Vector Based Data Mining

Connor Segneri

9/24/2017

Project Design and Assumptions

While many modern data mining tools abstract away the details of converting data to and from vector space, having a first-person experience with such conversions often helps make more effective use of various algorithms and approaches. This assignment provides a hands-on feel for vector space representation and application. Formally, a vector space is defined by a set of linearly independent basis vectors. The basis vectors correspond to the dimensions or directions of the vector space.

Prepatory Work

The documents that will be used for this assignment are the seven Harry Potter novels written by J.K. Rowling. These documents will be the corpus for this assignment.

Below is how to create a frequency text representation of the corpus.

First, the data is read in as strings. These strings are turned into vectors by each new line character and converted into data frames. Each book is then bound together to create one massive data frame that contains each line in the Harry Potter series and the book it came from.

```
library(readr)
library(dplyr)
library(tidytext)
library(ggplot2)
# read in the text file as a string
book1 = read_file("HarryPotterBooks/1Philosopher'sStone.txt")
book2 = read_file("HarryPotterBooks/2ChamberOfSecrets.txt")
book3 = read_file("HarryPotterBooks/3PrisonerOfAzkaban.txt")
book4 = read_file("HarryPotterBooks/4GobletOfFire.txt")
book5 = read_file("HarryPotterBooks/5OrderOfThePhoenix.txt")
book6 = read_file("HarryPotterBooks/6HalfBloodPrince.txt")
book7 = read_file("HarryPotterBooks/7DeathlyHallows.txt")
# split the string by new lines
book1 = unlist(strsplit(book1, split="\n"))
book2 = unlist(strsplit(book2, split="\n"))
book3 = unlist(strsplit(book3, split="\n"))
book4 = unlist(strsplit(book4, split="\n"))
book5 = unlist(strsplit(book5, split="\n"))
book6 = unlist(strsplit(book6, split="\n"))
book7 = unlist(strsplit(book7, split="\n"))
# create a data frame from the new line vector created above
book1 = data_frame(text = book1, book = rep(1, length(book1)))
book2 = data_frame(text = book2, book = rep(2, length(book2)))
book3 = data_frame(text = book3, book = rep(3, length(book3)))
book4 = data_frame(text = book4, book = rep(4, length(book4)))
book5 = data_frame(text = book5, book = rep(5, length(book5)))
book6 = data_frame(text = book6, book = rep(6, length(book6)))
book7 = data_frame(text = book7, book = rep(7, length(book7)))
# collect all of the books into a single data frame
series = rbind(book1, book2, book3, book4, book5, book6, book7)
```

Next, the `tidy` package is used to convert the data frame created above into another data frame that counts every unique word used in each book. The stop words are also removed from the text. Stop words are words like “if”, “on”, and “for”. They hold no real analytical purpose, so they are removed. Custom stop words are also removed from the text. This list contains words like “page”, which appears on every page in each book, and the titles of the books themselves.

```
data(stop_words)
custom_stop_words = c("rowling", "j.k", "page", "philosophers", "stone", "chamber", "sec
rets",
                      "prisoner", "azkaban", "goblet", "fire", "order", "phoenix", "hal
f", "blood", "prince",
                      "deathly", "hallows")
custom_stop_words = data.frame(word = custom_stop_words, lexicon = rep('SMART',length(cu
stom_stop_words)))
stop_words = rbind(stop_words, custom_stop_words)
seriesWords = series %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  count(book, word, sort = TRUE) %>%
  ungroup()

seriesWords$book = as.factor(seriesWords$book)
seriesWords
```

```
## # A tibble: 66,292 x 3
##      book     word     n
##    <fctr>    <chr> <int>
##  1      5    harry  3828
##  2      7    harry  3749
##  3      4    harry  3742
##  4      6    harry  3336
##  5      3    harry  2355
##  6      2    harry  1892
##  7      1    harry  1559
##  8      5   potter  1380
##  9      5 hermione  1242
## 10      5      ron  1191
## # ... with 66,282 more rows
```

Technical Prepatory Work

1. *Describe the V-dimensional space.*

The data frame that was created in the previous section does not reflect a data frame that contains a binary text representation of a corpus. In a binary representation, each row would represent a book in the corpus, and each column would represent a unique word that appears in the corpus. A binary 0 or 1 exists where each row/column meet, and this represents if that specific word (column) exists in that specific book (row). So, to answer the question above, the V-dimensional space is the amount of **unique** words that exist in the entire corpus.

The data frame that was actually created in the previous section is a frequency text representation of the corpus. This means that the frequency in which each word appears in **each** book is kept, instead of just having a binary value that represents whether the word exists or not in that book. This data frame was created instead of the

binary one because the computations requested in this assignment are only possible with a frequency representation, not a binary one. The rest of the questions in this report will be answered assuming a frequency representation was requested.

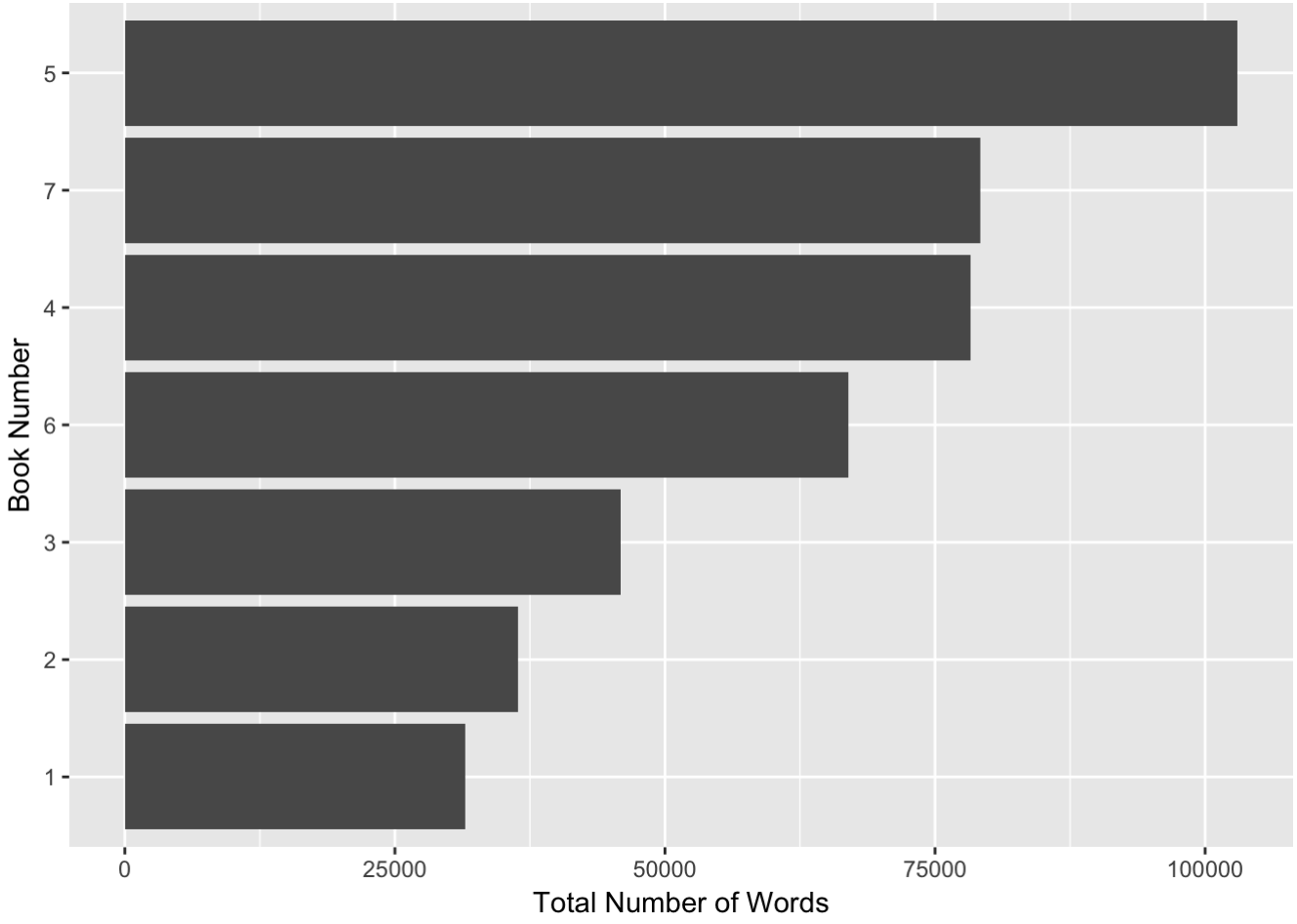
2. *A visual diagram of a V-D vector using words in your corpus*

