Mukesh Chaudhary Follow
Apr 24, 2020 · 6 min read · Listen



# TF-IDF Vectorizer scikit-learn

**Deep understanding *tf-idf* calculation by various examples, Why is so efficiency than other vectorizer algorithm.**


TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction. Let's take sample example and explore two different spicy sparse matrix before go into deep explanation . It gives overall view what i am trying to explain below .Simple basic example data :

```
Train Document Set:
d1: The sky is blue.
d2: The sun is bright.
Test Document Set:
d3: The sun in the sky is bright.
```

```
# TfidfVectorizer
# CountVectorizer

from sklearn.feature_extraction.text import
TfidfVectorizer,CountVectorizer
import pandas as pd

# set of documents

train = ['The sky is blue.','The sun is bright.']
test = ['The sun in the sky is bright', 'We can see the shining sun,
the bright sun.']

# instantiate the vectorizer object

countvectorizer = CountVectorizer(analyzer= 'word',
stop_words='english')
tfidfvectorizer = TfidfVectorizer(analyzer='word',stop_words=
'english')

# convert th documents into a matrix

count_wm = countvectorizer.fit_transform(train)
tfidf_wm = tfidfvectorizer.fit_transform(train)

#retrieve the terms found in the corpora
# if we take same parameters on both Classes(CountVectorizer and
TfidfVectorizer) , it will give same output of get_feature_names()
methods)

#count_tokens = tfidfvectorizer.get_feature_names() # no difference
count_tokens = countvectorizer.get_feature_names()
tfidf_tokens = tfidfvectorizer.get_feature_names()

df_countvect = pd.DataFrame(data = count_wm.toarray(),index =
['Doc1','Doc2'],columns = count_tokens)
df_tfidfvect = pd.DataFrame(data = tfidf_wm.toarray(),index =
['Doc1','Doc2'],columns = tfidf_tokens)

print("Count Vectorizer\n")
print(df_countvect)
print("\nTD-IDF Vectorizer\n")
print(df_tfidfvect)
```



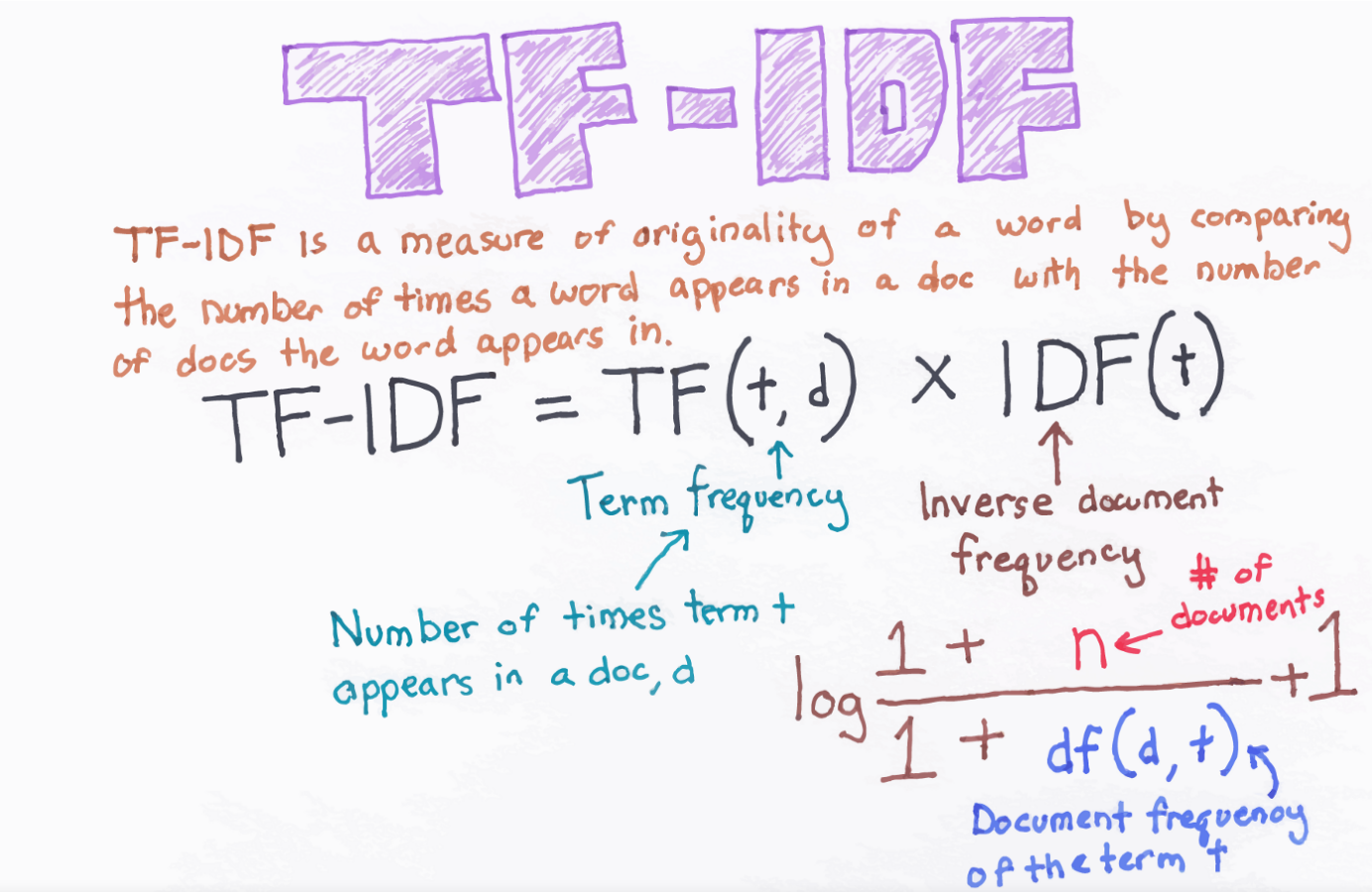
Output:

```
Doc1     1       0    1    0
Doc2     0       1    0    1

TD-IDF Vectorizer

          blue    bright       sky       sun
Doc1  0.707107  0.000000  0.707107  0.000000
Doc2  0.000000  0.707107  0.000000  0.707107
```

