

Welcome Here!!

NATURAL LANGUAGE PROCESSING (NLP)

TEXT PREPROCESSING AND TOKENIZATION

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X (TWITTER): @HABEREJO

Day 2

Tuesday, 11th March, 2025

Why are we here

To equip learners with the skills to understand how Artificial Intelligence models are built, explore Machine Learning and Deep Learning, and focus on Natural Language Processing (NLP).

we should build some project right?

Yes!!

*If you are in with me,
we should build some project.*

Classes Structure

Every Tuesday throughout the month of March.

4 classes in total.

Each class for about 1:45 minutes

*Days are NOT likely to change, although,
my schedule can be OUCH, I wil reachout
earlier before. understand me bikoooo*

Who should be here



- You're curious and ready to learn something new
- You dream of becoming a Data Scientist.
- You're passionate about building the future as a Machine Learning Engineer.
- You have a basic understanding of Python programming.
- You've worked with data and want to take your skills to the next level.
- You're a researcher exploring the exciting world of AI. *which of these are you?*

Who should be here



- You're familiar with the fundamentals of Machine Learning and Deep Learning.
- You love problem solving and are excited to tackle real world problems with AI.
- You want to understand the technology that is shaping the future.
- You're driven by a desire to learn and grow in the field of AI.
- You enjoy collaborating and learning from others.

*which of these are you?????
You ticked any of these boxes?
Let's Goooooo*

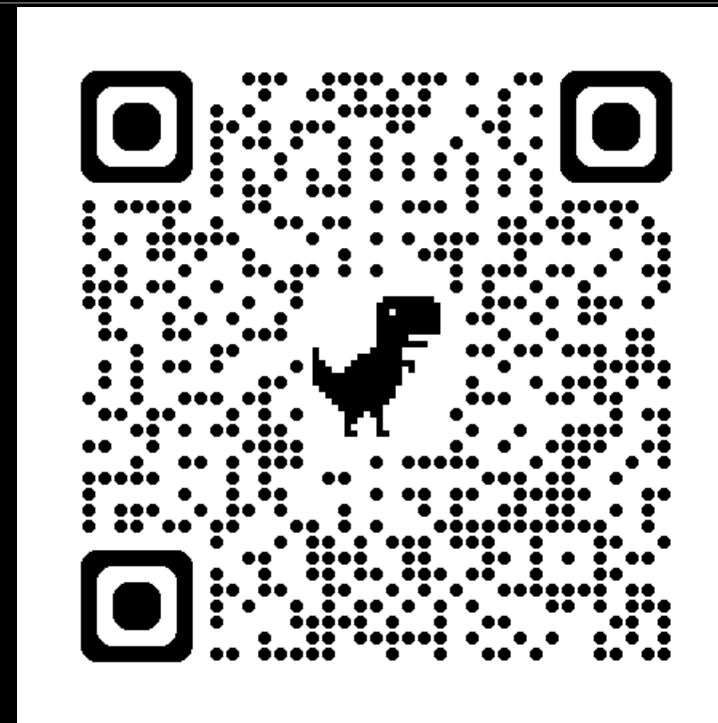
Before we begin

Who I am?

- A Machine Learning Engineer
- A young MAN eager to master AI

FUN FACT:

I've taught over 100 people Python, but I'm still learning new things every day. (Especially how to avoid typos when coding late at night!)



scan to visit my portfolio
or
<https://bheez.netlify.app>

Before we begin

Quick Recap

1 Understanding AI, ML, DL, and NLP

2 Introduction to NLP

3 NLP Use Cases: Translation Apps 🌐, Spam Filters 📧, Search Engines 🔍, Sentiment Analysis 😊 😡, Voice Assistants 🎤, Chatbots 💬, Autocorrect & Predictive Text 📱.

4 How Computer Understands Text: 1. Text Preprocessing 2. Tokenization, 3. Numerical Representation (Word Embedding)

5 Text Preprocessing Techniques: Lowercasing, Removing Punctuation Marks, Removing Stopwords (i.e. like A, of, in, etc), Stemming / Lemmatization.

Course Structure

- 1 Text Preprocessing Techniques
- 2 Stemming and Lemmatization
- 3 Hands-on NLP Tasks
- 4 Brain Teasers & Discussion
- 5 Tokenization: **Sentence Tokenization** (Sentence-Level Tokenization)
 - Word Tokenization** (Word-Level Tokenization)
 - Subword Tokenization**
 - Character-level Tokenization**
- 6 **Numerical Representation:** Vocabulary and Integer Encoding
- 7 **Numerical Representation:** One-Hot Encoding

And lots more

Step 1: Text Preprocessing

4. Stemming / Lemmatization:

Stemming: Stemming is the truncation of words down to their stem, **i.e their root form.**

For example, the words **house**, **housed** and **housing** both have the stem **hous**.

