Bag of Words Classification, Word Embeddings, Skip-Gram/CBOW Models Analysis

Mohammed Alrashidan

```
source("/Users/mo/Desktop/Desktop/School/USF/Courses/Fall 2021/NLP/Functions/load_NLP_env.R")
path <- "/Users/mo/Desktop/Desktop/School/USF/Courses/Fall 2021/NLP/Functions/"
load_NLP_env(path)
```

```
## [1] "functions loaded:  "
```

```
library(Rcpp)
library(text2vec)
```

```
file_path <- "/Users/mo/Desktop/Desktop/School/USF/Courses/Fall 2021/NLP/datasets/amazon_reviews.csv"
amazon <- read.csv(file_path)
amazon <- head(amazon,200000)
```

Convert this to a binary classifier where overall scores of 4 and 5 are 'positive,' and other values are 'negative.'

```
library(dplyr)
amazon$pos_neg <- ifelse(amazon$overall == 4 | amazon$overall == 5, 1, 0)

review_stat <- amazon %>% group_by(pos_neg) %>% tally()
review_stat$percent <- review_stat$n / sum(review_stat$n)*100
```

We can see that we have 81% 1s which is positive reviews compared to 18% of negative reviews

```
# cleaning the amazon reviews
text <- pre_process_corpus(amazon, "reviewText", replace_numbers = T, root_gen = "lemmatize")
amazon$review_preprocessed <- text
amazon$review_preprocessed[1]
```

```
## [1] "one thing book seem obvious original think however clarity author explain innovation happen remarkable al
an gregerman discuss mean human interaction kind situation tend inspire original clear think lead innovation thin
g include people communicate certain situation outside normal pattern gregerman identify ingredient make innovati
on likely include people compel interact normally lead serendipity sometimes phenomenon will occur collaboration
sometimes chance individual away home travel recommend book common sense truth apparent mastery subject author"
```

Construct a DTM that includes unigrams, bigrams, and trigrams.

```
# preparing the data for analysis.
# spliting the data to train and test
rand <- runif(nrow(amazon))
sets <- ifelse(rand < 0.9, "train", "test")
amazon$set <- sets

train <- amazon[amazon$set == "train",]
```

```
# constructing unigrams, bigrams, and trigrams.
it_train <- itoken(train$review_preprocessed, tokenizer = word_tokenizer, ids = train$id)
vocab <- create_vocabulary(it_train, ngram = c(1,3))

lbound <- round(0.009 * nrow(train))

vocab <- vocab[vocab$doc_count > lbound,]
head(vocab)
```

```
## Number of docs: 180161
## 0 stopwords:  ...
## ngram_min = 1; ngram_max = 3
## Vocabulary:
##            term term_count doc_count
## 1:   fast_ship       1625      1623
## 2: fit_perfect       1667      1663
## 3:     buy_new       1675      1627
## 4:    easy_use       1684      1654
## 5:     satisfy       1685      1662
## 6:       guess       1703      1629
```

```
vectorizer <- vocab_vectorizer(vocab)

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

```
## [1] 180161    425
```

```
test <- amazon[amazon$set == "test",]
it_test <- itoken(test$review_preprocessed,
                  tokenizer = word_tokenizer, ids = test$id)

dtm_test <- create_dtm(it_test, vectorizer)
dim(dtm_test)
```

```
## [1] 19839   425
```

Regualrizartion Method using Lasso

```
library(glmnet)
```

```
## Loaded glmnet 4.1-2
```

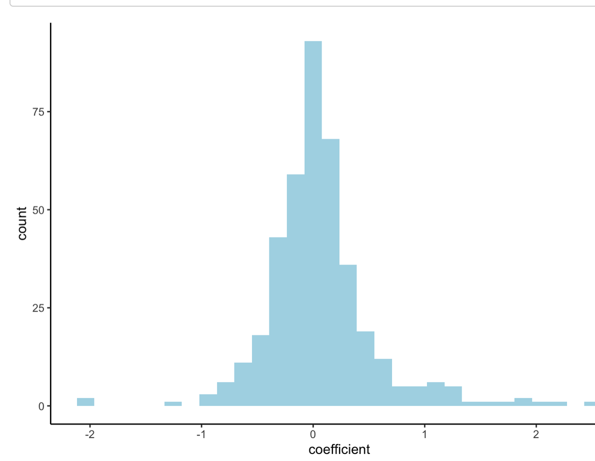
```
model_dtm <- cv.glmnet(x = dtm_train, y = train$pos_neg, type.measure = "auc",
                       family = "binomial", alpha = 1)
```

