Application of Deep Learning to Text and Image Data

Module 2, Lab 2: Using the BoW Method

This notebook will help you understand how to further process text data through *vectorization*. You will explore the bag-of-words (BoW) method to convert text data into numerical values, which will be used later for predictions with ML algorithms.

To convert text data to vectors of numbers, a vocabulary of known words (tokens) is extracted from the text. The occurrence of words is scored, and the resulting numerical values are saved in vocabulary-long vectors. A few versions of BoW exist with different word-scoring methods.

You will learn the following:

- How to use sklearn to process text in several ways
- When to use each method
- How to calculate BoW numerical values
- How to use binary classification, word counts, term frequency (TF), and term frequency-inverse document frequency (TF-IDF)

You will be presented with two kinds of exercises throughout the notebook: activities and challenges.

No coding is needed for an activity. You try to understand a concept, answer questions, or run a code cell.	Challenges are where you can practice your coding skills.

Index

Initial Setup

```
# Install libraries
!pip install -U -q -r requirements.txt

ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
autovizwidget 0.21.0 requires pandas<2.0.0,>=0.20.1, but you have
pandas 2.0.3 which is incompatible.
hdijupyterutils 0.21.0 requires pandas<2.0.0,>=0.17.1, but you have
pandas 2.0.3 which is incompatible.
sparkmagic 0.21.0 requires pandas<2.0.0,>=0.17.1, but you have pandas
2.0.3 which is incompatible.
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
autovizwidget 0.21.0 requires pandas<2.0.0,>=0.20.1, but you have
pandas 2.0.3 which is incompatible.
hdijupyterutils 0.21.0 requires pandas<2.0.0,>=0.17.1, but you have
pandas 2.0.3 which is incompatible.
sparkmagic 0.21.0 requires pandas<2.0.0,>=0.17.1, but you have pandas
2.0.3 which is incompatible.

import pandas as pd
import numpy as np

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

Binary classification

The first BoW method that you will use is *binary classification*. This method records whether a word is in a given sentence. You will also experiment with sklearn's vectorizers.

```
sentences = [
    "This document is the first document",
    "This document is the second document",
    "and this is the third one",
]

# Initialize the count vectorizer with the parameter binary=True
binary_vectorizer = CountVectorizer(binary=True)

# The fit_transform() function fits the text data and gets the binary
BoW vectors
x = binary_vectorizer.fit_transform(sentences)
```

As the vocabulary size grows, the BoW vectors get large. They usually have many zeros and few nonzero values. Sklearn stores these vectors in a compressed form. If you want to use them as NumPy arrays, call the `toarray()` function.

The following are the binary BoW features. Each row in the printed array corresponds to a single document binary encoded.

```
x.toarray()
```

```
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 1, 1, 0, 1, 1, 1]])
```

```
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 1, 1, 0, 1, 1, 1]])
```

To see what this array represents, check the vocabulary by using the `vocabulary` attribute. This returns a dictionary with each word as key and index as value. Notice that the indices are assigned in alphabetical order.

```
binary_vectorizer.vocabulary_
```

```
{'this': 8,
 'document': 1,
 'is': 3,
 'the': 6,
 'first': 2,
 'second': 5,
 'and': 0,
 'third': 7,
 'one': 4}
```

```
{'this': 8,
 'document': 1,
 'is': 3,
 'the': 6,
 'first': 2,
 'second': 5,
 'and': 0,
 'third': 7,
 'one': 4}
```

The `get_feature_names_out()` function displays similar information. The position of the terms in the output corresponds to the column position of the elements in the BoW matrix.

