Natural Language Processing

Text Similarity Measures

Text Similarity Measures

- Word Similarity
  - Levenshtein distance

- Document Similarity
  - Count vectorizer and the document-term matrix
  - Bag of words
  - Cosine similarity
  - Term frequency-inverse document frequency (TF-IDF)

Word Similarity

How similar are the following pairs of words?

MATH	**MATH**
MATH	**BATH**
MATH	**BAT**
MATH	**SMASH**

Word Similarity

Why is word similarity important? It can be used for the following:

- Spell check
- Speech recognition
- Plagiarism detection

What is a common way of quantifying word similarity?

- Levenshtein distance
- Also known as edit distance in computer science

Word Similarity

Levenshtein distance: Minimum number of operations to get from one word to another. Levenshtein operations are:

- Deletions: Delete a character
- Insertions: Insert a character
- Mutations: Change a character

Example: kitten —> sitting

- kitten —> **s**itten (1 letter change)
- sitten —> sittin (1 letter change)
- sittin —> sittin**g** (1 letter insertion)

Levenshtein distance = 3

Word Similarity

How similar are the following pairs of words?

MATH	**MATH**	Levenshtein distance = 0
MATH	**BATH**	Levenshtein distance = 1
MATH	**BAT**	Levenshtein distance = 2
MATH	**SMASH**	Levenshtein distance = 2

TextBlob

Another toolkit other than NLTK

- Wraps around NLTK and makes it easier to use

TextBlob capabilities

- Tokenization
- Parts of speech tagging
- Sentiment analysis
- Spell check
- … and more

TextBlob Demo: Tokenization

Input:

```
# Command line: pip install textblob
from textblob import TextBlob

my_text = TextBlob("We're moving from NLTK to TextBlob. How fun!")

my_text.words
```

Output:

```
WordList(['We', "'re", 'moving', 'from', 'NLTK', 'to', 'TextBlob', 'How', 'fun'])
```

TextBlob Demo: Spell Check

Input:

```
blob = TextBlob("I'm graat at speling.")

print(blob.correct())
```

Output:

```
I'm great at spelling.
```

How does the correct function work?

- Calculates the Levenshtein distance between the word ‘graat’ and all words in its word list
- Of the words with the smallest Levenshtein distance, it outputs the most popular word

Text Similarity Measures Checkpoint

- Word Similarity
  - Levenshtein distance

- Document Similarity
  - Count vectorizer and the document-term matrix
  - Bag of words
  - Cosine similarity
  - Term frequency-inverse document frequency (TF-IDF)

Text Similarity Measures Checkpoint

- Word Similarity
  - Levenshtein distance

- Document Similarity
  - Count vectorizer and the document-term matrix
  - Bag of words
  - Cosine similarity
  - Term frequency-inverse document frequency (TF-IDF)

Document Similarity

When is document similarity used?

- When sifting through a large number of documents and trying to find similar ones
- When trying to group, or cluster, together similar documents

To compare documents, the first step is to put them in a similar format so they can be compared

- Tokenization
- Count vectorizer and the document-term matrix

Text Format for Analysis

There are a few ways that text data can be put into a standard format for analysis

“This is an example”

Split Text Into Words

[‘This’,’is’,’an’,’example’]

Tokenization

Numerically Encode Words

This	[1,0,0,0]
is	[0,1,0,0]
an	[0,0,1,0]
example	[0,0,0,1]

One-Hot Encoding

Text Format for Analysis: Count Vectorizer

Input:

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

corpus = ['This is the first document.',
          'This is the second document.',
          'And the third one. One is fun.’]

