2020년도
지역 ICT 이노베이션스퀘어 조성 사업
[인공지능 : Python 기반의 Machine / Deep Learning 교육]

사람투자 과학기술정보통신부 nipa 정보통신산업진흥원 National IT Industry Promotion Agency KSA 한국표준협회

Day

22

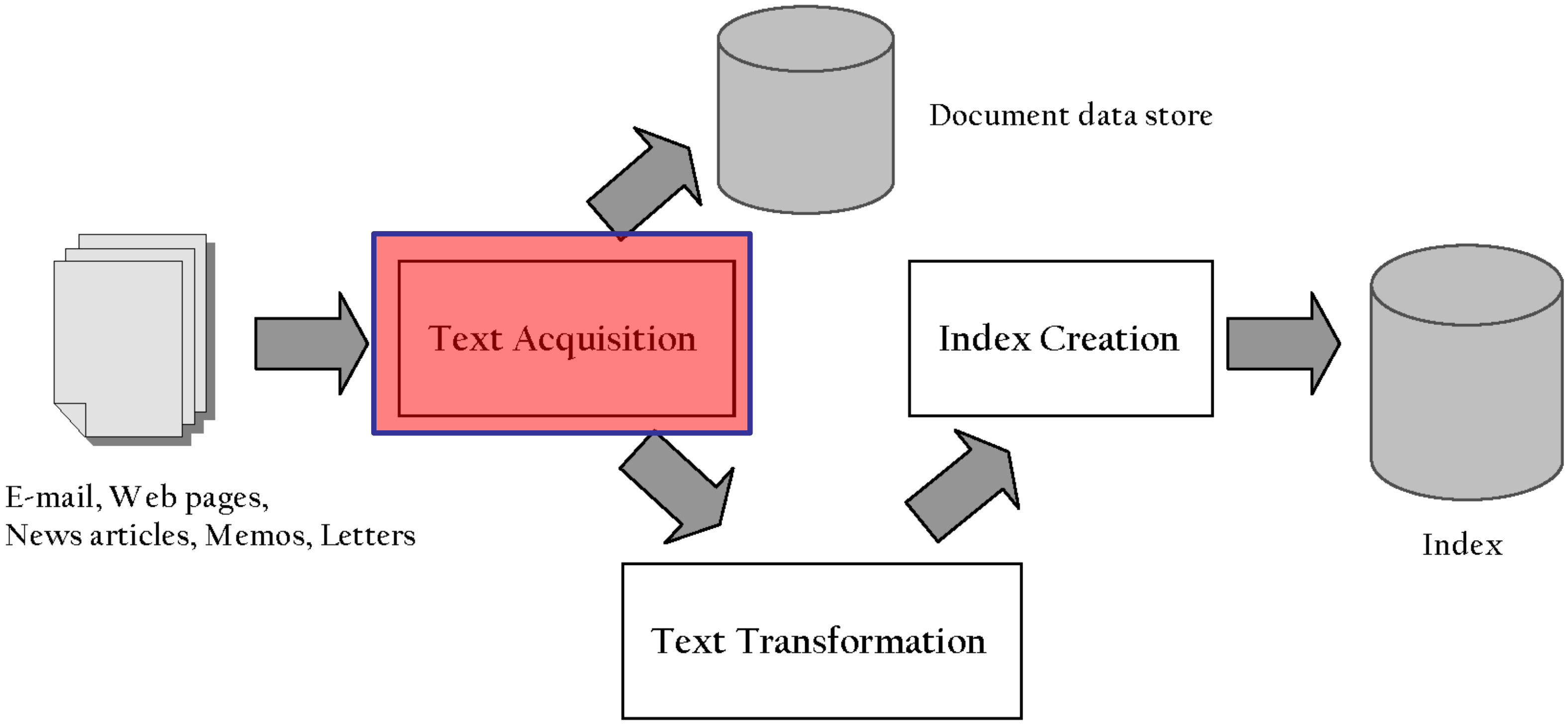
자연언어처리

CONTENTS

Text Acquisition

Text Acquisition

❖ How IR works

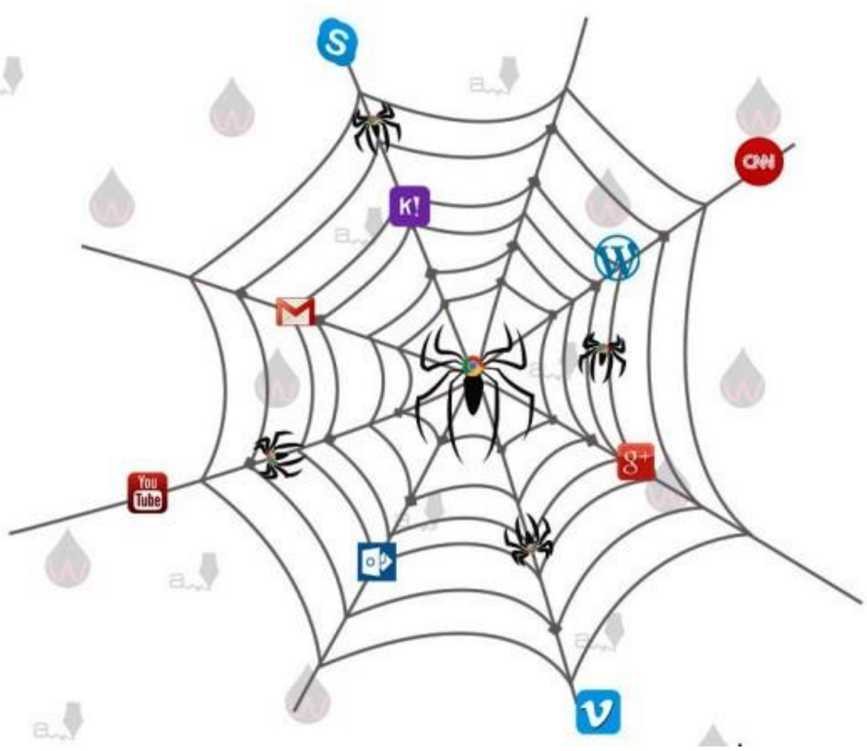


Text Acquisition

- **Text acquisition**
  - identifies and stores documents for indexing
- **Text transformation**
  - transforms documents into index terms or features
- **Index creation**
  - takes index terms and creates data structures (indexes) to support fast searching

Text Acquisition

- **Web Crawler**
  - Finds and downloads web pages automatically



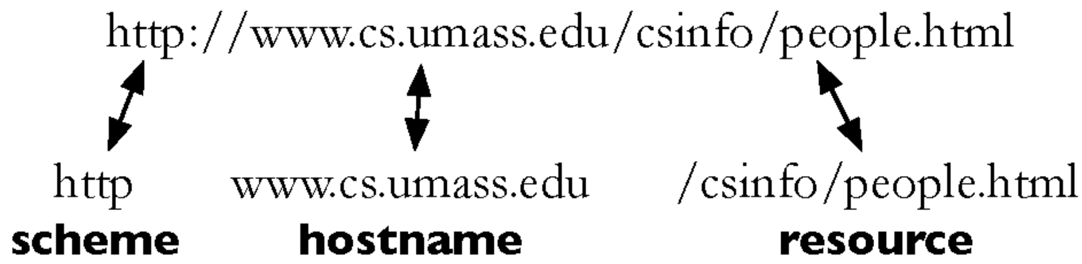
Text Acquisition

❖ Web Crawler

- Every page has a unique *uniform resource locator* (URL)

- Web pages are stored on web servers that use HTTP to exchange information with client software

