R Notebook

Code

Step 1 - Load library and source code

Hide

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

library(devtools)
library(pacman)
library(tm)
library(tidytext)
library(tidyverse)
library(DT)
library(RWeka)

pacman::p_load(knitr, readr, stringr, tesseract, vecsets)
#source('../lib/ifCleanToken.R')
file_name_vec <- list.files("../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.

Hide

```
# 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 c
ase)
#   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

Hide

```
#input: t is a txt file path; output is a cleaned txt
clean_txt<-function(t){
  current_txt <- readLines(t,warn=FALSE,encoding = "UTF-8")
  current_txt =gsub("[^A-Za-z ]","",current_txt)
  current_txt =trimws(current_txt)
  current_txt =gsub("\\s+"," ",current_txt)
  return(tolower(current_txt))
}

#get all the cleaned ground truth txt and cleaned orc txt
ground_truth<-list()
orc_txt<-list()
for(i in 1:length(file_name_vec)){
  ground_truth[[i]] <- clean_txt(paste("../data/ground_truth/",
                                       file_name_vec[i],sep=""))
  orc_txt[[i]] <- clean_txt(paste("../data/tesseract/",
                                  file_name_vec[i],sep=""))
}


#check the number of lines

not_equal_txt<-matrix(ncol=4)

for(i in 1:length(file_name_vec)){
  if(length(ground_truth[[i]])!=length(orc_txt[[i]])){
    not_equal_txt<-rbind(not_equal_txt,c(length(ground_truth[[i]]),
                                  length(orc_txt[[i]]),
                                  file_name_vec[i],i))
  }
}

#check the txt manually and make correction
ground_truth[[3]]<-ground_truth[[3]][1:291]
orc_txt[[3]]<-orc_txt[[3]][1:291]
#delete No.10 txt
#delete No.22 txt
ground_truth[[23]]<-ground_truth[[23]][1:222]
orc_txt[[23]]<-orc_txt[[23]][1:222]
ground_truth[[34]]<-ground_truth[[34]][1:466]
orc_txt[[34]]<-orc_txt[[34]][1:466]
ground_truth[[41]]<-ground_truth[[41]][1:740]
orc_txt[[41]]<-orc_txt[[41]][1:740]
ground_truth[[61]]<-ground_truth[[61]][-c(498,499)]
ground_truth[[63]]<-ground_truth[[63]][1:674]
orc_txt[[63]]<-orc_txt[[63]][1:674]
ground_truth[[68]]<-ground_truth[[68]][1:891]
orc_txt[[68]]<-orc_txt[[68]][1:891]
#delete No.70
ground_truth[[72]]<-ground_truth[[72]][-499]
#delete No.80
ground_truth[[100]]<-ground_truth[[100]][1:803]
```

```
#the txt pairs we can use
txt_file_num<-c(1:9,11:21,23:69,71:79,81:100)# total 96 txt pairs

#remove the lines with different length in pairs
for(i in txt_file_num){
  gt<-str_count(ground_truth[[i]],'\\w+')
  ot<-str_count(orc_txt[[i]],'\\w+')
  jud<-gt==ot
  ground_truth[[i]]<-ground_truth[[i]][jud]
  orc_txt[[i]]<-orc_txt[[i]][jud]
}



#the txt we can use now are ground_truth[[txt_file_num]] and orc_txt[[txt_file_num]]
#for text error detection and correction

source("./lib/ErrorDetection/2gram_error_detector.R")

source("./lib/ErrorDetection/orc_txt_error_detector.R")

ground_truth_use = ground_truth[txt_file_num]

orc_txt_use = orc_txt[txt_file_num]

truth_line = list()

truth_word = list()



for(i in 1:length(ground_truth_use)){
 truth_line[i] = paste(ground_truth_use[[i]], collapse = " ")

 truth_word[i] = list(str_split(truth_line[[i]]," ")[[1]])


}



orc_line = list()

orc_word = list()

for(i in 1:length(orc_txt_use)){
 orc_line[i] = paste(orc_txt_use[[i]], collapse = " ")

 orc_word[i] = list(str_split(orc_line[[i]]," ")[[1]])

