("jsonlite")) {
  install.packages(("jsonlite"), repos="https://cloud.r-project.org/",
    quiet=TRUE, type="binary")
  library("jsonlite")
}
if(!require("png")) {
  install.packages(("png"), repos="https://cloud.r-project.org/",#
    quiet=TRUE, type="binary")
  library("png")
}
if(!require("grid")) {
  install.packages(("grid"), repos="https://cloud.r-project.org/",
    quiet=TRUE, type="binary")
  library("grid")
}
if(!require("gridExtra")) {
  install.packages(("gridExtra"), repos="https://cloud.r-project.org/",
    quiet=TRUE, type="binary")
  library("gridExtra")
}
```

```
url_img <- paste("https://www.ebi.ac.uk/chembl/api/utils/smiles2image",
  "?size=300&engine=rdkit", sep="")
res <- POST(url_img,
  body=list(smiles=paste(smiles, collapse="\n")))
img <- readPNG(res$content, native=TRUE)
grid.arrange(rasterGrob(img))
```

Now generate MACCS keys for each compound.

```
mols <- parse.smiles(smiles)
fps <- sapply(mols,
  function(x) get.fingerprint(x, type="maccs"))
fps_bin <- sapply(fps, as.character)
fps_bin
```

Now let's compute the pair-wise similarity scores among them. To make higher scores easier to find, they are indicated with the "*" character(s).

```
dfDistance <- data.frame(cid1 = numeric(0), cid2 = numeric(0),
  score = numeric(0))

for(i in 1:(length(cids)-1)) {
  for(j in (i+1):length(cids)) {
    score <- distance(fps[[i]], fps[[j]], method="tanimoto")
    dfDistance <- rbind(dfDistance,
      list(cid1 = cids[i], cid2 = cids[j],
        score = score))
  }
}

dfDistance$value <- ""
dfDistance$value[dfDistance$score >= .55] <- "*"
dfDistance$value[dfDistance$score >= .65] <- "***"
dfDistance$value[dfDistance$score >= .75] <- "****"
dfDistance$value[dfDistance$score >= .85] <- "*****"

dfDistance
```

By default, the similarity score is generated using the Tanimoto equation. `fingerprint::distance` also supports other similarity metrics, including Dice, Cosine, Russel, SOkal-Michener (also known as simple matching), Kulczynski, McConnaughey, and Tversky. The definition of these metrics is available at the LibreTexts page (<https://bit.ly/2kx9NCd>).

```
print(paste("Tanimoto: ",
  distance(fps[[1]], fps[[2]], method="tanimoto")))
```

```
print(paste("Dice: ",
  distance(fps[[1]], fps[[2]], method="dice")))
```

```
print(paste("Cosine: ",
  distance(fps[[1]], fps[[2]], method="cosine")))
```

```
print(paste("Simple: ",
  distance(fps[[1]], fps[[2]], method="simple")))
```

```
print(paste("McConnaughey: ",
  distance(fps[[1]], fps[[2]], method="mcconnaughey")))
```

```
contTable <- as.matrix(table(seq(length(fps[[1]])) %in% fps[[1]]@bits,
  seq(length(fps[[2]])) %in% fps[[2]]@bits))
a <- contTable[2,2]
```

```
b <- contTable[2,1]
c <- contTable[1,2]
d <- contTable[1,1]
dist <- (a^2 - b * c)/((a + b) * (a + c))

print(paste("McConnaughey: ",
  dist))
```

The Tversky score is an asymmetric similarity measure, and its computation requires the weightings of the two molecules being compared.

```
for(i in 0:10) {
  print(paste("Tversky (alpha = ", i * 0.1, ", beta = ", 1-i * .1, ") = ",
    distance(fps[[1]], fps[[2]], a= i * .1, b = 1 - i * .1,
    method="tversky"),
    sep = ""))
}
```

Exercise 2a: Compute the Tanimoto similarity scores between the seven compounds used in this section, using the PubChem fingerprints

- Download the PubChem Fingerprint for the seven CIDs.
- Convert the downloaded fingerprints into bit vectors.
- Compute the pair-wise Tanimoto scores using the bit vectors

```
# Write your code here
```

3. Interpretation of similarity scores

Using molecular fingerprints, we can compute the similarity scores between molecules. However, how should these scores be interpreted? For example, the Tanimoto score between CID 60823 and CID 446155 is computed to be 0.662, but does it mean that the two compounds are similar? How similar is similar? The following analysis would help answer these questions.

Step 1. Randomly select 1,000 compounds from PubChem and download their SMILES strings.