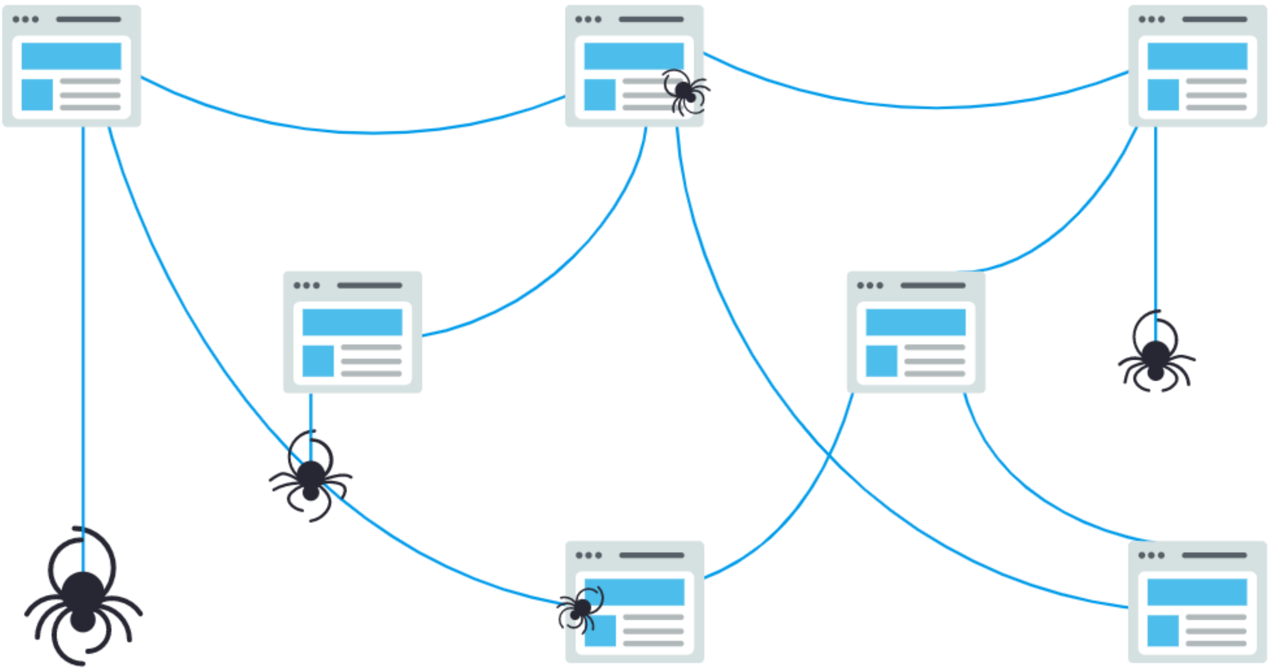
- Example



Text Acquisition

❖ Web Crawler

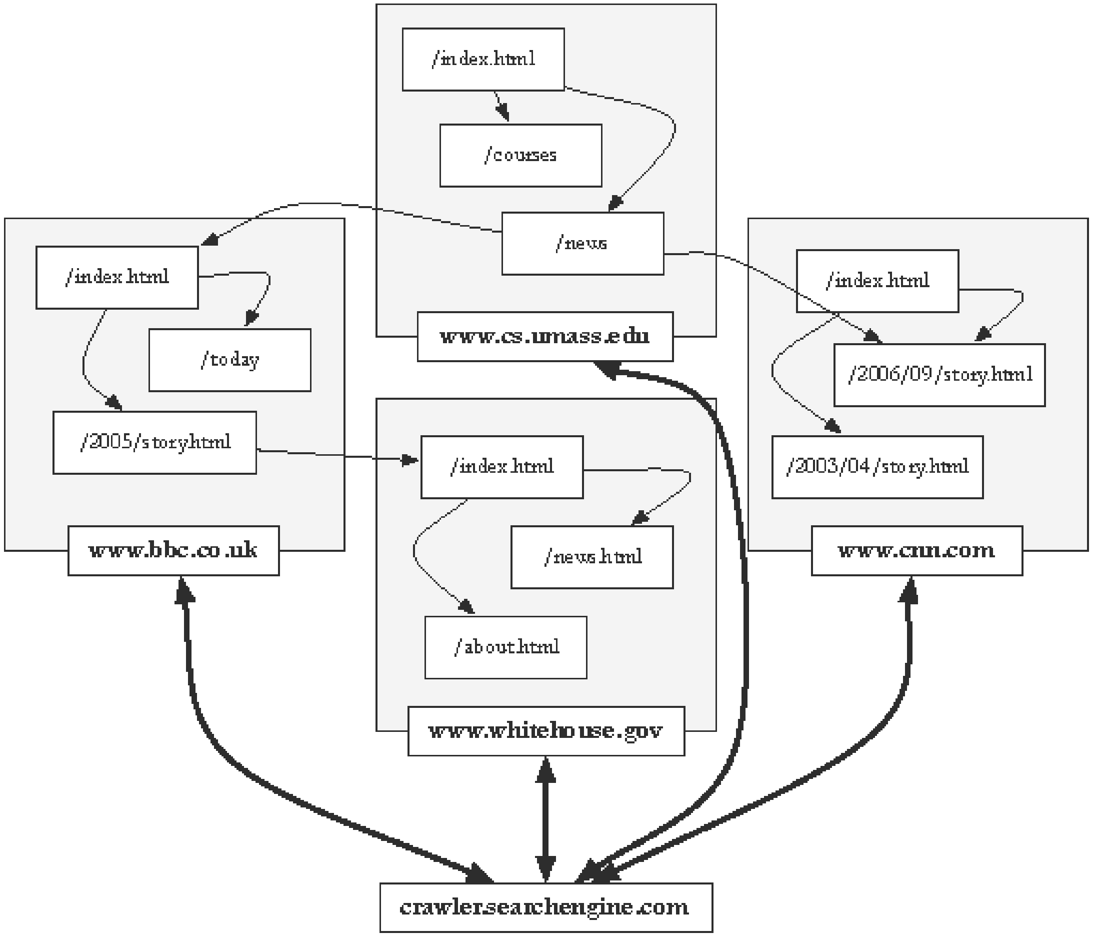
- Example
  - Googlebot starts out by fetching a few web pages, and then follows the links on those webpages to find new URLs



Text Acquisition

❖ Web Crawler

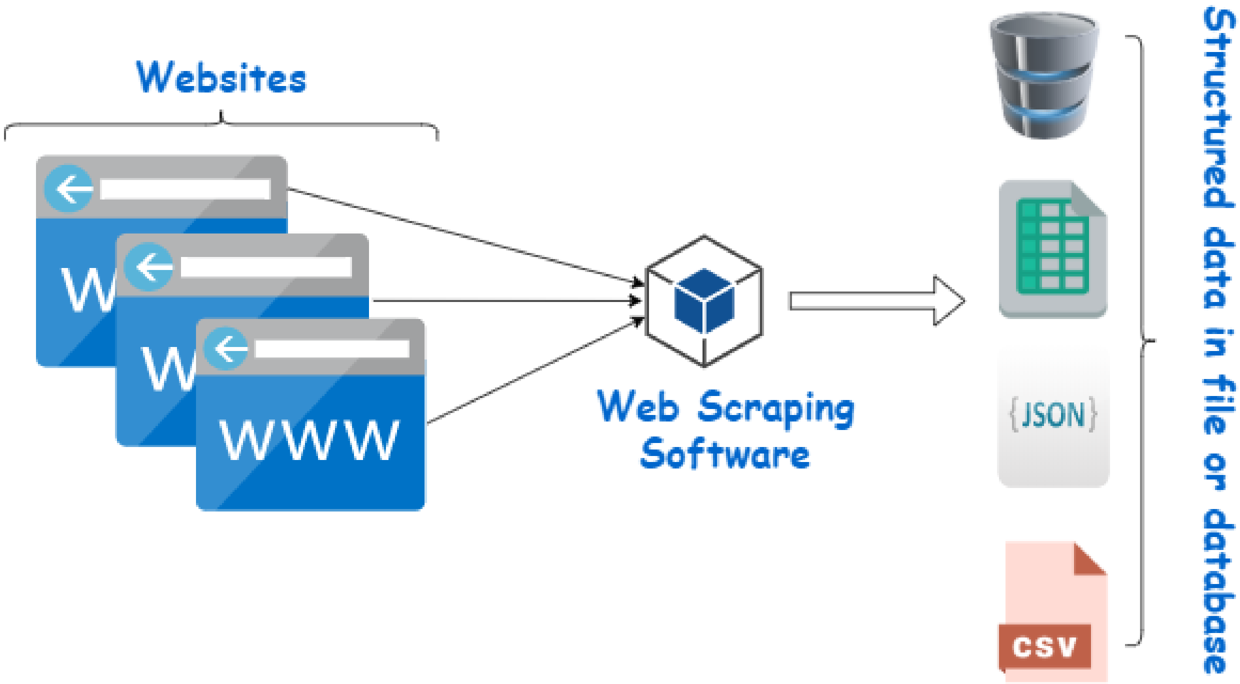
- Example



Text Acquisition

❖ **Web Scrapping**

- Searches for specific information/content on a websites
  - Naver news scrapping
  - Twitter scrapping



Text Acquisition

❖ Web Scrapping in Python

- Every webpage has a HTML structure

```html
<html>
    <head>
        <title>Geekyshows</title>
        <style>
            p.red {color: #FF0000;}
            p.look{font-size: 60px;}
             .algn{text-align: center;}
        </style>
    </head>
    <body>
    <h2 class="algn">I am Heading</h2>
    <p class ="red look">I am first paragraph</p>
    <p class="red algn">I am second paragraph</p>
    </body>
</html>
```

Text Acquisition

❖ Web Scrapping in Python

- Searching by tag

```
from bs4 import BeautifulSoup

import requests

page = requests.get("http://www.naver.com")

soup = BeautifulSoup(page.content, 'html.parser')

title = soup.find('title')

print(title.get_text())
```

Text Acquisition

❖ Web Scrapping in Python

- Searching by class

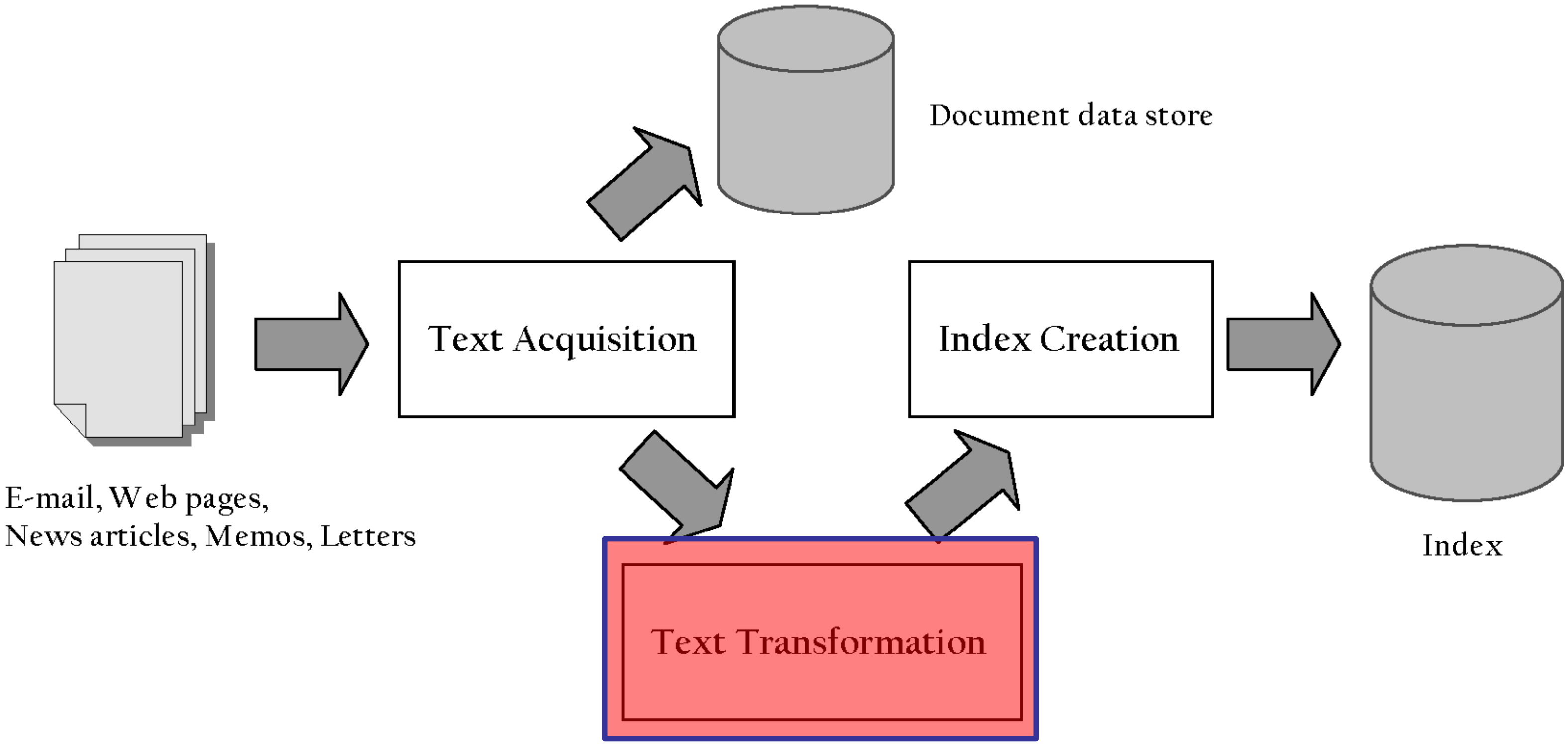
```
from bs4 import BeautifulSoup

import requests

page = requests.get("http://www.naver.com")

soup = BeautifulSoup(page.content, 'html.parser')

links = soup.find_all(class_="an_txt")

for link in links:
    print(link.get_text())
```

