Lecture 9
Textual Data: Vector Space Model and TF-IDF

1. Review

2. Vector Space Model

3. tf-idf (Term frequency-inverse document frequency)

1. Review
2. Vector Space Model
3. tf-idf

Textual Data

“Whoever has hate for his brother is in the darkness and walks in the darkness.”

“Hello darkness, my old friend.”

“Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that.”

Textual Data

“Whoever has hate for his brother is in the darkness and walks in the darkness.” —1 John 2:11

“Hello darkness, my old friend.”

“Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that.”

Textual Data

“Whoever has hate for his brother is in the darkness and walks in the darkness.” —1 John 2:11

“Hello darkness, my old friend.” —Simon & Garfunkel

“Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that.”

Textual Data

“Whoever has hate for his brother is in the darkness and walks in the darkness.” —1 John 2:11

“Hello darkness, my old friend.” —Simon & Garfunkel

“Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that.” —MLK

Textual Data

1. "Whoever has hate for his brother is in the darkness and walks in the darkness." —1 John 2:11
2. "Hello darkness, my old friend." —Simon & Garfunkel
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that." —MLK

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Textual Data

1. "Whoever has hate for his brother is in the darkness and walks in the darkness." —1 John 2:11
2. "Hello darkness, my old friend." —Simon & Garfunkel
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that." —MLK

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Which document is most similar to document 0?

Textual Data

1. "Whoever has hate for his brother is in the darkness and walks in the darkness." —1 John 2:11
2. "Hello darkness, my old friend." —Simon & Garfunkel
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that." —MLK

⇒

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Which document is most similar to document 0?

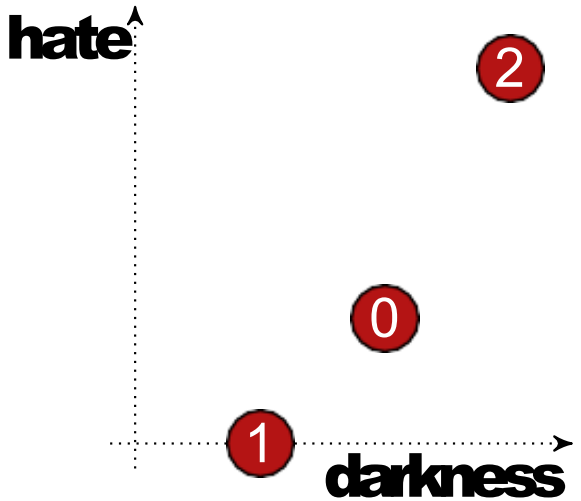
Textual Data

1. "Whoever has hate for his brother is in the darkness and walks in the darkness." —1 John 2:11
2. "Hello darkness, my old friend." —Simon & Garfunkel
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that." —MLK

⇒

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Which document is most similar to document 0?



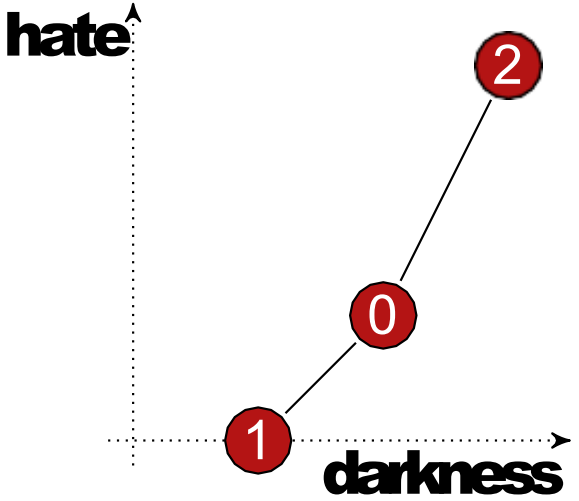
Textual Data

1. "Whoever has hate for his brother is in the darkness and walks in the darkness." —1 John 2:11
2. "Hello darkness, my old friend." —Simon & Garfunkel
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that." —MLK

⇒

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Which document is most similar to document 0?



