Testing Report

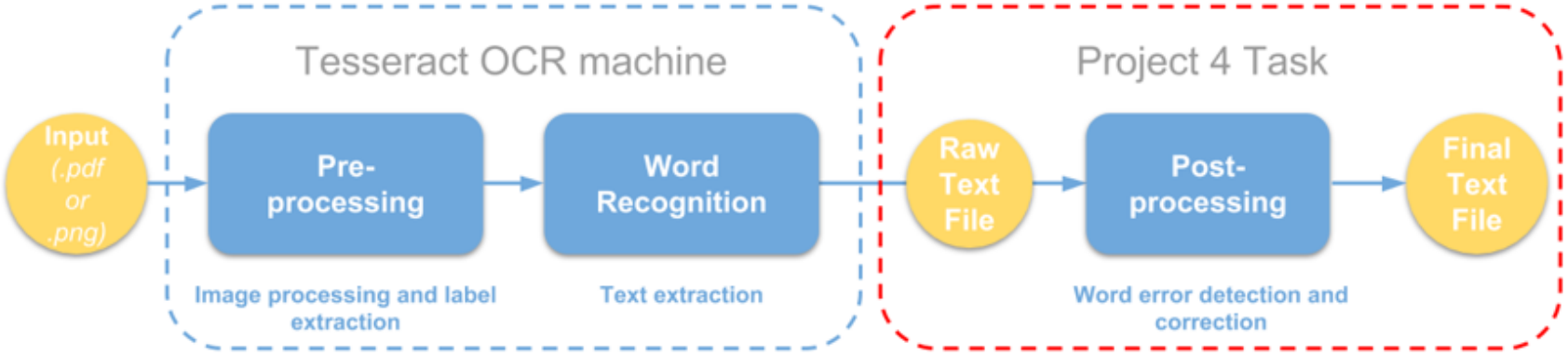
GU4243/GR5243: Applied Data Science

Group 3

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



To recieve input data for the project, raw scanned images were processed through the first two steps are relying on the Tessearct OCR machine (https://en.wikipedia.org/wiki/Tesseract_(software)). R package tutorial can be found here (https://www.r-bloggers.com/the-new-tesseract-package-high-quality-ocr-in-r/).

This project is aimed to **focus on the third stage – post-processing**, which includes two tasks: *error detection* and *error correction*. There is a total of 100 pairs of files representing ground truth and text output of OCR Machine.

Step 1 - Load libraries and source code

```
if (!require("devtools")) install.packages("devtools")
if (!require("pacman")) {
  ## devtools is required
  library(devtools)
  install_github("trinker/pacman")
}
pacman::p_load(knitr, readr, stringr, tesseract, vecsets)
library(stringr)
library(topicmodels)
library(tm)
library(tidytext)
library(dplyr)
library(ldatuning)
library(SnowballC)
library(NLP)
library(tm)
library(Hmisc)
library(R.oo)
source('../lib/Detection.R')
source('../lib/candidate_vector.R')
source('../lib/dtm.R')
source('../lib/candidate_word_score.R')
source('../lib/confusion_matrix.R')
source('../lib/get_letterlist.R')
```

Hide

```
#outputs steps 2,3
#load("../output/tesseract_error.Rdata") #ocr.error
#load("../output/tesseract_correct.Rdata") #ocr.correct
#load("../output/tesseract_full_data.Rdata") #ocr.all.text
#load("../output/ground_truth_full_data.Rdata") #ground.truth.all.text
#load("../output/candidates_list.Rdata") #candidates.list
#load("../output/candidates_list_nonempty.Rdata") #candidates.list.nonempty
#load("../output/ocr_true_error_nonempty.Rdata") #ocr.true.error.nonempty
#load("../output/ocr_final_output.Rdata") #ocr.final.output
```

Step 2 - Error detection

Error detection is the first step of post-processing, based on the Tessearct OCR output. First of all, we need to detect errors, or *incorrectly processed words*. For that we use Rule-based techniques (http://webpages.ursinus.edu/akontostathis/KulpKontostathisFinal.pdf), specifically rules are in the section 2.2. The code takes 100 documents, creates a single big corpus, which is splitted into two: one contains correct text elements, the other contains erroneous text elements. Only inuque tockens are left for alanysis to reduce the amount of computation.