cv = CountVectorizer()
X = cv.fit_transform(corpus)
pd.DataFrame(X.toarray(),columns=cv.get_feature_names())
```

A **Corpus** is a collection of texts

Output:

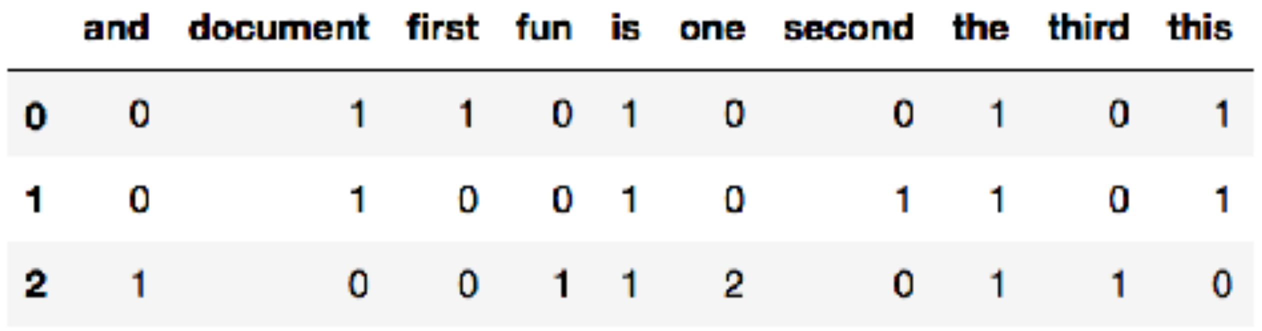
	and	document	first	fun	is	one	second	the	third	this
0	0	1	1	0	1	0	0	1	0	1
1	0	1	0	0	1	0	1	1	0	1
2	1	0	0	1	1	2	0	1	1	0

This is called a
Document-Term Matrix

Text Format for Analysis: Key Concepts

The Count Vectorizer helps us create a Document-Term Matrix

- Rows = documents
- Columns = terms



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

Bag of Words Model

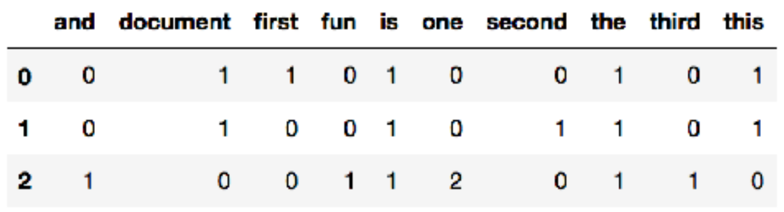
- Simplified representation of text, where each document is recognized as a bag of its words
- Grammar and word order are disregarded, but multiplicity is kept

Document Similarity Checkpoint

What was our original goal? Finding similar documents.

To compare documents, the first step is to put them in a similar format so they can be compared

- Tokenization
- Count vectorizer and the document-term matrix



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

The big assumption that we're making here is that each document is just a **Bag of Words**

Document Similarity: Cosine Similarity

Cosine Similarity is a way to quantify the similarity between documents

Step 1: Put each document in vector format

Step 2: Find the cosine of the angle between the documents

"I love you"

"I love NLP"

	i	love	you	nlp
Doc 1	1	1	1	0
Doc 2	1	1	0	1

a = [1, 1, 1, 0]
b = [1, 1, 0, 1]

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$$

= 0.667

Document Similarity: Cosine Similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$$

```
from numpy import dot

from numpy.linalg import norm


cosine = lambda v1, v2: dot(v1, v2) / (norm(v1) * norm(v2))


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

```
0.667
```

Document Similarity: Example

Here are five documents. Which ones seem most similar to you?

“The weather is hot under the sun”

“I make my hot chocolate with milk”

“One hot encoding”

“I will have a chai latte with milk”

“There is a hot sale today”

Let’s see which ones are most similar from a mathematical approach.