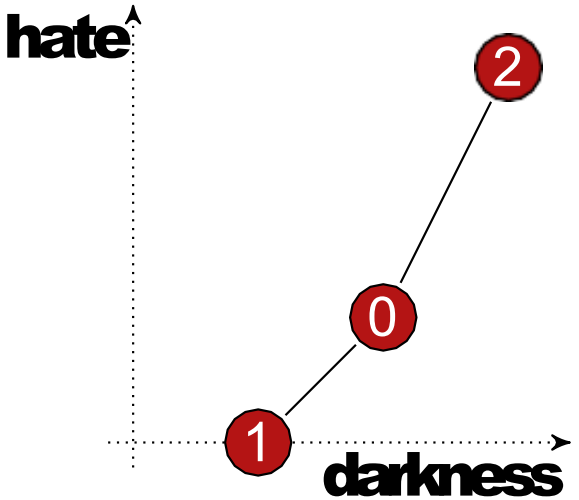
Textual Data

1. "Whoever has hate for his brother is in the darkness and walks in the darkness." —1 John 2:11
2. "Hello darkness, my old friend." —Simon & Garfunkel
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that." —MLK

⇒

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

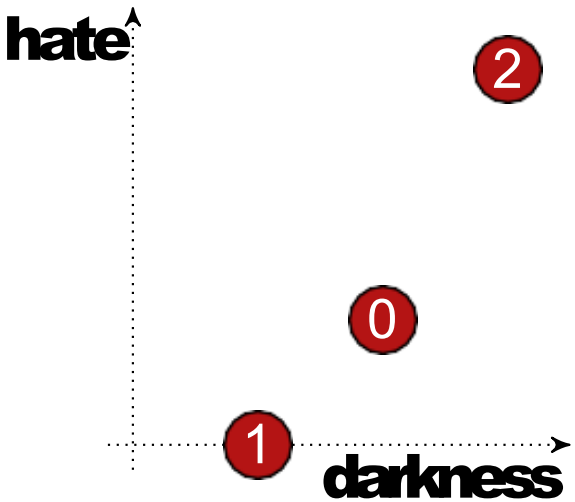
Which document is most similar to document 0?



Using Euclidean distance, document 1appears closer than document 2!

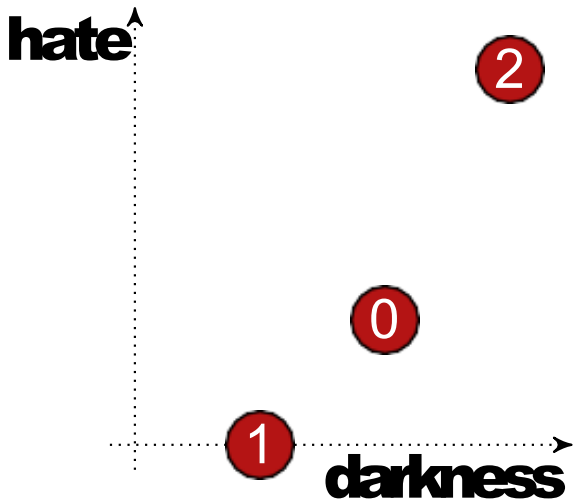
1. Review
2. Vector Space Model
3. tf-idf

Vector Space Model



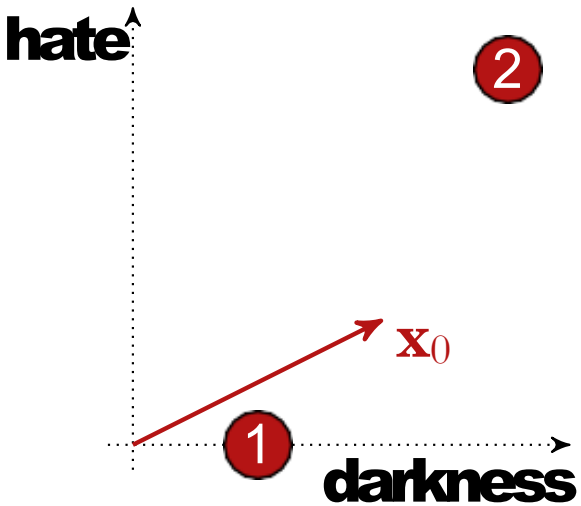
Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