}

############################### test for txt 5################################################
```

```
# while(orc_word[[5]][nchar(orc_word[[5]])>1]){
#
# error = neighbor_words(orc_word[[5]],truth_word[[5]]) # test for 1 document
# }
#
#
# orc5 = matrix(0,ncol=11,nrow=length(orc_word[[5]]))                                    #
#
# colnames(orc5) = c('word','correct_word',"neighbor1","neighbor2","neighbor3",
#                                                     "neighbor4","neighbor5","neighbor6",
#                                                     "neighbor7","neighbor8","truth")
#
# for(i in 1:nrow(orc5)){
#   orc5[i,] = error[[i]]
# }
#
# orc5 = write.csv(orc5,"./output/orc5.csv")
# # error = sapply(orc_dict,neighbor_words,truth_dict)  # for all documents
# ####### test for all#####################################
#
# orc_all = matrix(0,ncol=10,nrow=length(unlist(orc_dict)))
#
# colnames(orc_all) = c('word','correct_word',"neighbor1","neighbor2","neighbor3",
#                                                     "neighbor4","neighbor5","neighbor6",
#                                                     "neighbor7","neighbor8")
#
# file_name = file_name_vec[txt_file_num]
#
# for ( i in 1:length(file_name)){
#
#
#   error = neighbor_words(orc_dict[[i]],truth_dict[[i]])
#
#   orc = matrix(0,ncol=10,nrow=length(orc_dict[[i]]))
#
#   colnames(orc) = c('word','correct_word',"neighbor1","neighbor2","neighbor3",
#                                                     "neighbor4","neighbor5","neighbor6",
#                                                     "neighbor7","neighbor8")
#
#
#   for(j in 1:nrow(orc)){
#   orc[j,] = error[[j]]
# }
#
#   current_file_name <- sub(".txt","",file_name[i])
#
#   write.csv(orc, paste("./output/forcorrection/",current_file_name,".csv",sep=""))
#
#   print(i)
#
# }
```

Step 4 - Error correction

Generate n-gram candidate database

Create a corpus based on the ground_truth text

Hide

```
corpus <- VCorpus(VectorSource(ground_truth))%>%
  tm_map(removeWords, character(0))%>%
  tm_map(stripWhitespace)
dtm.docs <- DocumentTermMatrix(corpus)
```

Create n-grams candidate set

Hide

```
# Functions
OnegramTokenizer <- function(x) {RWeka::NGramTokenizer(x, RWeka::Weka_control(min=1, max=1))}
# BigramTokenizer <- function(x) {RWeka::NGramTokenizer(x, RWeka::Weka_control(min=2, max=2))}
ThreegramTokenizer <- function(x) {RWeka::NGramTokenizer(x, RWeka::Weka_control(min=3, max=3))}
# FourgramTokenizer <- function(x) {RWeka::NGramTokenizer(x, RWeka::Weka_control(min=4, max=4))}
FivegramTokenizer <- function(x) {RWeka::NGramTokenizer(x, RWeka::Weka_control(min=5, max=5))}

# #Onegram
options(mc.cores=1)
dtm.docs.1g <- DocumentTermMatrix(corpus, control=list(tokenize=OnegramTokenizer))
#
# # Bigrams
# options(mc.cores=1)
# dtm.docs.2g <- DocumentTermMatrix(corpus, control=list(tokenize=BigramTokenizer))
#
# #Threegrams
options(mc.cores=1)
dtm.docs.3g <- DocumentTermMatrix(corpus, control=list(tokenize=ThreegramTokenizer))
#
# #Fourgrams
# options(mc.cores=1)
# dtm.docs.4g <- DocumentTermMatrix(corpus, control=list(tokenize=FourgramTokenizer))

#Fivegrams
options(mc.cores=1)
dtm.docs.5g <- DocumentTermMatrix(corpus, control=list(tokenize=FivegramTokenizer))
```

Hide

```
# To get the onegram dist, we use the slam package for ops with simple triplet mat
sums.1g <- colapply_simple_triplet_matrix(dtm.docs.1g,FUN=sum)
sums.1g <- sort(sums.1g, decreasing=T)
write.csv(sums.1g, file = "onegram.csv")

