CountVectorizer and TfidfVectorizer

CountVectorizer and TfidfVectorizer are both methods used for converting textual data into numerical feature vectors, which can be used as input for machine learning algorithms.

```
CountVectorizer:

*   CountVectorizer converts a collection of text documents into a matrix of token counts.
*   It works by tokenizing the documents (splitting them into individual words or terms), and then building a vocabulary of known words.
*   It converts each document into a vector of term frequencies (counts of each word in the document).
*   The resulting matrix is a sparse matrix where each row represents a document and each column represents a unique word in the entire corpus.
*   CountVectorizer is simple and efficient but does not consider the relative importance of words in the documents
```

```
TfidfVectorizer:

*   TfidfVectorizer stands for "Term Frequency-Inverse Document Frequency Vectorizer".
*   It converts a collection of raw documents to a matrix of TF-IDF features.
*   TF-IDF reflects the importance of a word in a document relative to the entire corpus.
*   TF (Term Frequency) measures the frequency of a word in a document.
*   IDF (Inverse Document Frequency) measures how important a word is across multiple documents. Words that occur frequently across documents
    get a lower weight.
*   The resulting matrix contains TF-IDF values for each word in each document.
*   TfidfVectorizer is particularly useful when dealing with large corpora, as it helps to downweight common words occurring in many documents.
```

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

#sample document
documents = [
    "This is first document",
    "This is second document",
    "this is third document",
    "Finally forth"
]
```

```
#Initialize CountVectorizer
count_vectorizer = CountVectorizer()

#fit and transform the documents using CountVectorizer
count_matrix = count_vectorizer.fit_transform(documents)

#Get the features name
count_features_names = count_vectorizer.get_feature_names_out()

#print the count mattix into array
print("The count matrix into array")
print(count_matrix.toarray())
print("Feature names:")
print(count_features_names)
print()
```

```
The count matrix into array
[[1 0 1 0 1 0 0 1]
 [1 0 0 0 1 1 0 1]
 [1 0 0 0 1 0 1 1]
 [0 1 0 1 0 0 0 0]]
Feature names:
['document' 'finally' 'first' 'forth' 'is' 'second' 'third' 'this']
```

```
# Convert count matrix to DataFrame
count_df = pd.DataFrame(count_matrix.toarray(), columns=count_features_names)

# Print count matrix in DataFrame format
print("Count Vectorizer Output (Matrix Format):")
print(count_df)
```

```
Count Vectorizer Output (Matrix Format):
   document  finally  first  forth  is  second  third  this
0         1        0      1      0   1       0      0     1
1         1        0      0      0   1       1      0     1
2         1        0      0      0   1       0      1     1
3         0        1      0      1   0       0      0     0
```

```
#Initialize TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the documents using TfidfVectorizer
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

# Get the feature names
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()

# Print TF-IDF matrix
print("TF-IDF Vectorizer Output:")
print(tfidf_matrix.toarray())
print("Feature names:")
print(tfidf_feature_names)
```

```
TF-IDF Vectorizer Output:
[[0.42817512 0.         0.67081906 0.         0.42817512 0.
  0.         0.42817512]
 [0.42817512 0.         0.         0.         0.42817512 0.67081906
  0.         0.42817512]
 [0.42817512 0.         0.         0.         0.42817512 0.
  0.67081906 0.42817512]
 [0.         0.70710678 0.         0.70710678 0.         0.
  0.         0.        ]]
Feature names:
['document' 'finally' 'first' 'forth' 'is' 'second' 'third' 'this']
```

```
# Convert count matrix to DataFrame
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=count_features_names)

# Print count matrix in DataFrame format
print("Count Vectorizer Output (Matrix Format):")
print(tfidf_df)
```

```
Count Vectorizer Output (Matrix Format):
   document   finally     first     forth        is    second     third  \
0  0.428175  0.000000  0.670819  0.000000  0.428175  0.000000  0.000000
1  0.428175  0.000000  0.000000  0.000000  0.428175  0.670819  0.000000
2  0.428175  0.000000  0.000000  0.000000  0.428175  0.000000  0.670819
3  0.000000  0.707107  0.000000  0.707107  0.000000  0.000000  0.000000

       this
0  0.428175
1  0.428175
2  0.428175
3  0.000000
```

Conclusion

In conclusion, the demonstrations of CountVectorizer and TfidfVectorizer showcase two fundamental approaches to converting textual data into numerical representations for machine learning tasks. CountVectorizer efficiently transforms text documents into a matrix of token counts, providing a straightforward method for representing textual data numerically. It is well-suited for tasks where the frequency of word occurrence in documents is relevant, such as text classification and clustering.

On the other hand, TfidfVectorizer offers a more nuanced representation by considering both the frequency of words in documents and their importance across the entire corpus through TF-IDF weighting. This approach enables TfidfVectorizer to downweight common words and highlight important ones, making it particularly useful for tasks where word importance in distinguishing between documents is crucial, such as information retrieval and document similarity. Overall, while CountVectorizer offers simplicity and speed, TfidfVectorizer provides a more sophisticated representation of text data, making the choice between them dependent on the specific requirements and objectives of the natural language processing task at hand.