Optical character recognition (OCR)

Contents

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
- Word 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.

BUT this is not the FOCUS of this project!!!

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

```
project_path = "~/Desktop/GR5243 Applied Data Science/Project 4/Spring2019-Proj4-grp8"

if (!require("devtools")) install.packages("devtools")
```

```
## Loading required package: devtools
```

```
if (!require("pacman")) {
  ## devtools is required
  library(devtools)
```

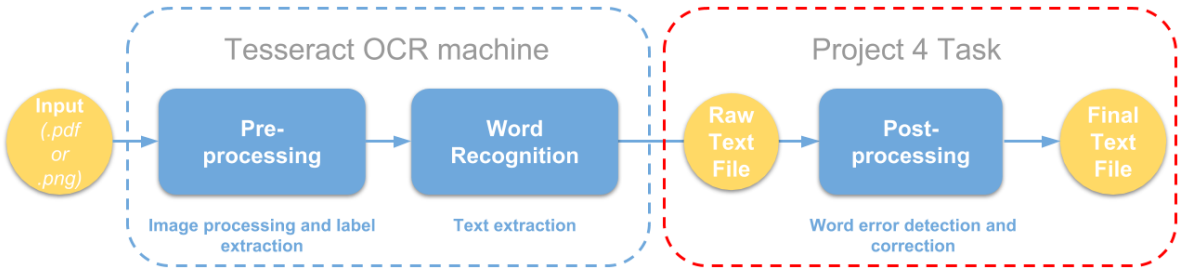

Figure 1:

```
  install_github("trinker/pacman")
}
```

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

```
pacman::p_load(knitr, readr, stringr, tesseract, vecsets)
source(paste0(project_path,'/lib/ifCleanToken.R'))
file_name_vec <- list.files(paste0(project_path,"/data/ground_truth")) #100 files in total
```

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 in order to make clear the whole pipeline to you.

```
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 for demonstration purpose)
  current_tesseract_txt <- tesseract::ocr(paste(project_path,"/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(project_path,"/data/tesseract/",current_file_name,".txt",sep="")
}
```

Step 3 - Error detection

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.

The referenced papers are:

1. Rule-based techniques

- rules are in the section 2.2

2. Letter n-gram

- focus on positional binary digram in section 3-a.error detection

3. Probabilistic techniques – SVM garbage detection

- features are in section 5 (you can choose not to implement 'Levenshtein distance' feature)

In this statercode, we implement the first three rules in the first paper – rule based techniques, as an example.

Read files

```
library(stringr)
library(tm)
```

```
## Loading required package: NLP
```

```
library(tidytext)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(topicmodels)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------- tidyverse 1.2.
```

```
## v ggplot2 3.1.0     v tidyr   0.8.3
## v tibble  2.0.1     v purrr   0.3.1
## v ggplot2 3.1.0     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------ tidyverse_conflicts()
## x ggplot2::annotate() masks NLP::annotate()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
```

```
library(knitr)
library(readr)

#project_path = "/Users/huyiyao/Desktop/ADS/Spring2019-Proj4-grp8/"
project_path = "~/Desktop/GR5243 Applied Data Science/Project 4/Spring2019-Proj4-grp8"
true_path = "/data/ground_truth/"
OCR_path = "/data/tesseract/"

ground_truth = list.files(path=paste0(project_path,true_path), pattern="*.txt", full.names=TRUE, recurs
```

```
OCR = list.files(path=paste0(project_path,OCR_path), pattern="*.txt", full.names=TRUE, recursive=FALSE)

n = length(ground_truth);n # number of files
```

```
## [1] 100
```

```
source(paste0(project_path,"/lib/Detection.R"))

build_dic = F
build_dig = F
detect_ocr = F

if(build_dic){
  dictionary = Truth_Extract(ground_truth)
  save(dictionary, file = paste0(project_path,"/output/GroundTruthWords.Rdata"))
} else {
  load(paste0(project_path,"/output/GroundTruthWords.Rdata"))
}


if(build_dig){
  dic_digrams = Digrams(dictionary)
  save(dic_digrams, file = paste0(project_path,"/output/digrams.Rdata"))
 # write_csv(as.data.frame(dictionary),
  #          path  = paste0(project_path,"/output/dictionary.csv"))
} else{
  load(paste0(project_path,"/output/digrams.Rdata"))
}
#length(dic_digrams)
#dic_digrams[[1]]
#dic_digrams[[2]]


if(detect_ocr){
  detection_ocr = Detection(OCR, digrams = dic_digrams)
  save(detection_ocr, file = paste0(project_path,"/output/detection_ocr.Rdata"))
} else{
  load(paste0(project_path,"/output/detection_ocr.Rdata"))
}

#length(detection_ocr)
#head(detection_ocr[[1]],20)
#tail(detection_ocr[[70]],20)
```