# To get the 3gram dist, we use the slam package for ops with simple triplet mat
sums.3g <- colapply_simple_triplet_matrix(dtm.docs.3g,FUN=sum)
sums.3g <- sort(sums.3g, decreasing=T)
write.csv(sums.3g, file = "3-gram.csv")


# To get the fivegram dist, we use the slam package for ops with simple triplet mat
sums.5g <- colapply_simple_triplet_matrix(dtm.docs.5g,FUN=sum)
sums.5g <- sort(sums.5g, decreasing=T)
data.frame(sums.5g)[21,]

write.csv(sums.5g, file = "5-gram.csv")
```

Create relaxed context n-grams candidate set

Create 5-gram relaxed context candidates set

Hide

```
# read in 5-gram candidates frequency data
five.g <- read.csv("5-gram.csv")
colnames(five.g) <- c("5-gram","frequency")
# split each 5-grams to 5 individual words
l <-lapply(as.character(five.g$`5-gram`), strsplit, " ")
relaxedL <- l

relaxed1 <- c()
relaxed2 <- c()
relaxed3 <- c()
relaxed4 <- c()
relaxed5 <- c()


for(i in 1:length(l)){
  #replace first word with *
  relaxedL[[i]][[1]][1] <- "*"
  # paste five words back together to 5-gram
  relaxed1 <- rbind(relaxed1, paste(relaxedL[[i]][[1]], collapse = ' '))

  #replace second word with *
  relaxedL[[i]][[1]] <- l[[i]][[1]]
  relaxedL[[i]][[1]][2] <- "*"
  # paste five words back together to 5-gram
  relaxed2 <- rbind(relaxed2, paste(relaxedL[[i]][[1]], collapse = ' '))

  #replace third word with *
  relaxedL[[i]][[1]] <- l[[i]][[1]]
  relaxedL[[i]][[1]][3] <- "*"
  # paste five words back together to 5-gram
  relaxed3 <- rbind(relaxed3, paste(relaxedL[[i]][[1]], collapse = ' '))

  #replace fourth word with *
  relaxedL[[i]][[1]] <- l[[i]][[1]]
  relaxedL[[i]][[1]][4] <- "*"
  # paste five words back together to 5-gram
  relaxed4 <- rbind(relaxed4, paste(relaxedL[[i]][[1]], collapse = ' '))

  #replace fifth word with *
  relaxedL[[i]][[1]] <- l[[i]][[1]]
  relaxedL[[i]][[1]][5] <- "*"
  relaxed5 <- rbind(relaxed5, paste(relaxedL[[i]][[1]], collapse = ' '))
}

relaxed1.df <- data.frame(relaxed1, five.g$frequency)
relaxed2.df <- data.frame(relaxed2, five.g$frequency)
relaxed3.df <- data.frame(relaxed3, five.g$frequency)
relaxed4.df <- data.frame(relaxed4, five.g$frequency)
relaxed5.df <- data.frame(relaxed5, five.g$frequency)

# merge frequency information for same relaxed context 5-gram candidates
relaxed1.df <- relaxed1.df %>%
  group_by(relaxed1) %>%
```

```
  summarise(frequency = sum(five.g.frequency)) %>%
  arrange(frequency)

relaxed2.df <- relaxed2.df %>%
  group_by(relaxed2) %>%
  summarise(frequency = sum(five.g.frequency)) %>%
  arrange(frequency)


relaxed3.df <- relaxed3.df %>%
  group_by(relaxed3) %>%
  summarise(frequency = sum(five.g.frequency)) %>%
  arrange(frequency)

relaxed4.df <- relaxed4.df %>%
  group_by(relaxed4) %>%
  summarise(frequency = sum(five.g.frequency)) %>%
  arrange(frequency)

relaxed5.df <- relaxed5.df %>%
  group_by(relaxed5) %>%
  summarise(frequency = sum(five.g.frequency)) %>%
  arrange(frequency)

write.csv(relaxed1.df, file = "relaxed1.df.csv")
write.csv(relaxed2.df, file = "relaxed2.df.csv")
write.csv(relaxed3.df, file = "relaxed3.df.csv")
write.csv(relaxed4.df, file = "relaxed4.df.csv")
write.csv(relaxed5.df, file = "relaxed5.df.csv")
```

Create 3-gram relaxed context candidates set

Hide

