Understanding TF-IDF for Machine Learning

A gentle introduction to term frequency-inverse document frequency

Anirudha Simha

October 6, 2021

TF-IDF stands for *term frequency-inverse document frequency* and it is a measure, used in the fields of information retrieval (IR) and machine learning, that can quantify the importance or relevance of string representations (words, phrases, lemmas, etc) in a document amongst a collection of documents (also known as a corpus).

Overview of TF-IDF

TF-IDF can be broken down into two parts *TF (term frequency)* and *IDF (inverse document frequency).*

What is TF (term frequency)?

Term frequency works by looking at the frequency of a *particular term* you are concerned with relative to the document. There are multiple measures, or ways, of defining frequency:

- Number of times the word appears in a document (raw count).
- Term frequency adjusted for the length of the document (raw count of occurences divided by number of words in the document).
- Logarithmically scaled frequency (e.g. log(1 + raw count)).
- Boolean frequency (e.g. 1 if the term occurs, or 0 if the term does not occur, in the document).

What is IDF (inverse document frequency)?

Inverse document frequency looks at how common (or uncommon) a word is amongst the corpus. IDF is calculated as follows where ***t*** is the term (word) we are looking to measure the commonness of and ***N*** is the number of documents (d) in the corpus (D).. The denominator is simply the number of documents in which the term, ***t***, appears in.

$$idf(t, D) = log\left(\frac{N}{count\,(d \in D : t \in d)} \right)$$

Image Source: https://monkeylearn.com/blog/what-is-tf-idf/

Note: It can be possible for a term to not appear in the corpus at all, which can result in a divide-by-zero error. One way to handle this is to take the existing count and add 1. Thus making the denominator (1 + count). An example of how the popular library scikit-learn handles this can be seen below.

Scikit-Learn

- $\text{IDF}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$

Standard notation

- $\text{IDF}(t) = \log \frac{n}{\text{df}(t)}$

Image Source: https://towardsdatascience.com/how-sklearns-tf-idf-is-different-from-the-standard-tf-idf-275fa582e73d

The reason we need IDF is to help correct for words like “of”, “as”, “the”, etc. since they appear frequently in an English corpus. Thus by taking inverse document frequency, we can minimize the weighting of frequent terms while making infrequent terms have a higher impact.

Finally IDFs can also be pulled from either a background corpus, which corrects for sampling bias, or the dataset being used in the experiment at hand.

Putting it together: TF-IDF

To summarize the key intuition motivating TF-IDF is the importance of a term is inversely related to its frequency across documents.TF gives us information on how often a term appears in a document and IDF gives us information about the relative rarity of a term in the collection of documents. By multiplying these values together we can get our final TF-IDF value.

