Count Vectorizer vs TFIDF Vectorizer | Natural Language Processing

Sheel Saket

5 articles

Follow

Data Science Senior Manager | LLM + MLOps | AWS | Azure ML | Fin Tech | Conversational AI...

January 13, 2020

Open Immersive Reader

Follow Geeky Dude AI for more AI related content.

One of the major challenges that any NLP Data Scientist faces is to choose the best possible numerical/vectorial representation of the text strings for running Machine Learning models. In a general scenario, we work on a bunch of text information that may or may not be tagged and we then want to build a ML model that can understand the pattern based on the words present in the strings to predict a new text data.

For example, we know that "Its raining heavily outside" and "Its pouring down outside" mean the same thing. But how to make the computer understand it. As far as we know, computers can only

understand numerical data, while natural language data, for computers, are just text strings without any numerical or statistical information.

Read the original article at my personal blog site:

So how to bridge the gap? How to make computers understand text data?

By converting texts to numbers and also conserving linguistic information for analysis.

In this article i am going to discuss about 2 different ways of converting Text to Numbers for analysis.

1. Count Vectorizers:

Count Vectorizer is a way to convert a given set of strings into a frequency representation.

Lets take this example:

```
Text1 = “Natural Language Processing is a subfield of AI”
tag1 = "NLP"

Text2 = “Computer Vision is a subfield of AI”
tag2 = "CV"
```

In the above two examples you have Texts that are Tagged respectively. This is a very simple case of NLP where you get tagged text data set and then using it you have to predict the tag of another text data.

The above two texts can be converted into count frequency using the CountVectorizer function of sklearn library:

```
from sklearn.feature_extraction.text import CountVectorizer as CV
import pandas as pd
cv = CV()
cv.fit([Text1, Text2])
x= cv.transform([Text1]).toarray()
y= cv.transform([Text2]).toarray()
columns = cv.get_feature_names()
df1 = pd.DataFrame(x, columns= columns, index= ["Text1"])
df2 = pd.DataFrame(y, columns= columns, index= ["Text2"])
df = pd.concat([df1,df2])
df["tag"] = ["NLP", "CV"]
df
```

	ai	computer	is	language	natural	of	processing	subfield	vision	tag
Text1	1	0	1	1	1	1	1	1	0	NLP
Text2	1	1	1	0	0	1	0	1	1	CV

Since Text1 doesn't contain words "computer" and "vision", hence their frequencies are calculated as 0 while other words are present once hence their frequencies are equal to 1.

This is, in a nutshell, how we use Count Vectorizer!

Count Vectors can be helpful in understanding the type of text by the frequency of words in it. But its major disadvantages are:

- Its inability in identifying more important and less important words for analysis.
- It will just consider words that are abundant in a corpus as the most statistically significant word.
- It also doesn't identify the relationships between words such as linguistic similarity between words.

1. TF-IDF:

TF-IDF means Term Frequency - Inverse Document Frequency. This is a statistic that is based on the frequency of a word in the corpus but it also provides a numerical representation of how important a word is for statistical analysis.

TF-IDF is better than Count Vectorizers because it not only focuses on the frequency of words present in the corpus but also provides the importance of the words. We can then remove the words that are less important for analysis, hence making the model building less complex by reducing the input dimensions.

This is how tf-idf is calculated:

TFIDF

For a term i in document j:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

The term "tf" is basically the count of a word in a sentence. for example, in the above two examples for Text1, the tf value of the word "subfield" will be 1.

the term "df" is called document frequency which means in how many documents the word "subfield" is present within corpus. In our case the corpus consists of Text1 and Text2 (N value is 2) and the word "subfield" is present in both. Hence its df value is 2.

Also since in the formula df is present in the denominator of log (N/df) its called inverse document frequency. Hence the name tf-idf.

Here is how we calculate tfidf for a corpus:

Text1 = “Natural Language Processing is a subfield of AI”
tag1 = "NLP"

Text2 = “Computer Vision is a subfield of AI”
tag2 = "CV"

```
from sklearn.feature_extraction.text import TfidfVectorizer as tf_idf
import pandas as pd
tfidf = tf_idf(norm = None)
tfidf.fit([Text1, Text2])
x= tfidf.transform([Text1]).toarray()
y= tfidf.transform([Text2]).toarray()
columns = tfidf.get_feature_names()
df1 = pd.DataFrame(x, columns= columns, index= ["Text1"])
df2 = pd.DataFrame(y, columns= columns, index= ["Text2"])
df = pd.concat([df1,df2])
df["tag"] = ["NLP", "CV"]
df
```

	ai	computer	is	language	natural	of	processing	subfield	vision	tag
Text1	1.0	0.000000	1.0	1.405465	1.405465	1.0	1.405465	1.0	0.000000	NLP
Text2	1.0	1.405465	1.0	0.000000	0.000000	1.0	0.000000	1.0	1.405465	CV

Here sklearn calculated the idf value as:

$$idf = \ln[(1+N)/(1+df)]+1$$

For Text1 the calculation is:

In our example since there are only 2 texts/documents in the corpus, hence N=2

N-=2	Text1			
	tf	df	idf	tfidf
ai	1	2	1	1
computer	0	1	1.405465	0
is	1	2	1	1
language	1	1	1.405465	1.405465
natural	1	1	1.405465	1.405465
of	1	2	1	1
processing	1	1	1.405465	1.405465
subfield	1	2	1	1
vision	0	1	1.405465	0

Importance of Words:

TFIDF is based on the logic that words that are too abundant in a corpus and words that are too rare are both not statistically important for finding a pattern. The Logarithmic factor in tfidf mathematically penalizes the words that are too abundant or too rare in the corpus by giving them low tfidf scores.

Higher value of tfidf signifies higher importance of the words in the corpus while lower values represent lower importance. In the above example the word "AI" is present in both the sentences while words "Natural" and "Computer" are present only in one sentences each. Hence the tfidf value of "AI" is lower than the other two. While for the word "Natural" there are more words in Text1 hence its importance is lower than "Computer" since there are less number of words in Text2.

Even though TFIDF can provide a good understanding about the importance of words but just like Count Vectors, its disadvantage is:

- It fails to provide linguistic information about the words such as the real meaning of the words, similarity with other words etc.

To train a model on the actual linguistic relationship of the words, there are two other word embedding techniques widely used in NLP, they are "word2vec" and "Glove". I will discuss about these two in another article.

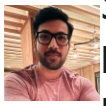
Let me know how you like this article.

Follow Geeky Dude AI for more AI related content.

Happy Coding.

Report this

Published by



Sheel Saket
Data Science Senior Manager | LLM + MLOps | A...
Published • 4y

5 articles

Follow

Here is my new article on the comparison between Count Vectorizer and TF-IDF Vectorizer. This article focusses on understanding the basics of converting words to numbers during text preprocessing phase of NLP.