```
coefs <- coef(model_dtm, s = 'lambda.min')
coefs <- data.frame(name = coefs@Dimnames[[1]][coefs@i + 1], coefficient = coefs@x)

print((nrow(coefs)/ncol(dtm_train)))
```

```
## [1] 0.9411765
```

```
ggplot(coefs, aes(coefficient)) + geom_histogram(fill = 'lightblue') + theme_classic()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
pred_test <- predict(model_dtm, dtm_test, type= 'response')[,1]

thresh <- 0.5
table(test$pos_neg, pred_test > thresh)
```

```
##
##     FALSE  TRUE
##   0  1555  2118
##   1   410 15756
```

3- Using the same techniques we covered in lecture, estimate a binomial logistic regression model. Extract or calculate your model's AUC value.

```
glmnet:::auc(test$pos_neg, pred_test > thresh)
```

```
## [1] 0.6989989
```

```
#model 2

model_dtm <- cv.glmnet(x = dtm_train,
                       y = train$pos_neg,
                       type.measure = "auc",
                       family = "binomial",
                       alpha = 1,
                       nfolds = 3)

coefs <- coef(model_dtm, s = 'lambda.min')
coefs <- data.frame(name = coefs@Dimnames[[1]][coefs@i + 1], coefficient = coefs@x)

print((nrow(coefs)/ncol(dtm_train)))
```

```
## [1] 0.9411765
```

```
pred_test <- predict(model_dtm, dtm_test, type= 'response')[,1]

thresh <- 0.5
table(test$pos_neg, pred_test > thresh)
```

```
##
##     FALSE  TRUE
##   0  1512  2161
##   1   391 15775
```

```
glmnet:::auc(test$pos_neg, pred_test > thresh)
```

```
## [1] 0.693733
```

in this model we notice that decreasing the folds will decrease the AUC which means number of iteration should be higher

```
#model 3

model_dtm <- cv.glmnet(x = dtm_train,
                       y = train$pos_neg,
                       type.measure = "auc",
                       family = "binomial",
                       alpha = 1,
                       thresh = 1e-07,
                       nfolds = 3)

coefs <- coef(model_dtm, s = 'lambda.min')
coefs <- data.frame(name = coefs@Dimnames[[1]][coefs@i + 1], coefficient = coefs@x)

print((nrow(coefs)/ncol(dtm_train)))
```

```
## [1] 0.9129412
```

```
pred_test <- predict(model_dtm, dtm_test, type= 'response')[,1]

thresh <- 0.5
table(test$pos_neg, pred_test > thresh)
```

```
##
##     FALSE  TRUE
##   0  1528  2145
##   1   397 15769
```

```
glmnet:::auc(test$pos_neg, pred_test > thresh)
```

```
## [1] 0.6957255
```

in this one, when we added a nfolds =3 and thresh = 1e-07, we saw some imporvement of 0.002 but tells us we have to increase our model iteration to better results and we should use cross validation for better AUC. our model training model predicted the test dataset of each vocab being either a negative or positive and the best AUC we come up to is approx 70% accurate that means our model is fairly doing good

```
data <- amazon
```

```
data$text_clean <- text

skipgrams <- unnest_tokens(data, ngram, text_clean, token = "ngrams", n = 9)
skipgrams$ngramID <- 1:nrow(skipgrams)
skipgrams$skipgramID <- paste(skipgrams$X_unit_id, skipgrams$ngramID, sep = '_')


head(skipgrams[, c('ngramID', 'ngram', 'skipgramID')])
```

```
##   ngramID
## 1       1
## 2       2
## 3       3
## 4       4
## 5       5
## 6       6
##                                                                          ngram
## 1                 one thing book seem obvious original think however clarity
## 2              thing book seem obvious original think however clarity author
## 3            book seem obvious original think however clarity author explain
## 4      seem obvious original think however clarity author explain innovation
## 5    obvious original think however clarity author explain innovation happen
## 6 original think however clarity author explain innovation happen remarkable
##   skipgramID
## 1         _1
## 2         _2
## 3         _3
## 4         _4
## 5         _5
## 6         _6
```