```
prolog <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
set.seed(0)
cid_max <- 138962044 # The maximum CID in PubChem as of September 2019

cids <- sample(seq(cid_max),1000)

chunk_size <- 100
num_chunks <- ceiling(length(cids) / chunk_size)

smiles = character(length(cids))

for(i in seq(num_chunks)) {
  print(i)

  idx1 <- chunk_size * (i - 1) + 1
```



```
idx2 <- chunk_size * i
str_cids <- paste(cids[idx1:idx2], collapse=",")
url <- paste(prolog, "/compound/cid/", str_cids, "/property/isomericsmiles/txt", sep="")
smiles[idx1:idx2] <- readLines(url)

Sys.sleep(0.5)
}
print("Done!")
```

```
print(paste("# Number of SMILES :", length(na.omit(smiles))))
```

Step 2. Generate the MACCSKeys for each compound.

```
mols <- parse.smiles(smiles)
fps <- lapply(mols, function(x) get.fingerprint(x, type = 'maccs',
  fp.mode = 'bit', verbose=FALSE))

str(fps[[1]])
```

```
print(paste("Number of compounds:", length(mols)))
```

```
print(paste("Number of Fingerprints:", length(fps)))
```

Step 3. Compute the Tanimoto scores between compounds.

```
print(paste("Number of compound pairs:", (length(fps) * (length(fps) - 1))/2))
```

```
scores <- numeric((length(fps) * (length(fps) - 1))/2)

k <- 1
for(i in 1:(length(fps)-1)) {
  for(j in (i+1):length(fps)) {
    scores[k] <- distance(fps[[i]], fps[[j]], method="tanimoto")
    k <- k + 1
  }
}

summary(scores)
```

Step 4. Generate a histogram that shows the distribution of the pair-wise scores.

```
if(!require("tidyverse")) {
  install.packages(("tidyverse"), repos="https://cloud.r-project.org/",
    quiet=TRUE, type="binary")
  library("tidyverse")
}
```

```
ggplot(NULL, aes(x=scores, y=..density..)) +  
  geom_histogram(fill="lightgrey", color="black", binwidth=.05) +  
  geom_density()
```

```
dfScoresTab <- data.frame(limit = 0:20*0.05,  
  countLT = sapply(0:20*0.05, function(x) sum(scores>=x)))  
dfScoresTab$propLT <- dfScoresTab$countLT / length(scores)  
dfScoresTab
```

```
print(paste("Average:", sum(scores)/length(scores)))
```

From the distribution of the similarity scores among 1,000 compounds, we observe the following:

- If you randomly select two compounds from PubChem, the similarity score between them (computed using the Tanimoto equation and MACCS keys) is ~0.35 on average.
- About %5 of randomly selected compound pairs have a similarity score greater than 0.55.
- About %1 of randomly selected compound pairs have a similarity score greater than 0.65.

If two compounds have a Tanimoto score of 0.35, it is close to the average Tanimoto score between randomly selected compounds and there is a 50% chance that you will get a score of 0.35 or greater just by selecting two compounds from PubChem. Therefore, it is reasonable to consider the two compounds are not similar.

The Tanimoto index may have a value ranging from 0 (for no similarity) to 1 (for identical molecules) and the midpoint of this value range is 0.5. Because of this, a Tanimoto score of 0.55 may not sound great enough to consider two compounds to be similar. However, according to the score distribution curve generated here, only ~5% of randomly selected compound pairs will have a score greater than this.

In the previous section, we computed the similarity scores between some cholesterol-lowering drugs, and CID 60823 and CID 446155 had a Tanimoto score of 0.662. Based on the score distribution curve generated in the second section, we can say that the probability of two randomly selected compounds from PubChem having a Tanimoto score greater than 0.662 is less than 1%.

The following code cell demonstrates how to find an appropriate similarity score threshold above which a given percentage of the compound pairs will be considered to be similar to each other.

```
# to find a threshold for top 3% compound pairs (i.e., 97% percentile)  
quantile(scores, .97)
```

Exercise 3a: In this exercise, we want to generate the distribution of the similarity scores among 1,000 compounds randomly selected from PubChem, using different molecular fingerprints and similarity metrics. For molecular fingerprints, use the following:

- PubChem Fingerprint - MACCS keys - Morgan Fingerprint (ECFP4 analogue, 1024-bit-long)

For similarity metrics, use the following: - Tanimoto similarity - Dice similarity - Cosine similarity

As a result, a total of 9 distribution curves need to be generated.

Here are additional instructions to follow: - When generating the histograms, bin the scores from 0 to 1 with an increment of 0.01. - For each distribution curve, determine the similarity score threshold so that 1% of the compound pairs have a similarity score greater than or equal to this threshold. - Use RDKit to generate the MACCS keys and Morgan fingerprint and download the PubChem fingerprints from PubChem. - For reproducibility, use random.seed(2019) before you generate random CIDs.

Step 1: Generate 1,000 random CIDs, download the isomeric SMILES for them, and create the RDKit mol objects from the downloaded SMILES strings.

```
# Write your code here
```

Step 2: Generate the fingerprints, compute the similarity scores, determine similarity thresholds, and make histograms.

```
# Write your code here
```

6.5: R Assignment is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.