Demonstrating Calculation of TF IDF From Sklearn

Explanation of Mathematical logic behind TF-IDF module from sklearn in python

[](https://medium.com/@chouksey98?source=post_page-----4f9526e7e78b--------------------------------)

[Shubham Chouksey](https://medium.com/@chouksey98?source=post_page-----4f9526e7e78b--------------------------------)

Follow

[Apr 21, 2020](https://medium.com/analytics-vidhya/demonstrating-calculation-of-tf-idf-from-sklearn-4f9526e7e78b?source=post_page-----4f9526e7e78b--------------------------------) · 6 min read

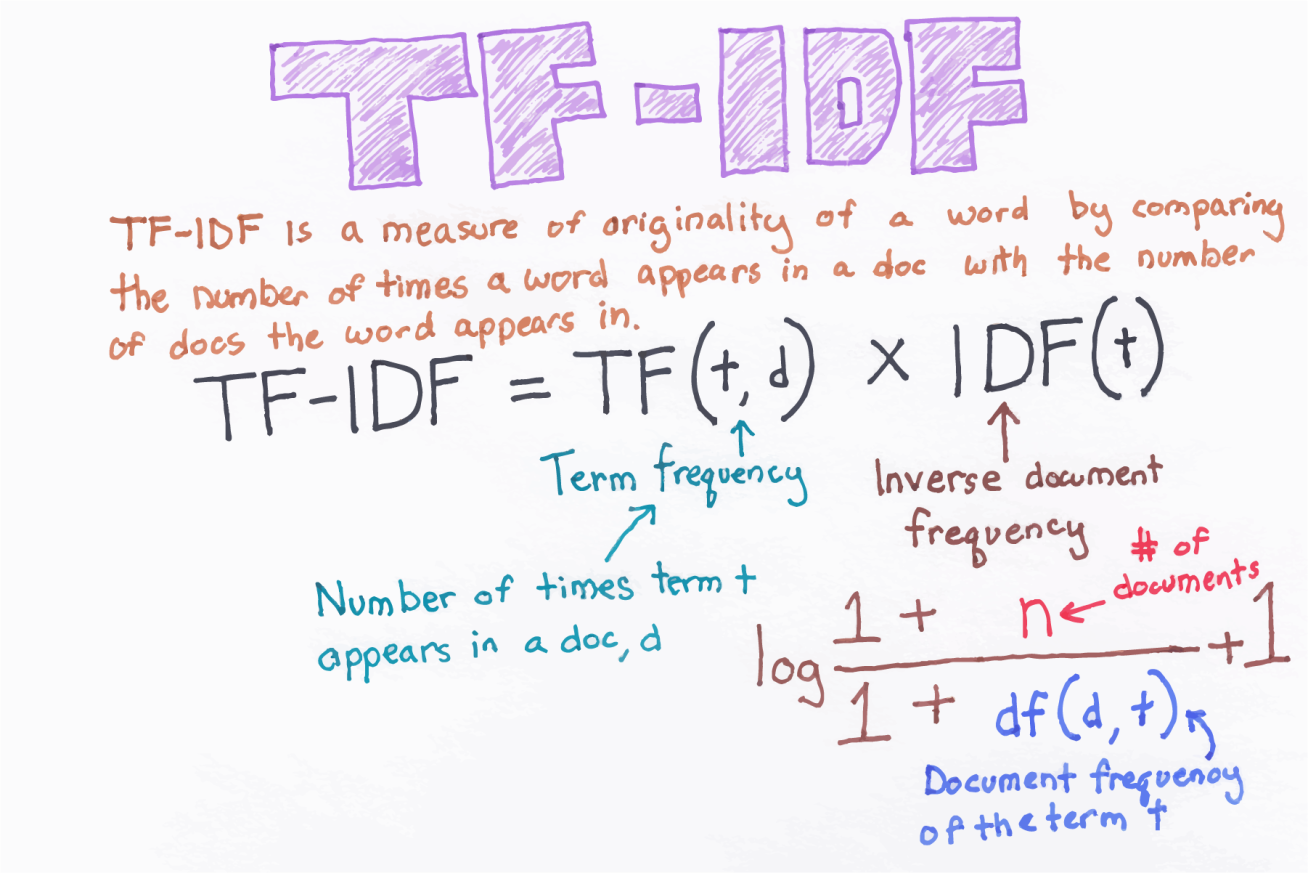


Fig: 1.1

**What is Feature Engineering of text data?**

The procedure of converting raw text data into machine understandable format(numbers) is called feature engineering of text data. Machine learning and deep learning algorithm performance and accuracy is fundamentally dependent on the type of feature engineering techniques used.

