



COMP30810

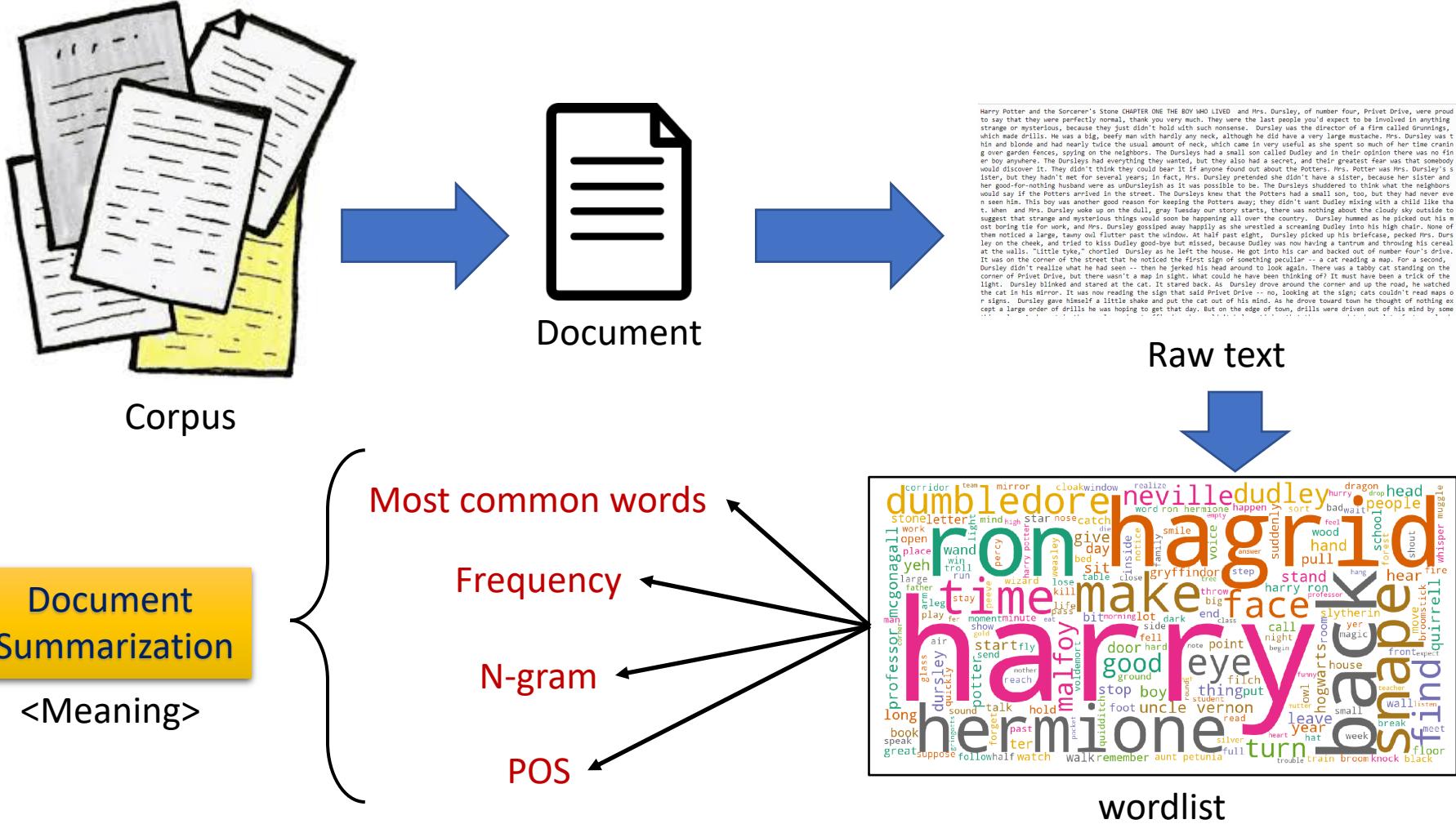
Intro to Text Analytics

Dr. Binh Thanh Le

thanhbinh.le@ucd.ie

Insight Centre for Data Analytics
School of Computer Science
University College Dublin

Recap of previous lecture



Today Goals:

- Understanding TF-IDF
- Understanding ranking
- Understanding distance and similarity measurement
- Document summarization

Example

- Given the corpus of 4 documents as:

Doc1: "The sky is blue."

Doc2: "The sun is bright today."

Doc3: "The sun in the sky is bright."

Doc4: "We can see the shining sun, the bright sun."

Our task: return the analysis for this corpus

1) TF-IDF

Term Frequency–Inverse Document Frequency

Term Frequency : gives us the *frequency of the word in each document in the corpus*.

It is the ratio of **number of times the word appears** in a document compared to the **total number of words in that document**.

$$tf(t, d) = \frac{f_d(t)}{\sum_{t' \in d} f_d(t')}$$

term document

↓ ↓

Note:

- It increases as the number of occurrences of that word within the document increases.
- Each document has its own tf.

