CoGrammar

Welcome to this session:

Task Walkthrough - Task 22 - 26 (Part 3.5)

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Skills Bootcamp Data Science

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions]

Skills Bootcamp Data Science

- For all **non-academic questions**, please submit a query: ***www.hyperiondev.com/support***
- **Report a safeguarding incident: *www.hyperiondev.com/safeguardreporting***
- We would love your feedback on lectures: ***Feedback on Lectures***
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Learning Outcomes

- **Explain the role of NLP and text preprocessing** in machine learning.
- **Use spaCy** for tokenization, stopword removal, and text normalization.
- **Perform sentiment analysis** on text data using TextBlob and spaCy.
- **Compare text similarity scores** to detect patterns in datasets.
- **Analyse the strengths and limitations** of sentiment analysis models.

Task Walkthrough

Imagine you work for a movie streaming service that wants to analyze user feedback. Your goal is to:

- ❖ Identify whether a movie review is positive, negative, or neutral.
- ❖ Compare reviews to detect similar opinions from different users.
- ❖ Summarize insights about how people feel about different movies.

Your task is to:

- ❖ Download the IMDb movie reviews dataset.
- ❖ Preprocess the reviews using spaCy (remove stopwords, lowercase text, clean punctuation).
- ❖ Perform sentiment analysis using TextBlob and spaCy to classify each review.
- ❖ Compare review similarity scores to group reviews with similar opinions.

What is the primary purpose of tokenization in NLP?

A. To convert text into numerical values
B. To split text into meaningful components like words and sentences
C. To translate text into another language
D. To remove all punctuation from text

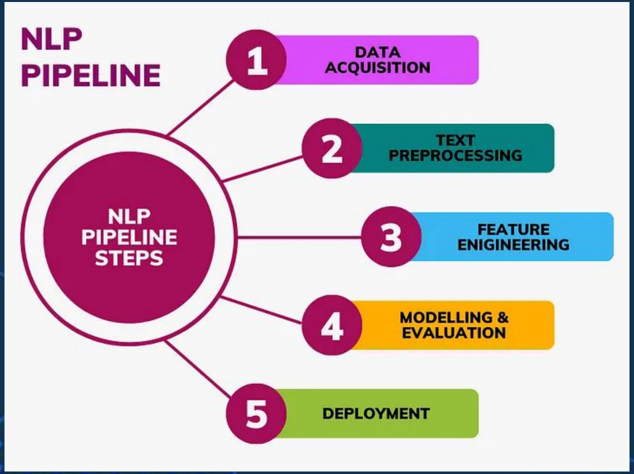
Why is it important to remove stopwords in text analysis?

A. They contain the most important parts of a sentence
B. They add unnecessary computational complexity and do not contribute much meaning
C. They improve the accuracy of text classification models
D. They always indicate sentiment in a sentence

CoGrammar

NLP Pipeline

- **Data Acquisition** - obtaining raw textual data e.g. **built-in or public datasets**, collect (scraping) data from websites (not in the lecture)



- **Text Preprocessing** - refine raw text data for meaningful analysis
  - **Basic Cleaning:** eliminate irrelevant elements unnecessary for linguistic analysis (e.g., stripping out HTML tags, handling emojis, spell checks.
  - **Basic Preprocessing:** tokenisation, stemming/lemmatisation, stop-word removal.
  - **Advanced Preprocessing:** POS tagging, NER

NLP Pipeline

- **Feature Engineering:** transforming **raw text data into numerical features** that machine learning models can comprehend and utilize effectively, e.g. bag-of-words, TF-IDF, word embeddings.

- **Modelling:** models are applied and evaluated using different approaches.

- **Evaluation:** comprehensively gauge model performance

- **Deployment:** transition of the developed model from the development environment to a production environment

CoGrammar

Text Cleaning

Regular Expressions

CoGrammar

Regular Expressions

- **Regular expressions** or **RegEx** is a sequence of characters mainly used to find or replace patterns embedded in the text.
- Strings with a **special syntax.**
- Allow to **match patterns** in other strings.
- **Applications**: Find all weblinks in a document, parse email addresses, remove/replace unwanted characters.

```python
import re

txt = "Across the Universe"
'''
Check if the string starts with (^) the word "Across" and ends with ($)
the letter "e". The .* is for any other characters.
'''
x = re.search("^Across.*e$", txt)

if x:
  print("Yes! We have a match!")
else:
  print("No match")
#Output: Yes! We have a match!
```