This counts the total amount of words for each book.

```
seriesTotalWords = seriesWords %>%
  group_by(book) %>%
  summarize(total = sum(n))
```

This plots the total amount of words per book.

```
seriesTotalWords %>%
  mutate(book = reorder(book, total)) %>%
  ggplot(aes(book, total)) +
  geom_col() +
  xlab("Book Number") +
  ylab("Total Number of Words") +
  coord_flip()
```



3. *What span(s) of text does this vector represent?*

The span of a single vector is the set of all vectors parallel to that vector. Because of this, the span does not have any significant meaning.

4. *Perform a query (i.e., compute a vector in V-dimensional space, where V is the number of terms in the vocabulary) and represent it in a diagram*

This question is redundant, as queries on the document vectors will be demonstrated and plotted in following sections.

Computations

Cosine similarity measures the cosine of the angle between two vectors. This idea can be used to measure the similarity between two documents.

$$similarity = cos(\theta) = \frac{A \cdot B}{||A||_2 ||B||_2}$$

Cosine similarity and inner product are equivalent, so only cosine similarity is going to be computed in this section.

The following code uses the `exploratory` library to find the cosine similarity between each book in the corpus against every other book. It is important to not that the similarity is computed using a frequency data frame that contains the amount of times a unique word appear in each book.

```
library(exploratory)
simBooks = seriesWords %>%
  do_cosine_sim.kv(book, word, n) %>%
  arrange(desc(value))
```

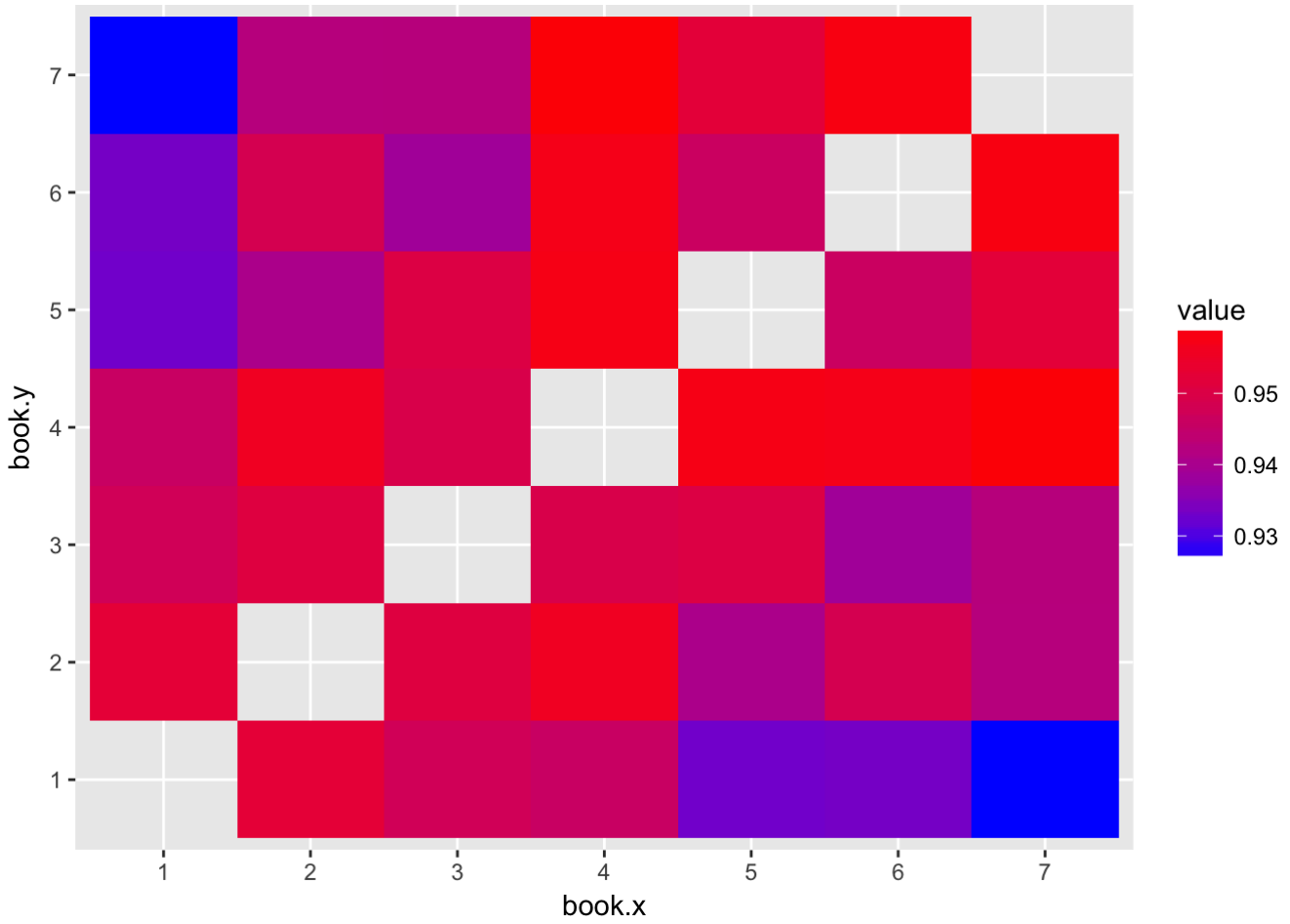
This is the data frame that is returned. The first row indicates that the similarity between book 4 and book 7 is about 0.96.

```
simBooks
```

```
## # A tibble: 42 x 3
##    book.x book.y     value
##     <int>  <int>     <dbl>
##  1      4      7 0.9592726
##  2      7      4 0.9592726
##  3      6      7 0.9583229
##  4      7      6 0.9583229
##  5      4      5 0.9578115
##  6      5      4 0.9578115
##  7      4      6 0.9573993
##  8      6      4 0.9573993
##  9      2      4 0.9558869
## 10      4      2 0.9558869
## # ... with 32 more rows
```

A heat map is useful for visualizing the similarities between every book in the corpus. Red signifies the highest similarities, and blue signifies the lowest.

```
simBooks$book.x = as.factor(simBooks$book.x)
simBooks$book.y = as.factor(simBooks$book.y)
freqSimPlot = ggplot(simBooks, aes(book.x, book.y, fill=value)) +
  scale_fill_gradient(low = "blue", high = "red") + geom_tile()
freqSimPlot
```



In order to rank the books by similarity, the average similarities between each book and the rest of the corpus need to be evaluated.