Variants of term frequency (tf) weight	
weighting scheme	tf weight
binary	0, 1
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

1) TF-IDF

Python code

First: build the wordlist for corpus:

```
import nltk
from nltk.tokenize import RegexpTokenizer

Doc1 = "The sky is blue."
Doc2 = "The sun is bright today."
Doc3 = "The sun in the sky is bright."
Doc4 = "We can see the shining sun, the bright sun."

## Retrieve the wordlist
alltext = ' '.join([Doc1,Doc2,Doc3,Doc4])
# tokenizer
pattern = r'\w+'
tokenizer = RegexpTokenizer(pattern)
corpus_tokens = tokenizer.tokenize(alltext)

# decapitalize
wordlist = list(set([word.lower() for word in corpus_tokens]))
wordlist
```

bag-of-words



```
['sky',
 'in',
 'bright',
 'sun',
 'today',
 'we',
 'can',
 'see',
 'shining',
 'is',
 'blue',
 'the']
```

1) TF-IDF

Python code

<Term Frequency>

Easy code for understanding:

```
import pandas as pd
statistic_table = pd.DataFrame([])
statistic_table['Terms'] = wordlist
statistic_table['Doc1'] = [0]*len(statistic_table)
statistic_table['Doc2'] = [0]*len(statistic_table)
statistic_table['Doc3'] = [0]*len(statistic_table)
statistic_table['Doc4'] = [0]*len(statistic_table)
for idx, row in statistic_table.iterrows():
    statistic_table['Doc1'].iloc[idx] = round(Doc1.lower().count(row['Terms']) / len(Doc1.lower().split()),2)
    statistic_table['Doc2'].iloc[idx] = round(Doc2.lower().count(row['Terms']) / len(Doc2.lower().split()),2)
    statistic_table['Doc3'].iloc[idx] = round(Doc3.lower().count(row['Terms']) / len(Doc3.lower().split()),2)
    statistic_table['Doc4'].iloc[idx] = round(Doc4.lower().count(row['Terms']) / len(Doc4.lower().split()),2)
statistic_table
```

	Terms	Doc1	Doc2	Doc3	Doc4
0	sky	0.25	0.0	0.14	0.00
1	in	0.00	0.0	0.14	0.22
2	bright	0.00	0.2	0.14	0.11
3	sun	0.00	0.2	0.14	0.22
4	today	0.00	0.2	0.00	0.00
5	we	0.00	0.0	0.00	0.11
6	can	0.00	0.0	0.00	0.11
7	see	0.00	0.0	0.00	0.11
8	shining	0.00	0.0	0.00	0.11
9	is	0.25	0.2	0.14	0.00
10	blue	0.25	0.0	0.00	0.00
11	the	0.25	0.2	0.29	0.22

$$f_d(t)$$

$$\sum_{t' \in d} f_d(t')$$

1) TF-IDF

Inverse Document Frequency

- used to calculate the **weight of rare words** across all documents in the corpus.
- The words that occur **rarely** in the corpus have a **high** IDF score.

$$idf(t, C) = \log \left(\frac{|C|}{|\{d \in C : t \in d\}|} \right)$$

$$= \log \left(\frac{\text{total number of documents}}{\text{the number of documents containing the term}} \right)$$

1) TF-IDF

Python code

< Inverse Document Frequency >

```
## Compute
# total number of documents
n_docs = len(statistic_table.columns) -1 # = 4
#the number of documents containing the term
n_term = ((statistic_table['Doc1'] != 0)*1 +
           (statistic_table['Doc2'] != 0)*1 +
           (statistic_table['Doc3'] != 0)*1 +
           (statistic_table['Doc4'] != 0)*1)
import numpy as np
statistic_table['idf'] = round(np.log(n_docs / n_term),2)
statistic_table
```

Or

```
#the number of documents containing the term
n_term = np.sum((statistic_table.iloc[:,1:]!=0)*1, axis =1)
n_term
```

Terms	TF				idf
	Doc1	Doc2	Doc3	Doc4	
sky	0.25	0.0	0.14	0.00	0.69
in	0.00	0.0	0.14	0.22	0.69
bright	0.00	0.2	0.14	0.11	0.29
sun	0.00	0.2	0.14	0.22	0.29
today	0.00	0.2	0.00	0.00	1.39
we	0.00	0.0	0.00	0.11	1.39
can	0.00	0.0	0.00	0.11	1.39
see	0.00	0.0	0.00	0.11	1.39
shining	0.00	0.0	0.00	0.11	1.39
is	0.25	0.2	0.14	0.00	0.29
blue	0.25	0.0	0.00	0.00	1.39
the	0.25	0.2	0.29	0.22	0.00