With smaller datasets in particular, stemming can be productive because it pools words with similar meanings into a single token.

The Quick! bRoWn foX juMPS.



quick brown fox jumps

*Depending on the NLP project,
some steps are compulsory*

Step 1: Text Preprocessing

There will be more examples of this stemmed token's context, enabling techniques like **word2vec** or **GloVe** to more accurately identify an appropriate location for the token in word-vector space

To stem words, you can use the Porter algorithm⁵ provided by **nlk**. To do this, you create an instance of a **PorterStemmer()** object and then add its `stem()` method

The Quick! bRoWn foX juMPS.



quick brown fox jumps

NLTK == Python Natural
Language Toolkit. It's a
package

Step 1: Text Preprocessing

Stemming vs Lemmatization

change
changing
changes
changed
changer

chang

change
changing
changes
changed
changer

change

From this comparison, Stemming returns to its root form. Stemming is quite harsh

While

Lemmatization is quite flexible and return it to a common word.

Choose based on the kind of words you have and your project.

Step 1: Text Preprocessing

🤖 Difference Between Stemming and Lemmatization

Both stemming and lemmatization reduce words to their root form, but they work differently!

Feature	Stemming 🛠️	Lemmatization 📖
How It Works	Chops off word endings based on rules (heuristics).	Uses a dictionary to find the actual base form of a word.
Speed	Fast but less accurate.	Slower but more accurate.
Output	May not return a real English word.	Always returns a real word.
Use Case	Search engines, keyword extraction.	NLP tasks requiring correct meaning, like chatbots and sentiment analysis.
Example 1	"running" → "run"	"running" → "run"
Example 2	"happiness" → "happi" ❌	"happiness" → "happiness" ✅
Example 3	"better" → "better" ❌	"better" → "good" ✅

Stemming follows simple rules to cut off suffixes like -ing, -ed, or -ly.

Lemmatization uses linguistic knowledge to find the true root form of a word.

Step 1: Text Preprocessing

0	I absolutely LOVE this product!! ❤️ It's super efficient and really worth the money. Definitely recommend! 🙌	i absolutely love this product!! ❤️ it's super efficient and really worth the money. definitely recommend! 🙌	i absolutely love this product its super efficient and really worth the money definitely recommend	absolutely love product super efficient really worth money definitely recommend	absolut love product super effici realli worth money definit recommend	absolutely love product super efficient really worth money definitely recommend
1	Worst purchase ever... 😡 Waste of money. DO NOT BUY!! Full of issues.	worst purchase ever... 😡 waste of money. do not buy!! full of issues.	worst purchase ever waste of money do not buy full of issues	worst purchase ever waste money buy full issues	worst purchas ever wast money buy full issu	worst purchase ever waste money buy full issue
2	This product does what it says, but nothing special. 🙄 It's okay for the price, I guess.	this product does what it says, but nothing special. 🙄 it's okay for the price, i guess.	this product does what it says but nothing special its okay for the price i guess	product says nothing special okay price guess	product say noth special okay price guess	product say nothing special okay price guess
3	AMAZING quality and fast shipping!!! 🚀 🔥 #satisfied #fastdelivery	amazing quality and fast shipping!!! 🚀🔥 #satisfied #fastdelivery	amazing quality and fast shipping satisfied fastdelivery	amazing quality fast shipping satisfied fastdelivery	amaz qualiti fast ship satisfi fastdeliveri	amazing quality fast shipping satisfied fastdelivery
4	Terrible! Had high expectations, but it broke in a week. Really disappointed. 😞	terrible! had high expectations, but it broke in a week. really disappointed. 😞	terrible had high expectations but it broke in a week really disappointed	terrible high expectations broke week really disappointed	terribl high expect broke week realli disappoint	terrible high expectation broke week really disappointed
5	This phone is great 📱, but the battery drains too fast. 🕒 😞	this phone is great 📱, but the battery drains too fast. 🕒 😞	this phone is great but the battery drains too fast	phone great battery drains fast	phone great batteri drain fast	phone great battery drain fast
6	I love how easy it is to use! 😊 Definitely a game-changer.	i love how easy it is to use! 😊 definitely a game-changer.	i love how easy it is to use definitely a gamechanger	love easy use definitely gamechanger	love easi use definit gamechang	love easy use definitely gamechanger
7	Do not buy this laptop! 💡 It crashes every 10 minutes. So frustrating! 😡	do not buy this laptop! 💡 it crashes every 10 minutes. so frustrating! 😡	do not buy this laptop it crashes every 10 minutes so frustrating	buy laptop crashes every 10 minutes frustrating	buy laptop crash everi 10 minut frustrat	buy laptop crash every 10 minute frustrating
8	The camera quality is excellent! 📷 Love the night mode. 🌙 ✨	the camera quality is excellent! 📷 love the night mode. 🌙 ✨	the camera quality is excellent love the night mode	camera quality excellent love night mode	camera qualiti excel love night mode	camera quality excellent love night mode
9	Meh... the product is just average. 😐 I expected more for this price.	meh... the product is just average. 😐 i expected more for this price.	meh the product is just average i expected more for this price	meh product average expected price	meh product averag expect price	meh product average expected price
10	Great customer service! 🛠️ They replaced my faulty item within 24 hours.	great customer service! 🛠️ they replaced my faulty item within 24 hours.	great customer service they replaced my faulty item within 24 hours	great customer service replaced faulty item within 24 hours	great custom servic replac faulti item within 24 hour	great customer service replaced faulty item within 24 hour

