ITESO

Bag of Words Meets Bags of Popcorn

Sentiment Analysis and Text Mining

Sara Eugenia Rodríguez Reyes

26/11/2015

Los Datos

Kaggle alberga varios desafíos científicos de datos que van desde tutoriales dirigidos a análisis de datos, hasta competencias con premios de $100,000.

Un desafío se llama "Bag of Words Meets Bags of Popcorn" que tiene el objetivo de formar un algoritmo que predice sentimientos en cuanto a la reseña de una película.

Los datos consisten en 25,000 reviews unidos a un score binario: 0 si la calificación de la película fue menor 5, o 1 si es mayor a 6. El objetivo es unir las puntuaciones con otros 25,000 reviews que no han sido etiquetados y hacer predicciones.

Se tienen 4 archivos:

- labeledTrainData: set de entrenamiento. Este archivo tiene una fila de encabezado seguido por 25,000 filas que contienen un id, sentimiento y texto de cada review.
- testData: set de prueba. Tiene una fila de encabezado seguido con 25,000 filas que contiene un id y texto para cada review. La tarea es predecir el sentimiento para cada una.
- unlabeledTrainData: un set de entrenamiento extra sin etiquetas. Tiene una fila con encabezado seguido con 50,000 filas que contienen un id y texto para cada review.
- sampleSubmission

Descripción de las columnas:

- id: ID único para cada review
- sentiment: sentimiento del review; 1 para reviews positivos y 0 para negativos
- review: texto con el review

Los reviews primero son limpiados de puntuaciones y señales tipo HTML

```
#Combinar los datos
all.data <- rbind(train.labeled[,-2], train.unlabeled,test)
all.data$sentiment <- c(train.labeled$sentiment,
rep(NA,nrow(train.unlabeled)+nrow(test)))
#índice
train.labeled.ind <- 1:nrow(train.labeled)
train.ind <- 1:(nrow(train.labeled)+nrow(train.unlabeled))
test.ind <- (nrow(train.labeled)+nrow(train.unlabeled)+1):nrow(all.data)
#Limpiar datos
#Quitar señales HTML
all.data$review.clean <- gsub('<.*?',' ',all.data$review)
#Quitar puntuaciones
all.data$review.clean <- tolower(gsub('[[:punct:]]', ' ',
all.data$review.clean))
```

AFINN List

La Universidad de Dinamarca provee una lista con 2,477 palabras y frases etiquetadas con rangos de sentimientos entre -5 y 5; esta lista es llamada AFINN.

```
#Leer la lista AFINN
afinn <- read.delim('./AFINN-111.txt', header=F, quote='', stringsAsFactors=F)
names(afinn) <- c('word', 'score')

#Los espacios están con "-" hay que limpiarlo
afinn$word.clean <- gsub('-', ' ', afinn$word)
#Quitar apóstrofes
afinn$word.clean <- gsub("[[:punct:]]",' ', afinn$word.clean)
```

La lista AFINN se puede encontrar en:

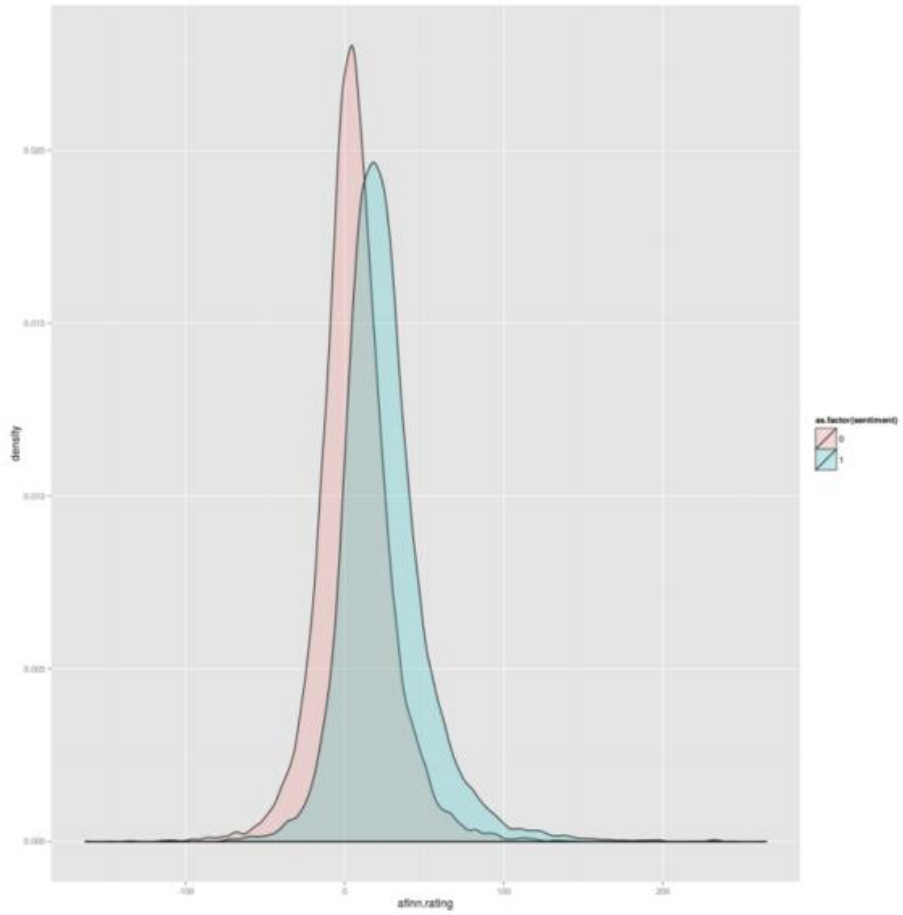
http://www2.imm.dtu.dk/pubdb/views/publication.details.php?id=6010.

Un método rápido sería sólo contar el número de veces que cada término ocurre en cada review y hacer un producto punto por el vector correspondiente de sentimiento. Aquí se usa la función str.count() del paquete stringr para contar el número de instancias de cada término AFINN en cada review para obtener una matriz de frecuencias de texto.

```
#Encontrar la matriz de frecuencias de texto
require(stringr)
tf <- t(apply(t(all.data$review.clean),2,function(x) str_count(x,
afinn$word.clean)))
#rating de sentimientos por cada fila
all.data$afinn.rating <- as.vector(tf %*% afinn$score)
```

Lo bueno de este método es que es muy simple y sólo utiliza un predictor entre las dos clases.

```
require (ggplot2)
ggplot (all.data[train.labeled.ind, ], aes(afinn.rating,
fill=as.factor(sentiment)))+geom_density(alpha = .2)
```



Como las distribuciones son aproximadamente normales, vamos a intentar entrenando un Clasificador Bayesiano a los datos.

```
require (e1071)
#ajustar un clasificador de Bayes
nb.model <- naiveBayes(sentiment~afinn.rating, data=
all.data[train.labeled.ind,])
#Valores predichos
nb.pred <- predict(nb.model, newdata=all.data[test.ind,],type='raw')
#Resultados
results <- data.frame(id=all.data$id[test.ind],sentiment=nb.pred[,2])
results$id <- gsub('"','',results$id)
```

Como era de esperarse, los resultados no son tan buenos, dando un desempeño de 0.71516. Pero es un comienzo.

Bag of Scores

Este método en lugar de aprender características vía word2vec y k-means clustering, vamos a usar la lista AFINN. En lugar de añadir los scores de la lista, usamos esos scores como clusters. Para cada fila, decimos que hay x_{-5} términos con la etiqueta de -5, x_{-4} téminos con la etiqueta -4 y así seguimos.

```
#rango de scores de -5 a 5
score.vectorizer <- function(sentence,words,word.scores){
  score.vector <- rep(0,length(table(word.scores)))
  k<-0
  for (i in as.numeric(names(table(word.scores)))){
    k <- k+1
    tempwords <- words[word.scores==i]
    score.vector[k]<-sum(str_count(sentence,tempwords))
  }
  return(score.vector)
}
score.data <- as.data.frame(t(apply(t(all.data$review.clean),2,function(x)
score.vectorizer(x,afinn$word.clean,afinn$score))))
#name columns
names(score.data) <-
c('n5','n4','n3','n2','n1','zero','p1','p2','p3','p4','p5')
```

Ahora ajustamos un random forest de clasificación a los datos:

```
require(randomForest)
bag.of.scores.forest <-
randomForest(score.data[train.labeled.ind,],as.factor(train.labeled$sentiment))
bag.of.scores.forest.pred<-
predict(bag.of.scores.forest,newdata=score.data[test.ind,],type='prob')
results <-
data.frame(id=all.data$id[test.ind],sentiment=bag.of.scores.forest.pred[,2])
results$id <- gsub('"','',results$id)
```

Este método tampoco tiene muy buen desempeño, 0.7858. Parece que agregando el score de AFINN no es un buen método.