Step 4 - Error correction

Given the detected word error, in order to find the best correction, we need to generating 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 referenced papers are:

1. Letter n-gram

- focus on section 3-b.error correction

2. Supervised model – correction regressor
3. Probability scoring without context

- focus on section 3

4. Probability scoring with contextual constraints

- focus on section 5

5. Topic models

Here, in our code, we just simply remove the detected-errors.

```
#tesseract_delete_error_vec <- tesseract_vec[tesseract_if_clean] #1095
```

Step 5 - Performance measure

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 **character-level** part.

```
ground_truth_vec <- str_split(paste(current_ground_truth_txt, collapse = " ")," ")[[1]] #1078

## Here, we compare the lower case version of the tokens
old_intersect_vec <- vecsets::vintersect(tolower(ground_truth_vec), tolower(tesseract_vec)) #607
new_intersect_vec <- vecsets::vintersect(tolower(ground_truth_vec), tolower(tesseract_delete_error_vec))

OCR_performance_table <- data.frame("Tesseract" = rep(NA,4),
                                    "Tesseract_with_postprocessing" = rep(NA,4))
row.names(OCR_performance_table) <- c("word_wise_recall","word_wise_precision",
                                      "character_wise_recall","character_wise_precision")
OCR_performance_table["word_wise_recall","Tesseract"] <- length(old_intersect_vec)/length(ground_truth_v
OCR_performance_table["word_wise_precision","Tesseract"] <- length(old_intersect_vec)/length(tesseract_v
OCR_performance_table["word_wise_recall","Tesseract_with_postprocessing"] <- length(new_intersect_vec)/l
OCR_performance_table["word_wise_precision","Tesseract_with_postprocessing"] <- length(new_intersect_vec
```

```
kable(OCR_performance_table, caption="Summary of OCR performance")
# ----------------------process data----------------------
wrong.words <- list()
# path <- "/Users/tianchenwang/Git/Spring2019-Proj4-grp8/data/ground_truth/"
path = paste0(project_path, "/data/ground_truth/")
groundtruth.name <- list.files(path)
groundtruth.docs <- list()
for(i in 1:length(groundtruth.name)){
  groundtruth.docs[i][[1]] <- read_file(paste0(path, groundtruth.name[i]))
}

# get tidy tibble data.
ground_truth <- NULL
for(i in 1:length(groundtruth.docs)){
  # tem <- tibble(text = str_split(groundtruth.docs[i][[1]], "\n")[[1]], docid = i)
  tem <- tibble(text = gsub("\n", " ", groundtruth.docs[i][[1]])[[1]], docid = i)
  ground_truth <- rbind(ground_truth, tem)
}

# token
ground_truth_tokenized <- ground_truth %>%
  group_by(docid) %>%
  # mutate(linenumber = row_number()) %>%
  ungroup() %>%
  unnest_tokens(word, text)

ground_truth_tokenized <- ground_truth_tokenized %>%
  count(docid, word, sort = TRUE)

ground_truth_dtm <- ground_truth_tokenized %>%
  cast_dtm(docid, word, n)

# ----------------------topic modeling----------------------
K <- 30
control_list <- list(
  seed = list(1,2,3,4,5),
  nstart = 5,
  best = TRUE,
  burnin = 1000,
  iter = 600,
  thin = 100
)

#Remove comment and run this line to run code for LDA, or just load the output we have already run
run_lda <- F
if(run_lda){
  gd_LDA <- LDA(ground_truth_dtm, K, method = "Gibbs", control = control_list)
  save(gd_LDA, file = paste0(project_path, "/output/gd_LDA.Rdata"))

} else {
  load(paste0(project_path,"/output/gd_LDA.Rdata"))
  }
```

```
word_prob <- gd_LDA %>%
  tidy(matrix = "beta") %>%
  arrange(topic, beta)

word_prob %>%
  group_by(topic) %>%
  top_n(5, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)
```

```