Love it Right????, I love it too

Some Discussion

Text -> Turn into numbers -> build a model -> train the model to find patterns -> use patterns (make predictions)

That's the flow





Everything in a tip

You've successfully completed text preprocessing on customer feedback data. We've covered:

- ✓ Lowercasing – Ensuring consistency in text.
- ✓ Removing Punctuation & Emojis – Cleaning unnecessary characters.
- ✓ Stopword Removal – Keeping only meaningful words.
- ✓ Stemming & Lemmatization – Reducing words to their root form for better NLP.



Let's move on

Text -> turn into numbers -> build a model -> train the model to find patterns -> use patterns (make predictions)

If you understand everything up to this point, identify yourself as a Machine Learning Engineer specializing in Natural Language Processing!



QUESTIONS AND ANSWER



Time for quick Brain Teaser

We've cleaned, transformed, and optimized text like real NLP pros! Now, let's see how well you followed along. 🤔

Question: Given this raw text: 👉 "I'm **LOVING** this **NLP session!!!** 😍 🔥 It's **AMAZING** how we can clean text properly. But... is it always the best approach?"

If we apply the following preprocessing steps, what will be the output at each stage?

❶ Lowercasing: What does the text look like after converting to lowercase?

Time for quick Brain Teaser



❶ Lowercasing: What does the text look like after converting to lowercase?

"I'm LOVING this NLP session!!! 🤩 🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

✅ "I'm loving this nlp session!!! 🤩 🔥 it's amazing how we can clean text properly. but... is it always the best approach?"

Time for quick Brain Teaser

Question: Given this raw text: 

"I'm LOVING this NLP session!!!   It's AMAZING how we can clean text properly. But... is it always the best approach?"

If we apply the following preprocessing steps, what the output?

2 Removing Punctuation & Emojis: What's left after getting rid of #,!,   , and other punctuation marks?

Time for quick Brain Teaser

② Removing Punctuation & Emojis: What's left after getting rid of #,!, 🤩 🔥 , and other punctuation marks?

"I'm LOVING this NLP session!!! 🤩 🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

✅ "im loving this nlp session its amazing how we can clean text properly but is it always the best approach"



Time for quick Brain Teaser 🔍

Question: Given this raw text: 🙌

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

If we apply the following preprocessing steps, what the output?

3 Stopword Removal: Which words will be removed, and what remains?

Time for quick Brain Teaser

3 Stopword Removal: Which words will be removed, and what remains?

"I'm LOVING this NLP session!!! 🥰🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

✓ "loving nlp session amazing clean text properly always best approach"

Time for quick Brain Teaser

Question: Given this raw text: 🙌

"I'm LOVING this NLP session!!! 🥰 🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

If we apply the following preprocessing steps, what's the output?

4 Stemming & Lemmatization: If we use:

Porter Stemmer, how would words like "loving" and "amazing" change?

Lemmatization, what would be the base form of "loving" and "clean"?

Time for quick Brain Teaser

4 Stemming & Lemmatization: If we use:

Porter Stemmer, how would words like "loving" and "amazing" change?

"I'm LOVING this NLP session!!! 🤩 🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

loving" → "love"

"amazing" → "amaz"

"clean" → "clean" (remains the same)

✓ "love nlp session amaz clean text proper always best approach"



Time for quick Brain Teaser 🔍

4 Stemming & Lemmatization: If we use: Lemmatization (Using WordNet Lemmatizer), what would be the base form of "loving" and "clean"?