1) TF-IDF

TF-IDF

Combining these two we come up with the TF-IDF score for a word in a document in the corpus. It is the product of TF and IDF

$$tfidf(t, d, C) = tf(t, d) \times idf(t, C)$$

Python code

```
statistic_table['tfidf-Doc1'] = statistic_table['Doc1']*statistic_table['idf']
statistic_table['tfidf-Doc2'] = statistic_table['Doc2']*statistic_table['idf']
statistic_table['tfidf-Doc3'] = statistic_table['Doc3']*statistic_table['idf']
statistic_table['tfidf-Doc4'] = statistic_table['Doc4']*statistic_table['idf']
statistic_table
```

1) TF-IDF

Terms	TF				idf	IDF				TF-IDF			
	Doc1	Doc2	Doc3	Doc4		tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4	tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4
sky	0.25	0.0	0.14	0.00	0.69	0.1725	0.000	0.0966	0.0000	0.1725	0.000	0.0966	0.0000
in	0.00	0.0	0.14	0.22	0.69	0.0000	0.000	0.0966	0.1518	0.0000	0.000	0.0966	0.1518
bright	0.00	0.2	0.14	0.11	0.29	0.0000	0.058	0.0406	0.0319	0.0000	0.058	0.0406	0.0319
sun	0.00	0.2	0.14	0.22	0.29	0.0000	0.058	0.0406	0.0638	0.0000	0.058	0.0406	0.0638
today	0.00	0.2	0.00	0.00	1.39	0.0000	0.278	0.0000	0.0000	0.0000	0.278	0.0000	0.0000
we	0.00	0.0	0.00	0.11	1.39	0.0000	0.000	0.0000	0.1529	0.0000	0.000	0.0000	0.1529
can	0.00	0.0	0.00	0.11	1.39	0.0000	0.000	0.0000	0.1529	0.0000	0.000	0.0000	0.1529
see	0.00	0.0	0.00	0.11	1.39	0.0000	0.000	0.0000	0.1529	0.0000	0.000	0.0000	0.1529
shining	0.00	0.0	0.00	0.11	1.39	0.0000	0.000	0.0000	0.1529	0.0000	0.000	0.0000	0.1529
is	0.25	0.2	0.14	0.00	0.29	0.0725	0.058	0.0406	0.0000	0.0725	0.058	0.0406	0.0000
blue	0.25	0.0	0.00	0.00	1.39	0.3475	0.000	0.0000	0.0000	0.3475	0.000	0.0000	0.0000
the	0.25	0.2	0.29	0.22	0.00	0.0000	0.000	0.0000	0.0000	0.0000	0.000	0.0000	0.0000

Doc1: " ^{The} sky is blue"

2) Information Retrieval (IR)

- What is the document having a good description for the “**sun**”?

Doc1: "The sky is blue."

Doc2: "The **sun** is bright today."

Doc3: "The **sun** in the sky is bright."

Doc4: "We can see the shining **sun**, the bright **sun**."

Query: “**sun**”

Terms	tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4
sun	0.0000	0.058	0.0406	0.0638

Terms	tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4
sky	0.1725	0.000	0.0966	0.0000
in	0.0000	0.000	0.0966	0.1518
bright	0.0000	0.058	0.0406	0.0319
sun	0.0000	0.058	0.0406	0.0638
today	0.0000	0.278	0.0000	0.0000
we	0.0000	0.000	0.0000	0.1529
can	0.0000	0.000	0.0000	0.1529
see	0.0000	0.000	0.0000	0.1529
shining	0.0000	0.000	0.0000	0.1529
is	0.0725	0.058	0.0406	0.0000
blue	0.3475	0.000	0.0000	0.0000
the	0.0000	0.000	0.0000	0.0000

Word vector

Document vector

2) Information Retrieval (IR)

1. What is the document having a good description for the “sun”?

Doc1: "The sky is blue."
Doc2: "The sun is bright today."
Doc3: "The sun in the sky is bright."
Doc4: "We can see the shining sun, the bright sun."

Query: “sun”

Terms	tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4
sun	0.0000	0.058	0.0406	0.0638

2. What is the document having a good description for the “sun” or “sky”?

Query: “sun” or “sky”

Terms	tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4
sky	0.1725	0.000	0.0966	0.0000
sun	0.0000	0.058	0.0406	0.0638

Why?: the description is stronger in Doc1, because the word “blue” has high information content.

3) Documents as vectors

- Each document is now represented by a real-valued vector of tf-idf $\in \mathcal{R}^{|V|}$
 - So, we have a $|V|$ - dimension vector space
 - Terms are axes of the space
 - Documents are samples in this space
- This is a very **high-dimensional space**: easily to get tens of thousands (millions) of dimensions.
- These are very **sparse vectors** – most entries are zero.

