This chapter will include the following topics: • Tokenization of text • Normalization of text • Substituting and correcting tokens • Applying Zipf's law to text • Applying similarity measures using the Edit Distance Algorithm • Applying similarity measures using Jaccard's Coefficient • Applying similarity measures using Smith Waterman

In [107]:
```
import warnings
warnings.filterwarnings('ignore')
```

Tokenization

Tokenization may be defined as the process of splitting the text into smaller parts called tokens, and is considered a crucial step in NLP.

Tokenization of text into sentences

In [2]:
```
import nltk
text=" Welcome readers. I hope you find it interesting. Please do reply."
```

In [3]:
```
from nltk.tokenize import sent_tokenize
sent_tokenize(text)    # Uses PunktSentenceTokenizer for tokenizing.
```

Out[3]:
```
[' Welcome readers.', 'I hope you find it interesting.', 'Please do reply.']
```

In [4]:
```
import nltk
tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
```

In [5]:
```
text="Welcome readers. I hope you find it interesting. Please do reply."
tokenizer.tokenize(text)   # We can load PunktSentenceTokenizer and use the to
kenize() function.
```

Out[5]:
```
['Welcome readers.', 'I hope you find it interesting.', 'Please do reply.']
```

Tokenization of text in other languages

In [6]:
```
import nltk
french_tokenizer=nltk.data.load('tokenizers/punkt/french.pickle')
```

In [10]:
```
french_tokenizer.tokenize("""Deux agressions en quelques jours,voilà ce qui a
 motivé hier matin le débrayage collège francobritanniquedeLevallois-Perret. D
euxagressions en quelques jours,voilà ce qui a motivé hier matin le débrayage
 Levallois. Léquipepédagogique de ce collège de 750 élèves avait déjà été choq
uéepar l'"agression, janvier , d'un professeur d'histoire. L'équipepédagogique
de ce collège de 750 élèves avait déjà été choquée parl'agression, mercredi ,
 d'un professeur d'histoire""")
```

Out[10]:
```
['Deux agressions en quelques jours,voilà ce qui a motivé hier matin le débra
yage collège francobritanniquedeLevallois-Perret.',
 'Deuxagressions en quelques jours,voilà ce qui a motivé hier matin le débray
age Levallois.',
 'Léquipepédagogique de ce collège de 750 élèves avait déjà été choquéepar l
\'"agression, janvier , d\'un professeur d\'histoire.',
 "L'équipepédagogique de ce collège de 750 élèves avait déjà été choquée par
l'agression, mercredi , d'un professeur d'histoire"]
```

Tokenization of sentences into words

In [11]:
```
text=nltk.word_tokenize("PierreVinken , 59 years old , will joinas a nonexecut
ive director on Nov. 29")
print(text) # word_tokenize function uses an instance of NLTK known as Treeban
kWordTokenizer.
```

```
['PierreVinken', ',', '59', 'years', 'old', ',', 'will', 'joinas', 'a', 'none
xecutive', 'director', 'on', 'Nov.', '29']
```

In [12]:
```
type(text)
```

Out[12]:
```
list
```

In [13]:
```
from nltk import word_tokenize
r=input("Please write a text")
print("The length of text is",len(word_tokenize(r)),"words")
```

```
Please write a texthello everyone how are you??
The length of text is 7 words
```

Tokenization using TreebankWordTokenizer

In [18]:
```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
result = tokenizer.tokenize("Don't hesitate to ask questions")
result
```

Out[18]:
```
['Do', "n't", 'hesitate', 'to', 'ask', 'questions']
```

In [17]:
```
type(result)
```

Out[17]:
```
list
```

In [19]:
```
from nltk.tokenize import WordPunctTokenizer
tokenizer=WordPunctTokenizer()
tokenizer.tokenize(" Don't hesitate to ask questions")
```

Out[19]: ['Don', "'", 't', 'hesitate', 'to', 'ask', 'questions']

Types of Tokenizers

Tokenizer

PunktWord Tokenizer Regex Tokenizer TreeBank Tokenizer

- WordPunct Tokenizer
- Whitespace Tokenizer

Tokenization using regular expressions

In [60]:
```
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import BlanklineTokenizer
from nltk.tokenize import WhitespaceTokenizer
from nltk.tokenize import LineTokenizer
from nltk.tokenize import SpaceTokenizer
from nltk.tokenize.util import spans_to_relative
from nltk.tokenize.util import string_span_tokenize
```

In [32]:
```
tokenizer=RegexpTokenizer('\s+',gaps=True)
tokenizer.tokenize("Don't hesitate to ask questions")
```

Out[32]: ["Don't", 'hesitate', 'to', 'ask', 'questions']

