Brownbag Session

March 6, 2026
@muslih

Today, I realize something interesting:
The date is March 6, 2026

6, 2026 is a strangely beautiful number combination

It made me pause for a moment and notice the pattern in today’s date

Some numbers look beautiful because our brains recognize patterns

And interestingly, our brains do something similar when reading text

For example, when whe read two sentences, we can easily tell if the mean the same thing.

But how does a computer know whether two sentences are similar?

Let’s try a simple example

Sentence A

“The teacher asked the student to submit the assignment.”

Sentence B

“The teacher requested the student to tun in the homework.”

As a humans, we instantly weel that these sentences mean almost the same thing

So how can a machine understand that they are actually similar??

And how can it tell when two sentences are completely different??

This is where the concept of Text Similarity become important.

Text similarity try to measure how close two pieces of text are in meaning

Today i want to briefly share how computers measure similarity between texts.

Welcome to the world of text similarity!!!

Text similarity measures quantify how similar two pieces of text are.

They measure the similarity or the distance between two text strings.

Similarity can be based on words **(lexical similarity)**

Similarity can also be based on meaning **(semantic similarity)**

In this session, we focus on **lexical similarity**.

On the next session we will focus on **semantic similarity**

Now let’s see how computers actually measure text similarity.

Computers cannot understand text directly, so they convert text into numbers.

Text Similarity

1. **Word Similarity**
   a. Levensthein Distance
2. **Document Similarity**
   a. Count vecotrizer and the document matrix
   b. Bag of words
   c. Cosine similarity
   d. Term Frequency-inverse document frequency (TF-IDF)

1. Word Similarity

Common way of quantifying word similarity by using levensthein distance,

It also known as edit distance

CODE - CODE	**None | levensthein distance = 0**
CODE - NODE	**Substituce C -> N | levensthein distance = 1**
CODE - COD_	**Delete E | levensthein distance = 1**
CODE - DECODE	**Insert D, insert E | levensthein distance = 2**

1. Word Similarity

Levensthein distance: minimum number of operations to get from one to another.

And the operations are:

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

MATH - SMASH

MATH -> SMATH

1 insertion

MATH -> SMASH

1 mutation

Levensthein distance = 2

Demo

https://github.com/muslih/brownbag-6-march-2026

2. Document Similarity

a. Count vecotrizer and the document matrix
b. Bag of words
c. Cosine similarity
d. Term Frequency-inverse document frequency (TF-IDF)

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

- Tokenization
- Count vectorizer and document-term matrix

2. Document Similarity (tokenization)

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

We are educator team

Split text into words

['We','are','educator','team']

Tokenization

Encode words numerically

We	[1,0,0,0]
are	[0,1,0,0]
educator	[0,0,1,0]
Team	[0,0,0,1]

One-hot encoding

2. Document Similarity (count vectorizer)

Input

```ruby
corpus = [
 "Student starts the online exam",
 "Teacher monitors the exam session",
 "Student submits the exam answer"
]

tokenized_docs = corpus.map { |doc| tokenize(doc) }
vocab = tokenized_docs.flatten.uniq.sort
matrix = tokenized_docs.map do |tokens|
  vocab.map { |word| tokens.count(word) }
end
```

A corpus is collection of texts

Output

```
Vocabulary:
["answer", "exam", "monitors", "online"]
["session", "starts", "student", "submits", "teacher", "the"]

Document-Term Matrix:
Doc0: [0, 1, 0, 1, 0, 1, 1, 0, 0, 1]
Doc1: [0, 1, 1, 0, 1, 0, 0, 0, 1, 1]
Doc2: [1, 1, 0, 0, 0, 0, 1, 1, 0, 1]
```

This called a **Document-Term Matrix**

The count vectorizer help us to create a
Document-Term Matrix

2. Document Similarity(Bag of words Model)

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

What was our original goal? Finding similar documents.

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

- Tokenization
- Count vectorizer and the document term matrix

```
Vocabulary:
["answer", "exam", "monitors", "online"]
["session", "starts", "student", "submits", "teacher", "the"]

Document-Term Matrix:
Doc0: [0, 1, 0, 1, 0, 1, 1, 0, 0, 1]
Doc1: [0, 1, 1, 0, 1, 0, 0, 0, 1, 1]
Doc2: [1, 1, 0, 0, 0, 0, 1, 1, 0, 1]
```

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

2. Document Similarity(Cosine Similarity)

Cosine similarity measures how similar two documents are.

- Step 1: Convert each document into a vector representation.
- Step 2: Calculate the cosine of the angle between the two document vectors.

It measures the similarity between two non-zero vectors with the cosine of the angle between them.

