

Assignment 1 & 2 Report

Course Title: Natural Language Processing

Student Name: Kumarkhanova Ayazhan

Date: 18.02.2025

1 Introduction

Natural language processing (NLP) is a collection of computational techniques for automatic analysis and representation of human languages, motivated by theory. The importance of NLP lies in the vast amount of data available on the World Wide Web, with at least 20 billion pages serving as a valuable resource, provided that meaningful information can be extracted through NLP. Some of the well-known applications of this data include:

- Indexing and searching large texts
- Information retrieval (IR)
- Classification of text into categories
- Information extraction (IE)
- Automatic language translation
- Automatic summarization of texts
- Question-answering (QA)
- Knowledge acquisition
- Text generation and dialogues

[1]

Deep Learning techniques have significantly improved NLP tasks such as tokenization, Named Entity Recognition (NER), and sentiment analysis. Through the use of deep neural networks, these models process large volumes of text data to identify patterns and relationships that traditional algorithms often miss. [2] Python provides several libraries for NLP, including:

- **NLTK**: A library for text processing, including tokenization, stemming, and parsing [3].
- **spaCy**: A fast and efficient NLP library that offers pre-trained models for named entity recognition, tokenization, and dependency parsing [4].
- **Transformers (Hugging Face)**: Provides pre-trained models for various NLP tasks, such as text classification, question answering, and sentiment analysis. [5]

2 Implementation and Code Snippets

2.1 Text Preprocessing with NLTK and spaCy

Tokenization is the process of breaking a text into individual components, such as words or sentences. In this task, we will split text into individual words or tokens. It uses two NLP

libraries: **NLTK** (Natural Language Toolkit) – the 'word_tokenize' function splits the text into tokens based on predefined rules.

```
1
2 import nltk
3 import spacy
4 from nltk.tokenize import word_tokenize
5 from nltk.corpus import stopwords
6 from nltk.stem import WordNetLemmatizer
7
8 # Download resources
9 nltk.download("punkt")
10 nltk.download("stopwords")
11 nltk.download("wordnet")
12
13 # Sample text
14 text = "Natural_language_processing_(NLP)_is_a_machine_\
15 learning_technology_that_gives_computers_the_ability_to_interpret,_\
16 manipulate,_and_comprehend_human_language."
17
18 # Tokenization
19 nltk_tokens = word_tokenize(text)
20
21 # Stopword removal & lemmatization
22 stop_words = set(stopwords.words("english"))
23 lemmatizer = WordNetLemmatizer()
24 nltk_processed = [lemmatizer.lemmatize(word.lower()) for word in
25                   nltk_tokens if word.lower() not in stop_words]
26 print("NLTK_Output:", nltk_processed)
```

spaCy – The 'en_core_web_sm' model processes the text, and tokens are extracted using list comprehension.

```
1
2 import spacy
3
4 nlp = spacy.load("en_core_web_sm")
5 doc = nlp(text)
6
7 # Tokenization, stopwords removal, lemmatization
8 spacy_processed = [token.lemma_.lower() for token in doc if not token.
9                   is_stop]
10 print("spaCy_Output:", spacy_processed)
```

```

•[9]: import nltk
import spacy
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Download resources
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")

# Sample text
text = "Natural language processing (NLP) is a machine \
learning technology that gives computers the ability to interpret, \
manipulate, and comprehend human language."

# Tokenization
nltk_tokens = word_tokenize(text)

# Stopword removal & Lemmatization
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
nltk_processed = [lemmatizer.lemmatize(word.lower()) for word in nltk_tokens if word.lower() not in stop_w
|
print("NLTK Output:", nltk_processed)

NLTK Output: ['natural', 'language', 'processing', '(', 'nlp', ')', 'machine', 'learning', 'technology',
'give', 'computer', 'ability', 'interpret', ',', 'manipulate', ',', 'comprehend', 'human', 'language',
'.']

```

Figure 1: Tokenization using NLTK

```

•[14]: import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp(text)

# Tokenization, stopword removal, Lemmatization
spacy_processed = [token.lemma_.lower() for token in doc if not token.is_stop]

print("SpaCy Output:", spacy_processed)

SpaCy Output: ['natural', 'language', 'processing', '(', 'nlp', ')', 'machine', 'learn', 'technology', 'g
ive', 'computer', 'ability', 'interpret', ',', 'manipulate', ',', 'comprehend', 'human', 'language', '.']

