Optical character recognition (OCR)

Contents

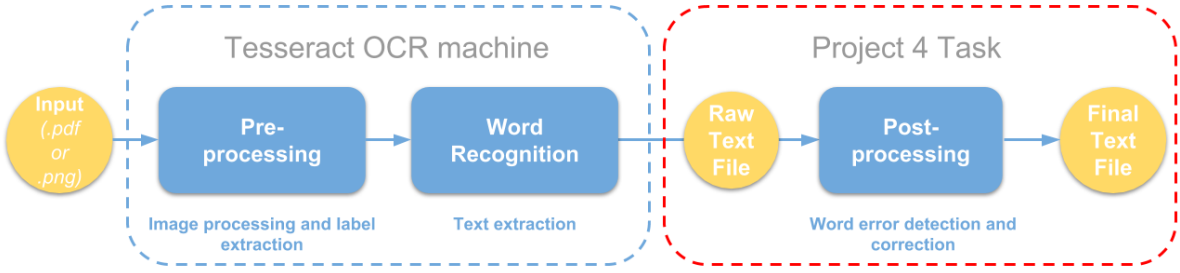
Jing Wu

GU4243/GR5243: Applied Data Science

Introduction

Optical character recognition (OCR) is the process of converting scanned images of machine printed or handwritten text (numerals, letters, and symbols), into machine readable character streams, plain (e.g. text files) or formatted (e.g. HTML files). As shown in Figure 1, the data *workflow* in a typical OCR system consists of three major stages:

- Pre-processing
- OCR character recognition
- Post-processing



We have processed raw scanned images through the first two steps are relying on the Tessearct OCR machine. R package tutorial can be found here.

In this project, we are going to **focus on the third stage – post-processing**, which includes two tasks: *error detection* and *error correction*.

Step 1 - Load library and source code

```
if (!require("devtools")) install.packages("devtools")
if (!require("pacman")) {
  ## devtools is required
  library(devtools)
  install_github("trinker/pacman")
}

if (!require("textreg")) install.packages("textreg")

#pacman::p_load(knitr, readr, stringr, tesseract, vecsets)

library(stringr)
library(tm)
library(dplyr)
library(tidytext)
# library(broom)
library(topicmodels)
library(ggplot2)
library(reshape2)

# source('../lib/ifCleanToken.R')
```

Step 2 - read the files and conduct Tesseract OCR

Although we have processed the Tesseract OCR and save the output txt files in the `data` folder, we include this chunk of code to clarify the whole pipeline.

```
for(i in c(1:length(file_name_vec))){
  current_file_name <- sub(".txt","",file_name_vec[i])
  ## png folder is not provided on github (the code is only on demonstration purpose)
  current_tesseract_txt <- tesseract::ocr(paste("../data/png/",current_file_name,".png",sep=""))

  ### clean the tessetact text (separate line by "\n", delete null string, transter to lower case)
  clean_tesseract_txt <- strsplit(current_tesseract_txt,"\n")[[1]]
  clean_tesseract_txt <- clean_tesseract_txt[clean_tesseract_txt!=""]

  ### save tesseract text file
  writeLines(clean_tesseract_txt, paste("../data/tesseract/",current_file_name,".txt",sep=""))
}
```

Step 3 - Error detection

3.1 - Data Preparation

```
# read files

truthPath = "../data/ground_truth/"
ocrPath = "../data/tesseract/"
correctPath = "../output/correct/"

truthFiles <- list.files(path=truthPath, pattern="*.txt", full.names=TRUE, recursive=FALSE)
ocrFiles <- list.files(path=ocrPath, pattern="*.txt", full.names=TRUE, recursive=FALSE)

if(length(truthFiles) != length(ocrFiles)) stop("the number of ground truth files is not equal to the nu

n <- length(truthFiles) # number of files

# select train + validation and testing data 80% and 20%
set.seed(1984)
training <- sample(1:n, round(0.8*n))
testing <- (1:n)[!1:n %in% training]

# set training data
truthTrain <- truthFiles[training]
ocrTrain <- ocrFiles[training]

# set testing data
truthTest <- truthFiles[testing]
ocrTest <- ocrFiles[testing]
```

Now, we are ready to conduct post-processing, based on the Tessearct OCR output. First of all, we need to detect errors, or *incorrectly processed words* – check to see if an input string is a valid dictionary word or if its n-grams are all legal.