Term Frequency-Inverse Document Frequency

Un método comúnmente utilizado para ponerle una puntuación a la importancia de una frase en un documento es el "term frequency-inverse document score", el cual compara la frecuencia de la frase en el documento con la frecuencia de ocurrencia en una colección de documentos.

El score del tf-idf es calculado en dos partes. La primera parte, la cual es la frecuencia del término, es simplemente la frecuencia de un término en un documento.

La segunda parte es el logaritmo del número de documentos en la colección dividido entre el número de documentos que contienen el término.

El score tf-idf es simpelemente el producto de tf e idf. Por ejemplo, si la palabra "popcorn" aparece dos veces en un review, y 314 reviews de 75,000 contienen la palabra "popcorn", el score tf-idf de la palabra "popcorn" para ese review sería 2*log(75000/314)=4.756. Aquí es donde el set de entrenamiento "unabeled" viene en juego. Por lo general este método nos da una mejor métrica cuando la colección de documentos es larga.

Lo primero que tenemos que decidir es cuáles palabras vamos a utilizar. Ya que tenemos una matriz tf de las palabras de la lista AFINN, empezamos con eso. Pero primero, es una buena idea convertirla en un data frame.

```
tf <- as.data.frame(tf)
```

A continuación, necesitamos nuestro vector idf. Para esto, vamos a usar ambos data sets de entrenamiento "labeled" y "unabeled".

```
idf <- log(length(train.ind)/colSums(sign(tf[train.ind,])))
```

Finalmente, obtenemos la matriz td-idf:

```
tf.idf <- as.data.frame(t(apply(tf,1,function(x)x*idf)))
```

Nótese que la palabra "absentees", una palabra de AFINN, no está presente en ninguno de los reviews. Esto significa que el scores td-idf para "absentees" no está definido para todos los reviews. Para esto podemos sobreescribir los valores no definidos como 0. Es más fácil manipular el vector idf primero antes de construir el data frame de tf.idf.

```
#Quitar valores infinitos de idf ya que convergen a cero cuando programamos tf-
idf
idf[is.infinite(idf)]<-0
tf.idf <- as.data.frame(t(apply(tf,1,function(x)x*idf)))
```

Finalmente, ajustamos un random forest de clasificación para los datos

```
#ajustar clasificador
tfidf.forest <-
randomForest(tf.idf[train.labeled.ind,],as.factor(all.data$sentiment[train.labe
led.ind]),ntree=100)
#Predicciones
tfidf.forest.pred <-
predict(tfidf.forest,newdata=tf.idf[test.ind,],type='prob')
results <- data.frame(id=all.data$id[test.ind],sentiment=tfidf.forest.pred[,2])
results$id <- gsub('"','',results$id)
```

El resultado incrementó, ahora es de 0.85, aunque podría ser mejor. Parece que la lista de palabras AFINN no nos da muy buenas características. Esta lista es mejor para inferir sentimientos de datos sin etiquetar que para un algoritmo supervisado. A continuación, vamos a ver cómo podemos extraer características usando sólo los datos sin etiquetas predefinidas.

Construir una matriz de frecuencia desde Corpus

En el tutorial de Kaggle, se construyen predictores basados en 5,000 de las palabras más frecuentes en el cuerpo de los reviews. Se puede hacer algo similar en R con el paquete "tm".