```
print(binary_vectorizer.get_feature_names_out())
```

```
['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
```

```
['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
```

But what does this data mean?

First, you created a list of three sentences. Each sentence contains six words.

Next, you created a vectorizer. This vectorizer collected all the words, ordered them alphabetically, and removed any duplicates.

You then converted the sentences to an array. The array has nine columns for each row. The nine columns correspond to the nine unique words from the sentences.

When you add column headers and identify the rows as sentences, as in the following table, you can see that the array tells you whether a word is included in the sentence. However, the array doesn't tell you how many times the word is used or where it appears in the sentence.

Number	Sentence	and	document	first	is	one	second	the	third	this
1	This document is the first d	no	yes	yes	yes	no	no	yes	no	yes

Number	Sentence	and	document	first	is	one	second	the	third	this
	ocument									
2	This document is the second docum	no	yes	no	yes	no	yes	yes	no	yes

Number	Sentence	and	document	first	is	one	second	the	third	this
3	ent and this is the third one	yes	no	no	yes	yes	no	yes	yes	yes

From this, you can compute how many sentences each word from the vocabulary appears in.

```
# Run this cell
sum_words = x.sum(axis=0)
words_freq = [
    (word, sum_words[0, idx])
    for (idx, word) in
enumerate(binary_vectorizer.get_feature_names_out())
]
words_freq
```

```
[('and', 1),
 ('document', 2),
 ('first', 1),
 ('is', 3),
 ('one', 1),
 ('second', 1),
 ('the', 3),
 ('third', 1),
 ('this', 3)]
```

```
[('and', 1),
 ('document', 2),
 ('first', 1),
 ('is', 3),
 ('one', 1),
 ('second', 1),
 ('the', 3),
 ('third', 1),
 ('this', 3)]
```

You can use the `binary_vectorizer` function to automatically create a table that shows the BoW vectors that are associated to each sentence.

```
df = pd.DataFrame(
    x.toarray(), columns=binary_vectorizer.get_feature_names_out(),
index=sentences
)
df
```

```
                                      and  document  first  is  one  
second  \
This document is the first document     0         1      1   1    0   
0   
This document is the second document    0         1      0   1    0   
1   
and this is the third one               1         0      0   1    1   
0   

                                      the  third  this  
This document is the first document     1      0     1  
This document is the second document    1      0     1  
and this is the third one               1      1     1  
```

```
                                      and  document  first  is  one  
second  \
This document is the first document     0         1      1   1    0   
0   
This document is the second document    0         1      0   1    0   
1   
and this is the third one               1         0      0   1    1
```

```
0

                                          the  third  this
This document is the first document         1      0     1
This document is the second document        1      0     1
and this is the third one                   1      1     1
```

How can you calculate BoW vectors for a new sentence?

You can use the `transform()` function. When you look at the results, notice that this doesn't change the vocabulary. New words are simply skipped.

```
new_sentence = ["This is the new sentence"]

new_vectors = binary_vectorizer.transform(new_sentence)

new_vectors.toarray()

array([[0, 0, 0, 1, 0, 0, 1, 0, 1]])
```

```
array([[0, 0, 0, 1, 0, 0, 1, 0, 1]])
```

```
df2 = pd.DataFrame(
    new_vectors.toarray(),
    columns=binary_vectorizer.get_feature_names_out(),
    index=new_sentence,
)
pd.concat([df, df2])

                                      and  document  first  is  one
second  \
This document is the first document     0         1      1   1    0
0
This document is the second document    0         1      0   1    0
1
and this is the third one               1         0      0   1    1
0
This is the new sentence                0         0      0   1    0
0

                                      the  third  this
This document is the first document     1      0     1
This document is the second document    1      0     1
and this is the third one               1      1     1
This is the new sentence                1      0     1
```

```
                                      and  document  first  is  one
second  \
This document is the first document     0         1      1   1    0
0
This document is the second document    0         1      0   1    0
```

```
1
and this is the third one               1         0      0   1    1
0
This is the new sentence                0         0      0   1    0
0

                                      the  third  this
This document is the first document     1      0     1
This document is the second document    1      0     1
and this is the third one               1      1     1
This is the new sentence                1      0     1
```

Notice that **new** and **sentence** aren't listed in the vocabulary, but the other words are listed correctly.

Word counts

You can calculate word counts by using the same `CountVectorizer()` function *without* the `binary` parameter.