In this article, we are going to see Feature Engineering technique using TF-IDF and mathematical calculation of TF, IDF and TF-IDF. After reading this article you will understand the insights of mathematical logic behind libraries such as *TfidfTransformer* from *sklearn.feature\_extraction*package in python.

**TF (Term Frequency) :**

* Term frequency is simply the count of a word present in a sentence
* TF is basically capturing the importance of the word irrespective of the length of the document.
* A word with the frequency of 3 with the length of sentence being 10 is not the same as when the word length of sentence being 100 words. It should get more importance in the first scenario; that is what TF does.

**IDF (Inverse Document Frequency):**

* IDF of each word is the log of the ratio of the total number of rows to the number of rows in a particular document in which that word is present.
* IDF will measure the rareness of a term. word like ‘a’ and ‘the’ show up in all the documents of corpus, but the rare words is not in all the documents.

**TF-IDF:**

It is the simplest product of TF and IDF so that both of the drawbacks are addressed above, which makes predictions and information retrieval relevant.

We are going to play with following corpus containing five documents as:

docs = ['the cat see the mouse',  
 'the house has a tiny little mouse',  
 'the mouse ran away from the house',  
 'the cat finally ate the mouse',  
 'the end of the mouse story'  
 ]

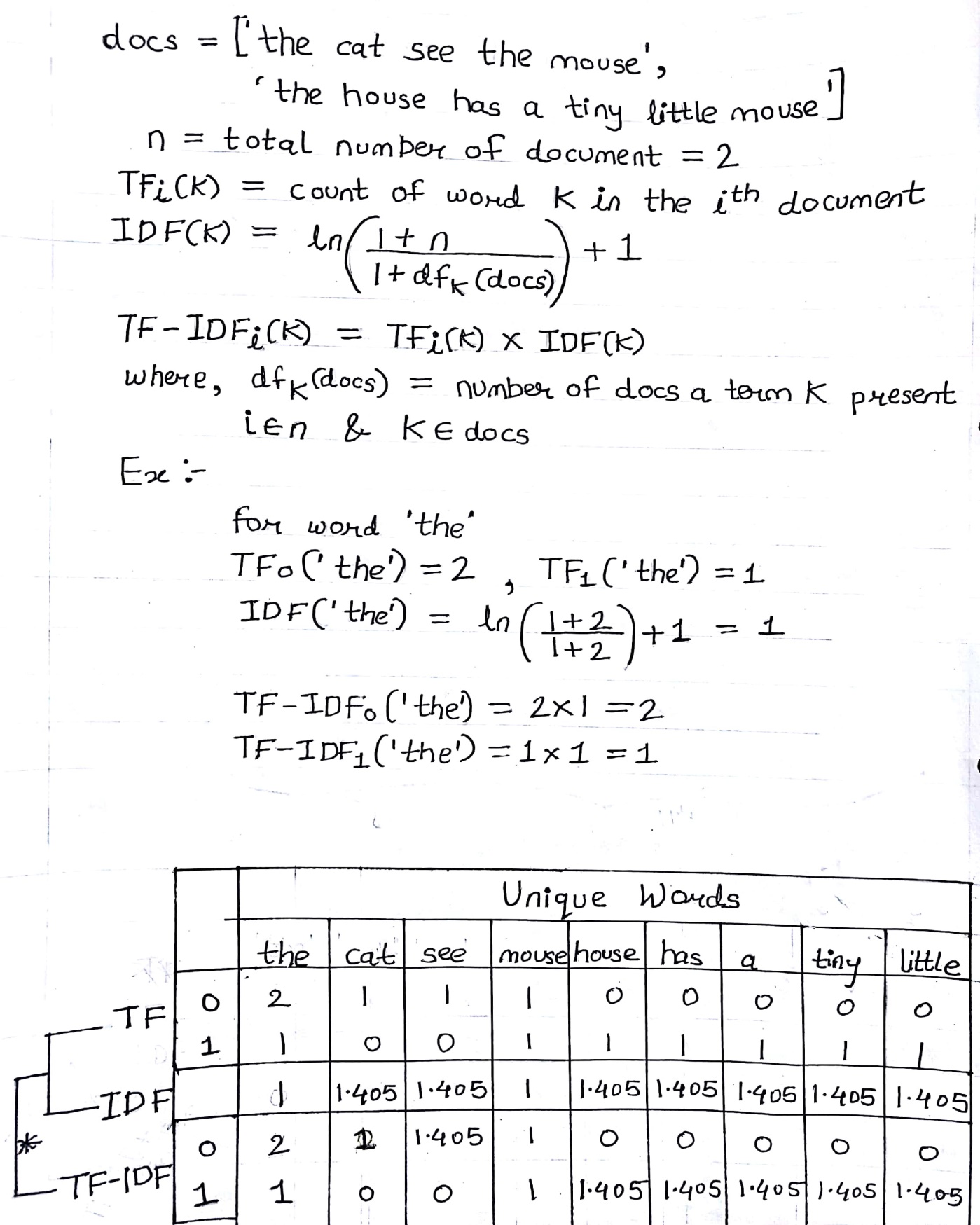


Fig: 1.2

**Extracting features by** using [TfidfTransformer](https://scikit-learn.org/stable/modules/classes.html" \l "module-sklearn.feature_extraction)from [sklearn.feature\_extraction](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html" \l "sklearn.feature_extraction.text.TfidfTransformer) package.

Now import TfidfTransformer and CountVectorizer from sklearn.feature\_extraction module.