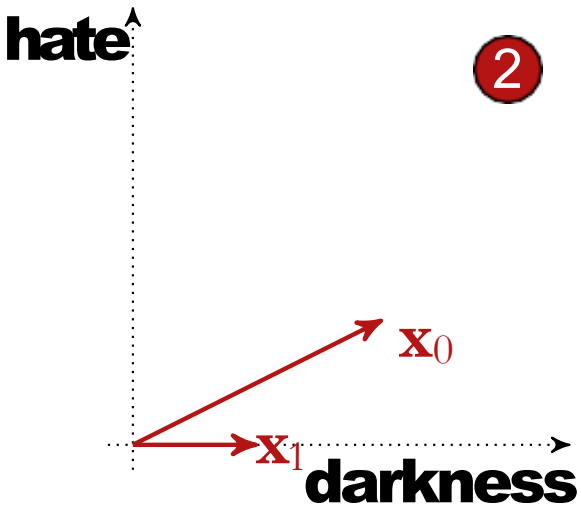
Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



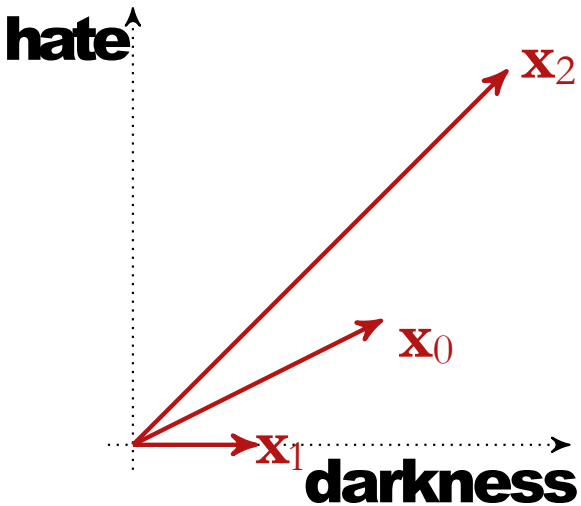
Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



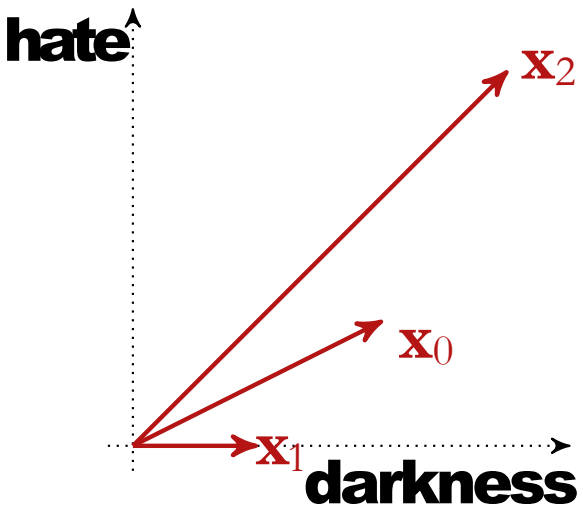
Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.

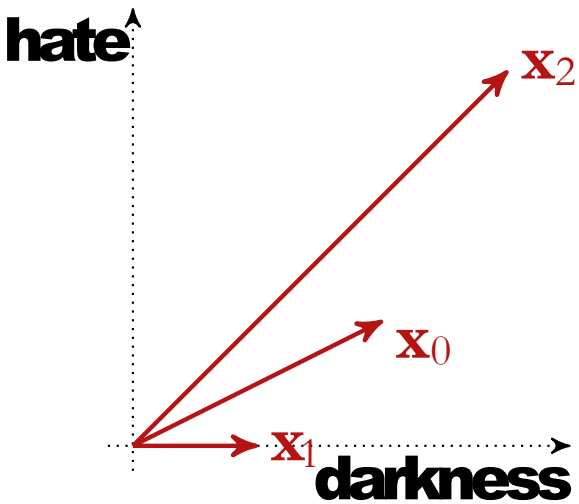


The **length of a vector** is its distance from the origin $\mathbf{0}$:

$$||\mathbf{v}|| =$$

Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.