CoGrammar

Regular Expressions

```
txt = "Across the Universe"
#Split the string at all white-space character:
print(re.split("\s+", txt))
#Split the string at the first white-space character
print(re.split("\s", txt, 1))
#Output ['Across', 'the', 'Universe']
# ['Across', 'the Universe']
```

Please see cheat sheet for more options

```
txt = 'The heart is a bloom, shoots up through the stony ground'
print(re.findall("oo", txt))
# Output: ['oo', 'oo']
```

```
txt = "But in the end, it doesn't even matter"
print(re.sub("doesn't even", "does really", txt))
# Output: But in the end, it does really matter
```

NLP tools

spaCy

spaCy

- **spaCy**, written in Python and Cython, is an open-source software library for NLP.
- **Fast and intuitive**, top contender for **beginners NLP tasks.**
- Specifically designed to be an useful library for implementing **production-ready systems.**
- In contrast, **natural language toolkit (NLTK)** is more comprehensive than spaCy, allows in-depth customization and implementation of specific algorithms for advanced **research projects.**

CoGrammar

spaCy

- spaCy can be installed using pip

```
pip install -U spacy
```

- If a **trained pipeline** is available for a language, it can be download using the spacy download command.
  Here the spaCy's trained pipelines for **English language** can be installed as Python packages

```
python -m spacy download en_core_web_sm
```

[spaCy models and languages]

- Once downloaded, the model can be imported as

```
import spacy

nlp = spacy.load("en_core_web_sm")
```

[spaCy VSCode extension]

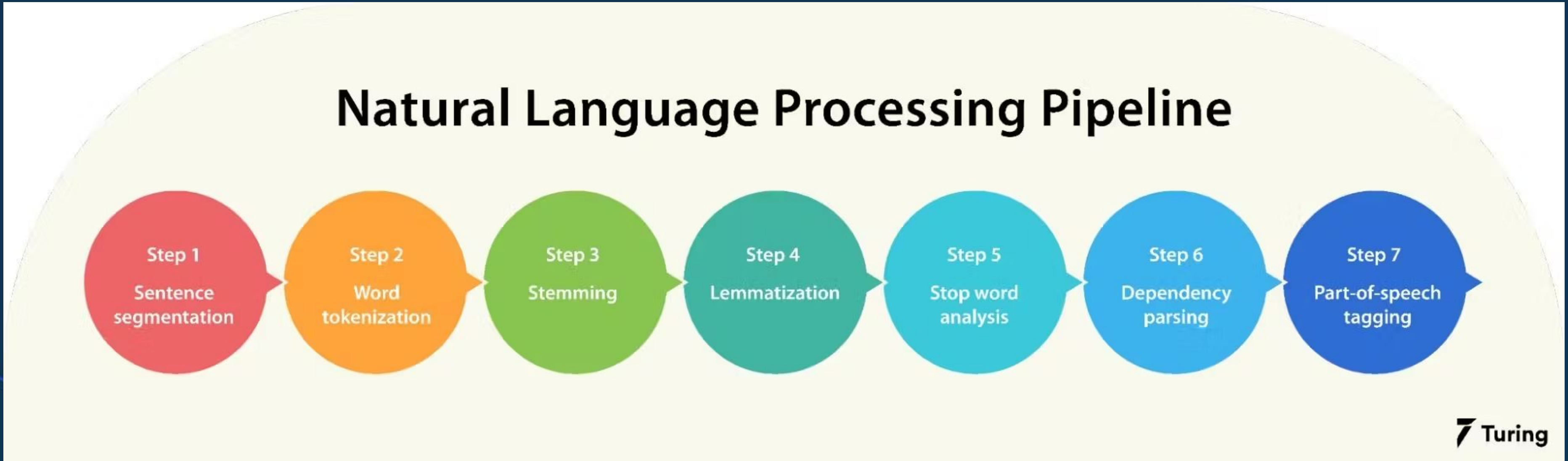
CoGrammar

NLP Pipeline

Text Preprocessing

CoGrammar

NLP Pipeline



CoGrammar

Sentence Fragmentation

Sentence Fragmentation is the first step in NLP pipeline, divides the entire paragraph into **different sentences** for better understanding.

```
text = ("Friends, Romans, countrymen, lend me your ears. I come to bury Julius Caesar, not to praise him."
"The evil that men do lives after them. The good is oft interred with their bones.")
```

