Project 4 - Main Script

Xuehan Liu, Hongyi Zhu, Mengchen Li

04/05/2017

This file is to combine two paper together and compare the algorithm performance

Paper3: Naive Bayes Algorithm

This file is an attempt to the Naive Bayes algorithm from "Two Supervised Learning Approaches for Name Disambiguation in Author Citations" Han(2004).

Step 0: Load the packages, specify directories

```r
if (!require("pacman")) install.packages("pacman")
```

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

```r
pacman::p_load(text2vec, dplyr, qlcMatrix, kernlab, knitr)

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

```
## Loading required package: splitstackshape

## Loading required package: data.table

## -------------------------------------------------------------------------

## data.table + dplyr code now lives in dtplyr.
## Please library(dtplyr)!

## -------------------------------------------------------------------------

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
library(splitstackshape)


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

```
## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following object is masked from 'package:dplyr':
##
##     count
```

```r
library(matrixStats)

if (!require("reshape2")) install.packages("reshape2")
```

```
## Loading required package: reshape2

##
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

```r
library(reshape2)
```

Step 1: Load and process the data

Please run "data cleaner.Rmd" first so that the .txt files are all in .csv format for processing.

```r
dat = read.csv("../data/nameset/A Kumar.csv")
test = data.frame()
train = data.frame()

# sample the dataset by author ID
for (i in 1:max(dat$author.ID))
{
  sub = subset(dat, author.ID == i)
  sam_index = sample(nrow(sub), floor(nrow(sub)/2))
  sam = sub[sam_index, ]
  train = rbind(train, sam)
  antisam = sub[-sam_index ,]
  test = rbind(test, antisam)
}
rm(sub, sam, i, sam_index, antisam)
```

Step 2: Setting up variable names for Naive Bayes

These are placeholder variables used for the subsequent calculations.

```r
  # number vectors

  # calculate the number of authors in the dataset
  numauthor = max(train$author.ID)
  # to count the number of paper for each author
  numpaper = rep(0, numauthor)
  # to count the number of paper authored with coauthors
  numpapercoauthor = rep(0, numauthor)
  # number of paper the author writes alone
  numpapernocoauthor = rep(0, numauthor)
  # number of coauthors each author has
  numcoauthor = rep("", numauthor)



  # probability vectors

  # initialize P(xi), prior probabilities vector
  p_xi = rep(0, numauthor)
```

```r
# P(seen|Co, Xi)
p_seen.co = rep(0, numauthor)
# initialize P(A1k|xi), likelihood vector
p_A1k.X = data.frame()
# P(A1k|co, seen, Xi)
p_A1k.seen = data.frame()
# P(A1K|co, unseen, Xi)
p_A1k.unseen = data.frame()
# P(A1|X)
p_A1.X = data.frame()
```

Step 3: Calculate the likelihood of seeing feature one, coauthors.

These values are the log of conditional probabilities of author Xi collaborating with each and every one of the coauthors.

```r
# calculate number of paper each author.ID writes
for (i in 1:numauthor)
{
  numpaper[i] = sum(train$author.ID==i)
}

# number of coauthor an author has
numpapercoauthor = numpaper - numpapernocoauthor


# find the number of paper author writes alone
for (i in 1:numauthor)
{
  numpapernocoauthor[i] = sum(train$coauthor.names==""&train$author.ID==i)
}



# extract all coauthors and put them into a list
coauthor.count = data.frame(cbind(train$author.ID, as.character(train$coauthor.names)))

colnames (coauthor.count) = c("author.ID", "coauthor.names")

coauthor.count = cSplit(coauthor.count, "coauthor.names", ",", "long")[
  , list(collaboaration.times = .N), .(author.ID, coauthor.names)][]



# create a complete coauthor matrix
coauthor.matrix = dcast(coauthor.count, author.ID~...)
```

```
## Using collaboaration.times as value column: use value.var to override.
```

```r
if("NA" %in% colnames(coauthor.matrix))
{
  coauthor.matrix$`NA`=NULL
}



coauthor.matrix[is.na(coauthor.matrix)]=0
```

```r
# calculate P(seen|Co, xi) and store in coauthor.matrix$seen
for (i in 1: numauthor)
{
  p_seen.co[i] = sum(as.numeric(coauthor.matrix[i,-1])>1) / sum(as.numeric(coauthor.matrix[i,-1])>0)
}


