

# Text Preprocessing and Vectorization

MMA 865

Moez Ali

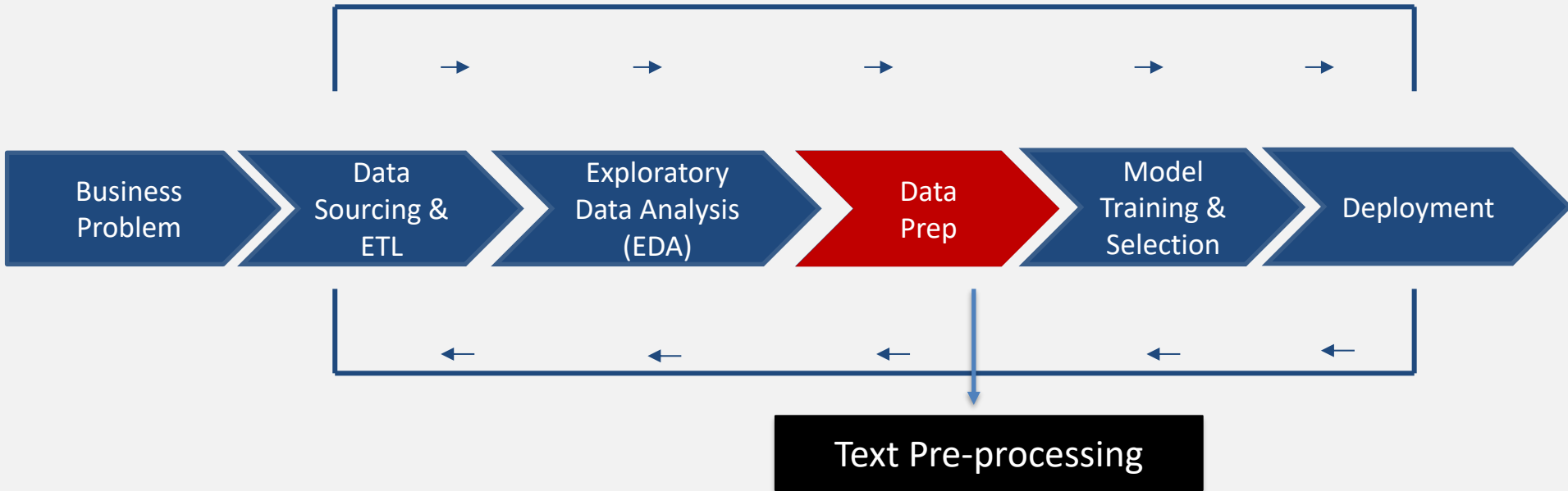
- **Vectorization**
  - Bag Of Words / TF-IDF
  - Embeddings
- **Text Preprocessing**
  - Tokenization
  - Case normalization
  - Spell checking
  - Removing unwanted characters/numbers
  - Stemming and lemmatizing
  - Removing stop words
  - Pruning rare and common words
  - n-grams

- THE U.S. LABORATORY FOR CLIMATE CHANGE research by the Department of Justice, issued recently by the Environmental Protection Agency (EPA), has been widely cited as the most comprehensive scientific proceedings on the issue. Despite the fact that the report was written by a team of 15 scientists, the report was not signed by any of them. The report was written by a team of 15 scientists, the report was not signed by any of them. The report was written by a team of 15 scientists, the report was not signed by any of them.