This code shows all of the similarity scores for book 1.

```
simBooks[simBooks$book.x==1,]
```

```
## # A tibble: 6 x 3
##   book.x book.y     value
##   <fctr> <fctr>     <dbl>
## 1      1      2 0.9525048
## 2      1      3 0.9474675
## 3      1      4 0.9458460
## 4      1      6 0.9332252
## 5      1      5 0.9327116
## 6      1      7 0.9269509
```

Using a list for each book like the one seen above, the average similarities between each book and the rest of the corpus can be computed. The below code does this, and sorts the books by their mean similarity values.

```
meanSimBooks = data.frame(book = 1:7, meanValue = rep(NA, 7))
meanSimBooks$meanValue[1] = mean(simBooks[simBooks$book.x==1,]$value)
meanSimBooks$meanValue[2] = mean(simBooks[simBooks$book.x==2,]$value)
meanSimBooks$meanValue[3] = mean(simBooks[simBooks$book.x==3,]$value)
meanSimBooks$meanValue[4] = mean(simBooks[simBooks$book.x==4,]$value)
meanSimBooks$meanValue[5] = mean(simBooks[simBooks$book.x==5,]$value)
meanSimBooks$meanValue[6] = mean(simBooks[simBooks$book.x==6,]$value)
meanSimBooks$meanValue[7] = mean(simBooks[simBooks$book.x==7,]$value)

meanSimBooks = meanSimBooks %>%
  arrange(desc(meanValue))
meanSimBooks
```

```
##   book meanValue
## 1    4 0.9542922
## 2    2 0.9484345
## 3    6 0.9470682
## 4    7 0.9468623
## 5    5 0.9466196
## 6    3 0.9465456
## 7    1 0.9397843
```

Data Analysis

1. *Choose a search term that in your opinion would be useful to study. For each document in your corpus, compute the inner-product and cosine similarity score for the query corresponding to your chosen term. Compare the results of the two methods and explain differences (if any).*

The search term is going to be “horcrux”. A separate document called “term” is added to the frequency data frame. The row that is added containing the search term is shown below.

```
searchTerm = data.frame(book = "term", word = "horcrux", n = 1)
simTerm = rbind(seriesWords, searchTerm)
searchTerm
```

```
##   book    word n
## 1 term horcrux 1
```

Now, the cosine similarity of between term and the rest of the books in the corpus can be evaluated.

```
simTerm = simTerm %>%
  do_cosine_sim.kv(book, word, n) %>%
  arrange(desc(value))

simTerm[simTerm$book.x=="term",]
```

```
## # A tibble: 7 x 3
##   book.x book.y      value
##    <chr>  <chr>      <dbl>
## 1   term      7 0.01985236
## 2   term      6 0.01076737
## 3   term      1 0.00000000
## 4   term      2 0.00000000
## 5   term      3 0.00000000
## 6   term      4 0.00000000
## 7   term      5 0.00000000
```

As can be seen above, only book 6 and book 7 have any similarity with the term “horcrux”. This makes sense, as the term is first introduced in the 6th book, and the 7th book uses the term the most.

2. *Term-frequency (TF): Find the top 30 most frequent words (i.e., the most important words) in your corpus and present a table with the words and the frequency*

Term frequency is simply the raw count of a term in a document, taking into account the amount of words total in that document.

$$TF = \frac{t}{d}$$

Where t is the the total count for a specific term in the document, and d is the total amount of terms in that document.

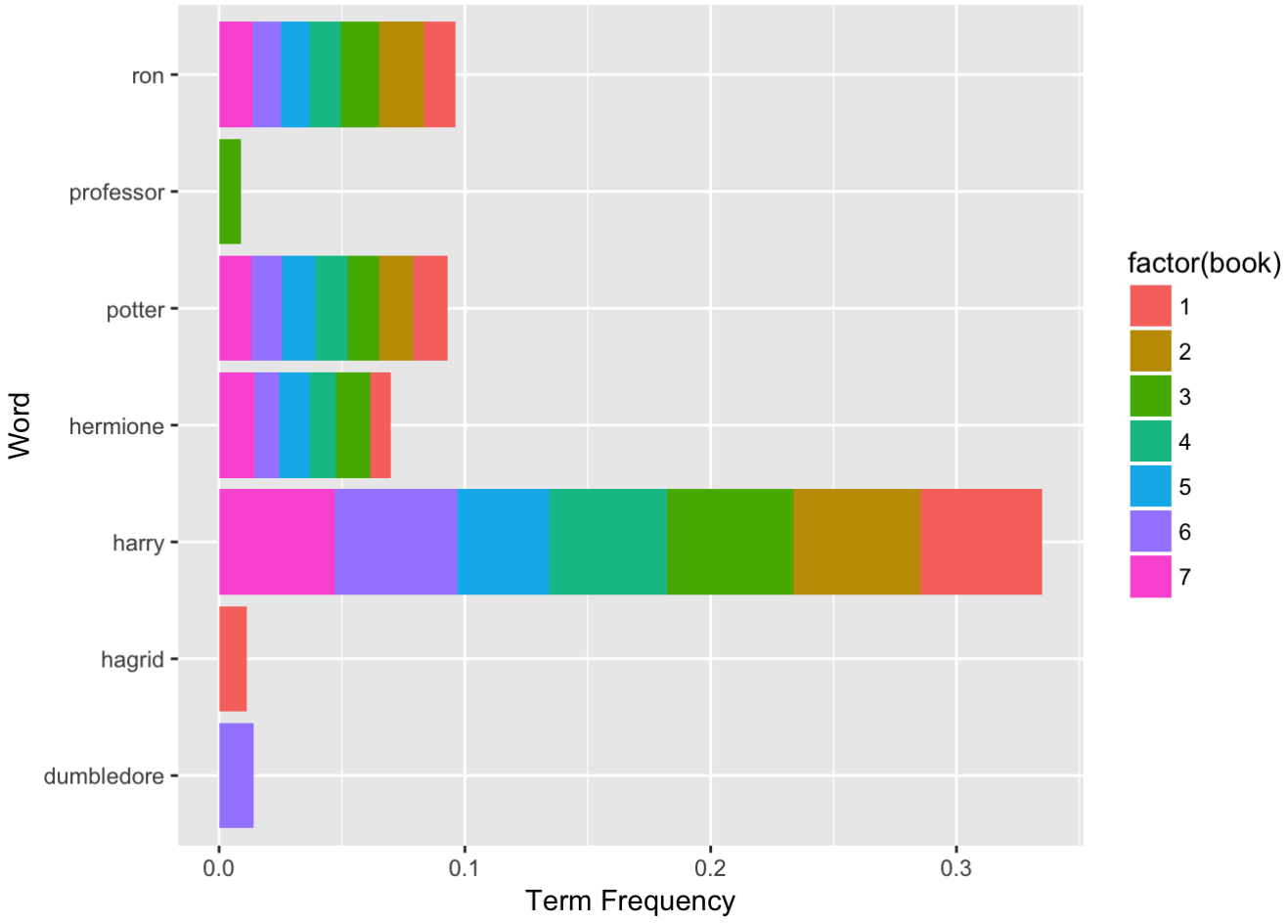
The term frequency, the inverse document frequency, and the TF.IDF is computed below.

```
seriesWords <- seriesWords %>%
  bind_tf_idf(word, book, n)

seriesWords
```

```
## # A tibble: 66,292 x 6
##      book     word     n         tf   idf tf_idf
##    <fctr>    <chr> <int>      <dbl> <dbl>  <dbl>
##  1      5    harry  3828 0.03715567     0      0
##  2      7    harry  3749 0.04732212     0      0
##  3      4    harry  3742 0.04781375     0      0
##  4      6    harry  3336 0.04978659     0      0
##  5      3    harry  2355 0.05135530     0      0
##  6      2    harry  1892 0.05202233     0      0
##  7      1    harry  1559 0.04951250     0      0
##  8      5   potter  1380 0.01339468     0      0
##  9      5 hermione  1242 0.01205521     0      0
## 10      5      ron  1191 0.01156019     0      0
## # ... with 66,282 more rows
```