```
# read in 3-gram data
three.g <- read.csv("3-gram.csv")
colnames(three.g) <- c("3-gram","frequency")

# split 3 grams lines to individual words
l <-lapply(as.character(three.g$`3-gram`), strsplit, " ")
relaxedL <- l


relaxed1.3gram <- c()
relaxed2.3gram <- c()
relaxed3.3gram <- c()


for(i in 1:length(l)){
  # replace first word with *
  relaxedL[[i]][[1]][1] <- "*"
  # combine three words back as * word1 word2 and append to dataframe
  relaxed1.3gram <- rbind(relaxed1.3gram, paste(relaxedL[[i]][[1]], collapse = ' '))

  # replace second word with *
  relaxedL[[i]][[1]] <- l[[i]][[1]]
  relaxedL[[i]][[1]][2] <- "*"

  # combine three words back as 'word1 * word3' and append to dataframe
  relaxed2.3gram <- rbind(relaxed2.3gram, paste(relaxedL[[i]][[1]], collapse = ' '))

  # replace third word with *
  relaxedL[[i]][[1]] <- l[[i]][[1]]
  relaxedL[[i]][[1]][3] <- "*"
  # combine three words back as 'word1 word2 *' and append to dataframe
  relaxed3.3gram <- rbind(relaxed3.3gram, paste(relaxedL[[i]][[1]], collapse = ' '))

}

relaxed1.3g.df <- data.frame(relaxed1.3gram, three.g$frequency)
relaxed2.3g.df <- data.frame(relaxed2.3gram, three.g$frequency)
relaxed3.3g.df <- data.frame(relaxed3.3gram, three.g$frequency)

# merge frequency information for same relaxed context 3-gram candidates
relaxed1.3g.df <- relaxed1.3g.df %>%
  group_by(relaxed1.3gram) %>%
  summarise(frequency = sum(three.g.frequency)) %>%
  arrange(frequency)

relaxed2.3g.df <- relaxed2.3g.df %>%
  group_by(relaxed2.3gram) %>%
  summarise(frequency = sum(three.g.frequency)) %>%
  arrange(frequency)


relaxed3.3g.df <- relaxed3.3g.df %>%
  group_by(relaxed3.3gram) %>%
```

```
  summarise(frequency = sum(three.g.frequency)) %>%
  arrange(frequency)


write.csv(relaxed1.3g.df, file = "relaxed1.3g.df.csv")
write.csv(relaxed2.3g.df, file = "relaxed2.3g.df.csv")
write.csv(relaxed3.3g.df, file = "relaxed3.3g.df.csv")
```

Step 5 - Performance measure

Hide

```
# word level
count_intersect<-function(truth,orc){
  ground_truth_vec <- str_split(paste(truth, collapse = " ")," ")[[1]]
  orc_vec<-str_split(paste(orc, collapse = " ")," ")[[1]]
  return(length(vecsets::vintersect(ground_truth_vec,orc_vec)))
}

word_recall_before<-rep(NA,length(txt_file_num))
word_precis_before<-rep(NA,length(txt_file_num))
# word_recall_after<-rep(NA,length(txt_file_num))
# word_precis_after<-rep(NA,length(txt_file_num))

for(i in txt_file_num){
  truth_length<-length(str_split(paste(ground_truth[[i]], collapse = " ")," ")[[1]])
  orc_length<-length(str_split(paste(orc_txt[[i]], collapse = " ")," ")[[1]])
  old_vec_len<-count_intersect(ground_truth[[i]],orc_txt[[i]])
  word_recall_before[i]<-old_vec_len/truth_length
  word_precis_before[i]<-old_vec_len/orc_length
}

#postprocessing for one txt
rlt<-read.csv("../output/rltX2.csv")
rlt<-as.data.frame(rlt)
word_wise_after<-sum(as.character(rlt$Correction)==as.character(rlt$GroundTruth))/nrow(rlt)#0.75
5