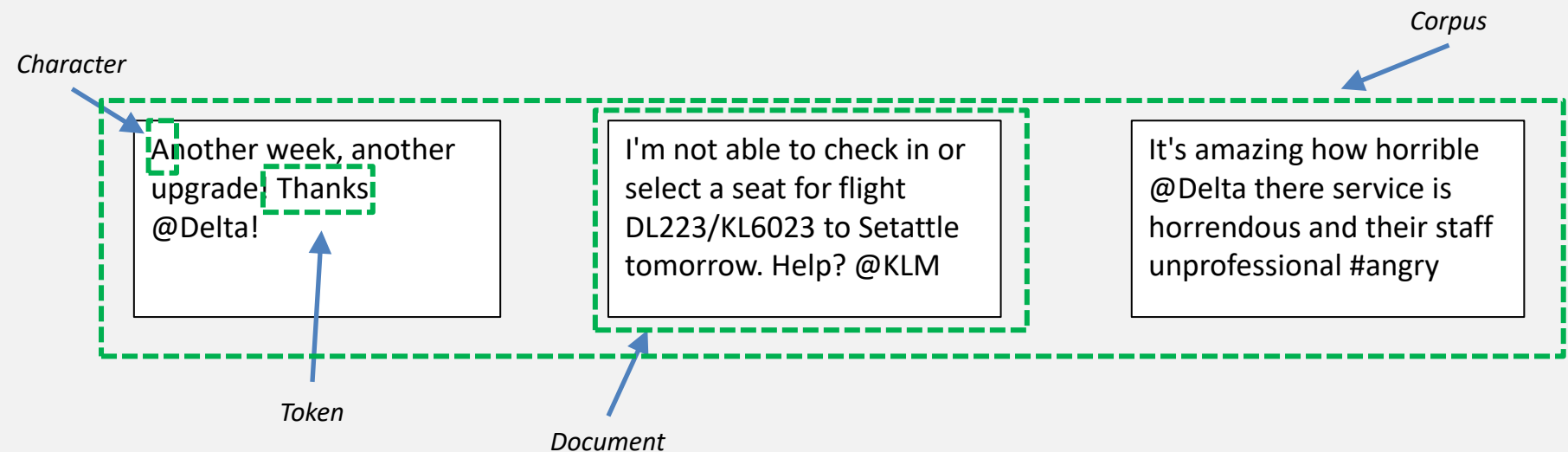
# Preprocessing

# Text Pre-processing in ML Life Cycle



# Definitions


- **Character:** smallest unit in a writing system
  - E.g., letters (abcdef), numbers (12345), punctuation (,.? \$), whitespace (spaces, tabs, carriage returns)
- **Word (Type):** sequence of characters with practical meaning
  - E.g., *Man, dog, Toronto*
- **Token (Term):** an instance of a word
  - E.g., *A rose is a rose is a rose (3 words, 8 tokens)*
- **Document:** A sequence of tokens
  - E.g., emails, blog posts, tweets, news articles, etc.
- **Corpus:** A collection of documents



# Vectorization

- If you want to do ML, your data frame can't have text!
  - It needs numbers
- The result of changing text to numbers is called **vectorization**
  - **Machine Learning: Use Bag of Words or similar**
  - **Deep Learning: Use embeddings**

ID	UID	Tweet	Sentiment
18741	2259	Another week, another upgrade! Thanks @Delta!	POS
31526	6836	I'm not able to check in or select a seat for flight DL223/KL6023 to Setattle tomorrow. Help? @KLM	NEG
38085	7722	Surprised and happy that @Delta helped me avoid the 3.5 hr layover I was scheduled for. Patient and helpful agents. #remarkable	POS
12429	9026	It's amazing how horrible @Delta there service is horrendous and their staff unprofessional #angry	NEG



ID	UID	Dim1	Dim2	Dim3	Dim4	DimN	Sentiment
18741	2259	0.53	0.36	0.54	0.73	0.40	POS
31526	6836	0.07	0.50	0.92	0.89	0.71	NEG
38085	7722	0.00	0.82	0.60	0.46	0.80	POS
12429	9026	0.75	0.54	0.96	0.03	0.61	NEG

- Can we do one hot encoding on text data similar to what we do for categorical features?

# Bag of Words

- Very common vectorization method for Machine Learning
  - Also called *term document matrix (TDM)* and *vector space model*
- Each word type becomes a column

ID	Text
1	My dog ate my homework.
2	The cat ate my sandwich.
3	A dolphin ate the homework and the sandwich.

