Project: Web scrapping, knowledge base construction

Part 1: Web scrapping and knowledge base construction

Duration: 3 sessions(4.5H) / 6 sessions (9h)

Deliverables:
This lab is considered as a preparation phase for your project. You need to integrate this lab session in your code and report for the project. Consider regester your work with the following forms:

- A Python script (or Jupyter Notebook) that performs the complete pipeline from text cleaning to knowledge graph construction.
- A brief report summarizing your methodology, challenges faced, and a few sample queries from your knowledge graph along with their results.

Objective:
In this lab, you will construct a pipeline for knowledge graph construction from raw text data. You will perform the following tasks:

1. **Text Cleaning & Preprocessing:** Clean and normalize the provided text dataset.
2. **Named Entity Recognition (NER):** Train a CRF model to extract named entities from the cleaned text, compare with spaCy's `en_ner_conll03` pre-trained NER model.
3. **Relation Extraction (RE):** Use spaCy's "en_core_web_sm" model to extract relations between the identified entities.
4. **Knowledge Graph Building:** Convert the extracted entities and relations into RDF triples and load them into a graph database.
5. **Web Scrapping:** Fetch and process web content from websites.

The tasks involves many NLP techniques that you haven't learnt so far, therefore, you are encouraged to use Genarative AI tools (such as ChatGPT).

Environment Setup

Before starting with the tasks, ensure that your environment is properly set up. You will need the following tools and libraries:

1. **Python:** Make sure you have Python 3.6 or higher installed.
2. **Jupyter Notebook:** Install Jupyter Notebook for running and documenting your code.
3. **Google Colab:** Alternatively, you can use Google Colab, which provides a free cloud-based environment with pre-installed libraries.
4. **Libraries:** Install the necessary Python libraries using pip:
    - `datasets` for loading datasets from Hugging Face
    - `sklearn-crfsuite` for training CRF models
    - `transformers` for using BERT and other transformer models
    - `BeautifulSoup`, `NLTK`, or `spaCy` for text preprocessing
    - `Rdflib` for knowledge graph construction

Example commands to install the required libraries:

```
pip install datasets sklearn-crfsuite transformers beautifulsoup4 nltk
spacy
```

To install these libraries in Google Colab, use the following command in the code block (just add ! at the begining):

```
!pip install datasets sklearn-crfsuite transformers beautifulsoup4 nltk
spacy
```

Datasets:
You will use two datasets for different purposes

1. **Dataset for NER and RE model training** CoNLL-2003 dataset: The CoNLL-2003 dataset is a widely-used dataset for training and evaluating named entity recognition (NER) systems. It contains annotated text for four types of entities: PERSON, ORGANIZATION, LOCATION, and MISC. The dataset is divided into training, validation, and test sets, making it suitable for developing and benchmarking NER models.

To use the CoNLL-2003 dataset with Python, you can follow these steps:

- Download the dataset from Hugging Face's `datasets` library.
- Load the dataset into your Python environment.
- Preprocess the text data as needed for your NER model.

Introduction to using Hugging Face: Hugging Face provides a `datasets` library that allows you to easily access and load a wide variety of datasets. The library simplifies the process of downloading, preprocessing, and using datasets for machine learning tasks.

Example code to load the CoNLL-2003 dataset using the `datasets` library:

```
from datasets import load_dataset

# Load the CoNLL-2003 dataset
dataset = load_dataset("conll2003")

# Access the training, validation, and test sets
train_dataset = dataset['train']
validation_dataset = dataset['validation']
test_dataset = dataset['test']

# Example: Print the first example from the training set
print(train_dataset[0])
```

2. **Data for application** You will need to build your own web scrapping program to obtain data from online websites.

Task 1: Model for NER

Instructions:

1. **Text Cleaning & Preprocessing:**
   - Remove punctuation, and any non-text elements. (you can leave the hyphen symbol `-`, because this synmole defines new words.)
   - Normalize the text by converting to lowercase, removing stop words, and applying tokenization and stemming/lemmatization.
   - Tools: Python (e.g., BeautifulSoup, NLTK or spaCy).

Tutorial: Text Preprocessing in Python

2. **Named Entity Recognition (NER):**
   - Train a Conditional Random Field (CRF) model to identify named entities such as PERSON, ORGANIZATION, LOCATION, etc. You can use libraries like `sklearn-crfsuite` for this task.
   - Use spaCy's `en_ner_conll03` pre-trained NER model to identify named entities. This model is also trained on `ConLL2003` dataset. The code for training process is available here.
   - Compare the performance of the CRF model with the spaCy pre-trained model. The performances should be compared by accuracy, precision and F1 measure.
   - Save the extracted entities along with their positions.

