Term Frequency- Inverse Document Frequency (TF-IDF)

Promothesh Chatterjee*

Term Frequency - Inverse Document Frequency (TFIDF) Representation of Text Data

Consider an online marketplace that wants to understand important complaint behavior from reviews. One way to do it is using TFIDF.

An important concept to representing words as features is TF-IDF or Term Frequency - Inverse Document Frequency. Two things which make it very popular: 1) it is very easy to interpret and 2) it is super easy to calculate. TF-IDF relies on the concept of the document term matrix from the bag-of-words model and aims to solve a few of the issues that we run into using the document term matrix model. TFIDF gauges the significance of a term in a document compared to its significance in a collection of documents (corpus).

If you think of the term frequency model in the document term matrix, the way the terms are represented is purely based on their frequencies. If a particular word is present twice in a given document or in a given record,the score will be two for that word in that particular document. If you have a thousand documents and you have a word which occurs in 900 of those 1000 documents, such a high representation of a particular word may not help in the task at hand, because the discriminability is low.

Put simply, when a term appears frequently in a particular document but is uncommon in others, it is considered to be highly important. This significance is calculated using two metrics: term frequency (TF) measures how often the term appears in the document, while inverse document frequency (IDF) measures how rare the term is across all documents. The TF-IDF score is the final result that determines the term's importance within a given document collection.

Let's look at some of the hotel reviews from the dataset we have been considering. As usual, I am using the Quanteda package to generate a DFM. Assume there are 4 documents (each a separate review): a,b,c,d. As before we will use the Quanteda function *tokens* for preprocessing, followed by *dfm* function to generate Document Term Matrix.

```
library(quanteda)
a<-"This Residence Marriott is not fancy."
b<-"It was clean, good service and suite style rooms;"
c<-"However, it's a little older than some Residence Marriotts's
    I've stayed at and probably could use an update soon."
d<-"You can count on Residence Marriott to deliver on its promises"

hotel_demo<-c(a,b,c,d)
hotel_demo_token <- quanteda::tokens(hotel_demo,
                        remove_numbers = TRUE, remove_punct = TRUE,
                        remove_symbols = TRUE, remove_hyphens = TRUE)

hotel_demo_token<-dfm(hotel_demo_token)

hotel_demo_token1<-as.matrix(hotel_demo_token)
hotel_demo_token1[1:4,1:10]
```

```
##        features
```

```
## docs    this residence marriott is not fancy it was clean good
##   text1    1         1        1  1   1     1  0   0     0    0
##   text2    0         0        0  0   0     0  1   1     1    1
##   text3    0         1        0  0   0     0  0   0     0    0
##   text4    0         1        1  0   0     0  0   0     0    0
```

If you look at the word "residence", it occurs in 3 out of 4 documents. Or think about stop words such as "the" etc. which occur a lot in any document. This really adds nothing to our understanding as it is a fairly common word in the context and doesn't help us discriminate between one review and another. So while term frequencies are useful in many cases, often we need a better representation for our textual data.

Let's look at a hypothetical example and compute cosine similarity between two hypothetical reviews (we will elaborate on this later, but for now it just a measure of similarity between documents)

```
x1<-"My stay at the Residence Marriott was terrible and location was bad"
x2<-"The highlight of my stay at the Residence Marriott was its service"
x3<-"Service at the hotel was very good"
x1<-dfm(x1)
x2<-dfm(x2)
x3<-dfm(x3)
x1
```

```
## Document-feature matrix of: 1 document, 11 features (0.00% sparse) and 0 docvars.
##        features
## docs    my stay at the residence marriott was terrible and location
##   text1  1    1  1   1         1        1   2        1   1        1
## [ reached max_nfeat ... 1 more feature ]
```

```
x2
```

```
## Document-feature matrix of: 1 document, 11 features (0.00% sparse) and 0 docvars.
##        features
## docs    the highlight of my stay at residence marriott was its
##   text1   2         1  1  1    1  1         1        1   1   1
## [ reached max_nfeat ... 1 more feature ]
```

```
x3
```

```
## Document-feature matrix of: 1 document, 7 features (0.00% sparse) and 0 docvars.
##        features
## docs    service at the hotel was very good
##   text1       1  1   1     1   1    1    1
```

```
# compute similarity between reviews x1 and x2
library(quanteda.textstats)# for computing stats between different texts
textstat_simil(
  x1,
  x2,
  margin = ("documents"),
  method = ( "cosine"),
  )
```

```
## textstat_simil object; method = "cosine"
##       text1
## text1 0.643
```

```
# compute similarity between reviews x2 and x3
textstat_simil(
  x2,
  x3,
  margin = ("documents"),
  method = ( "cosine"),
  )
```

```
## textstat_simil object; method = "cosine"
##       text1
## text1 0.505
```