```
skipgrams <- unnest_tokens(skipgrams, word, ngram)

skipgrams[skipgrams$ngramID == 1, c('skipgramID', 'word')]
```

```
##   skipgramID     word
## 1         _1      one
## 2         _1    thing
## 3         _1     book
## 4         _1     seem
## 5         _1  obvious
## 6         _1 original
## 7         _1    think
## 8         _1  however
## 9         _1  clarity
```

```
library(widyr)
skipgram_probs <- pairwise_count(skipgrams, word, skipgramID, diag = T, sort = T)
```

```
## Warning: `distinct_()` was deprecated in dplyr 0.7.0.
## Please use `distinct()` instead.
## See vignette('programming') for more help
```

```
skipgram_probs$p <- skipgram_probs$n/sum(skipgram_probs$n)

unigram_probs <- unnest_tokens(data, word, text_clean)
unigram_probs <- count(unigram_probs, word, sort = T)
unigram_probs$p <- unigram_probs$n/sum(unigram_probs$n)


lbound <- 20
normed_probs <- skipgram_probs[skipgram_probs$n > lbound,]

colnames(normed_probs) <- c('word1', 'word2', 'n', 'p_all')
normed_probs <- merge(normed_probs, unigram_probs[, c('word', 'p')], by.x = 'word2', by.y = 'word', all.x = T)
normed_probs <- merge(normed_probs, unigram_probs[, c('word', 'p')], by.x = 'word1', by.y = 'word', all.x = T)

head(normed_probs)
```

```
##   word1       word2   n        p_all          p.x          p.y
## 1    10          10 106 4.727658e-07 4.286862e-06 4.286862e-06
## 2    10        cost  28 1.248815e-07 1.449912e-03 4.286862e-06
## 3    10       price  35 1.561019e-07 5.790835e-03 4.286862e-06
## 4     2 originally  25 1.115014e-07 1.381322e-04 9.400135e-04
## 5     2        idea  23 1.025813e-07 3.798636e-04 9.400135e-04
## 6     2     concern  31 1.382617e-07 2.312524e-04 9.400135e-04
```

```
# p_all = probability of seeing a given pair of words in the same window across ALL pairs
# p.x and p.y = probability of seeing given word across all words
normed_probs$p_combined <- normed_probs$p_all/normed_probs$p.x/normed_probs$p.y


normed_probs <- normed_probs[order(normed_probs$p_combined, decreasing = T),]
brands <- c("dryer", "washerdryer","washer","refrigerator", "dishwasher", "stove")
appliances <- normed_probs[normed_probs$word1 %in% brands,]
appliances <- appliances[order(appliances$word1, appliances$word2,-appliances$p_combined),]
head(appliances)
```

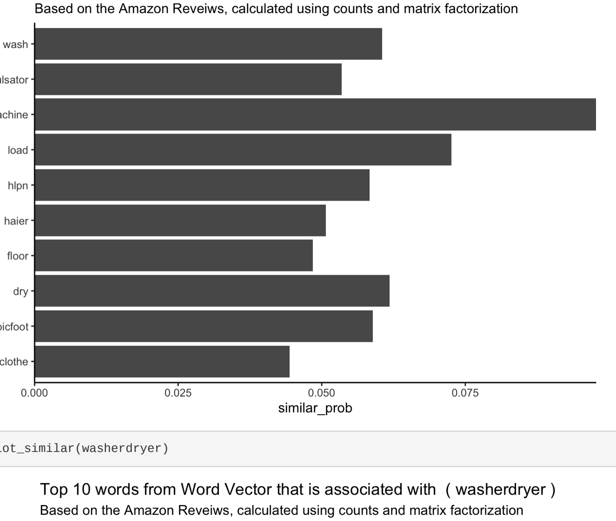
```
##             word1      word2   n        p_all          p.x         p.y
## 236885 dishwasher          2 321 1.431678e-06 9.400135e-04 0.002320621
## 237357 dishwasher          3 168 7.492892e-07 2.088654e-04 0.002320621
## 236611 dishwasher    ability  61 2.720634e-07 7.192401e-05 0.002320621
## 236575 dishwasher       able 569 2.537771e-06 1.141258e-03 0.002320621
## 237904 dishwasher   absolute  25 1.115014e-07 3.548569e-05 0.002320621
## 237255 dishwasher absolutely 153 6.823884e-07 3.119883e-04 0.002320621
##        p_combined
## 236885  0.6563069
## 237357  1.5458905
## 236611  1.6300162
## 236575  0.9582184
## 237904  1.3540128
## 237255  0.9425169
```