ID	a	and	ate	cat	dolphin	dog	homework	my	sandwich	the	...
1	0	0	1	0	0	1	1	2	0	0	
2	0	0	1	1	0	0	0	1	1	1	
3	1	1	1	0	1	0	1	0	1	2	

Number of times  
"the" occurs in  
document 3



# Term Weighting Schemes

- *Term frequency* is one scheme
- But, there are many others

ID	a	and	ate	cat	dolphin	...
1	4	0	0	1	2	
2	1	1	0	0	1	
3	1	0	6	0	0	

Term frequency

ID	a	and	ate	cat	dolphin	...
1	1	0	0	1	1	
2	1	1	0	0	1	
3	1	0	1	0	0	

Binary term frequency

ID	a	and	ate	cat	dolphin	...
1	0.7	0	0	0.3	0.5	
2	0.3	0.3	0	0	0.3	
3	0.3	0	0.8	0	0	

Log (1 + term frequency)

ID	a	and	ate	cat	dolphin	...
1	0.2	0	0	0.9	0.2	
2	0.5	0.2	0	0	0.3	
3	0.1	0	0.6	0	0	

Term frequency \* inverse document frequency

**MOST POPULAR**

# More About TF-IDF

- ***Term frequency-inverse document frequency***: A popular term weighting scheme
- Basic idea: take a term's frequency in a document, and then decrease it by how common that term is in the entire corpus

*term*      *doc*      *corpus*

↙      ↑      ↘

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

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

Number of times term occurs in doc

Number of terms in doc

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

Number of docs

Number of docs with term

# TF-IDF Example

ID	Text
1	my dog love your dog
2	the dog love the park
3	my dog and your dog love fleas

→

ID	fleas	dog	love	park
1	0	2	1	0
2	0	1	1	1
3	1	2	1	0

Term frequency

→

ID	fleas	dog	love	park
1	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.16
3	0.12	0.00	0.00	0.00

TF-IDF

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

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

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

$$\text{tfidf}(\text{"fleas"}, d_3, D) = \frac{1}{4} \times 0.48 = 0.12$$

$$\text{tf}(\text{"fleas"}, d_3) = \frac{1}{4}$$

$$\text{idf}(\text{"fleas"}, D) = \log\left(\frac{3}{1}\right) = 0.48$$

# Exercise

- Given the following four documents, create BOWs by hand:
  - BOW for now
  - TF-IDF (try it later)

He loves Samsung!

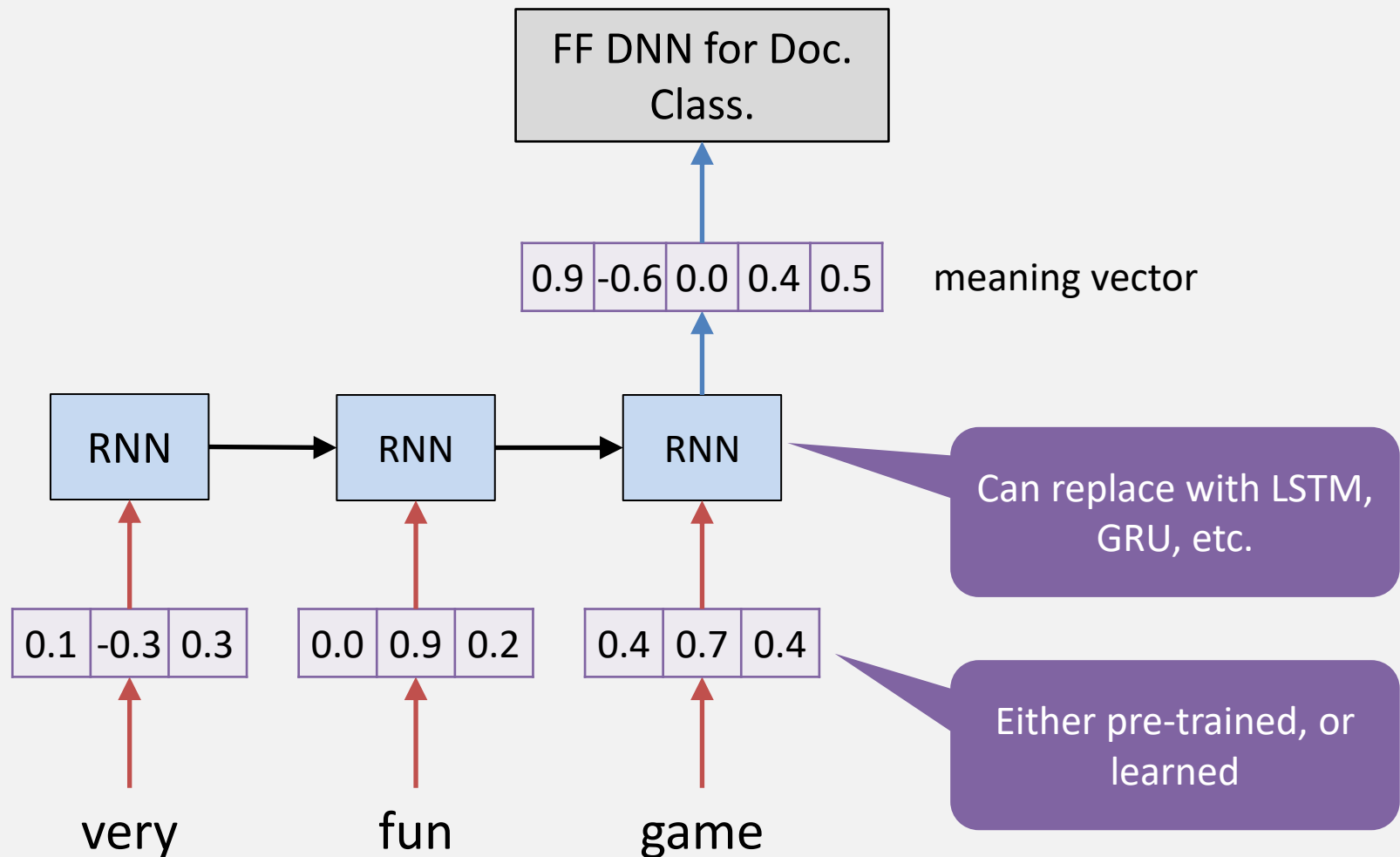
Samsung makes phones.