We know that the first two reviews are opposite in valence, but if you look at cosine value it is fairly high (indicating a high level of similarity) whereas reviews 2 and 3 are similar in nature but their cosine similarity is lower. The reason being that there are common terms across the reviews that have high frequencies. The main problem with the term frequency approach is that it scales up the frequent term and scales down the rare terms (the calculation entails computing a dot product–more on this later). Often it is the rare terms which may more important to identify different trends.

Another factor that affects term frequency is the length of the document. So the moment you are counting frequency of words in the document, it will be biased by the length of the document. If you have a review that is ten sentences long compared to another one that is fifty sentences long, some of the words will have higher probability of occurring and with higher frequency in the lengthy review than in the shorter review. So there has to be a way to balance these as factors in representation of textual data.

So, there are two issues that we need to address in textual representation:

1) Length of a document

2) High frequency of tokens in a document representing a particular context The implications of these issues are that not all terms are equally important. Common words may occur across different documents whereas important terms often occur within each document.

While there are different ways to address these issues, here is the most common one:

1) Can be solved by taking the proportion TF = (Number of time a word is present in document d) / (Total number of words in document d)

For example, if we have a document that has N words and then instead of counting the number of occurrence of a particular word, we divide all the term frequencies by N. Instead of looking at the frequencies, we are now looking at the proportions. Note that all the rows of the document term matrix will add up to one, because we are representing the words as a proportion of total number of words present in the document, this reduces the bias due to size of the document.

2) Can be solved by Inverse Document Frequency. There is the problem of high frequency of tokens in a document , that is, what to do to if there are certain words that are present across most of the documents. We compute something known as inverse document frequency, IDF. In here, we are trying to scale down the impact of frequency of words by taking the log of the total number of documents divided by number of documents having a particular term t. Consider the word 'residence' from our DFM above.

If we want to calculate the inverse document frequency, IDF we can look at the word 'residence' and then find that how many documents are there in our data which contain that word. We then divide it by the total number of documents in corpus. So in our case, it is 3/4. This gives us the proportion of documents in our data that contain this particular word. Note that instead of taking log of 3/4, we take the log of the reciprocal of 3/4. If we take log of a proportion, the resultant is negative term. Once we have our weighting factors, TF and IDF, we multiply TF with IDF to get a weighting factor.

IDF= log [(Total number of documents)/ (Total number of documents having term t)]

TF-IDF= TF x IDF

A few things to note. First, the term frequency of a word will vary from row to row in the document term matrix because one word may be present a certain number of times in one of the documents but a different number of times in others. However, the inverse document frequency of a word that doesn't vary because it is log of reciprocal of the proportion of documents in which that word is present.

```
hotel_demo_token1[1:4,1:10]
```

```
##        features
## docs    this residence marriott is not fancy it was clean good
##   text1    1         1        1  1   1     1  0   0     0    0
##   text2    0         0        0  0   0     0  1   1     1    1
##   text3    0         1        0  0   0     0  0   0     0    0
##   text4    0         1        1  0   0     0  0   0     0    0
```

In our DFM above, the TF for the word 'residence' in the first row is 1/6, as there is 1 occurrence of the term 'residence' out of 6 total terms in the document. IDF for 'residence' is the log of reciprocal of 3/4, log (4/3). So, the IDF for 'residence' is 0.1249. Thus, the TFIDF for the word 'residence' in the first document (row) is 1/6 x 0.1249 = 0.0208. Because the word 'residence' occurs fairly frequently, it is weighted by a low number. This is low value (near 0) implies that the word residence does not add much meaning and is a common term.

