hamza-waheed

November 29, 2024

0.1 Practice Task 1: Text Preprocessing

• Objective: Clean and preprocess a provided text. • Instructions: 1. Import necessary libraries (nltk, re). 2. Define a new text paragraph. 3. Remove punctuation and convert the text to lowercase. 4. Tokenize the cleaned text into words. 5. Print the cleaned and tokenized words.

[4]:
```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from collections import Counter
import matplotlib.pyplot as plt
import re
import nltk
```

[5]:
```
text = """Here's the code to generate a short description and
save it as a PDF file using the rmarkdown package. Make sure you have the␣
 ↪rmarkdown package installed. If not, you can install it with install.
 ↪packages("rmarkdown").
"""
```

[ ]:
```
sentences = sent_tokenize(text)
print("Sentences:\n", sentences)
```

```
Sentences:
 ['Here's the code to generate a short description and \nsave it as a PDF file
using the rmarkdown package.', 'Make sure you have the rmarkdown package
installed.', 'If not, you can install it with install.packages("rmarkdown").']
```

[7]:
```
cleaned_text = re.sub(r'[^\w\s]', '', text).lower()
cleaned_text
```

[7]:
```
'heres the code to generate a short description and \nsave it as a pdf file
using the rmarkdown package make sure you have the rmarkdown package installed
if not you can install it with installpackagesrmarkdown\n'
```

[8]:
```
tokenized_words = nltk.word_tokenize(cleaned_text)
tokenized_words
```

[8]:
```
['heres',
 'the',
 'code',
 'to',
 'generate',
 'a',
 'short',
 'description',
 'and',
 'save',
 'it',
 'as',
 'a',
 'pdf',
 'file',
 'using',
 'the',
 'rmarkdown',
 'package',
 'make',
 'sure',
 'you',
 'have',
 'the',
 'rmarkdown',
 'package',
 'installed',
 'if',
 'not',
 'you',
 'can',
 'install',
 'it',
 'with',
 'installpackagesrmarkdown']
```

0.2 Practice Task 2: Custom Stop Words Removal

• Objective: Implement a custom list of stop words. • Instructions: 1. Use the same text from Task 1. 2. Create a custom list of stop words (e.g., [“the”, “is”, “and”, “to”]). 3. Tokenize the text and remove both NLTK’s stop words and your custom stop words. 4. Print the remaining words after stop words removal.

[9]:
```
vocab = {}
stop_words = set(stopwords.words('english'))
for sentence in sentences:
    words = word_tokenize(sentence)
```

```
    for word in words:
        word = word.lower()
        if word not in stop_words and word.isalpha(): # Check if the word is␣
↪alphabetic
            if word not in vocab:
                vocab[word] = 0
            vocab[word] += 1
print("\n After removing stop words:\n", vocab)
```

```
 After removing stop words:
 {'code': 1, 'generate': 1, 'short': 1, 'description': 1, 'save': 1, 'pdf': 1,
'file': 1, 'using': 1, 'rmarkdown': 3, 'package': 2, 'make': 1, 'sure': 1,
'installed': 1, 'install': 1}
```

0.3 Practice Task 3: Word Frequency Analysis

• Objective: Perform frequency analysis on a new text. • Instructions: 1. Define a new text paragraph. 2. Tokenize the text into words and create a frequency distribution. 3. Print the frequency of each word. 4. Identify and print the three least common words in the text.

[ ]:
```
para = """In today's digital age, having a website is crucial for any business␣
↪or individual looking to establish an online presence. A well-designed␣
↪website can help you attract and engage your target audience, increase␣
↪conversions, and ultimately drive business growth. However, designing a␣
↪website can be a daunting task, especially for those who are new to web␣
↪design. In this post, we'll take you through a step-by-step guide on how to␣
↪design a website that meets your goals and resonates with your target␣
↪audience."""
```

[11]:
```
sentences = sent_tokenize(para)
print("Sentences:\n", sentences)
```

```
Sentences:
 ["In today's digital age, having a website is crucial for any business or
individual looking to establish an online presence.", 'A well-designed website
can help you attract and engage your target audience, increase conversions, and
ultimately drive business growth.', 'However, designing a website can be a
daunting task, especially for those who are new to web design.', "In this post,
we'll take you through a step-by-step guide on how to design a website that
meets your goals and resonates with your target audience."]
```

0.4 Practice Task 4: Visualization of Word Frequencies

- Objective: Visualize the frequency distribution of words.
- Instructions:1. Use the frequency distribution created in Task 3.

