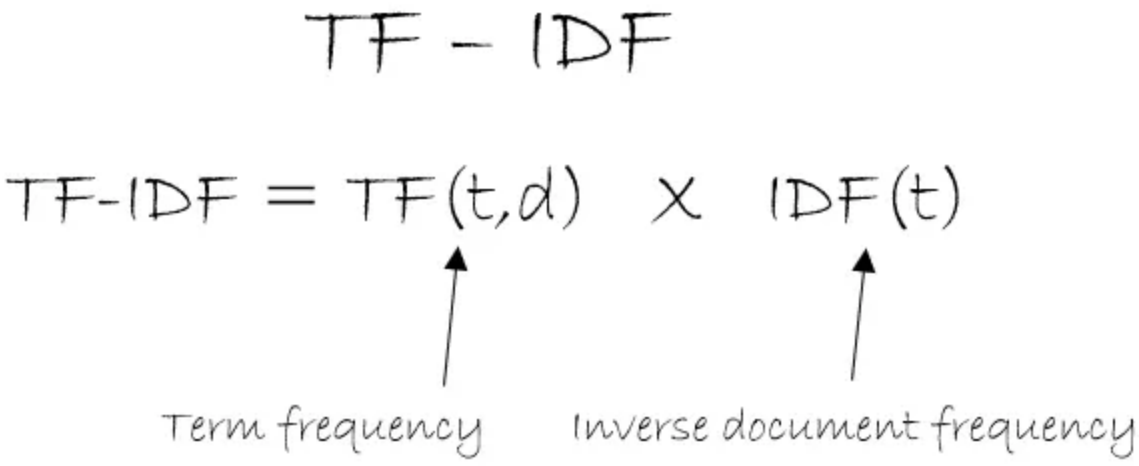
TF-IDF in Depth Understanding

Implementation in Python

Annamalai Swamy · Follow
5 min read · Dec 14, 2023

TF-IDF (Term Frequency — Inverse Document Frequency) is a statistical measure for text mining, NLP, Machine Learning. It is a measure of importance of a word / term within a document relative to a collection of documents (a.k.s corpus)

TF-IDF is type of text vectorization process where a words or terms within a document are transformed into importance numbers. TF-IDF scores of a word is calculated by multiplying the words Term Frequency (TF) and Inverse Document Frequency (IDF).



TF (Term Frequency):

Term Frequency [tf(t,d)] is a relative frequency of a term(t) of interest within a document(d)

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

Term Frequency Formula

There are multiple ways of calculating term frequency:

- Raw count -> Count of word appears in the document
- Boolean Frequency -> tf(t,d) = 1 if word(t) occurs in document (d) else 0
- Logarithmically Scaled -> tf(t,d) = log(1 + raw count)
- Augmented frequency -> To prevent bias towards longer documents
- Term frequency
- Double Normalization

IDF (Inverse Document Frequency)

IDF [idf(t,D)] is a measure of how much information the term(t) provides across all documents(D)

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Inverse Document Frequency Formula

There are multiple ways of calculating term frequency:

weighting scheme	idf weight ($n_t = |\{d \in D : t \in d\}|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(\frac{N}{1+n_t}\right)+1$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N-n_t}{n_t}$

Libraries to calculate Tf-idf

sklearn library has inbuilt classes like Tfidfvectorizer, TfidfTransformer, CountVectorizer to calculate tfidf:

CountVectorizer — Converts a collection of text documents to a matrix of token counts

TfidfVectorizer — Convert a collection of raw documents to a matrix of TF-IDF features

TfidfTransformer — Transform a count matrix to a normalized tf-idf representation

TfidfVectorizer Vs. TfidfTransformer Vs. CountVectorizer

TfidfVectorizer is used on sentences / documents, whereas **TfidfTransformer** is used on existing count matrix such as one returned by **CountVectorizer**

With **TfidfTransformer**, word counts are first computed using **CountVectorizer**, and then **Inverse Document Frequency** (IDF) and **Tf-idf** sources are computed using **TfidfTransformer**

With **TfidfVectorizer** all three values are computed at once. This class computes the word count, IDF, Tf-idf scores in single call.

