Search engine of Chinese food recipes and analysis of ingredients and reviews

Jing Wang

Introduction

Background

Chinese cuisine is an important part of Chinese history. Traditionally, Chinese people great each other with the sentence of "Have you eaten yet?" Today, Chinese cuisine has significant influenced in many countries. CNN has ranked China as the second place of the best food cultures. According to the report from the Chinese American Restaurant Association, there are more than 45,000 in the United States, which has exceeded the total number of McDonald's, KFCs, Pizza Hunts, Taco Bells and Wendy's. [1]

A lot of American people would like to cook Chinese food at home, however, it is hard for people to find the appropriate recipes from the other country's cuisine. The reason is that people always have preferred and unwanted ingredients. For example, most American people do not eat wood ears, which is a favored healthful mushroom in China. The popular dish, Gong Bao chicken has peanuts as the major ingredient, which is among the most common allergy-causing food. In addition, some dishes require long time for preparation or cooking, such as dumplings and Peking duck. However, people may find these undesired recipes through Google or many websites. To the best of my knowledge, recipe websites only provide the functions of searching recipes based on the food visitors want. However, none of these websites have the well-designed function to search the most appropriate recipes based on both visitor's like and dislike.

Overview of objectives and approaches

For the convenience of people who love Chinese cuisine, a recipe research engine was developed in this case study. The major objective of this study is to develop a search engine for Chinese recipes and analyze sentiment of reviews of each recipe. The main objective was achieved by the three sub-objectives:

(1) Develop a fast-speed search engine by regular expression and cosine similarity algorithms.
(2) Evaluate the popularity of recipe category based on the number of reviews.
(3) Score recipes by sentiment analysis of reviews and investigate the sentiment score for each recipe category.

To address these objectives, data of recipes and visitor's reviews was collected from a famous blog of Chinese cuisine, "The Works of Life". The website includes over 900 recipes and 14,000 reviews. All analysis was conducted in R and R studio.

Major findings

The designed search engine allows users type in preferred ingredient, undesired ingredient and the maximum hours they would like to spend on preparing food. By applying the regular expression and cosine similarity algorithms of documents, the search engine can provide the most similar 15 recipes within 0.25 second. Each recipe is displayed with HTML address, recipe name, ingredients and mean of sentiment score.

The analysis of recipe popularity shows that beef recipes have the highest number of reviews, which corresponds to the taste of American people. Therefore, recipe websites or even TV cooking show are suggested to provide more content related to beef.

In addition, the sentiment analysis revealed that recipe categories have very similar sentiment score. Recipes with tofu, turkey or egg have the highest ratings. Other categories have closed score. These results reveal that bloggers offered good recipes with each major ingredient.

Outline of the rest of the report

The remainder of this report is structured as follows. In the next section, the methodologies applied for the individual step in the analysis are presented, and important algorithms are highlighted. Furthermore, the results of sentiment analysis, the popularity of recipe categories and the performance of the search engine will be discussed. In the final part, the advantages and potential improvement of this study is discussed.

Methods

The Works of Life blog provides over 900 Chinese recipes. Each recipe includes ingredients, preparation time and cook time, and reviews. This information was grabbed from the website by SelectorGadget. After removing duplications, there are 648 unique recipes.

The original data of total time includes two parts, preparation time and cook time. The total time was grabbed and converted to the unit of hour.

The collected information of ingredients and reviews was further processed by the related queries of "rvest" package in R. Sentiment analysis of reviews was performed according to a list of English positive and negative opinion words or sentiment words (around 6800 words).[2] Considering the unique positive and negative words for cooking, a few of words were added in the dictionaries. After removing numbers and punctuations, and converting characters to lower case, each review was split into words and identified as positive or negative word by the two dictionaries. The sentiment score of each review was calculated by the difference between the number of positive words and the number of negative words. The sentiment score of each recipe was the mean of sentiment score of its all reviews.

Recipes are classified as the categories of nuts, noodle/dumpling, dessert/cake, lamb, beef, chicken, turkey, seafood, egg and vegetables/drinks. The category of a recipe was first determined by recipe name. If the recipe name does not include the key words of categories, the recipe category was decided by its ingredients.

