Introduction to NLTK

The Natural Language Processing Toolkit (NLTK) is a widely used and powerful Python library that helps you analyze and understand human language in written text.

Why NLTK

- NLTK provides easy-to-use interfaces to over 50 corpora and lexical resources.
- It is freely available for anyone to use and contribute to.
- It supports multiple languages, i.e., Hispanic, English, French, etc.

NLTK: Installation and Data Acquisition

- Install NLTK.

```
pip install nltk
```

- Import NLTK and download all NLTK data.

```
import nltk
nltk.download('all')
```

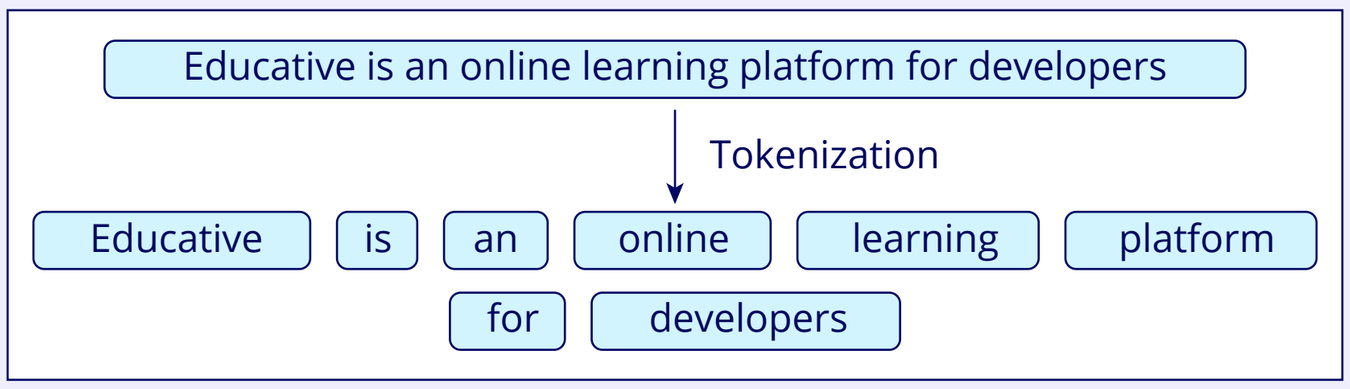
- Import a sample text.

```
# Load the raw text of the Emma novel by Jane Austen from the Gutenberg corpus
sample_text = nltk.corpus.gutenberg.raw('austen-emma.txt')

# Extract the first 500 characters from the loaded text
sample_text = sample_text[:500]
```

Text Processing

Tokenization

- The process of splitting text into individual words or sentences is called **tokenization**.



- Word tokenization of the sample text is:

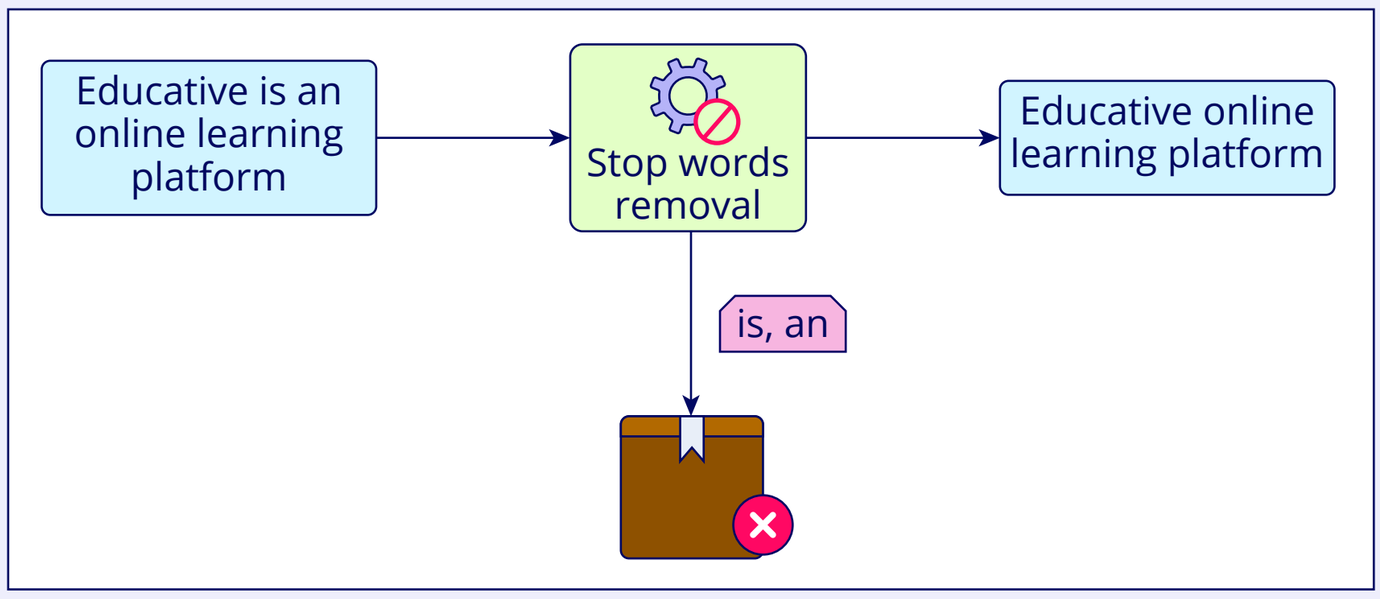
```
word_tokens = nltk.tokenize.word_tokenize(sample_text)
```

- Sentence tokenization of the sample text is:

```
sentence_tokens = nltk.tokenize.sent_tokenize(sample_text)
```

Stopwords removal

- **Stopwords** are common words that usually do not carry significant meaning (e.g., “and,” “the,” “is”).

