KDnuggets

WTF is TF-IDF?

<= Previous post
Next post =>

Tags: Information Retrieval, Python, Text Analytics, Text Mining, TF-IDF

Relevant words are not necessarily the most frequent words since stopwords like “the”, “of” or “a” tend to occur very often in many documents.

---

comments

By Enrique Fueyo, CTO & Co-founder @ Lang.ai

Frame from “The Incredibles” (2004) movie

TF-IDF, which stands for **term frequency—inverse document frequency,** is a scoring measure widely used in information retrieval (IR) or summarization. TF-IDF is intended to reflect **how relevant a term is in a given document.**

The intuition behind it is that if a word occurs *multiple times in a document*, we should boost its relevance as it should be more meaningful than other words that appear fewer times (TF). At the same time, if a word occurs many times in a document but also *along many other documents*, maybe it is because this word is just a frequent word; not because it was relevant or meaningful (IDF).

Defining what a “relevant word” means

We can come up with a more or less subjective definition driven by our intuition: a word’s relevance is proportional to the amount of information that it gives about its context (a sentence, a document or a full dataset). That is, the most relevant words are those that would help us, as humans, to better understand a whole document without reading it all.

As pointed out, **relevant words are not necessarily the most frequent words** since stopwords like “the”, “of” or “a” tend to occur very often in many documents.

There is another caveat: if we want to summarize a document compared to a whole dataset about an specific topic (let’s say, movie reviews), there will be words (other than stopwords, like *character* or *plot*), that could occur many times in the document as well as in many other documents. These words are not useful to summarize a document because they convey little discriminating power; they say very little about what the document contains compared to the other documents.

Let’s go through some examples to better illustrate how TF-IDF works.

Search engine example

Let’s suppose we have a database with thousands of cats descriptions and **a user wants to search for furry cats**, so she/he issues the query *“the furry cat”*. As a search engine, we have to decide which documents should be returned from our database.

If we have documents that match the exact query, there is no doubt but… what if we have to decide between partial matches? For simplicity, let’s say we have to choose between these two descriptions:

1. “the lovely cat”
2. “a furry kitten”

The first description contains **2 out of 3 words from the query** and the second one matches **just 1 out of 3**, then we would pick the first description. How can TF-IDF help us to choose the second description instead of the first one?

The TF is the same for each word, no difference here. However, we could expect that **the terms “cat” and “kitten” would appear in many documents** (large document frequency implies low IDF), while **the term “furry” will appear in fewer documents** (larger IDF). So the TF-IDF for cat & kitten has a low value whereas the TF-IDF is larger for “furry”, i.e. in our database **the word “furry” has more discriminative power than “cat” or “kitten”.**

Conclusion

If we use the TF-IDF to weight the different words that matched the query, “furry” would be more relevant than “cat” and so we could eventually choose “the furry kitten” as the best match.

Summarization example

Now, imagine we would like to **automatically summarize some movies using their wikipedia pages**. We plan to create a tagcloud that will help us to understand what each movie is about, so our task is to decide what words to put in it (and their sizes).

Given that a month ago my family adopted a new dog called *Mawi* (named after Moana's character Maui), we will use Moana's wikipedia page for this example.

My dogs: Cala & Mawi

The first idea would be to make a list with the most frequent words:

> **Moana**
> **Most frequent:** ['film', 'moana', 'the', 'million', 'disney', 'maui', 'day', 'release', 'te', 'animation', 'weekend', 'heart', 'ocean', 'it', 'story', 'island', 'fiti', 'version', 'in', 'animate']

We can see that some words could be useful to grasp a little bit of the film's plot: moana, maui, ocean, island… **but they still provide very little information.** Moreover, they are mixed with other words that are irrelevant and just add noise (like film, million, day, release or weekend).
If we run the same experiment with other movies, we can see that the results are somehow similar regarding irrelevant and noisy words. Wikipedia pages include information that is shared or have a similar structure among all them: They are *animated films* that made *millions* of dollars since the first *day* of their *release*… but this is something we already knew or, at least, it is not relevant for us in order to understand the plot.