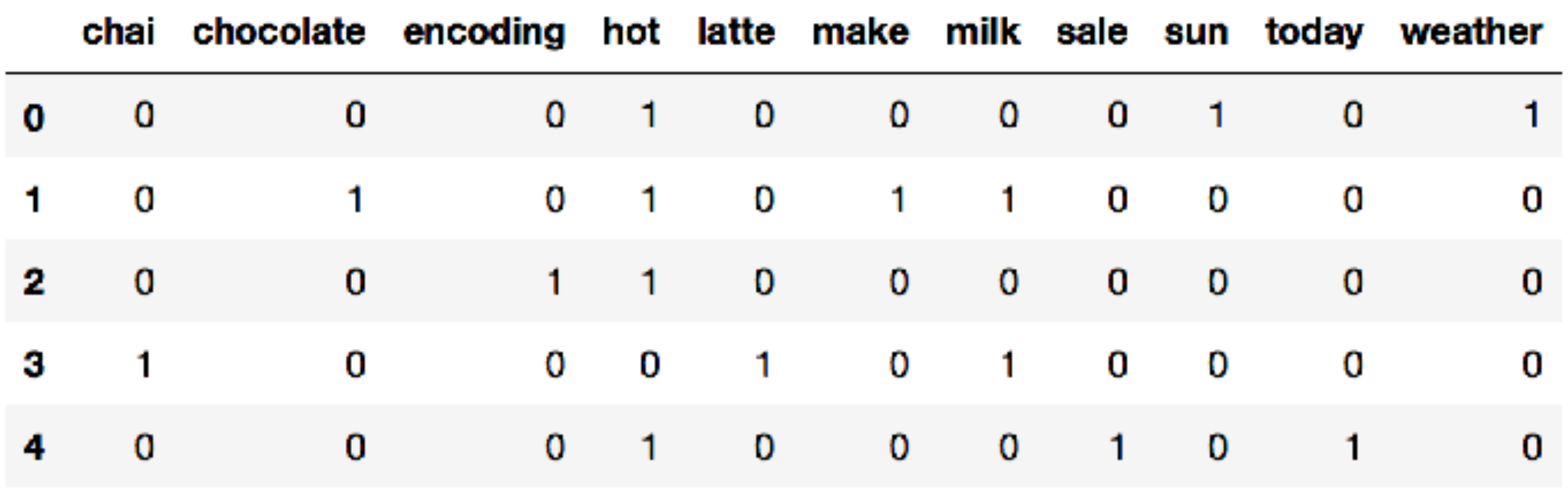
Document Similarity: Example

Input:

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

corpus = ['The weather is hot under the sun',
          'I make my hot chocolate with milk',
          'One hot encoding',
          'I will have a chai latte with milk',
          'There is a hot sale today']

# create the document-term matrix with count vectorizer
cv = CountVectorizer(stop_words="english")
X = cv.fit_transform(corpus).toarray()

