Experiment No 6

Bhuvi Ghosh
60009210191

Aim: Implement Information Retrieval for performing Name Entity Recognition, Relation Extraction and template Creation.

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

Exercise 3: Create a Template for the given text using NLTK using Information Extraction.

```
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
True
```

Named Entity Recognition:

```
text = "Barack Obama was born in Hawaii. He was elected president of the United States in 2008."
```

```
[7] import nltk
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
[8] words = word_tokenize(text)
pos_tags = pos_tag(words)
```

```
[9] named_entities = ne_chunk(pos_tags)
    print(named_entities)
```

```
(S
  (PERSON Barack/NNP)
  (PERSON Obama/NNP)
  was/VBD
  born/VBN
  in/IN
  (GPE Hawaii/NNP)
  ./.
  He/PRP
  was/VBD
  elected/VBN
  president/NN
  of/IN
  the/DT
  (GPE United/NNP States/NNPS)
  in/IN
  2008/CD
  ./.)
```

Relationship Extraction:

```
from nltk import word_tokenize, pos_tag, ne_chunk
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('punkt')
text = "Elon Musk founded SpaceX, a company based in California."
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
[11] words = word_tokenize(text)
     pos_tags = pos_tag(words)
```

```
[12] named_entities = ne_chunk(pos_tags)
```

```
grammar = r"""
    REL: {<NNP><VBZ><NNP>}
"""
cp = nltk.RegexpParser(grammar)
tree = cp.parse(named_entities)
print(tree)
```

```
(S
  (PERSON Elon/NNP)
  (PERSON Musk/NNP)
  founded/VBD
  (ORGANIZATION SpaceX/NNP)
  ,/,
  a/DT
  company/NN
  based/VBN
  in/IN
  (GPE California/NNP)
  ./.)
```

Template Extraction:

```
[15] from nltk import word_tokenize, pos_tag
```

```
[18] text = "John Doe lives in New York and works at Google."
     words = word_tokenize(text)
     pos_tags = pos_tag(words)
     grammar = r"""
         NP: {<DT>?<JJ>*<NNP>+}
         PLACE: {<IN><NNP>+}
     """
     cp = nltk.RegexpParser(grammar)
     tree = cp.parse(pos_tags)
```

```
def extract_template(text):
    template = text
    words = word_tokenize(text)
    pos_tags = pos_tag(words)
    named_entities = ne_chunk(pos_tags)

    for subtree in named_entities.subtrees():
        if subtree.label() == 'PERSON':
            name = ' '.join([token for token, pos in subtree.leaves()])
            template = template.replace(name, "[PERSON]")
        elif subtree.label() == 'GPE':
            place = ' '.join([token for token, pos in subtree.leaves()])
            template = template.replace(place, "[PLACE]")

    return template
```

```
print(extract_template(text))
```

```
[PERSON] Doe lives in [PLACE] and works at Google.
```