2. Create a bar chart showing the top 10 most common words.
3. Label the axes and give the chart a title

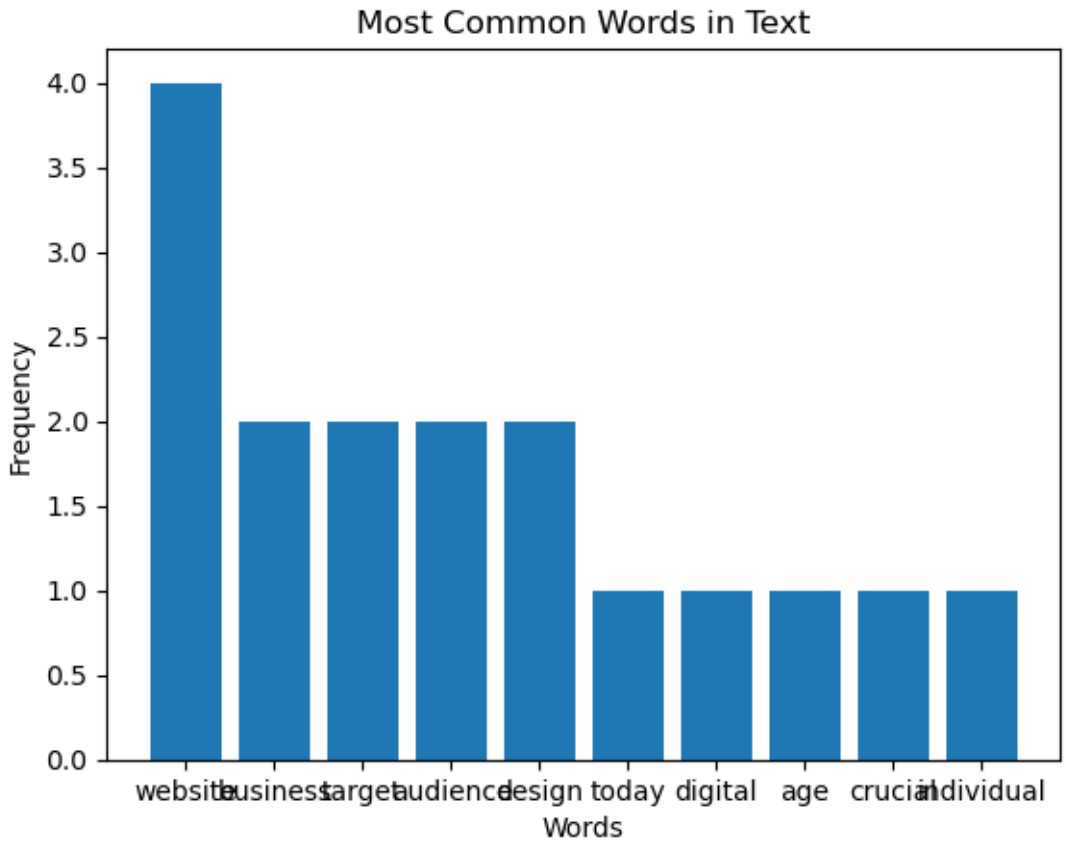
[12]:
```
vocab = {}
for sentence in sentences:
    words = word_tokenize(sentence)
    for word in words:
        word = word.lower()
        if word not in stop_words and word.isalpha(): # Check if the word is␣
 ↪alphabetic
            if word not in vocab:
                vocab[word] = 0
            vocab[word] += 1
print("\nVocabulary with Frequencies:\n", vocab)
```

```
Vocabulary with Frequencies:
 {'today': 1, 'digital': 1, 'age': 1, 'website': 4, 'crucial': 1, 'business': 2,
'individual': 1, 'looking': 1, 'establish': 1, 'online': 1, 'presence': 1,
'help': 1, 'attract': 1, 'engage': 1, 'target': 2, 'audience': 2, 'increase': 1,
'conversions': 1, 'ultimately': 1, 'drive': 1, 'growth': 1, 'however': 1,
'designing': 1, 'daunting': 1, 'task': 1, 'especially': 1, 'new': 1, 'web': 1,
'design': 2, 'post': 1, 'take': 1, 'guide': 1, 'meets': 1, 'goals': 1,
'resonates': 1}
```