```
sentences = [
    "This document is the first document",
    "This document is the second document",
    "and this is the third one",
]

# Initialize the count vectorizer
count_vectorizer = CountVectorizer()

xc = count_vectorizer.fit_transform(sentences)

xc.toarray()
```

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

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

```
df = pd.DataFrame(
    xc.toarray(), columns=binary_vectorizer.get_feature_names_out(),
index=sentences
)
df
```

```
                                      and  document  first  is  one  
second  \
This document is the first document     0         2      1   1    0   
0   
This document is the second document    0         2      0   1    0   
1   
and this is the third one               1         0      0   1    1   
0   

                                      the  third  this  
This document is the first document     1      0     1  
This document is the second document    1      0     1  
and this is the third one               1      1     1  

                                      and  document  first  is  one  
second  \
This document is the first document     0         2      1   1    0   
0   
This document is the second document    0         2      0   1    0   
1   
and this is the third one               1         0      0   1    1   
0   

                                      the  third  this  
This document is the first document     1      0     1  
This document is the second document    1      0     1  
and this is the third one               1      1     1  
```

```
new_sentence = ["This is the new sentence"]

############### CODE HERE ###############



############## END OF CODE ##############

df2 = pd.DataFrame(
    new_vectors.toarray(),
    columns=binary_vectorizer.get_feature_names_out(),
    index=new_sentence,
)
pd.concat([df, df2])
```

```
                                      and  document  first  is  one  
second  \
This document is the first document     0         2      1   1    0   
0   
This document is the second document    0         2      0   1    0   
1   
and this is the third one               1         0      0   1    1   
0   
```

```
This is the new sentence                0         0      0   1    0
0

                                      the  third  this
This document is the first document     1      0     1
This document is the second document    1      0     1
and this is the third one               1      1     1
This is the new sentence                1      0     1

                                      and  document  first  is  one
second  \
This document is the first document     0         2      1   1    0
0
This document is the second document    0         2      0   1    0
1
and this is the third one               1         0      0   1    1
0
This is the new sentence                0         0      0   1    0
0

                                      the  third  this
This document is the first document     1      0     1
This document is the second document    1      0     1
and this is the third one               1      1     1
This is the new sentence                1      0     1
```

Term frequency

Term frequency (TF) vectors show the importance of words in a document. These vectors are computed with the following formula:

$$tf(term, doc) = \frac{\text{Number of times that the term occurs in the doc}}{\text{Total number of terms in the doc}}$$

To calculate TF, you will use sklearn's `TfidfVectorizer` function with the parameter `use_idf=False`, which *automatically normalizes* the TF vectors by their Euclidean (L_2) norm.

```
tf_vectorizer = TfidfVectorizer(use_idf=False)

x = tf_vectorizer.fit_transform(sentences)

np.round(x.toarray(), 2)
```

```
array([[0.  , 0.71, 0.35, 0.35, 0.  , 0.  , 0.35, 0.  , 0.35],
       [0.  , 0.71, 0.  , 0.35, 0.  , 0.35, 0.35, 0.  , 0.35],
       [0.41, 0.  , 0.  , 0.41, 0.41, 0.  , 0.41, 0.41, 0.41]])
```

```
array([[0.  , 0.71, 0.35, 0.35, 0.  , 0.  , 0.35, 0.  , 0.35],
       [0.  , 0.71, 0.  , 0.35, 0.  , 0.35, 0.35, 0.  , 0.35],
       [0.41, 0.  , 0.  , 0.41, 0.41, 0.  , 0.41, 0.41, 0.41]])
```

```
new_sentence = ["This is the new sentence"]
new_vectors = tf_vectorizer.transform(new_sentence)
np.round(new_vectors.toarray(), 2)
```

