Lab Session #08

Introduction

Welcome to Lab #08. This lab, we'll cover the foundations for developing *Text Mining* systems, in particular how to build NLP (Natural Language Processing) pipelines using the spaCy library for Python. We'll also look into how to link natural language text with a knowledge graph using DBpedia Spotlight, which is an important concept for grounding-based chatbots.

Natural Language Processing (NLP) is a crucial area of Artificial Intelligence that focuses on the interaction between computers and humans through natural language. The goal is to enable computers to understand, interpret, and generate human languages in a valuable way. NLP technologies are behind a variety of applications, from voice-activated assistants to sophisticated text analysis tools. Within the broader domain of NLP, *Text Mining* is a specific kind of application aimed at extracting meaningful information from text. It involves a range of techniques and tools that allow for the discovery of patterns, trends, and insights within large volumes of text, making it an invaluable asset in the arsenal of Intelligent Systems.

One of the key aspects of building intelligent systems capable of understanding human language is the ability to ground textual entities to real-world concepts. This process, known as *entity grounding*, involves linking phrases and entities found in text to a structured form of knowledge. *DBpedia Spotlight* is an open-source tool that excels in this area by associating text with the DBpedia knowledge graph. The knowledge graph acts as a vast repository of structured information, where each entity is connected with others through a network of relationships. By grounding text to such a graph, systems gain a deeper understanding of the context and significance of words, enabling more intelligent and aware interactions. The use of DBpedia Spotlight in NLP applications is essential for developing systems that not only comprehend the literal meaning of text but also appreciate its broader implications in the real world.

Follow-up Lab #07

Solution Task #1.1 (kNN Regression)

Here's a sample program for the kNN regression exercise.

Solution Task #1.2 (kNN Classification)

And here's a solution for running kNN Classification on the *scikit-learn* wine dataset.

Solution Task #3 (Search-based Chatbots)

Here's a sample program for this task from last week that finds similar questions on the Amazon QnA dataset.

Task #1: Natural Language Processing with spaCy

Preprocessing text is a very crucial step for any NLP related tasks. It involves cleaning and formatting the raw text into a form that is analyzable and understandable by NLP tools. This step can include tokenization, part-of-speech tagging, lemmatization, and removing stop words, among other processes. Effective preprocessing improves the accuracy and efficiency of subsequent NLP tasks, such as parsing, named entity recognition, and sentiment analysis. There are many frameworks (GATE, UIMA) and libraries (NLTK, CoreNLP, spaCy) available out there to facilitate with this regard. Today, we will be looking into spaCy to get you started with the basic steps.

spaCy stands out among these options for several reasons. First, it is designed for practical, real-world applications, offering robust and efficient implementations of common NLP tasks. It boasts a user-friendly API that simplifies the development process, making it accessible to both beginners and experienced practitioners. Unlike some other libraries, spaCy emphasizes performance and scalability, making it ideal for processing large volumes of text quickly. Furthermore, spaCy includes pre-trained models for multiple languages, which are continually updated and improved. This makes it an excellent choice for this course, focusing on building intelligent systems, as it equips you with the tools needed to handle advanced NLP tasks in a variety of settings.

Installing spaCy

pip: Before installing spaCy lets first make sure your pip and setuptools are up to date and install them:

```
pip install -U pip setuptools wheel
pip install -U spacy
```

Conda:

```
conda install -c conda-forge spacy
```

Downloading Language Models

Each language, and indeed different text genres within a language, can require distinct approaches for tasks like tokenization, sentence splitting, and part-of-speech (POS) tagging. To accommodate this diversity, spaCy offers a range of pre-trained language pipelines tailored to specific languages and text types. For our initial experiments, we'll focus on the English model. The model can be downloaded with the following command:

```
python -m spacy download en_core_web_sm
```

To begin using spaCy and the model you've just installed, import the spaCy library and load the language model as shown below:

```
import spacy

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

Part-of-Speech (POS) Tagging

Next, we will preprocess a sample sentence to explore the POS tags spaCy assigns. Let's process the text:

```
doc = nlp("I prefer a direct flight to Chicago.")
```

With the language model applied to our input text, spaCy returns a "doc" object enriched with various linguistic annotations. These annotations, accessible as attributes of each token, include lemmas, POS tags, and syntactic dependencies, among others. To illustrate, here's how to print out some of the basic attributes for each token:

```
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.dep_)
```

Sentence Splitting

Understanding how to segment text into sentences is an essential step in NLP, as it allows for more granular analysis and processing of language. Sentence splitting, or sentence boundary detection, is a task that involves identifying the boundaries of sentences within a larger body of text. This can be particularly challenging due to the variety of punctuation and formatting used across different languages and writing styles. However, spaCy provides efficient tools for this task, leveraging its language models to accurately determine where sentences begin and end.

Let's apply spaCy's sentence splitting capabilities to a complex paragraph. This exercise will demonstrate how spaCy handles various challenges, such as conjunctions, proper nouns, and abbreviations that might otherwise complicate sentence boundary detection.

"Superman was born on the planet Krypton and was given the name Kal-El at birth. As a baby, his parents sent him to Earth in a small spaceship moments before Krypton was destroyed in a natural cataclysm. His ship landed in the American countryside, near the fictional town of Smallville. He was found and adopted by farmers Jonathan and Martha Kent, who named him Clark Kent. Clark developed various superhuman abilities, such as incredible strength and impervious skin. His adoptive parents advised him to use his abilities for the benefit of humanity, and he decided to fight crime as a vigilante. To protect his privacy, he changes into a colorful costume and uses the alias "Superman" when fighting crime. Clark Kent resides in the fictional American city of Metropolis, where he works as a journalist for the Daily Planet. Superman's supporting characters

include his love interest and fellow journalist Lois Lane, Daily Planet photographer Jimmy Olsen, and editor-in-chief Perry White. His classic foe is Lex Luthor, who is either a mad scientist or a ruthless businessman, depending on the story."

To process the paragraph with spaCy and identify sentence boundaries, you can use the following code:

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("""Superman was born on the planet Krypton and was given the name Kal-El at birth. As a baby, his parents sent him to Earth in a small spaceship moments before Krypton was destroyed in a natural cataclysm. His ship landed in the American countryside, near the fictional town of Smallville. He was found and adopted by farmers Jonathan and Martha Kent, who named him Clark Kent. Clark developed various superhuman abilities, such as incredible strength and impervious skin. His adoptive parents advised him to use his abilities for the benefit of humanity, and he decided to fight crime as a vigilante. To protect his privacy, he changes into a colorful costume and uses the alias "Superman" when fighting crime. Clark Kent resides in the fictional American city of Metropolis, where he works as a journalist for the Daily Planet. Superman's supporting characters include his love interest and fellow journalist Lois Lane, Daily Planet photographer Jimmy Olsen, and editor-in-chief Perry White. His classic foe is Lex Luthor, who is either a mad scientist or a ruthless businessman, depending on the story.""")
for sent in doc.sents:
    print(sent.text)
```

This exercise showcases spaCy's ability to effectively parse and segment a narrative into individual sentences, a fundamental task for further linguistic analysis.

Dependency Parse Trees

Building on our understanding of syntactic structure, *dependency parsing* offers a distinct perspective compared to constituent parsing (you've seen both in the lecture): Dependency parsing emphasizes the relationships between words in a sentence through the use of dependency parse trees. In these trees, connections between tokens are represented by arcs. The label on each arc denotes the type of dependency relationship, while the origin of the arc is known as the **Governor** (referred to as the **head** in spaCy terminology), and the end point of the arc is the **Dependent** (or **child** in spaCy). Each **Dependent/Child** is linked to only one **Governor/Head**, though a single Governor/Head can have multiple Dependents/Children. This model of grammar reveals how words in a sentence relate to and depend on each other, providing deep insights into sentence structure.

For a practical application of this concept, spaCy includes the "displaCy visualizer", a powerful tool for visualizing dependency parse trees. Let's apply displaCy to the sentence *"I prefer a direct flight to Chicago."* This will allow us to see the dependency relationships visually, highlighting the structural roles and interactions between words.

To visualize the dependency parse tree with displaCy, you can use the following code:

```
import spacy

from spacy import displacy


nlp = spacy.load("en_core_web_sm")

doc = nlp("I prefer a direct flight to Chicago.")

displacy.serve(doc, style="dep")
```

Note: The displacy.serve function will start a web server to display the visualization in your browser.

This exercise not only demonstrates spaCy's capability to parse and analyze sentence structure but also shows how dependency parse trees can be used to understand the syntactic relationships within sentences, offering valuable insights for linguistic analysis and NLP applications.

