tf-idf-pres-LAEA

Fiona R Lodge

5/12/2019

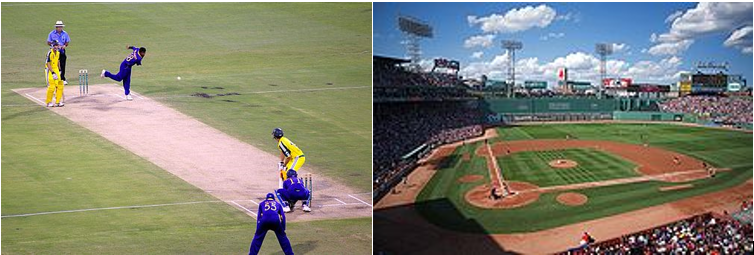
## Introduction and Use Case

During my internship at Securian Financial, I was given a dataset which contained medical conditions (as text) for life insurance applicants. In the exploratory phase, I used the tf-idf statistic to extract medical conditions characteristic to each underwriting (risk) class. As an example, diabetes may be a condition characteristic to an underwriting class associated with a higher risk. A little bit by accident, I also discovered underwriting cases that did not follow the set-upon rules by using this statistic.

More commonly the tf-idf statistic is used for tasks involved in information retrieval, but is also used as a text vectorization tool for predictive modeling.

## The Example Dataset

For the next few examples, I will be using the text generated from the Wikipedia articles on baseball and cricket. The tf-idf should find words that are characteristic to each sport.



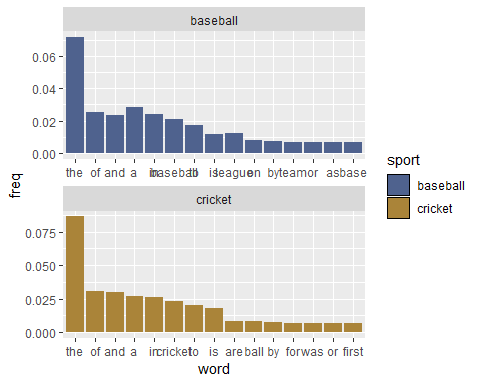
The data was tokenized by single words using the unnest\_tokens tool from the tidytext package. A sample tokenization of the sentence ‘Baseball is a bat-and-ball game’ would be ‘Baseball’, ‘is’, ‘a’, ‘bat-and-ball’, ‘game’. Data cleaning steps performed included removing digits and 's. The data was then combined into one dataset.

# tokenize  
tokenizer.func <- function(dat, sp = 'baseball'){  
 tmp <-   
 dat %>%  
 unnest\_tokens(output = word,   
 input = !!sym(colnames(dat)),   
 token = 'words') %>%  
 mutate(sport = sp) %>%  
 select(sport, word) %>%  
 filter(!grepl('\\d', word)) %>% # remove digits  
 mutate(word = str\_replace\_all(word, '\'s', "")) #Stemming to remove 's  
 return(tmp)  
}  
baseball.tokenized <- tokenizer.func(baseball)  
cricket.tokenized <- tokenizer.func(cricket, sp = 'cricket')  
  
# combine baseball and cricket  
sports <- bind\_rows(baseball.tokenized, cricket.tokenized)  
  
head(sports)

## sport word  
## 1 baseball baseball  
## 2 baseball from  
## 3 baseball wikipedia  
## 4 baseball the  
## 5 baseball free  
## 6 baseball encyclopedia

## The Distribution of text data

Text data often follows a lognormal distribution, because the English language repeatably uses the same words, such as ‘is’, ‘the’, ‘a’, etc. Below is the plot of the proportional distributions of words from both the baseball and cricket text.