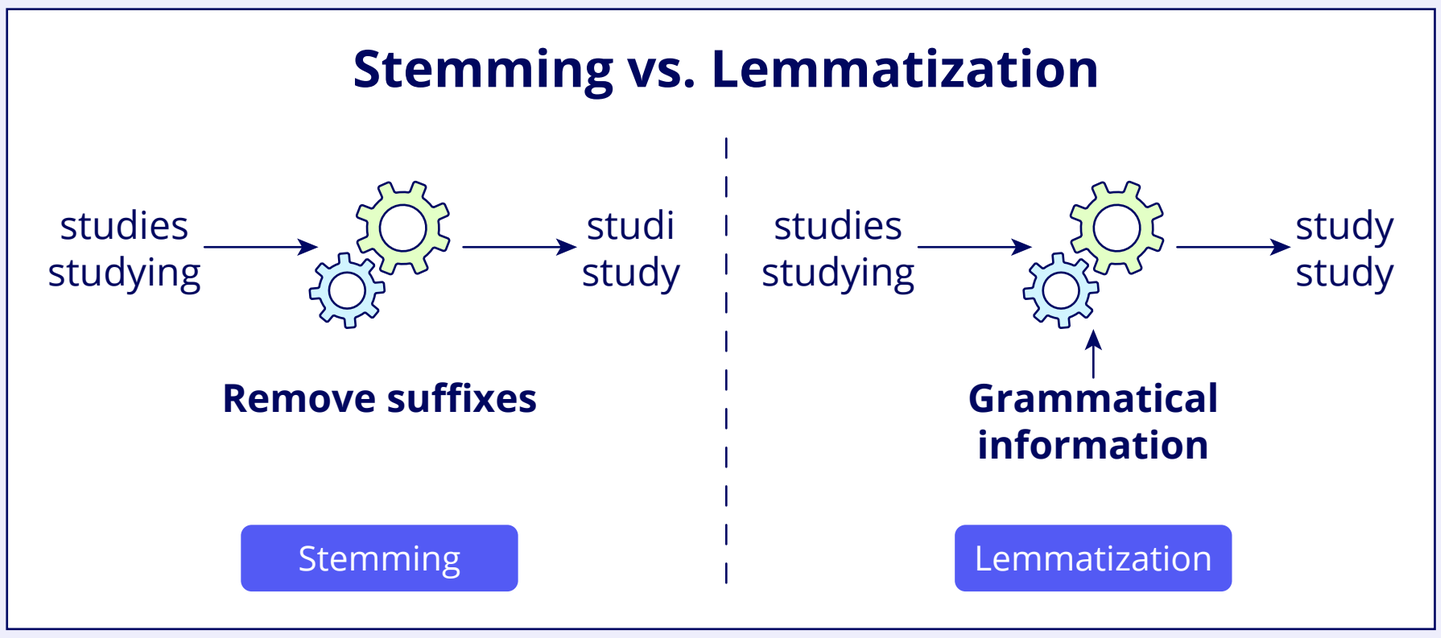


- Filter out the stopwords from word tokens.

```
# Get stopwords in English NLTK corpus
stop_words = set(nltk.corpus.stopwords.words('english'))

# Filter out stopwords from word_tokens
filtered_words = [word for word in word_tokens if word.lower() not in stop_words]
```

Stemming and lemmatization

- **Stemming** reduces words to their base form by removing suffixes (e.g., “eating” to “eat”).
- **Lemmatization** takes it one step further and reduces words to their dictionary base form, considering the context (e.g., “better” to “good”).

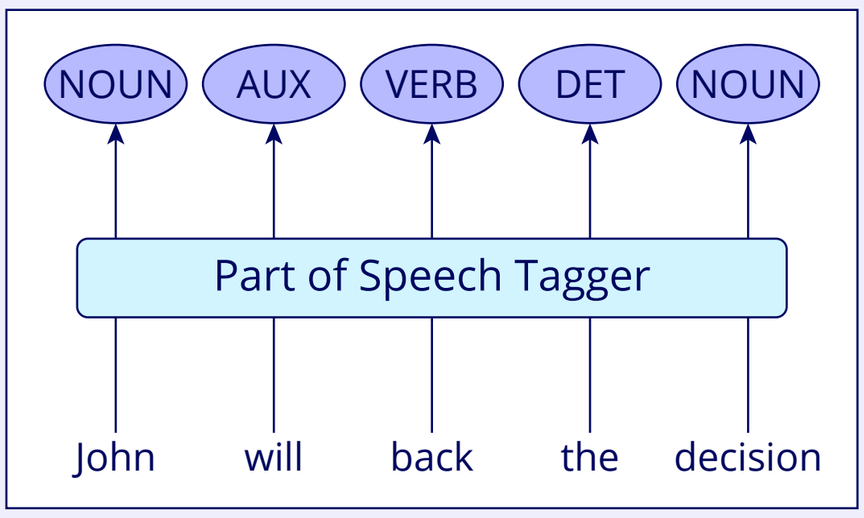


- Perform stemming and lemmatization on the filtered_words.

```
# Perform stemming on the filtered words
stemmer = nltk.stem.PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in filtered_words]

# Perform lemmatization on filtered words
lemmatizer = nltk.stem.WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word in filtered_words]
```

Part-of-speech (POS) tagging

- **POS tagging** assigns grammatical labels to each word in a sentence (e.g., noun (NN), verb (VB), adjective (JJ)).



- Extract part-of-speech tagging on each word in the text.

```
pos_tags = nltk.pos_tag(word_tokens)
print("POS Tags:", pos_tags)
```

N-grams

- N-grams are consecutively occurring sequences of n words in a given text sample.