"I love NTE"

"I love OTE"

	I	love	NTE	OTE
Doc 1	1	1	1	0
Doc 2	1	1	0	1

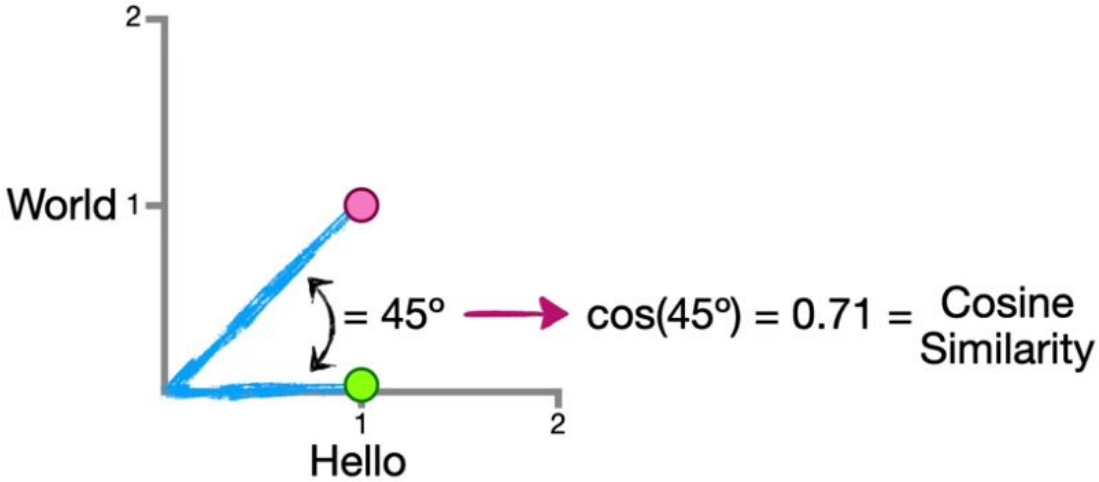
A = [1,1,1,0]

B = [1,1,0,1]

Here’s the visualizaiton


Hello
World
Hello, World!
1
1
Hello!
1
0

	Hello	World
Hello, World! →	1	1
Hello! →	1	0



Demo

https://github.com/muslih/brownbag-6-march-2026

```
STEP 3 - Vocabulary Index
0 -> weather
1 -> hot
2 -> sun
3 -> make
4 -> chocolate
5 -> milk
6 -> one
7 -> encoding
8 -> chai
9 -> latte
10 -> sale
11 -> today

STEP 4 - Document Term Matrix
doc_n | 0     | 1     | 2     | 3     | 4     | 5     | 6     | 7     | 8     | 9     | 10    | 11
doc_0 | 1     | 1     | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0
doc_1 | 0     | 1     | 0     | 1     | 1     | 1     | 0     | 0     | 0     | 0     | 0     | 0
doc_2 | 0     | 1     | 0     | 0     | 0     | 0     | 1     | 1     | 0     | 0     | 0     | 0
doc_3 | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 1     | 1     | 0     | 0
doc_4 | 0     | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 1

STEP 5 - Cosine Similarity (Sorted)
0.3333 - (The weather is hot under the sun, One hot encoding)
0.3333 - (The weather is hot under the sun, There is a hot sale today)
0.3333 - (One hot encoding, There is a hot sale today)
0.2887 - (The weather is hot under the sun, I make my hot chocolate with milk)
0.2887 - (I make my hot chocolate with milk, One hot encoding)
0.2887 - (I make my hot chocolate with milk, I will have a chai latte with milk)
0.2887 - (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)
```

- These three docs 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?

2. Document Similarity(Beyond Count Vectorizer)

Downside of Count Vectorizer

- Count can be too simplistic
- High count can dominate, especially for high frequency words or long
- Each word is treated equally, when some term might be more important than others

We want a metric that accos for these issue

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

2. Document Similarity (Term Frequency - Inverse Document Frequency)

Term Frequency

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

"We are educator team"