Terms	tfidf-Doc1	tfidf-Doc2	tfidf-Doc3	tfidf-Doc4
sky	0.1725	0.000	0.0966	0.0000
in	0.0000	0.000	0.0966	0.1518
bright	0.0000	0.058	0.0406	0.0319
sun	0.0000	0.058	0.0406	0.0638
today	0.0000	0.278	0.0000	0.0000
we	0.0000	0.000	0.0000	0.1529
can	0.0000	0.000	0.0000	0.1529
see	0.0000	0.000	0.0000	0.1529
shining	0.0000	0.000	0.0000	0.1529
is	0.0725	0.058	0.0406	0.0000
blue	0.3475	0.000	0.0000	0.0000
the	0.0000	0.000	0.0000	0.0000

Word vector

↑ Document vector

$|V|$: size of terms (=12)

4) What can we do with vectors?

- Assume we have a story (book) → corpus

Task: get the most informative sentence and word in that story

→ Book is a corpus

→ Sentences are documents

Document
Summarization

Document
Ranking

	Word 1	Word 2	Word 3	SUM
Sentence 1	tf-idf	tf-idf	tf-idf	0.124
Sentence 2	tf-idf	tf-idf	tf-idf	0.235
Sentence 3	tf-idf	tf-idf	tf-idf	0.254
Sentence 4	tf-idf	tf-idf	tf-idf	0.568
...				
SUM	0.256	1.568	1.245	

This is the
most
informative
sentence

This is the most informative word in story

5) Improvement for ranking

- To rank each sentence → use the tf-idf values.
- However, rather than **simply taking the summation of all the values** for a given sentence, it will be useful when using some **additional techniques** as follow:
 1. Only summing tf-idf values where the underling word is a **noun**.
 2. Add an **additional value** to a given sentence **if it has any words that are included in the title** of the document (title can be the first sentence of document).
 3. **Apply a position weighting**. For example if there are 10 sentences in a document, sentence nine's "position weighting" would be 0.9. This weighting is then multiplied by the value calculated in point 2.
 4. **Apply n-gram** to check the tf-idf of multiword in sentence.

6) Distance and Similarity

We will cover three basic distance measurements in Text Mining:

- Euclidean Distance
 - Cosine Similarity
 - Jaccard Similarity
-

Consider these three sentences:

`s1 = "David loves dogs"`

`s2 = "Dogs are ok with David"`

`s3 = "Cats love rain"`

6) Distance

1. We generate tokens for sentences:

s1 = ("david", "love", "dog")

s2 = ("dog", "ok", "david")

s3 = ("cat", "love", "rain")

2. We build a vocabulary with all the words in our corpus.

bag-of-words

{'cat': 0, 'david': 1, 'dog': 2, 'love': 3, 'ok': 4, 'rain': 5}

3. Then, we generate the vectors (6-dimensional space) by using counting

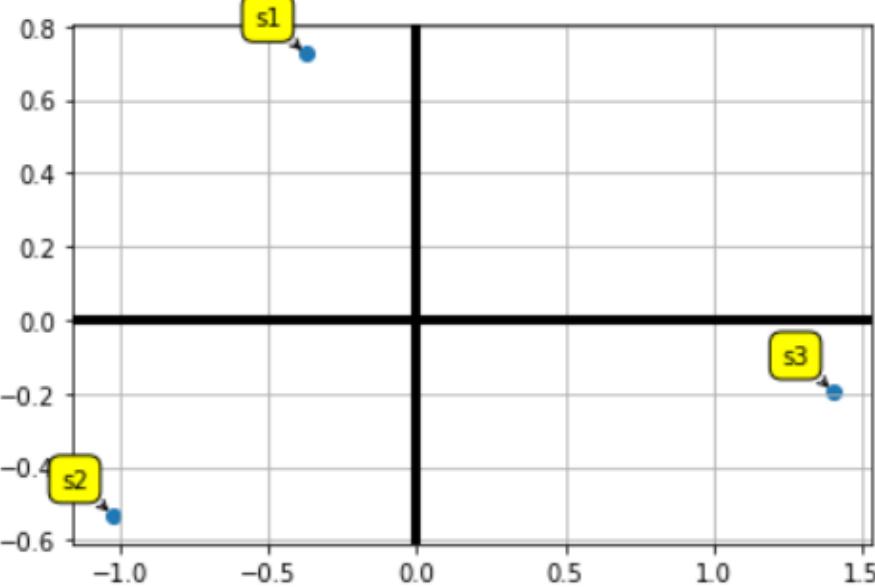
s1 = [0, 1, 1, 1, 0, 0]

s2 = [0, 1, 1, 0, 1, 0]

s3 = [1, 0, 0, 1, 0, 1]

In here, you can use tf-idf vectors, but for the example, I simply use counting

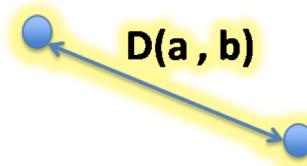
4. Reduce 6-d to 2-d for visualization (using PCA)