$$tf\,idf\,(t, d, D) = tf\,(t, d)\ .\ idf\,(t, D)$$

Image Source: https://monkeylearn.com/blog/what-is-tf-idf/

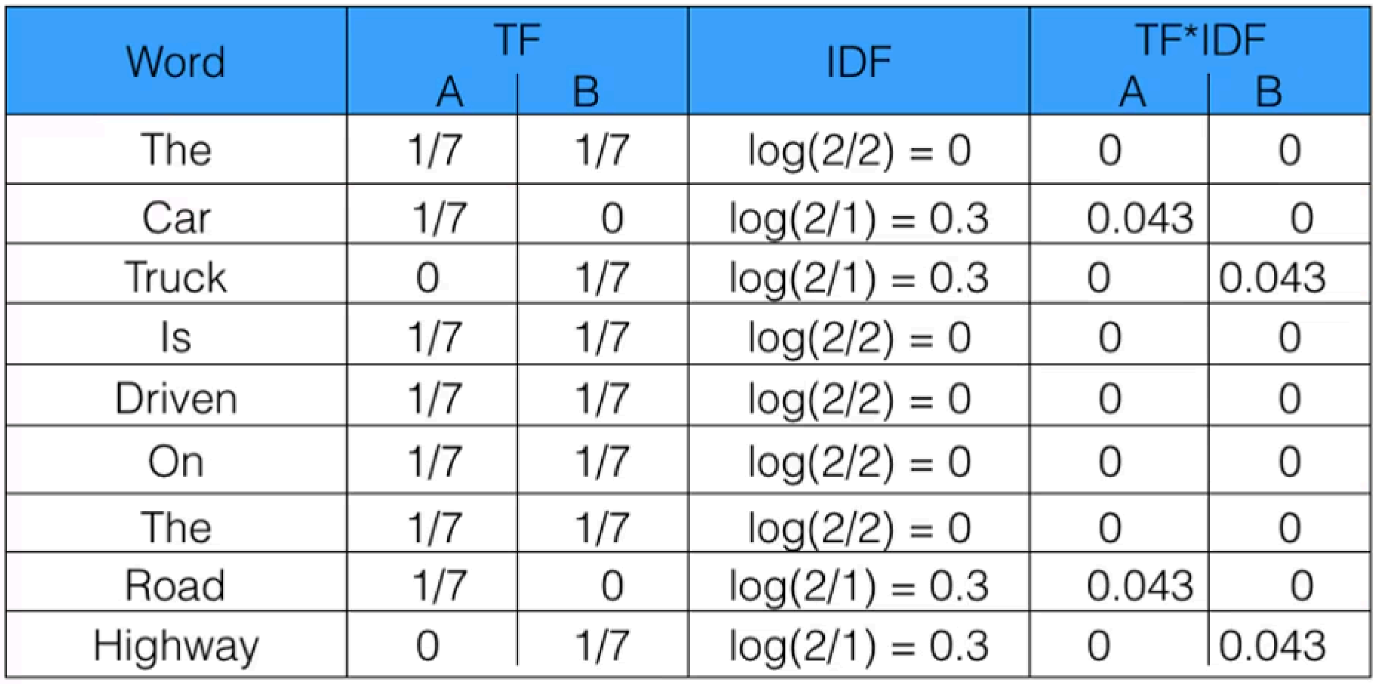
The higher the TF-IDF score the more important or relevant the term is; as a term gets less relevant, its TF-IDF score will approach 0.

Where to use TF-IDF

As we can see, TF-IDF can be a very handy metric for determining how important a term is in a document. But how is TF-IDF used? There are three main applications for TF-IDF. These are in *machine learning, information retrieval,* and *text summarization/keyword extraction.*

Using TF-IDF in machine learning & natural language processing

Machine learning algorithms often use numerical data, so when dealing with textual data or any natural language processing (NLP) task, a sub-field of ML/AI dealing with text, that data first needs to be converted to a vector of numerical data by a process known as vectorization. TF-IDF vectorization involves calculating the TF-IDF score for every word in your corpus relative to that document and then putting that information into a vector (see image below using example documents “A” and “B”). Thus each document in your corpus would have its own vector, and the vector would have a TF-IDF score for every single word in the entire collection of documents. Once you have these vectors you can apply them to various use cases such as seeing if two documents are similar by comparing their TF-IDF vector using cosine similarity.



Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

A = “The car is driven on the road”; B = “The truck is driven on the highway” Image from freeCodeCamp - How to process textual data using TF-IDF in Python (https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/)

Using TF-IDF in information retrieval

TF-IDF also has use cases in the field of information retrieval, with one common example being search engines. Since TF-IDF can tell you about the relevant importance of a term based upon a document, a search engine can use TF-IDF to help rank search results based on relevance, with results which are more relevant to the user having higher TF-IDF scores.

Using TF-IDF in text summarization & keyword extraction

Since TF-IDF weights words based on relevance, one can use this technique to determine that the words with the highest relevance are the most important. This can be used to help summarize articles more efficiently or to simply determine keywords (or even tags) for a document.