## # A tibble: 254 x 3
##    topic term         beta
##    <int> <chr>       <dbl>
##  1     1 cma       0.0540
##  2     1 its       0.0251
##  3     1 concealed 0.0249
##  4     1 mca       0.0126
##  5     1 which     0.00836
##  6     2 and       0.0582
##  7     2 cma       0.0460
##  8     2 the       0.0407
##  9     2 chemical  0.0235
## 10     2 of        0.0231
## # ... with 244 more rows
```

```
# path <- "/Users/tianchenwang/Git/Spring2019-Proj4-grp8/output/gd_LDA.Rdata"
# save(gd_LDA, file =path)

# candidate word list
# path <- "/Users/tianchenwang/Git/Spring2019-Proj4-grp8/output/"
# # load detection ...
load(paste0(project_path, "/output/detection_ocr.Rdata"))

incorect_word <- lapply(detection_ocr, function(x){
  x[,1][x[,2] == "1"]
})

# save candidate words
# save(incorect_word, file = paste0(path, "incorrect_words.Rdata") )

# -------------------correct word-----------------------
all_correct_words <- read_csv(paste0(project_path, "/output/dictionary.csv"))
```

```
## Parsed with column specification:
## cols(
##   dictionary = col_character()
## )
```

```
all_correct_words$l <- sapply(all_correct_words$dictionary, str_length)

set <- list(all = 2,
            sub = 2)
# # c is from the same length words
# agrep("apple", c("appre", "apply", "avxsa", "appple"), value = TRUE, max = set)
```

```
# run THIS!!
run_candidate_words = F
if(run_candidate_words){

  candidate_words <- lapply(incorect_word, function(x){
    contianer <- list()
    for(i in 1:length(x)){
      dic <- all_correct_words[all_correct_words$l == str_length(x[i]), 1]
      contianer[[x[i]]] <- agrep(x[i], dic$dictionary, value = TRUE,
                                 max = set)
   }
    contianer
  })

} else {load(paste0(project_path, "/output/candidate_words.Rdata"))}
# save(candidate_words, file = paste0(project_path, "candidate_words.Rdata") )
# load(paste0(project_path, "candidate_words.Rdata"))

# --------------------load confusion matrix-----------------------
# path <- "/Users/tianchenwang/Git/Spring2019-Proj4-grp8/data/confusion_matrix/"
path <- paste0(project_path, "/data/confusion_matrix/")
sub_matrix <- read.csv(paste0(path, "sub_matrix.csv"), row.names = 1)

# get probability sub_matrix
sub_probmatrix <- t(t(sub_matrix) / apply(sub_matrix, 2, sum))

# -------------------- words correction -----------------------
lda_model <- gd_LDA
corrected_word <- list()
run_corrected_word = F
if(run_corrected_word){
  for(i in 1:length(incorect_word)){
  corrected_word[[i]] = list()
  # doc position in lda model
  doc_pos <- which(lda_model@documents == as.character(i))
  # topic prob for the doc
  topic_prob <- lda_model@gamma[doc_pos, ]
  for (current in incorect_word[[i]]) {

    candidates <- candidate_words[[i]][[current]]
    if(length(candidates) > 0){
      # get current incorrect word letters
      current_letters <- str_split(current, "")[[1]]

      # given topics word probability
      scores <- sapply(candidates, function(x){
        col_id <- which(lda_model@terms == x)
        sum(lda_model@beta[,col_id] * topic_prob)
      })
      # compute letter probs
      letter_probs <- sapply(candidates, function(x){
        candidate_letters <- str_split(x, "")[[1]]
        temp <- 1
```

```
        for (k in 1:length(candidate_letters)){
          temp <- temp*sub_probmatrix[candidate_letters[k],current_letters[k]]
        }
        temp
      })
      # update final score
      scores <- scores * letter_probs

      # return the highest score candidate for the incorrect current
      correct <- candidates[sample(which(scores == max(scores)), 1)]
      corrected_word[[i]][[current]] <- correct
    }
    #
  }
}
  #save(corrected_word,file = paste0(project_path,"/output/corrected_word.Rdata"))
} else {load(paste0(project_path,"/output/corrected_word.Rdata"))}


# correction part ends!
```