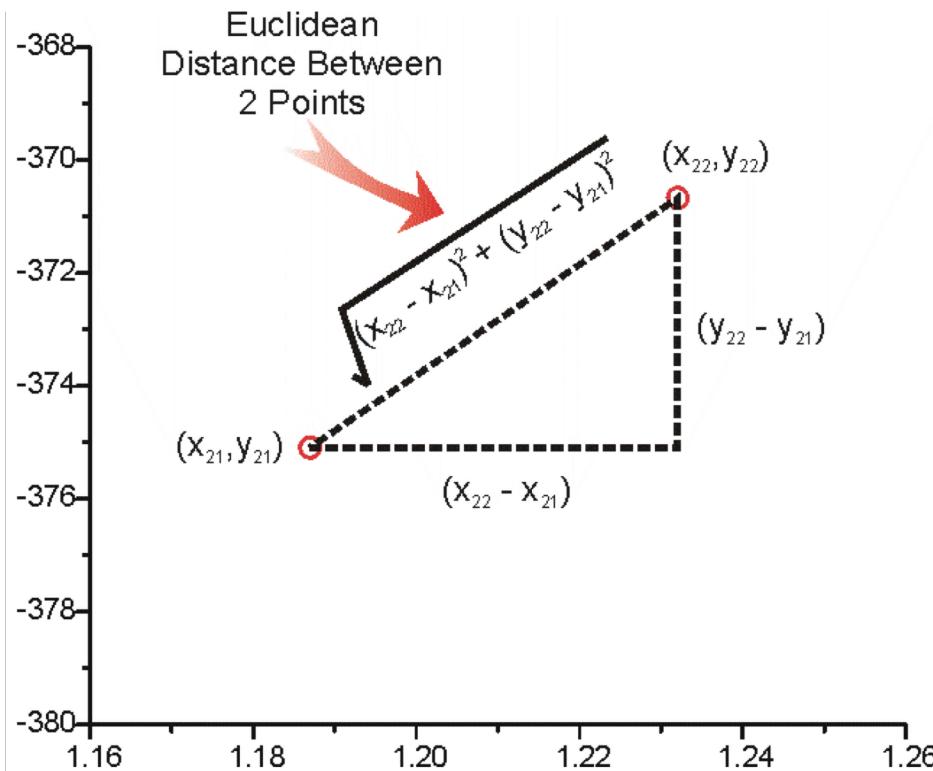


```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X = np.array([s1, s2, s3])
v = pca.fit_transform(X)
```

6) Distance

- So, the **Euclidean distance** is good in this case

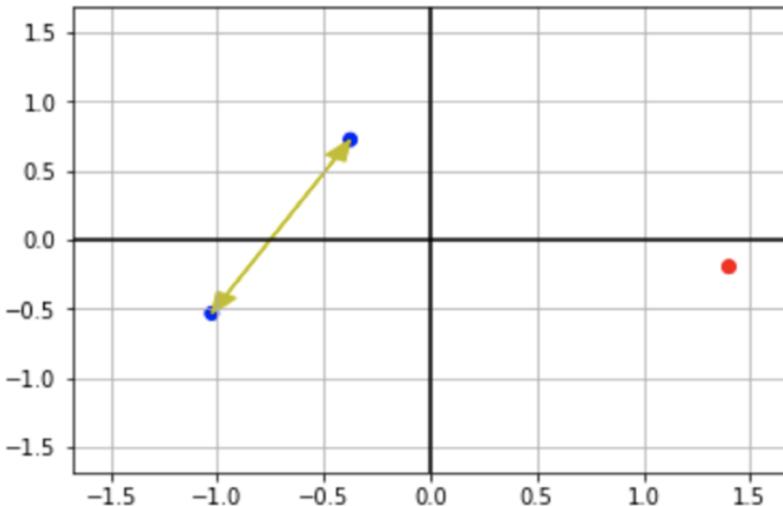
$$D(a, b) = \sqrt{\sum_{i=1}^n (b_i - a_i)^2}$$




6) Distance

- So, the Euclidean distance is good in this case

$$D(a, b) = \sqrt{\sum_{i=1}^n (b_i - a_i)^2}$$



Euclidean distance matrix

	s1	s2	s3
s1	0	1.41	2
s2	1.41	0	2.44
s3	2	2.44	0

```
from sklearn.metrics.pairwise import euclidean_distances
import numpy as np

np.round(euclidean_distances([s1,s2,s3],[s1,s2,s3]),2)
```

S1 to S2 are closer than S1 to S3, or S2 to S3

6) Distance

Now, consider these sentences:

$s1 = \text{"David loves dogs dogs dogs dogs dogs"}$

$s2 = \text{"Dogs are ok with David"}$

$s3 = \text{"Cats love rain"}$



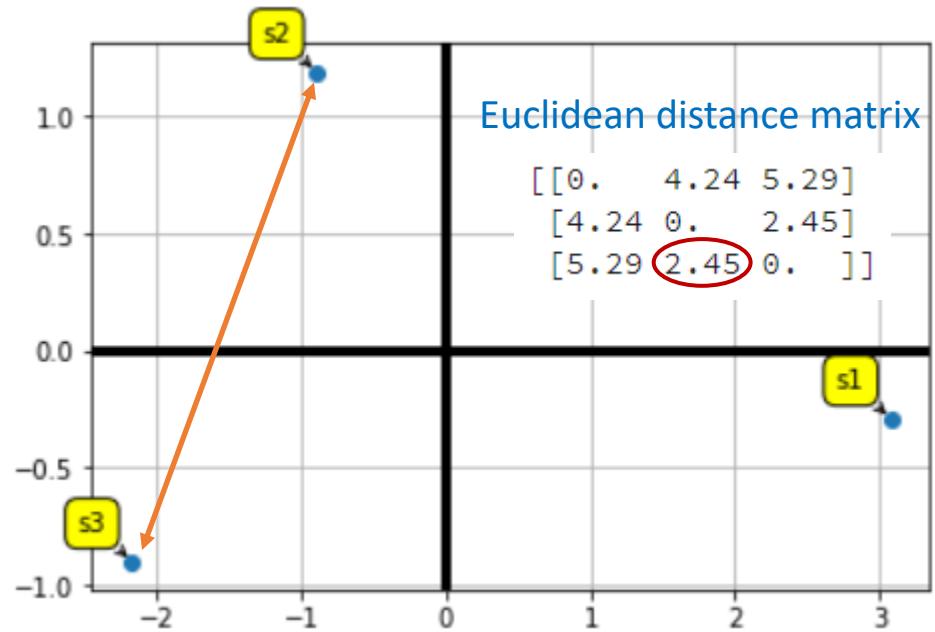
$\{\text{cat': 0, 'david': 1, 'dog': 2, 'love': 3, 'ok': 4, 'rain': 5}\}$

$s1 = [0, 1, 5, 1, 0, 0]$

$s2 = [0, 1, 1, 0, 1, 0]$

$s3 = [1, 0, 0, 1, 0, 1]$

Let's plot those vectors using PCA again



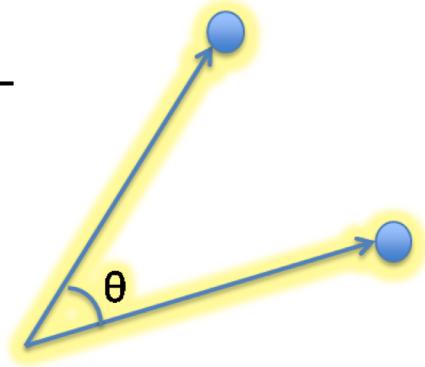
See what happened: now, Sentence 2 and 3 are closer than 1 and 2.

This is bad

6) Similarity

Cosine Similarity

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



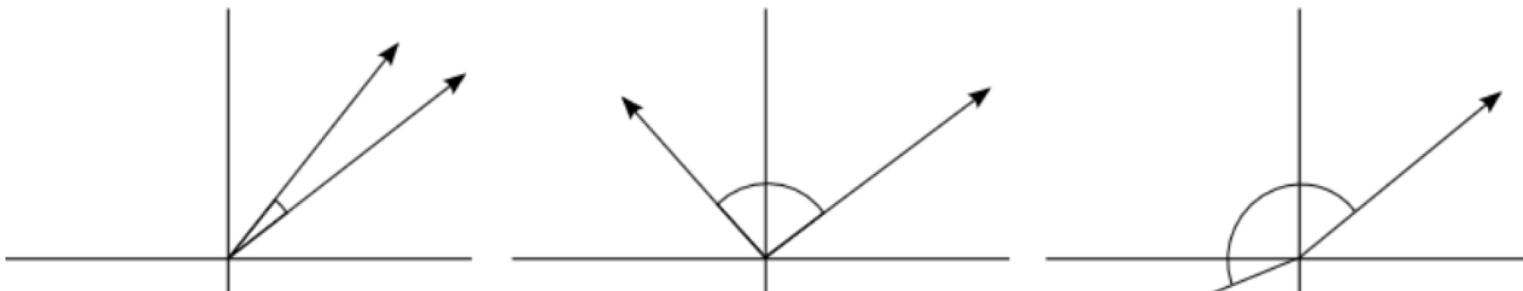
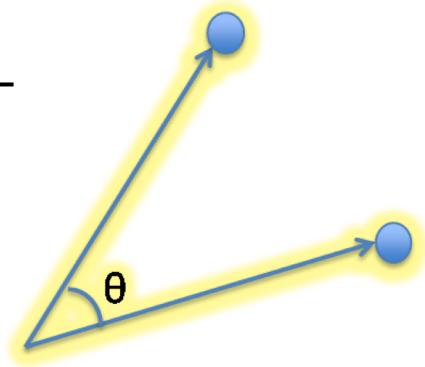
$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos \theta$$