B

Text Transformation

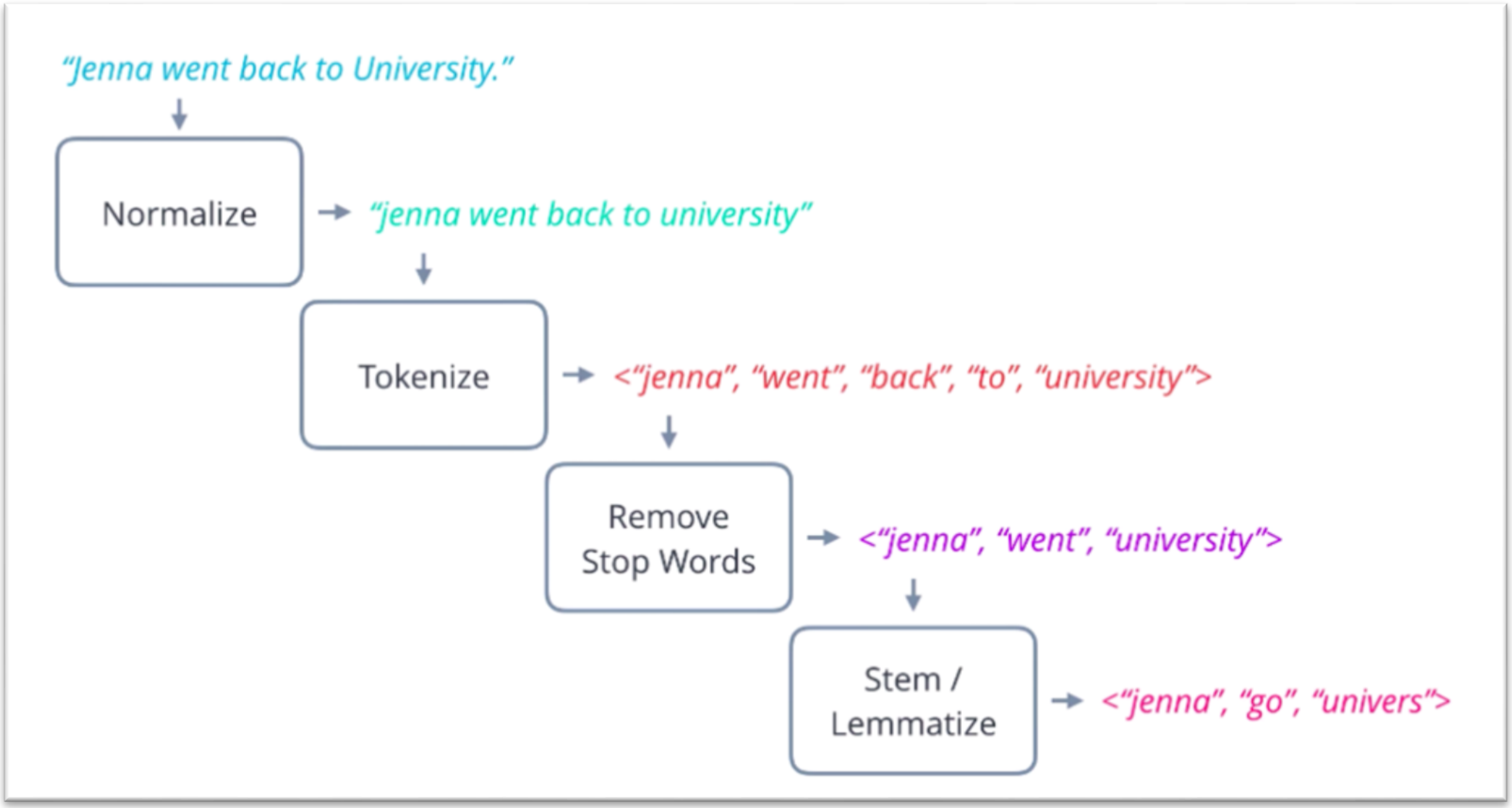
Text Transformation

❖ How IR works



Text Transformation

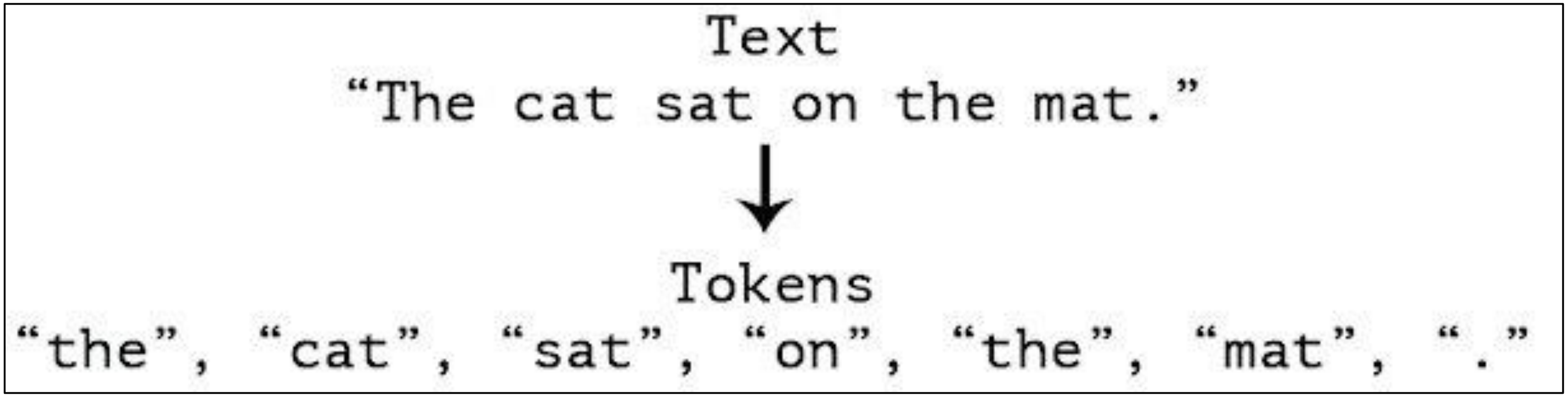
❖ Text transformation process



Text Transformation

❖ Normalizing and Tokenizing

- The task of chopping/breaking down the document up into pieces
  - called tokens



Text Transformation

- **Stopword removal**
  - Top 50 Words from AP89

Word	Freq.	r	$P_r(\%)$	$r.P_r$	Word	Freq	r	$P_r(\%)$	$r.P_r$
the	2,420,778	1	6.49	0.065	has	136,007	26	0.37	0.095
of	1,045,733	2	2.80	0.056	are	130,322	27	0.35	0.094
to	968,882	3	2.60	0.078	not	127,493	28	0.34	0.096
a	892,429	4	2.39	0.096	who	116,364	29	0.31	0.090
and	865,644	5	2.32	0.120	they	111,024	30	0.30	0.089
in	847,825	6	2.27	0.140	its	111,021	31	0.30	0.092
said	504,593	7	1.35	0.095	had	103,943	32	0.28	0.089
for	363,865	8	0.98	0.078	will	102,949	33	0.28	0.091
that	347,072	9	0.93	0.084	would	99,503	34	0.27	0.091
was	293,027	10	0.79	0.079	about	92,983	35	0.25	0.087
on	291,947	11	0.78	0.086	i	92,005	36	0.25	0.089
he	250,919	12	0.67	0.081	been	88,786	37	0.24	0.088
is	245,843	13	0.65	0.086	this	87,286	38	0.23	0.089
with	223,846	14	0.60	0.084	their	84,638	39	0.23	0.089
at	210,064	15	0.56	0.085	new	83,449	40	0.22	0.090
by	209,586	16	0.56	0.090	or	81,796	41	0.22	0.090
it	195,621	17	0.52	0.089	which	80,385	42	0.22	0.091
from	189,451	18	0.51	0.091	we	80,245	43	0.22	0.093
as	181,714	19	0.49	0.093	more	76,388	44	0.21	0.090
be	157,300	20	0.42	0.084	after	75,165	45	0.20	0.091
were	153,913	21	0.41	0.087	us	72,045	46	0.19	0.089
an	152,576	22	0.41	0.090	percent	71,956	47	0.19	0.091
have	149,749	23	0.40	0.092	up	71,082	48	0.19	0.092
his	142,285	24	0.38	0.092	one	70,266	49	0.19	0.092
but	140,880	25	0.38	0.094	people	68,988	50	0.19	0.093

Text Transformation

❖ Stopword removal

- Function words (determiners, prepositions) have little meaning on their own
- High occurrence frequencies
- Treated as stopwords (i.e. removed)
  - reduce index space, improve response time, improve effectiveness

Text Transformation

❖ Stopword removal

- List of stopwords in spacy library

```
In [23]: import spacy
         from spacy.lang.en.stop_words import STOP_WORDS
         STOP_WORDS -= {"Test_One","Test_Two"}

         print(len(STOP_WORDS))
         print(STOP_WORDS)
```

```
312
{'itself', 'how', 'two', 'eight', 'five', 'never', 'but', 'from', 'please', 'along', 'whereupon', 'not', 'more', 'few', 'if',
'noone', 'part', 'she', 'there', 'say', 'which', 'seem', 'however', 'each', 'being', 'many', 'others', 'with', 'through', 'seem
ed', 'yours', 'down', 'almost', 'nobody', 'only', 'side', 'than', 'thereupon', 'they', 'became', 'give', 'either', 'least', 'tw
enty', 'fifteen', 'afterwards', 'is', 'would', 'a', 'sometimes', 'show', 'his', 'for', 'someone', 'yet', 'behind', 'her', 'hers
elf', 'was', 'this', 'made', 'themselves', 'anything', 'thereafter', 'myself', 'among', 'therein', 'three', 'top', 'am', 'nex
t', 'fifty', 'around', 'become', 'bottom', 'between', 'due', 'same', 'while', 'or', 'mostly', 'him', 'everyone', 'herein', 'doe
s', 'why', 'call', 'throughout', 'your', 'very', 'mine', 'latterly', 'across', 'ours', 'alone', 'name', 'somewhere', 'neither',
'at', 'first', 'just', 'us', 'own', 'might', 'everywhere', 'together', 'no', 'forty', 'go', 'whenever', 'whoever', 'whereafte
r', 'then', 'enough', 'seems', 'off', 'beyond', 'though', 'when', 'keep', 'could', 'within', 'other', 'may', "'ve", 'upon', 'di
d', 'its', 'whereas', 'himself', 'take', 'these', 'doing', 'can', 'various', 'anyone', 'below', 'nine', 'during', 'any', 'anyho
w', 'becoming', 'has', 'put', 'whence', 'hereby', 'towards', 'even', 'without', 'get', 'most', 'about', 'out', 'besides', 'ever
y', 'several', 'used', 'wherever', 'one', 'into', 'already', 'must', "'m", "'d", 'less', 'move', 'anywhere', 'once', 'up', 'non
e', 'where', 'full', 'eleven', 'ca', 'rather', 'thru', 'thence', 'those', 'have', 'nor', 'thus', 'were', 'whom', 'per', 'as',
'hundred', 'perhaps', 'still', 'above', 'on', "'re", 'wherein', 'their', 'some', 'onto', 'anyway', 'everything', 'in', 'amongs
t', 'formerly', 'front', 'meanwhile', 'namely', 'becomes', 'nothing', 'really', 'further', 'somehow', 'sometime', 'twelve', 'an
other', 'whose', 'much', 'because', 'former', 'who', 'before', 'although', 'an', 'ten', 'therefore', 'four', 'using', 'quite',
'often', 'our', 'except', 'ever', 'last', 'after', 'regarding', 'here', 'he', 'yourself', 'also', 'whether', "'ll", 'do', 'alwa
ys', 'been', 'six', 'whereby', 'you', 'to', 'what', 'latter', 'empty', 'unless', 'back', 'now', 'beforehand', 'whatever', 'ar
e', 'such', 'make', 'since', 'cannot', 'will', 'elsewhere', 'that', 'see', 're', 'too', 'moreover', 'whole', 'beside', 'whithe
r', 'hence', 'hereupon', 'it', 'third', 'done', 'by', 'under', 'yourselves', 'both', 'otherwise', 'we', 'should', 'all', 'had',
'hers', 'so', 'and', 'hereafter', 'me', 'toward', "n't", 'serious', 'sixty', 'seeming', 'be', 'something', 'until', 'amount',
'indeed', 'via', "'s", 'nowhere', 'ourselves', 'my', 'thereby', 'well', 'against', 'of', 'again', 'them', 'else', 'the', 'i',
'over', 'nevertheless'}
```

Text Transformation

❖ Stopword removal

- Example

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

- Online tools
  - https://tools.fromdev.com/remove-stopwords-online.html

Text Transformation

❖ Stemming

- In linguistics, a stem is part of a word, that is common to all of its inflected variants
  - CONNECT
  - CONNECTED
  - CONNECTION
  - CONNECTING

- Word = Stem + Suffix(es)
  - E.g., connection = connect + tion
  - E.g., generalizations = general + ization + s

- The Porter Stemming algorithm (or Porter Stemmer) is used to remove the suffixes from an English word and obtain its stem

Text Transformation

Stemming

- A consonant in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant.
  - So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G.