"I'm LOVING this NLP session!!! 😍🔥 It's AMAZING how we can clean text properly. But... is it always the best approach?"

"loving" → "love"

"amazing" → "amazing" (unchanged)

"clean" → "clean"

✓ "love nlp session amazing clean text properly always best approach"



Bonus Question:



We removed **Stopwords** to clean up our text. But can you think of a situation where removing **Stopwords** might actually hurt the meaning or usefulness of our data? 🤔

What can be done in such case???

💡 Hint: Think about cases where small words like "not," "is," or "to" are important in understanding the sentence.

Open-ended answers are welcomed!!!



Bonus Question:



Answer:  Yes! There are several cases where stopwords play a crucial role:

1 Sentiment Analysis – Words like "not" and "very" can change sentiment.
Example: "The movie is not bad" (removing "not" changes the meaning).

2 Question Answering – Stopwords help retain context in queries.
Example: "What is the capital of France?" makes more sense than just "capital France?"

3 Machine Translation – Removing stopwords might lead to incorrect translations.
Example: Translating "I am going to the market" without "am" and "to" could alter the meaning.

So, while removing stopwords often helps clean data, it's important to consider the task before blindly dropping them! 

Try This

 Try This!

Modify the dataset with your own text and observe how preprocessing affects different types of input.

Generate into a CSV File and share before the end of the program.

Let's keep building!   

Let's move on

Text -> turn into numbers -> build a model -> train the model to find patterns -> use patterns (make predictions)

If you get the questions and attempts
right to this extent, identify yourself as
a Machine Learning Engineer
specializing in Natural Language
Processing!



Step 2: Tokenization

What is Tokenization?

Tokenization is the process of **splitting** a **text** into individual units, called **tokens**.

These tokens can be words, subwords, characters, or even punctuation marks. It's an important step in most NLP pipelines, as it prepares the text for further processing.