# filter only coauthors with collaboration times >1
seen <- coauthor.count[coauthor.count$collaboaration.times!=1,]
# convert this count list to a dataframe
coauthor.seen.matrix <- dcast(seen, author.ID~...)
```

```
## Using collaboaration.times as value column: use value.var to override.
```

```r
# change NAs to zero
coauthor.seen.matrix[is.na(coauthor.seen.matrix)]<- 0
if("NA" %in% colnames(coauthor.seen.matrix))
{
  coauthor.seen.matrix$`NA`=NULL
}

# do the same for unseen coauthors:
# unseen coauthors
unseen = coauthor.count[coauthor.count$collaboaration.times==1, ]
# convert list to df
coauthor.unseen.matrix <- dcast(unseen, author.ID~...)
```

```
## Using collaboaration.times as value column: use value.var to override.
```

```r
# change NAs to zero
coauthor.unseen.matrix[is.na(coauthor.unseen.matrix)]<- 0
if("NA" %in% colnames(coauthor.unseen.matrix))
{
  coauthor.unseen.matrix$`NA`=NULL
}


# calculate P(xi)
p_xi = table(train$author.ID)/sum(table(train$author.ID))

# P(N|xi), probability of writing next paper alone
p_numpapernocoauthor = numpapernocoauthor/numpaper


# generate an empty matrix with same dim with coauthor.seen.matrix
p_A1k.seen <- data.frame(matrix(rep(0,ncol(coauthor.seen.matrix)*numauthor),nrow = numauthor))

ID <- coauthor.matrix$author.ID
p_A1k.seen <- data.frame(ID, p_A1k.seen)
colnames(p_A1k.seen) <- colnames(coauthor.seen.matrix)
row.names(p_A1k.seen) <- coauthor.matrix$author.ID

# calculate P(Aik|Seen,Co, xi) and store in A
# remove column titled "NA"
p_A1k.seen <- p_A1k.seen[!is.na(names(p_A1k.seen))]
coauthor.seen.matrix = coauthor.seen.matrix[!is.na(names(coauthor.seen.matrix))]


p_A1k.seen=as.data.frame(p_A1k.seen, stringsAsFactors = False)

for (i in 1: nrow(coauthor.seen.matrix)) {
  for (j in 2: ncol(coauthor.seen.matrix)) {

    if(coauthor.seen.matrix[i,j] !=0)
      p_A1k.seen[i,j] <- as.numeric(coauthor.seen.matrix[i,j])/sum(as.numeric(coauthor.seen.matrix[i,
    else p_A1k.seen[i,j] =  (1/(ncol(coauthor.seen.matrix)-1))/(1+sum(as.numeric(coauthor.seen.matrix


  }
}

# calculate P(A1k|co, unseen, xi)

author.coauthor.total = ncol(coauthor.matrix) + numauthor

for (i in 1: numauthor)
{
  numcoauthor[i] = sum(coauthor.matrix[i,-1]!=0)
}

p_A1k.unseen = 1/(author.coauthor.total - as.numeric(numcoauthor))


## Caculate P(A1k|Xi)

# P(A1k|Seen,Co,Xi)
p_A1k.seen <- as.matrix(p_A1k.seen)
# P(Co|Xi)
p_coauthor <- 1-p_numpapernocoauthor #26*1

# P(A1k|Unseen,Co,Xi)
#dim(p_A1k.unseen) # 26*1
p_A1k.unseen <- as.matrix(p_A1k.unseen)
# P(Unseen|Co,Xi)
p_unseen.co <- 1-p_seen.co

############################################################################


# Caculate P(A1k|Xi)
p_A1k.X <- data.frame (matrix(rep(0,(ncol(p_A1k.seen)-1)*numauthor),nrow = numauthor))# 26*245
  p_A1k.seen <- data.frame(p_A1k.seen)
ID <- coauthor.matrix$ author.ID
#ID <- p_A1k.seen$author.ID
p_A1k.X <- data.frame(ID,p_A1k.X)
colnames(p_A1k.X) <- colnames(p_A1k.seen)
```