The search engine was developed according to the regular expression of texting mining and cosine similarity of documents. First, recipes were filtered by the excluded ingredient and total time. Recipes which includes undesired ingredient or total time is longer than the input time will be filtered out. Each recipe and favored ingredient were then combined as a corpus and transferred into the format of document term matrix (dtm) and weighted by term frequency–inverse document frequency. The favored ingredient exists as a document in dtm. To accelerate the searching speed, recipes were further filtered by the favored ingredient. Only the recipes including the favored ingredients are kept for the following analysis. The algorithm of cosine similarity was used to score the similarity between input ingredient and recipes. The top 15 recipes were selected and recommended based on the score ranking of cosine similarity. Cosine similarity was calculated according to the following formula:

$$\text{Similarity}(d_1, d_2) = \frac{d_1 \times d_2}{||d_1|| \times ||d_2||}$$

where d1 and d2 are both vectors of TF-IDF scores over the vocabulary assed within the corpus. [3]

Results and Discussions

The search engine manifested fast-speed performance. The searching time is about 0.2 second. The final table of the selected recipes provide the information of recipe name, HTML address, ingredients, total time and mean of sentiment scores. Screenshots the search engine in R shiny app were shown in Appendix A.

The distribution of the number of reviews for recipes is exhibited in Figure 1. The great majority of recipes have about 10 to 25 reviews. The recipe named "milk bread" has the highest number of reviews, over 300.

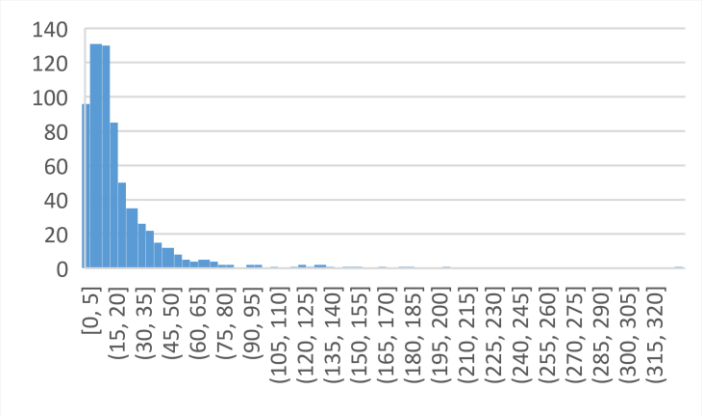


Figure 1. Histogram of the Number of Reviews

To reveal the popularity of recipe categories, the mean of number of reviews for each category was calculated and displayed in the Figure 2. Recipes with beef has the highest number of reviews, around 24 reviews per recipe. The categories of dessert/cake, noodle/dumplings and pork were popular categories as well. According to the analysis result, the bloggers, other recipe websites or even TV cooking shows may consider providing more recipes belonged to these top categories to satisfy the visitors' demand and increase the Click-through rate.

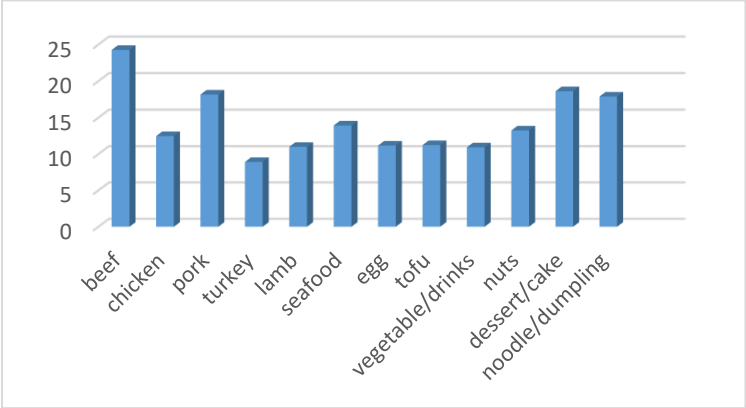


Figure 2. Mean of the Number of Reviews vs Recipe Category

The mean of sentiment score for each recipe category is presented in Figure 3. Different from the results of mean of number of reviews for categories, the variation among the mean of sentiment scores for categories is very small. Furthermore, all scores are positive. The phenomenon reveals that reviewers always have positive attitude to recipes. In the other words, the blogger dedicate effort for each recipe category and offered recipes with high quality. That is probably the major reason that the blog, "The Works of Life" is one of the top three English Chinese recipe blog.