Empezamos construyendo un cuerpo, del cual podemos construir una matriz tf de todas las palabras que contiene.

```
require(tm)
#Construir corpus (usando solo datos de entrenamiento)
corpus <- Corpus(VectorSource(all.data$review.clean[train.ind]))
#matriz tf usando todas las palabras (menos las stop words)
#La lista de stop words se puede ver en:
#http://jmlr.csail.mit.edu/papers/volume5/lewis04a/all-smart-stop-
list/english.stop
tf <- DocumentTermMatrix(corpus,
control=list(stopwords=stopwords('english'),removeNumbers=T))
```

Después de quitar las stop words, terminamos con una matriz muy grande. Muchas de estas palabras ocurren muy infrecuentemente. Podemos remover términos escasos y ponerlos como una matriz de densidad.

```
#solo incluir palabras que ocurren al menos en el 0.1% de los reviews
tf <- removeSparseTerms(tf,.999)
#convertir a una matriz de densidad para un análisis más fácil
tf <- as.matrix(tf)
head(colnames(tf))
```

Ahora tenemos datos 9,799 columnas. Veamos algunas de esas palabras.

aaron	abandon	abandoned	abbott	abc	abducted

Adicionalmente creamos un data frame con las frecuencias de las palabras.

```
#suma de las columnas para encontrar las ocurrencias de cada palabra en el
cuerpo
word.freq <- colSums(tf)
word.freq <- data.frame(word=names(word.freq),freq=word.freq)
rownames(word.freq)<- NULL
head(word.freq)
```

word	freq
aaron	145
abandon	146
abandoned	586
abbott	176
abc	204
abducted	118

O si queremos ver los términos más frecuentes

```
head(word.freq[order(-word.freq$freq),])
```

word	freq
movie	125307
film	113054
one	77447
like	59147
just	53132
good	43279

El siguiente acercamiento busca mejorar las características de la bolsa de palabras construyendo características en palabras que ocurren más veces en reviews positivos que en reviews negativos.

Índice de sentimiento de diferencia normalizada

El problema de usar las "N" palabras más frecuentes como características es que algunas palabras son muy comunes sin importar el sentimiento. Esperamos que las palabras "movie" y "film" sean comunes en ambos reviews, positivos y negativos. Una forma de eliminar esas palabras es considerando la diferencia de frecuencias de los reviews negativos de los positivos. Para empezar, construimos una lista de frecuencias para ambos reviews. Desafortunadamente, esto significa que no podemos usar el set de entrenamiento de 50,000 datos sin etiquetar.

```
word.freq <- function(document.vector, sparsity=.999)
{
  #Construir cuerpo
  temp.corpus <- Corpus(VectorSource(document.vector))
  #Construir matriz tf y remover términos escasos
  temp.tf <-
DocumentTermMatrix(temp.corpus,control=list(stopwords=stopwords('english'),remo
veNumbers=T))
  temp.tf <- removeSparseTerms(temp.tf,sparsity)
  temp.tf <- as.matrix(temp.tf)
  #construir data frame con frecuencia de palabras
  freq.df <- colSums(temp.tf)
  freq.df <- data.frame(word=names(freq.df),freq=freq.df)
  rownames(freq.df)
  return(freq.df)
}
word.freq.pos <- word.freq(all.data$review.clean[all.data$sentiment==1])
word.freq.neg <- word.freq(all.data$review.clean[all.data$sentiment==0])
head(word.freq.pos)
head(Word.freq.neg)
```

word (positive)	freq
ability	214
able	717
absolute	153
absolutely	642
academy	190
accent	173

word (negative)	freq
abandoned	214
ability	717
able	153
absolute	642
absolutely	190
absurd	173

A continuación, unimos las dos tablas, las limpiamos y vemos las diferencias en frecuencias.

```
#unir por palabra
freq.all <- merge(word.freq.neg, word.freq.pos, by='word', all=T)
#Limpiar
freq.all$freq.x[is.na(freq.all$freq.x)]<- 0
freq.all$freq.y[is.na(freq.all$freq.y)]<-0
#Diferencias
freq.all$diff <- abs(freq.all$freq.x-freq.all$freq.y)
```

Ahora podemos ver algunas de las palabras más importantes.

```
head(freq.all[order(-freq.all$diff),])
```

word	Freq.x	Freq.y	Diff
Movie	23668	18139	5529
Bad	7089	1830	5259
Great	2601	6294	3693
Just	10535	7098	3437
Even	7604	4899	2705
worst	2436	246	2190

Como fue esperado, vemos palabras como “great”, “bad” y “worst”, pero también vemos la palabra “movie” a pesar de que no tiene ningún valor sentimental. Esto es probable porque “movie” es tan común que cualquier diferencia pequeña es proporcionalmente grande. Para arreglar esto, se introduce una métrica llamada “normalized difference sentiment index (NDSI)”, la cual toma esto en cuenta.