Phones love to make time disappear.

He uses my phone.

# Deep Learning: Embeddings

- Each word has an embedding, i.e., vector
- Vectors get input into Deep NN



# Popular Embedding Model choices

---

- **OpenAI Embeddings (Ada, Babbage, Curie, and Davinci Models):**  
OpenAI offers a range of embedding models, with the Ada series being the most widely used due to its efficiency and balance between cost and performance. These models are employed for tasks like semantic search, text similarity, clustering, and more.
- **Google's Universal Sentence Encoder (USE) - Commercial Version:**  
Google's commercial version of the Universal Sentence Encoder is widely used in enterprise settings for generating high-quality text embeddings. It supports multilingual embeddings and is integrated into various Google Cloud services.
- **T5 (Text-To-Text Transfer Transformer):** T5 can be fine-tuned for various NLP tasks, including generating text embeddings. It's versatile and has versions like mT5 for multilingual embeddings.

# Example

Example embedding response

json ▾



```
1 {  
2   "object": "list",  
3   "data": [  
4     {  
5       "object": "embedding",  
6       "index": 0,  
7       "embedding": [  
8         -0.006929283495992422,  
9         -0.005336422007530928,  
10        ... (omitted for spacing)  
11        -4.547132266452536e-05,  
12        -0.024047505110502243  
13      ],  
14    }  
15  ],  
16  "model": "text-embedding-3-small",  
17  "usage": {  
18    "prompt_tokens": 5,  
19    "total_tokens": 5  
20  }  
21 }
```

# Questions?

---

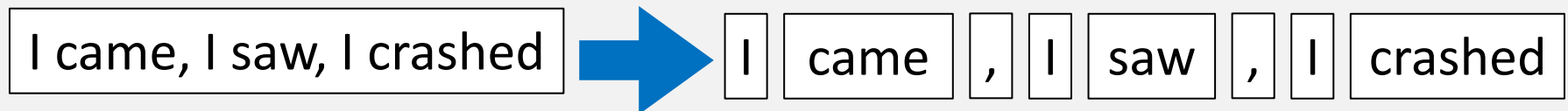
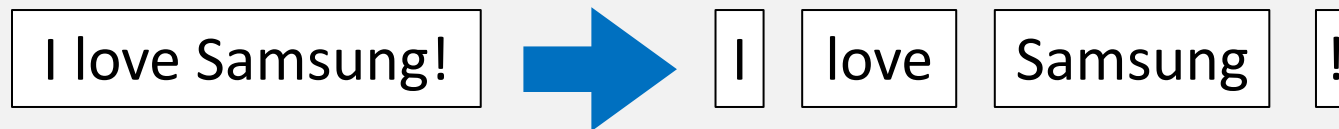