3.2 - Source the files

```
source("../lib/Dictionary.R") # dictionary for groundtruth dataset
source("../lib/OCRText.R")    # bag of word of tesseract ocr text
source("../lib/Digram.R")     # digram for all words in groundtruth
source("../lib/detect.R")     # detection function
source("../lib/wordsTable.R") # function to locate the words' position in which line and document
```

3.3 - Create bag of words and digram array for groungtruth and tesseract data

- step 1: Create the dictionary from 100 groudtruth files | ("dict")
- step 2: Create the list of incorrect words from tesseract OCR train dataset | ("OCRtext") and give the position of word in which line and which document
- step 3: Create the digram array(36*36) including numbers(0-9) and letters(a-z), from all length of words in 100 groundtruth files | ("digram")

```
# For our dictionary (derived from groundtruth), we use all 100 documents instead of training dataset f
```

```
if(doDic) {
  dict <- dictionary(truthFiles)
} else {
  load("../output/dict.RData")
}

# give the position for word in OCR trainingtext
if(doOCRText){
  OCRText <- wordsTable(ocrTrain, "ocrTrainTable")
} else{
  load("../output/ocrTrainTable.RData")
}

if(doOCRWordsTable) OCRTestTable <- wordsTable(ocrTest, "OCRTestTable") else load("../output/OCRTestTabl

# create the digram for all words in dictionary of groundtruth
if(doDigram == T){
  digram <- Digram(dict)
}else{
  load("../output/digrams.RData")
}
```

3.4 - Implement detection algorithm

- step 0: Get list of words from OCR Train dataset, it only includes the words that length is less than 25
- step 1: Detect the OCR test dataset, if there is an error, label the word with 1

```
# OCRText is the bag of word derived from function in OCRText.R which using OCRTrain data
# OCRTestTable <- OCRTestTable[nchar(OCRTestTable[,1]) < 25,]

if(doDetect){
  OCRTestErrors <- detect(OCRTestTable, digram)
  save(OCRTestErrors,file="../output/OCRTestErrors.RData")
} else {
  load("../output/OCRTestErrors.RData")
}
```

Step 4 - Error correction

Given the detected word error, in order to find the best correction, we need to generate the candidate corrections: a dictionary or a database of legal n-grams to locate one or more potential correction terms. Then we need invoke some lexical-similarity measure between the misspelled string and the candidates or a probabilistic estimate of the likelihood of the correction to rank order the candidates.

The reference paper (C5) is on topic models.

4.1 - Data Preparation

```
source("../lib/confusionMatrix.R")
source('../lib/outputtext.R')
# source("../lib/CandidateWords.R")
```

4.2 Implementation correction algorithm

- step 1: Train the LDA topic model using groundtruth training dataset("truthTrain")
- step 2: $P(t_k)$ Get probability of topics in each document
- step 3: $P(w|t_k)$ Get probability probability of words in the given topic

```
# step 1: train word from groundtruth with topic modeling, get the probability of each topic in each do


if(doConfusion) confMat <- commonMistakes(truthTrain, ocrTrain) else load("../output/confusionMatrix.RDa

if(doLDA) l <- topicModelling(truthTrain) else load("../output/LDA.RData") # here we choose #oftopic=30

if(doCorrect){
  wordTopic <- tidy(l, matrix = "beta")
  docTopic <- tidy(l, matrix = "gamma")
  wordTopic <- acast(wordTopic, term~topic, value.var="beta")
  docTopic <- acast(docTopic, document~topic, value.var="gamma")

  ocrcorrect <- correct(OCRTestErrors, docTopic, wordTopic)

  outputtext(ocrcorrect) # generates a text file for each document
}
```

Step 5 - Performance measure

```
if(doTruthWordsTable) truthTestTable <- wordsTable(truthTest, "truthTestTable") else load("../output/tru

if(doOCRWordsTable) OCRTestTable <- wordsTable(ocrTest, "OCRTestTable") else load("../output/OCRTestTabl

correctTest <- list.files(path=correctPath, pattern="*.txt", full.names=TRUE, recursive=FALSE)
if(doCorrectWordsTable) correctTestTable <- wordsTable(correctTest, "correctTestTable") else load("../ou

first50 <- cbind(head(truthTestTable[(nchar(OCRTestTable[,1]) > 1)[1:100] , 1], 50),
                 head(OCRTestTable[(nchar(OCRTestTable[,1]) > 1)[1:100],1], 50),
                 head(correctTestTable[,1], 50))

colnames(first50) <- c("truth", "tesseract", "processed tesseract")

write.csv(first50, file = "../output/compare50.csv", col.names = T)
```

```
## Warning in write.csv(first50, file = "../output/compare50.csv", col.names =
## T): attempt to set 'col.names' ignored
```

