Workshop Booklet - Part 1 - NLP

What is NLP?

Natural Language Processing (NLP) is a field of artificial intelligence (AI) that focuses on the interaction between computers and humans using natural language. The ultimate objective of NLP is to enable computers to understand, interpret, and generate human language in a way that is both meaningful and contextually relevant.

Word Representations

Word representation in natural language processing (NLP) refers to the methods used to represent words as numerical vectors, enabling machines to process and understand language.

Common methods to word representation:

- One-hot encoding
- Word Embeddings
- Word2Vec

Sentence Segmentation

Sentence segmentation is a crucial step in the natural language processing (NLP) pipeline that involves breaking a given text into individual sentences.

Code snippet:

```
# !pip install nltk
import nltk
nltk.download('punkt')

text = "Welcome to the LLM Workshop organized by CSI. This workshop is designed
to introduce you to Large Language Models (LLMs) and their functionalities."

sentences = nltk.sent_tokenize(text)
print(sentences)
```

Vocabulary

A vocabulary refers to the set of all unique words or tokens present in a given corpus (collection of documents). The size and composition of the vocabulary can significantly impact the efficiency and effectiveness of NLP models.

Code snippet:

```
def vocabulary(corpus):
    vocab = set()
    for document in corpus:
        vocab.update(document.split())
    return list(vocab)

documents = ["CSI's LLM Workshop: NLP, Transformers, LLM" ,
                        "Hands-on Learning Await"]

corpus_vocabulary = vocabulary(documents)
print("Vocabulary:", corpus_vocabulary)
```

Tokenization

Tokenization is a crucial step in natural language processing (NLP) pipelines, occurring after text preprocessing and before various downstream tasks. Tokenization involves breaking down a given text into smaller units called tokens, which could be words, sub-words, or characters.

Code Snippet:

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

text = "Did you know? LLMs speak every language, including emoji! 😀 🎭"

tokens = nltk.word_tokenize(text)
print(tokens)
```

Parts of Speech Tagging

Part-of-speech (POS) tagging, also known as grammatical tagging or word-category disambiguation, involves assigning a specific grammatical category (such as noun, verb,

adjective, etc.) to each word in a given text.

Code Snippet:

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

text = "GPT-3: Knows more words than me."

tokens = nltk.word_tokenize(text)
pos_tags = nltk.pos_tag(tokens)
print(pos_tags)
```

Stemming

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP).

Code Snippet:

```
from nltk.stem import PorterStemmer
stemming=PorterStemmer()

words=
["eating","eats","eaten","writing","writes","programming","programs","finally","f
inalized"]

for word in words:
    print(word+" ----> "+stemming.stem(word))
```

Lemmatization

Lemmatization technique is like stemming. The output we will get after lemmatization is called ‘lemma’, which is a root word rather than root stem, the output of stemming. After lemmatization, we will be getting a valid word that means the same thing.

Code Snippet:

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

```
lemmatizer = nltk.stem.WordNetLemmatizer()

words=
["eating","eats","eaten","writing","writes","programming","programs","finally","f
inalized"]

for word in words:
    print(word+" ----> "+lemmatizer.lemmatize(word,pos='v'))
```

Stop Words

Stop words are words that are commonly used in a language but are typically considered to be of little value in text analysis because they don't carry significant meaning. These words are often filtered out or excluded from the text during the preprocessing stage of an NLP pipeline. Common stop words include articles, prepositions, conjunctions, and other frequently occurring words.

Code Snippet:

```
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

text = "Stop words are common words that are often excluded in text processing"

stop_words = set(stopwords.words('english'))
tokens = nltk.word_tokenize(text)
filtered_tokens = [token for token in tokens if token.lower() not in stop_words]

print(filtered_tokens)
```

Dependency Parsing

Dependency parsing is a critical component in natural language processing (NLP) pipelines that aims to analyze the syntactic structure of a sentence by identifying the grammatical relationships between words. The result of dependency parsing is a tree-like structure known as a dependency tree, where each word in the sentence is a node, and the edges represent the syntactic relationships or dependencies between them.

Code Snippet:

```
# !pip install spacy
import spacy
nlp = spacy.load("en_core_web_sm")

text = "Dependency parsing reveals the grammatical structure of a sentence."

doc = nlp(text)
for token in doc:
    print(token.text, token.dep_, token.head.text)
```

Noun Phrases

Noun phrases are groups of words that function as a single unit and include a noun (or pronoun) along with its modifiers. These phrases often represent a person, place, thing, or idea. Extracting noun phrases is useful for various NLP applications, such as information extraction, text summarization, and syntactic analysis.

Code Snippet:

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

text = "Noun phrases are groups of words centered around a noun."

doc = nlp(text)
noun_phrases = [chunk.text for chunk in doc.noun_chunks]
print(noun_phrases)
```

Named Entity Recognition

Named Entity Recognition (NER) involves identifying and classifying entities, such as names of people, organizations, locations, dates, monetary values, and other specific types of entities, in a given text. The goal of NER is to extract structured information from unstructured text and categorize it into predefined classes.

Code Snippet:

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

text = "The first and only LLM workshop in VIT, Pune, is currently underway,
```

```
organized by CSI."

doc = nlp(text)
entities = [(ent.text, ent.label_) for ent in doc.ents]
print(entities)
```

Coreference Resolution

Coreference resolution involves identifying when two or more expressions in a text refer to the same entity. The primary goal is to link different mentions (expressions or phrases) that refer to the same real-world entity, allowing systems to understand the relationships between various parts of a text. check this out Demo

Code Snippet:

```
# !pip install fastcoref
from fastcoref import FCoref
coref = FCoref(device='cuda:0')

text = 'We are so happy to see you at our LLM Workshop. This Workshop will get
you NLP, Transformers, LLM ready. The mentors will guide you throughout the
workshop on hands-on experience.'

preds = coref.predict(texts=text)
print(preds)
```

Bag of Words

In the Bag of Words model, a document is represented as an unordered set of its words, disregarding grammar and word order but keeping track of the frequency of each word. The model assumes that the presence or absence of words in a document is important, but their order is not considered.

Steps involved:

1. Tokenization
2. Vocabulary Construction
3. Vectorization

Example:

Consider two documents:

```
Document 1: "The cat in the hat."

Document 2: "The quick brown fox jumps over the lazy dog."
```

Vocabulary:

```
{The, cat, in, hat, quick, brown, fox, jumps, over, lazy, dog}
```

Bag of Words Representation:

```
Document 1: [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

Document 2: [1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
```

N-gram

N-grams are contiguous sequences of n items (words, characters, or tokens) from a given sample of text or speech. The most common types of n-grams are bigrams (2-grams), trigrams (3-grams), and unigrams (1-grams).

Code Snippet:

```python
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np

corpus = ["The cat in the hat.", "A dog in the yard.", "Bird on the tree."]

vectorizer = CountVectorizer(ngram_range=(1, 3))
all_sentences_vectors = vectorizer.fit_transform(corpus).toarray()
print("N-grams: ", vectorizer.get_feature_names_out())
print("Vectors: " , all_sentences_vectors)
print("Vocabulary: " , vectorizer.vocabulary_)
```

Bigrams (2-grams)

Bigrams are sequences of two consecutive words or tokens in a text.

Example:

For the sentence "The cat in the hat," the bigrams would be {"The cat", "cat in", "in the", "the hat"}.

What's Next?

Workshop Booklet - Part 2 - Transformers