The code below plots the thirty most frequent words in the corpus. Any words that were found in multiple books were only plotted once, but their frequency in each book can be seen in the coloration of that term's frequency bar.

```
seriesWords = seriesWords %>%
  arrange(desc(tf))

seriesWords[1:30,] %>%
  ggplot(aes(word, tf, fill = factor(book))) +
  geom_col() +
  labs(x = "Word", y = "Term Frequency") +
  coord_flip()
```



Next a table is printed with the thirty most common words sorted by term frequency.

```
print(as_tibble(seriesWords), n=30)
```

```
## # A tibble: 66,292 x 6
##      book       word     n          tf   idf tf_idf
##    <fctr>      <chr> <int>       <dbl> <dbl>  <dbl>
##  1      2      harry  1892 0.052022327     0      0
##  2      3      harry  2355 0.051355300     0      0
##  3      6      harry  3336 0.049786586     0      0
##  4      1      harry  1559 0.049512497     0      0
##  5      4      harry  3742 0.047813754     0      0
##  6      7      harry  3749 0.047322116     0      0
##  7      5      harry  3828 0.037155669     0      0
##  8      2        ron   657 0.018064835     0      0
##  9      3        ron   722 0.015744597     0      0
## 10      7   hermione  1157 0.014604345     0      0
## 11      3   hermione   650 0.014174499     0      0
## 12      1     potter   441 0.014005780     0      0
## 13      6 dumbledore   937 0.013983822     0      0
## 14      2     potter   503 0.013830460     0      0
## 15      7        ron  1068 0.013480934     0      0
## 16      5     potter  1380 0.013394677     0      0
## 17      7     potter  1061 0.013392575     0      0
## 18      4     potter  1030 0.013160921     0      0
## 19      1        ron   410 0.013021247     0      0
## 20      3     potter   584 0.012735242     0      0
## 21      4        ron   979 0.012509264     0      0
## 22      6     potter   824 0.012297406     0      0
## 23      5   hermione  1242 0.012055209     0      0
## 24      6        ron   793 0.011834761     0      0
## 25      5        ron  1191 0.011560189     0      0
## 26      1     hagrid   351 0.011147458     0      0
## 27      4   hermione   843 0.010771511     0      0
## 28      6   hermione   667 0.009954332     0      0
## 29      3  professor   416 0.009071679     0      0
## 30      1   hermione   262 0.008320894     0      0
## # ... with 6.626e+04 more rows
```

3. *Inverse Document Frequency (IDF): Create a table ranking the top 30 terms in your corpus based on their IDF values.*

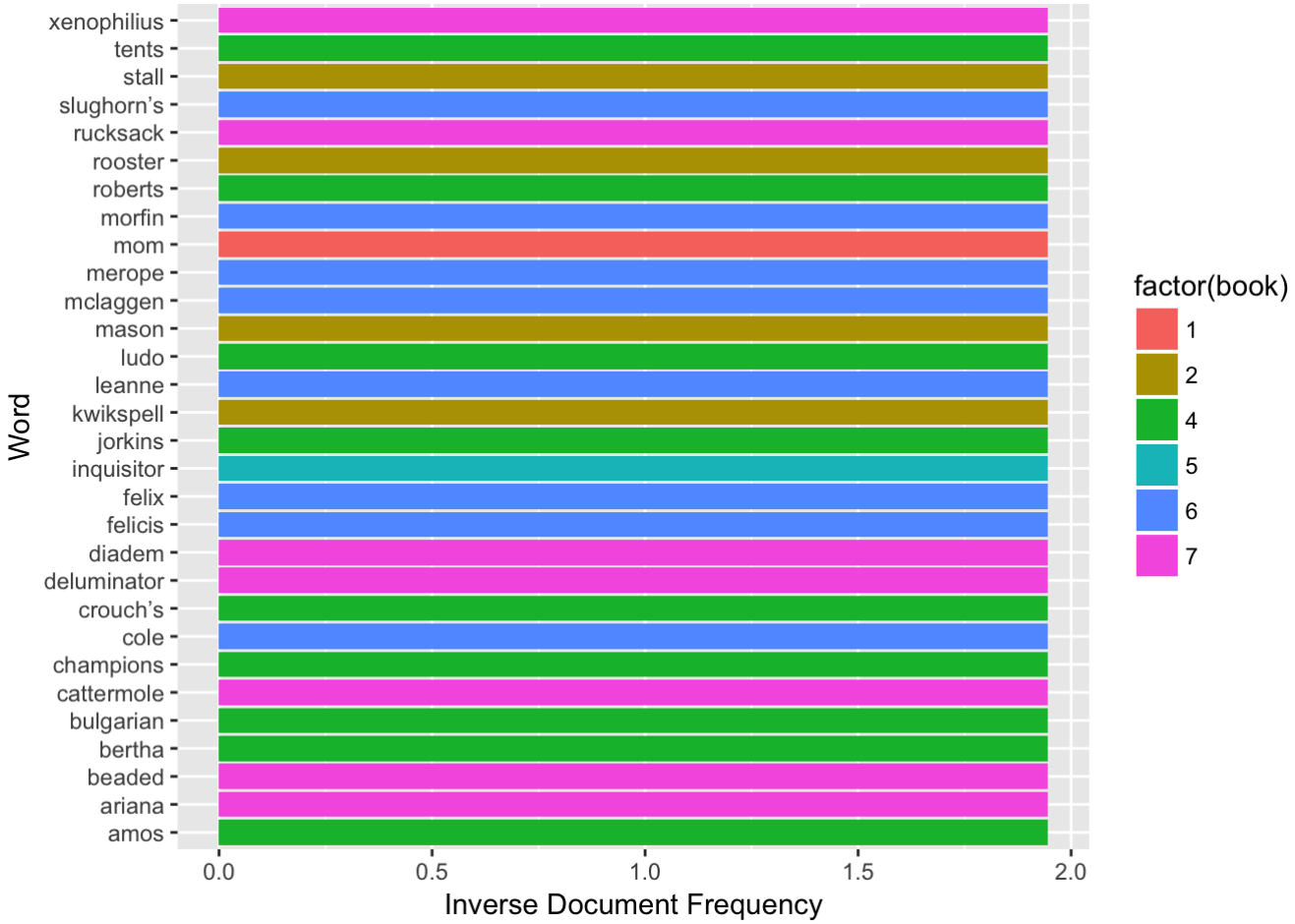
Inverse document frequency is the measure of how much information a word provides when compared to the corpus as a whole. This determines if a word is common or rare across every document.

$$IDF = log\frac{N}{n_t}$$

Where N is the total number of documents in the corpus, and n_t is the total number of documents where the term t appears.