Vectors & Word Embeddings: TF-IDF vs Word2Vec vs Bag-of-words vs BERT

As discussed above, TF-IDF can be used to vectorize text into a format more agreeable for ML & NLP techniques. However while it is a popular NLP algorithm it is not the only one out there.

Bag of Words

Bag of Words (BoW) simply counts the frequency of words in a document. Thus the vector for a document has the frequency of each word in the corpus for that document. The key difference between bag of words and TF-IDF is that the former does not incorporate any sort of inverse document frequency (IDF) and is only a frequency count (TF).

Word2Vec

Word2Vec is an algorithm that uses shallow 2-layer, not deep, neural networks to ingest a corpus and produce sets of vectors. Some key differences between TF-IDF and word2vec is that TF-IDF is a statistical measure that we can apply to terms in a document and then use that to form a vector whereas word2vec will produce a vector for a term and then more work may need to be done to convert that set of vectors into a singular vector or other format. Additionally TF-IDF does not take into consideration the context of the words in the corpus whereas word2vec does.

BERT - Bidirectional Encoder Representations from Transformers

BERT is an ML/NLP technique developed by Google that uses a transformer based ML model to convert phrases, words, etc into vectors. Key differences between TF-IDF and BERT are as follows: TF-IDF does not take into account the semantic meaning or context of the words whereas BERT does. Also BERT uses deep neural networks as part of its architecture, meaning that it can be much more computationally expensive than TF-IDF which has no such requirements.

Pros and cons of using TF-IDF

Pros of using TF-IDF

The biggest advantages of TF-IDF come from how simple and easy to use it is. It is simple to calculate, it is computationally cheap, and it is a simple starting point for similarity calculations (via TF-IDF vectorization + cosine similarity).

Cons of using TF-IDF

Something to be aware of is that TF-IDF cannot help carry semantic meaning. It considers the importance of the words due to how it weighs them, but it cannot necessarily derive the contexts of the words and understand importance that way.

Also as mentioned above, like BoW, TF-IDF ignores word order and thus compound nouns like “Queen of England” will not be considered as a “single unit”. This also extends to situations like negation with “not pay the bill” vs “pay the bill”, where the order makes a big difference. In both cases using NER tools and underscores, “queen_of_england” or “not_pay” are ways to handle treating the phrase as a single unit.

Another disadvantage is that it can suffer from memory-inefficiency since TF-IDF can suffer from the curse of dimensionality. Recall that the length of TF-IDF vectors is equal to the size of the vocabulary. In some classification contexts this may not be an issue but in other contexts like clustering this can be unwieldy as the number of documents increases. Thus looking into some of the above named alternatives (BERT, Word2Vec) may be necessary.

Conclusion

TF-IDF (Term Frequency - Inverse Document Frequency) is a handy algorithm that uses the frequency of words to determine how relevant those words are to a given document. It's a relatively simple but intuitive approach to weighting words, allowing it to act as a great jumping off point for a variety of tasks. This includes building search engines, summarizing documents, or other tasks in the information retrieval and machine learning domains.

Anirudha Simha, Principle Associate Software Engineer, Kai Chatbot Team

Anirudha Simha is a passionate software engineer working on a full stack team that develops and supports a quality NLP & ML powered chatbot. He graduated from Virginia Tech in 2017 with a degree in Computer Engineering and minors in Math, Computer Science, and Cyber Security. Anirudha's passion for technology can be summarized by the quote, "Innovation comes out of great human ingenuity and very personal passions". Outside of work, Anirudha is pursuing a Masters in Computer Science and Machine Learning at Georgia Tech; constantly traveling; and enthusiastically following the latest mobile, tech, and gaming news. You can connect with Anirudha on LinkedIn (https://www.linkedin.com/in/anirudha-simha) or Twitter (https://twitter.com/__anirudha__).

RELATED CONTENT

MACHINE LEARNING