PRACTICAL TECHNICAL ASSESMENT

[Document subtitle]

ARCHANA G.S

Activity 1

NLP Preprocessing Techniques

comprehensive guide to NLP preprocessing techniques using NLTK and Spacy, including tokenization, stemming, and lemmatization.

- Personal computer/laptop
- Google Collab

Procedure

1. Install Necessary Libraries

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

2. Import Libraries and Download NLTK Resources

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

from nltk.tokenize import word_tokenize, sent_tokenize
```

3. Define the Text

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

4. Tokenization Using NLTK

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

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

5. Tokenization Using Spacy

[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)
```

```
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."]
```

6. Stemming Using NLTK

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

7. 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 /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']
```

```
[6] 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']
```

Activity 2

Python implementation of BoW

Here's a step-by-step procedure documentation for converting a collection of text documents into a Bag of Words (BoW) representation using CountVectorizer from scikit-learn

Requirements

- Personal computer/laptop
- Google Collab

Procedure

1. Import the CountVectorizer class from the sklearn.feature_extraction.text module.

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

2. Prepare a list of example documents. Each document is a string of text.

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

3. Create an instance of CountVectorizer. This object will be used to convert the text documents into a matrix of token counts.

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

4. Fit the model and transform the documents into a BoW representation

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

5. Convert the resulting sparse matrix into a dense array for easier inspection.
   Print the vocabulary, which is a dictionary mapping terms to their indices in the matrix, and print the BoW representation, which shows the token counts for each document.

```
# 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]]
```

Activity 3

Python implementation of BoW

Here's a step-by-step procedure documentation for converting a collection of text documents into a Bag of Words (BoW) representation using CountVectorizer from scikit-learn

Requirements

- Personal computer/laptop
- Google Collab

Procedure

1. Start by importing the TfidfVectorizer from the sklearn.feature_extraction.text module.

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

2. Create a list of example documents to be transformed using TF-IDF.

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

3. Initialize the TfidfVectorizer object.

```
[3] # Create the TfidfVectorizer object
    tfidf_vectorizer = TfidfVectorizer()
```

4. Fit the TfidfVectorizer to the documents and transform them into a TF-IDF representation.

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

5. Display the 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.        ]]
```

This shows the mapping of each word to a unique index in the TF-IDF matrix.