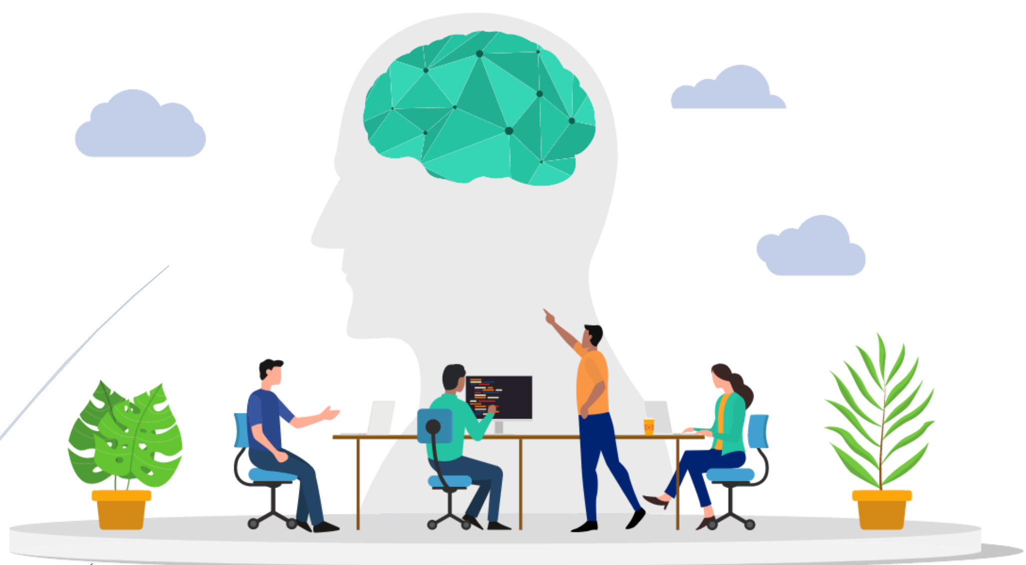
7/26/2024

Assignment



TIDKE ASHOK TATERAO

NSTI CALICUT

Assignment Task-01

AIM:

Lab 1 - NLP Preprocessing Technique

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with nltk spacy

Learning Outcome:

- Understand the emplementation NLP Preprocessing Technique

Procedure:

Step-1: install all necessary libraries

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

Step-2:

```
# 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)
```

Step-3:

```
# 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)
```

Step-4:

```
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']
```

Step-5:

```python
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)
```

Step-6:

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

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

Output:

```
[nltk_data] Downloading package punkt to /root/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."]
```

```
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."]
```

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
Lemmatized (Verb): ['run', 'runner', 'run', 'happiness', 'happily', 'better']
Lemmatized (Adjective): ['running', 'runner', 'runs', 'happiness', 'happily', 'good']
```

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

Assignment Task-02

AIM:

Lab 2 - Python implementation of BoW

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with sklearn, keras

Learning Outcome:

- Understand the implementation of BoW

Procedure:

Step-1:

```
[1]  from sklearn.feature_extraction.text import CountVectorizer
```

Step-2:

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

Step-3:

```
[3]  # Create the CountVectorizer object
     vectorizer = CountVectorizer()
```


```
[3]  # Create the CountVectorizer object
     vectorizer = CountVectorizer()
```


Step-4:

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

Output:

```
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]]
```

Assignment Task-03

AIM:

Lab 3 - Python implementation of TF-IDF

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with sklearn

Learning Outcome:

- Understand the implementation of TF-IDF

Procedure:

Step-1:

```
[1]  from sklearn.feature_extraction.text import TfidfVectorizer
```

Step-2:

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

Step-3:

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

+ Code + Text

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

Step-4:

```
[5] # 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())
```

Output:

```
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.        ]]
```

Assignment Task-04

AIM:

Lab 4 - Python Implementation for Word Embeddings using Word2vec

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with gensim, scipy, nltk

Learning Outcome:

- Understand the Implementation for Word Embeddings using Word2vec

Procedure:

Step-1:

