Exam-1

2024-10-02

1. (6 points, 1 point each). True or False. Answer True or False for each part below. You do not need to explain your rationale for credit.

a. The four stages of the data cleaning pipeline are as follows: 1. schema alignment, principal components analysis, entity resolution, and canonicalization. False [Citation: Data-Cleaning-Part-I Slide 4]

b. One application of entity resolution is predicting if two finger prints are matches (or not matches). True [Citation: Data-Cleaning-Part-I Slide 4]

c. One challenge of the data cleaning pipeline is scaling to large databases. True [Citation: Preliminaries Slide 15]

d. Suppose a data set has a total of R records. In the blocking stage, there are R total record comparisons that can be made. False

e. The output of entity resolution can tells us if record pairs are matches or non-matches. It does not tells us what record is the most representative entity (person) in the data set. True

f. The goal of schema alignment is to remove duplicate entities from large databases. False [Citation: Data-Cleaning-Part-I Slide 4]

2. 6 points (1 point each).

a. Compute the Jaccard similarity for each pair of the following three sets: S1 = {1, 2, 3, 4}, S2 = {2, 3, 5, 7}, and S3 = {2, 4, 6} Do this using R and do not compute this by hand.

```
library(textreuse)
library(tokenizers)
```

```
##
## Attaching package: 'tokenizers'
```

```
## The following objects are masked from 'package:textreuse':
##
##     tokenize_ngrams, tokenize_sentences, tokenize_skip_ngrams,
##     tokenize_words
```

```
S1 <- c(1, 2, 3, 4)
S2 <- c(2, 3, 5, 7)
S3 <- c(2, 4, 6)
```

```
jaccard_similarity(S1,S2)
```

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

```
jaccard_similarity(S1,S3)
```

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

```
jaccard_similarity(S2,S3)
```

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

Citation: Probabilistic-Blocking-PartI Slide 17-19

b. Consider two sentences: "The plane was ready for touch down" and "The quarterback scored a touchdown" Find the set of 9-shingles for each sentence, where you should ignore blanks (or spaces). Do this using R and do not compute this by hand.

```
token_1 <- tokenize_character_shingles("Theplanewasreadyfortouchdown", n=9)
unlist(token_1)
```

```
##  [1] "theplanew" "heplanewa" "eplanewas" "planewasr" "lanewasre" "anewasrea"
##  [7] "newasread" "ewasready" "wasreadyf" "asreadyfo" "sreadyfor" "readyfort"
## [13] "eadyforto" "adyfortou" "dyfortouc" "yfortouch" "fortouchd" "ortouchdo"
## [19] "rtouchdow" "touchdown"
```

```
token_2 <- tokenize_character_shingles("Thequarterbackscoredatouchdown", n=9)
unlist(token_2)
```

```
##  [1] "thequarte" "hequarter" "equarterb" "quarterba" "uarterbac" "arterback"
##  [7] "rterbacks" "terbacksc" "erbacksco" "rbackscor" "backscore" "ackscored"
## [13] "ckscoreda" "kscoredat" "scoredato" "coredatou" "oredatouc" "redatouch"
## [19] "edatouchd" "datouchdo" "atouchdow" "touchdown"
```

Citation: Probabilistic-Blocking-PartI Slide 17-19

c. Explain why we do not store data in the form of a characteristic matrix (for practical purposes).

A characteristic matrix can be inefficient to store. Minhashing and LSH helps with reducing the size while while preserving most of the information.

d. Explain what the minhash function does to the characterisic matrix. Hint: Think about the rows versus the columns.

We use minhash function to create signature matrix. During minhash process, we first permute the rows of characteristic matrix m times, going through each column of permuted matrix, and populate the signature matrix row-wise, with the row index from the first 1 value found in the column.

Citation: Probabilistic Blocking Part II Slide 14

e. Explain (in words) the connection between minhashing and the Jaccard similarity of the sets that are minhashed.

Jaccard similarity compares the ratio of the size of two set's intersection to the size of their union. Minhashing approximates the similarity - the probability of two sets having the same minmash value is proportional to their Jaccard similarity.

Citation: Probabilistic Blocking Part II Slide 16

f. Consider the following permuted matrix, where the columns are records and the rows are shingles:

```
s1 <- c(0, 0, 1, 0)
s2 <- c(0, 1, 1, 1)
s3 <- c(1, 0, 0, 0)
s4 <- c(1, 1, 0, 1)
permuted_matrix <- rbind(s1, s2 , s3, s4)
permuted_matrix
```

```
##    [,1] [,2] [,3] [,4]
## s1    0    0    1    0
## s2    0    1    1    1
## s3    1    0    0    0
## s4    1    1    0    1
```

Find the first row of the signature matrix S. You may answer this either by hand or using R.

