Day-17

26-6-2024

TOPICS:

Basics of NLP

RNN

NLP: NLP, or Natural Language Processing, is a type of computer technology that helps computers understand and work with human language. It's like teaching computers to understand what people say or write. With NLP, computers can do things like translate languages, figure out if a piece of writing sounds positive or negative, summarize text, and more.

A term **TOKENIZATION**,in the realm of Natural Language Processing (NLP) and machine learning, refers to the process of converting a sequence of text into smaller parts, known as tokens.

Explanation:

Downloading 'punkt': nltk.download('punkt') downloads the Punkt tokenizer models, which are pre-trained models for tokenizing text. Tokenizing Text: word_tokenize(text) splits the text into individual words and punctuation marks

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

text = "Hello world! Welcome to NLP."
tokens = word_tokenize(text)
print(tokens)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['Hello', 'world', '!', 'Welcome', 'to', 'NLP', '.']
```

2. Removing Stop Words

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

stop_words = set(stopwords.words('english'))
words = word_tokenize("This is a simple NLP example.")
filtered_words = [word for word in words if word.lower() not in stop_words]
print(filtered_words)
```

```
['simple', 'NLP', 'example', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

3. Stemming

Stemming is the process of reducing words to their base or root form.

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
words = ["running", "jumps", "easily", "fairly"]
stemmed_words = [stemmer.stem(word) for word in words]
print(stemmed_words)
```

```
['run', 'jump', 'easili', 'fairli']
```

4. Part-of-Speech Tagging

POS tagging assigns parts of speech to each word in a sentence.

```
nltk.download('averaged_perceptron_tagger')

sentence = "The quick brown fox jumps over the lazy dog."
pos_tags = nltk.pos_tag(word_tokenize(sentence))
print(pos_tags)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), (
```

5. Named Entity Recognition

NER identifies named entities in text.

```
#This imports the spaCy library and loads the English language model ("en_core_web_sm").

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

text = "Apple is looking at buying U.K. startup for $1 billion"
doc = nlp(text)
#Iterates over the entities recognized in the processed document (doc) and prints each entity's text (ent.text) along with its label (ent
for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
Apple ORG
U.K. GPE
$1 billion MONEY
```

RNN(Recurrent Neural Network):

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far.

Basic RNN Architecture A basic RNN consists of the following components:

1. Input Layer The input layer takes the sequential data (e.g., a series of words or characters) and feeds it into the network.

Example: If you are processing a sentence, each word or character in the sentence will be fed into the RNN one at a time.

2. Hidden Layer The hidden layer processes each input in the sequence and maintains a hidden state, which is updated at each time step.

Hidden State: A set of neurons that store information about previous inputs in the sequence. The hidden state is updated using the current input and the previous hidden state.

Activation Function: Usually, a non-linear function such as tanh or ReLU is applied to compute the hidden state.

3. Output Layer The output layer produces the output at each time step, based on the current hidden state.

TYPE OF RNN

1.Long Short-Term Memory (LSTM) Networks: LSTMs are a type of RNN designed to handle long-term dependencies. They introduce a memory cell that can maintain its state over time, and gates that control the flow of information.

2.(GRU) Gated Recurrent Unit:

A. GRU is a type of recurrent neural network (RNN) that uses gating mechanisms to process sequential data. It's to remember long-term dependencies and forget irrelevant information