Input text: “This is big data analytics course”

Uni-Gram

This	is	big	data	analytics	course

Bi-Gram

This is	is big	big data	data analytics	analytics course

Tri-Gram

This is big	is big data	big data analytics	data analytics course

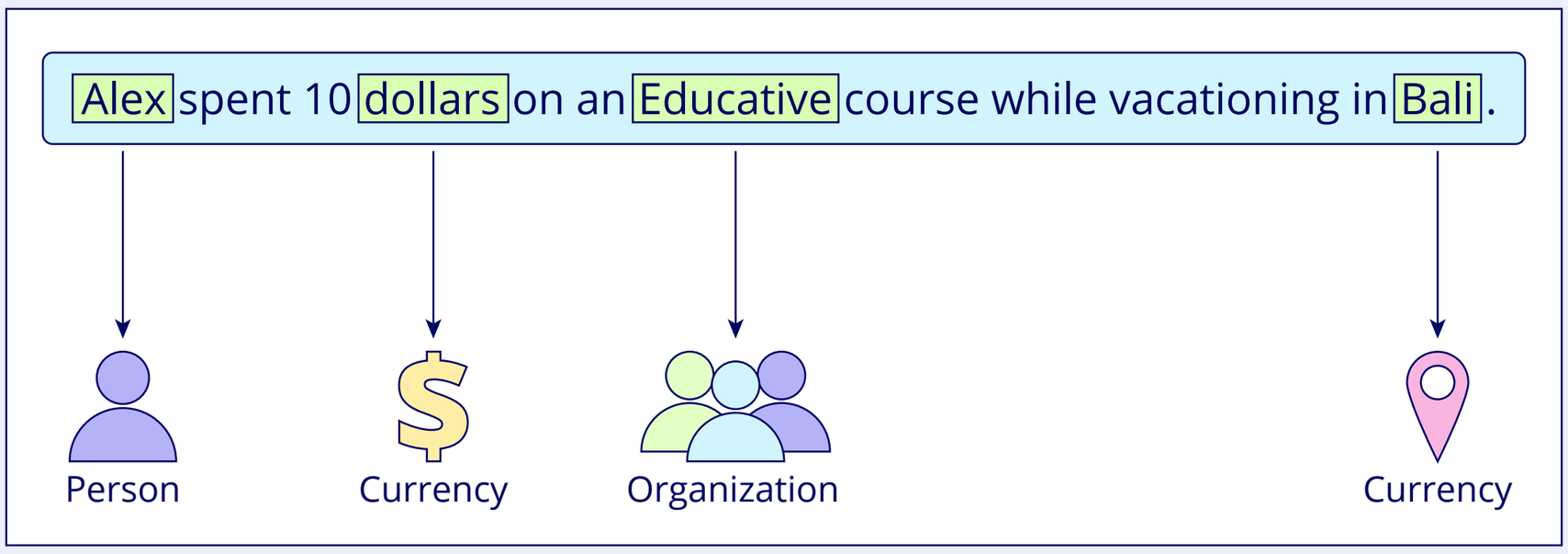
- Get unigrams, bigrams, and trigrams of a sequence of words.

```
from nltk.util import ngrams

# Generate unigrams
unigrams_list = list(ngrams(words,1))

# Generate bigrams
bigrams_list = list(ngrams(words,2))

# Generate trigrams
trigrams_list = list(ngrams(words,3))
```

NLTK: Real-World Applications

Named Entity Recognition (NER)

- **Named entity recognition (NER)** is a task that finds and classifies named entities (such as people, organizations, and locations) in text.