Named Entity Recognition

Named Entity Recognition (NER) is a key technique in NLP for identifying and classifying key information or entities within text into predefined categories, such as persons, locations, organizations, dates, and more. This process can highlight single tokens or groups of tokens as entities, providing a structured understanding of the text's main subjects. NER is invaluable for extracting critical information from texts, offering insights into the main topics discussed or identifying specific details for further analysis.

In spaCy, the *ents* property of a *doc* object holds the detected named entities. Each entity is associated with its text representation, starting and ending character positions, and a label denoting its category. The following code snippet demonstrates how to use spaCy to perform NER and access these entity properties:

```
import spacy

nlp = spacy.load("en_core_web_sm")

doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for ent in doc.ents:

    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

To further explore and visually understand the identified entities, spaCy's displaCy visualizer can be used. This tool allows for a graphical representation of entities, enhancing comprehension and analysis. Initially, visualize all named entities within a given text to confirm their presence. Subsequently, you can refine the visualization to focus on specific categories of interest. This step-by-step approach ensures that you are only viewing the entities relevant to your analysis. Here's how to visualize the entities in a provided text and then restrict the display to specific entity types:

```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
text = "European authorities fined Google a record $5.1 billion on Wednesday for abusing its power
in the mobile phone market and ordered the company to alter its practices"
doc = nlp(text)

# Visualize all named entities
displacy.serve(doc, style="ent")

# To restrict the visualization to specific entity types, modify the options parameter
options = {"ents": ["ORG", "MONEY", "DATE"]}
displacy.serve(doc, style="ent", options=options)
```

Note: As with dependency parsing visualization, `displacy.serve` starts a web server for displaying the visualizations.

This task not only highlights spaCy's robust capabilities in NER but also illustrates how visual tools like displaCy can make complex data more accessible and easier to analyze.

Task #2: Question Classification

Question classification is a pivotal component of developing intelligent systems, particularly in enhancing chatbot functionalities. By categorizing questions into distinct classes, a chatbot can route queries to the appropriate processing pipelines, ensuring that responses are both relevant and accurate. This task becomes crucial in scenarios where a chatbot must understand the intent behind a user's question to provide specific information, such as answering *"who"* questions with a person's details or *"where"* questions with locations.

We've so far looked into classification tasks using tf-idf and cosine similarity to identify the classes the query vector falls into. With an understanding of using spaCy to extract linguistic features, we'll explore creating more sophisticated *feature vectors* for our data. These feature vectors, enriched with linguistic insights, will then guide our classification using a decision tree classifier from the scikit-learn library, offering a blend of linguistic intelligence and machine learning precision.

The methodology of using linguistic features for classification tasks involves extracting and quantifying characteristics of text that are indicative of its category. Features such as parts of speech (POS) tags, entity types, and the lemma of the root word in a sentence can provide valuable insights into the semantics and syntactic structure of text. By analyzing these features, we can train machine learning models to recognize patterns associated with different classes.

In our question classification task, for example, the occurrence of certain POS tags might be more frequent in specific types of questions. "Who" questions may regularly contain personal pronouns, whereas "Where" questions might frequently involve proper nouns denoting places. Similarly, the presence of named entities like organizations, persons, or geographical locations can signal the question's focus. By compiling these features into a feature vector for each question, we create a data representation that encapsulates the linguistic properties relevant to classification. A decision tree classifier, or any other suitable machine learning model, can then learn from these feature vectors to predict the class of unseen questions accurately. This approach combines the nuanced understanding of language provided by NLP tools with the pattern recognition capabilities of machine learning, leading to robust classification systems capable of handling complex natural language queries.

Let's first import the necessary libraries and initialize some variables required for this task:

```
import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.preprocessing import LabelEncoder

from collections import Counter

import spacy


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

Now, let's create a function that accepts text as input and returns a list resembling a feature vector. Your feature vector can contain any combination of features you choose to experiment with. For this task, we are interested in experimenting with the following features:

1. Lemma of the ROOT token in the sentence.

2. Total number of occurrences of the POS tag "WP".
3. Total number of occurrences of the POS tag "WRB".
4. Total number of PERSON entity tags.
5. Total number of GPE entity tags.
6. Total number of ORG entity tags.
7. Length of the text (in characters).