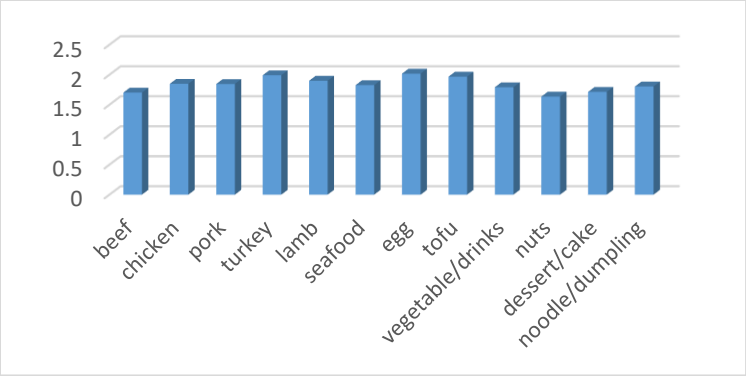


Figure 3. Mean of Sentiment Score vs Recipe Category

Conclusions

In the consideration of large demand for Chinese cuisine recipes, a search engine for Chinese food recipes was designed, and related analysis for major ingredients and recipe reviews was conducted. All data of recipes and corresponding reviews was collected from a famous English Chinese recipe blog, "The Works of Life". By using the regular expression and the algorithm of similarity of documents in R, a fast-speed search engine was successful designed. After typing in the favored ingredient, unwanted ingredient and maximum hours for cooking, the search engine can provide the most similar recipes within 0.3 second.

Recipes were clustered into different categories by major ingredients. Recipes with beef have the biggest number of reviews. Other popular ingredients include pork, dumplings/noodle and dessert/cake. Recipe website and cooking shows/channels could provide more recipes related to these top ingredients in order to fulfill people's demand and increase Click-through rate.

The sentiment analysis for reviews shows that each recipe category has very similar positive sentiment score. The result reveals that the bloggers provide for high quality recipe for each food category.

This case study also has some limitations. First, data was only collected from one website, which limited the further investigation about popular ingredients. Second, the sentiment analysis was not conducted for different time periods. It could be interesting to perform time-series sentiment analysis and explore the change of taste of English-speaking people to Chinese food.