# TEXT PREPROCESSING

- For most text analytics tasks, preprocessing is needed:
- Rough ordering:
  - Tokenization
  - Case normalization
  - Spell checking
  - Removing stop words
  - Removing unwanted characters/numbers/patterns
  - Stemming and lemmatizing
  - Pruning rare and common words
  - n-grams


# Tokenization

- Split sentences into *tokens*
- Common technique is to split on white space and special chars



# Example: Tokenization

python

 Copy code

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

# Sample text for tokenization
text = "Hello! My name is John. I'm learning Natural Language Processing with Python. Tokenization is the process of breaking down a text into smaller units, such as words or sentences."

# Tokenizing sentences
sentences = sent_tokenize(text)
print("Input Text:\n", text)

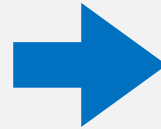
print("\nTokenized Sentences:")
print(sentences)

# Tokenizing words
words = word_tokenize(text)
print("\nTokenized Words:")
print(words)
```

# Case Normalization

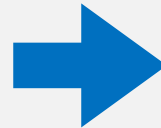
- Convert all characters to the best case
- ***Case folding***: Make everything lower case

He loves Samsung!



he loves samsung!

I work at General Motors.



i work at general motors.

- ***True casing***: Taking into account the type of word

He loves Samsung!



he loves Samsung!


I work at General Motors.



i work at General Motors.

# Example: Case Folding

python

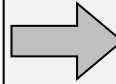
 Copy code

```
# Sample text for case normalization
text = "Hello! My name is John. I'm Learning Natural Language Processing with Python."

# Convert the text to lowercase
normalized_text = text.lower()

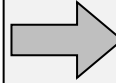
# Output both the original and normalized text
print("Original Text:\n", text)
print("\nNormalized Text (Lowercase):\n", normalized_text)
```

Back in shcool, we always  
played soccer.



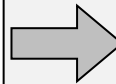
Back in **school**, we always  
played soccer.

When you hug a guy and smell  
his colon <3



When you hug a guy and smell  
his **cologne** <3

I'm joining the NAYV!!



I'm joining the **NAVY**!!

- **Spell checking**: fixing misspellings, typos, etc.
- Usually, lots of words will be misspelled
- Warnings: usually requires lots of time

# Spell Checkers / Rectifiers in Python


---

- pyspellchecker
- autocorrect
- TextBlob
- Tutorial: <https://theautomatic.net/2019/12/10/3-packages-to-build-a-spell-checker-in-python/>



# Example: Spell Correction

python

 Copy code

```
from textblob import TextBlob

# Sample text with spelling errors
text = "I havv a speling erorr in this sentnce."

# Create a TextBlob object
blob = TextBlob(text)

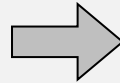
# Correct the spelling
corrected_text = blob.correct()