```
array([[0.  , 0.  , 0.  , 0.58, 0.  , 0.  , 0.58, 0.  , 0.58]])
```

```
array([[0.  , 0.  , 0.  , 0.58, 0.  , 0.  , 0.58, 0.  , 0.58]])
```

```
df = pd.DataFrame(
    np.round(x.toarray(), 2),
columns=tf_vectorizer.get_feature_names_out(), index=sentences
)
df2 = pd.DataFrame(
    np.round(new_vectors.toarray(), 2),
    columns=tf_vectorizer.get_feature_names_out(),
    index=new_sentence,
)
pd.concat([df, df2])
```

```
                                       and  document  first    is  
one  \
This document is the first document   0.00      0.71   0.35  0.35   
0.00   
This document is the second document  0.00      0.71   0.00  0.35   
0.00   
and this is the third one             0.41      0.00   0.00  0.41   
0.41   
This is the new sentence              0.00      0.00   0.00  0.58   
0.00   

                                      second   the  third  this  
This document is the first document     0.00  0.35   0.00  0.35  
This document is the second document    0.35  0.35   0.00  0.35  
and this is the third one               0.00  0.41   0.41  0.41  
This is the new sentence                0.00  0.58   0.00  0.58  
```

Inverse document frequency

Inverse Document Frequency (IDF) is a weight indicating how commonly a word is used. The more frequent its usage across documents, the lower its score. The lower the score, the less important the word becomes.

It is computed with the following formula:

$$idf(term)=\ln\left(\frac{n_{documents}}{n_{documentscontainingtheterm}}\right)$$

Term frequency-inverse document frequency

Term frequency-inverse document frequency (TF-IDF) is computed by the following formula:

$$tf-idf(term,doc)=tf(term,doc)*idf(term)$$

Using sklearn, vectors are computed using the `TfidfVectorizer()` function with the parameter `use_idf=True`.

Note: You don't need to include the parameter because it is `True` by default.

```
tfidf_vectorizer = TfidfVectorizer(use_idf=True)

sentences = [
    "This document is the first document",
    "This document is the second document",
    "and this is the third one",
]

xf = tfidf_vectorizer.fit_transform(sentences)

np.round(xf.toarray(), 2)

new_sentence = ["This is the new sentence"]
new_vectors = tfidf_vectorizer.transform(new_sentence)
np.round(new_vectors.toarray(), 2)

df = pd.DataFrame(
    np.round(xf.toarray(), 2),
    columns=tfidf_vectorizer.get_feature_names_out(),
    index=sentences,
)
df2 = pd.DataFrame(
    np.round(new_vectors.toarray(), 2),
    columns=tfidf_vectorizer.get_feature_names_out(),
    index=new_sentence,
)
pd.concat([df, df2])
```

Note: In addition to automatically normalizing the TF vectors by their Euclidean (L_2) norm, sklearn also uses a *smoothed version of idf* and computes the following:

$$idf(term) = \ln\left(\frac{n_{documents}+1}{n_{documents containing the term}+1}\right)+1$$

```
np.round(tfidf_vectorizer.idf_, 2)
```

Notice that the IDF is larger for the less common terms.

Now you can generate the IDF DataFrame and TF DataFrame, and then concatenate them as one DataFrame.

```
df = pd.DataFrame(
    [[str(a) for a in np.round(tfidf_vectorizer.idf_, 2)]],
    columns=tfidf_vectorizer.get_feature_names_out(),
    index=["IDF"],
)
df2 = pd.DataFrame(
    [[str(w[1]) for w in words_freq]],
    columns=tfidf_vectorizer.get_feature_names_out(),
    index=["TF"],
)
pd.concat([df2, df])
```

This table shows that when the TF is large, the IDF is small.

Conclusion

In this notebook, you observed how the BoW method converts text data into numerical values.

Next lab

In the next lab, you will explore advanced word embeddings and the relationships between words.