```
normed_probs$pmi <- log(normed_probs$p_combined)

pmi_matrix <- cast_sparse(normed_probs, word1, word2, pmi)

library(irlba)
pmi_svd <- irlba(pmi_matrix, 256, maxit = 1e3)
word_vectors <- pmi_svd$u

rownames(word_vectors) <- rownames(pmi_matrix)
```

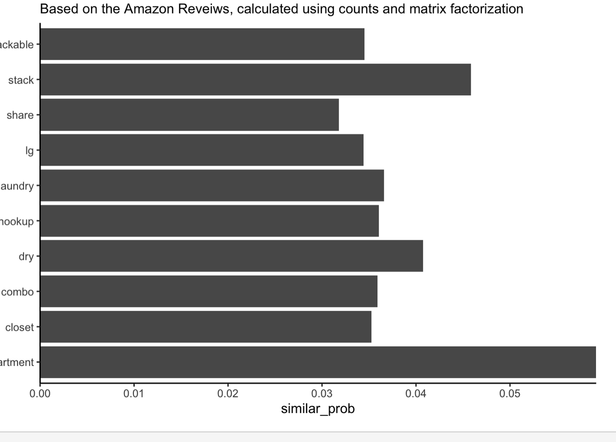
Matching Words

```
library("tibble")
matching_word <- function(word_vectors, selected_vector) {
  similarities <- word_vectors %*% selected_vector %>%
    as.data.frame() %>%
    rename(similar_prob = V1) %>%
    arrange(-similar_prob)


  similarities %>%
    mutate(word_similar = rownames(similarities)) %>%
    select(word_similar, similar_prob)
}




dryer <- matching_word(word_vectors, word_vectors["dryer",])
rownames(dryer) <- NULL


washerdryer <- matching_word(word_vectors, word_vectors["washerdryer",])
rownames(washerdryer) <- NULL


washer <- matching_word(word_vectors, word_vectors["washer",])
rownames(washer) <- NULL


refrigerator <- matching_word(word_vectors, word_vectors["refrigerator",])
rownames(refrigerator) <- NULL


dishwasher <- matching_word(word_vectors, word_vectors["dishwasher",])
rownames(dishwasher) <- NULL


stove <- matching_word(word_vectors, word_vectors["stove",])
rownames(stove) <- NULL
```

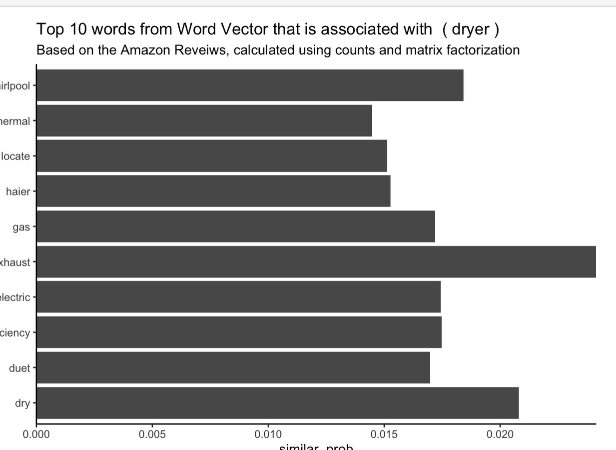
```
plot_similar <- function(df, title){
  string_title <- deparse(substitute(df))
  ggplot(df[2:11,], aes(word_similar, similar_prob)) + geom_bar(stat="identity") + coord_flip() + theme_classic()
+ scale_y_continuous(expand = c(0,0)) +
    labs(x = NULL, title = paste("Top 10 words from Word Vector that is associated with ","(",(string_title),")"
),
         subtitle = "Based on the Amazon Reveiws, calculated using counts and matrix factorization")
}