performance measure

```
# functions
#function to divide text into smaller strings between each unique intersection word
divide_string <- function(vec, intersect_vec){
  position <- c(0,which(vec %in% intersect_vec), length(vec))
  string_list = list()
  for(i in 1:(length(position)-1)){
    str = character(0)
    if(position[i+1] - position[i] > 1 ) {
      for(j in (position[i]+1) : (position[i+1]-1)){
        str = c(str, vec[j])
      }
     }

  string_list[[i]] = str
  }
  return(string_list)
}

# function to input two lists of strings, count number of error characters
count_error_char <- function(list1, list2){
  if(length(list1) == length(list2)){
    no_of_error <- 0
    for(i in 1:length(list1)){
      no_of_error <- no_of_error + sum(diag(adist(list1[[i]], list2[[i]])))
    }
  }
  else{ no_of_error <- NA }

  return(no_of_error)
}
```

```
# record all pairs of one word
pairsdoc<-function(word){
  pairs=c()
n=nchar(word)
if(n>1){
m=0
for (i in 1:(n-1)){
for(j in 2:n){
  if (i<j)
    {m=m+1
  pairs[m]<-c(paste(substr(word,i,i),substr(word,j,j),step=""))
  }}
}
}
else{pairs<-c("a")}
return(pairs)
}

# compare 2 words and return error rate
pairs_word_error<-function(word1,word2){
pairs1<-pairsdoc(word1)
pairs2<-pairsdoc(word2)
l<-max(nchar(word1),nchar(word2))
if(l>1){
ratio<-1-length(vecsets::vintersect(pairs1,pairs2))/(l*(l-1)/2)}
else {ratio=0}
return(ratio)
}

#give two lists, return sum of error rate of all words
sum_pairs_error <- function(list1, list2){
 if(length(list1) == length(list2)){
    sum_error <- 0
    for(i in 1:length(list1)){
      if(min(length(list1[[i]]),length(list2[[i]]))>0){
      for(j in 1:min(length(list1[[i]]),length(list2[[i]]))){
     sum_error <- sum_error + pairs_word_error(list1[[i]][j],list2[[i]][j])
    }}}
 }
    else{ sum_error <- NA }
  return(sum_error)
}
```

```
# ------------------- Tesseract with postprocessing ----------------------
#load(paste0(project_path,"/output/corrected_word.Rdata"))
file_name_vec <- list.files(paste0(project_path,"/data/ground_truth"))
n <- length(file_name_vec)
corrected_text <- vector("list", length = (n+1))
run_corrected_text = F
if(run_corrected_text){
  for(i in 1:n){
    current_file_name <- sub(".txt","",file_name_vec[i])
    current_tesseract_txt <-
      readLines(paste0(project_path, paste("/data/tesseract/",current_file_name,".txt",sep="")), warn=F
```

```
    tesseract_vec <- str_split(paste(current_tesseract_txt, collapse = " ")," ")[[1]]
    tesseract_vec <- tesseract_vec %>% removePunctuation() %>% tolower()

    #loop through tesseract_vec, create key for each word, store the word as value
    dic <- list()
    for(word in tesseract_vec){
      if(is.null(dic[[word]])){
        dic[[word]] = word
      }
    }

    #loop through corrected_word, for each key in dic, substitute the value
    for(key in names(corrected_word[[i]])){
      for(dic_key in names(dic)){
        if(key == dic_key){
          dic[[dic_key]] = corrected_word[[i]][[key]]
          break
        }
      }
    }

    #loop through tesseract_vec, create a vector containing word after correction
    for(words in tesseract_vec){
      corrected_text[[i]] <- c(corrected_text[[i]], dic[[words]])
    }
  }
  save(corrected_text, file = paste0(project_path,"/output/corrected_text.Rdata"))
} else {
  load(paste0(project_path,"/output/corrected_text.Rdata"))
  }


# ------------------- performance evaluation -----------------------
#load(paste0(project_path,"/output/corrected_text.Rdata"))
source(paste0(project_path,"/lib/Performance.R"))
word_recall_before <- numeric(n)
word_pres_before <- numeric(n)
word_recall_post <- numeric(n)
word_pres_post <- numeric(n)
char_recall_before <- numeric(n)
char_pres_before <- numeric(n)
char_recall_post <- numeric(n)
char_pres_post <- numeric(n)
pairs_recall_before <- numeric(n)
pairs_pres_before <- numeric(n)
pairs_recall_post <- numeric(n)
pairs_pres_post <- numeric(n)

for(i in 1:n){
  current_file_name <- sub(".txt","",file_name_vec[i])

  current_ground_truth_txt <-
    readLines(paste0(project_path, paste("/data/ground_truth/",current_file_name,".txt",sep="")), warn=
```

```
  current_tesseract_txt <-
    readLines(paste0(project_path, paste("/data/tesseract/",current_file_name,".txt",sep="")), warn=FALS

  ground_truth_vec <- str_split(paste(current_ground_truth_txt, collapse = " "),pattern=" ")[[1]]
  tesseract_vec <- str_split(paste(current_tesseract_txt, collapse = " "),pattern=" ")[[1]]
  ground_truth_vec <- ground_truth_vec %>% removePunctuation() %>% tolower()
  tesseract_vec <- tesseract_vec %>% removePunctuation() %>% tolower()
  tesseract_post_vec <- corrected_text[[i]]

  ### word level ###
  old_intersect_vec<- vecsets::vintersect(ground_truth_vec, tesseract_vec)
  new_intersect_vec <- vecsets::vintersect(ground_truth_vec, tesseract_post_vec)