# Output both the original and corrected text
print("Original Text:\n", text)
print("\nCorrected Text:\n", corrected_text)
```

# Spelling Normalization

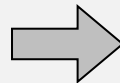
- ***Spelling normalization***: choosing a consistent style
  - American *versus* British, etc.

What color is his hair?



What **color** is his hair?

What colour is his hair?



What **color** is his hair?

# Stopping

ID	a	an	and	are	as	ass	at	athens	but	butterfly	for	forrest	if	in	into	inuit
1	1	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0
2	1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	0
3	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0
4	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0
5	1	1	1	1	1	0	1	0	1	0	1	1	1	1	1	0
6	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0
7	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	1
8	1	1	1	1	1	0	1	0	1	0	1	1	0	1	1	0
9	0	1	1	0	1	1	1	0	1	0	1	0	1	1	0	0
10	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0

What's wrong with this BOW?


- **Stopping:** Removing common words
- Helps to:
  - Reduce number of columns in BOW
  - Remove noise

a, an, and, are, as, at, be, but, by, for,  
if, in, into, is, it, no, not, of, on, or,  
such, that, the, their, then, there,  
these, they, this, to, was, will, with

Example stop word list

# Example: Stop words Removal

python

 Copy code

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

# Download stopwords list if not already downloaded
nltk.download('stopwords')
nltk.download('punkt')

# Sample text
text = "This is a sample sentence, showing off the stop words filtration."

# Tokenize the text into words
words = word_tokenize(text)

# Get the list of English stop words
stop_words = set(stopwords.words('english'))

# Remove stop words from the tokenized words
filtered_words = [word for word in words if word.lower() not in stop_words]


# Output both the original and filtered words
print("Original Text:\n", text)
print("\nFiltered Text (Without Stop Words):\n", ' '.join(filtered_words))
```

# Removing Unwanted Characters and Numbers

- Sometimes your text will contain unwanted characters
  - **Punctuation:** ,.'":;.!@#\$%^&\*()-\_+=
  - **Non-English characters:** Ábcdêãçoàúü
  - **Numbers:** 55, 45.66, 1,555,444
- Usually not helpful in BOW → Better to remove

# Example: Removing special characters

python

 Copy code

```
import re

# Sample text containing punctuation, non-English characters, and numbers
text = "Hello! This is an example text: 55, 45.66, 1,555,444. Non-English: Ábcdêãçoàúü, Sp

# Regular expression pattern to match unwanted characters
# [^a-zA-Z\s] means everything except letters and whitespace
pattern = r'[^a-zA-Z\s]'

# Removing unwanted special characters
cleaned_text = re.sub(pattern, '', text)

# Output both the original and cleaned text
print("Original Text:\n", text)
print("\nCleaned Text (Without Unwanted Special Characters):\n", cleaned_text)
```


# Removing Patterns

- In addition to removing whole words, can remove *patterns*
  - Called ***regular expressions***
- Tremendously useful for removing domain-specific noise
  - Email addresses, URLs, auto-generated lines, numbers, dates, signature lines
- E.g., Email address
  - `[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Zaz]{2,}`
- E.g., "Translated by Joe."
  - `Translated by \w*\.`
- E.g., Any number
  - `\d+`



# Example: Regex pattern for emails

python

 Copy code

```
import re

# Define a simple regex pattern for detecting emails
email_pattern = r'[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+'

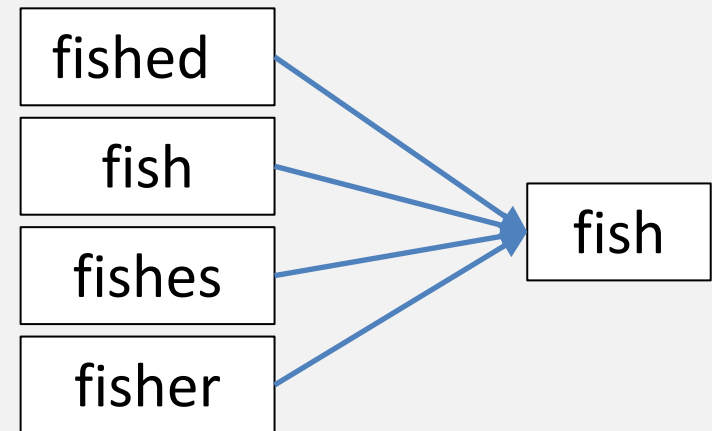
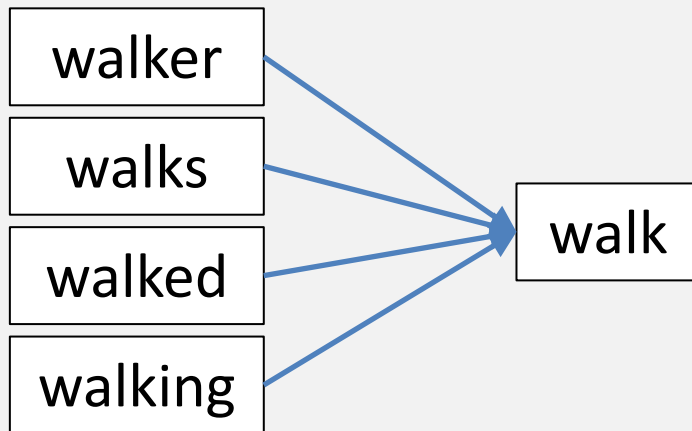
# Sample text containing email addresses
text = """
Please contact us at support@example.com for further assistance.
You can also reach out to john.doe123@gmail.com or jane-doe@company.co.uk.
Invalid emails like @missingusername.com or username@.com should not match.
"""

# Find all matching email addresses
matches = re.findall(email_pattern, text)

# Print the matches
print("Found email addresses:")
for match in matches:
    print(match)
```