The two most common OCR accuracy measures are precision and recall. Both are relative measures of the OCR accuracy because they are computed as ratios of the correct output to the total output (precision) or input (recall). More formally defined,

$$\text{precision} = \frac{\text{number of correct items}}{\text{number of items in OCR output}}$$

$$\text{recall} = \frac{\text{number of correct items}}{\text{number of items in ground truth}}$$

where *items* refer to either characters or words, and ground truth is the original text stored in the plain text file.

Both *precision* and *recall* are mathematically convenient measures because their numeric values are some decimal fractions in the range between 0.0 and 1.0, and thus can be written as percentages. For instance, recall is the percentage of words in the original text correctly found by the OCR engine, whereas precision is the percentage of correctly found words with respect to the total word count of the OCR output. Note that in the OCR-related literature, the term OCR accuracy often refers to recall.

Here, we only finished the **word level evaluation** criterions, you are required to complete the **letter-level** part.

Word-level Precision

```
recallTess <- c()
precisionTess <- c()
recallCorr <- c()
precisionCorr <- c()

for(i in 1:length(unique(OCRTestTable[,4]))){
  truth <- truthTestTable[truthTestTable[,4] == i,1]
  tess <- OCRTestTable[OCRTestTable[,4] == i,1]
  corr <- correctTestTable[correctTestTable[,4] == i,1]
  intersectTess <- vecsets::vintersect(tolower(truth), tolower(tess))
  intersectCorr <- vecsets::vintersect(tolower(truth), tolower(corr))
  recallTess[i] <- length(intersectTess)/length(truth)
  precisionTess[i] <- length(intersectTess)/length(tess)
  recallCorr[i] <- length(intersectCorr)/length(truth)
  precisionCorr[i] <- length(intersectCorr)/length(corr)
}


library(ggridges)
```

```
##
## Attaching package: 'ggridges'
```

```
## The following object is masked from 'package:ggplot2':
##
##     scale_discrete_manual
```

```
library(ggplot2)

precRecall <- cbind(c(recallTess, precisionTess, recallCorr, precisionCorr),
                c(rep("recall (truth VS tesseract)", length(recallTess)),
                  rep("precision (truth VS tesseract)", length(precisionTess)),
                  rep("recall (truth VS processed)", length(recallCorr)),
                  rep("precision (truth VS processed)", length(precisionCorr))
                  ))

precRecall <- as.data.frame(precRecall)
names(precRecall) <- c("precision", "test")

precRecall$precision <- as.numeric(levels(precRecall$precision)[precRecall$precision])
```