Sentence fragmentation using spaCy

```
doc = nlp(text)
for i in doc.sents:
    print(i)
```

Output

```
Friends, Romans, countrymen, lend me your ears.
I come to bury Julius Caesar, not to praise him.
The evil that men do lives after them.
The good is oft interred with their bones.
```

CoGrammar

Tokenisation

Word tokenisation breaks the sentence into **separate words** or **tokens**. This helps understand the context of the text.

```
text = ("Friends, Romans, countrymen, lend me your ears.")
doc = nlp(text)
doc.text.split()
```

```
['Friends,', 'Romans,', 'countrymen,', 'lend', 'me', 'your', 'ears.']
```

CoGrammar

Stemming

- **Stemming** normalises words into their **base or root form**, helps to **predict** the **parts of speech** for each token, involves **stripping** the **prefixes/suffixes** from words to **get their stem.**

- For example, converting the word “walking” to “walk”.

- Another example, “intelligently”, “intelligence”, and “intelligent”, all these words originate from a single root word “intelligen”. However, in English there is no such word as “intelligen”.

- Stemming chops off the part of word by assuming that the result is the expected word, **not grammar based**, hence **inaccurate**.

- **spaCy does not provide a built-in function for stemming** as its inaccuracy is not suitable for production level use.

Lemmatisation

Lemmatisation removes inflectional endings and returns the canonical form of a word or **lemma**. Similar to stemming except that the lemma is an **actual word.**

For example, ‘playing’ and ‘plays’ are forms of the word ‘play’. Hence, play is the lemma of these words. Unlike a stem (recall ‘intelligen’), ‘play’ is a proper word.

```
doc = nlp("The dogs saw bats with best stripes hanging upside down by their feet")

for token in doc:
    print(token.text + "-->" + token.lemma_)
```

Output

```
The-->the
dogs-->dog
saw-->see
bats-->bat
```

```
with-->with
best-->good
stripes-->stripe
hanging-->hang
```

```
upside-->upside
down-->down
by-->by
their-->their
feet-->foot
```

Stop words

- Consider the **importance** of each and every word in a given sentence.

- In English, some words appear **more frequently** than others such as "is", "a", "the", "and". As they appear often, the NLP pipeline flags them as **stop words.** They are **filtered out** so as to focus on more important words.

spaCy has **326** default stopwords (output shows only a few)

```
stopwords = nlp.Defaults.stop_words
print(len(stopwords))
print(stopwords)
```

```
326
{'of', 'made', 'hereupon', 'am', 'everything', 'my',
```

Stop words

Remove stop words from text

```
nlp = spacy.load("en_core_web_sm")
text = "This is not a good time to talk"

cleanedtext = []
for item in nlp(text):
    if not item.is_stop:
        cleanedtext.append(item.text)
print(' '.join(cleanedtext))
```

Output

```
good time talk
```

Add/remove stop words

```
# Adding single token as stopword
nlp.Defaults.stop_words.add("perfect")
# Adding multiple tokens
nlp.Defaults.stop_words|={"hot","cold"}
```

```
# Removing single token
nlp.Defaults.stop_words.remove("what")
# Removing multiple tokens
nlp.Defaults.stop_words -= {"who", "when"}
```

CoGrammar

Parts Of Speech Tagging

- **Parts of speech (POS)** depicts how a specific word is utilized in a sentence, giving each word in a text a **grammatical category,** such as nouns, pronoun, verbs, adjectives, adverbs, prepositions, conjunctions, interjections.

- To **understand grammatical structure** of a sentence, **disambiguate words with multiple meanings** (e.g., "bank" can have multiple meanings), improve accuracy of NLP tasks, facilitate research in linguistics.

- Essential for assigning a **syntactic category,** needed for text summarization, sentiment analysis, machine translation.

Feature Engineering

CoGrammar

Feature extraction

- Machine Learning algorithms learn from a predefined set of features from the training data to produce output for the test data.
- ML algorithms cannot work on the raw text directly.
- We need feature extraction techniques to **convert text into a matrix (or vector) of features** to analyse the similarities between pieces of text.
- **Semantic similarity:** degree of similarity or closeness between two sentences in terms of their meaning or semantic content, fundamental in NLP.