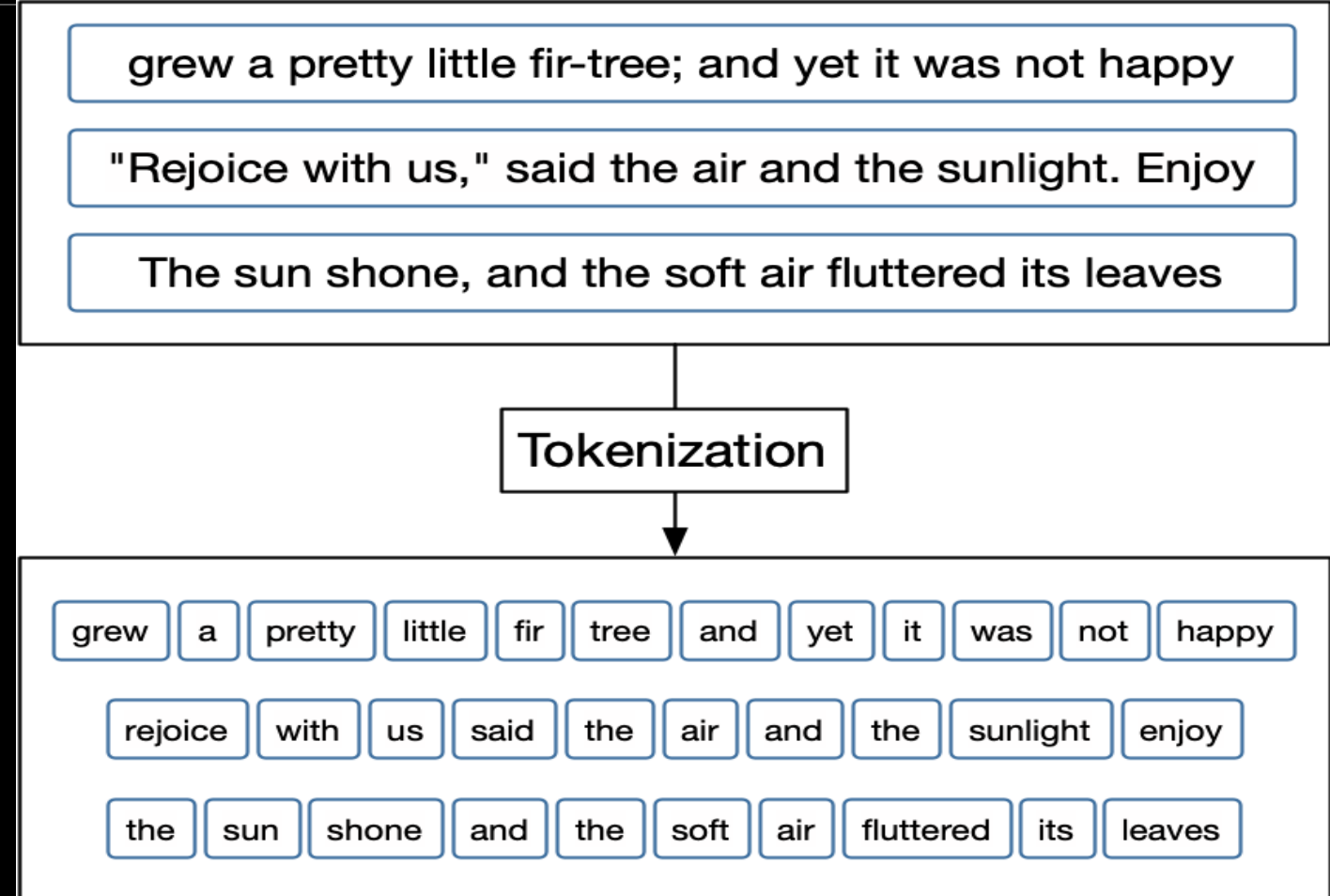
Step 2: Tokenization

What is a Token?

Tokens make it easier for computers to analyze text.

E.G: **grew** is a token.

Pretty is a token



Step 2: Tokenization

There are different types of tokenization:

- **Sentence Tokenization** (Sentence-Level Tokenization)
- **Word Tokenization** (Word-Level Tokenization)
- **Subword Tokenization**
- **Character-level Tokenization**

Step 2: Tokenization

- Sentence Tokenization** (Sentence-Level Tokenization): Splits text into individual sentences. Important for tasks like text summarization or machine translation.

Example:

"The cat sat on the mat. The dog barked."

becomes

["The cat sat on the mat.", "The dog barked."]

- Uses the `sent_tokenize()` function of the `nltk.tokenize` library

Step 2: Tokenization

Code sample:

```
import nltk
nltk.download('punkt') # Download sentence tokenizer data
#nltk.download('punkt_tab') # Download the 'punkt_tab' resource
from nltk.tokenize import sent_tokenize
```

```
text = "The cat sat on the mat. The dog barked."
```

```
sentences = sent_tokenize(text)
```

```
print("Sentences:", sentences)
```

```
Sentences: ['The cat sat on the mat.', 'The dog barked.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
```

Step 2: Tokenization

Word Tokenization (Word-Level Tokenization): Splits text into individual words. This is the most common type.

Example:

"The cat sat on the mat. The dog barked."

becomes

['The', 'cat', 'sat', 'on', 'the', 'mat', '.', 'The', 'dog', 'barked', '.']

-Uses the `word_tokenize()` function of the `nltk.tokenize` library

Step 2: Tokenization

Code sample:

```
import nltk
nltk.download('punkt') # Download sentence tokenizer data
#nltk.download('punkt_tab') # Download the 'punkt_tab' resource
from nltk.tokenize import word_tokenize
```

```
text = "The cat sat on the mat. The dog barked."
```

```
word = word_tokenize(text)
```

```
print("Word:", word)
```

```
Word: ['The', 'cat', 'sat', 'on', 'the', 'mat', '.', 'The', 'dog', 'barked', '.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
```

Step 2: Tokenization

Word tokenization and other examples are great, but what happens when our computer encounters a word it hasn't seen before?

Imagine you're learning English and you come across the word 'unbreakable.' You might not know it, but you know 'break' and 'able'.

Think of learning a new language. You might not know every single word, but you can often figure out the meaning of a new word if you recognize parts of it. Here is where Subword Tokenization comes in.

Subword tokenization is a way to handle these '**out-of-vocabulary**' (OOV) words. It breaks words into smaller parts, called **subwords**, so the computer can understand them even if it hasn't seen the whole word before.

Step 2: Tokenization

Subword Tokenization: Splits words into smaller units (subwords). This is crucial for handling out-of-vocabulary words and morphologically rich languages. Common methods include **Byte Pair Encoding (BPE)** and **WordPiece**.

Example:

"unbreakable"

might be tokenized as

["un", "break", "able"].

This allows the model to understand the meaning even if it hasn't seen "unbreakable" before, as it recognizes "break," "un," and "able."

Step 2: Tokenization

Code sample:

```
# Subword Tokenization (Byte Pair Encoding - BPE)
# Example using subword tokenization with Hugging Face tokenizers
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased") # Loads BERT's tokenizer (which uses BPE)
# Apart from BERT, i.e bert-base-uncased, there are other models like
# en_core_web_sm model used with SpaCy,
# SentencePiece by Google
# Even GPT 3 and 4 can be used for tokenization with an API in a way.

text = "I have a new unbreakable toy."
tokens = tokenizer.tokenize(text)
print(tokens) # Output will show how "unbreakable" is broken into subwords

['i', 'have', 'a', 'new', 'un', '##break', '##able', 'toy', '.']
```


Step 2: Tokenization

Character-level Tokenization: Splits text into individual characters. Useful for languages with complex morphology or when dealing with misspellings.