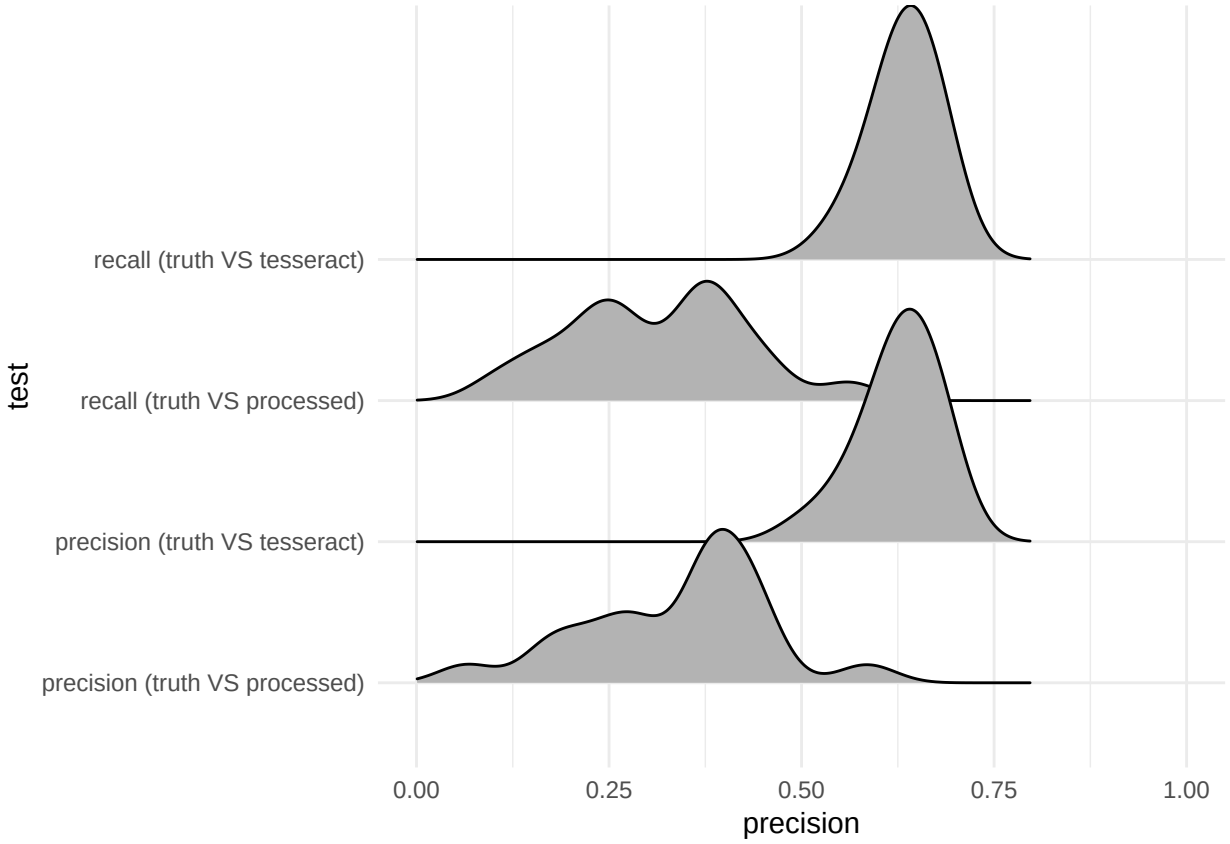
```
densitiesPrecRecall <- ggplot(precRecall, aes(x = precision, y = test)) +
  geom_density_ridges() + scale_x_continuous(limits = c(0, 1)) +
  theme_minimal()

ggsave(filename="../output/precRecall.pdf", plot=densitiesPrecRecall)
```

```
## Saving 6.5 x 4.5 in image
```

```
## Picking joint bandwidth of 0.0368
```

```
densitiesPrecRecall
```

```
## Picking joint bandwidth of 0.0368
```



Character-level Precision

```
if(doPerfTess) precisionTess <- performance(truthTestTable, OCRTestTable, "precisionTess") else load(".

if(doPerfTessProcess) precisionProcessed <- performance(truthTestTable, correctTestTable, "precisionPro

library(ggridges)
library(ggplot2)

precisions <- cbind(c(precisionTess, precision),
                c(rep("tesseract", length(precisionTess)),
                  rep("processed", length(precision))))
head(precisions)
```

```
##      [,1]                [,2]
## [1,] "0.714285714285714" "tesseract"
## [2,] "0"                 "tesseract"
## [3,] "1"                 "tesseract"
## [4,] "1"                 "tesseract"
## [5,] "0.75"              "tesseract"
## [6,] "1"                 "tesseract"
```

```
precisions <- as.data.frame(precisions)
names(precisions) <- c("precision", "text")

mean(precisionTess, na.rm = T)
```

```
## [1] 0.9241983
```

```
precisions$precision <- as.numeric(levels(precisions$precision)[precisions$precision])

densities <- ggplot(precisions, aes(x = precision, y = text)) +
  geom_density_ridges() + scale_x_continuous(limits = c(0, 1)) +
  theme_minimal()

ggsave(filename="../output/densities.pdf", plot=densities)
```

```
## Saving 6.5 x 4.5 in image
```

```
## Picking joint bandwidth of 0.0569
```

```
## Warning: Removed 42 rows containing non-finite values
## (stat_density_ridges).
```

References

1. Borovikov, E. (2014). *A survey of modern optical character recognition techniques.* arXiv preprint arXiv:1412.4183.pdf (This paper is the source of our evaluation criterion)
2. Kukich, K. (1992). *Techniques for automatically correcting words in text.* Acm Computing Surveys (CSUR), 24(4), 377-439. pdf (This paper is the benchmark review paper)