NLP Lab Documents

AKASH T

ADIT CALICUT

Lab -1

1. Installing Necessary Libraries

```
# To install the necessary libraries
!pip install nltk spacy
!python -m spacy download en_core_web_sm
```

- **!pip install nltk spacy:** This command installs the NLTK and SpaCy libraries.
- **!python -m spacy download en_core_web_sm:** This command downloads the small English model for SpaCy, which includes the necessary data for tokenization, parsing, and named entity recognition.

2. Sentence and Word Tokenization using NLTK

```
# Sentence and Word tokenization using NLTK.
from nltk import download
download('punkt')

from nltk.tokenize import word_tokenize, sent_tokenize

text = "Natural Language Processing is fun. Let's learn more about it."

# Word Tokenization
word_tokens = word_tokenize(text)
print("Word Tokens:", word_tokens)

# Sentence Tokenization
sentence_tokens = sent_tokenize(text)
print("Sentence Tokens:", sentence_tokens)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
Word Tokens: ['Natural', 'Language', 'Processing', 'is', 'fun', '.', 'Let', "'s", 'learn', 'more', 'about', 'it', '.']
Sentence Tokens: ['Natural Language Processing is fun.', "Let's learn more about it."]
```

- **download('punkt'):** Downloads the 'punkt' package from NLTK, which is used for tokenization.
- **word_tokenize:** A function to split a string into words.
- **sent_tokenize**: A function to split a string into sentences.

- **word_tokenize(text):** Splits the text into individual words and punctuation.
- **print("Word Tokens:", word_tokens):** Displays the list of word tokens.

3. Tokenization using SpaCy

```
# using Spacy
import spacy

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

# Word Tokenization
word_tokens = [token.text for token in doc]
print("Word Tokens:", word_tokens)

# Sentence Tokenization
sentence_tokens = [sent.text for sent in doc.sents]
print("Sentence Tokens:", sentence_tokens)
```

```
Word Tokens: ['Natural', 'Language', 'Processing', 'is', 'fun', '.', 'Let', "'s", 'learn', 'more', 'about', 'it', '.']
Sentence Tokens: ['Natural Language Processing is fun.', "Let's learn more about it."]
```

- **spacy.load('en_core_web_sm'):** Loads the small English model into the nlp variable.
- **nlp(text):** Processes the text, creating a SpaCy Doc object that contains tokenized words and sentences.
- **[token.text for token in doc]:** Extracts the text of each token in the Doc object.
- **print("Word Tokens:", word_tokens):** Displays the list of word tokens.
- **[sent.text for sent in doc.sents]:** Extracts the text of each sentence in the Doc object.
- **print("Sentence Tokens:", sentence_tokens):** Displays the list of sentence tokens.

4. Stemming using NLTK

```
from nltk.stem import PorterStemmer, SnowballStemmer

words = ["running", "runner", "runs", "happiness", "happily"]

# Porter Stemmer
porter = PorterStemmer()
porter_stems = [porter.stem(word) for word in words]
print("Porter Stemming:", porter_stems)

# Snowball Stemmer
snowball = SnowballStemmer(language='english')
snowball_stems = [snowball.stem(word) for word in words]
print("Snowball Stemming:", snowball_stems)
```

```
Porter Stemming: ['run', 'runner', 'run', 'happi', 'happili']
Snowball Stemming: ['run', 'runner', 'run', 'happi', 'happili']
```

- **porter = PorterStemmer():** Creates an instance of the PorterStemmer.
- **[porter.stem(word) for word in words]:** Applies the Porter stemming algorithm to each word.
- **print("Porter Stemming:", porter_stems):** Displays the list of stemmed words.
- **snowball = SnowballStemmer(language='english'):** Creates an instance of the SnowballStemmer for English.
- **[snowball.stem(word) for word in words]:** Applies the Snowball stemming algorithm to each word.
- **print("Snowball Stemming:", snowball_stems):** Displays the list of stemmed words.

5. Lemmatization using NLTK

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

download('wordnet')
download('omw-1.4')

lemmatizer = WordNetLemmatizer()
words = ["running", "runner", "runs", "happiness", "happily", "better"]

# Lemmatizing with part-of-speech tagging
lemmas = [lemmatizer.lemmatize(word, pos=wordnet.VERB) for word in words]
print("Lemmatized (Verb):", lemmas)

lemmas = [lemmatizer.lemmatize(word, pos=wordnet.ADJ) for word in words]
print("Lemmatized (Adjective):", lemmas)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
Lemmatized (Verb): ['run', 'runner', 'run', 'happiness', 'happily', 'better']
Lemmatized (Adjective): ['running', 'runner', 'runs', 'happiness', 'happily', 'good']
```

- Downloads the WordNet lexical database and additional multilingual data.
- **[lemmatizer.lemmatize(word, pos=wordnet.VERB) for word in words]:** Lemmatizes each word as a verb.
- **print("Lemmatized (Verb):", lemmas):** Displays the list of lemmatized words.
  [lemmatizer.lemmatize(word, pos=wordnet.ADJ) for word in words]: Lemmatizes each word as an adjective.
- **print("Lemmatized (Adjective):", lemmas):** Displays the list of lemmatized words.

6. Lemmatization using SpaCy