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

6) Similarity

Cosine Similarity

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



Similar scores
Score Vectors in same direction
Angle between them is near 0 deg.
Cosine of angle is near 1 i.e. 100%

Unrelated scores
Score Vectors are nearly orthogonal
Angle between them is near 90 deg.
Cosine of angle is near 0 i.e. 0%

Opposite scores
Score Vectors in opposite direction
Angle between them is near 180 deg.
Cosine of angle is near -1 i.e. -100%

Similar

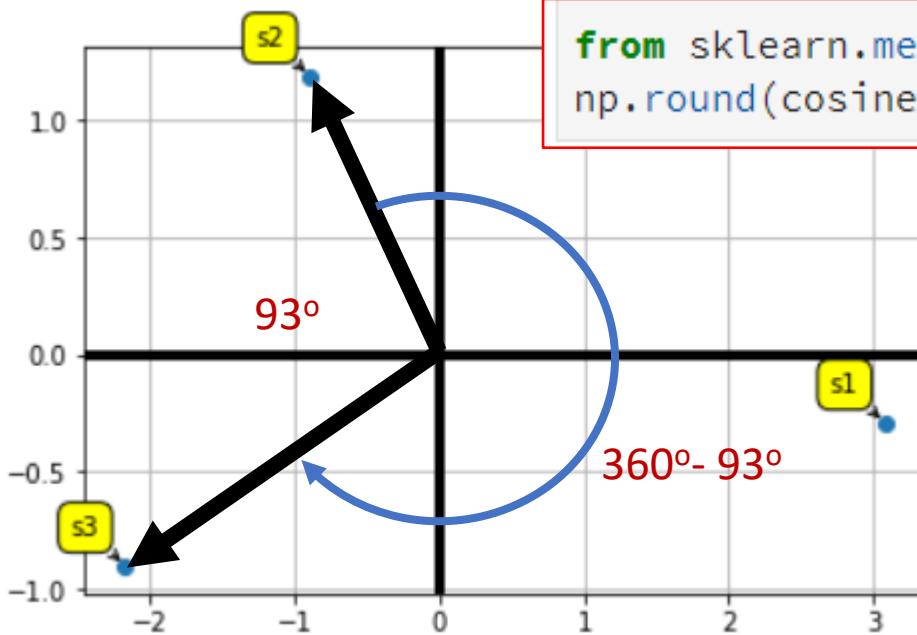
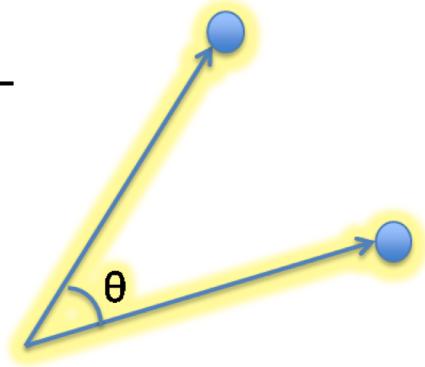
Unrelated

Opposite

6) Similarity

Cosine Similarity

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



```
from sklearn.metrics.pairwise import cosine_similarity  
np.round(cosine_similarity([s1,s2,s3], [s1,s2,s3]),3)
```

	S1	S2	S3
S1	[1.	0.667	0.111]
S2	[0.667	1.	0.]
S3	[0.111	0.	1.]

```
[round(np.cos(np.deg2rad(360-93)),3),round(np.cos(np.deg2rad(93)),3)]
```

[-0.052, -0.052]

S1 to S2 are more similar than S1 to S3, or S2 to S3

6) Similarity

Jaccard Similarity

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{size(Intersection of } A \text{ and } B\text{)}}{\text{size(Union of } A \text{ and } B\text{)}}$$

```
from sklearn.metrics import jaccard_similarity_score
J = np.zeros((3,3))
for idx1,tmps1 in enumerate([s1,s2,s3]):
    for idx2,tmps2 in enumerate([s1,s2,s3]):
        J[idx1,idx2] = round(jaccard_similarity_score(tmps1,tmps2),2)
        J[idx2,idx1] = J[idx1,idx2]
print(J)
```