Example:

"cat"

becomes

["c", "a", "t"]

Character-level Tokenization are useful in Spelling Corrections, New text Generation as learned from patterns of character sequences and also when handling noisy data.

Step 2: Tokenization

Code sample:

```
# Explaining charcoer-level tokenization

def character_tokenize(text):
    """Tokenizes text into individual characters."""
    return list(text)

def character_frequency(text):
    """Calculates the frequency of each character in the text."""
    tokens = character_tokenize(text)
    frequency = {}
    for token in tokens:
        if token in frequency:
            frequency[token] += 1
        else:
            frequency[token] = 1
    return frequency

# Sample text
text = "Hello, world! 123"

# Tokenize and calculate frequency
tokens = character_tokenize(text)
freq = character_frequency(text)

# Print results
print("Tokens:", tokens)
print("Character Frequency:", freq)

Tokens: ['H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!', ' ', ' ', '1', '2', '3']
Character Frequency: {'H': 1, 'e': 1, 'l': 3, 'o': 2, ',': 1, ' ': 2, 'w': 1, 'r': 1, 'd': 1, '!': 1, '1': 1, '2': 1, '3': 1}
```

Step 2: Tokenization

While character-level tokenization offers advantages, it also has drawbacks:

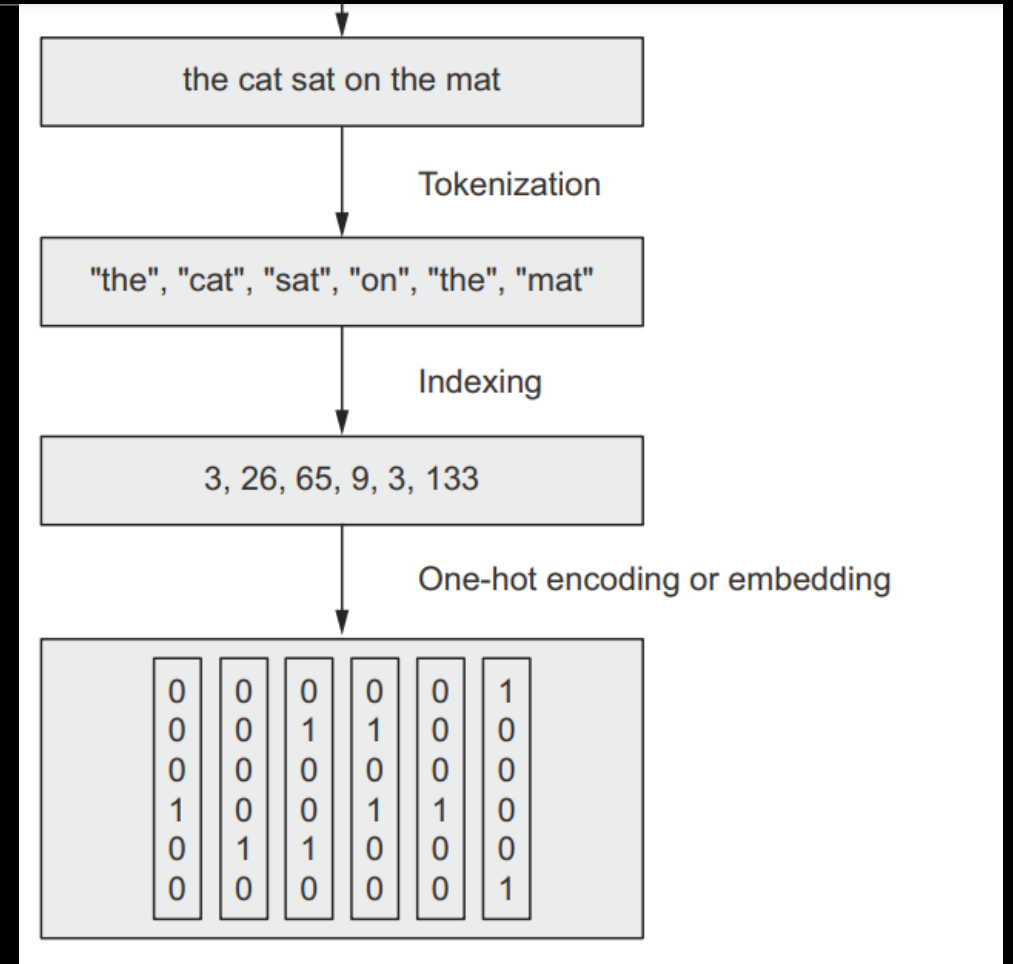
Longer Sequences: Character sequences are much longer than word sequences, which can make training models more computationally expensive.