The code below plots the top thirty terms in the corpus based on inverse document frequency. A lot of the terms had very similar IDF values, so the chart is fairly redundant.

```
seriesWords = seriesWords %>%
  arrange(desc(idf))

seriesWords[1:30,] %>%
  ggplot(aes(word, idf, fill = factor(book))) +
  geom_col() +
  labs(x = "Word", y = "Inverse Document Frequency") +
  coord_flip()
```



Next a table is printed with the thirty most common words sorted by inverse document frequency.

```
print(as_tibble(seriesWords), n=30)
```

```
## # A tibble: 66,292 x 6
##      book        word     n           tf     idf       tf_idf
##    <fctr>       <chr> <int>        <dbl>   <dbl>        <dbl>
##  1      4   champions    86 0.0010988730 1.94591 0.0021383082
##  2      7 xenophilius    86 0.0010855433 1.94591 0.0021123698
##  3      6    mclaggen    65 0.0009700624 1.94591 0.0018876542
##  4      4        ludo    71 0.0009072091 1.94591 0.0017653474
##  5      6      morfin    59 0.0008805182 1.94591 0.0017134092
##  6      6  slughorn's    39 0.0005820374 1.94591 0.0011325925
##  7      7      ariana    44 0.0005553943 1.94591 0.0010807473
##  8      7      diadem    44 0.0005553943 1.94591 0.0010807473
##  9      7  cattermole    43 0.0005427717 1.94591 0.0010561849
## 10      1         mom    17 0.0005399054 1.94591 0.0010506073
## 11      6       felix    33 0.0004924932 1.94591 0.0009583475
## 12      4      bertha    38 0.0004855485 1.94591 0.0009448338
## 13      6      merope    32 0.0004775692 1.94591 0.0009293067
## 14      4    crouch's    34 0.0004344382 1.94591 0.0008453776
## 15      6     felicis    27 0.0004029490 1.94591 0.0007841025
## 16      4        amos    28 0.0003577726 1.94591 0.0006961934
## 17      4   bulgarian    27 0.0003449950 1.94591 0.0006713293
## 18      2   kwikspell    12 0.0003299513 1.94591 0.0006420556
## 19      2       stall    12 0.0003299513 1.94591 0.0006420556
## 20      7      beaded    26 0.0003281875 1.94591 0.0006386234
## 21      4     jorkins    25 0.0003194398 1.94591 0.0006216012
## 22      7 deluminator    25 0.0003155649 1.94591 0.0006140610
## 23      6      leanne    20 0.0002984807 1.94591 0.0005808167
## 24      4       tents    23 0.0002938846 1.94591 0.0005718731
## 25      6        cole    19 0.0002835567 1.94591 0.0005517759
## 26      4     roberts    22 0.0002811071 1.94591 0.0005470091
## 27      7    rucksack    22 0.0002776971 1.94591 0.0005403737
## 28      2       mason    10 0.0002749594 1.94591 0.0005350464
## 29      2     rooster    10 0.0002749594 1.94591 0.0005350464
## 30      5  inquisitor    28 0.0002717761 1.94591 0.0005288518
## # ... with 6.626e+04 more rows
```

4. *TF.IDF: Compute the product tft x idft and provide a table of the top 30 terms based on their TF.IDF ranking. (Recall that idft = log(N/dft) where dft is the number of documents in which the term t appears.)*

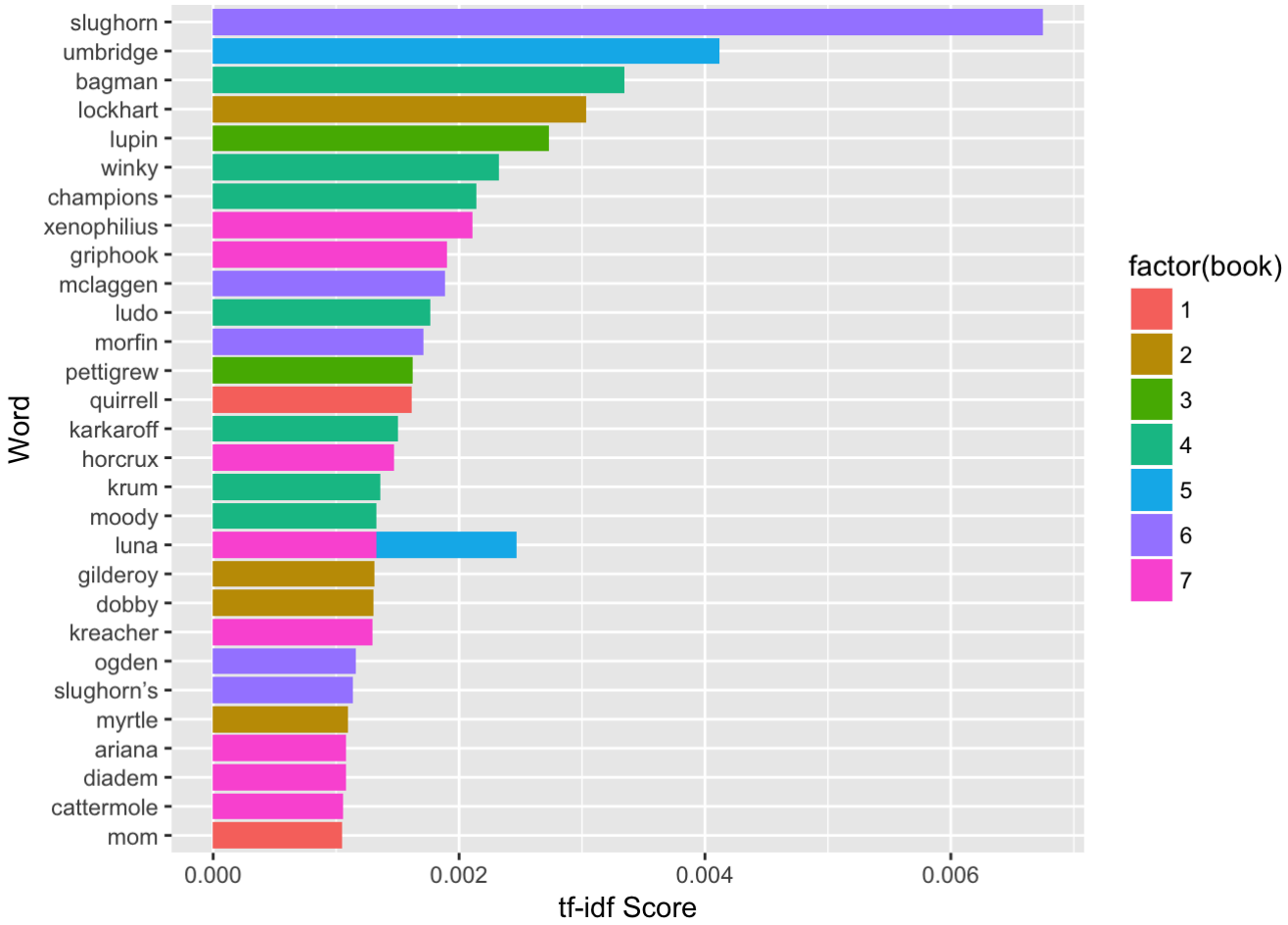
Term frequency-inverse document frequency is the term frequency weighted by the inverse document frequency. The terms that appear heavily across the entire corpus are filtered out so that only frequently unique terms in each book are visible.

$$TF.IDF = TF \cdot IDF$$

Where TF is term frequency and IDF is inverse document frequency.