To some it up, term frequency takes into account the frequency of the terms but the inverse document frequency finds out if a term is rare in the document. Let's compute the TFIDF for the document term matrix in our example. We will use the function dfm_tfidf from R package Quanteda. It take a DFM as an input. We additional specify the weighting schemes as proportion and inverse for term frequency and document frequency respectively.

```
#docfreq(hotel_demo_token)#### gives the frequency of words in DFM

tfidf<-data.frame(dfm_tfidf(hotel_demo_token, scheme_tf = "prop", scheme_df = "inverse", base = 10))
tfidf[1:4,1:5]
```

```
##   doc_id      this   residence   marriott        is
## 1  text1 0.1003433 0.020823123 0.05017167 0.1003433
## 2  text2 0.0000000 0.000000000 0.00000000 0.0000000
## 3  text3 0.0000000 0.006575723 0.00000000 0.0000000
## 4  text4 0.0000000 0.011358067 0.02736636 0.0000000
```

The r output verifies that the TFIDF for 'residence' in the first row is 0.0208 as we computed manually. And you can see the actual weight given to the frequently occurring term 'residence' is very low.

Interpreting TFIDF Let's create a short reviews dataset to understand how to interpret the weights TFIDF assigns to document term matrix.

```
# Preprocessing
library(quanteda)

# Dummy Dataset
reviews <- c("I absolutely love this product. It works perfectly and the design is sleek.",
"The product is of absolutely poor quality. It broke after a few uses.",
"The customer service was excellent. They promptly resolved my issue with the product.",
"The price of the product is reasonable for the features it offers.",
"I had a terrible experience with this product's quality. It doesn't work as advertised.")

ratings <- c(5, 2, 4, 3, 1)

data <- data.frame(review = reviews, rating = ratings)
```

After the usual preprocessing, let us apply the TFIDF function on the dataset.

```
# Create a corpus
corpus_df <- corpus(data$review)

# Tokenization
tokens_df <- tokens(corpus_df, remove_punct = TRUE, remove_numbers = TRUE,
                    remove_symbols = TRUE)

# Create a document-feature matrix (DFM)
dfm_df <- dfm(tokens_df) %>% dfm_remove(stopwords("english"))

# Calculate TF-IDF
tfidf_df <- dfm_tfidf(dfm_df)

# Convert TF-IDF to a data.frame for interpretation
tfidf_df <- convert(tfidf_df, to="data.frame")

# Add the rating column back to the TF-IDF dataframe
tfidf_df$rating <- data$rating
head(tfidf_df,c(5,6))
```

```
##   doc_id absolutely    love    product   works perfectly
## 1  text1    0.39794 0.69897 0.09691001 0.69897   0.69897
## 2  text2    0.39794 0.00000 0.09691001 0.00000   0.00000
## 3  text3    0.00000 0.00000 0.09691001 0.00000   0.00000
## 4  text4    0.00000 0.00000 0.09691001 0.00000   0.00000
## 5  text5    0.00000 0.00000 0.00000000 0.00000   0.00000
```

Now, let's interpret the findings:

1. High TF-IDF scores: Words with higher TF-IDF scores indicate their importance within a specific review compared to the rest of the corpus. These terms are considered more distinctive to that particular review. For example, if "love" and "perfectly" have high TF-IDF scores in a positive review , it suggests that these words contribute significantly to the sentiment and description of the positive experience.

2. Low TF-IDF scores: Words with lower TF-IDF scores are more common and occur in multiple reviews. These terms have less discriminative power as they are not specific to a particular review. For instance, the words "product" has a low TF-IDF score since it appears frequently across various reviews.

3. Relationship with ratings: You can observe how the TF-IDF scores relate to the ratings provided by customers. Compare the TF-IDF scores of words in positive reviews (high ratings) with those in negative reviews (low ratings). This analysis can provide insights into the important factors that contribute to positive or negative sentiments.

Overall, TF-IDF helps in identifying the words that carry more weight and significance within individual reviews. It allows you to extract meaningful insights from customer feedback by highlighting the terms that contribute the most to the uniqueness and sentiment of each review.