Hide

Hide

```
file.name.vector <- list.files("../data/ground_truth")
ground.truth.all = c()
for (doc in 1:length(file.name.vector)){
  ground.truth.text.vector = readLines(paste("../data/ground_truth/", file.name.ve
ctor[doc],sep=""))
  ground.truth.all = c(ground.truth.all, ground.truth.text.vector)
  ground.truth.all.text = paste(ground.truth.all, collapse = " ")
}
save(ground.truth.all.text, file = "../output/ground_truth_full_data.Rdata")
ocr.all = c()
for (doc in 1:length(file.name.vector)){
  ocr.text.vector = readLines(paste("../data/tesseract/", file.name.vector[doc],se
p =""))
  ocr.all = c(ocr.all, ocr.text.vector)
  ocr.all.text = paste(ocr.all, collapse = " ")
}
save(ocr.all.text, file = "../output/tesseract_full_data.Rdata")
ocr.tokens = str_split(ocr.all.text," ")
ocr.tokens.unique = unique(ocr.tokens[[1]])
ocr.boolean.if.clean = lapply(ocr.tokens.unique,Detection)
ocr.boolean.if.clean = unlist(ocr.boolean.if.clean)
ocr.error = ocr.tokens.unique[!ocr.boolean.if.clean] #6072
ocr.correct = ocr.tokens.unique[ocr.boolean.if.clean]
length(ocr.error)
```

```
[1] 6072
```

Hide

Hide

```
save(ocr.error, file = "../output/tesseract_error.Rdata")
save(ocr.correct, file = "../output/tesseract_correct.Rdata")
```

Step 3 - Error correction

To correct errors detected in the previous step, we followed the paper Topic models (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4377099). Error correction algorithm considered in the paper consists of two models: a topic model that provides information about word probabilities and OCR model that represents the probability of character errors. For each error word we compute a vector of candidates to substitute the erroneous word and pick the one that provides the best score according to the formula proposed in the paper.

Step 3.1 Generate vector of candidate words for each word detected as incorrect.

Hide

Hide

```
load("../output/tesseract_error.Rdata") #ocr.error
load("../output/tesseract_correct.Rdata") #ocr.correct
load("../output/tesseract_full_data.Rdata") #ocr.all.text
load("../output/ground_truth_full_data.Rdata") #ground.truth.all.text
dictionary = unique(unlist(strsplit(ground.truth.all.text, split = " ")))
#Removing intersection between dictionary and error vector
not.in.dictionary.boolean = which(ocr.error %nin% dictionary)
ocr.true.error = ocr.error[not.in.dictionary.boolean] #5331
length(ocr.true.error)
```

```
[1] 5331
```

Hide

Hide

```
#using of grid provides many-fold time optimivation vs for-loop (0.07sec vs 10sec/
error.word)
tic  = Sys.time()
candidates.list = lapply(ocr.true.error, get.candidate.vector, dictionary = dictio
nary) #dictionary_letters =  dictionary_letters if option above is active
toc = Sys.time()
toc-tic
```

```
Time difference of 11.33428 mins
```

Hide

Hide

```
ocr.true.error.nonempty = ocr.true.error[lapply(candidates.list, length)>0]
length(ocr.true.error.nonempty) #3105
```

```
[1] 3105
```

Hide

Hide

```
candidates.list.nonempty = candidates.list[lapply(candidates.list, length)>0]
length(candidates.list.nonempty) #3105
```

```
[1] 3105
```

Hide

Hide

```
save(candidates.list, file = "../output/candidates_list.Rdata")
save(candidates.list.nonempty, file = "../output/candidates_list_nonempty.Rdata")
save(ocr.true.error.nonempty, file = "../output/ocr_true_error_nonempty.Rdata")
```

Step 3.2 Generate confusion matrix.