Implementation of CountVectorizer in Python

Steps:

1. Make each of the words in corpus to lower
2. Remove the stopwords in corpus
3. Make a unique wordlist (union of words in particular document)
4. Count the frequency of each of words occurrence in particular document

```
import numpy as np
import pandas as pd
# import nltk
```

```
# nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, ENGLISH_STOP_WORDS

def CountVectorizer_IL(corpus, stop_words='english'):
    vector = pd.DataFrame().astype(int)
    for line in corpus:
        wordlist = set()
        term_counter = dict()

        # split each line to words
        # make it lower
        # remove the stopwords
        # make the union of wordlist
        wordlist = wordlist.union(set([word for word in line.lower().split()
                                       if word not in ENGLISH_STOP_WORDS]))

        # count the frequency of each word occurence
        for word in wordlist:
            if(word in term_counter.keys()):
                term_counter[word] += line.count(word)
            else:
                term_counter[word] = line.count(word)
        vector = vector._append(term_counter,ignore_index=True).fillna(0)
        vector = vector.convert_dtypes()
        vector = vector[sorted(vector.columns)]
    return vector

corpus = ['this is the first document',
          'this document is the second document',
          'and this is the third one',
          'is this the first ']

cv = CountVectorizer(stop_words='english')
df_cv = pd.DataFrame(data=cv_terms.toarray(),
                     columns=cv.get_feature_names_out())
print("CountVectorizer")
display(df_cv)

print("Custom CountVectorizer")
display(CountVectorizer_IL(corpus))
```

Output:

CountVectorizer

	document	second
0	1	0
1	2	1
2	0	0
3	0	0

Custom CountVectorizer

	document	second
0	1	0
1	2	1
2	0	0
3	0	0

Output of dataset using sklearn and custom CountVectorizer

Implementation of TfidVectorizer in Python

Steps

1. Follow Steps mentioned in CountVectorizer for text preprocessing and building the unique wordlist
2. Count the word frequency (raw count) of each word in the document
3. Calculate the Inverse document frequency (IDF) using formula by either enabling / disabling smooth_idf, setting Normalization to ‘l1’, ‘l2’ or ‘none’

```
import numpy as np
import pandas as pd
# import nltk
# nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS

# split each line to words
# make it lower
# remove the stopwords
# make the union of wordlist
def TextPreProcessing(corpus,stop_words='english'):
    # change the sentence to lower case
    wordlist = set()
    for line in corpus:
        wordlist = wordlist.union(set([word for word in line.lower().split()
                                       if word not in ENGLISH_STOP_WORDS]))
    return wordlist

# count the frequency (raw count) of each word occurence
def calculateTF(corpus,wordlist,stop_words='english'):
    vector = pd.DataFrame().astype(int)
    for line in corpus:
        term_counter = dict()
        for word in wordlist:
            if(word in term_counter.keys()):
                term_counter[word] += line.count(word)
            else:
                term_counter[word] = line.count(word)
        vector = vector._append(term_counter,ignore_index=True).fillna(0)
        vector = vector.convert_dtypes()
        vector = vector[sorted(vector.columns)]
    return vector


# calculate IDF for each word using following :
#     idf = log(1+ No. of documents)/(1 + '1' if word is present
#                                     else '0') + 1

#smooth_idf -
#      Purpose of enabling smooth_idf to true, is to add '1' to document
# frequencies, as if an extra document contains every term in collection
# exactly onces. It mainly prevents zero division

def calculateIDF(corpus,tf,wordlist,smooth_idf=True,norm=None):
    vector = pd.DataFrame().astype(int)
    term_counter = dict()
    _idf = int(smooth_idf);
```

```
        for word in wordlist:
            term_counter[word] = np.log((_idf+len(corpus))/(len(tf[tf[word]>0]) + _i

        vector = vector._append(term_counter,ignore_index=True).fillna(0)
        vector = vector.convert_dtypes()
        vector = vector[sorted(vector.columns)]
        return vector

    # tfidf = tf * idf
    def calculateTFIDF(tf,idf):
        return pd.DataFrame(tf.values * idf.values, columns=tf.columns)

    def TfidfVectorizer_custom(corpus,smooth_idf=True,norm=None,stop_words='english'
        wordlist = TextPreProcessing(corpus,stop_words)
        tf = calculateTF(corpus,wordlist,stop_words)
        idf = calculateIDF(corpus,tf,wordlist,smooth_idf,norm)
        tfidf = calculateTFIDF(tf,idf)
        return tfidf


    corpus = ['this is the first document',
              'this document is the second document',
              'and this is the third one',
              'is this the first document']

    vectorizer = TfidfVectorizer(smooth_idf=False,norm=None,stop_words='english')
    vec_model = vectorizer.fit_transform(corpus)

    print("TfidfVectorizer")
    display(pd.DataFrame(vec_model.toarray(),columns=vectorizer.get_feature_names_ou

    print("Custom TfidfVectorizer")
    display(TfidfVectorizer_custom(corpus,smooth_idf=False))
```