```
for (i in 1:max(numauthor))
{
  if(!(i %in% as.numeric(coauthor.unseen.matrix$author.ID)))
  {
    coauthor.unseen.matrix = rbind(coauthor.unseen.matrix, rep(1/sum(as.numeric(numcoauthor)), ncol(co
    coauthor.unseen.matrix[nrow(coauthor.unseen.matrix), 1] = i
  }
}
# creating unseen coauthor probability matrix
for(i in 2:ncol(coauthor.unseen.matrix))
{
  coauthor.unseen.matrix[, i] = p_A1k.unseen
}

#coauthor.unseen.matrix$author.ID = coauthor.seen.matrix$author.ID

p_A1k.X.unseen <- data.frame (matrix(rep(0.1/ncol(coauthor.unseen.matrix),(ncol(coauthor.unseen.matri
ID <- coauthor.matrix$ author.ID
p_A1k.X.unseen <- data.frame(ID,p_A1k.X.unseen)
colnames(p_A1k.X.unseen) <- colnames(coauthor.unseen.matrix)

for (i in 2: ncol(coauthor.unseen.matrix))
{
 p_A1k.X.unseen[, i] <-   as.numeric(as.matrix(coauthor.unseen.matrix[, i])) * p_unseen.co * p_coautho
}

for (i in 2: ncol(p_A1k.X))
{
  p_A1k.X[,i] <-  as.numeric(as.matrix(p_A1k.seen[,i]))*p_seen.co * p_coauthor #+ p_A1k.unseen * p_uns
}

p_A1k.X.final = cbind(p_A1k.X.unseen, p_A1k.X[,-1])

# get rid of leading and trailing white space
## Caculate P(A1|Xi) = P(A11|Xi)...P(A1k|Xi)...P(A1K|Xi)

# the result from A1
p_A1k.X.logged = log(p_A1k.X.final[, -1])

p_A1.X.logged <- rowSums(p_A1k.X.logged)


p_A1.X.logged <- data.frame(ID,p_A1.X.logged)
colnames(p_A1.X.logged) <- c('ID','log(P(A1|Xi))')
```

Step 4: Calculate the likelihood of seeing feature two, keywords in paper title.

These values are the log of conditional probabilities of author Xi writing a paper with a specific keyword in the title.

```r
# extract all paper titles and put them into a list
paper.count <- data.frame(cbind(train$author.ID, as.character(train$paper.title)))
colnames (paper.count) = c("author.ID", "Paper.title")

# clean paper title
library(tm)
```

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

```r
library(tidytext)
str(paper.count$Paper.title)
```

```
##  Factor w/ 120 levels "A CAD integrated analysis of flatness in a form tolerance zone ",..: 57 38 93
```

```r
corpus <- Corpus(VectorSource(paper.count$Paper.title))
corpus <- tm_map(corpus, tolower)
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, PlainTextDocument)
#corpus <- tm_map(corpus, removeWords, stopwords('english'))
#corpus <- tm_map(corpus, stemDocument)

paper.count[,2] <- data.frame(corpus$content$content)

# caculate number of words in sentence and store in paper.count
library(stringr)

word.count <- data.frame(matrix(rep(0,nrow(paper.count)),nrow = nrow(paper.count)))
for(i in 1: nrow(paper.count)){
  word.count[i,] <- str_count(paper.count[i,2], '\\s+')+1
}
colnames(word.count) <-'word.count'
paper.count <- data.frame(paper.count,word.count)

# split sentence into word and store in voc.count
List <- strsplit(as.character(paper.count$Paper.title), " ")
voc.count <- data.frame(author.ID=rep(paper.count$author.ID, sapply(List, length)), Words=unlist(List

# generate a paper matrix
library(reshape2)
paper.matrix <-dcast(voc.count, author.ID~Words)
```

```
## Using Words as value column: use value.var to override.
```

```
## Aggregation function missing: defaulting to length
```

```r
paper.matrix[is.na(paper.matrix)]=0

## calculate P(A2k| Xi)
# generate an empty matrix with same dim with paper.matrix

p_A2k.X <- data.frame(matrix(rep(0,ncol(paper.matrix)*nrow(paper.matrix)),nrow = nrow(paper.matrix)))
author.ID <- paper.matrix$author.ID
p_A2k.X <- data.frame(author.ID, p_A2k.X)
colnames(p_A2k.X) <- colnames(paper.matrix)

p_A2k.X <- p_A2k.X[!is.na(names(p_A2k.X))]
```

```r
paper.matrix <- paper.matrix[!is.na(names(paper.matrix))]