Use the following snippet as a starting point and fill in the code to extract the features listed above:

```
def create_features(text):
    doc = nlp(text)
    # Create a list with all the POS tags in the text
    pos_list = [token.pos_ for token in doc]  # Hint: Use a list comprehension with token.pos_
    pos_count_dict = Counter(pos_list)  # Count the number of POS tags within the POS list

    # Create a list with all the entities within the text
    entity_list = [ent.label_ for ent in doc.ents]  # Hint: Use a list comprehension with ent.label_ for entities in doc.ents
    ent_count_dict = Counter(entity_list)  # Count the number of entity types within the entity list

    # Lemma of the root token in the text (You may need to find the token that is its own head)
    root_lemma = # Your code here. Hint: The root token is its own head. Find this token and get its lemma.

    # Length of the text (number of characters within the text)
    sentence_length = # Your code here. Hint: Use Python's len function on the text.

    return [root_lemma, pos_count_dict['WP'], pos_count_dict["WRB"], ent_count_dict["PERSON"], ent_count_dict["GPE"], ent_count_dict['ORG'], sentence_length]
```

Let's define our corpus:

```
corpus = [
    'Who is Bill Gates',
    'Where is Concordia located',
```

```
    'What is AI',
    'What city is McGill located in',
    'Who is McGill'
]
```

Iterate through each document within the corpus and create a feature matrix. This matrix will represent our text data in a numerical format that can be used for machine learning:

```
features = []
for text in corpus:
    features.append(create_features(text))

feature_mtrx = np.array(features) # Convert the feature matrix to a numpy array.
```

The first column in our data contains a string value, which is not a suitable data structure for the DecisionTreeClassifier we are intending to use. To resolve this, we'll use sklearn's LabelEncoder to convert these string values to a numerical format:

```
le = LabelEncoder()
feature_mtrx[:, 0] = le.fit_transform(feature_mtrx[:, 0])
```

Create a label array to represent the class of each document within the corpus. These classes are what our model will learn to predict:

```
# 0=Who, 1= Where, 2=What
y = np.array([0, 1, 2, 2, 0])
```

Train a DecisionTreeClassifier using our feature matrix. This classifier will learn from the provided features to accurately classify the text based on the labels:

```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(feature_mtrx, y)
```

Now, test your system by trying to predict the class for a new query text. This will demonstrate how well the classifier has learned to categorize questions:

```
q = "Where is Canada"
```

You can also visualize the decision-making process of your trained classifier. This visualization, created with Matplotlib, shows which features are most influential for predictions and how the classifier makes its decisions:

```
import matplotlib.pyplot as plt
```

```
plt.figure()
plot_tree(clf, filled=True, feature_names=["root", "wp_count", "wrb_count", "person", "gpe", "org",
"length"])
plt.show()
```

This task encourages you to apply spaCy functionalities to extract and utilize linguistic features from text. By experimenting with different features, you'll gain insights into how various aspects of the text contribute to the classification results. Explore, experiment, and observe the impact of different linguistic features on the classification outcome. To experiment with more data, you can download this dataset with questions and their types, which look like this:

NUM:count How many people in the world speak French ?

LOC:other Where is the Orinoco ?

DESC:def What is a Canada two-penny black ?

Task #3: Entity Tagging with DBpedia-Spotlight

After exploring the extraction of linguistic features and named entity recognition with spaCy, we now turn our attention to grounding these entities in a knowledge graph using DBpedia-Spotlight. Grounding text to a knowledge graph involves linking identified entities to unique identifiers (URIs) within a global information space, thus disambiguating them and providing a wealth of structured knowledge about the entities. This process is critical for enhancing the semantic understanding of text, enabling applications such as question answering systems, content recommendation, and semantic search.

While spaCy excels at identifying and categorizing entities within text, it operates within the scope of the document itself, without considering the broader context that a knowledge graph provides. For instance, spaCy can recognize "Paris" as a proper noun and likely classify it as a location, but it does not distinguish between "Paris" (France) and "Paris" (Texas) or Paris (Hilton), the person. Each of these interpretations of "Paris" has a distinct representation in a knowledge graph like DBpedia, with unique URIs that connect to detailed information about each entity. This limitation highlights the necessity for an additional layer of processing where entities detected by NLP tools are further resolved and linked to a knowledge base, providing a deeper, more nuanced understanding of the text.

