Information Retrieval Exp 1

Mark Lopes

BE Comps A

9913

Q1 Write a python code to remove punctuations, URLs and stop words.

Q1 Write a python code to remove punctuations, URLs and stop words.

Generate + Code + Markdown

```
import re
import string
from nltk.corpus import stopwords
from nltk.tokenize import TreebankWordTokenizer
import nltk

# Force download again to fix possible issues
nltk.download('stopwords', force=True)
text = "Visit https://example.com! This is a sample text, with punctuations and stop words."
# Remove URLs
text = re.sub(r"http\S+|www\S+|https\S+", '', text)
# Remove punctuations
text = text.translate(str.maketrans('', '', string.punctuation))
# Lowercase
text = text.lower()
# Tokenization
tokenizer = TreebankWordTokenizer()
tokens = tokenizer.tokenize(text)
# Remove stopwords
filtered_tokens = [word for word in tokens if word not in stopwords.words('english')]
print("Cleaned Text:", filtered_tokens)
```

[1]

```
[nltk_data] Downloading package stopwords to C:\Users\Mark
[nltk_data]     Lopes\AppData\Roaming\nltk_data...
Cleaned Text: ['visit', 'sample', 'text', 'punctuations', 'stop', 'words']
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Q2 Write a python code perform stemmer operation using Porterstemmer , Snowballstemmer, Lancasterstemmer, RegExpStemmer.

Q 2 Write a python code perform stemmer operation using Porterstemmer ,Snowballstemmer, Lancasterstemmer, RegExpStemmer

```
from nltk.stem import PorterStemmer, SnowballStemmer, LancasterStemmer, RegexpStemmer

# Sample text corpus (can be replaced with any other)
words = ["caresses", "flies", "dies", "mules", "denied", "died",
         "agreed", "owned", "humbled", "sized", "meeting", "sings", "happiness"]

# Initialize stemmers
porter = PorterStemmer()
snowball = SnowballStemmer("english")
lancaster = LancasterStemmer()
regexp = RegexpStemmer('ing$|s$|e$', min=4)  # Removes 'ing', 's', 'e' endings if word has at least 4 chars

# Display results
print(f"{'Word':<12}{'Porter':<12}{'Snowball':<12}{'Lancaster':<12}{'Regexp':<12}")
print("-" * 60)
for word in words:
    print(f"{word:<12}{porter.stem(word):<12}{snowball.stem(word):<12}{lancaster.stem(word):<12}{regexp.stem(word):<12}")
```

[2]

```
Word        Porter      Snowball    Lancaster   Regexp
------------------------------------------------------------
caresses    caress      caress      caress      caresse
flies       fli         fli         fli         flie
dies        die         die         die         die
mules       mule        mule        mul         mule
denied      deni        deni        deny        denied
died        die         die         died        died
agreed      agre        agre        agree       agreed
owned       own         own         own         owned
humbled     humbl       humbl       humbl       humbled
sized       size        size        siz         sized
meeting     meet        meet        meet        meet
sings       sing        sing        sing        sing
happiness   happi       happi       happy       happines
```

Q 3 Write a python code to demonstrate the comparative study of all 4 stemmers for a given text corpus.

Q 3 Write a python code to demonstrate the comparative study of all 4 stemmers for a given text corpus.

```
from nltk.stem import PorterStemmer, SnowballStemmer, LancasterStemmer, RegexpStemmer

# Sample corpus for comparison
words = [
    "caresses", "flies", "dies", "mules", "denied", "died",
    "agreed", "owned", "humbled", "sized", "meeting", "sings",
    "happiness", "relational", "conditional", "rational", "valency", "digitizer"
]

# Initialize stemmers
porter = PorterStemmer()
snowball = SnowballStemmer("english")
lancaster = LancasterStemmer()
regexp = RegexpStemmer('ing$|s$|e$', min=4)

# Create table of results
print(f"{'Word':<15} {'Porter':<15} {'Snowball':<15} {'Lancaster':<15} {'Regexp':<15}")
print("-" * 75)

for word in words:
    p = porter.stem(word)
    s = snowball.stem(word)
    l = lancaster.stem(word)
    r = regexp.stem(word)
    print(f"{word:<15} {p:<15} {s:<15} {l:<15} {r:<15}")
```

[3]

```
Word            Porter          Snowball        Lancaster       Regexp
------------------------------------------------------------------------
caresses        caress          caress          caress          caresse
flies           fli             fli             fli             flie
dies            die             die             die             die
mules           mule            mule            mul             mule
denied          deni            deni            deny            denied
died            die             die             died            died
agreed          agre            agre            agree           agreed
owned           own             own             own             owned
humbled         humbl           humbl           humbl           humbled
sized           size            size            siz             sized
meeting         meet            meet            meet            meet
sings           sing            sing            sing            sing
happiness       happi           happi           happy           happines
relational      relat           relat           rel             relational
conditional     condit          condit          condit          conditional
rational        ration          ration          rat             rational
valency         valenc          valenc          val             valency
digitizer       digit           digit           digit           digitizer
```

- 🟢 Porter and Snowball stemmers often produce similar results and are generally conservative in cutting.
- 🟡 Lancaster is more aggressive, often shortening words too much (e.g., "happiness" → "happy" vs "happi").
- 🔴 RegexpStemmer is rule-based and limited; it only removes predefined suffixes like 'ing', 's', or 'e'.
- ✅ For real-world NLP tasks, SnowballStemmer is preferred due to balance between accuracy and simplicity.

Q 4 Write a python code perform lemmatization using NLTK library.

Q 4 Write a python code perform lemmatization using NLTK library.