# lol
for (i in 1: nrow(p_A2k.X)) {
  for (j in 2: ncol(p_A2k.X)) {

    if(paper.matrix[i,j] != 0)
      p_A2k.X[i,j] <- as.numeric(paper.matrix[i,j])/sum(as.numeric(paper.matrix[i,]))
    else p_A2k.X[i,j] <- (1/(ncol(paper.matrix)-1))/(1+sum(as.numeric(paper.matrix[i,])))
  }
}
# lol

p_A2k.X.logged = log(p_A2k.X[, -1])

p_A2.X.logged <- rowSums(p_A2k.X.logged)


p_A2.X.logged <- data.frame(ID,p_A2.X.logged)
colnames(p_A2.X.logged) <- c('ID','log(P(A2|Xi))')
```

Step 5: Calculate the likelihood of seeing feature two, keywords in journal title.

These values are the log of conditional probabilities of author Xi publishing in a journal with a name that includes a keyword.

```r
# extract all paper titles and put them into a list
paper.count <- data.frame(cbind(train$author.ID, as.character(train$journal.title)))
colnames (paper.count) = c("author.ID", "journal.title")

# clean journal title
str(paper.count$journal.title)
```

```
##  Factor w/ 99 levels " Artificial Intelligence Engineering",..: 92 88 80 19 27 25 20 31 70 NA ...
```

```r
corpus <- Corpus(VectorSource(paper.count$journal.title))
corpus <- tm_map(corpus, tolower)
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, PlainTextDocument)
#corpus <- tm_map(corpus, removeWords, stopwords('english'))
#corpus <- tm_map(corpus, stemDocument)

paper.count[,2] <- data.frame(corpus$content$content)

# caculate number of words in sentence and store in paper.count
library(stringr)

word.count <- data.frame(matrix(rep(0,nrow(paper.count)),nrow = nrow(paper.count)))
for(i in 1: nrow(paper.count)){
  word.count[i,] <- str_count(paper.count[i,2], '\\s+')+1
}
colnames(word.count) <-'word.count'
```

```
paper.count <- data.frame(paper.count,word.count)

# split sentence into word and store in voc.count
List <- strsplit(as.character(paper.count$journal.title), " ")
voc.count <- data.frame(author.ID=rep(paper.count$author.ID, sapply(List, length)), Words=unlist(List

# generate a paper matrix
journal.matrix <-dcast(voc.count, author.ID~Words)
```

```
## Using Words as value column: use value.var to override.
```

```
## Aggregation function missing: defaulting to length
```

```
journal.matrix[is.na(journal.matrix)]=0

## calculate P(A2k| Xi)
# generate an empty matrix with same dim with journal.matrix

p_A3k.X <- data.frame(matrix(rep(0,ncol(journal.matrix)*nrow(journal.matrix)),nrow = nrow(journal.mat
author.ID <- journal.matrix$author.ID
p_A3k.X <- data.frame(author.ID, p_A3k.X)
colnames(p_A3k.X) <- colnames(journal.matrix)

p_A3k.X <- p_A3k.X[!is.na(names(p_A3k.X))]
journal.matrix <- journal.matrix[!is.na(names(journal.matrix))]


for (i in 1: nrow(p_A3k.X)) {
  for (j in 2: ncol(p_A3k.X)) {
    if(journal.matrix[i,j] != 0)
      p_A3k.X[i,j] <- as.numeric(journal.matrix[i,j])/sum(as.numeric(journal.matrix[i,]))
    else p_A3k.X[i,j] <- (1/(ncol(journal.matrix)-1))/(1+sum(as.numeric(journal.matrix[i,])))
  }
}


p_A3k.X.logged = log(p_A3k.X[, -1])

p_A3.X.logged <- rowSums(p_A3k.X.logged)


p_A3.X.logged <- data.frame(ID,p_A3.X.logged)
colnames(p_A2.X.logged) <- c('ID','log(P(32|Xi))')
```

Step 6: Combining results into one dataframe

Here the dataframe "df" has author ID as rows, and the three features: coauthor, paper title, and journal title as columns.