from sklearn.feature\_extraction.text import TfidfTransformer  
from sklearn.feature\_extraction.text import CountVectorizer

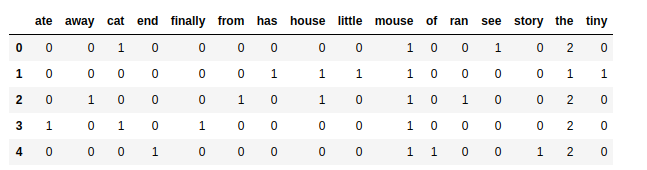
CountVectorizer is used to find the word count in each document of a dataset. Also known as to calculate Term Frequency. To know more click [here](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer)

Let’s see an example,

From the above docs we are going to fit all the sentences using count-vectorizer and get an array of word counts of each document.

import pandas as pd  
cv = CountVectorizer()  
word\_count\_vector = cv.fit\_transform(docs)tf = pd.DataFrame(word\_count\_vector.toarray(), columns=cv.get\_feature\_names())  
print(tf)

output:

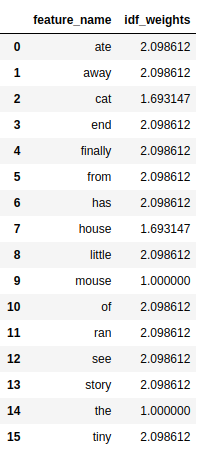


Now, declare TfidfTransformer() instance and fit it with the above word\_count\_vector to get IDF and final normalized features.

**Note:**TfidfTransformer(*norm=’l2'*) by default use ‘l2’ parameter for normalization, where sum of squares of vector(document) element is 1, which is also called as euclidean norm.

tfidf\_transformer = TfidfTransformer()  
X = tfidf\_transformer.fit\_transform(word\_count\_vector)  
idf = pd.DataFrame({'feature\_name':cv.get\_feature\_names(), 'idf\_weights':tfidf\_transformer.idf\_})  
print(idf)

output:



Final feature vectors are:

tf\_idf = pd.DataFrame(X.toarray() ,columns=cv.get\_feature\_names())  
print(tf\_idf)

output:

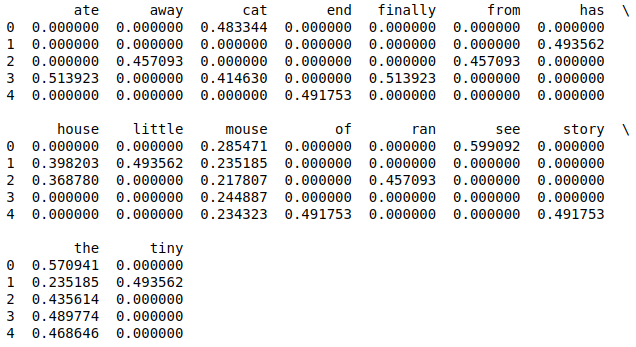


Fig: 1.3

Confused with the above result? don’t worry, Let’s take a look on below formula and calculation for vector 0.