$s1 = [0, 1, 5, 1, 0, 0]$
 $s2 = [0, 1, 1, 0, 1, 0]$

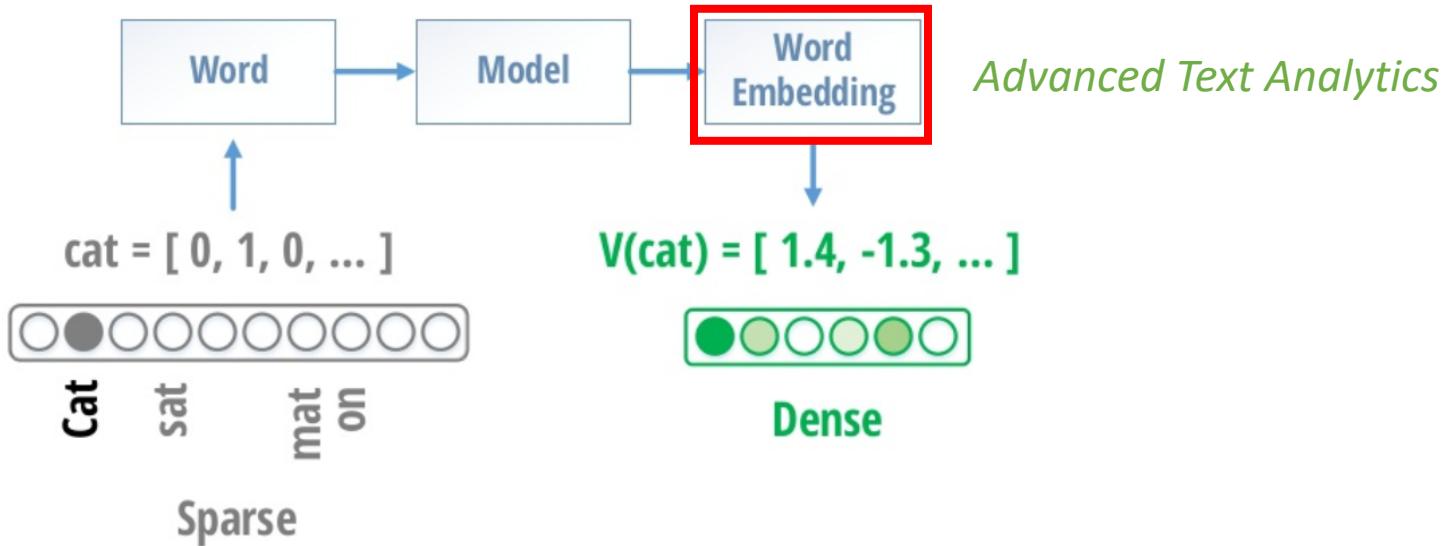
$$\rightarrow J(s1, s2) = \frac{3}{6} = 0.5$$

	S1	S2	S3
S1	[[1.	0.5	0.33]
S2	[0.5	1.	0.]
S3	[0.33	0.	1.]]

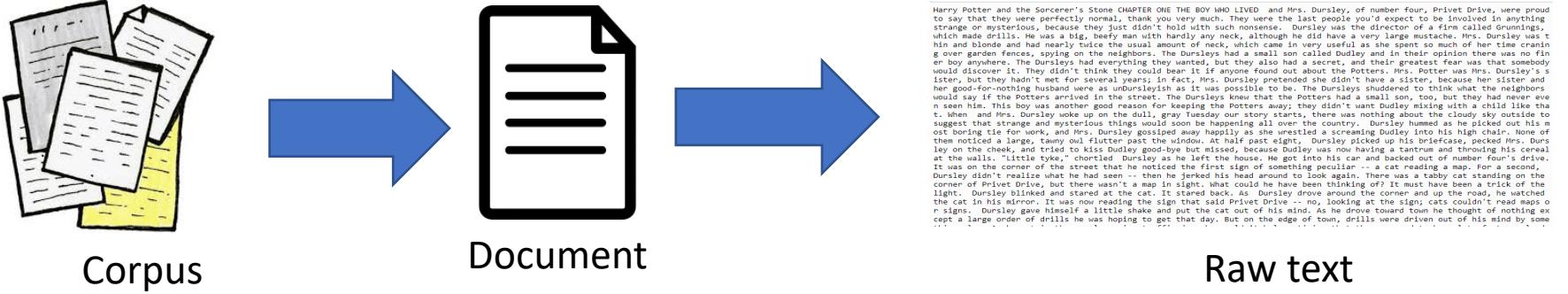
S1 to S2 are more similar than S1 to S3, or S2 to S3

7) Conclusion

- The techniques are **simple** for finding the similarity of two documents if having common words
- Advantage: text is represented as a **vector of numbers**
- Limitation:
 - The techniques do not cover the **synonym scenario** (e.g. [dog, puppy] , [buy, purchase], [funny, amusing], etc.)
 - The vectors of documents are sparse vectors (many zeros)



Summary



Text → Vector

TF-IDF

Information Retrieval

Ranking

Distance & Similarity

The diagram consists of four black arrows originating from the bottom right and pointing towards a large curly brace on the left. The top arrow is labeled "Frequency" in red text. The second arrow from the top is labeled "N-gram" in red text. The third arrow from the top is labeled "POS" in red text. The bottom arrow is unlabeled.