```
df = cbind(p_A1k.X.logged, p_A2k.X.logged, p_A3k.X.logged)

trim.leading <- function (x)  sub("^\\s+", "", x)
trim.trailing <- function (x) sub("\\s+$", "", x)
```

```
  colnames(df) = trim.leading(trim.trailing(colnames(df)))
  colnames(df) <- sub("\\s+", ".", colnames(df))
  colnames((df))

  df = cbind(df, p_xi)

  i = 1
  while (i < ncol(df))
  {
    if(-Inf %in% df[,i])
    {
      df[, i]=NULL
    }
    i = i+1
  }
```

Step 7: Testing the model

We feed the features one at a time into the dataframe “df” and look up the corresponding conditional probability. We then sum the log of conditional probabilities across all the rows, resulting in a value corresponding to each author ID. Lastly, we find the maximum of the sum, and find the corresponding author ID with which.max function.

```
  # this vector holds the final result of the test: rows correspond to author ID, and columns correspon
  final.result = data.frame(matrix(ncol = 0, nrow =  max(test$author.ID)))
  # actual author ID of the papers
  truth = test$author.ID

## Testing with feature one: coauthors
    for( i in 1:nrow(test))
      {# this holds the result from one paper: row corresponds to author ID,
      # columns correspond to logged probability of feature (A1, A2, A3, but so far only A1 is there)
       current.result = data.frame(matrix(ncol = 0, nrow =  max(test$author.ID)))
       # this holds one paper
       current = test[i, ]
       if(current$coauthor.names!="" & !is.na(current$coauthor.names))
       {
         coauthor.list = trim.trailing(trim.leading(unlist(strsplit(as.character(current$coauthor.names
         coauthor.list = sub("\\s+", ".", coauthor.list)
         # for each coauthor, get the logged probability
         for (j in 1:length(coauthor.list))
        { # 489 unique coauthors
           not_null = !(coauthor.list[j] !="")
           in_colnames = ifelse(not_null, (coauthor.list[j] %in% colnames(df)), FALSE)
           in_colnames = ifelse(is.na(in_colnames), FALSE, coauthor.list[j] %in% colnames(df))

           if(in_colnames)
           {
             current.result = cbind(current.result, as.data.frame(df[, coauthor.list[j]]))
           }
         }
       }
       # rind the transposed column sum of "current.result". This is the probability for A1.
       final.result = rbind(final.result, t(rowSums(current.result)))

    }
  # get the prior probability in the result
  final.result = rbind(final.result, t(rowSums(current.result)), t(df$Freq))


## Testing with feature two: paper title
  for( i in 1:nrow(test))
  {
    current.result = data.frame(matrix(ncol = 0, nrow =  max(test$author.ID)))
    current = test[i, ]
    if(current$paper.title!="" & !is.na(current$paper.title))
    {
      paper.list = trim.trailing(trim.leading(unlist(strsplit(as.character(current$paper.title), ","))))
      # for each paper keyword, get the logged probability
      for (j in 1:length(paper.list))
      {
        not_null = !(paper.list[j] !="")
        in_colnames = ifelse(not_null, (paper.list[j] %in% colnames(df)), FALSE)
        in_colnames = ifelse(is.na(in_colnames), FALSE, paper.list[j] %in% colnames(df))

        if(in_colnames)
        {
          current.result = cbind(current.result, as.data.frame(df[, paper.list[j]]), df$Freq)
        }
      }
    }
    # rind the transposed column sum of "current.result". This is the probability for A2.
    final.result = rbind(final.result, rowSums(current.result))
  }