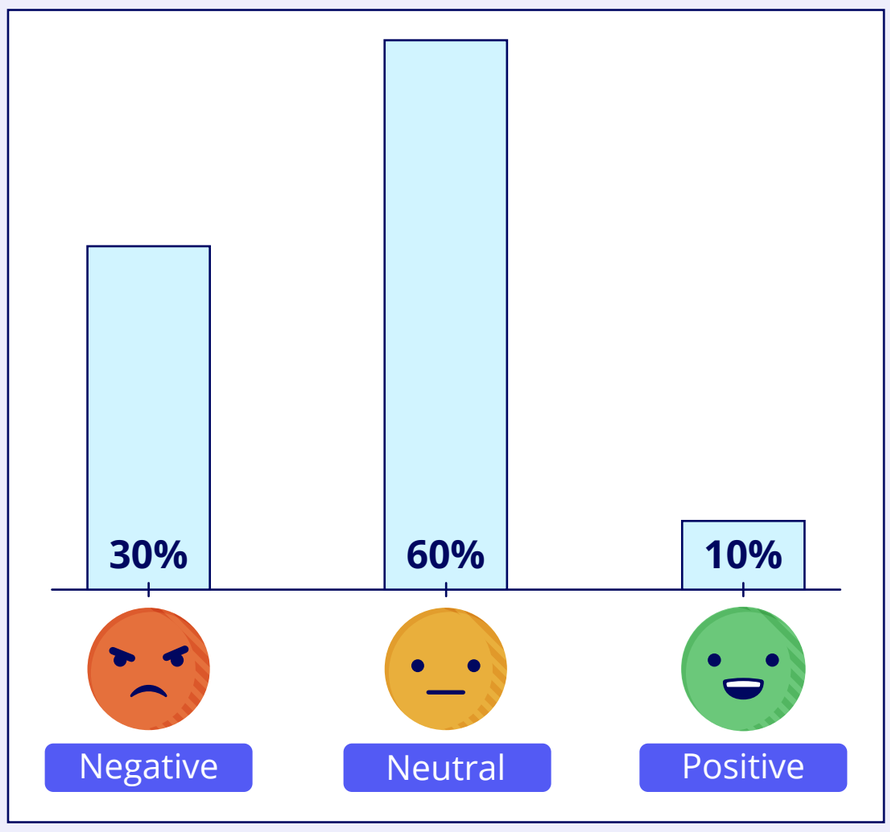


- Identify and classify entities of a sentence using NLTK.

```
sentence = "Alex was born in Hawaii."
tokens = word_tokenize(sentence)
entities = nltk.ne_chunk(nltk.pos_tag(tokens))
```

Sentiment Analysis

- **Sentiment analysis** determines the overall emotional tone, such as positive, negative, or neutral sentiment, expressed in a text.