In [34]:
```
sent=" She secured 90.56 % in class X . She is a meritorious student"
capt = RegexpTokenizer('[A-Z]\w+')
capt.tokenize(sent)
```

Out[34]: ['She', 'She']

In [36]:
```
sent=" She secured 90.56 % in class X . She is a meritorious student"
BlanklineTokenizer().tokenize(sent)
```

Out[36]: [' She secured 90.56 % in class X . She is a meritorious student']

In [39]:
```
WhitespaceTokenizer().tokenize(sent)
```

Out[39]:
```
['She',
 'secured',
 '90.56',
 '%',
 'in',
 'class',
 'X',
 '.',
 'She',
 'is',
 'a',
 'meritorious',
 'student']
```

In [40]:
```
sent.split()
```

Out[40]:
```
['She',
 'secured',
 '90.56',
 '%',
 'in',
 'class',
 'X',
 '.',
 'She',
 'is',
 'a',
 'meritorious',
 'student']
```

In [42]:
```
sent.split(' ')
```

Out[42]:
```
['',
 'She',
 'secured',
 '90.56',
 '%',
 'in',
 'class',
 'X',
 '.',
 'She',
 'is',
 'a',
 'meritorious',
 'student']
```

In [43]:
```
sent=" She secured 90.56 % in class X \n. She is a meritorious student\n"
sent.split('\n')
```

Out[43]:
```
[' She secured 90.56 % in class X ', '. She is a meritorious student', '']
```

In [50]:
```
LineTokenizer(blanklines='discard').tokenize(sent)
```

Out[50]:
```
[' She secured 90.56 % in class X ', '. She is a meritorious student']
```

In [49]:
```
LineTokenizer(blanklines='keep').tokenize(sent)
```

Out[49]:
```
[' She secured 90.56 % in class X ', '. She is a meritorious student']
```

In [52]:
```
SpaceTokenizer().tokenize(sent)
```

Out[52]:
```
['',
 'She',
 'secured',
 '90.56',
 '%',
 'in',
 'class',
 'X',
 '\n.',
 'She',
 'is',
 'a',
 'meritorious',
 'student\n']
```

In [53]:
```
list(WhitespaceTokenizer().span_tokenize(sent))
```

Out[53]:
```
[(1, 4),
 (5, 12),
 (13, 18),
 (19, 20),
 (21, 23),
 (24, 29),
 (30, 31),
 (33, 34),
 (35, 38),
 (39, 41),
 (42, 43),
 (44, 55),
 (56, 63)]
```

In [61]:
```
list(string_span_tokenize(sent, " "))
```

Out[61]:
```
[(1, 4),
 (5, 12),
 (13, 18),
 (19, 20),
 (21, 23),
 (24, 29),
 (30, 31),
 (32, 34),
 (35, 38),
 (39, 41),
 (42, 43),
 (44, 55),
 (56, 64)]
```

Normalization

In order to carry out processing on natural language text, we need to perform normalization that mainly involves eliminating punctuation, converting the entire text into lowercase or uppercase, converting numbers into words, expanding abbreviations, canonicalization of text, and so on.

Eliminating punctuation

In [67]:
```
from nltk.tokenize import word_tokenize
import re
import string
```

In [64]:
```
text=[" It is a pleasant evening.","Guests, who came from US arrived at the ve
nue","Food was tasty."]
tokenized_docs=[word_tokenize(doc) for doc in text]
print(tokenized_docs)
```

```
[['It', 'is', 'a', 'pleasant', 'evening', '.'], ['Guests', ',', 'who', 'cam
e', 'from', 'US', 'arrived', 'at', 'the', 'venue'], ['Food', 'was', 'tasty',
'.']]
```

In [66]:
```
type(tokenized_docs)
```

Out[66]:
```
list
```

In [75]:
```
x=re.compile('[%s]' % re.escape(string.punctuation))
tokenized_docs_no_punctuation = []
for review in tokenized_docs:
    new_review = []
    for token in review:
        new_token = x.sub(u'', token)
        if not new_token == u'':
            new_review.append(new_token)
    tokenized_docs_no_punctuation.append(new_review)
```

In [76]:
```
print(tokenized_docs_no_punctuation)
```

```
[['It', 'is', 'a', 'pleasant', 'evening'], ['Guests', 'who', 'came', 'from',
'US', 'arrived', 'at', 'the', 'venue'], ['Food', 'was', 'tasty']]
```

Conversion into lowercase and uppercase

In [77]:
```
text='HARdWork IS KEy to SUCCESS'
print(text.lower())
```

```
hardwork is key to success
```

In [78]:
```
print(text.upper())
```

```
HARDWORK IS KEY TO SUCCESS
```

Dealing with stop words

Stop words are words that need to be filtered out during the task of information retrieval or other natural language tasks, as these words do not contribute much to the overall meaning of the sentence. There are many search engines that work by deleting stop words so as to reduce the search space. Elimination of stopwords is considered one of the normalization tasks that is crucial in NLP.