```
# Import necessary Libraries
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk import download
download("punkt")
```

Step-2:

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

    # Tokenize sentences
    tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
    tokenized_sentences
```

Step-3:

```
[ ] # Train the Word2Vec model
    model = Word2Vec(sentences=tokenized_sentences, vector_size=5, window=5, min_count=1, workers=4, sg=0)
    # Here sg=0 means the model will use Continuous bag of words architecture and if sg=1 then it will use Skip-gram Model

    # Get word vectors
    word_vectors = model.wv
    print("Word Vector for 'language':", word_vectors['language'])
```

Output:

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

```
[['natural', 'language', 'processing', 'is', 'fun', '.'],
 ['language', 'models', 'are', 'improving', 'every', 'day', '.']]
```

```
Word Vector for 'language': [-0.14233617  0.12917745  0.17945977 -0.10030856 -0.07526743]
```

Assignment Task-05

AIM:

Lab 5 - Python Implementation for Word Embeddings using GloVe

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with gensim, scipy

Learning Outcome:

Understand the Implementation for Word Embeddings using GloVe

-

Procedure:

Step-1:

```
[1] import gensim.downloader as api

# Load the pre-trained GloVe model with 50 dimensions
glove_vectors_50d = api.load("glove-wiki-gigaword-50")
print("Dimensions of 50d GloVe vector:", len(glove_vectors_50d['language']))

[==================================================] 100.0% 66.0/66.0MB downloaded
Dimensions of 50d GloVe vector: 50
```

Step-2:

```
[3] # Load the pre-trained GloVe model with 100 dimensions
glove_vectors_100d = api.load("glove-wiki-gigaword-100")
print("Dimensions of 100d GloVe vector:", len(glove_vectors_100d['language']))

[==================================================] 100.0% 128.1/128.1MB downloaded
Dimensions of 100d GloVe vector: 100
```

Step-3:

```
# Load the pre-trained GloVe model with 200 dimensions
glove_vectors_200d = api.load("glove-wiki-gigaword-200")
print("Dimensions of 200d GloVe vector:", len(glove_vectors_200d['language']))
```

Step-4:

```
# Load the pre-trained GloVe model with 300 dimensions
glove_vectors_300d = api.load("glove-wiki-gigaword-300")
print("Dimensions of 300d GloVe vector:", len(glove_vectors_300d['language']))
```

Output:

```
[==================================================] 100.0% 66.0/66.0MB downloaded
Dimensions of 50d GloVe vector: 50
```

```
[==================================================] 100.0% 128.1/128.1MB downloaded
Dimensions of 100d GloVe vector: 100
```

```
[==================================================] 100.0% 252.1/252.1MB downloaded
Dimensions of 200d GloVe vector: 200
```

```
[==================================================] 100.0% 376.1/376.1MB downloaded
Dimensions of 300d GloVe vector: 300
```

Assignment Task-06

AIM:

Lab 6 - Python Implementation for Word Embeddings using Fasttext

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with gensim, scipy

Learning Outcome:

Understand the Implementation for Word Embeddings using Fasttext

Procedure:

Step-1:

```
# Import necessary libarries
from gensim.models import FastText
from nltk.tokenize import word_tokenize
from nltk import download
# Download required NLTK data
download('punkt')
```

Step-2:

```
[2]
    # Example sentences
    sentences = [
        "Natural Language Processing is fun.",
        "Language models are improving every day."
    ]

    # Tokenize sentences
    tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

    # Train the FastText model
    model = FastText(sentences=tokenized_sentences, vector_size=5, window=5, min_count=1, workers=4, sg=1)

    # Get word vectors
    word_vectors = model.wv
    print("Word Vector for 'language':", word_vectors['language'])

    # Get vector for an OOV word
    print("Word Vector for 'NLPfun':", word_vectors['nlpfun'])
```

Output:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
Word Vector for 'language': [-0.00461428  0.01921903 -0.00035116 -0.00750383 -0.02619313]
Word Vector for 'NLPfun': [ 0.01152632  0.00589536 -0.01608402 -0.00613909  0.00409522]
```