# Stemming

- Reduce words to their root form
- Popular algorithms:
  - Porter (most gentle), Snowball, Lancaster (most aggressive)



# Example: Stemming

python

 Copy code

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

# Download necessary resources
nltk.download('punkt')

# Sample text for stemming
text = "The leaves were falling off the trees as the wind was blowing strongly. He studies"

# Tokenize the text into words
words = word_tokenize(text)


# Stemming using PorterStemmer
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in words]

# Output the results
print("Original Words:\n", words)
print("\nStemmed Words:\n", stemmed_words)
```

- "Advanced stemming": takes POS into account
- Examples:
  - *meeting* as a verb gets stemmed to *meet*
  - *meeting* as a noun gets stemmed to *meeting*
  - *is* gets stemmed to *be*
- Stemming is faster, but worse
- Lemmatization is slower, but better

# Example: Lemmatization

python

 Copy code

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

# Download necessary resources
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

# Sample text for lemmatization
text = "The leaves were falling off the trees as the wind was blowing strongly. He studies

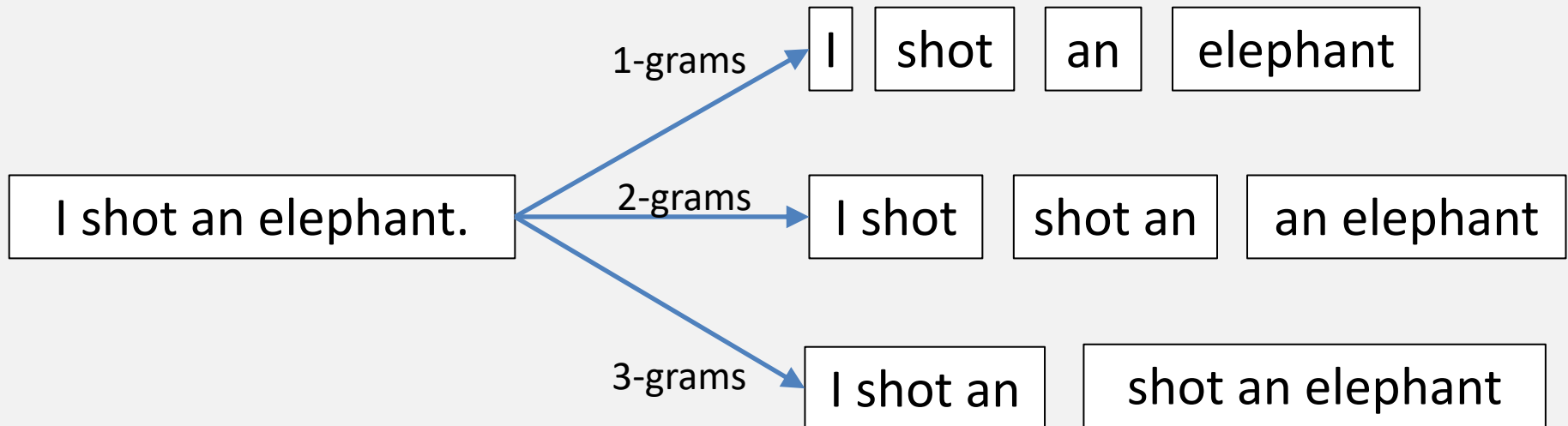
# Tokenize the text into words
words = word_tokenize(text)

# Lemmatization using WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

# Output the results
print("Original Words:\n", words)
print("\nLemmatized Words:\n", lemmatized_words)
```