- If a letter is not a consonant it is a vowel.
  - B, C, D, F, G, H, J, K, L, M, N, P, Q, R, S, T, V, W, X, Z, and usually Y

- Stemming notation
  - A consonant will be denoted by c, a vowel by v
    - A list of one or more consecutive consonants (ccc…) will be denoted by C
    - A list of one or more consecutive vowels (vvv…) will be denoted by V

- Examples
  - Tree -> CV
  - Trouble -> CVCV
  - Troubles -> CVCVC

Text Transformation

❖ Stemming

- Any word, or part of a word, therefore has one of the four forms given below.
  - CVCV ... C
  - CVCV ... V
  - VCVC ... C
  - VCVC ... V

❖ These may all be represented by the single form

- [C]VCVC … [V]

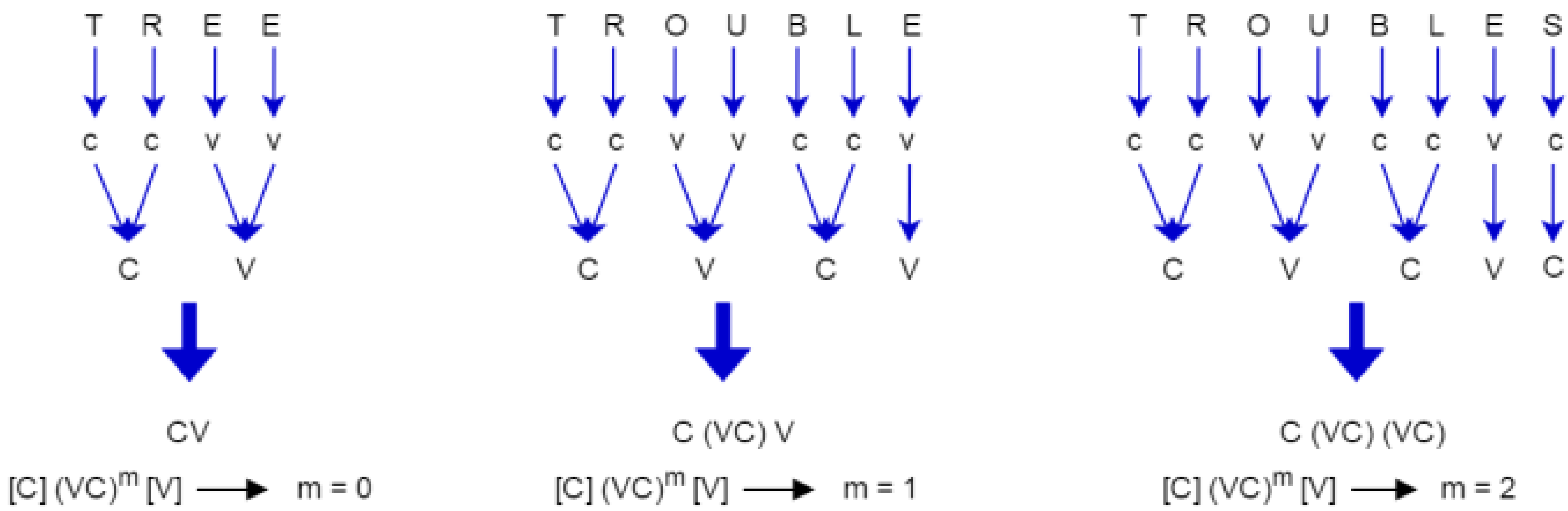
❖ Using $(VC)^m$ to denote VC repeated m times, this may again be written as

- $[C](VC)^m[V]$

Text Transformation

❖ Stemming

- m will be called the measure of any word or word part when represented in this form.



Text Transformation

❖ Stemming

- The rules for removing a suffix will be given in the form
  - (condition) S1 -> S2
- This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition
- The condition is usually given in terms of m.

- Example
  - (m > 1) EMENT ->
    - Here S1 is ‘EMENT’ and S2 is null.
  - This would change REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2

Text Transformation

Stemming

- The ‘condition’ part may also contain the following:
  - *S - the stem ends with S (and similarly for the other letters).
  - *v* - the stem contains a vowel.
  - *d - the stem ends with a double consonant (e.g. -TT, -SS).
  - *o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).
  - We can also have cases when condition is not given

Text Transformation

Stemming

- Step 1a
  - SSES -> SS
    - caresses -> caress
  - IES -> I
    - ponies -> poni
    - ties -> ti
  - SS -> SS
    - caress -> caress
  - S ->
    - cats -> cat

Text Transformation

Stemming

- Step 1b
  - (m>0) EED -> EE
    - feed -> feed
    - agreed -> agree
  - (*v*) ED ->
    - plastered -> plaster
    - bled -> bled
  - (*v*) ING ->
    - motoring -> motor
    - sing -> sing

Text Transformation

❖ Stemming

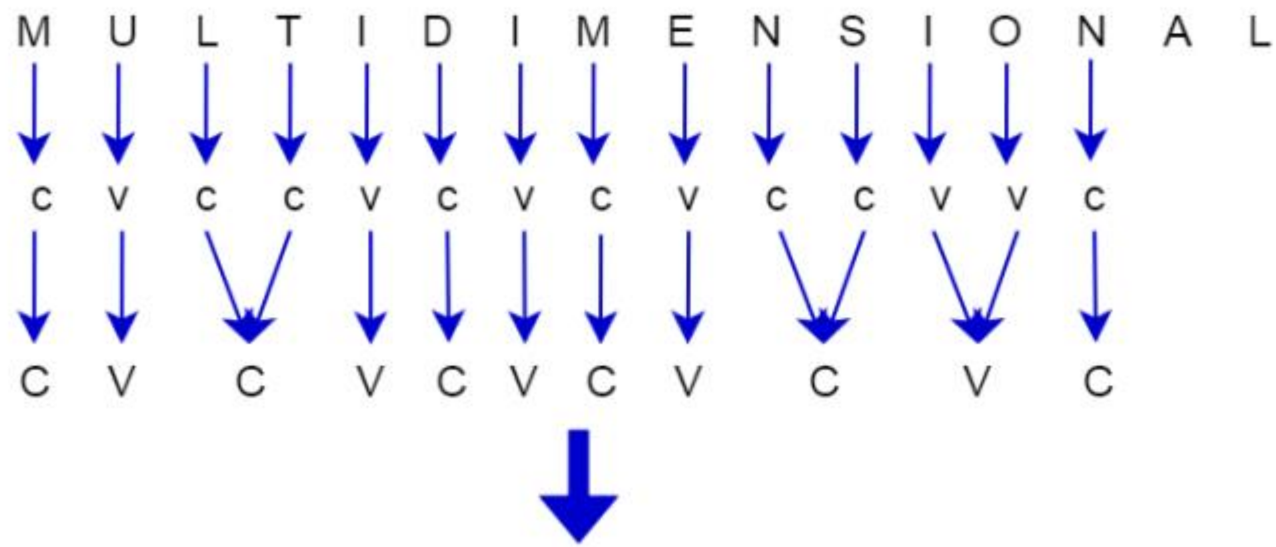
- Step 1b
  - AT -> ATE
    - conflat(ed) -> conflate
  - BL → BLE
    - troubl(ed) -> trouble
  - IZ → IZE
    - siz(ed) -> size
  - (*d and not (*L or *S or *Z)) → single letter
    - fall(ing) -> fall
  - (m=1 and *o) → E
    - fail(ing) -> fail

Text Transformation

- **Stemmer**
  - The rest of the rules can be found here:
    - http://snowball.tartarus.org/algorithms/porter/stemmer.html

Text Transformation

- **Stemming**
  - Example



C (VC) (VC) (VC) (VC) (VC)

$[C]\,(VC)^m\,[V] \longrightarrow m = 5$

Text Transformation

❖ Text transformation in Python

- Tokenizing

```
from nltk.tokenize import sent_tokenize, word_tokenize

text = "Natural language processing (NLP) is a field " + ₩
       "of computer science, artificial intelligence " + ₩
       "and computational linguistics concerned with " + ₩
       "the interactions between computers and human " + ₩
       "(natural) languages, and, in particular, " + ₩
       "concerned with programming computers to " + ₩
       "fruitfully process large natural language " + ₩
       "corpora. "

print(sent_tokenize(text))
print(word_tokenize(text))
```

Text Transformation

❖ Text transformation in Python

- Stopword Removal

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example_sent = "This is a sample sentence, showing off the stop words filtration.
"
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(word_tokens)
print(filtered_sentence)
```

Text Transformation

❖ Text transformation in Python

- Stemming

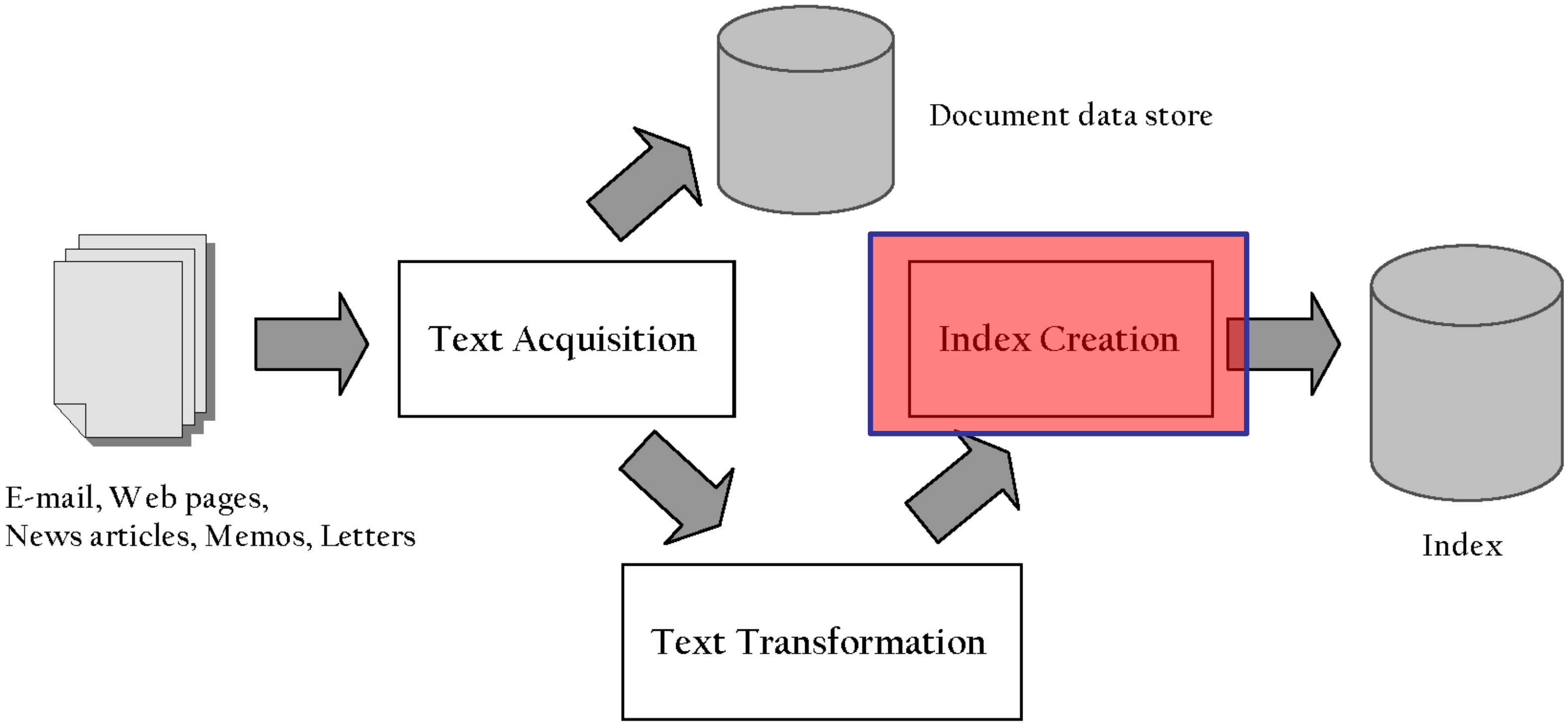
```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()