[13]:
```
vocab_size = 3
common_vocab = Counter(vocab).most_common(vocab_size)
print("\nMost Common Words:\n", common_vocab)
```

```
Most Common Words:
 [('website', 4), ('business', 2), ('target', 2)]
```

[14]:
```
labels, values = zip(*Counter(vocab).most_common(10))
plt.bar(labels, values)
plt.title('Most Common Words in Text')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.show()
```



0.5 Practice Task 5: Sentence Length Analysis

- Objective: Analyze the length of sentences in a text.
- Instructions:

1. Define a new text paragraph.
2. Tokenize the text into sentences.
3. Calculate the length (number of words) of each sentence.
4. Print the lengths of all sentences and their average length.

[15]:
```
new_text = """n modern software development, choosing the right architectural␣
↪pattern is critical to building scalable, maintainable, and testable␣
↪applications. Among the most popular patterns are MVC␣
↪(Model-View-Controller), MVP (Model-View-Presenter), and MVVM␣
↪(Model-View-ViewModel). These patterns ensure that code is well-organized by␣
↪separating responsibilities across different layers."""
```

[16]:
```
sentences = sent_tokenize(new_text)
sentences
```

[16]:
```
['n modern software development, choosing the right architectural pattern is
critical to building scalable, maintainable, and testable applications.',
 'Among the most popular patterns are MVC (Model-View-Controller), MVP (Model-
View-Presenter), and MVVM (Model-View-ViewModel).',
 'These patterns ensure that code is well-organized by separating
responsibilities across different layers.']
```

[17]:
```
words = [word.lower() for sentence in sentences for word in␣
 ↪word_tokenize(sentence) if
word.isalpha()]
print("\nAll Words:\n", words)
```

```
All Words:
 ['n', 'modern', 'software', 'development', 'choosing', 'the', 'right',
'architectural', 'pattern', 'is', 'critical', 'to', 'building', 'scalable',
'maintainable', 'and', 'testable', 'applications', 'among', 'the', 'most',
'popular', 'patterns', 'are', 'mvc', 'mvp', 'and', 'mvvm', 'these', 'patterns',
'ensure', 'that', 'code', 'is', 'by', 'separating', 'responsibilities',
'across', 'different', 'layers']
```

[22]:
```
count = 0
for sentence in sentences:
    count += 1

print("Total Length of sentences is: " + str(count))
```

```
Total Length of sentences is: 3
```

[23]:
```
sentence_lengths = [len(nltk.word_tokenize(re.sub(r'[^\w\s]', '', sentence)))␣
 ↪for sentence in sentences]
sentence_lengths
```

[23]:
```
[18, 13, 13]
```

[24]:
```
average_length = sum(sentence_lengths) / len(sentence_lengths)
average_length
```

[24]:
```
14.666666666666666
```

0.6 Practice Task 6: Part-of-Speech Tagging

• Objective: Perform part-of-speech (POS) tagging on a text. • Instructions: 1. Define a new text paragraph. 2. Tokenize the text into sentences. 3. For each sentence, tokenize it into words. 4. Use NLTK's pos_tag function to tag each word with its part of speech. 5. Print the original sentences along with their corresponding POS tags.