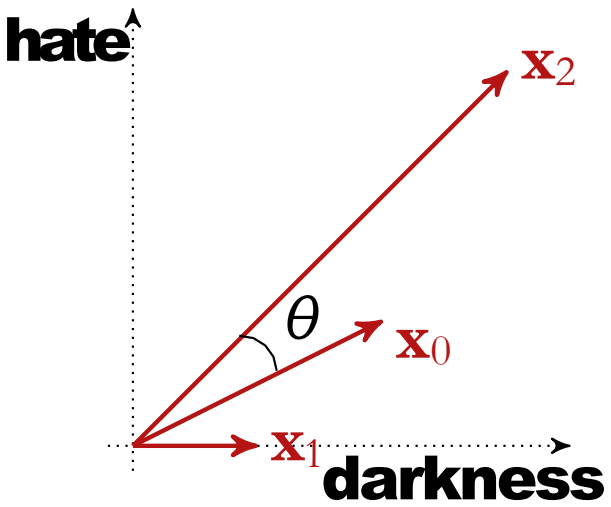


The **length of a vector** is its distance from the origin $\mathbf{0}$:

$$||\mathbf{v}|| = \sqrt{\sum_{j=1}^{|V|} v_j^2}.$$

Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



The **length of a vector** is its distance from the origin $\mathbf{0}$:

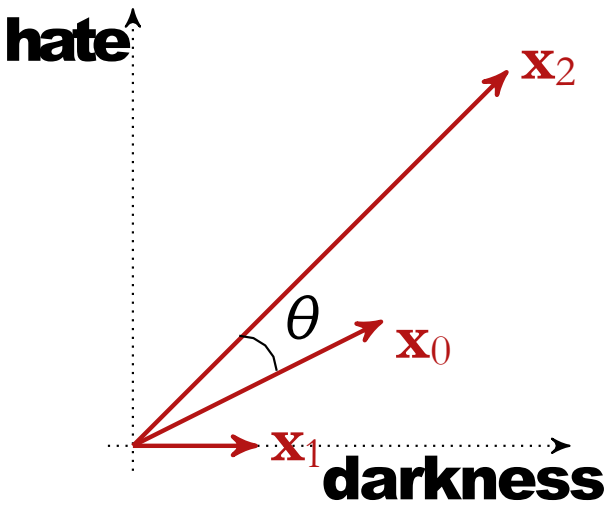
$$||\mathbf{v}|| = \sqrt{\sum_{j=1}^{|V|} v_j^2}.$$

The distance between two vectors corresponds to the angle between them:

$$d(\mathbf{x}, \mathbf{x}') = \qquad .$$

Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



The **length of a vector** is its distance from the origin $\mathbf{0}$:

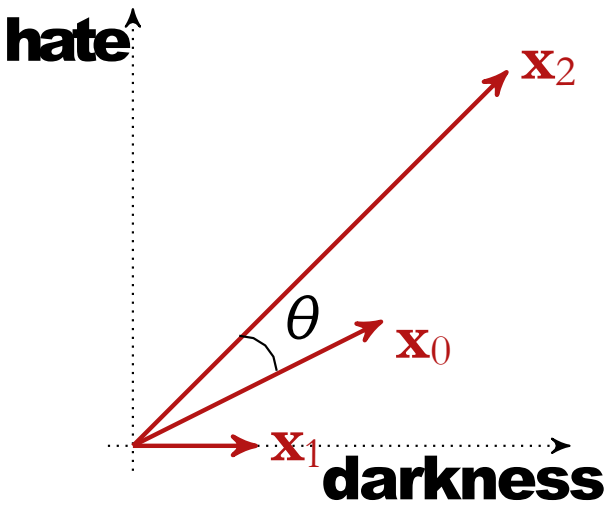
$$||\mathbf{v}|| = \sqrt{\sum_{j=1}^{|V|} v_j^2}.$$

The distance between two vectors corresponds to the angle between them:

$$d(\mathbf{x}, \mathbf{x}') = 1 - \cos\theta \quad .$$

Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



The **length of a vector** is its distance from the origin $\mathbf{0}$:

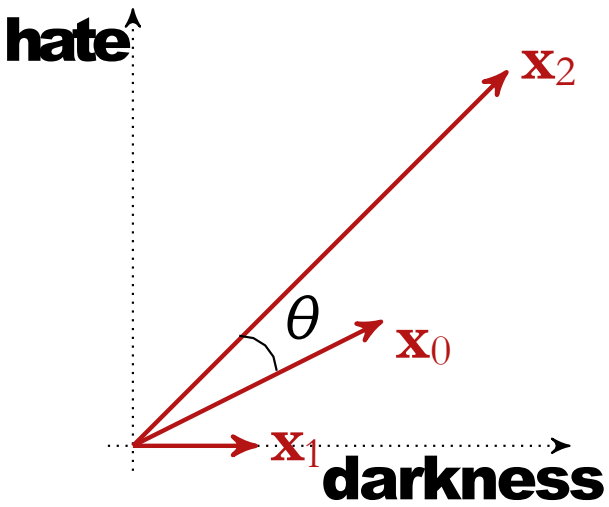
$$||\mathbf{v}|| = \sqrt{\sum_{j=1}^{|V|} v_j^2}.$$