```

Figure 2: Tokenization using spaCy

2.2 Named Entity Recognition (NER)

NER identifies entities like names, locations, and dates in text. Here is an implementation using spaCy:

```

1 from spacy import displacy
2
3 text = "Zhang_Yiming_is_one_of_the_richest_individuals_in_the_world,\
4 with_an_estimated_net_worth_of_US$45.6_billion_as_of_October_2024,\
   according\
5 to_Forbes_and_US$43.1_billion_according_to_Bloomberg_Billionaires_Index."
6
7 doc = nlp(text)
8
9 # Extract Named Entities
10 print("Named_Entities:")
11 for ent in doc.ents:
12     print(f"{ent.text}_-{ent.label_}")
13

```

```

14 # Visualize Entities
15 displacy.render(doc, style="ent", jupyter=True)

```

It extracts and visualizes named entities (e.g., persons, organizations, and monetary values) from a given text. It identifies entities like "Zhang Yiming" (PERSON), "US\$45.6 billion" (MONEY), and "Forbes" (ORG) from the text (Figure 3).

```

from spacy import displacy

text = "Zhang Yiming is one of the richest individuals in the world, \
with an estimated net worth of US$45.6 billion as of October 2024, according\
to Forbes and US$43.1 billion according to Bloomberg Billionaires Index."

doc = nlp(text)

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

# Visualize Entities
displacy.render(doc, style="ent", jupyter=True)

```

Named Entities:
Zhang Yiming - PERSON
US\$45.6 billion - MONEY
October 2024 - DATE
Forbes - ORG
US\$43.1 billion - MONEY
Bloomberg Billionaires Index - ORG

Zhang Yiming PERSON is one of the richest individuals in the world, with an estimated net worth of US\$45.6 billion MONEY as of October 2024 DATE, according to Forbes ORG and US\$43.1 billion MONEY according to Bloomberg Billionaires Index ORG.

Figure 3: Named Entity Recognition output using spaCy

2.3 Text Vectorization using Transformers

BertTokenizer and BertModel are imported from Hugging Face's Transformers library. The tokenizer converts raw text into token IDs (numbers that correspond to words or subwords), and the model converts these IDs into embeddings (dense vector representations).

```

1 import torch
2 from transformers import BertTokenizer, BertModel
3
4 # Load BERT tokenizer
5 tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
6 model = BertModel.from_pretrained("bert-base-uncased")
7
8 text = "Natural_language_processing_is_a_collection_of_computational_
9         techniques\
10         for_automatic_analysis_and_representation_of_human_languages"
11
12 # Tokenization
13 inputs = tokenizer(text, return_tensors="pt")

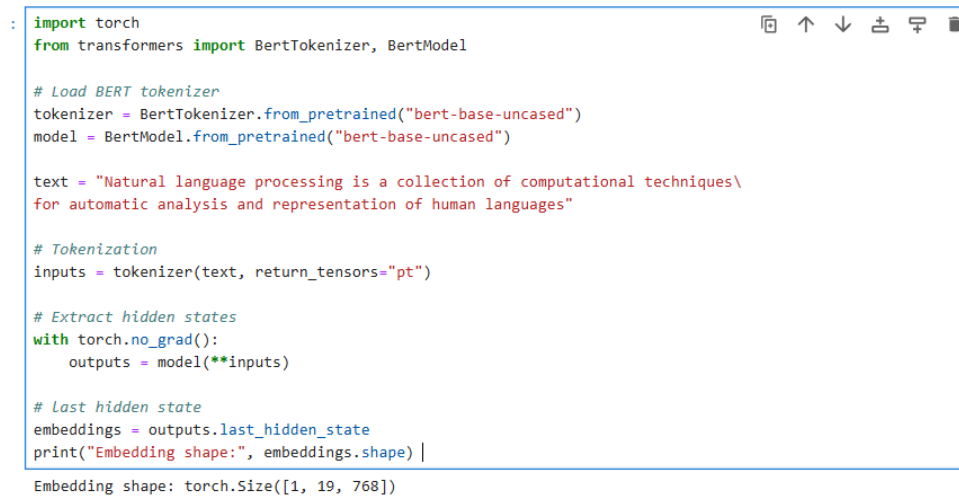
```

```

13
14 # Extract hidden states
15 with torch.no_grad():
16     outputs = model(**inputs)
17
18 # last hidden state
19 embeddings = outputs.last_hidden_state
20 print("Embedding_shape:", embeddings.shape)

```

The text is split into tokens, and the tokenizer creates a dictionary with token IDs. The model processes the tokens, and with `torch.no_grad()` we ensure that no gradients are computed (which saves memory and computation during inference). The output is a tensor that contains the embedding vectors for each token. Its shape is (batch_size, sequence_length, hidden_size).



```

: import torch
  from transformers import BertTokenizer, BertModel

  # Load BERT tokenizer
  tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
  model = BertModel.from_pretrained("bert-base-uncased")

  text = "Natural language processing is a collection of computational techniques\
  for automatic analysis and representation of human languages"

  # Tokenization
  inputs = tokenizer(text, return_tensors="pt")

  # Extract hidden states
  with torch.no_grad():
      outputs = model(**inputs)

  # Last hidden state
  embeddings = outputs.last_hidden_state
  print("Embedding shape:", embeddings.shape) |

```

Embedding shape: torch.Size([1, 19, 768])