```python
from nltk.stem import WordNetLemmatizer
import nltk

# Download required resources
nltk.download('wordnet')
nltk.download('omw-1.4')  # WordNet data
nltk.download('punkt')    # For tokenization if needed
lemmatizer = WordNetLemmatizer()
# Sample words (can include any corpus you like)
words = ["walking", "is", "main", "animals", "foxes", "are", "jumping", "sleeping"]
# Lemmatize as verbs (for better accuracy in many cases)
lemmatized = [lemmatizer.lemmatize(word, pos='v') for word in words]
print("Original Words : ", words)
print("Lemmatized Words (NLTK) :", lemmatized)
```

[4]

```
[nltk_data] Downloading package wordnet to C:\Users\Mark
[nltk_data]     Lopes\AppData\Roaming\nltk_data...
[nltk_data] Downloading package omw-1.4 to C:\Users\Mark
[nltk_data]     Lopes\AppData\Roaming\nltk_data...
[nltk_data] Downloading package punkt to C:\Users\Mark
[nltk_data]     Lopes\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Original Words :  ['walking', 'is', 'main', 'animals', 'foxes', 'are', 'jumping', 'sleeping']
Lemmatized Words (NLTK) : ['walk', 'be', 'main', 'animals', 'fox', 'be', 'jump', 'sleep']
```

Q 5 Write a python code perform lemmatization using Spacy library.

```
Q 5 Write a python code perform lemmatization using Spacy library.

import spacy

# Load English language model
nlp = spacy.load("en_core_web_sm")
# Sample text/corpus
text = "walking is main animals foxes are jumping sleeping"
# Process text
doc = nlp(text)
# Extract and print lemmatized tokens
lemmatized = [token.lemma_ for token in doc]
print("Original Words : ", [token.text for token in doc])
print("Lemmatized Words (spaCy) :", lemmatized)

[11]

Original Words :  ['walking', 'is', 'main', 'animals', 'foxes', 'are', 'jumping', 'sleeping']
Lemmatized Words (spaCy) : ['walk', 'be', 'main', 'animal', 'fox', 'be', 'jump', 'sleep']
```

Q 6 Compare the results lemmatization with Spacy and NLTK for the corpus given below walking, is , main, animals , foxes, are, jumping , sleeping. Write your conclusion for the results obtained

Q6 Compare the results lemmatization with Spacy and NLTK for the corpus given below walking, is , main, animals , foxes, are, jumping , sleeping. Write your conclusion for the results obtained.

--- Comparison of NLTK and spaCy Lemmatization --- Original Corpus: ['walking', 'is', 'main', 'animals', 'foxes', 'are', 'jumping', 'sleeping'] Lemmatized (NLTK) : ['walk', 'be', 'main', 'animals', 'fox', 'be', 'jump', 'sleep'] Lemmatized (spaCy) : ['walk', 'be', 'main', 'animal', 'fox', 'be', 'jump', 'sleep']

--- Conclusion --- For the given corpus:

- NLTK with `pos='v'` correctly lemmatized 'walking', 'jumping', and 'sleeping' to their base verb forms.
- NLTK, when not explicitly given the part-of-speech, defaults to noun lemmatization, which wouldn't change 'walking', 'jumping', 'sleeping'. However, the previous code already set pos='v'.
- spaCy processes the words within the context of a sentence (even if it's just space-separated words). It correctly identifies the base forms for 'walking', 'jumping', and 'sleeping' as verbs.
- Both NLTK and spaCy correctly lemmatized the plural nouns 'animals' and 'foxes' to their singular forms.
- Both NLTK and spaCy handled the auxiliary verbs 'is' and 'are' correctly, reducing them to '-PRON-' in spaCy's case (which represents pronouns or pro-adjectives that act as a coreference), and 'be' in NLTK's verb lemmatization.
- For 'main', both return 'main' as it's already in its base form.

Overall, both NLTK and spaCy performed well on this specific corpus for lemmatization. spaCy tends to be more context-aware due to its dependency parsing and POS tagging, which can lead to more accurate lemmatization in complex sentences, although for this simple list of words, the results are largely comparable when NLTK's POS is specified.

Colab paid products - Cancel contracts here

Post Lab Questions:

1. What all python Libraries are available to work with Indian languages like Hindi, Punjabi, Marathi..etc?

Python offers several powerful libraries and tools that support Natural Language Processing (NLP) tasks in Indian languages such as Hindi, Marathi, Tamil, Telugu, Punjabi, and others. These libraries help in tokenization, transliteration, lemmatization, translation, and more.

1. Indic NLP Library

- Developed by IIT Bombay.
- Supports tasks like tokenization, normalisation, transliteration, syllabification, and script conversion.
- Supports over 11 Indian languages.
- GitHub: https://github.com/anoopkunchukuttan/indic_nlp_library

2. iNLTK (Indian Natural Language Toolkit)

- Built on top of fastai and ULMFiT.
- Supports text preprocessing, language modeling, and sentence embeddings in Indian languages.
- Languages supported: Hindi, Marathi, Bengali, Kannada, etc.
- Installation: pip install inltk

3. AI4Bharat Models

- Offers pre-trained BERT-based models for many Indian languages.
- Includes multilingual ASR, translation, and OCR systems.