# n-grams

- Sequences of  $n$  adjacent words
- Useful for some models: provides context around words
- Also called: k-grams, k-shingles



# Example: Ngrams

python

 Copy code

```
import nltk
from nltk import ngrams
from nltk.tokenize import word_tokenize

# Sample text for generating n-grams
text = "Natural Language Processing with Python is fun and educational."

# Tokenize the text into words
words = word_tokenize(text)

# Generate unigrams (1-grams)
unigrams = list(ngrams(words, 1))

# Generate bigrams (2-grams)
bigrams = list(ngrams(words, 2))

# Generate trigrams (3-grams)
trigrams = list(ngrams(words, 3))

# Output the results
print("Original Words:\n", words)
print("\nUnigrams:\n", unigrams)
print("\nBigrams:\n", bigrams)
print("\nTrigrams:\n", trigrams)
```

# Pruning Rare and Common Words

- Some words will only occur once or twice in the entire corpus.
  - Prune with **min\_df** in scikit-learn
- Some word will occur in almost every document.
  - Prune with **max\_df** in scikit-learn
- Sometimes you only want to keep the top N words total
  - Set with **max\_features** in scikit-learn


E.g.,  
**minDF = 2**  
**maxDF = 8**  
**vocabSize = 4**

Doc	a	an	and	are	as	ask	at	athens	but	butter	for	forrest	if	in	into	inuit
1	1	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0
2	1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	0
3	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0
4	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0
5	1	1	1	1	1	0	1	0	1	0	1	1	1	1	1	0
6	1	1	1	1	1	0	1	0	1	0	0	0	1	1	1	0
7	1	1	1	1	1	0	1	0	1	0	0	0	1	1	1	1
8	1	1	1	1	1	0	1	0	1	0	0	1	0	1	1	0
9	0	1	1	0	1	1	1	0	1	0	1	0	1	1	0	0
10	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0



# Example: CountVectorizer

python

 Copy code

```
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
documents = [
    "The cat sat on the mat",
    "The dog sat on the log",
    "The cat chased the dog",
    "The dog chased the cat",
    "The mat was on the log"
]

# Initialize CountVectorizer with min_df and max_df
vectorizer = CountVectorizer(min_df=2, max_df=0.8)

# Fit and transform the documents
X = vectorizer.fit_transform(documents)
```

# Summary

Task	Python Function	Before	After
Tokenization	<code>split</code> , <code>nltk.word_tokenize</code>	Where are you today?	"where" "are", "you" "today"
Case normalization	<code>lower</code>	How are you, Steve?	how are you, steve?
N-grams	<code>nltk.util.ngrams</code>	Hey there, I'm awesome.	"Hey there", "there I'm", "I'm awesome"
Removing punctuation	<code>re.sub</code>	Hey! Let's go to the bar...	Hey Lets go to the bar
Replacing weird characters	<code>unidecode.unidecode</code>	Ábcdêãçoàúü	Abcdeacoauu
Remove extra whitespace	<code>re.sub</code>	I like    coffee.	I like coffee.
Removing numbers	<code>re.sub</code>	There are only 4 classes left.	There are only    classes left.
Stemming	<code>nltk.stem</code>	We are writing code like hackers.	We are write code like hacker.
Spell checking	<code>pattern.en.suggest</code>	This is not spelld correctly.	This is not spelled correctly.
Stopping	<code>nltk.corpus.stopwords</code>	My name is Steve and I am a good chef	Steve am chef
Removing rare words		Steve really is a goodfella	Steve really is a

---

# SUMMARY

- **Preprocessing**: cleaning up text
- **Vectorization**: turning text into numbers
  - ML: BOW (TF, TF-IDF, Binary)
  - DL: Embeddings
- **Common preprocessing steps**:
  - Case normalization, tokenization, n-grams, removing unwanted characters/numbers, stemming and lemmatizing, spell checking, removing stop words, removing rare words