Course Title: Natural Language Processing

Topic Name: Text Processing and Analysis with NLTK, spaCy, and Transformers

Student's Name: Alikhan Zhamankhan

Date: 16.02.2025

1. Introduction

Natural Language Processing (NLP) is a crucial field within Artificial Intelligence that enables computers to understand, interpret, and manipulate human language. Deep learning techniques, along with robust NLP libraries, have significantly improved language understanding. This document explores the implementation of text processing techniques using Python libraries such as NLTK, spaCy, and Transformers.

Importance of Python Libraries for NLP

- **NLTK**: A comprehensive toolkit for linguistic processing, offering tokenization, lemmatization, and stopword removal.
- **spaCy**: A powerful library for efficient and scalable NLP tasks, including named entity recognition and dependency parsing.
- **Transformers (Hugging Face)**: Utilizes pre-trained deep learning models to extract word embeddings and perform sentiment analysis.

2. Implementation and Code Snippets

2.1 Tokenization, Lemmatization, and Stopword Removal

Task Explanation

Text preprocessing is a crucial step in Natural Language Processing (NLP), involving:

1. **Tokenization** – Splitting text into words or sentences.
2. **Lemmatization** – Reducing words to their base form.
3. **Stopword Removal** – Eliminating common words like "is", "the", "and" that don't add much meaning.

Using NLTK

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
sample_text = "Natural language processing (NLP) is an exciting field. It combines linguistics and computer science."
nltk_tokens = word_tokenize(sample_text)
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
nltk_processed = [lemmatizer.lemmatize(token.lower()) for token in nltk_tokens if token.lower() not in stop_words and token.isalpha()]
nltk_processed
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\aliha\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\aliha\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\aliha\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
['natural',
 'language',
 'processing',
 'nlp',
 'exciting',
 'field',
 'combine',
 'linguistics',
 'computer',
 'science']
```

Figure 1: Tokenization using NLTK

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(sample_text)
spacy_processed = [token.lemma_.lower() for token in doc if not token.is_stop and token.is_alpha]
spacy_processed
```

```
['natural',
 'language',
 'processing',
 'nlp',
 'exciting',
 'field',
 'combine',
 'linguistic',
 'computer',
 'science']
```

Figure 2: Tokenization using spaCy

Comparison of Outputs

1. **Tokenization**
   - NLTK uses word_tokenize(), splitting words based on punctuation.
   - spaCy automatically tokenizes text when you load it into nlp().
2. **Lemmatization**
   - NLTK's WordNetLemmatizer() requires manual specification of the part of speech (default is noun).

- spaCy automatically detects the part of speech and applies more accurate lemmatization.

3. **Stopword Removal**
    - NLTK has a predefined stopword list but requires manual filtering.
    - spaCy's token.is_stop is a direct attribute for checking stopwords.

2.2 Named Entity Recognition (NER) with spaCy

Task Explanation

Named Entity Recognition (NER) is a technique used in NLP to identify and classify proper nouns, such as names of people, organizations, locations, dates, and more.
spaCy provides a **pre-trained NER model** that automatically detects and labels named entities in a given text.

In this task, we will:

1. Load **spaCy's** English language model.
2. Extract **named entities** from a sample text.
3. **Visualize** the named entities using displacy.

```
import spacy
from spacy import displacy

nlp = spacy.load('en_core_web_sm')
ner_text = "Apple is looking at buying U.K. startup for $1 billion."
doc = nlp(ner_text)

# Print Named Entities
for ent in doc.ents:
    print(f" - {ent.text} ({ent.label_})")

displacy.render(doc, style='ent', jupyter=True)
```

```
 - Apple (ORG)
 - U.K. (GPE)
 - $1 billion (MONEY)
```

Apple **ORG** is looking at buying U.K. **GPE** startup for $1 billion **MONEY** .

Figure 3: Named Entity Recognition output using spaCy

Explanation of the Code

1. **Loading spaCy Model**

- We load en_core_web_sm, a small English model with built-in NER capabilities.

2. **Processing Text**
   - The doc object contains tokens, entities, and metadata.
3. **Extracting Named Entities**
   - We loop through doc.ents to print entity names, their labels, and explanations.
4. **Visualization**
   - displacy.render(doc, style="ent", jupyter=True) highlights named entities in a visually appealing wa

2.3 Text Vectorization using Transformers

Task Explanation

Text vectorization is the process of converting text into numerical representations for machine learning and NLP tasks. **Transformers**, such as BERT (bert-base-uncased), generate **contextual embeddings**, meaning that the representation of a word depends on the context in which it appears.