dt = pd.DataFrame(X, columns=cv.get_feature_names())
dt
```

Document Similarity: Example

Output:



	chai	chocolate	encoding	hot	latte	make	milk	sale	sun	today	weather
0	0	0	0	1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	1	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0	0
3	1	0	0	0	1	0	1	0	0	0	0
4	0	0	0	1	0	0	0	1	0	1	0

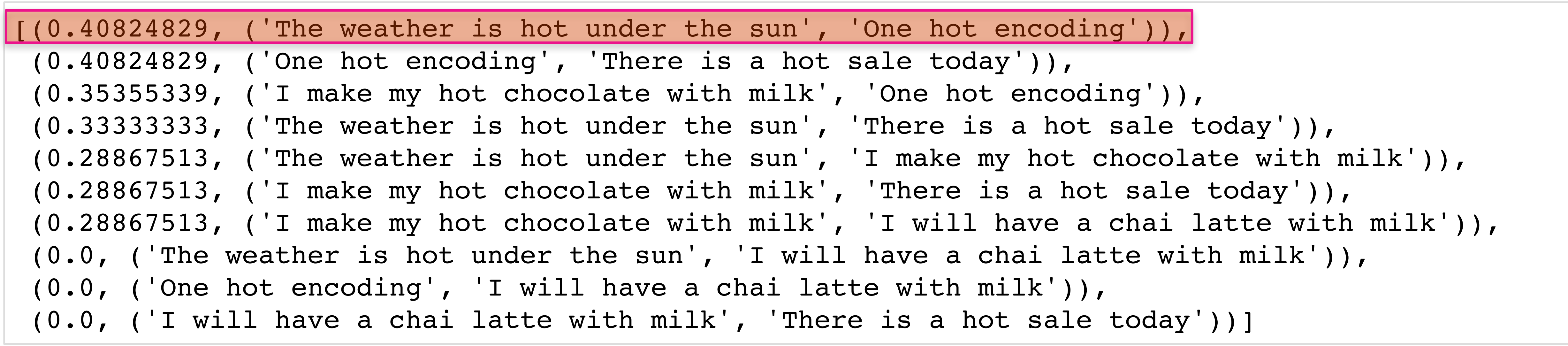
Document Similarity: Example

Input:

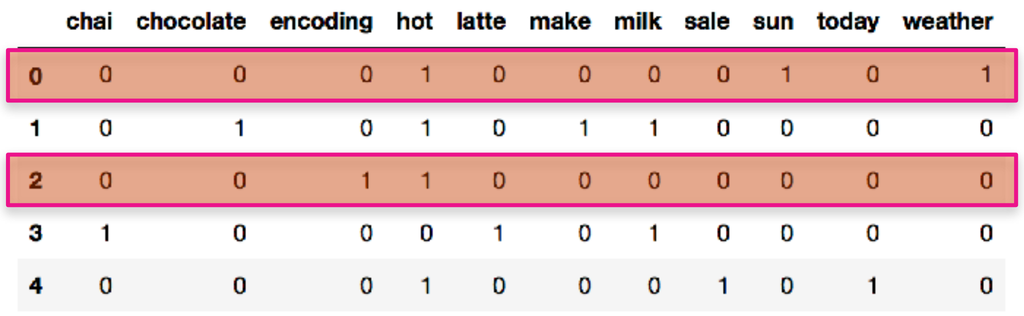
```
# calculate the cosine similarity between all combinations of documents
from itertools import combinations

# list all of the combinations of 5 take 2 as well as the pairs of phrases
pairs = list(combinations(enumerate(corpus),2))
combos = [(a[0], b[0]) for a, b in pairs]
phrases = [(a[1], b[1]) for a, b in pairs]

# calculate the cosine similarity for all pairs of phrases and sort by most similar
results = [cosine_similarity(dt.iloc[a], dt.iloc[b]) for a, b in combos]
sorted(zip(results, phrases), reverse=True)
```

Document Similarity: Example

Output:



```
[(0.40824829, ('The weather is hot under the sun', 'One hot encoding')),
 (0.40824829, ('One hot encoding', 'There is a hot sale today')),
 (0.35355339, ('I make my hot chocolate with milk', 'One hot encoding')),
 (0.33333333, ('The weather is hot under the sun', 'There is a hot sale today')),
 (0.28867513, ('The weather is hot under the sun', 'I make my hot chocolate with milk')),
 (0.28867513, ('I make my hot chocolate with milk', 'There is a hot sale today')),
 (0.28867513, ('I make my hot chocolate with milk', 'I will have a chai latte with milk')),
 (0.0, ('The weather is hot under the sun', 'I will have a chai latte with milk')),
 (0.0, ('One hot encoding', 'I will have a chai latte with milk')),
 (0.0, ('I will have a chai latte with milk', 'There is a hot sale today'))]
```



	chai	chocolate	encoding	hot	latte	make	milk	sale	sun	today	weather
0	0	0	0	1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	1	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0	0
3	1	0	0	0	1	0	1	0	0	0	0
4	0	0	0	1	0	0	0	1	0	1	0

- These two documents are most similar, but it’s just because the term “hot” is a popular word
- “Milk” seems to be a better differentiator, so how we can mathematically highlight that?

Document Similarity: Beyond Count Vectorizer

Downsides of Count Vectorizer

- Counts can be too simplistic
- High counts can dominate, especially for high frequency words or long documents
- Each word is treated equally, when some terms might be more important than others

We want a metric that accounts for these issues

- Introducing Term Frequency-Inverse Document Frequency (TF-IDF)

Term Frequency-Inverse Document Frequency

TF-IDF = (Term Frequency) * (Inverse Document Frequency)

Different value for every document / term combination

$$\frac{\text{Term Count in Document}}{\text{Total Terms in Document}}$$

$$\log\left(\frac{\text{Total Documents} \quad +1}{\text{Documents Containing the Term} \quad +1}\right)$$

Term Frequency-Inverse Document Frequency

Term Frequency

- So far, we've been recording the term (word) count

"This is an example"

This	is	an	example
1	1	1	1

- However, if there were two documents, one very long and one very short, it wouldn't be fair to compare them by word count alone
- A better way to compare them is by a normalized term frequency, which is (term count) / (total terms).

This	is	an	example
0.25	0.25	0.25	0.25

- There are many ways to do this. Another example is log(count +1)

Term Frequency-Inverse Document Frequency

Inverse Document Frequency

- Besides term frequency, another thing to consider is how common a word is among all the documents
- Rare words should get additional weight

The log dampens the effect of IDF

$$\log\left(\frac{\text{Total Documents} + 1}{\text{Documents Containing the Term} + 1}\right)$$

Want to make sure that the denominator is never 0

Term Frequency-Inverse Document Frequency

TF-IDF = (Term Frequency) * (Inverse Document Frequency)

Different value for every document / term combination

$$\frac{\text{Term Count in Document}}{\text{Total Terms in Document}}$$

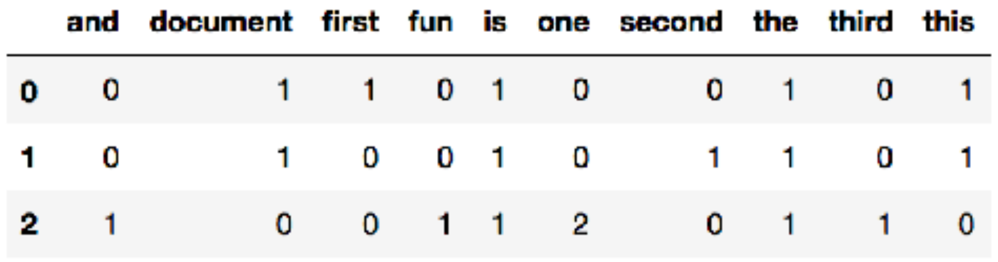
$$\log\left(\frac{\text{Total Documents} + 1}{\text{Documents Containing the Term} + 1}\right)$$

Count Vectorizer vs TF-IDF Vectorizer

```python
import pandas as pd
corpus = ['This is the first document.',
          'This is the second document.',
          'And the third one. One is fun.’]

# original Count Vectorizer
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(corpus).toarray()
pd.DataFrame(X, columns=cv.get_feature_names())

