Advanced Computational Linguistics

Experiment No 4

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: K – K1 / D11

Aim: Implement Information Retrieval for extracting Text from Webpages and Image.

Theory:

Information extraction (IE) in natural language processing (NLP) refers to the process of automatically extracting structured information from unstructured or semi-structured text data. The goal is to convert the textual data into a more organized and structured form that can be easily analysed and processed by machines. Information extraction typically involves identifying entities, relationships, and attributes within the text.

Here are the key components and steps involved in information extraction:

1. **Named Entity Recognition (NER):**
   - Named Entity Recognition is the task of identifying and classifying specific entities mentioned in the text, such as names of people, organizations, locations, dates, percentages, etc.
   - For example, in the sentence "Apple Inc. was founded by Steve Jobs in Cupertino," NER would identify "Apple Inc." as an organization, "Steve Jobs" as a person, and "Cupertino" as a location.

2. **Relation Extraction:**
   - Relation extraction involves identifying and classifying relationships between entities in the text.
   - For example, from the sentence "Barack Obama was born in Hawaii," relation extraction would determine that "Barack Obama" is related to "Hawaii" through the "born_in" relationship.

3. **Event Extraction:**
   - Event extraction focuses on identifying events and their participants from the text.
   - For instance, from the sentence "Apple unveiled its new product," event extraction would identify "Apple" as the entity performing the action "unveiled" and "new product" as the event.

4. **Attribute Extraction:**
    - Attribute extraction involves identifying descriptive attributes associated with entities.
    - In the sentence "The Eiffel Tower is a tall iron structure in Paris," attribute extraction would identify "tall" and "iron" as attributes of "Eiffel Tower."

5. **Dependency Parsing:**
    - Dependency parsing analyzes the grammatical structure of a sentence to identify how words depend on each other.
    - It helps in understanding the relationships between words and their roles in the sentence.

6. **Coreference Resolution:**
    - Coreference resolution identifies when different expressions in the text refer to the same entity.
    - For instance, resolving that "he" in the sentence "John is an engineer. He builds bridges" refers to the same person.

7. **Template Filling:**
    - Template filling involves populating predefined templates with extracted information to create structured data.
    - For instance, from the sentence "Microsoft was founded by Bill Gates in 1975," the template "ORG was founded by PERSON in YEAR" can be filled to create structured data.

8. **Information Integration:**
    - After extracting relevant information, it can be integrated with existing knowledge bases or databases to enhance the understanding of relationships and context.

Information extraction has applications in various fields, including text mining, data enrichment, question answering, sentiment analysis, knowledge graph construction, and more. It often involves a combination of rule-based methods, machine learning techniques, and domain-specific knowledge to accurately extract and structure information from textual data.

Lab Assignment to be performed:

- Dataset: Select any Text Paragraph containing relationships between various entities
- Exercise 1: Perform Name Entity Recognition using NLTK
- Exercise 2: Perform Relationship Extraction using NLTK

NAME: UMANG KIRIT LODAYA

SAP ID: 60009200032

BATCH: K - K1 / D11

In [1]:
```
import pandas as pd
```

In [2]:
```
data = pd.read_excel('/kaggle/input/ner-dataset/NER Dataset lsx')
data['sentences'] = data['Person'] + " " + data['Action or Achievement']
data.head()
```

Out[2]:

	Person	Action or Achievement	Verb	Label	sentences
0	Marie Curie	discovered radium	discovered	found	Marie Curie discovered radium
1	William Shakespeare	wrote 'Hamlet'	wrote	wrote	William Shakespeare wrote 'Hamlet'
2	Neil Armstrong	walked on the moon	walked_on	walked	Neil Armstrong walked on the moon
3	Thomas Edison	invented the light bulb	invented	found	Thomas Edison invented the light bulb
4	Rosa Parks	refused to give up her bus seat	refused_to	refused	Rosa Parks refused to give up her bus seat