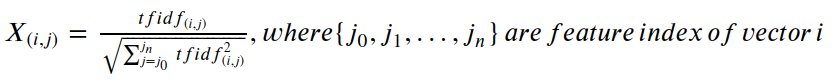


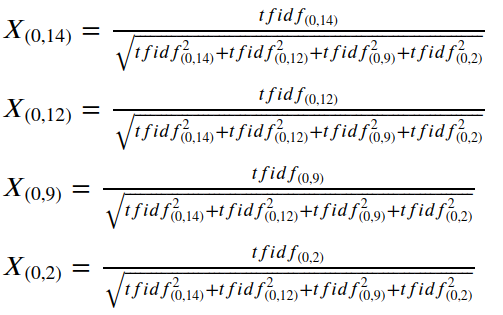
Fig: 1.4

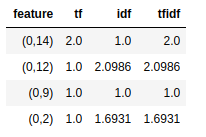
Here in above document set, Vector 0 is docs[0] i.e the cat see the mouse

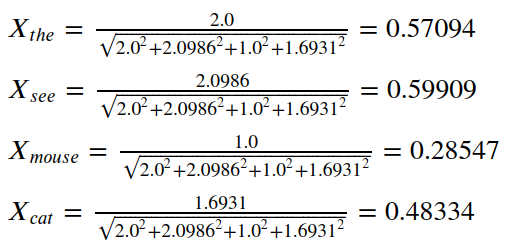
Feature index of vector 0 are:

{‘the’: 14, ‘see’:12, ‘mouse’:9, ‘cat’:2} see indexes ofidfdataframe.

Therefore,







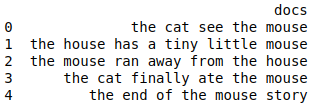
This is how the normalize features are calculated by sklearn package.

Now, if you want to see a practical approach behind sklearn. I am going to illustrate manual approach without using sklearn library. Just require few lines of code and i am sure this will give a better understanding towards TF-IDF calculations, Let’s move ahead.

Create a dataframe as df for the above docs,

df = pd.DataFrame({‘docs’: [‘the cat see the mouse’,  
 “the house has a tiny little mouse”,  
 ‘the mouse ran away from the house’,  
 ‘the cat finally ate the mouse’,  
 ‘the end of the mouse story’  
 ]})print(df)

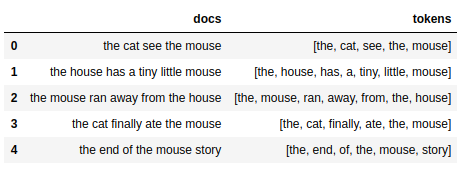
output:



Make an extra column called tokens contains a list of words formed my tokenizing each document in the above corpus.

df[‘tokens’] = [x.lower().split() for x in df.docs.values]   
print(df)

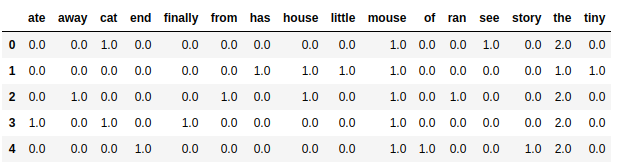
output:



Now, to calculate the Term Frequency apply an anonymous function on the above dataframe columntokens so that it determine count of each word in a row for each rows. fill nan values with 0 and at last sort resultant dataframe by column name using sort\_index() function as mention in below code.

tf = df.tokens.apply(lambda x: pd.Series(x).value\_counts()).fillna(0)   
tf.sort\_index(inplace=True, axis=1)  
# 'a' feature doesn't seems good feature so, removing it from tf   
tf.drop('a', axis=1, inplace=True)  
tf

output:



Now, we are going to apply same formula for IDF calculation refer in [Fig: 1.2](https://cdn-images-1.medium.com/max/800/1*nmxX1R2E-YSaozmCwMQyJg.jpeg) i.e,

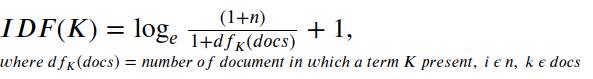
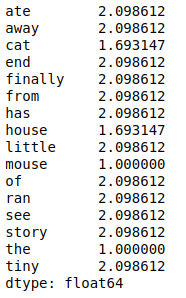