En otras palabras, NDSI es la diferencia de frecuencias normalizada por su suma. Los valores están entre 0 y 1 con valores mayores indicando mayor correlación con sentimientos.

Antes de implementar esto, hay que notar que algunas palabras que ocurren en reviews positivos no ocurren en todos los reviews negativos (o viceversa). Necesitamos penalizar palabras infrecuentes. Si una palabra ocurre solo una vez en un review positivo y en ningún negativo, le ponemos un valor al NDSI de 1 aunque sepamos que no es un muy buen predictor de sentimiento. Para remediar esto, añadimos un término de suavización que penalice palabras infrecuentes.

```
#Término de suavización
alpha <- 2**7
#NDSI
freq.all$ndsi <- abs(freq.all$freq.x-
freq.all$freq.y)/(freq.all$freq.x+freq.all$freq.y+2*alpha)
```

Ahora vemos cuáles términos fueron elegidos como mejores predictores.

```
head(freq.all[order(-freq.all$ndsi),])
```

word	Freq.x	Freq.y	diff	Ndsi
Worst	2436	246	2190	0.7454
Waste	1351	94	1257	0.7389
Poorly	620	0	620	0.7077
Lame	618	0	618	0.7070
Awful	1441	159	1282	0.6907
mess	498	0	498	0.6604

Ahora podemos usar N palabras con el valor más alto de NDSI para hacer un data frame del clasificador tf-idf y ver nuestros resultados.

```
#número de palabras a considerar en la matriz tf-idf
num.features <- 2**10
#ordenar la frecuencia
freq.all<- freq.all[order(-freq.all$ndsi),]
#word to string
freq.all$word <- as.character(freq.all$word)
#construir la matriz tf
tf <-
t(apply(t(all.data$review.clean),2,function(x)str_count(x,freq.all$word[1:num.f
eatures])))
#vector idf
idf <- log(length(train.ind)/colSums(sign(tf[train.ind,])))
idf[is.infinite(idf)]<-0
#tf-idf matriz
tf.idf <- as.data.frame(t(apply(tf,1,function(x)x*idf)))
colnames(tf.idf)<- freq.all$word[1:num.features]
#entrenar random forest
ndsi.forest<-
randomForest(tf.idf[train.labeled.ind,],as.factor(all.data$sentiment[train.labe
led.ind]),ntree=100)
#predecir
ndsi.pred <- predict(ndsi.forest,newdata=tf.idf[test.ind,],type='prob')
results<-
data.frame(id=all.data$id[test.ind],sentiment=bag.of.scores.forest.pred[,2])
```

El resultado mejoró de gran manera ahora con un valor de 0.91068, una gran mejora comparado en el método AFINN, usando menos de la mitad del número de predictores. Ese resultado fue obtenido usando el set de entrenamiento.

Optimización

Hasta ahorita, los métodos usados han sido ineficientes ya que construimos las matrices nosotros mismos (y tarda mucho el código en correr), la buena noticia es que existen métodos pre-construidos que pueden ayudar. La función DocumentTermMatrix() es una.

```
tf<- tf[,colnames(tf)%in% freq.all$word]
```

El paquete NLP provee varios métodos. Si queremos programar una matriz tf-idf con el paquete tm.

```
tf.idf <-
DocumentTermMatrix(corpus,control=list(stopwords=stopwords('english'),removeNum
bers=T,weighting=function(x)weightTfIdf(x,normalize=F)))
```

Cosine Similarity

Es una medida de similitud entre el espacio dos vectores, el producto interno mide el coseno del ángulo entre ellos. El coseno de 0° es 1, y es inferior a 1 para cualquier otro ángulo. Dos vectores con la misma orientación tienen una similitud coseno de 1, dos vectores a 90° tienen una similitud de 0, y dos vectores opuestos tienen una similitud de -1 independientemente de su magnitud.

En text mining, a cada término se le asigna una dimensión diferente y a cada documento se le caracteriza por un vector, donde el valor de cada dimensión corresponde a la cantidad de veces que aparece el término en el documento. El coseno es una medida útil para ver lo propensos que son los términos en los documentos.