#character level
intersect_number<-function(a,b){
  ta<-data.frame(table(a))
  tb<-data.frame(table(b))
  mt<-merge(ta,tb,by.x = "a",by.y = "b")
  mt$min_freq<-apply(mt[,c(2,3)],1,min)
  return(sum(mt$min_freq))
}

compare_word<-function(bi_words){
  if(nchar(bi_words[1])==0 |nchar(bi_words[2])==0 ){
    return(0)
  }else{
    a<-strsplit(bi_words[1],split="")[[1]]
    b<-strsplit(bi_words[2],split="")[[1]]
    return(intersect_number(a,b))
  }
}



chara_recall_before<-rep(NA,length(txt_file_num))
chara_precis_before<-rep(NA,length(txt_file_num))
#chara_recall_after<-rep(NA,length(txt_file_num))
#chara_precis_after<-rep(NA,length(txt_file_num))

for(i in txt_file_num){
```

```
  gt<-str_split(paste(ground_truth[[i]], collapse = " ")," ")[[1]]
  ot<-str_split(paste(orc_txt[[i]], collapse = " ")," ")[[1]]
  exa<-cbind(gt,ot)
  rt<-apply(exa,1,compare_word)
  chara_recall_before[i]<-sum(rt)/sum(nchar(gt))#ratio of correct with ground_truth
  chara_precis_before[i]<-sum(rt)/sum(nchar(ot))#ratio of correct with ORC output

}

#postprocessing for one txt
rlt_chara<-apply(rlt[,1:2],1,compare_word)
length_intersect_chara<-sum(rlt_chara)
chara_wise_after_recall<-length_intersect_chara/sum(nchar(as.character(rlt$GroundTruth)))#0.928
chara_wise_after_precis<-length_intersect_chara/sum(nchar(as.character(rlt$ORC)))
#0.951554