The distance between two vectors corresponds to the angle between them:

$$d(\mathbf{x}, \mathbf{x}') = 1 - \cos\theta = 1 - \frac{\sum_{j=1}^{|V|} x_j x'_j}{||\mathbf{x}||\ ||\mathbf{x}'||}.$$

Vector Space Model

In the **vector space model**, documents are represented as *vectors* instead of points.



The **length of a vector** is its distance from the origin $\mathbf{0}$:

$$||\mathbf{v}|| = \sqrt{\sum_{j=1}^{|V|} v_j^2}.$$

The distance between two vectors corresponds to the angle between them:

$$d(\mathbf{x}, \mathbf{x}') = 1 - \cos\theta = 1 - \frac{\sum_{j=1}^{|V|} x_j x'_j}{||\mathbf{x}||\ ||\mathbf{x}'||}.$$

Using cosine distance, document 2 now appears closer!

Implementing the Vector Space Model

```
documents = [
    "whoever has hate for his brother is in the darkness and walks in the dark
    "hello darkness my old friend",
    "returning hate for hate multiplies hate adding deeper darkness to a night
]
```

Creates a list named documents.
The list contains 3 text strings.

Implementing the Vector Space Model

```
documents = [
    "whoever has hate for his brother is in the darkness and walks in the dark
    "hello darkness my old friend",
    "returning hate for hate multiplies hate adding deeper darkness to a night
]
```

First, we use Pandas to get the term-frequency matrix.

```
import pandas as pd
from collections import Counter

tf = pd.DataFrame(
    [Counter(doc.split()) for doc in documents],
).fillna(0)
tf
```

Implementing the Vector Space Model

```
documents = [
    "whoever has hate for his brother is in the darkness and walks in the dark
    "hello darkness my old friend",
    "returning hate for hate multiplies hate adding deeper darkness to a night
]
```

First, we use Pandas to get the term-frequency matrix.

```
import pandas as pd
from collections import Counter

tf = pd.DataFrame(
    [Counter(doc.split()) for doc in documents],
).fillna(0)
tf
```

	whoever	has	hate	for	...	light	can	do	that
0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	3.0	1.0	...	1.0	1.0	1.0	1.0

3 rows × 35 columns

Counter(doc.split()) → Splits each document into words and counts how many times each word appears.
Creates a list → Combines these word-count dictionaries for all documents into a list.pd.
DataFrame(...) → Converts that list into a pandas
DataFrame.fillna(0) → Replaces missing values with 0

builds a Term Frequency **(TF)** matrix that shows how often each word occurs in each document.

Implementing the Vector Space Model

```
documents = [
    "whoever has hate for his brother is in the darkness and walks in the dark
    "hello darkness my old friend",
    "returning hate for hate multiplies hate adding deeper darkness to a night
]
```

First, we use Pandas to get the term-frequency matrix.

```
import pandas as pd
from collections import Counter

tf = pd.DataFrame(
    [Counter(doc.split()) for doc in documents],
).fillna(0)
tf
```

	whoever	has	hate	for	...	light	can	do	that
0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	3.0	1.0	...	1.0	1.0	1.0	1.0

3 rows × 35 columns

Now we just have to implement the formula for cosine distance.

Implementing the Vector Space Model

$tf =$

	whoever	has	hate	for	...	light	can	do	that
0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	3.0	1.0	...	1.0	1.0	1.0	1.0

3 rows × 35 columns

Implementing the Vector Space Model

tf =

	whoever	has	hate	for	...	light	can	do	that
0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	3.0	1.0	...	1.0	1.0	1.0	1.0

3 rows × 35 columns

Now we just have to implement the formula for cosine distance.

$$d(\mathbf{x}, \mathbf{x}') = 1 - \frac{\sum_{j=1}^{|V|} x_j x'_j}{||\mathbf{x}||\ ||\mathbf{x}'||}.$$

Cosine distance measures the similarity or difference between two vectors.

It is based on the **angle** between the vectors — the more aligned their directions, the more similar they are.