Para este ejemplo, se encontrará el coseno entre los reviews de las películas; obteniendo como producto final un data frame con valores de la similitud que tienen los vectores de reviews. Los valores varían entre 0 y 1, siendo más cercano al uno cuando la similitud es mayor.

```
#Cargar librerías
library(tm)
library(XML)
#Seleccionar directorio
setwd("~/ITESO/PAP2/BagOfWords")
#Cargar datos
reviews <- read.delim('./labeledTrainData.tsv',header=T,quote='',
stringsAsFactors = F)
train.unlabeled <- read.delim('./unlabeledTrainData.tsv',header=T,quote='',
stringsAsFactors = F)
test <- read.delim('./testData.tsv',header=T,quote='', stringsAsFactors = F)
review_text <- train.labeled[1:25000,3] #Tomamos los reviews
review_source <- VectorSource(review_text) #Se transforman en vectores
```

Una vez que los documentos están cargados correctamente, se van a pre-procesar los textos. Este paso permite eliminar números, mayúsculas, palabras comunes, puntuación, etc. Esto puede ser un poco lento y exigente, pero vale la pena al final, en términos de análisis de mejor calidad.

```
corpus <- Corpus(review_source)
#Cambiar de mayúsculas a minúsculas
corpus <- tm_map(corpus, content_transformer(tolower))
#Quitar puntuación
corpus <- tm_map(corpus, removePunctuation)
#Quitar espacios en blanco
corpus <- tm_map(corpus, stripWhitespace)
```

Extracción de " palabras comunes " que por lo general no tienen valor analítico.
En cada texto, hay un montón de éstas, y las palabras sin interés (a, y, también, la, etc.). Tales palabras son frecuentes, por su naturaleza, y confunden el análisis si permanecen en el texto.

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

Se crea un documento en forma de matriz

```
dtm <- DocumentTermMatrix(corpus)
dtm2 <- as.matrix(dtm)
```

Frecuencia de las palabras y creación matriz tf-idf.

```
frequency <- colSums(dtm2)
frequencyT <- sort(frequency, decreasing=TRUE)
#Palabras por cada review
tf <- list()
idf <- list()
datos <- list()
not <- list()
peso <- list()
for(i in 1:length(review_text)){
  review_source <- VectorSource(reviews$review[i])
  corpus <- Corpus(review_source)
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, stripWhitespace)
  corpus <- tm_map(corpus, removeWords, stopwords("english"))
  stopwords("english")
  dtm <- DocumentTermMatrix(corpus)
  dtm2 <- as.matrix(dtm)
  frequency <- colSums(dtm2)
  frequency <- sort(frequency, decreasing=TRUE)
  datos[[i]] <- frequency
  ban <- frequencyT
  ban[which(names(frequencyT)%in% names(datos[[i]]))] <- datos[[i]]
  ban[which(!(names(frequencyT)%in% names(datos[[i]])))] <- 0
  #TF es el numero de veces que el termino t ocurre en el documento d
  tf[[i]] <- ban/length(datos[[i]])
  #IDF numero total de documentos por el numero de documentos que contienen el
termino
  idf[[i]] <- length(datos)/frequencyT
  not[[i]] <- idf[[i]] * tf[[i]]
  peso[[i]] <- not[[i]]/sqrt(sum(not[[i]]^2))
}
```

A continuación se calcula la similitud del coseno de todos los reviews en una matriz. Se muestra una muestra de la matriz.

```
datos2 <- data.frame()
for(i in 1:length(review_text)){
  for(j in 1:length(review_text)){
    datos2[i,j] <- cos(sum(peso[[i]]*peso[[j]]))
  }
}
```

1	0.99999	1	1	1	1	1	1
0.999996	1	1	1	1	1	1	1
1	1	0.99999	1	1	1	0.9999	0.9999

Se observa que de los primeros reviews, tienen una distancia muy cercana.