Word Embeddings

- **Word embeddings** are dense vector representations of words, each word is represented as a high-dimensional vector in a continuous space.

- The embeddings capture **semantic** and **syntactic similarities** between words based on their contextual usage in large text corpora.

- Use various models and find the sentence similarity between a query and some examples sentence.

- **One-hot coding** can be used; however challenging for large corpora, feature vector length gets expanded, **out of vocabulary (OOV)** problem if the training data does not contain the exact word. We have better models.

Semantic Similarity

- **Text similarity:** calculate how two words/phrases/documents are close to each other.

- **Semantic similarity** is about the meaning closeness, and **lexical similarity** is about the closeness of the word set.

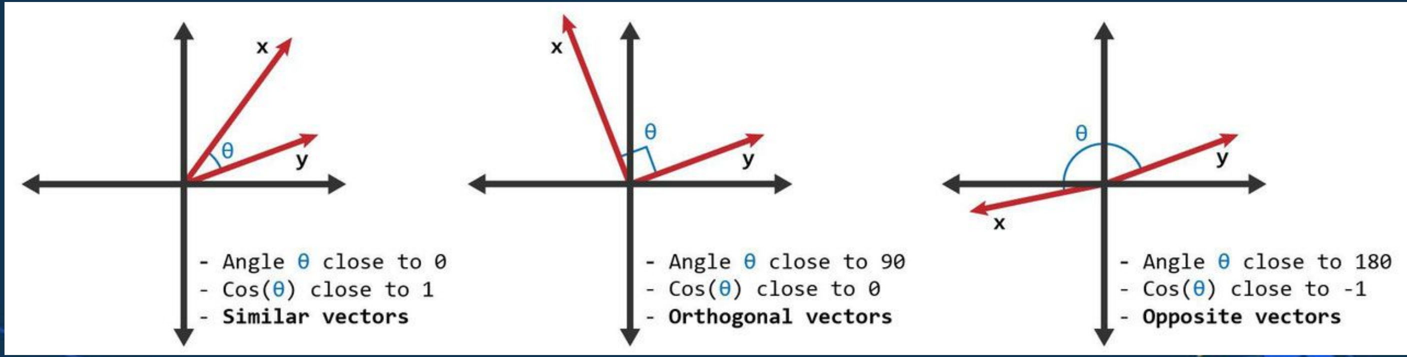
  - *"The dog bites the man"* and *"The man bites the dog"*
  - Identical considering lexical similarity; however entirely different considering semantic similarity

- Metrics to measure how 'close' two points: Euclidean distance, Manhattan distance or Hamming distance, less reliable for different length corpus.

Semantic Similarity

- **Cosine similarity** in NLP domain: measures the cosine of the angle between vectors of two points.

- Value of 1 indicates smallest angle between the vectors and the more similar the documents are.



$$Similarity(A,B) = \frac{A.B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Kaggle

CoGrammar

spaCy model

Use word embeddings from the pre-trained spaCy model ***"en_core_web_md"***

```python
import spacy

# To use word vectors, install larger models ending in md or lg
# en_core_web_md or en_core_web_lg

# Run the next line only the first time to download
#!python -m spacy download en_core_web_md

# Load SpaCy model with pre-trained word embeddings
nlp = spacy.load("en_core_web_md")

# Process the sentences to obtain Doc objects
doc1 = nlp("I like cats and dogs")
doc2 = nlp("I love all animals")

# Access the vector representations of the entire sentences
embedding1, embedding2 = doc1.vector, doc2.vector

# Calculate the similarity between the embeddings
similarity = doc1.similarity(doc2)

# Print the similarity
print("Similarity between the sentences:", similarity)
```

```
Similarity between the sentences: 0.8570134262541451
```

CoGrammar

Bag of Words

- Text is represented as a **bag (collection) of words disregarding grammar and word order,** but keeping the **frequency of words.**

- Assumes text from a class is characterized by unique set of words; if two text pieces have nearly the same words, then they belong to the same bag (class). Analyzing the words present in a piece of text, one can identify the class (bag) it belongs to.

- Used in text classification, document similarity, and text clustering.

- **Bag of n-gram** considers the **phrases or word order**, by breaking text into chunks of n continuous words.