spicy sparse matrix of count and tf-idf vectorizer

Here , we can see clearly that Count Vectorizer give number of frequency with respect to index of vocabulary where as *tf-idf* consider overall documents of weight of words.This is my main purpose to explain in this blog post.

Let's try to understand step by step. I got one picture from internet showing summary of mathematical meaning of TF-IDF. I think it is useful to understand little bit behind mathematic concept.



. I believe that we can handle parameters of TF-IDF Vectorizer with better way if we understand core concept of TF-IDF functionality. let's take again above same example data :

```
Train Document Set:
d1: The sky is blue.
d2: The sun is bright.
Test Document Set:
d3: The sun in the sky is bright.
d4: We can see the shining sun, the bright sun.
```

Now , We are creating index vocabulary(dictionary) of the words of the train documents set, using the documents *d1* and *d2* from document set.

$$\mathrm{E}(t) = \begin{cases} 0, & \text{if } t \text{ is "blue"} \\ 1, & \text{if } t \text{ is "bright"} \\ 2, & \text{if } t \text{ is "sky"} \\ 3, & \text{if } t \text{ is "sun"} \end{cases}$$

Here index vocabulary is denoted by *E(t)* where the *t* is the term. Note that the terms like "is" , "the" are ignored because there are stop words which is repeating frequently and give less information.

Now , We can convert the test document set into a vector space where each term of vector is indexed as our index vocabulary. Example first term of the vector represents "blue" term of our vocabulary. the second term represents "sun" and so on. Now we are going to use term — frequency which means more than a measure of how many times the terms present in our vocabulary ( *E(t)*). We can define the term-frequency as counting function:

$x \in d$

where the $\mathrm{fr}(x, t)$ is a simple function defined as:

$$\mathrm{fr}(x, t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

Here the *tf(t,d)* returns is how many times is the term *t* present in document *d*. Example *tf("sun",d4)* could be 2.

When we represent d3 and d4 of test document set as vectors:

$$\vec{v_{d_3}} = (\mathrm{tf}(t_1, d_3), \mathrm{tf}(t_2, d_3), \mathrm{tf}(t_3, d_3), \ldots, \mathrm{tf}(t_n, d_3))$$
$$\vec{v_{d_4}} = (\mathrm{tf}(t_1, d_4), \mathrm{tf}(t_2, d_4), \mathrm{tf}(t_3, d_4), \ldots, \mathrm{tf}(t_n, d_4))$$

which evaluates to:

$$\vec{v_{d_3}} = (0, 1, 1, 1)$$
$$\vec{v_{d_4}} = (0,1,0,2)$$

As you can see, since the documents $d_3$ and $d_4$ are:

```
d3: The sun in the sky is bright.
d4: We can see the shining sun, the bright sun.
```

Here , example "sun" item occurs 2 time on vectors *Vd4* and so on . We can represent them as matrix with |D| * F shape where |D| is the cardinality of the document space.