Clustering

```
#Cargar librerías
library (tm)
library (SnowballC)
library (RColorBrewer)
library (ggplot2)
library (wordcloud)
library (biclust)
library (cluster)
library (igraph)
library (fpc)
install.packages("Rcampdf", repos = "http://datacube.wu.ac.at/", type =
"source")
```

Se crea un documento en forma de matriz, luego se transpone

```
dtm <- DocumentTermMatrix(docs)
tdm <- TermDocumentMatrix(docs)
```

Se organizan los términos por su frecuencia

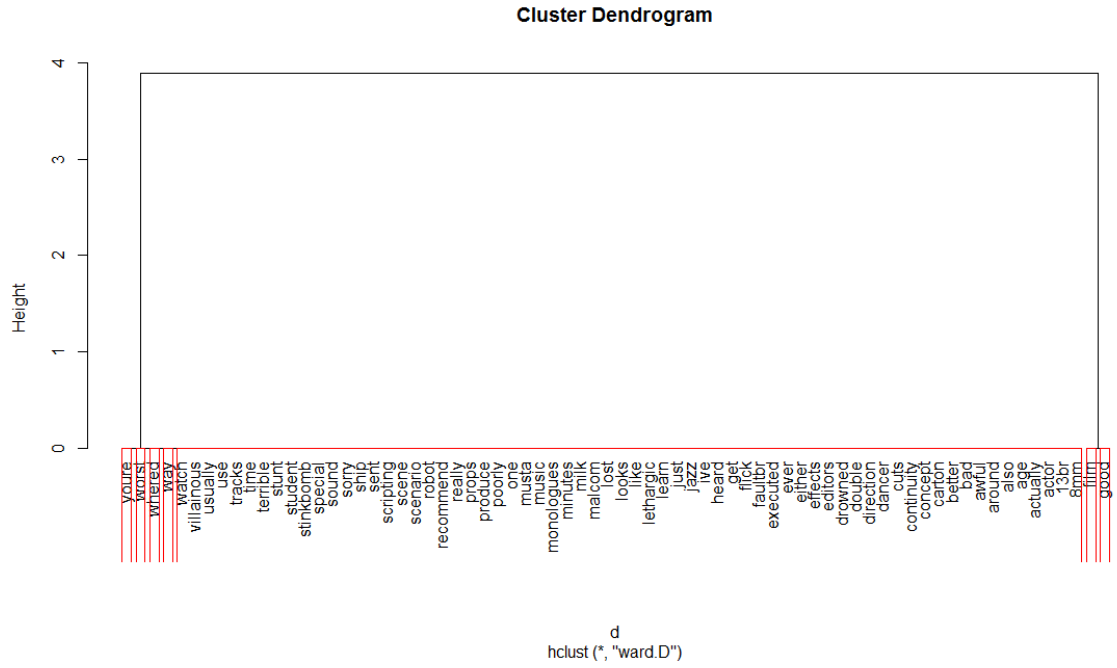
```
 freq <- colSums(as.matrix(dtm))
length(freq)
ord <- order(freq)
```

Remover términos dispersos. Se hace una matriz con el 10% de espacio vacío.

```
dtms <- removeSparseTerms(dtm, 0.1)
inspect(dtms)
```

Frecuencia de las palabras y cluster

```
freq[head(ord)]
freq[tail(ord)]
head(table(freq), 20)
tail(table(freq), 20)
freq <- colSums(as.matrix(dtms))#Tabla de los términos elegidos cuando se
removieron los términos dispersos
wf <- data.frame(word=names(freq), freq=freq)
head(wf)
d <- dist(t(dtms), method="euclidian")
fit <- hclust(d=d, method="ward")
fit
plot.new()
plot(fit, hang=-1)
groups <- cutree(fit, k=7)
rect.hclust(fit, k=7, border="red")
```



Se hicieron 7 clusters; se observa que las palabras más similares son: film y good; youre y worst.

Conclusión

Este es el fin de la demostración, pero se puede hacer más. Añadir más predictores puede mejorar las predicciones pero tiene más costo computacional. Mientras que el método tf-idf es bueno para ver qué tan importante es una palabra para un documento, no considera el significado de las palabras ni similitudes. Finalmente, ninguno de estos métodos considera el orden de las palabras ni el contexto (ej. “very awesome” y “not awesome”).

Bibliografía

- “Bag of Words Meets Bags of Popcorn” https://www.kaggle.com/c/word2vec-nlptutorial
- AFINN list. http://www2.imm.dtu.dk/pubdb/views/publication.details.php?id=6010
- Wikipedia tf-idf. https://en.wikipedia.org/wiki/Tf%E2%80%93idf