Code

correction

Lingyi Zhao

2018/11/24

R Markdown

Combine data together

Hide

```
#rm(list=ls())
setwd("/Users/lingyizhao/Desktop")
data<-read.table('/Users/lingyizhao/Desktop/test_clean.txt')
data<-t(data)
data<-data.frame(data)
tesseract_vec<-read.csv('/Users/lingyizhao/Desktop/Detection_list.csv', header = FALSE)
data[,2]<-tesseract_vec[,1]
errorwords<-matrix(data[data$V2==0,1])
names(errorwords)<-"error"
```

Candidate dictionary:

For the total candidate set, we want to find a small specific subset for every error word. Here, we use minimum edit distance to calculate every error word with each word in candidate set. We choose threshold to 20 since we believe top 20 minimum edit distance are enough for our each error???s candidate. For example, if we choose threshold is equal to 20, this means we can accept at least 0 edit distance and at most around 3 or 4 characters different with error word.

Hide

```
candidate<-read.table('/Users/lingyizhao/Desktop/test_clean_truth.txt')
candidate<-t(candidate)
candidate<-data.frame(candidate)
#data[,3]<-candidate[,1]
#names(data)<-c("error","list","truth")
```

Candidate Search:

We choose candidate from this part:

Hide

```
candidate<-data.frame(candidate)
row.names(candidate)<-c(1:nrow(candidate))
for (p in 1:ncol(distances)){
  candidate_order<-data.frame(candidate[order(distances[,p], decreasing = TRUE),1])
  candidate_error<-candidate_order[1:threshold_c,1]
  candidate_errors<-data.frame(candidate_errors, candidate_error)
}
```

Feature Scoring:

Levenshtein edit distance:

In this feature scoring, we want to calculate the difference between two strings in spell correction. We use threshold here is 100, since we want to make every score within 0 to 1.

Hide

```
score_L<-load("/Users/lingyizhao/Desktop/score.RData")
```

String Similarity:

Longest common subsequence is an alternative approach than first one. Here we use three method for our calculation, subsequence from 1st character, subsequence from middle character and subsequence ending at last character. Notice, we also normalized our score inthsi method.

Hide

```
variable2<-load("/Users/lingyizhao/Desktop/score_SM.RData")
```

Language popularity:

We check the frequency of each candidate for each error in the ground truth and normalized every score within candidate and error.

Hide

```
##load candidate data
candidate_errors<-as.data.frame(candidate_errors)
head(candidate_errors)
##load clean text data
data_clean<-candidate
##remove NA
data_clean<-data_clean[!is.na(data_clean)]
#candidate_errors
#data_clean=="as"
##function to calculate the frequency
freq<-function(word){
  number<-sum(data_clean==word)
  return(number)
}

##frequency
freq_table<-c()
for (i in 1:nrow(candidate_errors)){
  freq_table<-rbind(freq_table,apply(candidate_errors[i,],2,freq))
}

##frequency/max
freq_table<-as.data.frame(freq_table)
max_frequency<-apply(freq_table,1,max)
freq_table$maxfreq<-max_frequency

##function to calculate porpotion
freq_score<-c()
for ( i in 1:20){
  freq_score<-cbind(freq_score,freq_table[,i]/freq_table[,21])
}
row.names(freq_score)<-row.names(candidate_errors)
##score table
variable3<-freq_score
```

Lexion existance:

In this method, we want to use a domain specific lexicon to capture terminologies. For example, if this candidate of first error appeared in the lexicon, we give score for the candidate and the error to 1. Otherwise, we give the score to 0.

Hide

```
variable4 <- all_grps$group1
```

Exact-context popularity:

This feature scoring method used context for our candidates since approach correction should be coherent in context. First of all, we find the context of every error in tesseract and make sure there have 4 words before error and 4 words after words (the total words should be 9 words). And then we need to replace error with their each candidate and construct 5-gram of these small ???sentence???. Finally, we calculate the frequency of each small ???sentence??? and give the score for each error and each its candidate.