Example code to train a CRF model:

Instruction: For the feature extraction, you can refer to the official tutorial of sklearn-crfsuite: https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html

```
import sklearn_crfsuite
from sklearn_crfsuite import metrics
from datasets import load_dataset

# Load the CoNLL-2003 dataset
dataset = load_dataset("conll2003")
train_dataset = dataset['train']
validation_dataset = dataset['validation']
test_dataset = dataset['test']

# Define features and labels for training
# ... (feature extraction code, refer to sklearn-crfsuite tutorial) ...

# Train the CRF model
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=False
)
crf.fit(X_train, y_train)

# Evaluate the model
```

```
y_pred = crf.predict(X_test)
print(metrics.flat_classification_report(y_test, y_pred))
```

Example code to use spaCy's `en_ner_conll03` pre-trained NER model:

The pre-trained model is an zip file available through in DVL platform. You should unzip it before using.

If you are using Google Colab, it is adviced to upload the zip file first, then unzip it through `!unzip en_ner_conll03.zip`

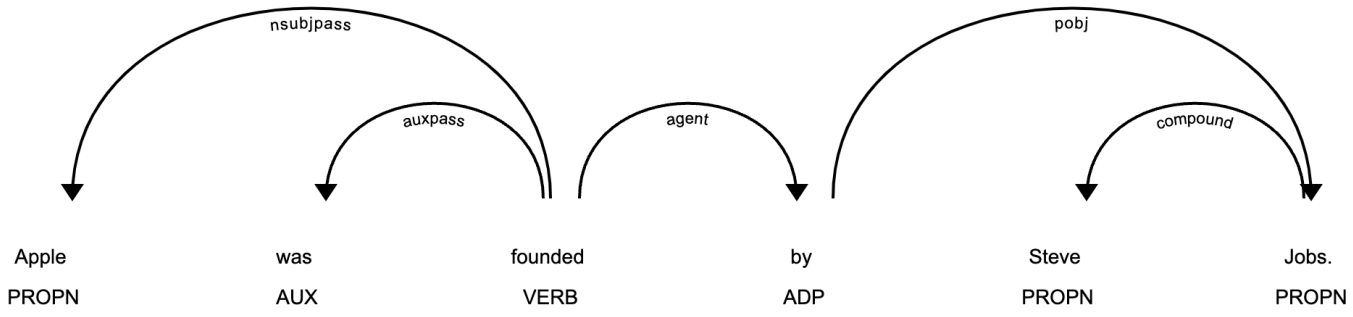
```
import spacy

# Load spaCy's pre-trained NER model
nlp = spacy.load("./en_ner_conll03")

# Example text
text = "Apple was founded by Steve Jobs."

# Process the text with spaCy
doc = nlp(text)

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

Tutorial: Named Entity Recognition with spaCy

3. **Relation Extraction (RE):**
    - Use spaCy's "en_core_web_sm" model to extract relations between entities.
    - Document the extraction rules or model setup.
    - [Optionally], align the extracted relations with ontology, such as Schema.org, DBpedia Ontology, or FOAF (You may need third part tool for this purpose, such as silkframework)

The RE task is often done through syntax dependency tags obtained by language model. We can use syntax tree to extract the relation words.

Here is an example:



In this example, `nsubjpass` indicate the token `Apple` is the subject in passive tense. In `spaCy`, we use the attribute `head` to refer to the precedent token, which is `founded`. Token `by` (which is one of `children` of

token `founded`) is tagged as `agent`, pointing to the object with tag `pobj`. Tag `compound` indicates a compound entity, making two tokens `Steve Jobs` one entity.

code for relation extraction using spaCy:

```
import spacy

# Load spaCy's pre-trained model
nlp = spacy.load("en_core_web_sm")

# Example text
text = "Apple was founded by Steve Jobs."

# Process the text with spaCy
doc = nlp(text)

# Extract relations
relations = []
for token in doc:
    if (token.dep_ == "nsubj" or token.dep_ == "nsubjpass") and
token.head.dep_ == "ROOT":
        subject = token.text
        predicate = token.head.text
        for child in token.head.children:
            if child.dep_ == "prep" or child.dep_ == "agent":
                for obj in child.children:
                    if obj.dep_ == "pobj":
                        relations.append((subject, predicate, obj.text))

print(relations)
```

(Optional) More rules to add Of course, the relations are not limited by this rule. If you find some important relation undetected, you can use spaCy's visualizer for further analysis. The figure above is generated by this tool. **Please mark the new rules you create in your report**

Why don't we use `en_ner_conll03` model?

-If you use `en_ner_conll03` model as in NER task, you may encounter errors concerning token taggs. This is because `en_ner_conll03` model is limited only in NER task, trained by **ConLL2003** dataset containing only NER labels. However,

4. **Knowledge Graph Building:**
    - Convert the extracted entities and relations into RDF triples (subject, predicate, object).
    - Load the triples into a graph database (e.g., Apache JENA or RDFLib) and perform simple SPARQL queries to verify your graph.
    - [Optionally], map entities to external identifiers using tools like **DBpedia Spotlight** for improved integration.

Example code for building a knowledge graph using RDFLib:

```
from rdflib import Graph, URIRef, Literal, Namespace
from rdflib.namespace import RDF, RDFS

# Create a new RDF graph
g = Graph()

# Define namespaces
EX = Namespace("http://example.org/")

# Add triples to the graph
g.add((URIRef(EX.Apple), RDF.type, URIRef(EX.Company)))
g.add((URIRef(EX.SteveJobs), RDF.type, URIRef(EX.Person)))
g.add((URIRef(EX.Apple), URIRef(EX.founded_by), URIRef(EX.SteveJobs)))

# Serialize the graph in RDF/XML format
print(g.serialize(format="xml"))

# Perform a SPARQL query
query = """
SELECT ?subject ?predicate
WHERE {
  ?subject ?predicate <http://example.org/SteveJobs>
}
"""
for row in g.query(query):
    print(f"{row.subject} {row.predicate} ")
```

After this step, try running your programme over the following text:

```
Star Wars IV is a Movie where there are different kinds of creatures, like
humans and wookies. Some creatures are Jedis; for instance, the human Luke
is a Jedi, and Master Yoda – for whom the species is not known – is also a
Jedi. The wookie named Chewbacca is Han’s co-pilot on the Millennium
Falcon starship. The speed of Millennium Falcon is 1.5 (above the speed of
light!)
```

Show the triples that your program extracts.

Task 2: Pipeline for Knowledge Graph Construction

Instructions:

1. **Fetch News Articles:**
   - Write a web scraping script to fetch at least 10 news articles from reuters.com. (It is adviced to choose news from a specific category to obtain NEs in common)
   - Extract the main content, title, and publication date of each article.
   - Tools: Python (e.g., BeautifulSoup, requests).

You may encounter 401 error. This is due to that a direct get request is forbidden by the site. You can use selenium to simulate a visiting with a browser (such as a google chrome). An example code is given below.

You can also scrap on other site to your own interests. But it must be in English because all the language model we used until now are only trained for English language processing.

Learn the example code to fetch news articles:

The parser of html may not work. You need to analyse the html of the page and apply your own strategy.

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from bs4 import BeautifulSoup


def fetch_reuters_articles():
    # Configure Chrome options for headless browsing (optional)
    chrome_options = Options()
    chrome_options.add_argument("--headless")  # Run Chrome in headless
mode
    chrome_options.add_argument("--no-sandbox")  # Bypass OS security
model
    chrome_options.add_argument("--disable-dev-shm-usage")  # Overcome
limited resource problems

    # Initialize the Chrome webdriver
    driver = webdriver.Chrome(options=chrome_options)

    # URL of Reuters World section
    url = "https://www.reuters.com/world/"

    # Load the page using Selenium
    driver.get(url)

    # Get the page source after JavaScript execution
    page_source = driver.page_source

    # Close the browser
    driver.quit()

    # Parse the HTML content with BeautifulSoup
    soup = BeautifulSoup(page_source, "html.parser")

    for article in soup.find_all('article', limit=10):
        title = article.find('h3').get_text()
        link = article.find('a')['href']
        article_url = f"https://www.reuters.com{link}"
        article_response = requests.get(article_url)
        article_soup = BeautifulSoup(article_response.content,
'html.parser')
        content = article_soup.find('div',
class_='StandardArticleBody_body').get_text()
        publication_date = article_soup.find('meta', {'name':
```

```
'article:published_time'})['content']

        articles.append({
            'title': title,
            'url': article_url,
            'content': content,
            'publication_date': publication_date
        })

    return articles

articles = fetch_reuters_articles()
for article in articles:
    print(article['title'], article['publication_date'])
    print(article['content'][:200])  # Print the first 200 characters of
the content
    print()
```

Tutorial: Web Scraping with BeautifulSoup

2. **Use Methods from Task 1:**
   - Apply the text cleaning and preprocessing methods from Task 1 to the fetched news articles.
   - Use spaCy's `en_ner_conll03` pre-trained NER model to identify named entities in the news articles.
   - Use spaCy's "en_core_web_sm" model to extract relations between entities in the news articles.
   - Convert the extracted entities and relations into RDF triples and build the knowledge graph using RDFLib.