Figure 4: Text Vectorization using Transformers

2.4 Sentiment Analysis with Transformers

This task is to perform sentiment analysis using Hugging Face's Transformers pipeline and then compare the results with a traditional lexicon-based approach (using NLTK's VADER).

```

1 import os
2 import tensorflow as tf
3 from transformers import pipeline
4 import torch
5
6 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppress warnings
7 tf.get_logger().setLevel('ERROR') # Suppress logs from TensorFlow
8
9 # Load pipeline
10 sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert/
    distilbert-base-uncased-finetuned-sst-2-english")

```

```

11
12 sentences = [
13     "The_battery_life_was_a_bit_disappointing.",
14     "The_movie_was_okay,_but_not_great.",
15     "I_am_happy_with_the_purchase."
16 ]
17
18 # Analyze sentiment
19 results = sentiment_pipeline(sentences)
20
21 # Results
22 for sentence, result in zip(sentences, results):
23     print(f"Sentence: {sentence} => {result}")
24
25 import nltk
26 nltk.download('vader_lexicon')
27
28 from nltk.sentiment.vader import SentimentIntensityAnalyzer
29
30 # Initialize VADER sentiment analyzer
31 sia = SentimentIntensityAnalyzer()
32
33 # Analyze sentiment with VADER
34 for sentence in sentences:
35     vader_score = sia.polarity_scores(sentence)
36     print(f"VADER->Sentence: {sentence} => {vader_score}")

```

The pipeline tokenizes the text, encodes it, and then processes it through the transformer model. The model outputs sentiment predictions (labels such as POSITIVE or NEGATIVE along with confidence scores).

```

import os
import tensorflow as tf
from transformers import pipeline
import torch

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppress warnings
tf.get_logger().setLevel('ERROR') # Suppress logs from TensorFlow

# Load pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert/distilbert-base-uncased-finetuned-sst-2-english")

sentences = [
    "The battery life was a bit disappointing.",
    "The movie was okay, but not great.",
    "I am happy with the purchase."
]

# Analyze sentiment
results = sentiment_pipeline(sentences)

# Results
for sentence, result in zip(sentences, results):
    print(f"Sentence: {sentence} => {result}")

Device set to use cpu
Sentence: The battery life was a bit disappointing. => {'label': 'NEGATIVE', 'score': 0.9996581077575684}
Sentence: The movie was okay, but not great. => {'label': 'NEGATIVE', 'score': 0.9984203577041626}
Sentence: I am happy with the purchase. => {'label': 'POSITIVE', 'score': 0.9998704195022583}

```

Figure 5: Sentiment analysis using pipeline

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that comes with NLTK. VADER uses a predefined dictionary of sentiment-related words and rules to calculate a sentiment score (showing positive, negative, neutral, and

compound scores).

```
|: import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Analyze sentiment with VADER
for sentence in sentences:
    vader_score = sia.polarity_scores(sentence)
    print(f"VADER -> Sentence: '{sentence}' => {vader_score}")

[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\Asus\anaconda3\nltk_data...
VADER -> Sentence: 'The battery life was a bit disappointing.' => {'neg': 0.39, 'neu': 0.61, 'pos': 0.0, 'compound': -0.4939}
VADER -> Sentence: 'The movie was okay, but not great.' => {'neg': 0.408, 'neu': 0.459, 'pos': 0.133, 'compound': -0.6112}
VADER -> Sentence: 'I am happy with the purchase.' => {'neg': 0.0, 'neu': 0.519, 'pos': 0.481, 'compound': 0.5719}
```

Figure 6: Sentiment analysis using traditional approach

3 Results and Discussion

With the transformer pipeline, each sentence receives a nearly binary decision with extremely high confidence scores (close to 1). The model is fine-tuned on large datasets, which leads to strong confidence even when the sentence might seem mildly negative or positive.

VADER provides a breakdown into negative, neutral, and positive scores along with a compound score. VADER's scores indicate that while the overall sentiment might lean negative or positive, the sentiment is not as extreme. For instance, even though the pipeline classified "The battery life was a bit disappointing." as negative with high confidence, VADER shows a mix of neutral (0.61) and negative (0.39) tones, resulting in a moderately negative compound score.

Both methods agree on the overall sentiment (negative for the first two sentences, positive for the third), but the transformer-based approach tends to deliver more extreme confidence levels compared to VADER's more balanced scoring.

4 Conclusion

This report explored NLP implementations using NLTK, spaCy, and transformers. SpaCy performed better in Named Entity Recognition, while transformers provided more accurate sentiment analysis.

5 References

References

- [1] K. R. Chowdhary. *Natural Language Processing*, pages 603–649. Springer India, New Delhi, 2020.

- [2] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. An introduction to deep learning in natural language processing: Models, techniques, and tools. *Neurocomputing*, 470:443–456, 2022.
- [3] Steven Bird, Edward Loper, and Ewan Klein. Nltk: The natural language toolkit. <http://www.nltk.org>, 2009. Accessed: 2025-02-18.
- [4] spacy: Industrial-strength natural language processing. <https://spacy.io>, 2015. Accessed: 2025-02-18.
- [5] Thomas Wolf et al. Transformers: State-of-the-art natural language processing. <https://huggingface.co/transformers>, 2020. Accessed: 2025-02-18.