# choose some words to be stemmed
words = ["program", "programs", "programer", "programing", "programers"]

for w in words:
    print(w, " : ", ps.stem(w))
```

C

Index Creation

Index Creation

❖ How IR works



Index Creation

- **The purpose of IR is to create index**
  - Indexes are a specialized data structure designed to make search faster

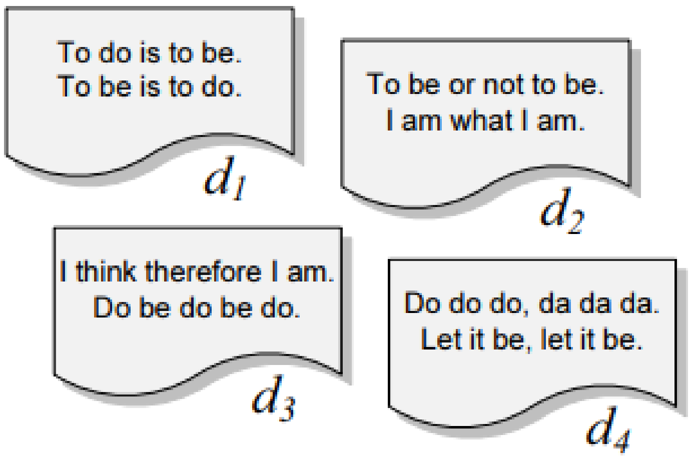
INDEX

Index Creation

❖ Term-document matrix

- The simplest way to represent the documents that contain each word of the vocabulary (sometimes called dictionary)

Vocabulary	n_i	d_1	d_2	d_3	d_4
to	2	4	2	-	-
do	3	2	-	3	3
is	1	2	-	-	-
be	4	2	2	2	2
or	1	-	1	-	-
not	1	-	1	-	-
I	2	-	2	2	-
am	2	-	2	1	-
what	1	-	1	-	-
think	1	-	-	1	-
therefore	1	-	-	1	-
da	1	-	-	-	3
let	1	-	-	-	2
it	1	-	-	-	2



Index Creation

❖ Problem of term-document matrix

- The main problem of this simple solution is that it requires too much space

- Example
  - Suppose that we have 1 million documents
    - Currently, 4.2 billion pages on Internet
  - For 1 million documents, there are about 500,000 distinct terms
  - We will come up with a sparse matrix (500k x 1m) with most of the numbers are zeros
    - Too many to fit in a computer's memory

Index Creation

❖ Inverted index

- The inverted index structure is composed of two elements
  - Vocabulary
  - Occurrences (Also called postings)

- Example

S_1	Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.
S_2	Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.
S_3	Tropical fish are popular aquarium fish, due to their often bright coloration.
S_4	In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Four sentences from the Wikipedia entry for *tropical fish*

Index Creation

❖ **Inverted index**

Inverted Index
with counts

- supports better ranking algorithms

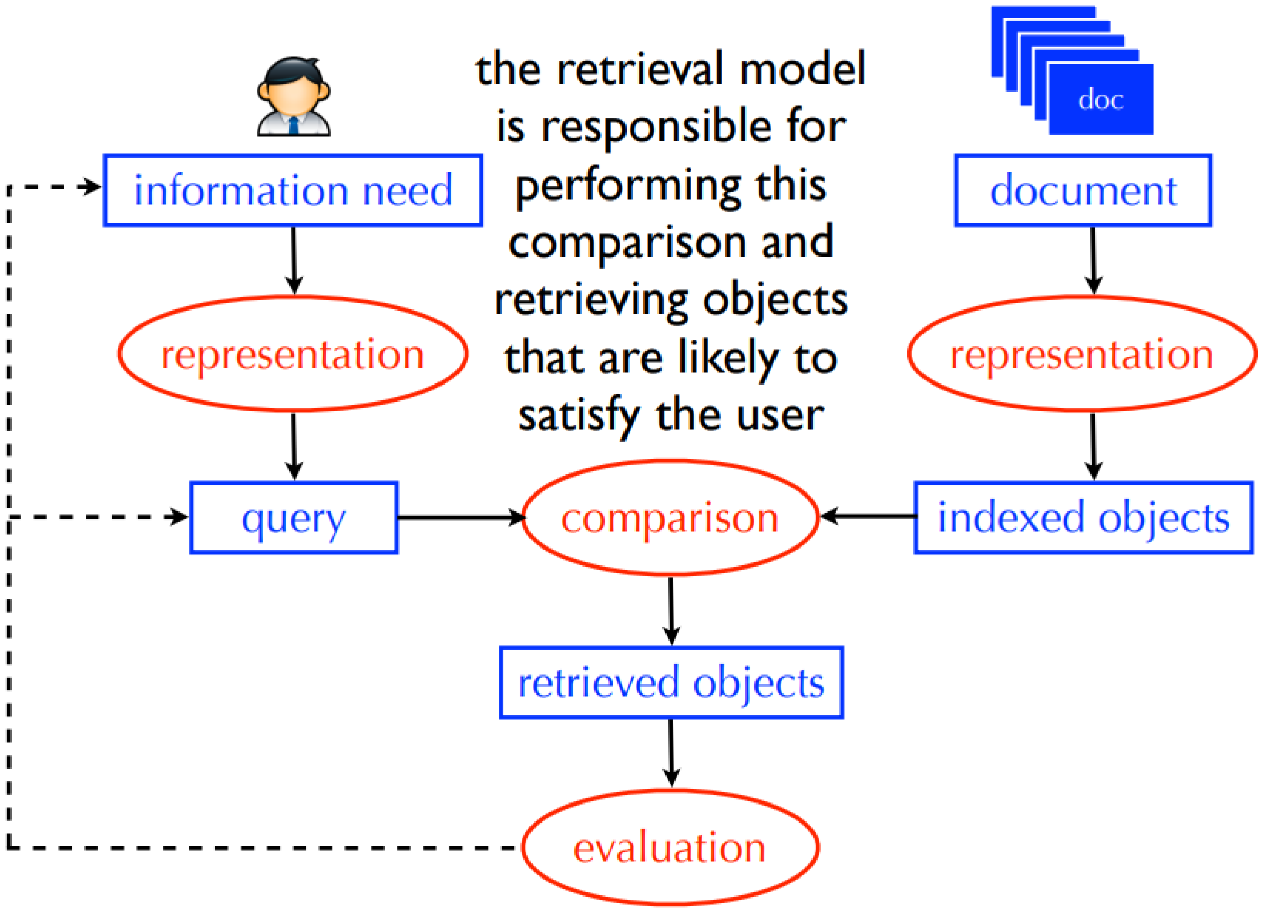
and	1:1
aquarium	3:1
are	3:1, 4:1
around	1:1
as	2:1
both	1:1
bright	3:1
coloration	3:1, 4:1
derives	4:1
due	3:1
environments	1:1
fish	1:2, 2:3, 3:2, 4:2
fishkeepers	2:1
found	1:1
fresh	2:1
freshwater	1:1, 4:1
from	4:1
generally	4:1
in	1:1, 4:1
include	1:1
including	1:1
iridescence	4:1
marine	2:1
often	2:1, 3:1
only	2:1
pigmented	4:1
popular	3:1
refer	2:1
referred	2:1
requiring	2:1