Output:

TfidfVectorizer

	document	second
0	1.287682	0.000000
1	2.575364	2.386294
2	0.000000	0.000000
3	1.287682	0.000000

Custom TfidfVectorizer

	document	second
0	1.287682	0.0
1	2.575364	2.386294
2	0.0	0.0
3	1.287682	0.0

Output of dataset using sklearn and custom TfidfVectorizer

Implementation of TfidTransformer in Python

Steps

1. Follow Steps mentioned in CountVectorizer for text preprocessing and building the unique wordlist
2. Follow the steps mentioned for TfidfVectorizer by passing wordlist obtained from above step (step 1) to calculate the Inverse document frequency (IDF) using formula by either enabling / disabling smooth_idf, setting Normalization to ‘l1’, ‘l2’ or ‘none’

Please Note that TfidfVectorizer is equivalent to CountVectorizer followed by TfidfTransformer

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer

corpus = ['this is the first document',
          'this document is the second document',
          'and this is the third one',
          'is this the first document']


cntvec = CountVectorizer(stop_words='english')
words = cntvec.fit_transform(corpus)

# print(cntvec.get_feature_names_out())
vectorizer = TfidfTransformer(smooth_idf=False,norm=None)
vec_model =  vectorizer.fit_transform(words)

print("TfidfTransformer")
display(pd.DataFrame(vec_model.toarray(),columns=cntvec.get_feature_names_out()))

print("TfidfTransformer_custom")
tf = CountVectorizer_IL(corpus,stop_words='english')
idf = calculateIDF(corpus,tf,tf.columns,smooth_idf=False,norm=None)
display(calculateTFIDF(tf,idf))
```

Output:

TfidfTransformer

	document	second
0	1.287682	0.000000
1	2.575364	2.386294
2	0.000000	0.000000
3	1.287682	0.000000

TfidfTransformer_custom

	document	second
0	1.287682	0.0
1	2.575364	2.386294
2	0.0	0.0
3	1.287682	0.0

Output of dataset using sklearn and custom TfidfTransformer

Conclusion:

I've explained the working details and concepts behind vectorizer and Transformer and how the Tf-idf calculation is done.

Reference:

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Written by Annamalai Swamy

5 Followers · 1 Following

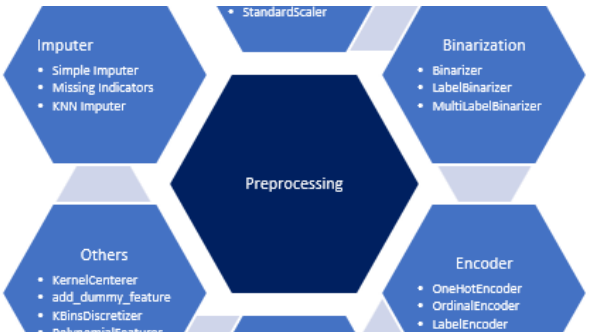
Follow

Senior Architect with Expertise in Automation / AI & ML

More from Annamalai Swamy

	Col1	Col2
0	D1	D1
1	D2	D2
2	D3	D3
3	D4	D4