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_precisi
on")
OCR_performance_table["word_wise_recall","Tesseract"] <-
  mean(word_recall_before,na.rm = T)
OCR_performance_table["word_wise_precision","Tesseract"] <-
  mean(word_precis_before,na.rm = T)
OCR_performance_table["word_wise_recall","Tesseract_with_postprocessing"]<-
  word_wise_after
OCR_performance_table["word_wise_precision","Tesseract_with_postprocessing"]<-
  word_wise_after
OCR_performance_table["character_wise_recall","Tesseract"] <-
  mean(chara_recall_before,na.rm = T)
OCR_performance_table["character_wise_precision","Tesseract"] <-
  mean(chara_precis_before,na.rm = T)
OCR_performance_table["character_wise_recall","Tesseract_with_postprocessing"] <-
  chara_wise_after_recall
OCR_performance_table["character_wise_precision","Tesseract_with_postprocessing"] <-
  chara_wise_after_precis
kable(OCR_performance_table, caption="Summary of OCR performance")
```

In [ ]:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 18 15:18:26 2018

@author: Chenghao
"""
from pyxdameraulevenshtein import damerau_levenshtein_distance
from nltk.metrics.distance import edit_distance
import py_common_subseq.py_common_subseq as CS #Need to change xrange() as range
()
import math
import pandas as pd


# Candidate search
class project4():

    def candidate_search(Dictionary, We, threshold):
        candidate = {}
        for Wc in Dictionary:
            dist = damerau_levenshtein_distance(Wc, We)
            if dist <= threshold:
                candidate[Wc] = Dictionary[Wc]

        return(candidate)

#-------------------------------------
# Feature scoring
# Levenshtein edit distance
    def distance_score(candidates, We, threshold):
        Score = {}
        for Wc in candidates:
            score = 1 - edit_distance(Wc, We, substitution_cost=1, transposition
s=False)/(threshold + 1)
            Score[Wc] = score

        return(Score)

# String similarity
    def similarity_score(candidates, We, a1=0.25, a2=0.25, a3=0.25, a4=0.25):
        Score = {}
        for Wc in candidates:
            common_subsequences = CS.find_common_subsequences(Wc, We)
            lcs = sorted(common_subsequences, key=lambda x: len(x))[-1]


            IniLetter = We[0]
            EndLetter = We[-1]
            MidLetter = We[math.ceil(len(We)/2)]

            #LCS_1
            common_subseq_IntLetter = set([])
            for W in common_subsequences:
                if W.startswith(IniLetter):
                    common_subseq_IntLetter.add(W)

            if len(common_subseq_IntLetter) == 0:
                lcs1 = ''
            else:
```

```
                lcs1 = sorted(common_subseq_IntLetter, key=lambda x: len(x))[-1]

            #LCS_z
            common_subseq_EndLetter = set([])
            for W in common_subsequences:
                if W.endswith(EndLetter):
                    common_subseq_EndLetter.add(W)

            if len(common_subseq_EndLetter) == 0:
                lcsz = ''
            else:
                lcsz = sorted(common_subseq_EndLetter, key=lambda x: len(x))[-1]

            #LCS_n
            common_subseq_MidLetter = set([])
            for W in common_subsequences:
                if W.startswith(MidLetter):
                    common_subseq_MidLetter.add(W)

            if len(common_subseq_MidLetter) == 0:
                lcsn = ''
            else:
                lcsn = sorted(common_subseq_MidLetter, key=lambda x: len(x))[-1]

            denom = len(Wc) + len(We)
# ================================================================================
#             nlcs = (2*len(lcs)**2)/denom
#             nmnlcs1 = (2*len(lcs1)**2)/denom
#             nmnlcsn = (2*len(lcsn)**2)/denom
#             nmnlcsz = (2*len(lcsz)**2)/denom
# ================================================================================
            # original paper
            nlcs = (2*len(lcs))/denom
            nmnlcs1 = (2*len(lcs1))/denom
            nmnlcsn = (2*len(lcsn))/denom
            nmnlcsz = (2*len(lcsz))/denom
            score = a1*nlcs + a2*nmnlcs1 + a3*nmnlcsn + a4*nmnlcsz
            Score[Wc] = score

        return(Score)

# Language popularity
    def popularity_score(candidates):
        Score = {}
        denom = max(candidates.values())
        for Wc in candidates:
            score = candidates[Wc]/denom
            Score[Wc] = score

        return(Score)

# Lexicon existance
    def existance_score(candidates, lexicon): # lexicon is a set {candidate1, ca
ndidate2}
        Score = {}
        for Wc in candidates:
            score = int(Wc in lexicon)
            Score[Wc] = score

        return(Score)
```

```
# Exact-context popularity
    def exact_popularity_score(candidates, We, five_gram_E, five_gram_dic): # fi
ve_gram_E is 5-gram with We, five_gram_dic is dictionary {5 gram w/ Wc: frequenc
y}
        Score = {}
        Numer = {}
        for Wc in candidates:
            five_gram_C = []
            for five_gram in five_gram_E:
                five_gram_C.append(five_gram.replace(We, Wc, 1))
            numer = 0
            for five_gram_c in give_gram_C:
                numer = numer + five_gram_dic[five_gram_c]
            Numer[Wc] = numer

        Denom = max(Numer.values())
        for Wc in candidates:
            Score[Wc] = Numer[Wc]/Denom

        return(Score)

# Relaxed-context popularity
    def relaxed_popularity_score(candidates, We, five_gram_E, five_gram_dic_X):
        Score = {}
        Numer = {}
        for Wc in candidates:
            five_gram_C = []
            grams_C_X = []
            for five_gram in five_gram_E:
                five_gram_C.append(five_gram.replace(We, Wc, 1))
            for i in range(5):
                five_gram_s = list(jieba.cut(five_gram_C[i]))
                for k in range(5):
                    if -i+4==k:
                        continue
                    else:
                        five_gram_s_copy = copy.deepcopy(five_gram_s)
                        five_gram_s_copy[2*k] = "*"
                        gram_c_x = "".join(five_gram_s_copy)
                        grams_C_X.append(gram_c_x)
            numer = 0
            for five_gram_c_x in grams_C_X:
                numer = numer + five_gram_dic_X[five_gram_c_x]
            Numer[Wc] = numer

        Denom = max(Numer.values())
        for Wc in candidates:
            Score[Wc] = Numer[Wc]/Denom

        return(Score)
```