PROJECT REPORT ON

Project Report: Custom Named Entity Recognition (NER) using spaCy

SUBMITTED TO MIT SCHOOL OF COMPUTING, LONI, PUNE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

BACHELOR OF TECHNOLOGY
(INFORMATION TECHNOLOGY)

Ayush Athare	MITU23BTITD003
Karthik Reddy	MITU22BTIT0040
Mahima Kela	MITU22BTIT0044
Parth Datar	MITU22BTIT0056

Prof. Kalyani Lokhande

Abstract

The project “Custom Named Entity Recognition (NER) using spaCy” demonstrates how Natural Language Processing (NLP) techniques can be applied to detect domain-specific entities in text data. By utilizing the spaCy library and PhraseMatcher, this system identifies custom entities such as organizations, geographical locations, and technologies that are not necessarily covered by the default spaCy model. The project enhances text understanding for specific business or technical domains through efficient and customizable entity detection.

Aim

To design a Python program that performs custom Named Entity Recognition (NER) using the spaCy library, allowing recognition of user-defined entities beyond pre-trained model capabilities.

Methodology

1. **Library Utilization (spaCy):**
    - **The project uses the en_core_web_sm model from spaCy for linguistic processing such as tokenization and part-of-speech tagging.**
    - **It employs PhraseMatcher to define and detect custom entities.**
2. **Custom Entity Creation:**
    - **Entities were grouped into categories such as:**
        - **ORG (Organizations): Tata Consultancy Services, Infosys, Wipro**
        - **GPE (Geopolitical Entities): India, Mumbai, New York**
        - **TECH (Technologies): Python, Java, Machine Learning**
    - **These entities are matched against the text to identify relevant mentions.**
3. **Pattern Matching:**
    - **The PhraseMatcher object compares input text to pre-defined entity patterns based on lowercase string matching (attr="LOWER").**
4. **Span Filtering:**
    - **The filter_spans() function ensures that overlapping entity spans are removed, resulting in clean, non-redundant entity detection.**
5. **Entity Assignment and Output:**
    - **Recognized entities are added to the document’s entity list (doc.ents), and results are displayed as text-label pairs for clarity.**

Results

The implemented system successfully:

- **Detected and labeled custom entities based on predefined patterns.**
- **Identified organizations, geographical places, and technologies in text accurately.**
- **Filtered overlapping entities, ensuring clean output.**
- **Produced clear, human-readable results such as:**

- **Tata Consultancy Services --> ORG**
- **Infosys --> ORG**
- **India --> GPE**
- **Python --> TECH**
- **Java --> TECH**
- **Mumbai --> GPE**

Code:-

```
import spacy

from spacy.matcher import PhraseMatcher

from spacy.tokens import Span

from spacy.util import filter_spans


# Load the English model

nlp = spacy.load("en_core_web_sm")


# Define custom entities grouped by label

custom_entities = {

   "ORG": ["Tata Consultancy Services", "Infosys", "Wipro"],

   "GPE": ["India", "Mumbai", "New York"],

   "TECH": ["Python", "Java", "Machine Learning"]

# Input text

}


# Initialize PhraseMatcher

matcher = PhraseMatcher(nlp.vocab, attr="LOWER")


# ✅Correct way to add patterns

for label, terms in custom_entities.items():

   patterns = [nlp.make_doc(text) for text in terms]

   matcher.add(label, None, *patterns)  # <-- add patterns correctly
```

```
text = "Tata Consultancy Services and Infosys are based in India. They use Python and Java in Mumbai."

doc = nlp(text)


# Apply matcher

matches = matcher(doc)

new_ents = []


for match_id, start, end in matches:

    label = nlp.vocab.strings[match_id]

    span = Span(doc, start, end, label=label)

    new_ents.append(span)

# ✅ Filter  overlapping  spans

filtered_ents = filter_spans(list(doc.ents) + new_ents)


# ✅ Assign non-overlapping entities

doc.ents = filtered_ents


# Output the recognized custom entities

print("=== Custom Entity Matches ===")

for ent in doc.ents:

    print(ent.text, "-->", ent.label_)
```

Conclusion

The Simple Emotional Tone Detector successfully demonstrates the integration of natural language processing and data storage in a journaling context. By analyzing emotional tone using predefined word lists, the project offers an accessible way for users to reflect on their moods and mental well-being. It can serve as a foundational step toward more advanced emotional analytics and mental health applications.

Recommendations

- **Expand Word Lists:** Incorporate a larger and more context-aware emotional vocabulary for improved accuracy.
- **Machine Learning Integration:** Implement sentiment analysis using ML models (e.g., Naive Bayes, BERT) for deeper contextual understanding.
- **Graphical User Interface (GUI):** Develop a desktop or mobile interface for better user interaction.
- **Data Visualization:** Add charts to display mood trends over time for self-analysis.
- **Cloud Synchronization:** Store data securely online to allow users to access their journals from multiple devices.