An Activity Based Report on Natural Language Processing

Submitted by S. SAI RAHUAL

Abstract

This activity explores the foundational techniques of Information Retrieval (IR) and Natural Language Processing (NLP) using standard corpora such as Brown, Inaugural, Reuters, and UDHR. It involves the practical use of Python's NLTK library to access, manipulate, and analyze large bodies of text through various methods like tokenization, tagging, frequency distribution, and custom corpus creation. Additionally, it demonstrates part-of-speech tagging with rule-based and unigram taggers, as well as word segmentation from unstructured text using dictionary-based scoring approaches. The goal is to understand how corpus-based analysis supports linguistic insights and real-world language processing tasks.

Introduction

Natural Language Processing (NLP) and Information Retrieval (IR) are key areas in artificial intelligence that deal with understanding and processing human language. This activity focuses on hands-on exploration of multiple text corpora using Python and the NLTK toolkit. By studying well-known datasets like the Brown Corpus, Inaugural addresses, Reuters news articles, and the Universal Declaration of Human Rights, learners gain insight into text structures, linguistic features, and tagging methods.

The tasks include examining corpus content with methods such as words(), sents(), and categories(), creating and using custom corpora, analyzing conditional frequency distributions, and performing POS tagging using tagged corpora. Furthermore, learners explore rule-based and statistical tagging methods, build dictionaries to map words to properties, and apply algorithms for segmenting text without spaces by referencing known word lists. This combination of theoretical and practical learning helps illustrate the significance of corpora and tagging in developing efficient NLP applications

Activity-1: Text Classification Game

- **Objective:** Learn supervised learning and text classification.
- **Activity:** Provide students with a set of documents (e.g., movie reviews) labeled as positive or negative. Divide them into groups and have them create a simple classification model using keywords or phrases. They can then test their model on new reviews.

- **Implementation:**

CODE:

```
#!pip install scikit-learn
# Install libraries if needed (run only once)
#!pip install nltk langdetect

# Import libraries import nltk import pandas as pd import
re import string from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords from nltk.stem import
PorterStemmer from langdetect import detect from
sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.model_selection import train_test_split from
sklearn.naive_bayes import MultinomialNB from
sklearn.pipeline import Pipeline

# Download necessary NLTK data
nltk.download('punkt') nltk.download('stopwords') # Load the IMDB Dataset
df = pd.read_csv(r"C:\Users\S. Sai Rahual\Documents\SSR22\B.E AIML\6th
Sem\NLP Mini Project\nlpdts1.csv")   # No extra ".csv" at the end, and use r"" for
Windows path

# Show the first 5 rows to verify df.head()
#  Preprocessing  function
def preprocess_text(text):
# Tokenization
   tokens = word_tokenize(text)

   # Remove punctuation and special characters
   tokens = [word for word in tokens if word.isalnum()]
```

```
    # Language Validation: Ensure text is English
try:        if detect(text) != 'en':            return []
except:
        return []
    # Stop Word Removal
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

    # Stemming     stemmer = PorterStemmer()     stemmed_tokens =
[stemmer.stem(word) for word in filtered_tokens]

    return stemmed_tokens
# Apply preprocessing to all reviews
df['processed_review'] = df['review'].apply(preprocess_text)

# Show the first 5 rows df.head()
# Extract reviews and labels
X = df['review']
y = df['sentiment'].map({'positive': 1, 'negative': 0})  # Convert labels to binary (1 for
positive, 0 for negative)
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Build a pipeline (TF-IDF + Naive Bayes Classifier) model
= Pipeline([
    ('vectorizer', TfidfVectorizer()),
    ('classifier', MultinomialNB())
])
# Train the model
model.fit(X_train, y_train)

# **User Input for Sentiment Prediction** user_review
= input("Enter your review: ") # Predict sentiment
prediction = model.predict([user_review])
# Output result
sentiment_label = "Positive" if prediction[0] == 1 else "Negative" print(f"Sentiment
Analysis Result: {sentiment_label}")
```

- **Result:**
  1. Enter your review: This movie was absolutely amazing! The storyline was ca ptivating, and the performances were top-notch.
     Sentiment Analysis Result: Positive

  2. Enter your review: The film was a waste of time. The plot was boring, and the acting was terrible.
     Sentiment Analysis Result: Negative

  3. Enter your review: An incredible experience! The action scenes were breathtaking, and the soundtrack was fantastic.
     Sentiment Analysis Result: Positive

  4. Enter your review: This film didn't meet my expectations. The script was weak, and it dragged on for too long.
     Sentiment Analysis Result: Negative

Activity-2: Grammar Check and Correction

- **Objective:** Learn about language structure and NLP tools.
- **Activity:** Provide sentences with grammatical errors. Students can use grammar checking tools (like Grammarly or LanguageTool) to identify errors and suggest corrections, discussing why each suggestion is made.