# new TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
cv_tfidf = TfidfVectorizer()
X_tfidf = cv_tfidf.fit_transform(corpus).toarray()
pd.DataFrame(X_tfidf, columns=cv_tfidf.get_feature_names())
```

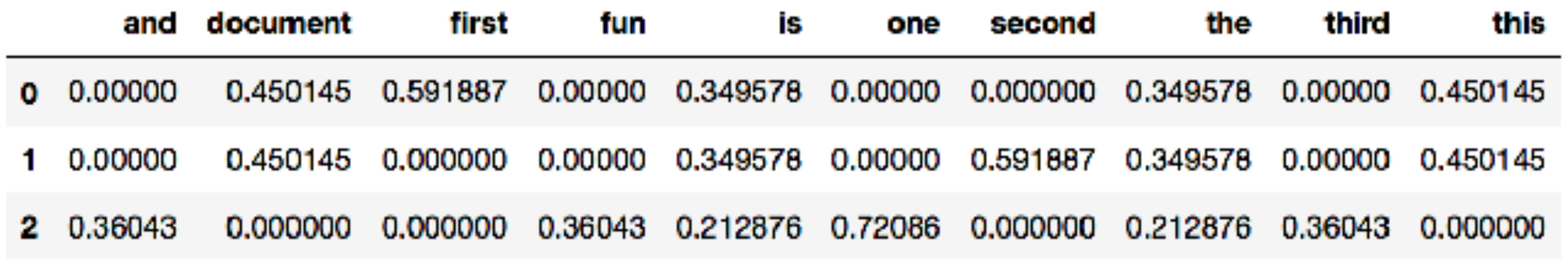
Count Vectorizer vs TF-IDF Vectorizer

Count Vectorizer Output:



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

TF-IDF Vectorizer Output:



	and	document	first	fun	is	one	second	the	third	this
0	0.00000	0.450145	0.591887	0.00000	0.349578	0.00000	0.000000	0.349578	0.00000	0.450145
1	0.00000	0.450145	0.000000	0.00000	0.349578	0.00000	0.591887	0.349578	0.00000	0.450145
2	0.36043	0.000000	0.000000	0.36043	0.212876	0.72086	0.000000	0.212876	0.36043	0.000000

Document Similarity: Example

Let’s go back to the problem we were originally trying to solve.

Here are five documents. Which ones seem most similar to you?

“The weather is hot under the sun”

“I make my hot chocolate with milk”

“One hot encoding”

“I will have a chai latte with milk”

“There is a hot sale today”

With Count Vectorizer, these two documents were the most similar

Document Similarity: Example with TF-IDF

Input:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# create the document-term matrix with TF-IDF vectorizer
cv_tfidf = TfidfVectorizer(stop_words="english")
X_tfidf = cv_tfidf.fit_transform(corpus).toarray()

dt_tfidf = pd.DataFrame(X_tfidf,columns=cv_tfidf.get_feature_names())
dt_tfidf
```

Output:

	chai	chocolate	encoding	hot	latte	make	milk	sale	sun	today	weather
0	0.000000	0.000000	0.000000	0.370036	0.000000	0.000000	0.000000	0.0000	0.6569	0.0000	0.6569
1	0.000000	0.580423	0.000000	0.327000	0.000000	0.580423	0.468282	0.0000	0.0000	0.0000	0.0000
2	0.000000	0.000000	0.871247	0.490845	0.000000	0.000000	0.000000	0.0000	0.0000	0.0000	0.0000
3	0.614189	0.000000	0.000000	0.000000	0.614189	0.000000	0.495524	0.0000	0.0000	0.0000	0.0000
4	0.000000	0.000000	0.000000	0.370036	0.000000	0.000000	0.000000	0.6569	0.0000	0.6569	0.0000

Document Similarity: Example with TF-IDF

```
# calculate the cosine similarity for all pairs of phrases and sort by most similar
results_tfidf = [cosine_similarity(dt_tfidf.iloc[a], dt_tfidf.iloc[b])
                 for a, b in combos]
sorted(zip(results_tfidf, phrases), reverse=True)
```

```
[(0.23204485, (‘I make my hot chocolate with milk', 'I will have a chai latte with milk')),
 (0.18165505, ('The weather is hot under the sun', 'One hot encoding')),
 (0.18165505, ('One hot encoding', 'There is a hot sale today')),
 (0.16050660, ('I make my hot chocolate with milk', 'One hot encoding')),
 (0.13696380, ('The weather is hot under the sun', 'There is a hot sale today')),
 (0.12101835, ('The weather is hot under the sun', 'I make my hot chocolate with milk')),
 (0.12101835, ('I make my hot chocolate with milk', 'There is a hot sale today')),
 (0.0, (‘The weather is hot under the sun', 'I will have a chai latte with milk')),
 (0.0, ('One hot encoding', 'I will have a chai latte with milk')),
 (0.0, ('I will have a chai latte with milk', 'There is a hot sale today'))]
```

By weighting “milk” (rare) > “hot” (popular), we get a smarter similarity score

Text Similarity Measures Summary

- Word Similarity
  - Levenshtein distance is a popular way to calculate word similarity
  - TextBlob, another NLP library, uses this concept for its spell check function
- Document Similarity
  - Cosine similarity is a popular way to calculate document similarity
  - To compare documents, they need to be put in document-term matrix form
  - The document-term matrix can be made using Count Vectorizer or TF-IDF Vectorizer