Cosine distance = 1 − cosine similarity
shows the difference (0 → very similar, 1 → completely different).

Implementing the Vector Space Model

tf =

	whoever	has	hate	for	...	light	can	do	that
0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	3.0	1.0	...	1.0	1.0	1.0	1.0

3 rows × 35 columns

Now we just have to implement the formula for cosine distance.

$$d(\mathbf{x}, \mathbf{x}') = 1 - \frac{\sum_{j=1}^{|V|} x_j x'_j}{||\mathbf{x}||\ ||\mathbf{x}'||}.$$

```
import numpy as np

def length(v):
  return np.sqrt((v ** 2).sum())

def cos_dist(v, w):
  return 1 - (v * w).sum() / (length(v) * length(w))

cos_dist(tf.loc[0], tf.loc[1]), cos_dist(tf.loc[0], tf.loc[2])
```

Implementing the Vector Space Model

tf =

	whoever	has	hate	for	...	light	can	do	that
0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	3.0	1.0	...	1.0	1.0	1.0	1.0

3 rows × 35 columns

Now we just have to implement the formula for cosine distance.

$$d(\mathbf{x}, \mathbf{x}') = 1 - \frac{\sum_{j=1}^{|V|} x_j x'_j}{||\mathbf{x}||\ ||\mathbf{x}'||}.$$

```
import numpy as np

def length(v):
  return np.sqrt((v ** 2).sum())

def cos_dist(v, w):
  return 1 - (v * w).sum() / (length(v) * length(w))

cos_dist(tf.loc[0], tf.loc[1]), cos_dist(tf.loc[0], tf.loc[2])
```

(0.8048199854102933, 0.6460038372976056)

Vector Space Model in Scikit-Learn

It’s always easier to do it in Scikit-Learn.

Vector Space Model in Scikit-Learn

It's always easier to do it in Scikit-Learn.

```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(token_pattern=r"\w+")
vec.fit(documents)
tf_matrix = vec.transform(documents)
tf_matrix.todense()
```

sparse vs todense matrices
Converts a sparse matrix (with mostly zeros) into a regular dense matrix for easier viewing or manual calculations.

Before todense(), the matrix is sparse — it only stores the positions and values of non-zero elements, not the zeros themselves.

Vector Space Model in Scikit-Learn

It's always easier to do it in Scikit-Learn.

```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(token_pattern=r"\w+")
vec.fit(documents)
tf_matrix = vec.transform(documents)
tf_matrix.todense()
```

```
matrix([[0, 0, 0, 1, 1, 0, 0, 2, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 2, 1, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 1],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
         0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 1, 0, 0, 1, 1, 3, 1, 1, 1, 1, 1, 0, 0, 3, 0, 0, 0, 0, 1,
         1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0]])
```

CountVectorizer(token_pattern=r"\w+") → Creates a CountVectorizer that splits text into words (\w+ Find words made up of one or more letters/numbers).
vec.fit(documents) → Learns the vocabulary from all documents.
vec.transform(documents) → Converts each document into a vector of word counts.
tf_matrix.todense() → Converts the sparse matrix into a dense matrix (normal numeric table).

Vector Space Model in Scikit-Learn

It's always easier to do it in Scikit-Learn.

```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(token_pattern=r"\w+")
vec.fit(documents)
tf_matrix = vec.transform(documents)
tf_matrix.todense()
```

```
matrix([[0, 0, 0, 1, 1, 0, 0, 2, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 2, 1, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 1],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
         0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 1, 0, 0, 1, 1, 3, 1, 1, 1, 1, 1, 0, 0, 3, 0, 0, 0, 0, 1,
         1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0]])
```

```
from sklearn.metrics import pairwise_distances
pairwise_distances(tf_matrix[0, :], tf_matrix,
                   metric="cosine")
```

Vector Space Model in Scikit-Learn

It's always easier to do it in Scikit-Learn.

```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(token_pattern=r"\w+")
vec.fit(documents)
tf_matrix = vec.transform(documents)
tf_matrix.todense()
```

```
matrix([[0, 0, 0, 1, 1, 0, 0, 2, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 2, 1, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 1],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
         0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 1, 0, 0, 1, 1, 3, 1, 1, 1, 1, 1, 0, 0, 3, 0, 0, 0, 0, 1,
         1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0]])
```

```
from sklearn.metrics import pairwise_distances
pairwise_distances(tf_matrix[0, :], tf_matrix,
                   metric="cosine")
```

```
array([[0.        , 0.80481999, 0.64600384]])
```

a smaller cosine distance means higher similarity.
The first document is most similar to the third one (0.6460).