## Testing with feature three: journal title

  for( i in 1:nrow(test))
  {current.result = data.frame(matrix(ncol = 0, nrow =  max(test$author.ID)))
   current = test[i, ]
    if(current$journal.title!="" & !is.na(current$journal.title))
    {
      journal.list = trim.trailing(trim.leading(unlist(strsplit(as.character(current$journal.title), ",
      # for each journal keyword, get the logged probability
      for (j in 1:length(journal.list))
      {not_null = !(journal.list[j] !="")
        in_colnames = ifelse(not_null, (journal.list[j] %in% colnames(df)), FALSE)
        in_colnames = ifelse(is.na(in_colnames), FALSE, journal.list[j] %in% colnames(df))
        if(in_colnames)
        {
          current.result = cbind(current.result, as.data.frame(df[, journal.list[j]]), df$Freq)
        }
      }
    }
    # rind the transposed column sum of "current.result". This is the probability for A3.
    final.result = rbind(final.result, rowSums(current.result))
  }



## Combining the results
  answer = matrix()
  for(i in 1:nrow(test))
  {
    answer[i] = which.max(as.numeric(final.result[i, ]))
  }

  # this variable pits predicted value with the actual value.
  presentation = cbind(as.data.frame(answer), as.data.frame(truth))
```

Step 8: Evaluation

```
source('../lib/evaluation_measures.R')
matching_matrix_NB <- matching_matrix(test$author.ID, answer)
performance_NB <- performance_statistics(matching_matrix_NB)
compare_df <- data.frame(method=c("Naive Bayes"),
                         precision=performance_NB$precision,
                         recall=performance_NB$recall,
                         f1=performance_NB$f1,
                         accuracy=performance_NB$accuracy)
kable(compare_df,caption="Comparision of performance for Naive Bayes",digits = 2)
```

Table 1: Comparision of performance for Naive Bayes

method	precision	recall	f1	accuracy
Naive Bayes	0.33	0.8	0.47	0.62

Paper5: Error Driven Online Training Algorithm

In this file, we illustrate our step-by-step procedure on the error driven online training algorithm on the nameset AKumar.txt (step0 - step3). We implement the algorithm on all name sets based on the step-by-step procedure described from step0-step3, and saved it in the "lib" folder. In step4, we report the evaluation to all namesets provided.

Step 0: Load the packages, specify directories

```
setwd("~/Desktop/Spr2017-proj4-team-9")
# here replace it with your own path or manually set it in RStudio
# to where this rmd file is located


if (!require("pacman")) install.packages("pacman")
pacman::p_load(text2vec, dplyr, qlcMatrix, kernlab, knitr)
```

```
#Create useable csv for each name set
source("~/Desktop/Spr2017-proj4-team-9/lib/data cleaner.R")
```

Step 1: Load and process the data

```
AKumar <- data.frame(scan("../data/nameset/AKumar.txt",
                          what = list(Coauthor = "", Paper = "", Journal = ""),
                          sep=">", quiet=TRUE),stringsAsFactors=FALSE)

# extract canonical author id befor "_"
  AKumar$AuthorID <- sub("_.*","", AKumar$Coauthor)
  # extract paper number under same author between "_" and first whitespace
  AKumar$PaperNO <- sub(".*_(\\w*)\\s.*", "\\1",  AKumar$Coauthor)
  # delete "<" in AKumar$Coauthor, you may need to further process the coauthor
  # term depending on the method you are using
  AKumar$Coauthor <- gsub("<","",sub("^.*?\\s","",  AKumar$Coauthor))
  # delete "<" in AKumar$Paper
  AKumar$Paper <- gsub("<","", AKumar$Paper)
  # add PaperID for furthur use, you may want to combine all the nameset files and
  # then assign the unique ID for all the citations
   AKumar$PaperID <- rownames( AKumar)
```

Step 2: Feature Design

As mentioned in the paper, we can use TF-IDF to collect all unique terms in each citation.

```
it_train <- itoken(AKumar$Paper,
             preprocessor = tolower,
             tokenizer = word_tokenizer,
             ids = AKumar$PaperID,
             # turn off progressbar because it won't look nice in rmd
             progressbar = FALSE)
vocab <- create_vocabulary(it_train, stopwords = c("a", "an", "the", "in", "on",
                                                   "at", "of", "above", "under"))

#vocab

vectorizer <- vocab_vectorizer(vocab)
dtm_train <- create_dtm(it_train, vectorizer)
dim(dtm_train)
```

```
## [1] 244 666
```

```
tfidf <- TfIdf$new()
dtm_train_tfidf <- fit_transform(dtm_train, tfidf)
```

Step 3: Implementing hierirchical clustering and training parameters

In this section, our goal is to train the lambda on hierirchical clustering on our text file AKumar by the error driven online training method introduced in the paper5. We use the ranking perceptron to update the parameters.