```
doc = nlp("running runner runs happiness happily better")

lemmas = [token.lemma_ for token in doc]
print("Lemmatized:", lemmas)
```

```
Lemmatized: ['run', 'runner', 'run', 'happiness', 'happily', 'well']
```

- **[token.lemma_ for token in doc]:** Extracts the lemma of each token in the Doc object.
- **print("Lemmatized:", lemmas):** Displays the list of lemmatized words.

Lab – 2

1. Import CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
```

- Imports the CountVectorizer class from the sklearn.feature_extraction.text module, which is used for converting a collection of text documents to a matrix of token counts.

2. Define Example Documents

```
# Example documents
documents = [
    "Natural Language Processing is fun.",
    "Language models are improving every day."
]
```

- Defines a list of example documents to be used for creating the Bag of Words model.

3. Create CountVectorizer Object

```
# Create the CountVectorizer object
vectorizer = CountVectorizer()
```

- Creates an instance of CountVectorizer, which will be used to transform the text documents into a Bag of Words representation.

4. Fit and Transform the Documents

```
# Fit the model and transform the documents into a BoW representation
X = vectorizer.fit_transform(documents)
```

- Fits the CountVectorizer model to the documents and transforms them into a Bag of Words representation. The result X is a sparse matrix containing the token counts.

5. Display Vocabulary and BoW Representation

```
# Convert the sparse matrix to a dense array and display vocabulary and BoW representation
print("Vocabulary:", vectorizer.vocabulary_)
print("BoW Representation:\n", X.toarray())
```

```
Vocabulary: {'natural': 8, 'language': 6, 'processing': 9, 'is': 5, 'fun': 3, 'models': 7, 'are': 0, 'improving': 4, 'every': 2, 'day': 1}
BoW Representation:
 [[0 0 0 1 0 1 1 0 1 1]
 [1 1 1 0 1 0 1 1 0 0]]
```

- Converts the sparse matrix X to a dense array using **X.toarray().**
- Prints the vocabulary, which is a dictionary mapping each token (word) to its corresponding index in the matrix.
- Prints the Bag of Words representation, showing the token counts for each document.

Lab 3

1. Import TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

- Imports the **TfidfVectorizer** class from the **sklearn.feature_extraction.text** module, which is used for converting a collection of text documents to a matrix of TF-IDF features.

2. Define Example Documents

```
# Example documents
documents = [
    "Natural Language Processing is fun.",
    "Language models are improving every day."
]
```

- Defines a list of example documents to be used for creating the TF-IDF model.

3. Create TfidfVectorizer Object

```
# Create the TfidfVectorizer object
tfidf_vectorizer = TfidfVectorizer()
```

- Creates an instance of **TfidfVectorizer**, which will be used to transform the text documents into a TF-IDF representation.

4. Fit and Transform the Documents

```
# Fit the model and transform the documents into a TF-IDF representation
X_tfidf = tfidf_vectorizer.fit_transform(documents)
```

- Fits the **TfidfVectorizer** model to the documents and transforms them into a TF-IDF representation. The result **X_tfidf** is a sparse matrix containing the TF-IDF features.

5. Display Vocabulary and TF-IDF Representation

```
# Convert the sparse matrix to a dense array and display vocabulary and TF-IDF representation
print("Vocabulary:", tfidf_vectorizer.vocabulary_)
print("TF-IDF Representation:\n", X_tfidf.toarray())
```

```
Vocabulary: {'natural': 8, 'language': 6, 'processing': 9, 'is': 5, 'fun': 3, 'models': 7, 'are': 0, 'improving': 4, 'every': 2, 'day': 1}
TF-IDF Representation:
 [[0.         0.         0.         0.47107781 0.         0.47107781
  0.33517574 0.         0.47107781 0.47107781]
 [0.4261596  0.4261596  0.4261596  0.         0.4261596  0.
  0.30321606 0.4261596  0.         0.        ]]
```

- Converts the sparse matrix X_tfidf to a dense array using X_tfidf.toarray().
- Prints the vocabulary, which is a dictionary mapping each token (word) to its corresponding index in the matrix.
- Prints the TF-IDF representation, showing the TF-IDF values for each document.