1. Review
2. Vector Space Model
3. tf-idf

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

tf(d, t) = how frequently the word appears in that specific document

idf(t, D) = how rare the word is across all documents.

If the TF-IDF value is high → the word is unique to that document and carries strong distinguishing meaning.

If a word appears in every document → df(t, D) ≈ |D| → log(1) = 0 → idf ≈ 0

- This word has no discriminative power (it's unimportant).

If a word appears in few documents → df is small → |D|/df is large → log is large → idf is high

- This word is important for distinguishing between documents.

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

Problem: Common words like "is" or "the" tend to dominate because they have high counts.

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

Problem: Common words like "is" or "the" tend to dominate because they have high counts.

We need to adjust for how common each word is:

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

Problem: Common words like "is" or "the" tend to dominate because they have high counts.

We need to adjust for how common each word is:

1. Count the fraction of documents the term appears in:

$$\mathrm{df}(t, D) = \frac{\#\ \text{documents containing term } t}{\#\ \text{documents}} = \frac{|\{d \in D : t \in d\}|}{|D|}$$

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

Problem: Common words like "is" or "the" tend to dominate because they have high counts.

We need to adjust for how common each word is:

1. Count the fraction of documents the term appears in:

$$\mathrm{df}(t, D) = \frac{\#\ \text{documents containing term } t}{\#\ \text{documents}} = \frac{|\{d \in D : t \in d\}|}{|D|}$$

2. Invert and take a log to obtain **inverse document frequency**:

$$\mathrm{idf}(t, D) = 1 + \log \frac{1}{\mathrm{df}(t, D)}.$$

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

Problem: Common words like "is" or "the" tend to dominate because they have high counts.

We need to adjust for how common each word is:

1. Count the fraction of documents the term appears in:

$$\mathrm{df}(t, D) = \frac{\#\ \text{documents containing term } t}{\#\ \text{documents}} = \frac{|\{d \in D : t \in d\}|}{|D|}$$

2. Invert and take a log to obtain **inverse document frequency**:

$$\mathrm{idf}(t, D) = 1 + \log \frac{1}{\mathrm{df}(t, D)}.$$

3. Multiply tf by idf to get tf-idf:

$$\text{tf-idf}(d, t, D) = \mathrm{tf}(d, t) \cdot \mathrm{idf}(t, D).$$

tf-idf

So far, we've simply counted the **term frequency** $\mathrm{tf}(d, t)$: how many times each term t appears in each document d.

Problem: Common words like "is" or "the" tend to dominate because they have high counts.

We need to adjust for how common each word is:

1. Count the fraction of documents the term appears in:

$$\mathrm{df}(t, D) = \frac{\#\ \text{documents containing term } t}{\#\ \text{documents}} = \frac{|\{d \in D : t \in d\}|}{|D|}$$

2. Invert and take a log to obtain **inverse document frequency**:

$$\mathrm{idf}(t, D) = 1 + \log \frac{1}{\mathrm{df}(t, D)}.$$

3. Multiply tf by idf to get tf-idf:

$$\text{tf-idf}(d, t, D) = \mathrm{tf}(d, t) \cdot \mathrm{idf}(t, D).$$

Now we can use the **tf-idf matrix** just like we used the term-frequency matrix.

tf-idf by Hand

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

⇒

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

⇒

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	**hate**	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1$$

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	**hate**	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1 \qquad \text{df}(\text{“hate”}, D) = \frac{2}{3}$$

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1 \qquad \text{df}(\text{“hate”}, D) = \frac{2}{3}$$

2. Calculate the inverse document frequencies:

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1 \qquad \text{df}(\text{“hate”}, D) = \frac{2}{3}$$

2. Calculate the inverse document frequencies:

$$\text{idf}(\text{“darkness”}, D) = 1 + \log 1 = 1$$

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

Darkenss is seen in 3 documents,
hate is seen in 2 documnets

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1 \qquad \text{df}(\text{“hate”}, D) = \frac{2}{3}$$