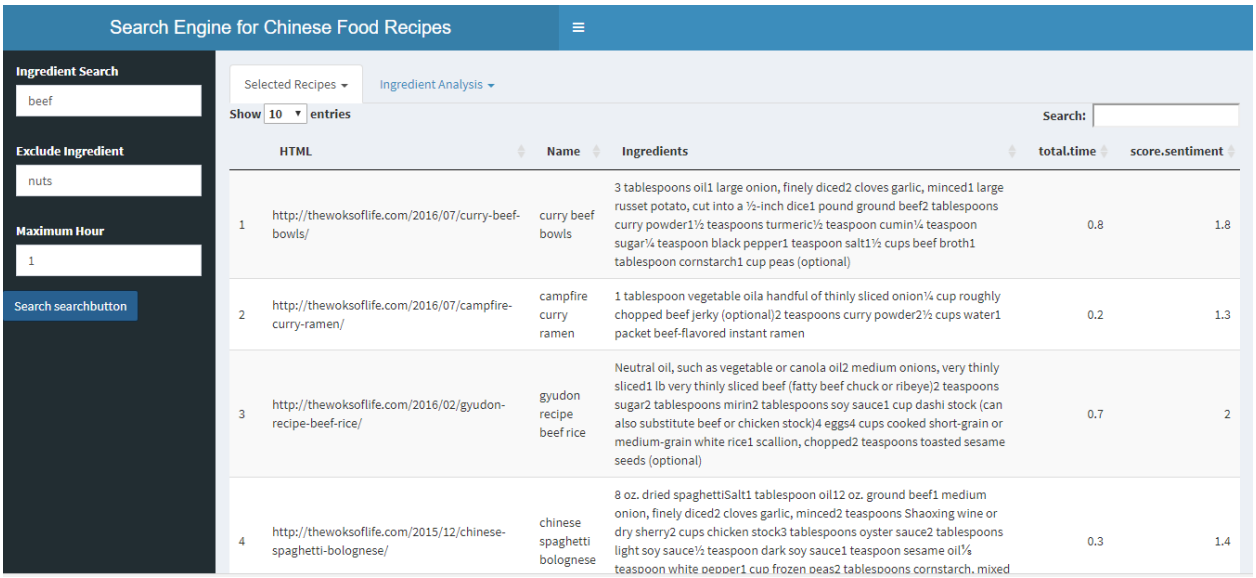
References

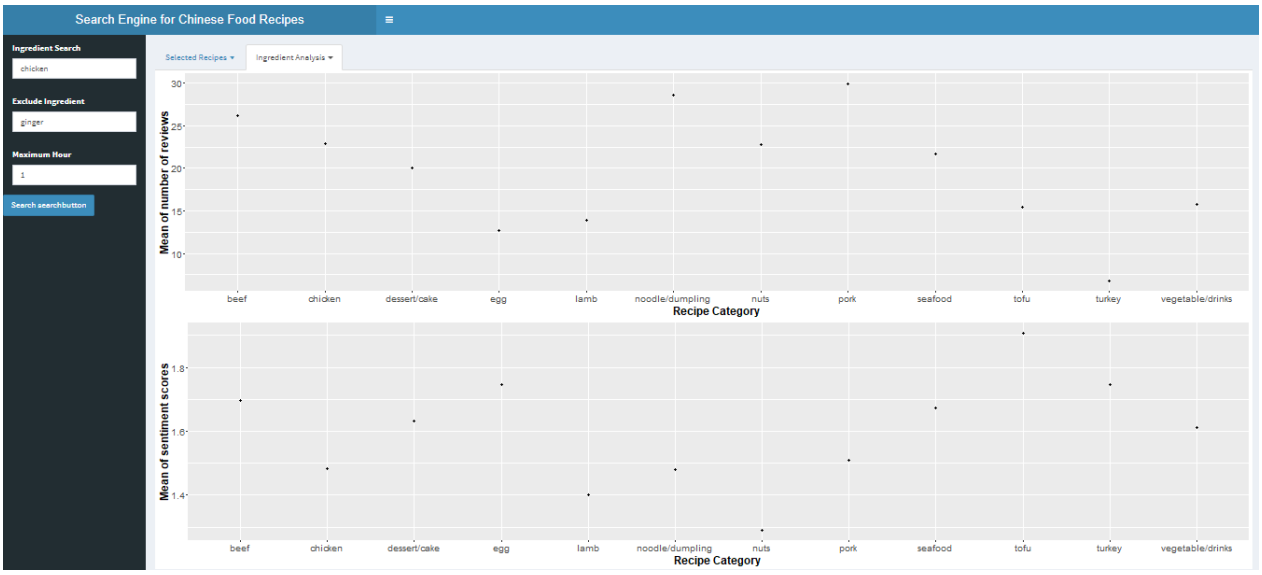
http://time.com/4211871/chinese-food-history/

Liu, Bing. 2012. *Sentiment Analysis and Opinion Mining*, Morgan & Claypool Publishers.

B. Larsen, C. Aone. 1999. "Fast and effective text mining using linear time document clustering," *in Proceedings of the Conference on Knowledge Discovery and Data Mining*, pp. 16-22.

Appendix A:





Appendix B:

```
rm(list = ls())
library(tm)
library(SnowballC)
library(tidytext)
library(dplyr)
library(reshape2)
library(tidyverse)
```

```
library(lsa)
require(lubridate)
require(plotly)
require(shinydashboard)
require(shiny)
require(ggplot2)
require(scales)
require(lubridate)
require(qpcR)
require(shinyBS)
require(DT)
require(rvest)
require(dplyr)
require(stringr)
library(tidyr)
require("coreNLP")
require("pander")
require("syuzhet")
library(qdapRegex)
#
# ########################################
#  url <- "http://thewoksoflife.com/recipe-list/"
#  html <- url%>% read_html()%>%html_nodes(".entry-content li a")%>%html_attr("href")
#
#  df <- data.frame(matrix(NA,nrow = 928,ncol = 2))
#  colnames(df) <- c("Name","Ingredients")
#  total.time <- c()
#  for(i in 1:928){
#    ## recipe name
#    a <- ex_between(html[i], "/","/")[[1]][5]
#    df[i,1] <- gsub("-"," ",a)
#    ## ingredients
#    ingredient <- html[i]%>% read_html()%>%html_nodes(".ingredient")%>%html_text()
#    df[i,2] <- paste(ingredient,sep = "" ,collapse = "")
#    ## prepare time
#                b <- html[i]%>% read_html()%>%html_nodes(".ERSTimeRight+ .ERSTimeRight
time")%>%html_text()
#    # grep("+hour", b,perl=TRUE, value=TRUE)
```

```
#   time <- b %>% str_match_all("[0-9]+") %>% unlist %>% as.numeric
#   if (length(time)==0){total.time[i] <- NA}else{
#     if (grepl("hour",b)==T&grepl("mins",b)==T){
#       total.time[i] <- time[1]+time[2]/60
#     }else if (grepl("hour",b)==T&grepl("mins",b)==F){
#       total.time[i] <- time[1]
#     }else if (grepl("hour",b)==F&grepl("mins",b)==T){
#       total.time[i] <- time[1]/60
#     }
#
#   }
#   print(i)
# }
# df <- cbind(html[1:928],df,total.time)
# colnames(df)[1] <- "HTML"
# df <- df[!duplicated(df),]
# s <- list(length=648)
# df$HTML <- as.character(df$HTML)
# for(i in 1:648){
#   # grab the reviews for each recipe
#       s[[i]] <- df$HTML[i]%>% read_html()%>%html_nodes(".depth-1 > article .comment-content p")%>%html_text()
# }
#
# setwd("C:/Text mining/Final Project")
# hu.liu.pos <- scan('positive-words.txt', what='character', comment.char=';')
# hu.liu.neg <- scan('negative-words.txt', what='character', comment.char=';')
#
# ## check whether some important in liu's dictionary
# # which(hu.liu.pos%in%"delicious")
# # which(hu.liu.neg%in%"long")
# # Add a few twitter and industry favorites
# pos.words <- c(hu.liu.pos, "can't wait","can not wait ")
# neg.words <- c(hu.liu.neg, "long")
#
# score.sentiment = function(sentences, pos.words, neg.words, .progress='none')
# {
#   require(plyr)
```

```
#   require(stringr)
#
#   scores = laply(sentences, function(sentence, pos.words, neg.words) {
#
#     # clean up sentences with R's regex-driven global substitute, gsub():
#     ## remove non
#     sentence <- stringr::str_replace_all(sentence,"[^a-zA-Z\\s]", " ")
#     sentence = gsub('[[:punct:]]', '', sentence)
#     sentence = gsub('[[:cntrl:]]', '', sentence)
#     sentence = gsub('\\d+', '', sentence)
#     # and convert to lower case:
#     sentence = tolower(sentence)
#
#     # split into words. str_split is in the stringr package
#     word.list = str_split(sentence, '\\s+')
#     # sometimes a list() is one level of hierarchy too much
#     words = unlist(word.list)
#
#     # compare our words to the dictionaries of positive & negative terms
#     pos.matches = match(words, pos.words)
#     neg.matches = match(words, neg.words)
#
#     # match() returns the position of the matched term or NA
#     # we just want a TRUE/FALSE:
#     pos.matches = !is.na(pos.matches)
#     neg.matches = !is.na(neg.matches)
#
#     # and conveniently enough, TRUE/FALSE will be treated as 1/0 by sum():
#     score = sum(pos.matches) - sum(neg.matches)
#
#     return(score)
#   }, pos.words, neg.words, .progress=.progress )
#
#   scores.df = data.frame(score=scores, text=sentences)
#   return(scores.df)
# }
#
```

```
#  # Score the sentiments for each review of each recipe
#  for(i in 1:648){
#                          df$score.sentiment[i] <-score.sentiment(s[[i]], pos.words, neg.words,
.progress='text')$score%>%mean()%>%round(.,1)