The terms across the corpus are sorted by the TF.IDF and are plotted below.

```
seriesWords = seriesWords %>%
  arrange(desc(tf_idf))

plot_potter = seriesWords %>%
  mutate(word = factor(word, levels = rev(unique(word))))

plot_potter %>%
  top_n(30) %>%
  ggplot(aes(word, tf_idf, fill = factor(book))) +
  geom_col() +
  labs(x = "Word", y = "tf-idf Score") +
  coord_flip()
```



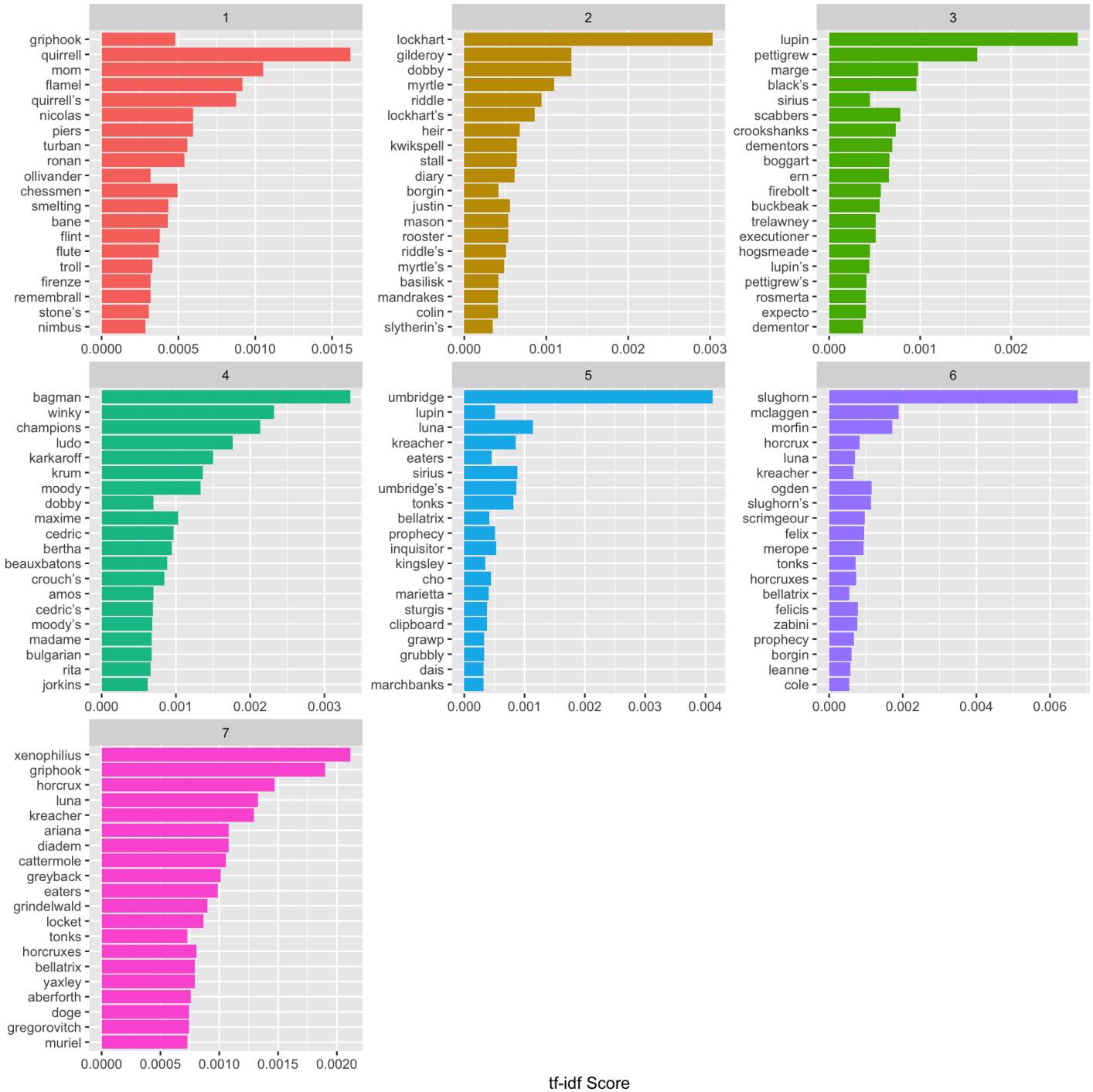
Next a table is printed with the thirty most common words sorted by term frequency-inverse document frequency.

```
print(as_tibble(seriesWords), n=30)
```

```
## # A tibble: 66,292 x 6
##      book        word     n           tf       idf      tf_idf
##    <fctr>       <chr> <int>        <dbl>     <dbl>       <dbl>
##  1      6    slughorn   361 0.0053875772 1.2527630 0.006749357
##  2      5    umbridge   501 0.0048628502 0.8472979 0.004120283
##  3      4      bagman   209 0.0026705170 1.2527630 0.003345525
##  4      2    lockhart   197 0.0054167010 0.5596158 0.003031271
##  5      3       lupin   372 0.0081121748 0.3364722 0.002729522
##  6      4       winky   145 0.0018527510 1.2527630 0.002321058
##  7      4   champions    86 0.0010988730 1.9459101 0.002138308
##  8      7 xenophilius    86 0.0010855433 1.9459101 0.002112370
##  9      7    griphook   120 0.0015147116 1.2527630 0.001897575
## 10      6    mclaggen    65 0.0009700624 1.9459101 0.001887654
## 11      4        ludo    71 0.0009072091 1.9459101 0.001765347
## 12      6      morfin    59 0.0008805182 1.9459101 0.001713409
## 13      3   pettigrew    88 0.0019190091 0.8472979 0.001625972
## 14      1    quirrell    91 0.0028900816 0.5596158 0.001617335
## 15      4   karkaroff   139 0.0017760855 0.8472979 0.001504873
## 16      7     horcrux    93 0.0011739015 1.2527630 0.001470620
## 17      4        krum   190 0.0024277427 0.5596158 0.001358603
## 18      4       moody   309 0.0039482763 0.3364722 0.001328485
## 19      7        luna   124 0.0015652020 0.8472979 0.001326192
## 20      2    gilderoy    38 0.0010448459 1.2527630 0.001308944
## 21      2       dobby   141 0.0038769282 0.3364722 0.001304479
## 22      7    kreacher   121 0.0015273342 0.8472979 0.001294107
## 23      6       ogden    62 0.0009252903 1.2527630 0.001159169
## 24      5        luna   139 0.0013491740 0.8472979 0.001143152
## 25      6  slughorn's    39 0.0005820374 1.9459101 0.001132593
## 26      2      myrtle    47 0.0012923094 0.8472979 0.001094971
## 27      7      ariana    44 0.0005553943 1.9459101 0.001080747
## 28      7      diadem    44 0.0005553943 1.9459101 0.001080747
## 29      7  cattermole    43 0.0005427717 1.9459101 0.001056185
## 30      1         mom    17 0.0005399054 1.9459101 0.001050607
## # ... with 6.626e+04 more rows
```

The below plot shows the top twenty terms as sorted by TF.IDF for each book in the corpus.

```
plot_potter %>%
  group_by(book) %>%
  top_n(20) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = factor(book))) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf Score") +
  facet_wrap(~book, ncol = 3, scales = "free") +
  coord_flip()
```



5. *What have you learned so far about the subject of your queries, by representing queries as TF.IDF vectors?*

So far it has become apparent that the names of characters who play large roles in unique books are normally the most common words found when the corpus is sorted by TF.IDF. It also seems like this would be the best way to sort the corpus if the main themes from each book need to be determined.

6. *Rank the documents in your corpus based on cosine similarity to the query, also using binary weights. Present the result as a 2D vectors diagram and explain the findings while explaining the importance of linear independence of basis vectors.*

This question is pretty confusing, so I'm assuming that this is asking us to rank the documents in the corpus based off of cosine similarity using the TF.IDF values instead of just raw frequency. I also tried looking it up, and as far as I can tell the phrase "binary weights" is meaningless. Also, a set of basis vectors are linearly independent by definition, so I'm unsure what that has to do with anything.