```r
####Initialize Parameter lambda
lambda<-rep(1,nrow(dtm_train_tfidf))

#Add the Author's ID as the label column to the feature matrix for future use
dtm_train_tfidf<-cbind(dtm_train_tfidf,as.numeric(AKumar$AuthorID))


#Given the training set, we are able to generate the true clusters.
#Based on the paper, we define true score S_star as the distance of the sum of clusterwise distance. We


#Compute the true score S_star for the giving training data
element<-list()
S_star<-vector(length=length(unique(AKumar$AuthorID)))
for (i in 1:length(unique(AKumar$AuthorID))){
  element[[i]]<-dtm_train_tfidf[dtm_train_tfidf[,ncol(dtm_train_tfidf)]==i,]
  S_star[i]<-sum(dist(element[[i]]))/2
}
S_star<-mean(S_star)
T_star<-dtm_train_tfidf[,ncol(dtm_train_tfidf)]

K=14
k=1
lambda1 <- matrix(NA, nrow = nrow(AKumar), ncol = K)
S1 <- numeric(K)
acc <- numeric(K)
while (k<(K+1)){

#Implement Hierirchical Clustering
h<-hclust(dist(dtm_train_tfidf*lambda))
#Check the result for the number of cluster equals to the number of unique authors in the dataset.
h_result<-cutree(h,k=length(unique(AKumar$AuthorID)))

#Compute the our own score function S
S<-vector(length=length(unique(AKumar$AuthorID)))
element_s<-list()
for (i in 1:length(unique(AKumar$AuthorID))){
  element_s[[i]]<-dtm_train_tfidf[which(h_result==i),]
  S[i]<-sum(dist(element_s[[i]]))/2
}
S<-mean(S)



#Identify true author for each cluster generated by hclust() function, and assign it to each element of

label<-dtm_train_tfidf[,ncol(dtm_train_tfidf)]
author.clust <- vector(length=length(unique(AKumar$AuthorID)))
for (i in 1:length(unique(AKumar$AuthorID))){
    author.clust[i]<-as.numeric(names(which.max(table(label[which(h_result==i)]))))
}

for (i in 1:unique(AKumar$AuthorID)){
    h_result[h_result==i]<-author.clust[i]
}
T_hat<-h_result

#Update lambda

for (i in 1:length(T_star)){
  if (T_hat[i]!=T_star[i]){
    lambda[i]<-lambda[i]-((S-S_star)/S) #!!!
  }
  else {
    lambda[i]<-lambda[i]
  }
}
lambda1[,k] <- lambda
S1[k] <- S
acc[k] <- mean(label == h_result)
k=k+1
}

#plot(acc)
#lambda1[,K]
h_new<-hclust(dist(dtm_train_tfidf*lambda))
T_hat_overall<-cutree(h_new,k=unique(AKumar$AuthorID))
```

Step 5: Evaluation on all dataset

```
source('../lib/evaluation_measures.R')
source('../lib/data cleaner.R')

#performance statistics for AKumar
matching_matrix_hclust <- matching_matrix(AKumar$AuthorID,T_hat_overall)
performance_hclust.AK <- performance_statistics(matching_matrix_hclust)
performance_hclust.AK
```

```
## $precision
## [1] 0.785056
##
## $recall
## [1] 0.9940082
##
## $f1
## [1] 0.8772613
##
## $accuracy
## [1] 0.9404979
```

```
source('../lib/Y Chen.R')
performance_hclust
```

```
## $precision
## [1] 0.1215654
##
```

```
## $recall
## [1] 0.4054701
##
## $f1
## [1] 0.1870506
##
## $accuracy
## [1] 0.7753102
```

```r
source('../lib/KTanaka.R')
performance_hclust.KT
```

```
## $precision
## [1] 0.5623479
##
## $recall
## [1] 0.663977
##
## $f1
## [1] 0.6089513
##
## $accuracy
## [1] 0.8024834
```