Loss of Word-Level Meaning: It doesn't capture the semantic meaning of words as effectively as word-level or subword tokenization.

Step 3: Numerical Representation

The next step, which is almost always done immediately after tokenization, is to convert those **tokens** into **numerical representations**.

So, this step converts the **tokens** into **numbers**.



Step 3: Numerical Representation

Why is Numerical Representation necessary:

Machine learning models (and deep learning models) operate on numbers. They cannot directly process text.

The numerical representation is the bridge between human-readable text and machine-understandable data.

Almost every NLP task, requires a numerical representation of text. Without it, we can't apply any machine learning algorithms.

Step 3: Numerical Representation

How does Numerical Representation Works in NLP Projects:

Preprocessing: You'd typically tokenize your text and immediately convert the tokens to numbers (using one of the mentioned in a slide before) before feeding the data to your model.

Model Input: The numerical representation (integer encoded, one-hot encoded, or embeddings) becomes the input to your NLP model.

Model Training: The model learns patterns in these numbers, which correspond to patterns in the original text.

Step 3: Numerical Representation

There are several ways to do this:

- a) Word Embeddings (aka Word Vectors) which uses techniques like **Word2Vec, GloVe, FastText, BERT, or by creating an embedding layer and training** (i.e creating your model to be used for embedding).
- b) One-Hot Encoding
- c) Vocabulary and Integer Encoding

Step 3: Numerical Representation

Vocabulary and Integer Encoding

This method of Numerical Representation convert token to numbers in three steps

a) Build a Vocabulary: Create a list of all **unique tokens** in your training data. This list is your vocabulary.

"The cat sat on the mat. The dog barked."

becomes

["the", "cat", "sat", "on", "mat", ".", "dog", "barked"]

NB: Notice that the word **THE** appeared just once in the generated list called **Vocabulary**

Step 3: Numerical Representation

b) Integer Encoding: Assign a unique integer to each token in the vocabulary.

i.e: The == 0, cat == 1, sat == 2, on == 3, mat == 4, . == 5, dog == 6, barked == 7

c) Convert Tokens to Integers: Replace each token in your tokenized text with its corresponding integer ID from the vocabulary.

i.e [0, 1, 2, 3, 0, 4, 5, 0, 6, 7]

Step 3: Numerical Representation

Example: Let's say the sentence is "The cat sat on the mat. The dog barked."

Stage 1: Vocabulary is: ["the", "cat", "sat", "on", "mat", ".", "dog", "barked"]

Stage 2: The == 0, cat == 1, sat == 2, on == 3, mat == 4, . == 5, dog == 6, barked == 7

Stage 3: encoded token to integer becomes: [0, 1, 2, 3, 0, 4, 5, 0, 6, 7]

NB: The "The" and "the" are normally considered different tokens (unless you do lowercasing first, which was expected to have been done in this regard).

Each word is assigned a unique integer ID.

Step 3: Numerical Representation

The **Vocabulary and Integer Encoding** method is simple to implement and understand, but doesn't capture **semantic relationships**, therefore not so used in the industry like the other two methods.

Semantic Relationship means how words relates, e.g King and Prince, King and Queen.

Step 3: Numerical Representation

One-Hot Encoding

This is a traditional approach to encoding natural language numerically for processing it with a machine.

In this approach, the words of natural language in a sentence (**e.g. The bat sat on the cat:- “the”, “bat”, “sat”, “on”, “the” and “cat”**) are represented by the columns of a matrix.