Overcoming the problem

To eliminate what is shared among all movies and extract **what individually identifies each one**, TF-IDF should be a very handy tool. With the most frequent words (TF) we got a first approximation, but IDF should help us to refine the previous list and get better results. Taking into account the document frequency, we can try to penalize generic words (these that appeared in many wikipedia pages) reducing their relevance.

> **Moana**
> **TF-IDF:** ['maui', 'te', 'moana', 'fiti', 'cravalho', 'goddess', 'tui', 'polynesian', 'tala', 'kā', 'māori', 'auli'i', 'clement', 'fishhook', 'tamatoa', 'jemaine', 'tattoo', 'dubbing', 'musker', 'clements']

Thanks to TF-IDF words like *film, million* or *release* have disappeared from the top of the list and we got some new more meaningful words like *polynesian* or *tattoo.*

We can also run the same analysis with the other reviews and get the following results where, again, the generic words have disappeared in favor of more movie-specific words:

MOANA

Most relevant	TF - IDF
film	maui
moana	te
the	moana
million	fiti
disney	cravalho
maui	goddess
day	tui
release	polynesian

THE INCREDIBLES

Most relevant	TF - IDF
film	parrs
the	syndrome
incredibles	violet
bird	omnidroid
pixar	parr
release	mirage
bob	nomanisan
jack	helen

MONSTERS INC.

Most relevant	TF - IDF
film	sulley
sulley	waternoose
monsters	boo
the	cda
mike	randall
monster	scarer
pixar	fizt
story	celia

Most relevant VS TF-IDF comparison

For reference, these are the sorted lists with more words:

Moana

Most frequent: [‘film’, ‘moana’, ‘the’, ‘million’, ‘disney’, ‘maui’, ‘day’, ‘release’, ‘te’, ‘animation’, ‘weekend’, ‘heart’, ‘ocean’, ‘it’, ‘story’, ‘island’, ‘fiti’, ‘version’, ‘in’, ‘animate’]

TF-IDF: [‘maui’, ‘te’, ‘moana’, ‘fiti’, ‘cravalho’, ‘goddess’, ‘tui’, ‘polynesian’, ‘tala’, ‘kā’, ‘māori’, ‘auli‘i’, ‘clement’, ‘fishhook’, ‘tamatoa’, ‘jemaine’, ‘tattoo’, ‘dubbing’, ‘musker’, ‘clements’]

—

The Incredibles

Most frequent: [‘film’, ‘the’, ‘incredibles’, ‘bird’, ‘pixar’, ‘release’, ‘bob’, ‘jack’, ‘i’, ‘award’, ‘good’, ‘animate’, ‘feature’, ‘it’, ‘syndrome’, ‘character’, ‘family’, ‘work’, ‘superhero’, ‘animation’]

TF-IDF: [‘parrs’, ‘syndrome’, ‘violet’, ‘omnidroid’, ‘parr’, ‘mirage’, ‘nomanisan’, ‘helen’, ‘bird’, ‘edna’, ‘superhero’, ‘frozone’, ‘underminer’, ‘iron’, ‘wallin’, ‘suburban’, ‘metroville’, ‘dash’, ‘incredibles’, ‘incredible’]

—

Monsters, Inc

Most frequent: [‘film’, ‘sulley’, ‘monsters’, ‘the’, ‘mike’, ‘monster’, ‘pixar’, ‘story’, ‘child’, ‘randall’, ‘boo’, ‘disney’, ‘character’, ‘work’, ‘good’, ‘in’, ‘release’, ‘he’, ‘docter’, ‘fur’]

TF-IDF: [‘sulley’, ‘waternoose’, ‘boo’, ‘cda’, ‘randall’, ‘scarer’, ‘fizt’, ‘celia’, ‘kahrs’, ‘sullivan’, ‘fur’, ‘factory’, ‘scare’, ‘monster’, ‘wazowski’, ‘mike’, ‘tentacle’, ‘madrid’, ‘pidgeon’, ‘laughter’]