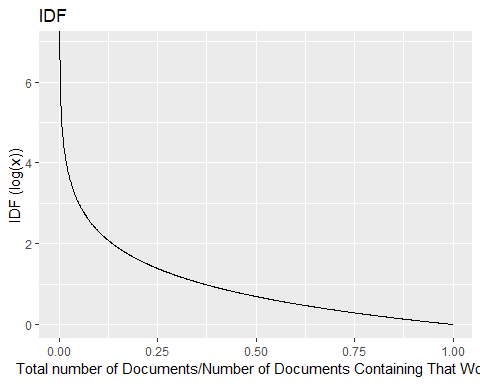


In the combined dataframe, the words ‘the’, ‘of’, ‘a’, ‘and’, ‘in’, ‘to’ occur around 20% of the time! These type of words are often referred to as ‘stop words’ and sometimes removed in the preprocessing step.

Sometimes, it may be appropriate to use frequency counts as a vector representation for text, but often it may not be an accurately describe the features. This is the ‘bag of words’ vector representation.

## Forming the idf with the natural logarithm

Frequency counts don’t really tell us much about the sport of baseball and cricket, and we need to be able to differentiate characteristic words from noise. The idf (inverse document frequency) achieves just this by decreasing its weight for words that occur more often. See the graph below.



For example, since the word ‘the’ occurs in both documents, idf: log(2/2) = 0. However, since the word ‘cricket’ occurs in only one of the documents, idf: log(2/1) = 0.69. Hence, the words that are unique to each document will receive a higher weight than those that occur frequently.

## The tf-idf statistic

Term frequency is still important, i.e. if baseball occurs once in a document it may not be as much about baseball as a document that contains the word baseball 50 times. So, we have arrived at the full version of the tf-idf statistic:

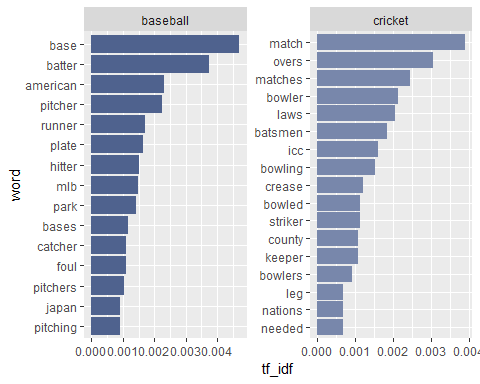
Insert here $ (2) $

Variations include: - various normalization techniques for the idf, including addition of 1 to the logarithm to avoid division by 0. - squaring of the statistic in practice (?) - normalization of the tf statistic to avoid bias on text length

## Using the tf-idf statistic on the cricket/baseball example

Below are the results of calculating the tf-idf on the cricket/baseball data, using the bind\_tf\_idf function in the tidytext package.

sports\_tfidf <-   
 wordfreq %>%  
 bind\_tf\_idf(term = word, document = sport, n = N)  
  
sports\_tfidf %>%  
 arrange(desc(tf\_idf)) %>%  
 mutate(word = factor(word, levels = rev(unique(word)))) %>%  
 group\_by(sport) %>%  
 top\_n(15, tf\_idf) %>%  
 ungroup %>%  
 ggplot(aes(word, tf\_idf, fill = sport)) +   
 scale\_fill\_manual(values = pal[c(1,2,4)])+  
 geom\_col(show.legend = FALSE) +  
 facet\_wrap(~sport, scales = 'free') +   
 coord\_flip()



This is much more informative! For example, I now know that bowler is to cricket as pitcher is to baseball. From the ‘american’ and ‘mlb’, I can guess that baseball is popular in America, but ‘icc’ and ‘nations’ suggests that cricket is more popular as an international sport. From an exploratory sense, this is much more informative.

## Simple Information Retrieval

In its simplest form an information retrieval engine will sum up the tf\_idf of the words in the query.

query <- c('What', 'sport', 'is', 'played', 'on', 'a', 'diamond')  
  
sports\_tfidf %>%  
 group\_by(sport) %>%  
 filter(word %in% query) %>%  
 summarize(tfidf\_sum = sum(tf\_idf))

## # A tibble: 2 x 2  
## sport tfidf\_sum  
## <chr> <dbl>  
## 1 baseball 0.000184  
## 2 cricket 0

## Various methods for text vectorization

There are several commands in both R and Python that will produce a Document Term Matrix with tf-idf weightings. Something to note is that often these matrices take a term frequency vector instead of a vector of text. Below is a list and notes on a few of them. As is typical of R, there is probably a plethora of functions to do the same thing, and conversely in Python, one function with a plethora of options.