Your next task is to experiment with the *DBpedia-Spotlight* service, an open source tool that specializes in this very task. We have already explored its web-based demo interface during the lecture exercises; now, your task is to understand its API. Spotlight offers a RESTful interface for annotating text with DBpedia URIs, effectively grounding

detected entities in the expansive DBpedia knowledge graph. Here is how you can access the API using a simple HTTP GET request:

curl -X GET "https://api.dbpedia-spotlight.org/en/annotate?text=Concordia%20advanced%20six%20spots%20to%2010th%20place%20among%20Canada%E2%80%99s%20engineering%20schools%20in%20the%20Maclean%E2%80%99s%202018%20Program%20Rankings%2C%20while%20computer%20science%20advanced%20three%20spots%20into%2011th%20position%20this%20year." -H "accept: application/json"

The result is in JSON format:


```
{
  "@text": "Concordia advanced six spots to 10th place among Canada’s engineering schools in the Maclean’s 2018 Program Rankings, while computer science advanced three spots into 11th position this year.",
  "@confidence": "0.5",
  "@support": "0",
  "@types": "",
  "@sparql": "",
  "@policy": "whitelist",
  "Resources": [
    {
      "@URI": "http://dbpedia.org/resource/Concordia_University",
      "@support": "1835",
      "@types": "Wikidata:Q43229,Wikidata:Q3918,Wikidata:Q24229398,DUL:SocialPerson,DUL:Agent,Schema:Organization,Schema:EducationalOrganization,Schema:CollegeOrUniversity,DBpedia:University,DBpedia:Organisation,DBpedia:EducationalInstitution,DBpedia:Agent",
      "@surfaceForm": "Concordia",
      "@offset": "0",
      "@similarityScore": "0.9999782207698248",
      "@percentageOfSecondRank": "1.995676280252352E-5"
    },
    {
```

```
      "@URI": "http://dbpedia.org/resource/Canada",
      "@support": "220739",
      "@types":
"Wikidata:Q6256,Schema:Place,Schema:Country,DBpedia:PopulatedPlace,DBpedia:Place,DBpedia
:Location,DBpedia:Country",
      "@surfaceForm": "Canada",
      "@offset": "49",
      "@similarityScore": "0.9997965603941569",
      "@percentageOfSecondRank": "1.318801347665404E-4"
    },
    {
      "@URI": "http://dbpedia.org/resource/Computer_science",
      "@support": "11027",
      "@types": "",
      "@surfaceForm": "computer science",
      "@offset": "124",
      "@similarityScore": "0.9999981246676802",
      "@percentageOfSecondRank": "6.422167580532429E-7"
    }
  ]
}
```


Make sure you understand the conceptual difference between the *surface form*, which appears in the text you are analyzing, and the link (URI) itself. This is where ambiguities need to be resolved: for example, the surface form *"Hilton"* could be linked to the company or the person *"Paris Hilton"*, depending on the context.

Try writing a simple Python program that accepts a query string, sends a REST request and displays the detected entities.

Note that for frequent requests, you should setup your own Spotlight server. An easy way to install one is to use the provided *Docker* image, which you can find on its Github page.

Finally, note it's also possible to ground entities to Wikidata using tools like spaCy fishing.

Conclusions

In this lab, you've gained practical experience with core Natural Language Processing tasks using spaCy, explored text classification with decision trees, and tackled entity linking with DBpedia-Spotlight. These tasks have equipped you with the skills to preprocess text, extract linguistic features, recognize named entities, and disambiguate them within a knowledge graph context.

Applications like customer support chatbots, content recommendation systems (e.g., Netflix or Amazon), and social media sentiment analysis (e.g., Twitter/X data analysis) rely heavily on NLP techniques similar to those you've practiced. Your ability to link textual entities to unique identifiers in a knowledge graph is critical for systems requiring a nuanced understanding of text content.

In professional contexts, such as job interviews or collaborative projects, detailing your experience with these NLP tasks demonstrates a solid foundation in processing and analyzing text data. For instance, discussing how you used spaCy for NER and DBpedia-Spotlight for entity linking in a project showcases your skills in applying NLP techniques to real-world problems, such as enhancing the accuracy of information retrieval systems or improving the relevancy of recommendations in a content delivery platform.

Looking ahead, the lab sets the stage for further exploration into Text Mining and other advanced NLP topics. The skills developed here serve as a baseline for tackling more complex problems in the field, preparing you for future challenges in NLP and related areas of artificial intelligence and data science.

That's all for this lab!