  word_recall_before[i] <- length(old_intersect_vec)/length(ground_truth_vec)
  word_pres_before[i] <- length(old_intersect_vec)/length(tesseract_vec)
  word_recall_post[i] <- length(new_intersect_vec)/length(ground_truth_vec)
  word_pres_post[i] <- length(new_intersect_vec)/length(tesseract_post_vec)


  ### character level ###
  # extract non-duplicate words
  truth_unique <- ground_truth_vec[which(!(duplicated(ground_truth_vec)|
                                           duplicated(ground_truth_vec, fromLast=TRUE)))]

  tesseract_unique <- tesseract_vec[which(!(duplicated(tesseract_vec)|
                                            duplicated(tesseract_vec, fromLast=TRUE)))]

  tesseract_post_unique <- tesseract_post_vec[which(!(duplicated(tesseract_post_vec)|
                                            duplicated(tesseract_post_vec, fromLast=TRUE)))]

  #intersection of unique words
  char_old_intersect_vec<- vecsets::vintersect(truth_unique, tesseract_unique)
  char_new_intersect_vec<- vecsets::vintersect(truth_unique, tesseract_post_unique)

  #divide into string segments
  truth_string <- divide_string(ground_truth_vec, char_old_intersect_vec)
  tesseract_string <- divide_string(tesseract_vec, char_old_intersect_vec)
  truth_post_string <- divide_string(ground_truth_vec, char_new_intersect_vec)
  tesseract_post_string <- divide_string(tesseract_post_vec, char_new_intersect_vec)

  #count error characters
  old_error <- count_error_char(truth_string, tesseract_string)
  new_error <- count_error_char(truth_post_string, tesseract_post_string)

  char_recall_before[i] <- 1 - old_error/sum(nchar(ground_truth_vec))
  char_pres_before[i] <- 1 - old_error/sum(nchar(tesseract_vec))
  char_recall_post[i] <- 1 - new_error/sum(nchar(ground_truth_vec))
  char_pres_post[i] <- 1 - new_error/sum(nchar(tesseract_vec))

#------------------pairs level-----------------------------------
pairs_recall_before[i]<-1-sum_pairs_error(truth_string, tesseract_string)*(1/length(ground_truth_vec))
pairs_pres_before[i]<-1-sum_pairs_error(truth_string, tesseract_string)*(1/length(tesseract_vec))
pairs_recall_post[i] <- 1-sum_pairs_error(truth_post_string, tesseract_post_string)*(1/length(ground_tr
```

```
pairs_pres_post[i]<- 1-sum_pairs_error(truth_post_string, tesseract_post_string)*(1/length(tesseract_ve

}



#------------------measure_lsit-----------------
measure_list<-cbind.data.frame( word_recall_before,word_pres_before,word_recall_post,word_pres_post,cha
```

```
# ------------------- performance table ------------------------
OCR_performance_table <- data.frame("Tesseract" = rep(NA,6),
                                    "Tesseract_with_postprocessing" = rep(NA,6),"Improvement"= rep(NA,6)
row.names(OCR_performance_table) <- c("word_wise_recall","word_wise_precision","character_wise_recall",

OCR_performance_table["word_wise_recall","Tesseract"] <-
  mean(measure_list$word_recall_before)
OCR_performance_table["word_wise_precision","Tesseract"] <-
  mean(measure_list$word_pres_before)
OCR_performance_table["word_wise_recall","Tesseract_with_postprocessing"]<-
  mean(measure_list$word_recall_post)
OCR_performance_table["word_wise_precision","Tesseract_with_postprocessing"]<-
  mean(measure_list$word_pres_post)
OCR_performance_table["word_wise_recall","Improvement"] <-OCR_performance_table[1,2]-OCR_performance_tab
OCR_performance_table["word_wise_precision","Improvement"]<-OCR_performance_table[2,2]-OCR_performance_


OCR_performance_table["character_wise_recall","Tesseract"] <-
  mean(measure_list$char_recall_before)
OCR_performance_table["character_wise_precision","Tesseract"] <- mean(measure_list$char_pres_before)
OCR_performance_table["character_wise_recall","Tesseract_with_postprocessing"] <- mean(measure_list$cha
OCR_performance_table["character_wise_precision","Tesseract_with_postprocessing"] <- mean(measure_list$
OCR_performance_table["character_wise_recall","Improvement"] <-OCR_performance_table[3,2]-OCR_performanc
OCR_performance_table["character_wise_precision","Improvement"]<-OCR_performance_table[4,2]-OCR_performa

OCR_performance_table["pairs_wise_recall","Tesseract"] <-mean(measure_list$pairs_recall_before)
OCR_performance_table["pairs_wise_precision","Tesseract"] <- mean(measure_list$pairs_pres_before)
OCR_performance_table["pairs_wise_recall","Tesseract_with_postprocessing"] <- mean(measure_list$pairs_re
OCR_performance_table["pairs_wise_precision","Tesseract_with_postprocessing"] <- mean(measure_list$pairs
OCR_performance_table["pairs_wise_recall","Improvement"] <-OCR_performance_table[5,2]-OCR_performance_ta
OCR_performance_table["pairs_wise_precision","Improvement"]<-OCR_performance_table[6,2]-OCR_performance_

kable(OCR_performance_table, caption="Summary of OCR performance")
```