In [79]:
```
from nltk.corpus import stopwords
stops=set(stopwords.words('english'))
words=["Don't", 'hesitate','to','ask','questions']
[word for word in words if word not in stops]
```

Out[79]:
```
["Don't", 'hesitate', 'ask', 'questions']
```

In [82]:
```
len(stops)
```

Out[82]:
```
179
```

In [85]:
```
len(stopwords.fileids())
stopwords.fileids()
```

Out[85]:
```
['arabic',
 'azerbaijani',
 'danish',
 'dutch',
 'english',
 'finnish',
 'french',
 'german',
 'greek',
 'hungarian',
 'indonesian',
 'italian',
 'kazakh',
 'nepali',
 'norwegian',
 'portuguese',
 'romanian',
 'russian',
 'spanish',
 'swedish',
 'turkish']
```

Substituting and correcting tokens

In this section, we will discuss the replacement of tokens with other tokens. We will also see how we can correct the spelling of tokens by replacing incorrectly spelled tokens with correctly spelled tokens.

Replacing words using regular expressions

Previously, we faced problems while performing tokenization for contractions. Using text replacement, we can replace contractions with their expanded versions. For example, doesn't can be replaced by does not.

In [90]:
```
# save the below code as a python file  - replacers.py
import re
replacement_patterns = [
(r'won\'t', 'will not'),
(r'can\'t', 'cannot'),
(r'i\'m', 'i am'),
(r'ain\'t', 'is not'),
(r'(\w+)\'ll', '\g<1> will'),
(r'(\w+)n\'t', '\g<1> not'),
(r'(\w+)\'ve', '\g<1> have'),
(r'(\w+)\'s', '\g<1> is'),
(r'(\w+)\'re', '\g<1> are'),
(r'(\w+)\'d', '\g<1> would')
]
class RegexpReplacer(object):
        def __init__(self, patterns=replacement_patterns):
            self.patterns = [(re.compile(regex), repl) for (regex, repl)in pat
terns]
        def replace(self, text):
            s = text
            for (pattern, repl) in self.patterns:
                (s, count) = re.subn(pattern, repl, s)
            return s
```

from the above saved pythobn file import your desired class - RegexpReplacer

from replacers import RegexpReplacer

replacer= RegexpReplacer()

replacer.replace("Don't hesitate to ask questions")

Performing substitution before tokenization

>>> import nltk >>> from nltk.tokenize import word_tokenize >>> from replacers import RegexpReplacer >>> replacer=RegexpReplacer() >>> word_tokenize("Don't hesitate to ask questions") ['Do', "n't", 'hesitate', 'to', 'ask', 'questions'] >>> word_tokenize(replacer.replace("Don't hesitate to ask questions")) ['Do', 'not', 'hesitate', 'to', 'ask', 'questions']

Dealing with repeating characters

Sometimes, people write words involving repeating characters that cause grammatical errors. For instance consider a sentence, I like it lotttttt. Here, lotttttt refers to lot. So now, we'll eliminate these repeating characters using the backreference approach, in which a character refers to the previous characters in a group in a regular expression. This is also considered one of the normalization tasks.

#append the below class to the already created replacers.py class RepeatReplacer(object): def __init__(self): self.repeat_regexp = re.compile(r'(\w*)(\w)\2(\w*)') self.repl = r'\1\2\3' def replace(self, word): repl_word =

self.repeat_regexp.sub(self.repl, word) if repl_word != word: return self.replace(repl_word) else: return repl_word

The problem with RepeatReplacer is that it will convert happy to hapy, which is inappropriate. To avoid this problem, we can embed wordnet along with it.

import re from nltk.corpus import wordnet class RepeatReplacer(object): def __init__(self): self.repeat_regexp = re.compile(r'(\w*)(\w)\2(\w*)') self.repl = r'\1\2\3' def replace(self, word): if wordnet.synsets(word): return word repl_word = self.repeat_regexp.sub(self.repl, word) if repl_word != word: return self.replace(repl_word) else: return repl_word