Hide

```
save(score_Econtext, file = "score_Econtext.RData")
```

Regression model: random forest:

Hide

```
# dim(label)
# dim(trainset)
# model
load("total_final_data.RData")
```

Correct text:

Evaluation:

Hide

```
#corrected_text<-readLines("corrected_text_latest.txt")
ifCleanToken <- function(cur_token){
  now <- 1
  if_clean <- TRUE

  ## in order to accelerate the computation, conduct ealy stopping
  rule_list <- c("str_count(cur_token, pattern = '[A-Za-z0-9]') <= 0.5*nchar(cur_token)", # If the number of punctuation characters in a string is greater than the number of alphanumeric characters, it is garbage
                 "length(unique(strsplit(gsub('[A-Za-z0-9]','',substr(cur_token, 2, nchar(cur_token)-1)),'')[[1]]))>1", #Ignoring the first and last characters in a string, if there are two or more different punctuation characters in thestring, it is garbage
                 "nchar(cur_token)>20") #A string composed of more than 20 symbols is garbage
  while((if_clean == TRUE)&now<=length(rule_list)){
    if(eval(parse(text = rule_list[now]))){
      if_clean <- FALSE
    }
    now <- now + 1
  }
  return(if_clean)
}
## read the ground truth text
current_ground_truth_txt <- readLines("~/Desktop/test_clean_truth.txt", warn=FALSE)
## read the tesseract text
current_tesseract_txt <- readLines("~/Desktop/test_clean.txt", warn=FALSE)
clean_tesseract_txt <- paste(current_tesseract_txt, collapse = " ")
## detect tesseract word error
tesseract_vec <- str_split(clean_tesseract_txt," ")[[1]]
#tesseract_if_clean <- unlist(lapply(tesseract_vec,ifCleanToken)) # source code of ifCleanToken in in lib folder
#tesseract_delete_error_vec <- tesseract_vec[tesseract_if_clean]
## postprocessing text
tesseract_post_txt <- readLines("~/Desktop/corrected_text_latest.txt", warn=FALSE)
tesseract_clean_post_txt <- paste(tesseract_post_txt, collapse = " ")
```

Hide

```r
##string split the text file
#ground truth
ground_truth_vec <- str_split(paste(current_ground_truth_txt, collapse = " ")," ")[[1]]
ground_truth_char<-unlist(strsplit(ground_truth_vec,"",fixed = TRUE))
#original ocr file
tesseract_char<-unlist(strsplit(tesseract_vec,"",fixed = TRUE))
#postprocessing file
tesseract_post_vec <- str_split(tesseract_clean_post_txt," ")[[1]]
tesseract_post_char<-unlist(strsplit(tesseract_clean_post_txt,"",fixed = TRUE))
## Here, we compare the lower case version of the tokens
old_intersect_vec <- vecsets::vintersect(tolower(ground_truth_vec), tolower(tesseract_vec))
old_intersect_char <- vecsets::vintersect(tolower(ground_truth_char), tolower(tesseract_char))
```

Hide

```r
new_intersect_vec <- vecsets::vintersect(tolower(ground_truth_vec), tolower(tesseract_post_vec))
new_intersect_char <- vecsets::vintersect(tolower(ground_truth_char), tolower(tesseract_post_char))
##postprocessing vectors
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")
##word evaluation
OCR_performance_table["word_wise_recall","Tesseract"] <- length(old_intersect_vec)/length(ground_truth_vec)
OCR_performance_table["word_wise_precision","Tesseract"] <- length(old_intersect_vec)/length(tesseract_vec)
OCR_performance_table["word_wise_recall","Tesseract_with_postprocessing"] <- length(new_intersect_vec)/lengt
h(ground_truth_vec)
OCR_performance_table["word_wise_precision","Tesseract_with_postprocessing"] <- length(new_intersect_vec)/le
ngth(tesseract_vec)
kable(OCR_performance_table, caption="Summary of OCR performance")
```

Summary of OCR performance

	Tesseract	Tesseract_with_postprocessing
word_wise_recall	0.4750531	0.6158096
word_wise_precision	0.5322857	0.6900000
character_wise_recall	NA	NA
character_wise_precision	NA	NA

Hide

```r
##character evaluation
OCR_performance_table["character_wise_recall","Tesseract"] <- length(old_intersect_char)/length(ground_truth
_char)
OCR_performance_table["character_wise_precision","Tesseract"] <- length(old_intersect_char)/length(tesseract
_char)
OCR_performance_table["character_wise_recall","Tesseract_with_postprocessing"] <- length(new_intersect_char)
/length(ground_truth_char)
OCR_performance_table["character_wise_precision","Tesseract_with_postprocessing"] <- length(new_intersect_ch
ar)/length(tesseract_char)
kable(OCR_performance_table, caption="Summary of OCR performance-character evaluation")
```

Summary of OCR performance-character evaluation

	Tesseract	Tesseract_with_postprocessing
word_wise_recall	0.4750531	0.6158096
word_wise_precision	0.5322857	0.6900000
character_wise_recall	0.8664771	0.9023068
character_wise_precision	0.9420154	0.9809685