Hide

Hide

```
load("../output/tesseract_error.Rdata") #ocr.error
load("../output/tesseract_correct.Rdata") #ocr.correct
load("../output/tesseract_full_data.Rdata") #ocr.all.text
load("../output/ground_truth_full_data.Rdata") #ground.truth.all.text
load("../output/candidates_list.Rdata") #candidates.list
load("../output/candidates_list_nonempty.Rdata") #candidates.list.nonempty
load("../output/ocr_true_error_nonempty.Rdata") #ocr.true.error.nonempty
letterlist = get_letterlist(ocr.true.error.nonempty, candidates.list.nonempty)
```

Hide

Hide

```
ground_true_loc <- "../data/ground_truth/"
tesseract_loc <- "../data/tesseract/"
file_names <- list.files(ground_true_loc)
truth_list <- paste0(ground_true_loc, file_names)
ocr_list <- paste0(tesseract_loc, file_names)
```

Hide

Hide

```
#cm = confusion_count_num("../data/ground_truth/group1_00000005.txt", "../data/tes
seract/group1_00000005.txt")
confusion.prob = confusion_count_num(truth_list, ocr_list, letterlist)
save(confusion.prob, file = "../output/confusion_prob.Rdata")
#print(cm)
```

Step 3.3 Choose optimal number of Topics for LDA topic model.

Hide

Hide

```
gt <- Corpus(DirSource("../data/ground_truth"), readerControl = list(language = "e
nglish"))
gt <- tm_map(gt, PlainTextDocument)
```

```
transformation drops documents
```

Hide

Hide

```
gt <- tm_map(gt, stripWhitespace)
gt <- tm_map(gt, tolower)
```

```
transformation drops documents
```

Hide

Hide

```
gt <- tm_map(gt,removeNumbers)
```

```
transformation drops documents
```

Hide

Hide

```
gt <- tm_map(gt,removePunctuation)
```

```
transformation drops documents
```

Hide

Hide

```
gt <- tm_map(gt, removeWords, stopwords("english"))
```

```
transformation drops documents
```