```
Index  RangeIndex(start=0, stop=4, step=1)
```



Annamalai Swamy

Index in Pandas DataFrame

With Python Samples.

Nov 29, 2023

Annamalai Swamy

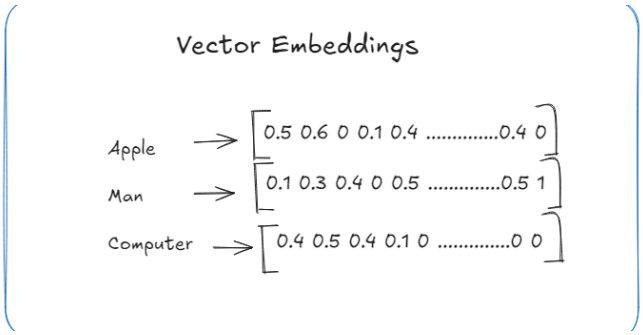
Data Preprocessing

Quick Overview

Jan 8

See all from Annamalai Swamy

Recommended from Medium



In Towards AI by Mdabdullahalhasib

A Complete Guide to Embedding For NLP & Generative AI/LLM

Understand the concept of vector embedding, why it is needed, and...

Oct 18 110

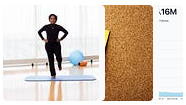
In Level Up Coding by Shrinivasan Sankar

TF-IDF and BM25 for RAG— a complete guide

TF-IDF and BM25 are commonly used techniques in information retrieval. while TF...

Oct 7 43

Lists



Staff picks

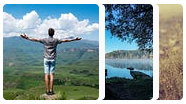
772 stories · 1459 saves

Stories to Help You Level-Up at Work

19 stories · 875 saves

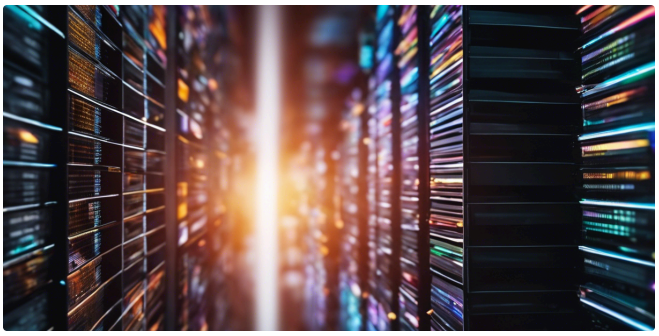
Self-Improvement 101

20 stories · 3073 saves



Productivity 101

20 stories · 2588 saves



In Towards Data Science by Chien Vu

Muneeb S. Ahmad

BM25S—Efficacy improvement of BM25 algorithm in document...

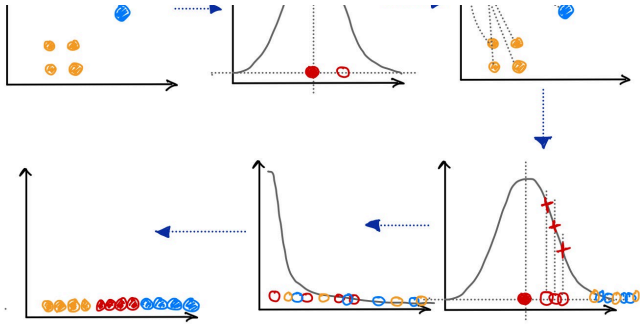
bm25s, an implementation of the BM25 algorithm in Python, utilizes Scipy and helps...

Aug 12 184

Mastering NLP with GloVe Embeddings: Word Similarity,...

Introduction

Oct 21



Rishabh Singh

Clustering Text Data with K-Means and Visualizing with t-SNE

In NLP, analyzing and grouping text data into meaningful clusters is a vital task. Clustering...

Oct 11

Henok Solomon

TF-IDF

A statistical measurement used in text mining and information retrieval to evaluate...

Nov 15 30

See more recommendations