How we did it

Here it is the code we have used to obtain the relevant words lists used in this article:

```
# Install dependencies first
#     pip install spacy wikipedia
#     python -m spacy download en
# Run
#     python tfidf.py

from collections import Counter
import math

import wikipedia
import spacy

nlp = spacy.load('en')

pages = [
  "The Tigger Movie 2000", "Dinosaur 2000 Movie", "The Emperor's New Groove 2000", "Recess School's Out", "Atlantis: The Lost Empire", "Monste
  "A Christmas Carol 2009", "The Princess and the Frog 2009", "Toy Story", "Toy Story 2", "Toy Story 3", "Tangled", "Mars Needs Moms", "Cars 2
  "Shrek", "Shrek 2", "Shrek 3", "Antz", "A bugs life", "Bee movie", "Madagascar 2005 film", "Madagascar 2", "Kung fu panda", "Kung fu panda 2
]

def valid_token(tk):
    is_valid = tk.is_alpha
    return is_valid and not tk.is_stop

def get_lemma(tk):
    if tk.pos_ == 'PRON' or tk.lemma_ == '-PRON-':
        return tk.text.lower()
    return tk.lemma_.lower()

def read_wikipedia_page(page_name):
    page = wikipedia.page(page_name)
    content = page.content
    return content

def tokenize_page(page_name):
  text = read_wikipedia_page(page_name)
  return [
    get_lemma(t)
    for t in nlp(text)
    if valid_token(t)
  ]

vocabulary = set()
idf_counter = Counter()

for page in pages:
  print("   Processing page {}...".format(page))
  page_words = set(tokenize_page(page))
  vocabulary = vocabulary | page_words
  idf_counter.update(page_words)

print("All pages processed")

idf = {
  word: math.log(len(pages)/df, 2)
  for word, df in idf_counter.items()
}

print("vocabulary size: {}".format(len(vocabulary)))


def analyze_page(target_page):
  target_words = tokenize_page(target_page)
```

```
    tfidf =  {
      word: (1 + math.log(_tf, 2)) * idf[word]
      for word, _tf in Counter(target_words).items()
    }
    num_words = 20
    most_frequent = [
      w  for (w, _) in Counter(target_words).most_common(num_words)
    ]
    sorted_tfidf = [
      w  for (w, _) in sorted(tfidf.items(), key=lambda kv: kv[1], reverse=True)
    ]
    print(target_page)
    print("Most frequent: {}".format(most_frequent))
    print("Higher TF-IDF: {}".format(sorted_tfidf[:num_words]))


analyze_page("Moana")
analyze_page("The Incredibles 2004")
analyze_page("Monsters, Inc")
```

tfidf.py hosted with ♡ by GitHub view raw

Next steps

For the base set of documents we have used a collection of 65 hand-picked movies. This was only for demo purposes but, to get better results with the TF-IDF, we should use a much larger document base.

Some other experiments we could easily try while using this code as a starting point are:

a) Try different weighting scores for either term-frequency or document-frequency. How do they affect the list?
b) Use a different document base. What would have happened if we had chosen any movies instead of focusing on a single genre?
c) How are the results if we try a different domain like music bands?

Check the other articles in our Building Lang.ai publication. We write about Machine Learning, Software Development, and our Company Culture.

Bio: Enrique Fueyo is CTO and co-founder at Lang.ai, working on unsupervised AI for language understanding to build products and services that help both companies and developers who have to deal with unstructured text data.

Original. Reposted with permission.

Related:

- Understanding What is Behind Sentiment Analysis – Part 1
- Understanding What is Behind Sentiment Analysis – Part 2
- A General Approach to Preprocessing Text Data

<= Previous post
Next post =>

Top Stories Past 30 Days

Most Popular

1. **Five Cool Python Libraries for Data Science**
2. **The Super Duper NLP Repo: 100 Ready-to-Run Colab Notebooks**
3. **Natural Language Processing Recipes: Best Practices and Examples**
4. **24 Best (and Free) Books To Understand Machine Learning**
5. **Free High-Quality Machine Learning & Data Science Books & Courses: Quarantine Edition**
6. **Mathematics for Machine Learning: The Free eBook**

Most Shared

1. **Free High-Quality Machine Learning & Data Science Books & Courses: Quarantine Edition**
2. **Beginners Learning Path for Machine Learning**
3. **Should Data Scientists Model COVID19 and other Biological Events**
4. **The Super Duper NLP Repo: 100 Ready-to-Run Colab Notebooks**
5. **Natural Language Processing Recipes: Best Practices and Examples**
6. **AI and Machine Learning for Healthcare**