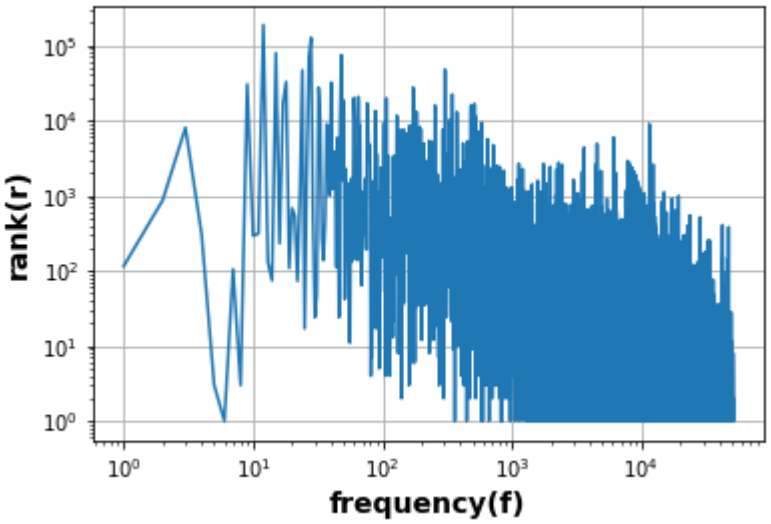
Replacing a word with its synonym

Now we will see how we can substitute a given word by its synonym. To the already existing replacers.py, we can add a class called WordReplacer that provides mapping between a word and its synonym: class WordReplacer(object): def __init__(self, word_map): self.word_map = word_map def replace(self, word): return self.word_map.get(word, word)

Applying Zipf's law to text

Zipf's law states that the frequency of a token in a text is directly proportional to its rank or position in the sorted list. This law describes how tokens are distributed in languages: some tokens occur very frequently, some occur with intermediate frequency, and some tokens rarely occur.

In [108]:
```
import nltk
from nltk.corpus import gutenberg
from nltk.probability import FreqDist
import matplotlib
import matplotlib.pyplot as plt
matplotlib.use('TkAgg')
fd = FreqDist()
for text in gutenberg.fileids():
    for word in gutenberg.words(text):
        fd[word] += 1
ranks = []
freqs = []
for rank, word in enumerate(fd):
    ranks.append(rank+1)
    freqs.append(fd[word])
plt.loglog(ranks, freqs)
plt.xlabel('frequency(f)', fontsize=14, fontweight='bold')
plt.ylabel('rank(r)', fontsize=14, fontweight='bold')
plt.grid(True)
plt.show()
```



Similarity measures

There are many similarity measures that can be used for performing NLP tasks. The nltk.metrics package in NLTK is used to provide various evaluation or similarity measures, which is conducive to perform various NLP tasks. In order to test the performance of taggers, chunkers, and so on, in NLP, the standard scores retrieved from information retrieval can be used

In [111]:
```
from __future__ import print_function
from nltk.metrics import *
training='PERSON OTHER PERSON OTHER OTHER ORGANIZATION'.split()
testing='PERSON OTHER OTHER OTHER OTHER OTHER'.split()
print(accuracy(training,testing))
```

```
0.6666666666666666
```

In [113]:
```
train_set=set(training)
test_set=set(testing)
precision(train_set,test_set)
```

Out[113]:
```
1.0
```

In [115]:
```
print(recall(train_set,test_set))
```

```
0.6666666666666666
```

In [117]:
```
print(f_measure(train_set,test_set))
```

```
0.8
```

Applying similarity measures using Ethe edit distance algorithm

Edit distance or the Levenshtein edit distance between two strings is used to compute the number of characters that can be inserted, substituted, or deleted in order to make two strings equal.

In [118]:
```
from nltk.metrics import *
edit_distance("relate","relation")
```

Out[118]:
```
3
```

In [119]:
```
edit_distance("suggestion","calculation")
```

Out[119]:
```
7
```

In [120]:
```
edit_distance("sunil","sunilkumar")
```

Out[120]:
```
5
```

Here, when we calculate the edit distance between relate and relation, three operations (one substitution and two insertions) are performed. While calculating the edit distance between suggestion and calculation, seven operations (six substitutions and one insertion) are performed.

Applying similarity measures using Jaccard's Coefficient

Jaccard's coefficient, or Tanimoto coefficient, may be defined as a measure of the overlap of two sets, X and Y.

In [129]:
```
X=set([50,10])
Y=set([20,10])
print(jaccard_distance(X,Y))
```

```
0.6666666666666666
```

Applying similarity measures using the Smith Waterman distance

The Smith Waterman distance is similar to edit distance. This similarity metric was developed in order to detect the optical alignments between related protein sequences and DNA. It consists of costs to be assigned to and functions for alphabet mapping to cost values (substitution); cost is also assigned to gap G (insertion or deletion)

Summary

In this chapter, you have learned various operations that can be performed on a text that is a collection of strings. You have understood the concept of tokenization, substitution, and normalization, and applied various similarity measures to strings using NLTK. We have also discussed Zipf's law, which may be applicable to some of the existing documents. In the next chapter, we'll discuss various language modeling techniques and different NLP tasks.