salt	1:1, 4:1
saltwater	2:1
species	1:1
term	2:1
the	1:1, 2:1
their	3:1
this	4:1
those	2:1
to	2:2, 3:1
tropical	1:2, 2:2, 3:1
typically	4:1
use	2:1
water	1:1, 2:1, 4:1
while	4:1
with	2:1
world	1:1

D

Information Retrieval

Information Retrieval

❖ Information retrieval process



Information Retrieval

❖ Boolean retrieval

- The Boolean Retrieval is arguably the simplest model to base on information retrieval system on.

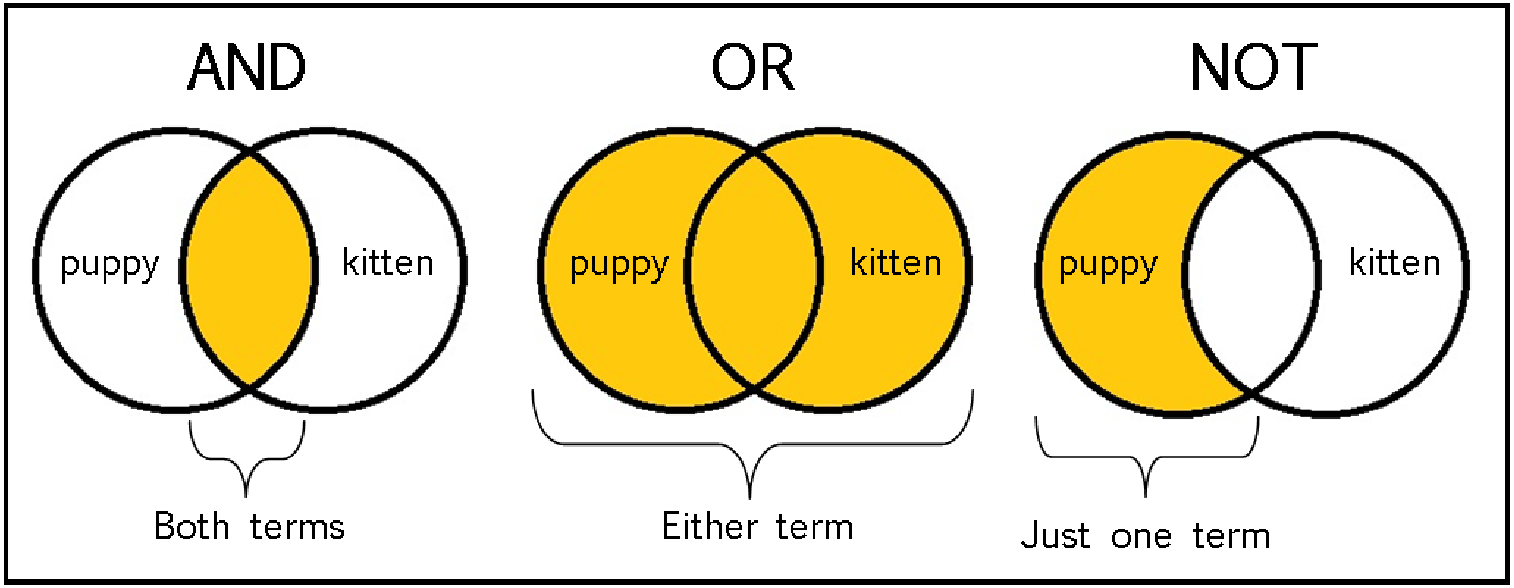
- Queries are based on Boolean logic
  - AND, OR, NOT
    - E.g., "Frodo" AND "Sam"

- The search engine returns all documents from the collection that satisfy the Boolean expression

- Does Google use Boolean Retrieval?
  - Google's default interpretation of the query "Frodo gave Sam the sword" is "Frodo" AND "gave" AND "Sam" AND "sword"

Information Retrieval

❖ Boolean Retrieval



Information Retrieval

❖ Boolean retrieval

- Let us have the following set of index terms
  - K = { “Frodo”, “Sam”, “blue”, “sword”, “orc”, “Mordor”}

- Let us have the following collection of documents
  - d1: “Frodo stabbed the orc with the red sword”
  - d2: “Frodo and Sam used the blue lamp to locate orcs”
  - d3: “Sam killed many orcs in Mordor with the blue sword”

- Which documents are relevant for the following queries?
  - q1: (“Frodo” AND “orc” AND “sword”) OR (“Frodo” AND “blue”)
    - {d1, d2}
  - q2: (“Sam” AND “blue” AND NOT “Frodo”) OR (“Sam” AND “orc” AND “Mordor”)
    - {d3}

Information Retrieval

❖ Boolean retrieval

- Attempt 2: Use Term-Document Matrix
  - q3: "Sam" AND "blue" AND NOT "Frodo"

Term	d1: „Frodo stabbed the orc with the red sword"	d2: Frodo and Sam used the blue lamp to locate orcs	d3: Sam killed many orcs in Mordor with the blue sword
Frodo	True (1)	True (1)	False (0)
Sam	False (0)	True (1)	True (1)
blue	False (0)	True (1)	True (1)
sword	True (1)	False (0)	True (1)
orc	True (1)	True (1)	True (1)
Mordor	False (0)	False (0)	True (1)

- "Sam" : d1 – False; d2 – True; d3 – True -> [0, 1, 1]
- "blue" : d1 – False; d2 – True; d3 – True -> [0, 1, 1]
- "Frodo" : d1 –True; d2 – True; d3 – False -> [1, 1, 0]

Information Retrieval

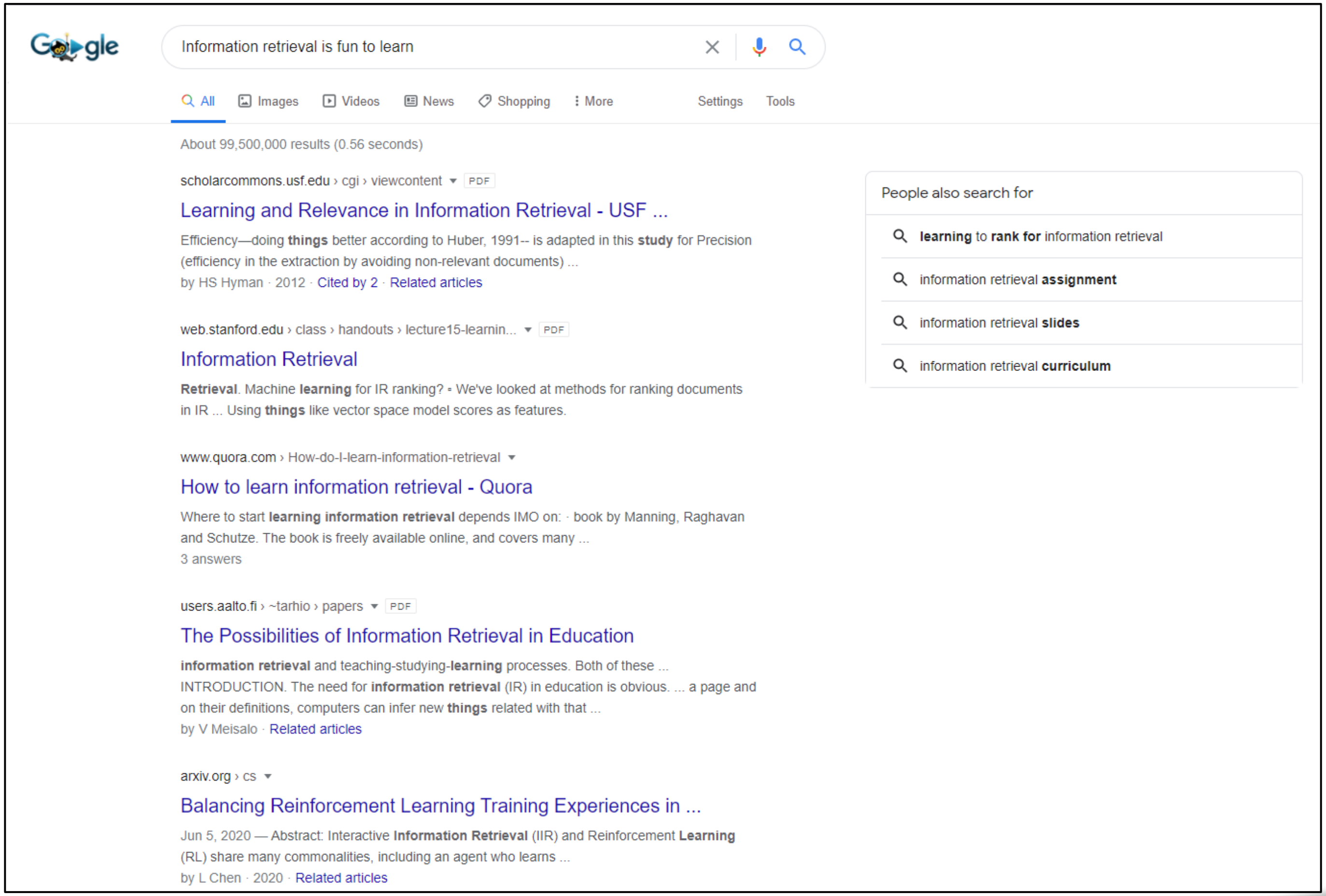
Disadvantages of Boolean Retrieval Model