CoGrammar

Bag of Words

```python
# Use steps of a recipe as phrases
corpus = [''
    'Preheat the oven',
    'lightly spray the baking dish',
    'combine the sugar, flour, cocoa powder, chocolate chips',
    'Sprinkle the dry mix',
    'Pour the batter',
]

# import and instantiate the vectorizer
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()

# apply the vectorizer to the corpus
X = vectorizer.fit_transform(corpus)

# display the document-term matrix as a
# pandas dataframe to show the tokens
vocab = vectorizer.get_feature_names_out()
docterm = pd.DataFrame(X.todense(), columns=vocab)
```

Import **CountVectorizer** from **sklearn**

baking	batter	chips	chocolate	cocoa	combine	dish	dry	flour	lightly
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	1
0	0	1	1	1	1	0	0	1	0
0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0

lightly	mix	oven	pour	powder	preheat	spray	sprinkle	sugar	the
0	0	1	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	1
0	0	0	0	1	0	0	0	1	1
0	1	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	1

Semantic Similarity

```python
# Example sentences
sentences = [
    "The tourism industry is collapsing",
    "The COVID-19 travel shock hit  tourism-dependent economies hard",
    "Poaching and illegal wildlife trafficking trends in Southern Africa",
]

# Query
query = "The collapse of tourism and its impact on wildlife"
```

```
spaCy similarity
Query: The collapse of tourism and its impact on wildlife
Nearest neighbors: 2
Poaching and illegal wildlife trafficking trends in Southern Africa - Distance: 0.2246145
The tourism industry is collapsing - Distance: 0.3102746
```

```
BoW similarity
Query: The collapse of tourism and its impact on wildlife
Nearest neighbors: 2
The tourism industry is collapsing - Distance: 0.5527864045000421
Poaching and illegal wildlife trafficking trends in Southern Africa - Distance: 0.6666666666666667
```

CoGrammar

TF-IDF

- **BoW** considers only word frequencies within a document and treats all words equally

- **Term Frequency-Inverse Document Frequency (TF-IDF)**: differentiates between common and rare words, and thereby reflects on the TF-IDF scores.

- **TF:** measures the frequency of a term (word) within a document.

- **IDF:** measures the rarity of a term across the entire corpus (collection of documents). Words that are frequent in many documents (such as “the,” “and,” etc.) receive a lower IDF weight, while words that are unique to specific documents receive a higher IDF weight.

CoGrammar

TF-IDF

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
new_corpus = [
    "The quick brown fox jumps over the lazy dog",
    "The dog barks at the fox",
    "The fox is quick and the dog is lazy"
]

# import and instantiate the BoW vectorizer
bow_vectorizer = CountVectorizer()
# Initialize TfidfVectorizer
vectorizer = TfidfVectorizer()

# Learn the vocabulary and transform the documents into a BoW and TF-IDF matrix
bow_matrix = bow_vectorizer.fit_transform(new_corpus)
tfidf_matrix = vectorizer.fit_transform(new_corpus)
```

Import **TfidfVectorizer** from sklearn

BoW vs TF-IDF

- For basic classification tasks, clustering, or counting word occurrences, **BoW** might be sufficient.

- However, for more advanced NLP tasks (language understanding, semantic analysis, and relationship extraction), BoW **feature space** can be very **high dimensional**, does not consider the **associations between words**, does not capture **semantic relationships.**

- **TF-IDF** reflects **importance of a word** in a document **relative** to an entire corpus.

- More advanced: contextual embeddings (BERT or GPT)

CoGrammar

Task Walkthrough

Imagine you work for a movie streaming service that wants to analyze user feedback. Your goal is to:

- ❖ Identify whether a movie review is positive, negative, or neutral.
- ❖ Compare reviews to detect similar opinions from different users.
- ❖ Summarize insights about how people feel about different movies.

Your task is to:

- ❖ Download the IMDb movie reviews dataset.
- ❖ Preprocess the reviews using spaCy (remove stopwords, lowercase text, clean punctuation).
- ❖ Perform sentiment analysis using TextBlob and spaCy to classify each review.
- ❖ Compare review similarity scores to group reviews with similar opinions.

Summary

- **Natural Language Processing (NLP)** allows computers to understand and analyze text.
- **spaCy** provides tools for tokenization, stopword removal, and text similarity comparison.
- **Sentiment analysis** helps classify text as positive, negative, or neutral.
- **Text similarity analysis** is useful for grouping reviews with similar opinions.
- **Preprocessing steps like cleaning text** improve the accuracy of NLP models.

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education