Hide

Hide

```
tm_map(gt, stemDocument)
```

```
transformation drops documents
```

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 100
```

Hide

Hide

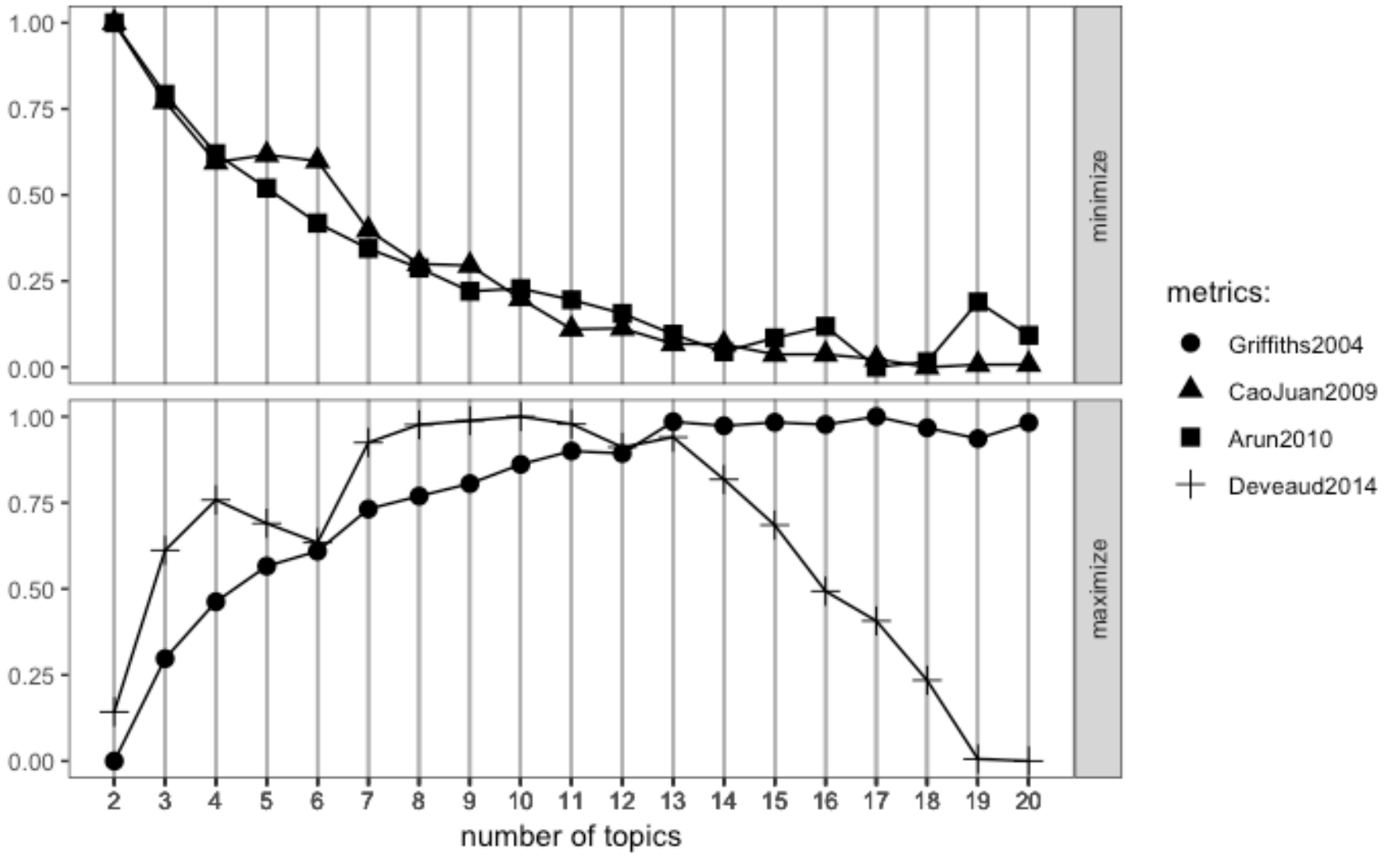
```
dtm <- DocumentTermMatrix(gt)
result <- FindTopicsNumber(
  dtm,
  topics = seq(from = 2, to = 20, by = 1),
  metrics = c("Griffiths2004", "CaoJuan2009", "Arun2010", "Deveaud2014"),
  method = "Gibbs",
  control = list(seed = 77),
  mc.cores = 2L,
  verbose = TRUE
)
```

```
fit models... done.
calculate metrics:
  Griffiths2004... done.
  CaoJuan2009... done.
  Arun2010... done.
  Deveaud2014... done.
```

Hide

Hide

```
#save(result, file = "../output/topic_number_finder.Rdata")
#load("../output/topic_number_finder.Rdata") #result
FindTopicsNumber_plot(result)
```



Step 3.4 Train LDA topic model.

Single LDA model is trained using all text data out of 100 documents provided.

Hide

Hide

```
n.topics = 13 #optimal number of clusters based on the result above
dtm.ground.truth = get.dtm(ground.truth.all.text)
lda = LDA(dtm.ground.truth, k = n.topics, method = "Gibbs", control = list(seed =
2019))
```

Step 3.5 Application of LDA topic model.

Probability of topics $P(t_k)$ computed by applying the trained topic model to the correctly recognized words in the document. Will be applied as constant for all candidate words.

Hide

Hide

```
dtm.ocr = get.dtm(ocr.all.text)
probability.topic.term = posterior(lda, dtm.ocr)
probability.topic = probability.topic.term$topics #P(t_k)
beta.matrix = tidy(lda, matrix = "beta") #restart R if "Error: No tidy method for
objects of class LDA_Gibbs", this is package bug
```

Probability of a word given topic $P(w|t_k)$ from candidate vector. This code is for display purposes ONLY, applied to all data within ocr.correct(). **score** demonstrates the application of

$$P(w) = \sum_{k}^{M} P(t_k)P(w|t_k)$$

Hide

Hide