- Very rigid: AND means all, OR means any
- Similarity function is Boolean
  - Exact-match only, no partial matches
  - Retrieved documents not ranked
- All terms are equally important
  - Boolean operator usage has much more influence than a critical word
- Query language is expressive but complicated

Information Retrieval

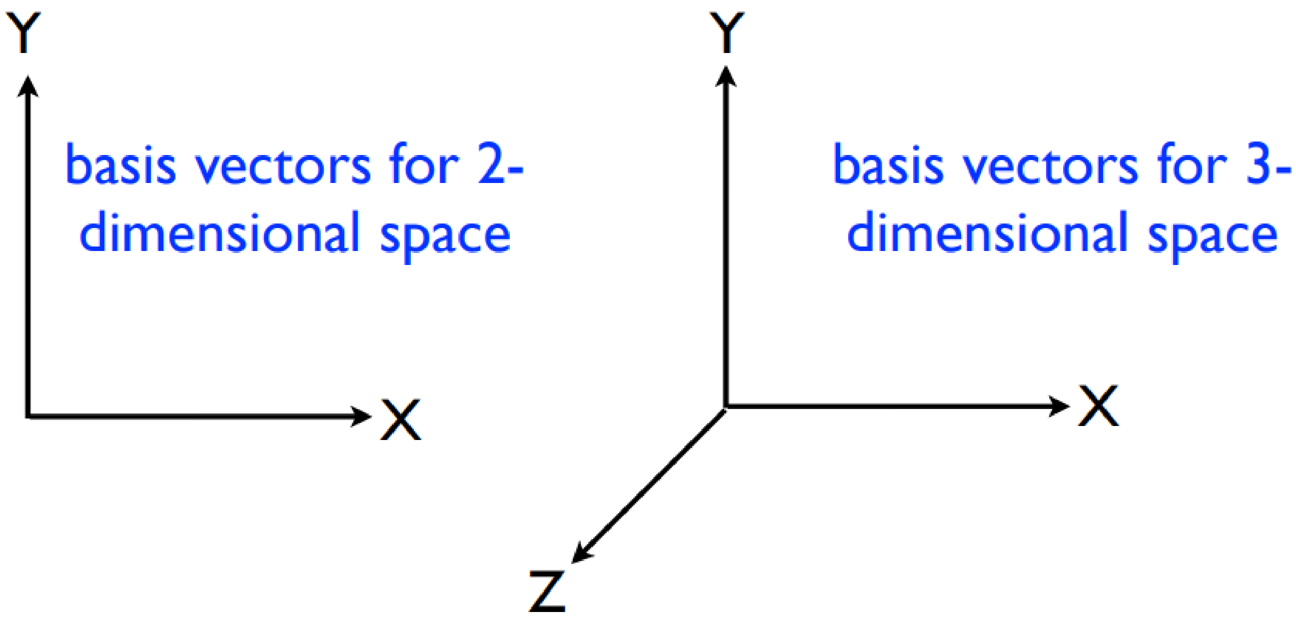
Disadvantages of Boolean Retrieval Model

- Exact match vs. partial match



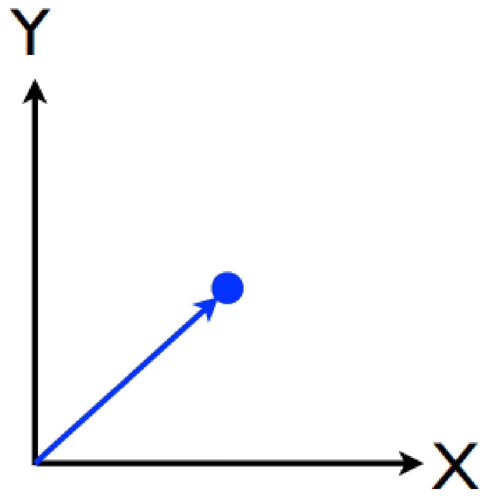
Vector Space Model

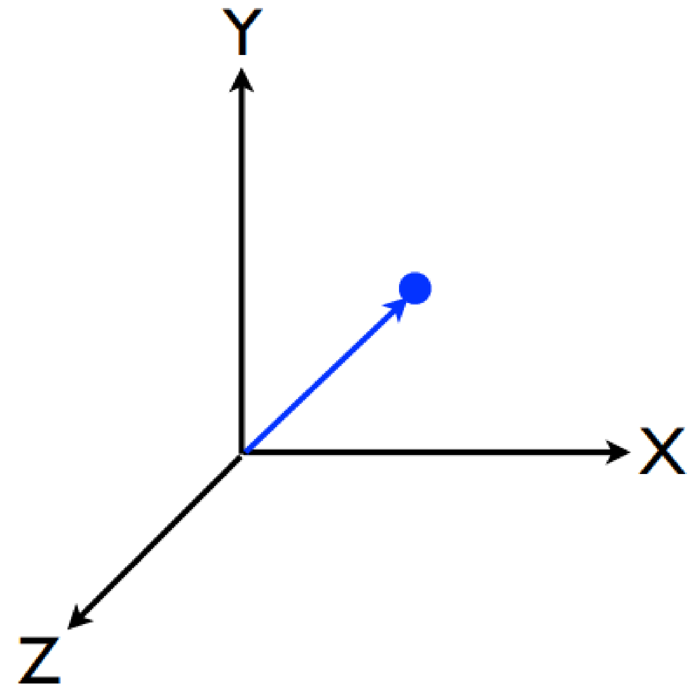
- **Vector space model**
  - Represents text documents as vectors
- **The basis vectors correspond to the dimensions or directions of the vector space**



Vector Space Model

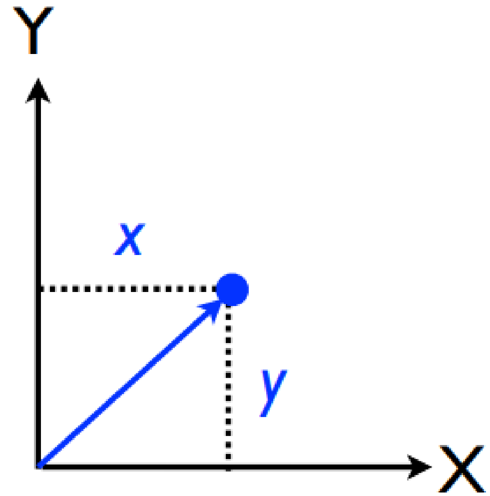
❖ **A vector is a point in a vector space and has length (from the origin to the point) and direction**

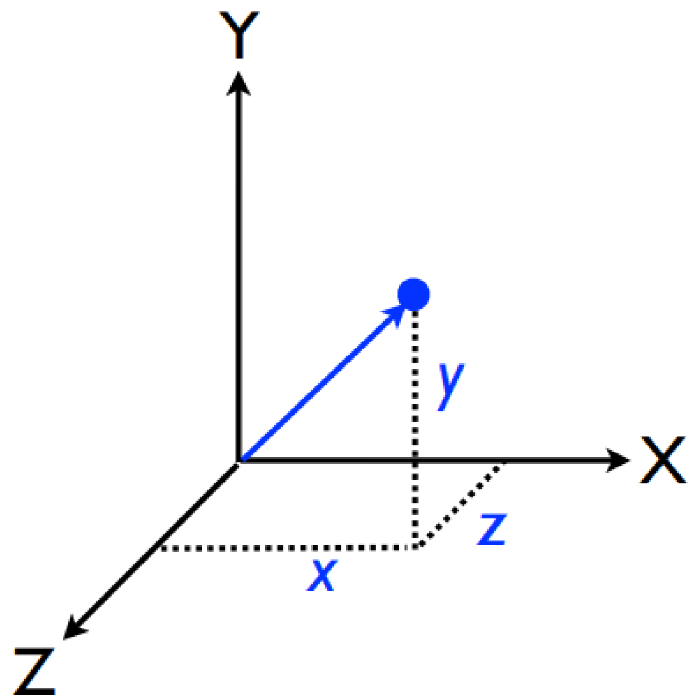




Vector Space Model

- **A 2-dimensional vector can be written as [x,y]**
- **A 3-dimensional vector can be written as [x,y,z]**





Vector Space Model

- **Let V denote the size of the indexed vocabulary**
  - V = the number of unique terms
  - V = the number of unique terms excluding stopwords
  - V = the number of unique stems, etc

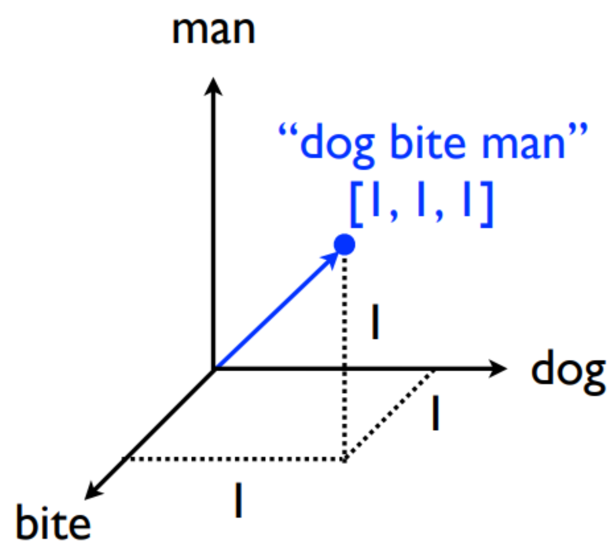
- **Any arbitrary span of text (i.e., a document, or a query) can be represented as a vector in V-dimensional space**

- **For simplicity, let's assume three index terms: dog, bite, man**
  - i.e., V=3

Vector Space Model

- **1 = the term appears at least once**
- **0 = the term does not appear**

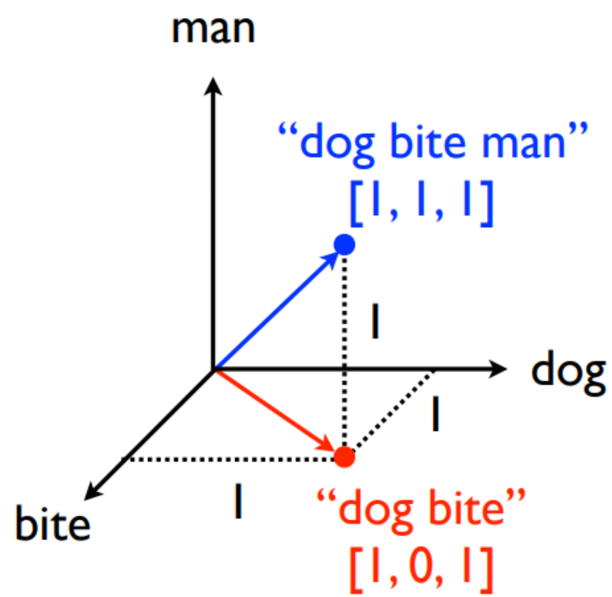
	dog	*man*	*bite*
doc_1	1	1	1



Vector Space Model

- 1 = the term appears at least once
- 0 = the term does not appear

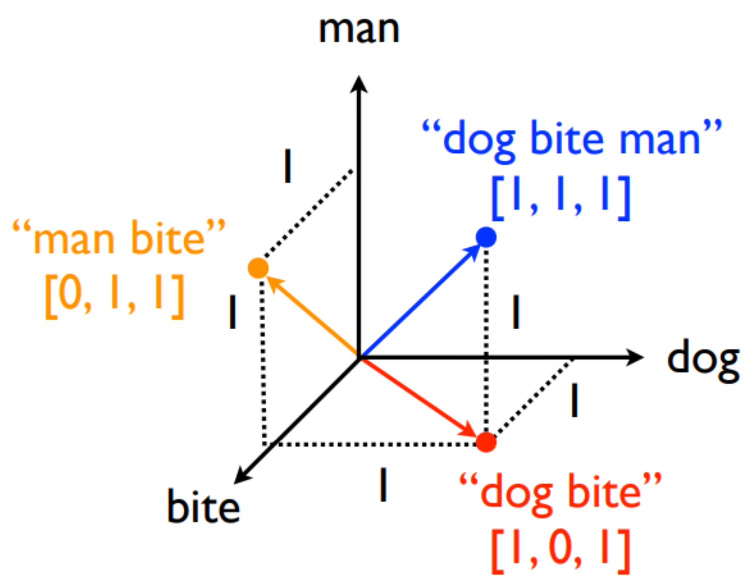
	dog	man	bite
doc_1	1	1	1
doc_2	1	0	1



Vector Space Model

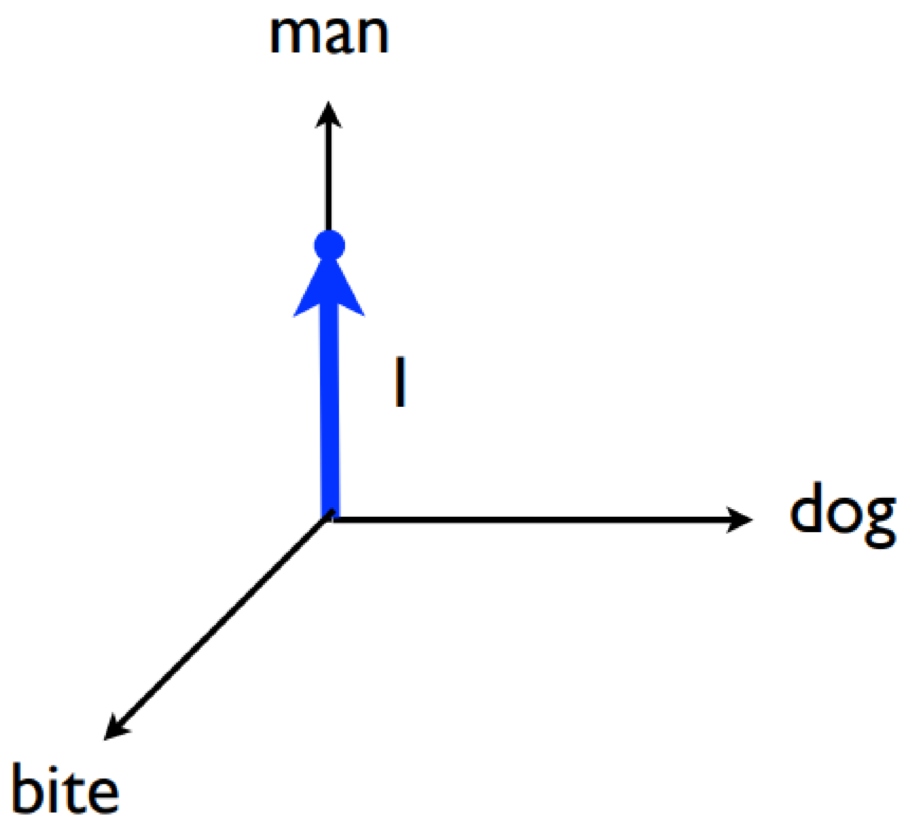
- 1 = the term appears at least once
- 0 = the term does not appear

	dog	*man*	*bite*
doc_1	1	1	1
doc_2	1	0	1
doc_3	0	1	1



Vector Space Model

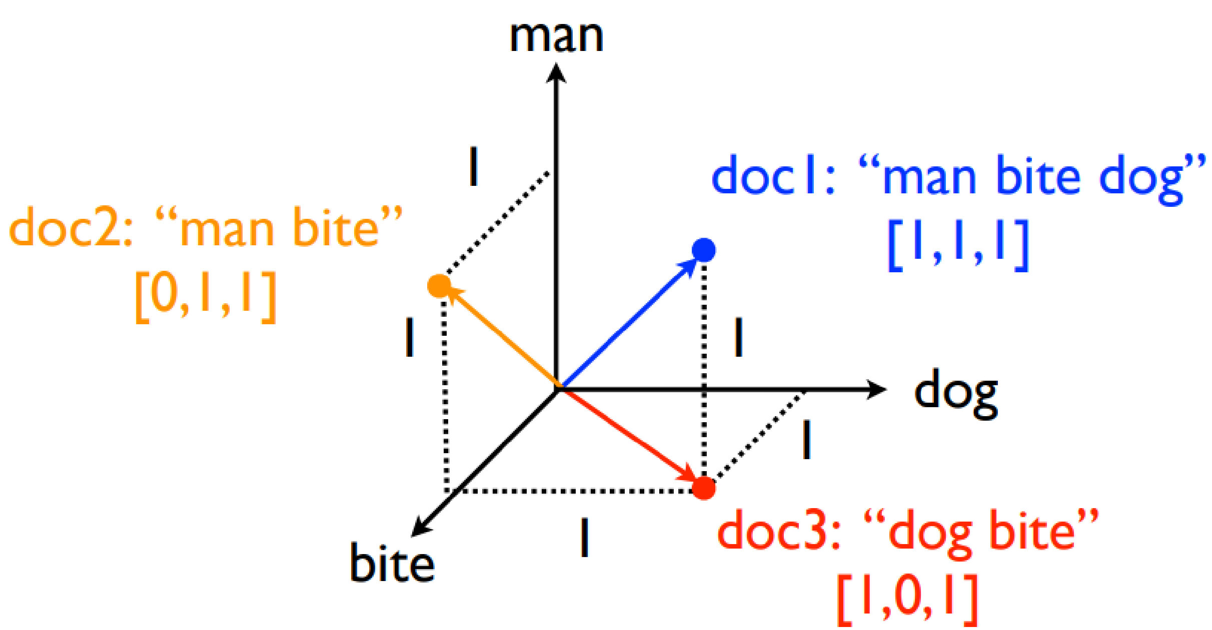
- What span(s) of text does this vector represent?



Vector Space Model

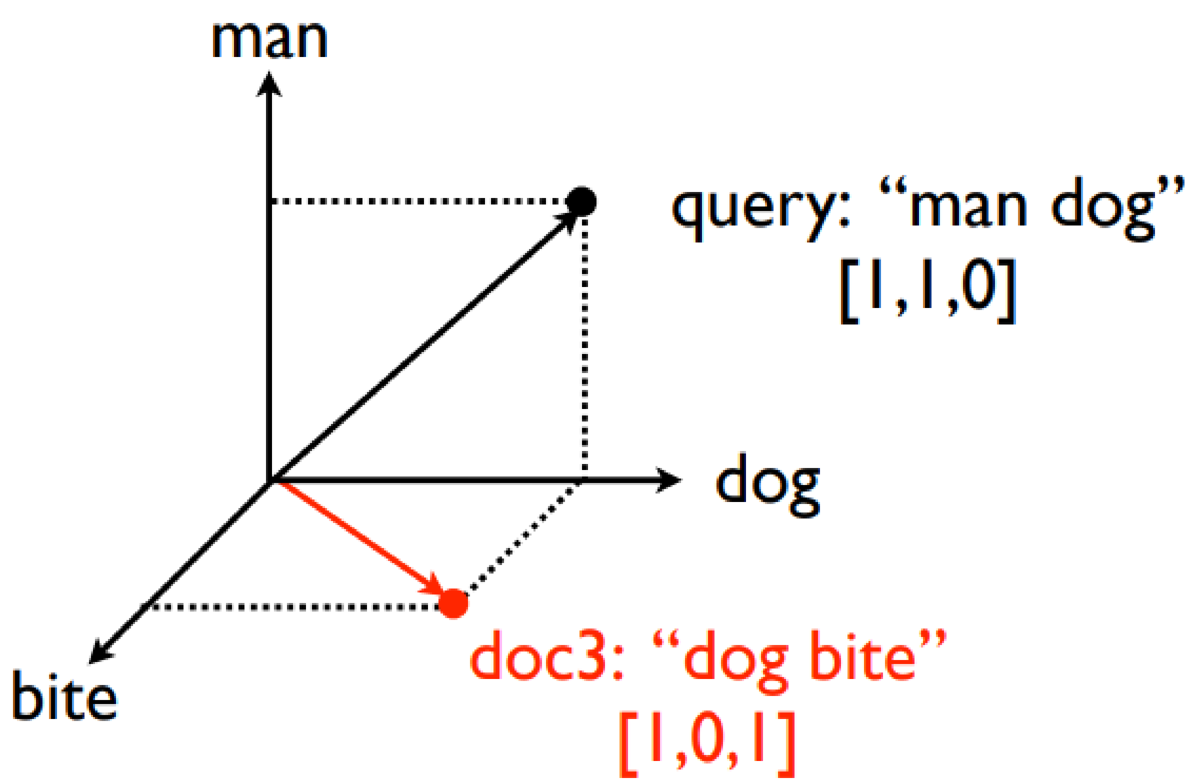
❖ Summary