- **Implementation:**

```
CODE:
#!pip install language-tool-python
#!pip install pymupdf
#!pip install pytesseract
import language_tool_python  # NLP-based grammar checking tool
import pandas as pd import fitz import pytesseract
from PIL import Image

# Initialize LanguageTool for English tool =
language_tool_python.LanguageTool('en') def
grammar_check(text):     matches =
tool.check(text)
   corrected_sentences = []

   for match in matches:         correction = match.replacements[0] if
match.replacements else "No suggestion"         corrected_sentences.append({
         "Original Text": text,
         "Error": match.message,
         "Suggestion": correction,
         "Incorrect Word": text[match.offset:match.offset + len(match.context)],
         "Rule": match.ruleId
      })

   return corrected_sentences sentences
= [
   "She go to the market everyday.",  # Incorrect verb form
   "I has a new laptop.",  # Subject-verb agreement error
   "Their is a big problem in the system.",  # Confused words: "Their" vs. "There"
]
# Apply grammar check
```

```
results = [] for sentence
in sentences:
    results.extend(grammar_check(sentence))

# Convert to DataFrame for better visualization df_results
= pd.DataFrame(results)
print(df_results) #
**Ask user for input**
sentence = input("Enter a sentence with grammatical errors: ")

# Apply grammar check
results = grammar_check(sentence)

# Print results if
results:
    for res in results:
        print(f"Error: {res['Error']}")
print(f"Suggestion: {res['Suggestion']}")
print(f"Incorrect Word: {res['Incorrect Word']}")
print(f"Rule ID: {res['Rule']}\n") else:
    print("No grammar issues found!") def
extract_text_from_pdf(pdf_path):     doc
= fitz.open(pdf_path)
    text = ""     for page in doc:         text
+= page.get_text("text") + "\n"     return
text.strip()
pdf_path = r"C:\Users\S. Sai Rahual\Documents\SSR22\B.E AIML\6th Sem\NLP
Mini Project\grammatical_errors_paragraphs.pdf"  # Update with your PDF file path
extracted_text = extract_text_from_pdf(pdf_path)
# Perform grammar check
results = grammar_check(extracted_text) #
Convert to DataFrame for better visualization
df_results = pd.DataFrame(results)
print(df_results)
```

- **Result:**

```
Error: Subject-verb agreement
Suggestion: goes
Incorrect Word: go
Rule ID: ENGLISH_WORD
```

Error: Verb form
Suggestion: buy
Incorrect Word: buys
Rule ID: ENGLISH_WORD

Error: Subject-verb agreement
Suggestion: asks
Incorrect Word: ask
Rule ID: ENGLISH_WORD

Error: Adverb placement
Suggestion: always forgets
Incorrect Word: forget always
Rule ID: ADVERB_WORD_ORDER

Error: Incorrect auxiliary verb
Suggestion: doesn't like
Incorrect Word: don't likes
Rule ID: DO_VBZ

Error: Subject-verb agreement Suggestion:
does
Incorrect Word: do
Rule ID: ENGLISH_WORD

Error: Subject-verb agreement
Suggestion: play
Incorrect Word: plays
Rule ID: ENGLISH_WORD

Error: Verb form
Suggestion: rain
Incorrect Word: raining
Rule ID: PROGRESSIVE_VERBS

Error: Noun form
Suggestion: information
Incorrect Word: informations

Rule ID: NOUN_COUNT

Error: Subject-verb agreement
Suggestion: read
Incorrect Word: reads
Rule ID: ENGLISH_WORD

Error: Verb form
Suggestion: impressed
Incorrect Word: impress
Rule ID: PAST_PARTICIPLE

Error: Subject-verb agreement Suggestion:
were
Incorrect Word: was
Rule ID: WAS_WERE

Error: Verb tense
Suggestion: broke down
Incorrect Word: break down
Rule ID: PAST_TENSE

Error: Subject-verb agreement
Suggestion: need
Incorrect Word: needs
Rule ID: ENGLISH_WORD

Error: Subject-verb agreement
Suggestion: was
Incorrect Word: were
Rule ID: WAS_WERE

Error: Subject-verb agreement Suggestion:
are
Incorrect Word: is
Rule ID: THERE_IS_ARE

Error: Noun form
Suggestion: people

Incorrect Word: peoples
Rule ID: NOUN_COUNT

Error: Subject-verb agreement
Suggestion: think
Incorrect Word: thinks
Rule ID: ENGLISH_WORD