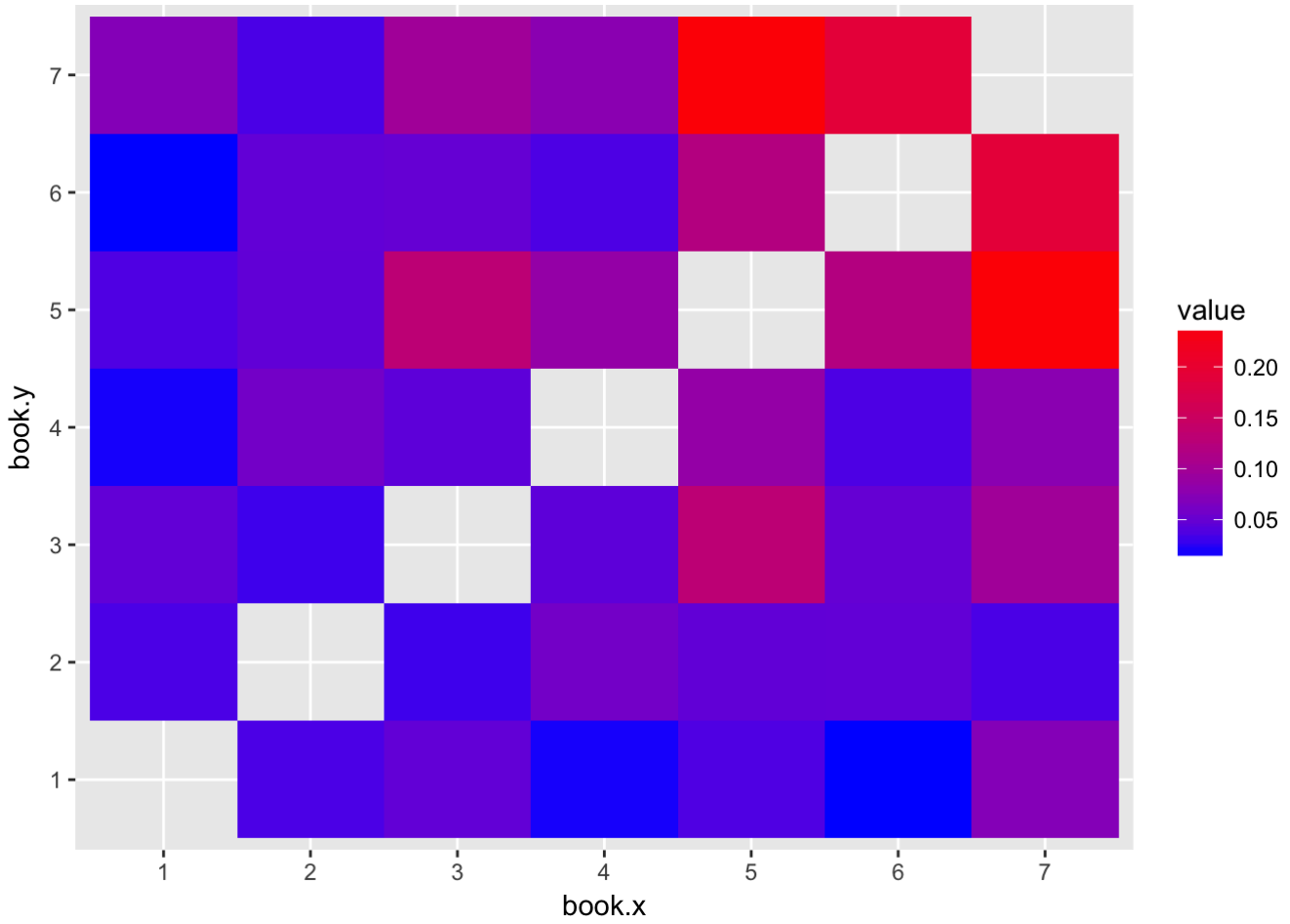
The following code is going to be very similar to the cosine similarity computations carried out in Computations section. The only difference is, again, that the similarities are computed using the TF.IDF values instead of the raw frequency.

```
library(exploratory)
simBooks = seriesWords %>%
  do_cosine_sim.kv(book, word, tf_idf) %>%
  arrange(desc(value))

simBooks
```

```
## # A tibble: 42 x 3
##    book.x book.y      value
##     <int>  <int>      <dbl>
##  1      5      7 0.23905808
##  2      7      5 0.23905808
##  3      6      7 0.19008952
##  4      7      6 0.19008952
##  5      3      5 0.13026786
##  6      5      3 0.13026786
##  7      5      6 0.11994061
##  8      6      5 0.11994061
##  9      3      7 0.09730943
## 10      7      3 0.09730943
## # ... with 32 more rows
```

A heat map is used again to visualize the similarities between every book in the corpus. Red signifies the highest similarities, and blue signifies the lowest.

```
simBooks$book.x = as.factor(simBooks$book.x)
simBooks$book.y = as.factor(simBooks$book.y)
TFIDFSimPlot =  ggplot(simBooks, aes(book.x, book.y, fill=value)) +
  scale_fill_gradient(low = "blue", high = "red") + geom_tile()
TFIDFSimPlot
```



In order to rank the books by similarity, the average similarities between each book and the rest of the corpus need to be evaluated.

This code shows all of the similarity scores for book 1.

```
simBooks[simBooks$book.x==1,]
```

```
## # A tibble: 6 x 3
##   book.x book.y      value
##   <fctr> <fctr>      <dbl>
## 1      1      7 0.07137585
## 2      1      3 0.04766752
## 3      1      5 0.03793448
## 4      1      2 0.03614648
## 5      1      4 0.01988056
## 6      1      6 0.01761241
```

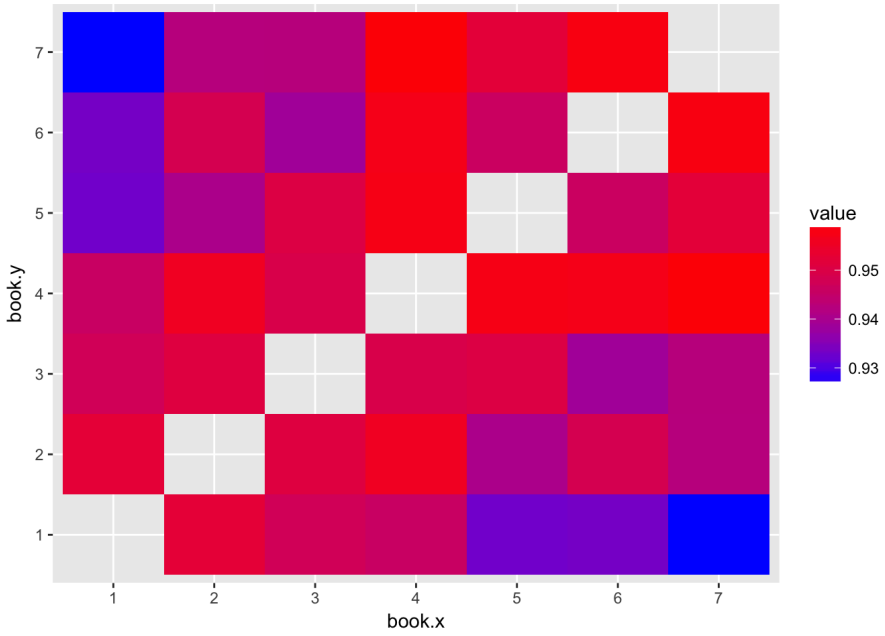
Using a list for each book like the one seen above, the average similarities between each book and the rest of the corpus can be computed. The below code does this, and sorts the books by their mean similarity values.