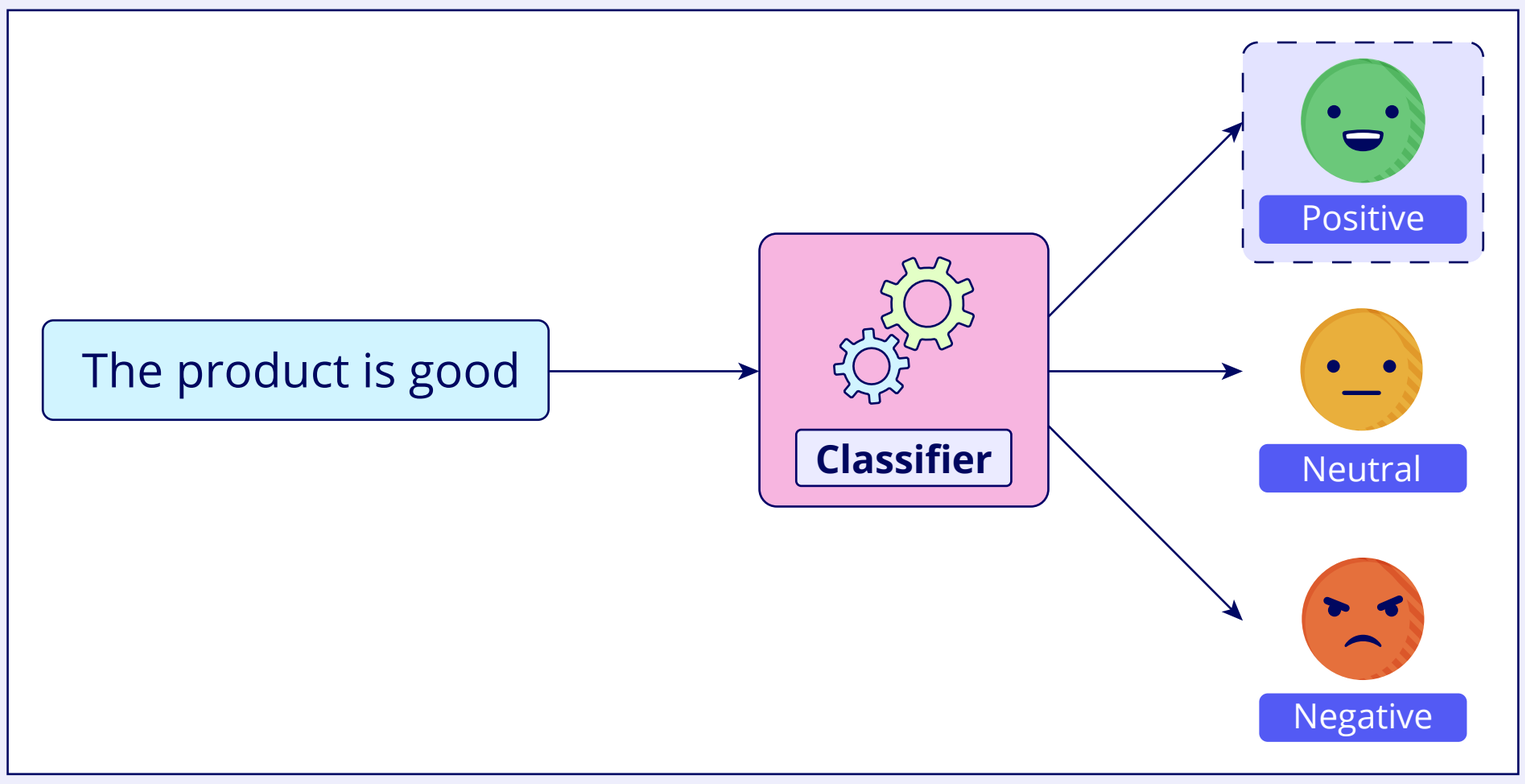


- Perform sentiment analysis on a sentence using NLTK.

```
sia = nltk.sentiment.SentimentIntensityAnalyzer()
sentence = "The perfume was absolutely amazing!"
sentiment = sia.polarity_scores(sentence)
```

Text Classification

- Text classification involves categorizing text into predefined classes or labels.



Natural Language Processing with NLTK: A Quick Guide

educative

- Text classification is used in spam detection, sentiment analysis, news article categorization (classifying news articles by topic), and document summarization (identifying key points based on category).
- Use sentiment analysis to extract features from text and train **`NaiveBayesClassifier`** with those features for text classification.

```
# Data
positive_reviews = [
    "Loved it!", "Great acting, engaging story!", "Beautiful visuals, must-see!"
]

negative_reviews = [
    "Disappointing!", "Boring plot, bad acting.", "Waste of time!",
]

# Feature extraction (basic word presence)
def word_feats(text):
    words = word_tokenize(text)
    features = {}
    analyzer = nltk.sentiment.SentimentIntensityAnalyzer()
    sentiment = analyzer.polarity_scores(text)
    features['neg'] = sentiment['neg']  # Negative sentiment score
    features['neu'] = sentiment['neu']  # Neutral sentiment score
    features['pos'] = sentiment['pos']  # Positive sentiment score
    return features

# Combine reviews and labels into training data
documents = []
for review in positive_reviews:
    documents.append((word_feats(review), "pos"))
for review in negative_reviews:
    documents.append((word_feats(review), "neg"))

# Train a NaiveBayesClassifier
classifier = nltk.classify.NaiveBayesClassifier.train(documents)

# Classify a new sentence
new_sentence = "The story felt amazing."
text_features = word_feats(new_sentence)
print(classifier.classify(text_features))
```