```
tic  = Sys.time()
list.scores.word = c()
for (i in 1:length(candidates.list.nonempty[[15]])){

  prob.word.candidate = get.probability.word(candidates.list.nonempty[[15]][i], be
ta.matrix)#P(w|t_k)vector of 12 conditional probabilities
  score = sum(prob.word.candidate * probability.topic) # sum of P(w|t_k)*P(t_k)
  list.scores.word = c(list.scores.word, score)
}
toc = Sys.time()
toc-tic
```

```
Time difference of 0.1489089 secs
```

Hide

Hide

```
print("Candidate list with LDA probability scores")
```

```
[1] "Candidate list with LDA probability scores"
```

Hide

Hide

```
candidate.scores.word =rbind(candidates.list.nonempty[[15]],list.scores.word)
print(candidate.scores.word)
```

```
                  [,1]
                  "lending"
list.scores.word "2.4778381675403e-05"
                  [,2]
                  "pending"
list.scores.word "0.000205372800910924"
                  [,3]
                  "sending"
list.scores.word "5.79932131812266e-06"
                  [,4]
                  "tending"
list.scores.word "1.10741492920096e-05"
                  [,5]
                  "pendent"
list.scores.word "3.88466286171391e-06"
```

Hide

Hide

```
cat("Final choice: ", candidates.list.nonempty[[15]][which.max(list.scores.word)])
```

```
Final choice:  pending
```

Step 3.6 Combine application of LDA topic midel and confusion matrix for correct word selection.

Score of each word accounts for both models: LDA topic model that provides information about word probabilies and confusion matrix that represents the probability pf character errors.This code is for display purposes ONLY, applied to all data within ocr.correct().

$$Score(w_c) = P(w_c) \prod_{j}^{N} P(l_j^f | l_j^s)$$

Hide

Hide

```
cat("Error detected: ", ocr.true.error.nonempty[[4]], "\n")
```

```
Error detected:  commlttee
```

Hide

Hide

```
cat("Candidate list: ",candidates.list.nonempty[[4]], "\n")
```

```
Candidate list:  conmlttee committee conmittee coumittee committed
```

Hide

Hide

```
score_example = candidate.word.score(ocr.true.error.nonempty[[4]], candidates.list
.nonempty[[4]], probability.topic, confusion.prob, beta.matrix)
print(score_example)
```

```
[1] 7.630149e-11 1.718493e-04 3.684476e-10 4.642372e-13
[5] 6.332239e-12
```

Hide

Hide

```
cat("Final choice: ", candidates.list.nonempty[[4]][which.max(score_example)])
```

```
Final choice:  committee
```

Hide

Hide

```
finalist_list = c()
for (i in 1:length(ocr.true.error.nonempty)){
  score = candidate.word.score(ocr.true.error.nonempty[[i]], candidates.list.nonem
pty[[i]], probability.topic, confusion.prob, beta.matrix)
  if (is.na(sum(score)) == F){ #4 gibbrish entities in the detected vector produce
"NaN", eg 1618th element
  finalist = candidates.list.nonempty[[i]][which.max(score)]

  }else{
   finalist = candidates.list.nonempty[[i]][1]
  }
  finalist_list = c(finalist_list, finalist)
}
length(finalist_list)
```