[25]:
```
last_text = """While MVC and MVP are well-known in web development, MVVM has␣
 ↪gained prominence thanks to Google's recommendation for Android applications.
 ↪ However, these patterns are versatile and can also be adapted to backend␣
 ↪and frontend web development."""
```

[26]:
```
words = [word.lower() for sentence in sentences for word in␣
 ↪word_tokenize(sentence) if
word.isalpha()]
print("\nAll Words:\n", words)
```

```
All Words:
 ['n', 'modern', 'software', 'development', 'choosing', 'the', 'right',
'architectural', 'pattern', 'is', 'critical', 'to', 'building', 'scalable',
'maintainable', 'and', 'testable', 'applications', 'among', 'the', 'most',
'popular', 'patterns', 'are', 'mvc', 'mvp', 'and', 'mvvm', 'these', 'patterns',
'ensure', 'that', 'code', 'is', 'by', 'separating', 'responsibilities',
'across', 'different', 'layers']
```

[27]:
```
pos_tags = nltk.pos_tag(words)

print(pos_tags)
```

```
[('n', 'JJ'), ('modern', 'JJ'), ('software', 'NN'), ('development', 'NN'),
('choosing', 'VBG'), ('the', 'DT'), ('right', 'JJ'), ('architectural', 'JJ'),
('pattern', 'NN'), ('is', 'VBZ'), ('critical', 'JJ'), ('to', 'TO'), ('building',
'VBG'), ('scalable', 'JJ'), ('maintainable', 'JJ'), ('and', 'CC'), ('testable',
'JJ'), ('applications', 'NNS'), ('among', 'IN'), ('the', 'DT'), ('most', 'RBS'),
('popular', 'JJ'), ('patterns', 'NNS'), ('are', 'VBP'), ('mvc', 'JJ'), ('mvp',
'NN'), ('and', 'CC'), ('mvvm', 'VB'), ('these', 'DT'), ('patterns', 'NNS'),
('ensure', 'VB'), ('that', 'IN'), ('code', 'NN'), ('is', 'VBZ'), ('by', 'IN'),
('separating', 'VBG'), ('responsibilities', 'NNS'), ('across', 'IN'),
('different', 'JJ'), ('layers', 'NNS')]
```

[28]:
```
def tag_and_print_sentences(sentences):
    for sentence in sentences:
        words = nltk.word_tokenize(re.sub(r'[^\w\s]', '', sentence))
        pos_tags = nltk.pos_tag(words)
        print(f"Original Sentence: {sentence}")
        print(f"POS Tags: {pos_tags}\n")

# Tag and print POS tags for each sentence
tag_and_print_sentences(sentences)
```

```
Original Sentence: n modern software development, choosing the right
architectural pattern is critical to building scalable, maintainable, and
testable applications.
POS Tags: [('n', 'JJ'), ('modern', 'JJ'), ('software', 'NN'), ('development',
```

```
'NN'), ('choosing', 'VBG'), ('the', 'DT'), ('right', 'JJ'), ('architectural',
'JJ'), ('pattern', 'NN'), ('is', 'VBZ'), ('critical', 'JJ'), ('to', 'TO'),
('building', 'VBG'), ('scalable', 'JJ'), ('maintainable', 'JJ'), ('and', 'CC'),
('testable', 'JJ'), ('applications', 'NNS')]

Original Sentence: Among the most popular patterns are MVC (Model-View-
Controller), MVP (Model-View-Presenter), and MVVM (Model-View-ViewModel).
POS Tags: [('Among', 'IN'), ('the', 'DT'), ('most', 'RBS'), ('popular', 'JJ'),
('patterns', 'NNS'), ('are', 'VBP'), ('MVC', 'NNP'), ('ModelViewController',
'NNP'), ('MVP', 'NNP'), ('ModelViewPresenter', 'NNP'), ('and', 'CC'), ('MVVM',
'NNP'), ('ModelViewViewModel', 'NNP')]

Original Sentence: These patterns ensure that code is well-organized by
separating responsibilities across different layers.
POS Tags: [('These', 'DT'), ('patterns', 'NNS'), ('ensure', 'VB'), ('that',
'IN'), ('code', 'NN'), ('is', 'VBZ'), ('wellorganized', 'VBN'), ('by', 'IN'),
('separating', 'VBG'), ('responsibilities', 'NNS'), ('across', 'IN'),
('different', 'JJ'), ('layers', 'NNS')]
```