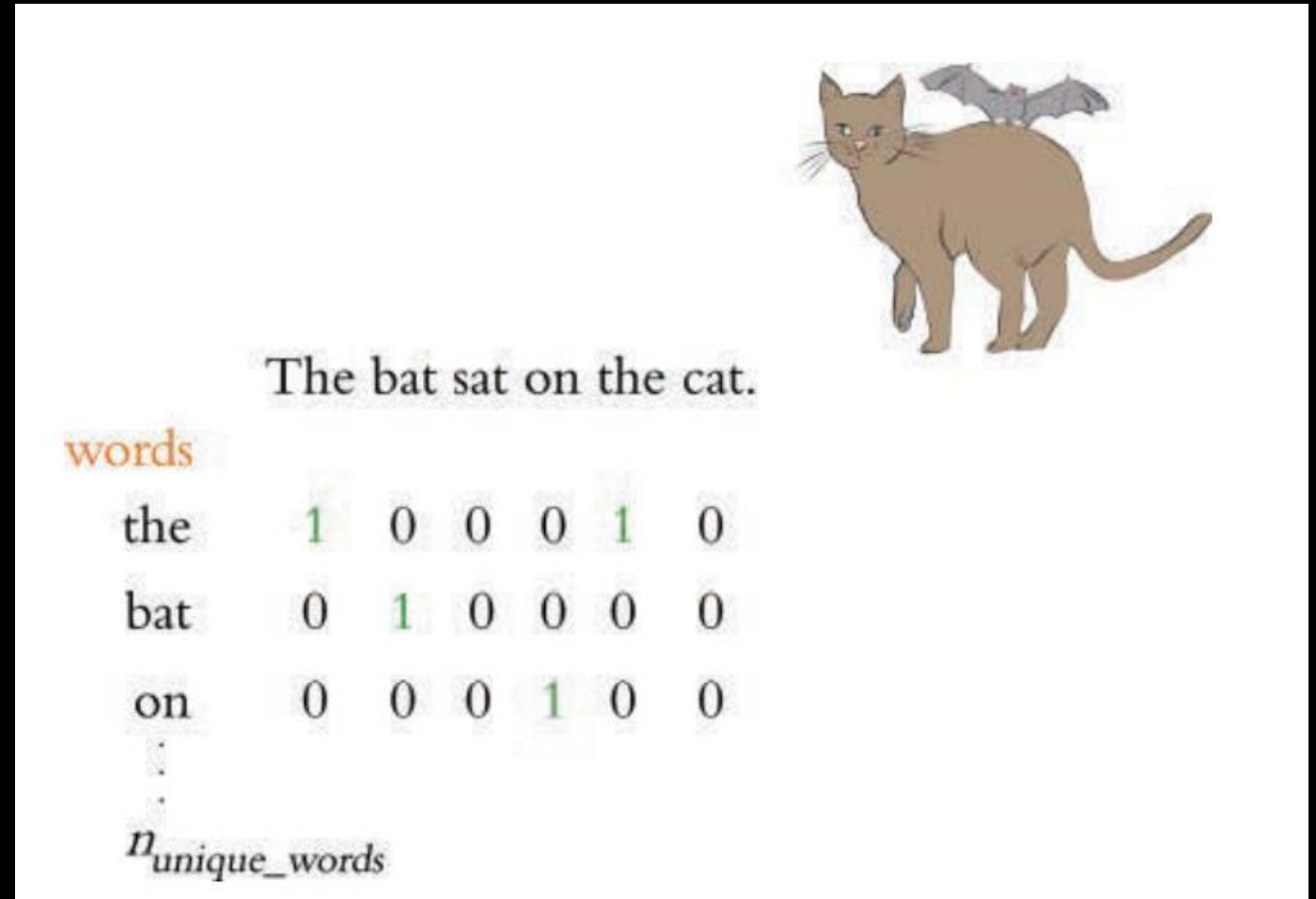
Each row in the matrix, meanwhile, represents a unique word.

Step 3: Numerical Representation

Cells within one-hot matrices consist of binary values, that is, they are a 0 or a 1.

Each column contains at most a single 1, but is otherwise made up of 0s.

In this example, the entire corpus has only **six** words, **five** of which are unique.



Step 3: Numerical Representation

In this approach,

If there are 100 unique words across the corpus of documents you're feeding into your natural language algorithm, then your matrix of one-hot-encoded words will have 100 rows.

If there are 1,000 unique words across your corpus, then there will be 1,000 rows in your one-hot matrix, and so on.

Step 3: Numerical Representation

One-hot word representations like this are fairly straightforward, and they are an acceptable format for feeding into a deep learning model (or, indeed, other machine learning models).

However, the **simplicity and sparsity of one-hot representations are limiting when incorporated into a natural language application.**

It's can be used for **simpler NLP tasks** or as a basic representation. It's also simple but doesn't capture **semantic relationships and therefore not the best option for major NLP projects.**

The bat sat on the cat.

words						
the	1	0	0	0	1	0
bat	0	1	0	0	0	0
on	0	0	0	1	0	0
⋮						
$n_{\text{unique_words}}$						

QUESTIONS AND ANSWER



The END

Day 2

DSN LEKKI-AJAH

BY ABEREJO HABEEBLAH O.

X: @HABEREJO

Next Class schedule:
Tuesday, 18th March, by
5:00pm.