```
[1] 3105
```

Hide

Hide

```
length(ocr.true.error.nonempty)
```

```
[1] 3105
```

Hide

Hide

```
length(candidates.list.nonempty)
```

```
[1] 3105
```

Hide

Hide

```
save(finalist_list, file = "../output/final_output_vector.Rdata")
compare.results.word = cbind(ocr.true.error.nonempty, finalist_list)
ocr.all.text.vec = unlist(str_split(paste(ocr.all.text, collapse = " "), " "))
for (i in 1:length(ocr.all.text.vec)){
  for (j in 1:length(compare.results.word[,1])){

    if (ocr.all.text.vec[i] == compare.results.word[j,1]){
      ocr.all.text.vec[i] = compare.results.word[j,2]
    }
  }
}
corrected.text = paste(ocr.all.text.vec, collapse = " ")
save(corrected.text, file = "../output/corrected_text.Rdata")
```

Step 4 - Performance Measure

For performance measure, the two most common OCR accuracy measures are precision and recall. To be more specific, precision is the proportion of the correct number of correct items (i.e. both words and characters) and the number of items in OCR output. Recall means that the ratio of the number of correct items and the number of items in ground truth. Here, to compute the numbers, we just need to get the intersection part bewteen OCR output version (before and after improvement) and ground truth. In other words, the alignment in this case is not necessary.

Hide

Hide

```
#words
ground.truth.vec = unlist(str_split(paste(ground.truth.all.text, collapse = " "),
" "))
ocr.vec = unlist(str_split(paste(ocr.all.text, collapse = " "), " "))
ocr.final.vec = unlist(str_split(paste(corrected.text, collapse = " "), " "))
#characters
ground.truth.vec.char = unlist(str_split(paste(ground.truth.all.text, collapse = "
"), ""))
ground.truth.vec.char = ground.truth.vec.char[ground.truth.vec.char != " "]
ocr.vec.char = unlist(str_split(paste(ocr.all.text, collapse = " "), ""))
ocr.vec.char = ocr.vec.char[ocr.vec.char != " "]
#intersect_words
original.intersect.vec = vecsets::vintersect(tolower(ground.truth.vec), tolower(oc
r.vec))
final.intersect.vec = vecsets::vintersect(tolower(ground.truth.vec), tolower(ocr.f
inal.vec))
#intersect_char
ocr.intersect.vec.char = vecsets::vintersect(unlist(str_extract_all(ocr.vec, "[:gr
aph:]")), unlist(str_extract_all(ground.truth.vec, "[:graph:]")))
ocr.intersect.final.vec.char = vecsets::vintersect(unlist(str_extract_all(ocr.fina
l.vec, "[:graph:]")), unlist(str_extract_all(ground.truth.vec, "[:graph:]")))
#"[:graph:]" refers to any character defined as a printable character except those
defined as part of the space character class. In this way, we can split the vector
s into characters.
ocr.performance.table = data.frame("ocr" = rep(NA, 4), "final_ocr"= rep(NA, 4))
row.names(ocr.performance.table) = c("word_wise_recall", "word_wise_precision", "c
haracter_wise_recall", "character_wise_precision")
ocr.performance.table["word_wise_recall","ocr"] = length(original.intersect.vec)/l
ength(ground.truth.vec)
ocr.performance.table["word_wise_precision","ocr"] = length(original.intersect.vec
)/length(ocr.vec)
ocr.performance.table["word_wise_recall","final_ocr"] = length(final.intersect.vec
)/length(ground.truth.vec)
ocr.performance.table["word_wise_precision","final_ocr"] = length(final.intersect.
vec)/length(ocr.vec)
ocr.performance.table["character_wise_recall","ocr"] = length(ocr.intersect.vec.ch
ar)/length(ground.truth.vec.char)
ocr.performance.table["character_wise_precision","ocr"] = length(ocr.intersect.vec
.char)/length(ocr.vec.char)
ocr.performance.table["character_wise_recall","final_ocr"] = length(ocr.intersect.
final.vec.char)/length(ground.truth.vec.char)
ocr.performance.table["character_wise_precision","final_ocr"] = length(ocr.interse
ct.final.vec.char)/length(ocr.vec.char)
kable(ocr.performance.table, caption = "Summary of OCR performance")
```

	ocr	final_ocr
word_wise_recall	0.6304754	0.6765579
word_wise_precision	0.6222718	0.6677546
character_wise_recall	0.9006393	0.9119577
character_wise_precision	0.9143944	0.9258856