We	are	educator	team
1	1	1	1

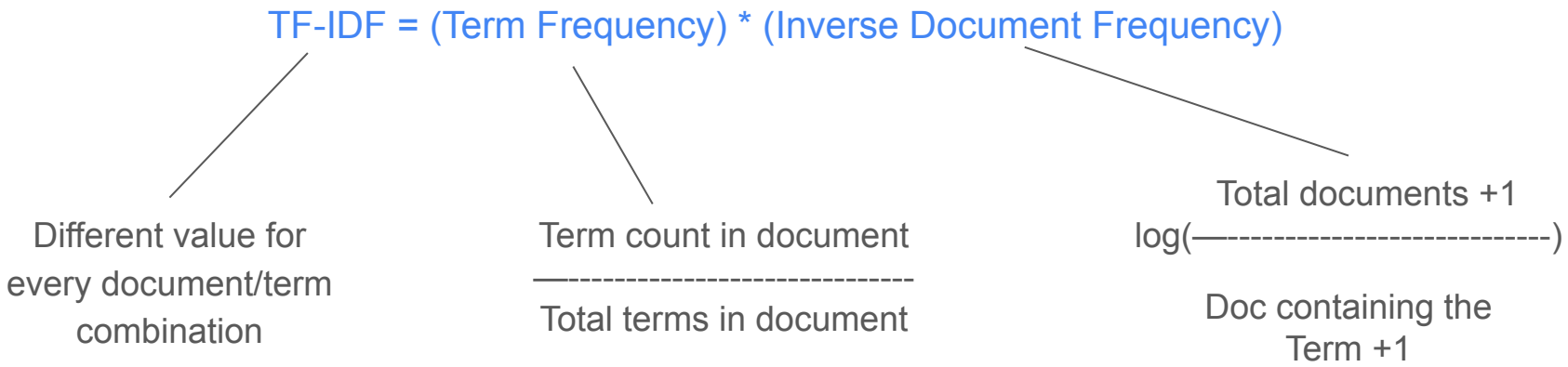
- However, if there where two document, one very long and once very short, it wouldn't be fair to compare them by word count alone
- A better way to compare them is by normalized term frequency, whic is (term count) / (total items)
- There are many ways to do this. Another example is log(count+1)

We	are	educator	team
0.25	0.25	0.25	0.25

2. Document Similarity (Term Frequency - Inverse Document Frequency)

Inverse Document Frequency

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



2. Document Similarity (Term Frequency - Inverse Document Frequency)

TF-IDF Intuition

- TF-IDF assign more weight to rare words and less weight to commonly occurring words.
- Tels us how frequent a words is in a document relative to its frequency in the entire corpus
- Tells us that two document are similar when they have more rare words in common

STEP 6 - TF-IDF Matrix

doc_n	0	1	2	3	4	5	6	7	8	9	10	11
doc_0	1.6094	0.2231	1.6094	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
doc_1	0.0	0.2231	0.0	1.6094	1.6094	0.9163	0.0	0.0	0.0	0.0	0.0	0.0
doc_2	0.0	0.2231	0.0	0.0	0.0	0.0	1.6094	1.6094	0.0	0.0	0.0	0.0
doc_3	0.0	0.0	0.0	0.0	0.0	0.9163	0.0	0.0	1.6094	1.6094	0.0	0.0
doc_4	0.0	0.2231	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.6094	1.6094

2. Document Similarity (Term Frequency - Inverse Document Frequency)

Lets back to the problem where we originally trying to solve. Here are five document. Which one seem most similar to us?

'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

2. Document Similarity (Term Frequency - Inverse Document Frequency)

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

```
STEP 7 - Cosine Similarity using TF-IDF
0.1389 - (I make my hot chocolate with milk, I will have a chai latte with milk)
0.0095 - (The weather is hot under the sun, One hot encoding)
0.0095 - (The weather is hot under the sun, There is a hot sale today)
0.0095 - (One hot encoding, There is a hot sale today)
0.0088 - (The weather is hot under the sun, I make my hot chocolate with milk)
0.0088 - (I make my hot chocolate with milk, One hot encoding)
0.0088 - (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)
```

Text Similarity Measurement Summary

World Similarity

- Word similarity can be measured in different ways. One common method is Levenshtein distance, which calculates how similar two words are by counting the number of edits needed to transform one word into another.

Document Similarity

- Document similarity is often measured using cosine similarity.
- To compare documents using this method, the text must first be converted into a document-term matrix.
- This matrix representation can be created using techniques such as Count Vectorizer or TF-IDF Vectorizer.

Text Similarity Measurement Summary

World Similarity

- Word similarity can be measured in different ways. One common method is Levenshtein distance, which calculates how similar two words are by counting the number of edits needed to transform one word into another.

Document Similarity

- Document similarity is often measured using cosine similarity.
- To compare documents using this method, the text must first be converted into a document-term matrix.
- This matrix representation can be created using techniques such as Count Vectorizer or TF-IDF Vectorizer.

But…

This method cannot be applied to our product at Quipper, right?

Of course…

Of course not!

We can introduce a new feature such as essay-based question answers.

For the next brownbag session

semantic similarity

I hope I'll still be alive

Thank you!!