#    print(i)
#  }
#
#  ## major ingredients: chicken, beef, turkey, egg, fish, seafood, shrimp,pork,tofu
#  for(i in 1:648){
#    if(grepl("nut",df$Name[i])==T){
#      df$category[i] <- "nuts"
#    }else if(grepl("noodles",df$Name[i])==T|grepl("dumpling",df$Name[i])==T){
#      df$category[i] <- "noodle/dumpling"
#                                                                                  }else if
(grepl("pancake",df$Name[i])==T|grepl("cake",df$Name[i])==T|grepl("dumpling",df$Name[i])==T){
#      df$category[i] <- "dessert/cake"
#    }else if (grepl("lamb",df$Name[i])==T){
#      df$category[i] <- "lamb"
#    }else if (grepl("beef",df$Name[i])==T|grepl("steak",df$Name[i])==T){
#      df$category[i] <- "beef"
#    }else if(grepl("turkey",df$Name[i])==T){
#      df$category[i] <- "turkey"
#                                                                                  }else
if(grepl("shrimp",df$Name[i])==T|grepl("fish",df$Name[i])==T|grepl("seafood",df$Name[i])==T){
#      df$category[i] <- "seafood"
#    }else if(grepl("pork",df$Name[i])==T|grepl("rib",df$Ingredients[i])==T){
#      df$category[i] <- "pork"
#    }else if(grepl("tofu",df$Name[i])==T){
#      df$category[i] <- "tofu"
#    }else if(grepl("chicken",df$Name[i])==T){
#      df$category[i] <- "chicken"
#    }else if(grepl("eggs",df$Name[i])==T|grepl("tea eggs",df$Name[i])==T){
#      df$category[i] <- "egg"
#                                                                                  }else
if(grepl("cabbage",df$Name[i])==T|grepl("eggplant",df$Name[i])==T|grepl("broccoli",df$Name[i])==T|
#             grepl("tea",df$Name[i])==T){
#      df$category[i] <- "vegetable/drinks"
#    }else if (grepl("beef",df$Ingredients[i])==T|grepl("steak",df$Ingredients[i])==T){
#      df$category[i] <- "beef"
```

```
#   }else if(grepl("turkey",df$Ingredients[i])==T){
#     df$category[i] <- "turkey"
#                                                                                                   }else
if(grepl("fish",df$Ingredients[i])==T|grepl("seafood",df$Ingredients[i])==T|grepl("shrimp",df$Ingredie
nts[i])==T){
#     df$category[i] <- "seafood"
#   }else if(grepl("pork",df$Ingredients[i])==T|grepl("rib",df$Ingredients[i])==T){
#     df$category[i] <- "pork"
#   }else if(grepl("tofu",df$Ingredients[i])==T){
#     df$category[i] <- "tofu"
#   }else if(grepl("chicken",df$Ingredients[i])==T){
#     df$category[i] <- "chicken"
#   }else if(grepl("pancake",df$Ingredients[i])==T|grepl("flour",df$Ingredients[i])==T){
#     df$category[i] <- "dessert/cake"
#  }else if(grepl("tea eggs",df$Ingredients[i])==T|grepl("eggs",df$Ingredients[i])==T){
#     df$category[i] <- "egg"
#   }else{
#     df$category[i] <- "vegetable/drinks"
#   }
#   print(i)
# }
# ## summary the number of reviews
# for(i in 1:648){
#   df$number_reviews[i] <- length(s[[i]])
# }
# ## make category as a factor variable
# df$category <- as.factor(df$category)
# df1 <- aggregate(df$number_reviews,by=list(df$category),mean)
# colnames(df1) <- c("category","mean.number_reviews")
# df2 <- df[!is.na(df$score.sentiment),]
# df3 <- aggregate(df2$score.sentiment,by=list(df2$category),mean)
# colnames(df3) <- c("category","score.sentiment")
# df.category <- merge(df1,df3,by="category",all = T)
# output <- list(df,df.category)
# saveRDS(output, file = "finalproject.rds")

setwd("C:/Text mining/Final Project")
output <- readRDS(file = "finalproject.rds")
```

```
df <- output[[1]]
df.category <- output[[2]]
##############################################
ui <- dashboardPage(
 ## create Header
 dashboardHeader(
  title="Search Engine for Chinese Food Recipes",
  titleWidth = 600),
 dashboardSidebar(
 textInput("include","Ingredient Search","chicken"),
 textInput("exclude","Exclude Ingredient","ginger"),
 textInput("time","Maximum Hour","1"),
 submitButton("searchbutton","Search")
 ),
 dashboardBody(tabsetPanel(
  navbarMenu(
   title = "Selected Recipes",
   tabPanel("Selected Recipes",
         fluidRow(
          column(width = 12,