$$M_{|D| \times F} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

```
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer

train = ('The sky is blue.','The sun is bright.')

test = ('The sun in the sky is bright', 'We can see the shining sun,
the bright sun.')

# instantiate the vectorizer object
# use analyzer is word and stop_words is english which are responsible
for remove stop words and create word vocabulary

countvectorizer = CountVectorizer(analyzer='word' ,
stop_words='english')

terms = countvectorizer.fit_transform(train)
term_vectors  = countvectorizer.transform(test)

print("Sparse Matrix form of test data : \n")
print(term_vectors.todense())
```

Output:

```
Sparse Matrix form of test data :

matrix([[0, 1, 1, 1],
        [0, 1, 0, 2]])
```

### The term frequency — inverse document frequency(tf-idf) weight

We saw above how to calculate term frequency . Now let's come to i*df(inverse document frequency)* topic that how it is calculate and multiplication with tf (term frequency) .
The *idf* is defined :

$$\text{idf}(t) = \log \frac{|D|}{1 + |\{d : t \in d\}|}$$

where $|\{d : t \in d\}|$ is the **number of documents** where the term $t$ appears, when the term-frequency function satisfies $\text{tf}(t, d) \neq 0$, we're only adding 1 into the formula to avoid zero-division.

This formula has an importance consequence that a high weight of the *tf-idf* calculation is reached when we have a high term frequency(*tf*) in the given document(*local parameter*) and a low document frequency of the term in the whole collection ( *global parameter*).

We have calculated matrix of test data above and have 4 features like " *blue,bright,sky,sum*" , we have to calculated *idf(t) :*

```
idf vector= (2.09861229 1. 1.40546511 1.)

matrix form of idf =
[[2.09,0,0,0],
 [0,1,0,0],
 [0,0,1.40,0],
 [0,0,0,1]]
```

After that we calculated *tf-idf (t) by multiplication of tf(t,d) * idf(t) like:*

*matrix [[0 ,1,1,1],[0, 1,0,2]] * matrix form idf*

After normalization of result of *tf-idf* is actually *tf-idf* sparse matrix form:

Let's see by python code:

```
# Tranfer  sparse matrix of Countvectorizer to tf-idf by
# using TfidfTransformer

from sklearn.feature_extraction.text import TfidfTransformer

tfidf = TfidfTransformer(norm='l2')

term_vectors.todense()

#[0, 1, 1, 1]
# [0, 1, 0, 2]

tfidf.fit(term_vectors)
```


```
print(tf_idf_matrix.todense())
```

Output:

```
Vector of idf

[2.09861229 1.         1.40546511 1.        ]

Final tf-idf vectorizer matrix form by TfidfTransformer :

[[0.         0.50154891 0.70490949 0.50154891]
 [0.         0.4472136  0.         0.89442719]]
```

Here we can understand how to calculate *TfidfVectorizer* by using *CountVectorizer* and *TfidfTransformer* in sklearn module in python and we also understood by mathematical concept.

Now we can get both functionality like *CountVectorizer* , *TfidfTransformer in TfidfVectorizer* . We can customize all parameters which have the above both classes. Let's see by python code :

```
from sklearn.feature_extraction.text import TfidfVectorizer

train = ('The sky is blue.','The sun is bright.')

test = ('The sun in the sky is bright', 'We can see the shining sun,
the bright sun.')

# instantiate the vectorizer object
# use analyzer is word and stop_words is english which are responsible
for remove stop words and create word vocabulary

tfidfvectorizer = TfidfVectorizer(analyzer='word' ,
stop_words='english',)

tfidfvectorizer.fit(train)
tfidf_train = tfidfvectorizer.transform(train)
tfidf_term_vectors  = tfidfvectorizer.transform(test)

print("Sparse Matrix form of test data : \n")
```


```
Sparse Matrix form of test data by TfidfVectorizer:

matrix([[0.        , 0.57735027, 0.57735027, 0.57735027],
        [0.        , 0.4472136 , 0.        , 0.89442719]])
```

Here , we can see that both outputs are almost same. As beginner, i think we should be careful at least three parameters like **analyzer, stop_words , ngram_range** because this is responsible of size of matrix . In real world data , we know that data is very huge . So we have to manipulate parameters carefully.

**Conclusion:**

I tried to explain how to work *tf-idf* calculation by using *CountVectorizer* and *Tfidftranformer*. I tried to explain mathematical concept behind the all process. In **TfidfVectorizer** we consider **overall document weightage** of a word. It helps us in dealing with most frequent words. Using it we can penalize them. TfidfVectorizer weights the word counts by a measure of how often they appear in the documents.

References :

For Photo and Pictures:

https://machinelearningflashcards.com/