plot_similar(washer)
```



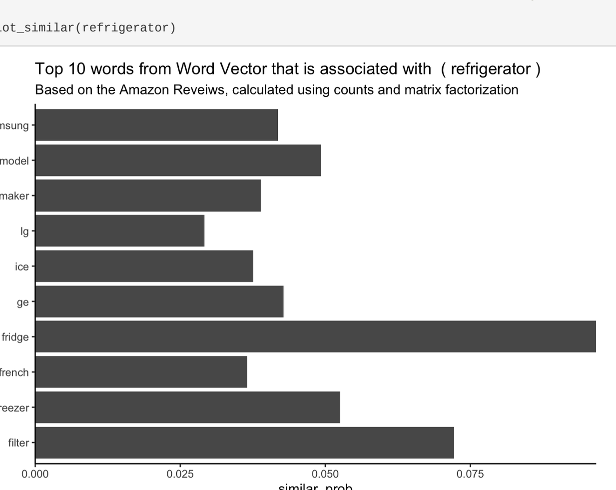

```
plot_similar(washerdryer)
```



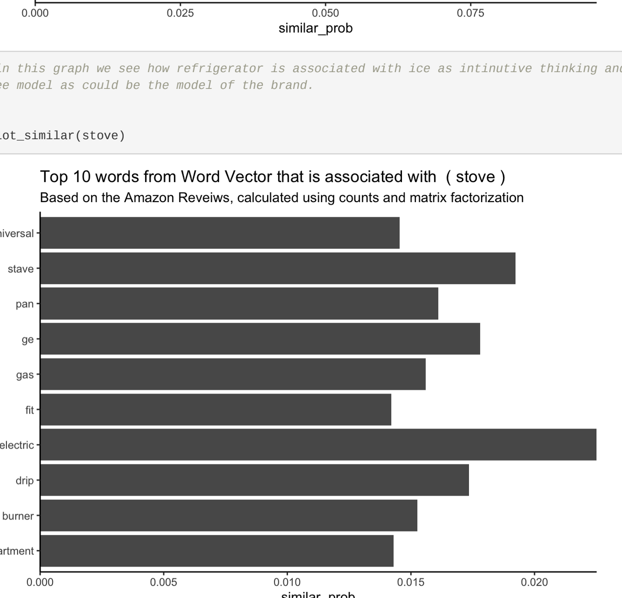

```
plot_similar(dryer)
```



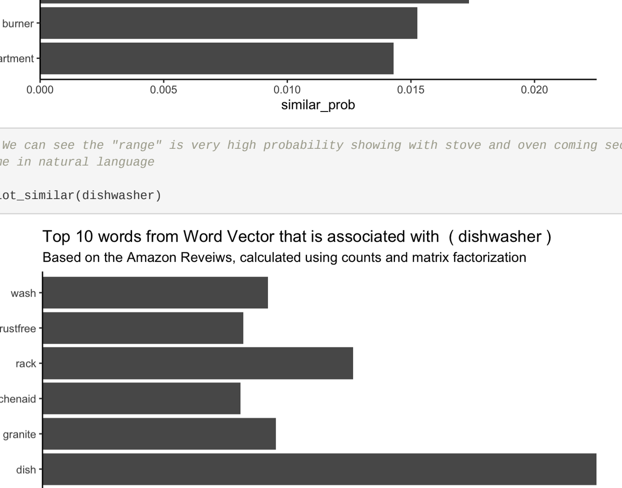

```
# We can see the word "Machine" is the most associated with washer then "dryer" and "washing".

plot_similar(refrigerator)
```




```
#in this graph we see how refrigerator is associated with ice as intinutive thinking and samsung as brand, and we
see model as could be the model of the brand.


plot_similar(stove)
```




```
# We can see the "range" is very high probability showing with stove and oven coming second since they mean the s
ame in natural language

plot_similar(dishwasher)
```




```
#The advantage of the word embedding is giving you a intuitive relationships between words.
#As going with all the appliances words, all top 10 similar word are making sense. I noticed that there are some
 off words that can be off the relationship. Also, usually the results would give you a synonym of the same word
 rather than the words that come with it. I would say that word embeding take a very high performance when factor
ing and processing large data which needs more instnaces and VM that can handle the computation
```