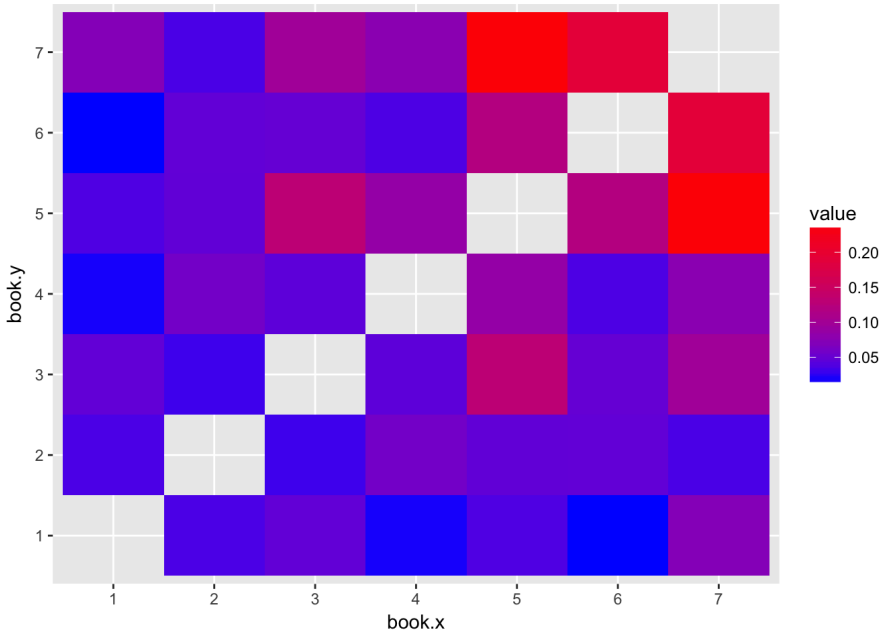
```
meanSimBooksTFIDF = data.frame(book = 1:7, meanValue = rep(NA, 7))
meanSimBooksTFIDF$meanValue[1] = mean(simBooks[simBooks$book.x==1,]$value)
meanSimBooksTFIDF$meanValue[2] = mean(simBooks[simBooks$book.x==2,]$value)
meanSimBooksTFIDF$meanValue[3] = mean(simBooks[simBooks$book.x==3,]$value)
meanSimBooksTFIDF$meanValue[4] = mean(simBooks[simBooks$book.x==4,]$value)
meanSimBooksTFIDF$meanValue[5] = mean(simBooks[simBooks$book.x==5,]$value)
meanSimBooksTFIDF$meanValue[6] = mean(simBooks[simBooks$book.x==6,]$value)
meanSimBooksTFIDF$meanValue[7] = mean(simBooks[simBooks$book.x==7,]$value)

meanSimBooksTFIDF = meanSimBooksTFIDF %>%
  arrange(desc(meanValue))
meanSimBooksTFIDF
```

```
##   book  meanValue
## 1    7 0.11826091
## 2    5 0.11002816
## 3    6 0.07700990
## 4    3 0.06679243
## 5    4 0.05377804
## 6    2 0.04268268
## 7    1 0.03843622
```

To compare the cosine similarity between using raw frequency and the TF.IDF values, the heat maps are plotted side by side below. The similarity rankings are also shown side by side after the maps.

```
freqSimPlot
TFIDFSimPlot
```






```
simComp = data.frame(
  frequency = meanSimBooks$book,
  TFIDF = meanSimBooksTFIDF$book,
  freqSimVal = meanSimBooks$meanValue,
  TFIDFSimVal = meanSimBooksTFIDF$meanValue)
simComp
```

```
##   frequency TFIDF freqSimVal TFIDFSimVal
## 1         4     7  0.9542922  0.11826091
## 2         2     5  0.9484345  0.11002816
## 3         6     6  0.9470682  0.07700990
## 4         7     3  0.9468623  0.06679243
## 5         5     4  0.9466196  0.05377804
## 6         3     2  0.9465456  0.04268268
## 7         1     1  0.9397843  0.03843622
```

From looking at the results above, it seems as though the similarity scores in general tend to severely decrease when TF.IDF scores are used. This makes sense, as the TF.IDF scores do not consider words that appear often across the corpus as a whole. The ranking of the corpus is also different. When using raw frequency to compute cosine similarity, it seems like book 4 was the most similar to all of the other books. When using TF.IDF, it seems like book 7 is the most similar to every other book.

Information Analysis

1. *How would you find documents on the Internet that are similar to your earlier defined query?*

I would find documents like these by performing Google searches. There are R libraries that also have data sets full of books and other documents that I could look into.

2. *How would you find internet ads that are similar to a given search result?*

Not sure what this means, but I suppose if I searched for a term, the ads that show up could be related to that term in some way.

3. *How would you find a query similar to the given query?*

What query? This makes no sense. Also you don't find queries, you make them.

4. *How would you use the technique you developed to automatically assign documents to a category (i.e., topic categorization)?*

I'm assuming this is just in the wrong assignment entirely. Categorical analysis was never used in this project.

5. *If a document is similar to a query, is it likely to be relevant (topical relevance)? Explain.*

Again, what query? Assuming the query is another document vector, and "similar" means a high score in cosine similarity, then yes, it is likely that these two vectors would be relevant to each other.

References

(n.d.). Retrieved September 29, 2017, from http://thejuniverse.org/PUBLIC/LinearAlgebra/LOLA/spans/def.html (http://thejuniverse.org/PUBLIC/LinearAlgebra/LOLA/spans/def.html)

Cosine similarity. (2017, September 29). Retrieved September 29, 2017, from https://en.wikipedia.org/wiki/Cosine_similarity (https://en.wikipedia.org/wiki/Cosine_similarity)

Linear combinations and span. (n.d.). Retrieved September 29, 2017, from https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/linear-combinations/v/linear-combinations-and-span (https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/linear-combinations/v/linear-combinations-and-span)

Nishida, K. (2016, June 24). Demystifying Text Analytics part 3 - Finding Similar Documents with Cosine Similarity in R. Retrieved September 29, 2017, from https://blog.exploratory.io/demystifying-text-analytics-finding-similar-documents-with-cosine-similarity-e7b9e5b8e515 (https://blog.exploratory.io/demystifying-text-analytics-finding-similar-documents-with-cosine-similarity-e7b9e5b8e515)

Robinson, J. S. (2016, August 23). Tidy Text Mining in R. Retrieved September 29, 2017, from http://tidytextmining.com/_book/tidying-and-casting-document-term-matrices.html (http://tidytextmining.com/_book/tidying-and-casting-document-term-matrices.html)

Robinson, J. S. (2017, May 07). Retrieved September 29, 2017, from http://tidytextmining.com/tfidf.html# (http://tidytextmining.com/tfidf.html#)

Tf–idf. (2017, September 21). Retrieved September 29, 2017, from https://en.wikipedia.org/wiki/Tf%E2%80%93idf (https://en.wikipedia.org/wiki/Tf%E2%80%93idf)

Writing Mathematic Fomulars in Markdown. (2017, February 12). Retrieved September 29, 2017, from http://csrgxtu.github.io/2015/03/20/Writing-Mathematic-Fomulars-in-Markdown/ (http://csrgxtu.github.io/2015/03/20/Writing-Mathematic-Fomulars-in-Markdown/)