              dataTableOutput("table1")
          )
          )
         )
   ),
  navbarMenu(
   title = "Ingredient Analysis",
   tabPanel("Ingredient Analysis",
         fluidRow(
          column(width = 12,
              plotOutput("plot1")
          ),
          column(width = 12,
              plotOutput("plot2")

          )
         )
```

```
    )
  ))))

server <- function(input,output){
  run.model <- reactive({
    t1 <- Sys.time()
    ## filter recipes by excluded ingredient
    df.length <- nrow(df)
    for(i in 1:df.length){
      if(grepl(input$exclude,df$Ingredients[i],perl = T)==T){
        df <- df[-i,]
      }
    }
    print("exclude")
    time <- as.numeric(input$time)
    print("time")
    ## filter recipes by time
    if(length(time)>0){
      df <- df[!is.na(df$total.time),]
      df <- df[as.numeric(df$total.time)<=time,]
    }
    print(df$total.time[length(df$total.time)])
    row.names(df) <- 1:nrow(df)
    ## the index of the include document
    include.index <- nrow(df)+1
    ## remove all numbers
    df3 <- stringr::str_replace_all(df$Ingredients,"[^a-zA-Z\\s]", " ")
    my.docs <- VectorSource(c(df3,input$include))

    text <- Corpus(my.docs)
    text2 <- tm_map(text,removePunctuation)
    text3 <- tm_map(text2, removeNumbers)

    dtm <- DocumentTermMatrix(text3,
                              control = list(
                                weighting = function(x) weightTfIdf(x, normalize = FALSE)
                              ))
```

```
 ## select documents have the query terms, extract document ID which has sum.tfidf >0,
 ## use unlist to separate query to individual character
 sum.tfidf <- apply(dtm[,colnames(dtm)%in%unlist(strsplit(input$include," "))],1,sum)
 # print(unlist(strsplit(input$include," ")))
 doc <- which(sum.tfidf>0)
 doc <- unique(doc)
 l <- length(which(sum.tfidf>0))
 ## exclude the last document, which is the input
 l.length <- l-1
 ### cosine similiarity
 print("cosine similiarity")
 dff <- data.frame(document=character(),cosine_similarity=numeric(),stringsAsFactors = F)

 ## exclude the last document, because the last document is query.
 for (j in 1:l.length){
  ## column 1 for document ID
  dff[j,1] <- doc[j]
  ## convert each column of tdm to vector
  dff[j,2] <- cosine(as.vector(dtm[include.index,]),as.vector(dtm[doc[j],]))
  print("here 3")
  print(j)
 }
 ###### ranking cosine similarity, select documnet names of top 20
 top10_doc <- dff[order(dff[,2],decreasing = T),][1:20,]
 top10.records <- df[as.numeric(top10_doc$document),]
 top10.records$total.time <- round(top10.records$total.time,1)
 top10.records <- top10.records[,1:5]
 rownames(top10.records) <- NULL
 print("top10")
 df.category <- df.category
 outdata=list(data1=top10.records,
          data2=df.category)

 print(Sys.time()-t1)
 return(outdata)
})
```

```
 output$table1 <- renderDataTable(data.frame(run.model()$data1))
 output$plot1 = renderPlot({

  data.plot = run.model()$data2
  ggplot(data = data.plot,
         aes(x = category,
             y = mean.number_reviews)) +
    geom_point(size = 5)+labs(x="Recipe Category",y="Mean of the number of reviews")+
    theme(axis.text.y=element_text(size=18),
          axis.text.x=element_text(size=18),
          axis.title.x = element_text(size=18, face="bold"),
          axis.title.y = element_text(size=18, face="bold"),
          legend.position="bottom", legend.title=element_blank(),
          legend.text=element_text(size=14))
 })
 output$plot2 = renderPlot({

  data.plot = run.model()$data2
  ggplot(data = data.plot,
         aes(x = category,
             y = score.sentiment)) +
    geom_point(size = 5)+labs(x="Recipe Category",y="Mean of sentiment scores")+
    theme(axis.text.y=element_text(size=18),
          axis.text.x=element_text(size=18),
          axis.title.x = element_text(size=18, face="bold"),
          axis.title.y = element_text(size=18, face="bold"),
          legend.position="bottom", legend.title=element_blank(),
          legend.text=element_text(size=14))
 })

}

runApp(list(ui=ui,server=server))
```