2. Calculate the inverse document frequencies:

$$\text{idf}(\text{“darkness”}, D) = 1 + \log 1 = 1 \quad \text{idf}(\text{“hate”}, D) = 1 + \log \frac{3}{2} \approx 1.176$$

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1 \qquad \text{df}(\text{“hate”}, D) = \frac{2}{3}$$

2. Calculate the inverse document frequencies:

$$\text{idf}(\text{“darkness”}, D) = 1 + \log 1 = 1 \quad \text{idf}(\text{“hate”}, D) = 1 + \log \frac{3}{2} \approx 1.176$$

3. Multiply tf by idf to get tf-idf:

tf-idf by Hand

The term-frequency matrix for this corpus is:

1. "Whoever has hate for his brother is in the darkness and walks in the darkness."
2. "Hello darkness, my old friend."
3. "Returning hate for hate multiplies hate, adding deeper darkness to a night already devoid of stars. Darkness cannot drive out darkness; only light can do that."

$\Rightarrow$

	darkness	hate	…
0	2	1	…
1	1	0	…
2	3	3	…

hate higher idf fewer appearance more distinctive
darkness appear all docs. so idf 1 less important
TF-IDF calculation increases the weight of "hate" and decreases the weight of "darkness."

Now let's calculate the tf-idf matrix!

1. Calculate the document frequencies:

$$\text{df}(\text{“darkness”}, D) = \frac{3}{3} = 1 \qquad \text{df}(\text{“hate”}, D) = \frac{2}{3}$$

2. Calculate the inverse document frequencies:

$$\text{idf}(\text{“darkness”}, D) = 1 + \log 1 = 1 \qquad \text{idf}(\text{“hate”}, D) = 1 + \log \frac{3}{2} \approx 1.176$$

3. Multiply tf by idf to get tf-idf:

	darkness	hate	…
0	2	1.176	…
1	1	0	…
2	3	3.528	…

tf-idf in Scikit-Learn

```
from sklearn.feature_extraction.text import TfidfVectorizer

# The options ensure that the numbers match our example above.
vec = TfidfVectorizer(smooth_idf=False, norm=None)
vec.fit(documents)
tfidf_matrix = vec.transform(documents)
```

from sklearn.feature_extraction.text import TfidfVectorizer→ Imports the TfidfVectorizer class, which converts text into TF-IDF features.

vec = TfidfVectorizer(smooth_idf=False, norm=None)→ Creates a TF-IDF vectorizer object.

smooth_idf=False: uses the **exact** IDF formula (no smoothing).

norm=None: does not normalize the vectors (keeps raw TF-IDF values).

vec.fit(documents)→ Learns the vocabulary and computes IDF (Inverse Document Frequency) values from all documents.

tfidf_matrix = vec.transform(documents)→ Converts each document into a TF-IDF weighted vector, based on the learned vocabulary.

tf-idf in Scikit-Learn

```
from sklearn.feature_extraction.text import TfidfVectorizer

# The options ensure that the numbers match our example above.
vec = TfidfVectorizer(smooth_idf=False, norm=None)
vec.fit(documents)
tfidf_matrix = vec.transform(documents)
```

Now we can use this tf-idf matrix just as we used the term frequency matrix!

tf-idf in Scikit-Learn

```
from sklearn.feature_extraction.text import TfidfVectorizer

# The options ensure that the numbers match our example above.
vec = TfidfVectorizer(smooth_idf=False, norm=None)
vec.fit(documents)
tfidf_matrix = vec.transform(documents)
```

Now we can use this tf-idf matrix just as we used the term frequency matrix!

```
pairwise_distances(tfidf_matrix[0, :], tfidf_matrix,
                   metric="cosine")
```

tf-idf in Scikit-Learn

```
from sklearn.feature_extraction.text import TfidfVectorizer

# The options ensure that the numbers match our example above.
vec = TfidfVectorizer(smooth_idf=False, norm=None)
vec.fit(documents)
tfidf_matrix = vec.transform(documents)
```

Now we can use this tf-idf matrix just as we used the term frequency matrix!

```
pairwise_distances(tfidf_matrix[0, :], tfidf_matrix,
                   metric="cosine")
```

```
array([[0.        , 0.94612045, 0.84453506]])
```

Dr. Seuss Example

Let's go into Colab and find the Dr. Seuss book that is most similar to *One Fish, Two Fish, Red Fish, Blue Fish*.