```r
source('../lib/JSmith.R')
performance_hclust.JS
```

```
## $precision
## [1] 0.2017767
##
## $recall
## [1] 0.4320772
##
## $f1
## [1] 0.275089
##
## $accuracy
## [1] 0.7509349
```

```r
source('../lib/MMiller.R')
performance_hclust.MM
```

```
## $precision
## [1] 0.4083057
##
## $recall
## [1] 0.3601701
##
## $f1
## [1] 0.3827303
##
## $accuracy
## [1] 0.5965795
```

```r
source('../lib/MBrown.R')
performance_hclust.MB
```

```
## $precision
## [1] 0.2902839
##
## $recall
## [1] 0.8924401
##
## $f1
## [1] 0.4380751
##
## $accuracy
## [1] 0.6796526
```

```r
source('../lib/MJones.R')
performance_hclust.MJ
```

```
## $precision
## [1] 0.3828712
##
## $recall
## [1] 0.692079
##
## $f1
## [1] 0.4930036
##
## $accuracy
## [1] 0.8009207
```

```r
source('../lib/JLee.R')
performance_hclust.JL
```

```
## $precision
## [1] 0.08453005
##
## $recall
## [1] 0.6672021
##
## $f1
## [1] 0.1500498
##
## $accuracy
## [1] 0.8175904
```

```r
source('../lib/JMartin.R')
performance_hclust.JM
```

```
## $precision
## [1] 0.132294
##
## $recall
## [1] 0.9769737
##
## $f1
## [1] 0.2330326
##
## $accuracy
## [1] 0.3709781
```

```r
source('../lib/JRobinson.R')
performance_hclust.JR
```

```
## $precision
## [1] 0.2897944
##
## $recall
## [1] 0.7596567
##
## $f1
## [1] 0.4195417
##
## $accuracy
## [1] 0.696732
```

```r
source('../lib/DJohnson.R')
performance_hclust.DJ
```

```
## $precision
## [1] 0.3393976
##
## $recall
## [1] 0.4149859
##
## $f1
## [1] 0.3734048
##
## $accuracy
## [1] 0.6124422
```

```r
source('../lib/CChen.R')
performance_hclust.CC
```

```
## $precision
## [1] 0.1275084
##
## $recall
## [1] 0.6198893
##
## $f1
## [1] 0.2115102
##
## $accuracy
## [1] 0.770671
```

```r
source('../lib/AGupta.R')
performance_hclust.AG
```

```
## $precision
## [1] 0.2103288
##
## $recall
## [1] 0.6031183
##
## $f1
## [1] 0.3118903
```

```
##
## $accuracy
## [1] 0.7380789
```

```
#compute average of the all four performance statistics
Precision<-c(performance_hclust.AK$precision,performance_hclust$precision,performance_hclust.MB$precisio

Recall<-c(performance_hclust.AK$recall,performance_hclust$recall,performance_hclust.MB$recall,performanc


F1<-c(performance_hclust.AK$f1,performance_hclust$f1,performance_hclust.MB$f1,performance_hclust.KT$f1,p



Accuracy<-F1<-c(performance_hclust.AK$accuracy,performance_hclust$accuracy,performance_hclust.MB$accurac

#Final Performance Summary combining all 14 namesets.
error.driven<-data.frame(Precision=mean(Precision),Recall=mean(Recall),F1=mean(F1),Accuracy=mean(Accurac
error.driven
```

```
##   Precision    Recall        F1  Accuracy
## 1 0.3027739 0.6524652 0.7194517 0.7194517
```

Compare the performance of two algorithms

```
compare_df <- data.frame(method=c("Naive Bayes","Error Driven Algorithm"),
                         precision=c(performance_NB$precision,error.driven$Precision),
                         recall=c(performance_NB$recall,error.driven$recall),
                         f1=c(performance_NB$f1,error.driven$F1),
                         accuracy=c(performance_NB$accuracy,error.driven$Accuracy))
kable(compare_df,caption="Comparision of performance for Naive Bayes and Error Driven Online Training",c
```

Table 2: Comparision of performance for Naive Bayes and Error Driven Online Training

method	precision	recall	f1	accuracy
Naive Bayes	0.33	0.8	0.47	0.62
Error Driven Algorithm	0.30	0.8	0.72	0.72