Exam 1

Alex Whang

2024-10-7

```r
# Load packages
if(!require("pacman")) {
  install.packages("pacman")
  library(pacman)
}
```

```
## Loading required package: pacman
```

```r
p_load(RecordLinkage, blink, knitr, textreuse, tokenizers, devtools, cora,
ggplot2, dplyr, numbers)
data(cora) # load the cora data set
data(cora_gold)
data(cora_gold_update)
```

Question 1

a. False
b. True
c. True
d. False
e. True
f. False

Question 2

a.

```r
# Create sets
one <- c(1, 2, 3, 4)
two <- c(2, 3, 5, 7)
three <- c(2, 4, 6)

# Computer Jaccard similarity for each pair
jaccard_similarity(one, two)
```

```
## [1] 0.3333333
```

```r
jaccard_similarity(two, three)
```

```
## [1] 0.1666667
```

```r
jaccard_similarity(one, three)
```

```
## [1] 0.4
```

The Jaccard similarity between the first and second set is $\frac{1}{3}$.

The Jaccard similarity between the second and third set is $\frac{1}{6}$.

The Jaccard similarity between the first and third set is $\frac{2}{5}$.

b.

```r
# Initialize sentences
one <- "The plane was ready for touch down"
two <- "The quarterback scored a touchdown"

# Find set of 9-shingles for each sentence ignoring blanks
token.1 <- tokenize_character_shingles(one, 9)
token.2 <- tokenize_character_shingles(two, 9)
```

c. Storing data in the form of a characteristic matrix is impractical because they can become very large when dealing with large datasets. A characteristic matrix can contain many repeated values and leads to unnecessary data duplication. Additionally, operations on large matrices can be computationally expensive.

d. The minhash function is used to get an approximation of Jaccard similarity using signatures. The hash function is first applied to the rows and then takes the minimum hash value for each column, reducing the matrix's dimensionality.

e. Minhashing approximates the Jaccard similarity of two sets. The probability that two columns have the same minhash value is equal to the Jaccard similarity of the two sets that they are supposed to represent.

f. From the permuted matrix, the first row of S is $3, 2, 1, 2$. This was done by identifying the first 1 in each column of the permuted matrix.

Question 3

a.

```r
# Merge cora_gold with cora_gold_update once on id1 then once on id2
check <- merge(cora_gold, cora_gold_update, by.x = "id1", by.y = "cora_id")
check <- merge(check, cora_gold_update, by.x = "id2", by.y = "cora_id", all.x = TRUE, suffixes = c("1",

# Check for mismatches by comparing unique ids
mismatches <- check[check$unique_id1 != check$unique_id2, ]
```

There are errors in `cora_gold_update`. The id pair $(113, 122)$ in `cora_gold` means that both ids should map to the same unique id in `cora_gold_update`. However, id 113 maps to unique id 16 while id 122 maps to unique id 17.

b.

```r
# Calculate number of rows in mismatches
nrow(mismatches)
```

```
## [1] 21856
```

There are 21856 that exist between `cora_gold` and `cora_gold_update`.

c.

(i)

The below code was taken from hw-3-partial-solutions-steorts.Rmd

```r
n <- nrow(cora)

# initialize each row to be its own "group"
unique_ids <- 1:n

# update "groups" as iterate over rows
# know that assigning the new ID will be a valid group since id1 < id2 for all rows
# the unique_id for indexes corresponding to cora_ids will be the same if they
#   are really the same entity
for (row in 1:nrow(cora_gold)) {
  id1 <- cora_gold[row,1]
  id2 <- cora_gold[row,2]
  unique_ids[id2] <- unique_ids[id1]
}

# count number of unique entities
n_unique <- length(unique(unique_ids))

# map the raw group IDs to a simpler group IDs format
# allows the unique_id to only go from 1 to n_unique instead of 1 to n
map <- setNames(1:n_unique, unique(unique_ids))

# apply the mapping
unique_ids <- map[as.character(unique_ids)]

# create dataframe storing unique IDs
unique_df <- data.frame(
  cora_id = 1:n,
  unique_id = unique_ids
)

# VERIFY: the number of matches in cora_gold equals the number of matches we
#   would expect from these unique IDs
# the result of this should be TRUE
sum(choose(table(unique_df$unique_id),2)) == nrow(cora_gold)
```

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

The number of matches in `cora_gold` equals the number of matches we would expect from these unique IDs.

(ii)

```
# Merge cora_gold with cora_gold_update once on id1 then once on id2
check_update <- merge(cora_gold, unique_df, by.x = "id1", by.y = "cora_id")
check_update <- merge(check_update, unique_df, by.x = "id2", by.y = "cora_id", all.x = TRUE, suffixes =

# Check for mismatches by comparing unique ids
mismatches_update <- check_update[check_update$unique_id1 != check_update$unique_id2, ]

# Calculate number of rows in mismatches
nrow(mismatches_update)
```

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

The number of inconsistencies is now 0.