Fig: 1.5

Refer the following code for implementation of above formula,

import numpy as np  
idf = pd.Series([np.log((float(df.shape[0])+1)/(len([x for x in df.tokens.values if token in x])+1))+1 for token in tf.columns])  
idf.index = tf.columns  
print(idf)

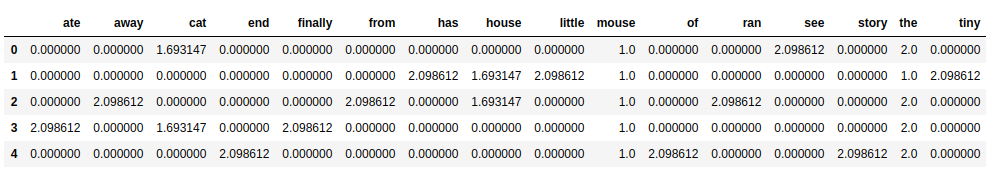
output:



To calculate TF-IDF simply multiply above tf dataframe and idf, so Let’s see the below code and final result.

tfidf = tf.copy()  
for col in tfidf.columns:  
 tfidf[col] = tfidf[col]\*idf[col]  
   
print(tfidf)

output:

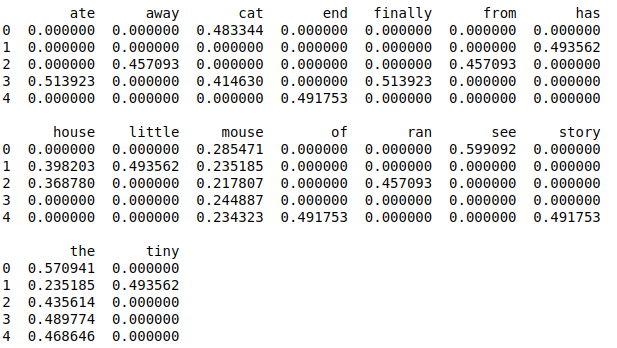


Now, We have to normalize the above result using euclidean norm

1. refer [Fig: 1.3](https://cdn-images-1.medium.com/max/800/1*yYuR79ejxnVZKY5Ft6Bimg.png) so, to implement this we are going to apply numpy.sqrt() on sum of square of each element for a row in the above tfidf dataframe to get sqrt\_vec.
2. Use Floating division of dataframe [DataFrame.div](https://pandas.pydata.org/pandas-docs/version/0.24.2/reference/api/pandas.DataFrame.div.html) to divide tfidf with sqrt\_vec on index.

sqrt\_vec = np.sqrt(tfidf.pow(2).sum(axis=1))  
tfidf.div(sqrt\_vec, axis=0)

output:



If you see the output of tfidf using sklearn library in [Fig: 1.3](https://cdn-images-1.medium.com/max/800/1*-bpwLYs6Bduo3390pntx7g.png) and the above output both are same. This is how the way sklearn finds normalized TF-IDF feature values from given corpus of textual data.

In Real world Scenario there is huge lots of raw unstructured text data, So to process those data we can use Tf-IDF feature engineering technique from sklearn library to convert raw text data to a machine readable format.

You can refer a real world use-case on text data on my GitHub link below.

[shubhamchouksey/Consumer-Complaint-Classification](https://github.com/shubhamchouksey/Consumer-Complaint-Classification" \t "_blank)

[Problem: Each week the Consumer Financial Protection Bureau sends thousands of consumer's complaints about financial…](https://github.com/shubhamchouksey/Consumer-Complaint-Classification" \t "_blank)

[github.com](https://github.com/shubhamchouksey/Consumer-Complaint-Classification" \t "_blank)

Reference:

[TF(Term Frequency)-IDF(Inverse Document Frequency) from scratch in python .](https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558" \t "_blank)

[Creating TF-IDF Model from Scratch](https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558" \t "_blank)

[towardsdatascience.com](https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558" \t "_blank)

I hope you found this article useful! Leave a comment below if you have any questions.

*You can connect me on*[***LinkedIn***](https://www.linkedin.com/in/insvk/)*or by following me on*[***GitHub***](https://github.com/shubhamchouksey)

[Analytics Vidhya](https://medium.com/analytics-vidhya?source=post_sidebar--------------------------post_sidebar-----------)

Analytics Vidhya is a community of Analytics and Data…

Follow

Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! [Take a look.](https://medium.com/analytics-vidhya/newsletters/analytics-vidhya-news-bytes?source=newsletter_v3_promo--------------------------newsletter_v3_promo-----------)