- Any span of text is a vector in V-dimensional space, where V is the size of the vocabulary



Similarity Measure

- **A query is a vector in V-dimensional space, where V is the number of terms in the vocabulary**
  - A query is a special type of document



Similarity Measure

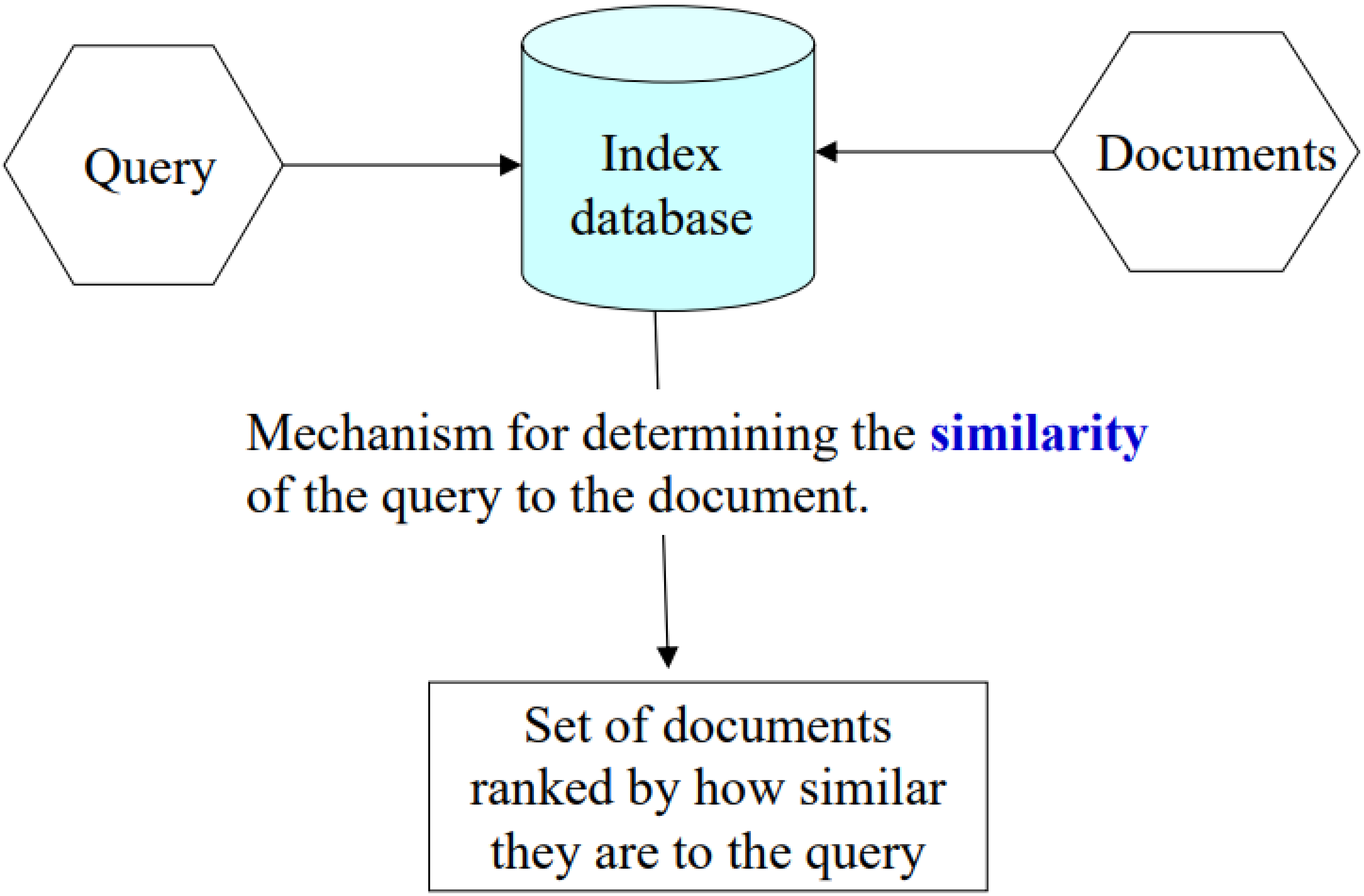
- **The vector space model ranks documents based on the vector-space similarity between the query vector and the document vector**

- **Retrieve the most similar documents to a query**

- **There are many ways to compute the similarity between two vectors**
  - Cosine similarity
    - We focus on cosine similarity
  - Jaccard similarity
  - Dice's coefficient
  - Inner product

Similarity Measure

❖ **How it works**



Similarity Measure

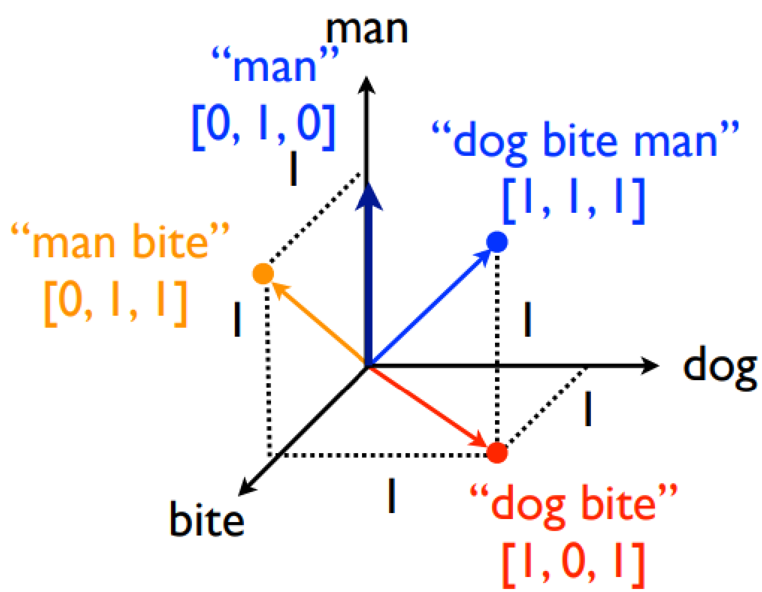
- **Cosine similarity**
  - Measures the cosine of the angle between the query and document vectors
  - Ranges from 0 to 1
    - Here, the documents with the highest scores are the most similar to the query
      - Equals 1 if the vectors are identical

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

Similarity Measure

- **Cosine similarity**
  - 1 = the term appears at least once
  - 0 = the term does not appear

	dog	*man*	*bite*
doc_1	1	1	1
doc_2	1	0	1
doc_3	0	1	1
doc_4	0	1	0



Similarity Measure

❖ Cosine similarity

- Given the query ‘man dog’ [1, 1, 0], cosine similarity calculation looks as follows

$$Cosine(D_1, Q) = \frac{(1\cdot 1) + (1\cdot 1) + (1\cdot 0)}{\sqrt{(1^2 + 1^2 + 1^2)\cdot(1^2 + 1^2 + 0^2)}} = \frac{2}{\sqrt{3\cdot 2}} \approx 0.816$$

$$Cosine(D_2, Q) = \frac{(1\cdot 1) + (0\cdot 1) + (1\cdot 0)}{\sqrt{(1^2 + 0^2 + 1^2)\cdot(1^2 + 1^2 + 0^2)}} = \frac{1}{\sqrt{2\cdot 2}} = 0.5$$

Practice in Python

Sample code for calculating Cosine Similarity

- Install scipy Python library first

```
from scipy import spatial

v1 = [1, 1, 1]
v2 = [1, 1, 0]

result = 1 - spatial.distance.cosine(v1, v2)
print(result)
```

Practice in Python

❖ Sample code for calculating Cosine Similarity

- Install sklearn Python library first
  - Returns an array with shape

```
from sklearn.metrics.pairwise import cosine_similarity

v1 = [1, 1, 1]
v2 = [1, 1, 0]

result = cosine_similarity([v1], [v2])

print(result)
```

감사합니다!