{3,2,1,2} For the first column, the first 1 is in row 3. For the second column, the first 1 is in row 2. For the third column, the first 1 is in row 1. For the fourth column, the first 1 is in row 2.

3. (10 points) Consider the cora data set that we considered in class and in homework

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

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

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

Observe that cora_gold contains record pairs that refer to the same person (entity). The data set cora_gold_update contains two columns, cora_id and unique_id. The first column, cora_id, refers to the row number in the cora data set. The second column, unique_id, refers to a unique identifier for each record. For instance, the first six records correspond to the unique identifer 1.

Unfortunately, upon further inspection, there are errors in cora_gold_update.

a. (2 points) Validate (empirically) that there are errors in cora_gold_update. Hint: Check to see which id pairs in cora_gold do not map to the same unique id in cora_gold_update.

```
#If the id paris do not map to the same unique id
#then they are not the same entity, hence error
#merge cora_gold_update with cora_gold on cora_id to get unique id for id1
merged_cora_gold <- merge(cora_gold, cora_gold_update, by.x = "id1",
                          by.y = "cora_id", all.x = TRUE)
#merge again to get unique id for id2
merged_cora_gold <- merge(merged_cora_gold, cora_gold_update, by.x = "id2",
                          by.y = "cora_id", all.x = TRUE, suffixes = c("_id1", "_id2"))

errors <- merged_cora_gold[merged_cora_gold$unique_id_id1 != merged_cora_gold$unique_id_id2, ]
head(errors)
```

```
##     id2 id1 unique_id_id1 unique_id_id2
## 771 122 113            16            17
## 772 122 108            16            17
## 773 122 107            16            17
## 774 122 109            16            17
## 775 122 112            16            17
## 776 122 117            16            17
```

There are errors in cora_gold_update because some of the matches have different unique ids.

b. (2 points) Calculate how many inconsistencies exist between cora_gold and cora_gold_update based upon part (a).

```
# Count the number of inconsistencies
nrow(errors)
```

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

There are 21856 inconsistencies.

c. (6 points) Using cora_gold, write code such that cora_gold_update that maintains the same mapping.

```
cora_gold_update_2 <- cora_gold_update
# To make sure we propagate corrected id across the data set we need a loop to make
# sure that there is no more inconsistency
# Initialize a variable to track if any changes were made in an iteration
changed <- TRUE

# Repeat the process until no more inconsistencies are found
while (changed) {
  # Merge cora_gold with cora_gold_update_2 to get current unique ids for id1 and id2
  merged_cora_gold_2 <- merge(cora_gold, cora_gold_update_2, by.x = "id1",
                              by.y = "cora_id", all.x = TRUE)
  merged_cora_gold_2 <- merge(merged_cora_gold_2, cora_gold_update_2, by.x = "id2",
                              by.y = "cora_id", all.x = TRUE, suffixes = c("_id1", "_id2"))

  # Identify inconsistencies where unique_id of id1 and id2 are different
  errors <- merged_cora_gold_2[merged_cora_gold_2$unique_id_id1 != merged_cora_gold_2$unique_id_id2, ]

  # If no errors are found, set changed to FALSE to exit the loop
  if (nrow(errors) == 0) {
    changed <- FALSE
  } else {
    # For each error, update the unique_id to the minimum of the two for both id1 and id2
    for (i in 1:nrow(errors)) {
      id1 <- errors$id1[i]
      id2 <- errors$id2[i]
      correct_unique_id <- min(errors$unique_id_id1[i], errors$unique_id_id2[i])

      # Update the unique_id for all records with cora_id equal to id1 or id2
      cora_gold_update_2$unique_id[
        cora_gold_update_2$cora_id %in% c(id1, id2)
      ] <- correct_unique_id
    }
  }
}
```

(i) Validate that all matches have the same unique id. Provide a Boolean result.

```
#If the id paris do not map to the same unique id
#then they are not the same entity, hence error
#merge cora_gold_update with cora_gold on cora_id to get unique id for id1
merged_cora_gold_2 <- merge(cora_gold, cora_gold_update_2, by.x = "id1",
                            by.y = "cora_id", all.x = TRUE)
#merge again to get unique id for id2
merged_cora_gold_2 <- merge(merged_cora_gold_2, cora_gold_update_2, by.x = "id2",
                            by.y = "cora_id", all.x = TRUE, suffixes = c("_id1", "_id2"))
#check if all matches have same id
all(merged_cora_gold_2$unique_id_id1 == merged_cora_gold_2$unique_id_id2)
```

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

All of the matches have the same id. (ii) Validate that the number of inconsistencies is now 0.

```r
errors_2 <- merged_cora_gold_2[merged_cora_gold_2$unique_id_id1 != merged_cora_gold_2$unique_id_id2, ]
#count number of inconsistencies
nrow(errors_2)
```

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