In this exercise, we will:

1. **Load a pre-trained BERT model** from Hugging Face.
2. **Tokenize and encode a sample sentence** using BERT's tokenizer.
3. **Extract word embeddings** from the model's hidden states

```
from transformers import AutoTokenizer, AutoModel
import torch

model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

sample_sentence = "This is a sample sentence for embedding extraction."
inputs = tokenizer(sample_sentence, return_tensors="pt")
outputs = model(**inputs)
embeddings = outputs.last_hidden_state
embeddings
```

```
tensor([[[-0.3291, -0.5040, -0.2125,  ..., -0.6529,  0.0756,  0.8283],
         [-0.6841, -0.8352, -0.4526,  ..., -0.3956,  0.7534,  0.2176],
         [-0.4530, -0.6353,  0.1921,  ..., -0.1519, -0.0703,  0.7846],
         ...,
         [-0.4442, -0.0778, -0.1558,  ..., -0.5368, -0.3660,  0.4046],
         [ 0.5656, -0.0401, -0.7370,  ...,  0.2329, -0.5112, -0.3431],
         [ 0.1879, -0.1038, -0.3582,  ...,  0.6637, -1.0281, -0.1297]]],
       grad_fn=<NativeLayerNormBackward0>)
```

Figure 4: Word Embeddings using BERT

Explanation of the Code

1. **Loading the Model**
   - We use bert-base-uncased, a pre-trained BERT model from Hugging Face.
   - The tokenizer converts text into token IDs, matching BERT's vocabulary.
2. **Tokenization & Encoding**
   - tokenizer(text, return_tensors="pt") converts the text into PyTorch tensors for processing.
3. **Extracting Word Embeddings**
   - model(**inputs) processes the input text and outputs hidden states.
   - outputs.last_hidden_state gives the **contextual embeddings** for each token.
4. **Displaying Embeddings**
   - We print the tokenized words alongside a preview of their vectorized embeddings.

2.4 Sentiment Analysis with Transformers

Task Explanation

Sentiment analysis is the process of determining whether a piece of text expresses **positive, negative, or neutral sentiment**. Traditionally, this was done using **rule-based** or **machine learning** models, but **transformers** like BERT and DistilBERT provide **context-aware sentiment analysis**.

In this exercise, we will:

1. **Use Hugging Face's pipeline module** to perform sentiment analysis.
2. **Compare transformer-based results** with traditional text-processing approaches like Vader (NLTK).

```python
from transformers import pipeline
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk
nltk.download("vader_lexicon")
sentiment_pipeline = pipeline("sentiment-analysis")
sentences = [
    "I absolutely love this product! It works perfectly.",
    "The service was terrible. I'm never coming back!",
    "The experience was okay, not the best, but not the worst either."
]
print("Transformer-based Sentiment Analysis:")
for sentence in sentences:
    result = sentiment_pipeline(sentence)
    print(f"Sentence: {sentence}\nResult: {result}\n")
sia = SentimentIntensityAnalyzer()
print("\nTraditional (Vader) Sentiment Analysis:")
for sentence in sentences:
    scores = sia.polarity_scores(sentence)
    sentiment = "Positive" if scores["compound"] > 0 else "Negative" if scores["compound"] < 0 else "Neutral"
    print(f"Sentence: {sentence}\nScores: {scores}\nVader Sentiment: {sentiment}\n")
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\aliha\AppData\Roaming\nltk_data...
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingf
istilbert-base-uncased-finetuned-sst-2-english).
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cuda:0
Transformer-based Sentiment Analysis:
Sentence: I absolutely love this product! It works perfectly.
Result: [{'label': 'POSITIVE', 'score': 0.9998786449432373}]

Sentence: The service was terrible. I'm never coming back!
Result: [{'label': 'NEGATIVE', 'score': 0.9989019632339478}]

Sentence: The experience was okay, not the best, but not the worst either.
Result: [{'label': 'NEGATIVE', 'score': 0.9642871618270874}]


Traditional (Vader) Sentiment Analysis:
Sentence: I absolutely love this product! It works perfectly.
Scores: {'neg': 0.0, 'neu': 0.358, 'pos': 0.642, 'compound': 0.8746}
Vader Sentiment: Positive

Sentence: The service was terrible. I'm never coming back!
Scores: {'neg': 0.326, 'neu': 0.674, 'pos': 0.0, 'compound': -0.5255}
Vader Sentiment: Negative

Sentence: The experience was okay, not the best, but not the worst either.
Scores: {'neg': 0.128, 'neu': 0.527, 'pos': 0.345, 'compound': 0.5729}
Vader Sentiment: Positive
```

Figure 5: Sentiment Analysis Results

Explanation of the Code

1. **Transformer-based Sentiment Analysis**
   - We use Hugging Face's pipeline("sentiment-analysis"), which loads a pre-trained model (default: **DistilBERT**).
   - The model predicts the sentiment as either **positive or negative**.
2. **Traditional Sentiment Analysis (NLTK Vader)**
   - **Vader** (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon-based method.
   - It assigns a **compound score**:
     - **Positive (> 0)** → Positive Sentiment

- **Negative (< 0)** → Negative Sentiment
- **Neutral (≈ 0)** → Neutral Sentiment

3. Results and Discussion

The processed outputs from different libraries highlight their unique strengths:

- **NLTK** provides foundational text processing features but requires additional steps for accuracy.
- **spaCy** offers robust and efficient NLP operations, including context-aware lemmatization.
- **Transformers-based approaches** leverage deep learning to capture contextual word meanings and perform sentiment analysis with higher accuracy.

4. Conclusion

This document explored various NLP techniques using NLTK, spaCy, and Transformers. The comparison highlights:

- Traditional NLP tools like NLTK and spaCy are effective for tokenization, lemmatization, and named entity recognition.
- Transformers offer state-of-the-art performance in contextual understanding and sentiment analysis.
- Deep learning-based approaches outperform traditional methods in complex text-processing tasks.

5. References

1. Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.
2. spaCy Documentation: https://spacy.io
3. Hugging Face Transformers: https://huggingface.co/transformers/
4. nltk Documentation: https://www.nltk.org/