In [ ]:
```
import nltk
from nltk import sent_tokenize, word_tokenize, pos_tag, ne_chunk
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

In [4]:
```
text = "Apple Inc. was founded by Steve Jobs and Steve Wozniak in Cupertino, California in 1976. It is a technology company."
```

In [ ]:
```
for text in data['sentences']:
    sentences = sent_tokenize(text)
    words = [word_tokenize(sentence) for sentence in sentence
s]
    pos_tags = [pos_tag(sentence) for sentence in words]
    named_entities = [ne_chunk(tagged_sentence) for tagged_se
ntence in pos_tags]
    print(named_entities)
    for ne_tree in named_entities:
        for subtree in ne_tree:
            if type(subtree) == nltk.Tree:
                entity = " ".join([token for token, pos in su
btree.leaves()])
                entity_type = subtree.label()
                print(f"\tEntity: {entity}, Type: {entity_typ
e}")
```

In [6]:
```
print(*words, sep='\n')
```

```
['Wright', 'brothers', 'Wilbur', 'and', 'Orville', 'pioneer
d', 'powered', 'flight']
```

In [7]:
```
pos_tags = [pos_tag(sentence) for sentence in words]
print(*pos_tags, sep='\n')
```

```
[('Wright', 'NNP'), ('brothers', 'NNS'), ('Wilbur', 'NNP'),
('and', 'CC'), ('Orville', 'NNP'), ('pioneered', 'VBD'), ('po
wered', 'JJ'), ('flight', 'NN')]
```

In [8]:
```
# Perform named entity recognition
named_entities = [ne_chunk(tagged_sentence) for tagged_senten
ce in pos_tags]
```

In [9]:
```
# Extract and display named entities
for ne_tree in named_entities:
    for subtree in ne_tree:
        if type(subtree) == nltk.Tree:
            entity = " ".join([token for token, pos in subtre
e.leaves()])
            entity_type = subtree.label()
            print(f"Entity: {entity}, Type: {entity_type}")
```

```
Entity: Wright, Type: GPE
Entity: Wilbur, Type: PERSON
Entity: Orville, Type: PERSON
```

In [10]:
```
import numpy as np

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_seque
nces
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Den
se
```

In [12]:
```
sentences = [sentence for sentence in data['sentences']]
labels = [label for label in data['Label']]

tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
```

In [13]:
```
# Convert sentences to sequences of word indices
sequences = tokenizer.texts_to_sequences(sentences)
```

In [14]:
```
# Padding sequences for consistent input length
max_sequence_length = max([len(sequence) for sequence in sequ
ences])
padded_sequences = pad_sequences(sequences, maxlen = max_sequ
ence_length, padding = 'post')
```

In [15]:
```
# Create labels dictionary
label_indices = {v: k for k, v in enumerate(labels)}
labels_encoded = [label_indices[label] for label in labels]
```

In [16]:
```
labels_one_hot = tf.keras.utils.to_categorical(labels_encod
d)
```

In [17]:
```
# Build RNN model
model = Sequential()
model.add(Embedding(input_dim = len(tokenizer.word_index) +
1, output_dim = 128, input_length = max_sequence_length))
model.add(SimpleRNN(64, return_sequences=False))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(len(labels_one_hot), activation='softmax'))
```

In [18]:
```
model.compile(loss='categorical_crossentropy', optimizer='a
m', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param
#
=================================================================
 embedding (Embedding)       (None, 13, 128)           55296

 simple_rnn (SimpleRNN)      (None, 64)                12352

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 85)                2805

=================================================================
Total params: 76,693
Trainable params: 76,693
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
history = model.fit(padded_sequences, labels_one_hot, epoch
= 500, validation_split = 0.3)
```

In [20]:
```
# Sample sentence for prediction
sample_sentence = "I founded Python"

# Tokenize and pad the sample sentence
sample_sequence = tokenizer.texts_to_sequences([sample_senten
ce])
sample_padded_sequence = pad_sequences(sample_sequence, maxle
n=max_sequence_length, padding='post')
```

In [21]:
```
# Predict the relation for the sample sentence
predicted_label_index = np.argmax(model.predict(sample_padded
_sequence))
predicted_label = None
for k, l in label_indices.items():
    if l == predicted_label_index:
        predicted_label = k
        break

print(f"Predicted relation: {predicted_label}")
```

```
1/1 [==============================] - 0s 208ms/step
Predicted relation: found
```