R: - DocumentTermMatrix or TermDocumentMatrix. I didn’t try this one as my data was in a dataframe and not structured as a corpus. It does have a control function to choose the tf-idf weightings (weightTfIdf) with the option to normalize the term frequencies. - cast\_dtm. Better option if you have a tidy dataframe of terms and don’t want to work through generating a corpus structure. Requires a count column (value). The tf-idf weighting is weighting = tm::weightTfIdf and can normalize the term frequency.

Python: - TfIdfTransformer. Contains a couple idf normalization techniques, and a smoother. This function merely transforms the matrix, which means you will have to form the frequency counts first, CountVectorizer can achieve this. This function is useful if you want to take a more step-by-step approach. - TfIdfVectorizer. Performs CountVectorizer and TfIdfTransformer at once, also has some preprocessing options.

## Pitfalls of tf-idf

The tf-idf statistic pays little attention to relevancy/context. For example, for the query, ‘unique red hat’, I can game a simple system. A user may try this approach if they want their product to appear more often in a search result.

table <- tibble('Document' = c('doc1', 'doc2','doc3'), 'Results' = c('unique, unique, unique, unique, purple, purple, shirt', 'I tipped my cap to the red team', 'This red hat sells for cheap at JcPenny.'))  
table2 <-   
 unnest\_tokens(table, output = 'word', input = Results) %>%  
 group\_by(Document, word) %>%  
 summarise(N = n()) %>%  
 ungroup() %>%  
 bind\_tf\_idf(term = word, document = Document, n = N) %>%  
 filter(word %in% c('unique', 'red', 'hat')) %>%  
 group\_by(Document) %>%  
 summarize(sum.tfidf = sum(tf\_idf)) %>%  
 select(sum.tfidf)  
   
kable(table %>% bind\_cols(table2), format = 'markdown')

|  |  |  |
| --- | --- | --- |
| Document | Results | sum.tfidf |
| doc1 | unique, unique, unique, unique, purple, purple, shirt | 0.6277785 |
| doc2 | I tipped my cap to the red team | 0.0506831 |
| doc3 | This red hat sells for cheap at JcPenny. | 0.1880097 |

In this context, hat is probably more important than the word unique, but the tf-idf ignores this relevancy.

## Use-case - Document Similarity

Since the tf-idf extracts the characteristic words for a document, it is also useful in identifying the similarity between documents. As a use-case, if a company uses common language in its documents, the resulting idf would be low, essentially labeling those documents as ‘typical’. Put through some type of similarity measure/clustering model, this will result in a strong relationship to the similar documents.

I have been coaching high school softball for 6 years, and I was curious to know if I have started to repeat myself. As an example, I extracted two different emails about tryouts and one spam email. I found the tf-idf and then calculated the cosine similarity for each document. My emails are not similar (they should have a ratio close to 1) - but the spam email is not similar at all!

print(mat)

## spam tryouts2018 tryouts2019  
## spam 1.000000000 0.009587791 0.002783789  
## tryouts2018 0.009587791 1.000000000 0.158711104  
## tryouts2019 0.002783789 0.158711104 1.000000000

## How does Google search?

There it is, but if I search tf idf, I can no longer find it.

## Citations

[1] Silge, J., & Robinson, D. (2019, March 23). Text Mining with R. Retrieved from <https://www.tidytextmining.com/>

[2] LuceneSolrRevolution. (2013, May 29). Beyond TF-IDF: Why, What and How. Retrieved from <https://www.youtube.com/watch?v=C25txE_dq90>

[3] Tf–idf. (2019, April 25). Retrieved from <https://en.wikipedia.org/wiki/Tf–idf>