Table 1: Summary of OCR performance

	Tesseract	Tesseract_with_postprocessing	Improvement
word_wise_recall	0.6311040	0.7078918	0.0767878
word_wise_precision	0.6224899	0.6996062	0.0771162
character_wise_recall	0.7234524	0.7253635	0.0019111
character_wise_precision	0.7173900	0.7196558	0.0022658
pairs_wise_recall	0.7127195	0.7189846	0.0062651
pairs_wise_precision	0.7182570	0.7240757	0.0058187

Besides the above required measurement, you are encouraged to explore more evaluation measurements. Here are some related references:

1. Karpinski, R., Lohani, D., & Belaïd, A. *Metrics for Complete Evaluation of OCR Performance.* pdf

- section 2.1 Text-to-Text evaluation

2. Mei, J., Islam, A., Wu, Y., Moh'd, A., & Milios, E. E. (2016). *Statistical learning for OCR text correction.* arXiv preprint arXiv:1611.06950. pdf

- section 5, separate the error detection and correction criterions

3. Belaid, A., & Pierron, L. (2001, December). *Generic approach for OCR performance evaluation.* In Document Recognition and Retrieval IX (Vol. 4670, pp. 203-216). International Society for Optics and Photonics. pdf

- section 3.2, consider the text alignment

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)