Unit 2.

Text Preprocessing

2.1. Tokenization

2.2. Stop Words

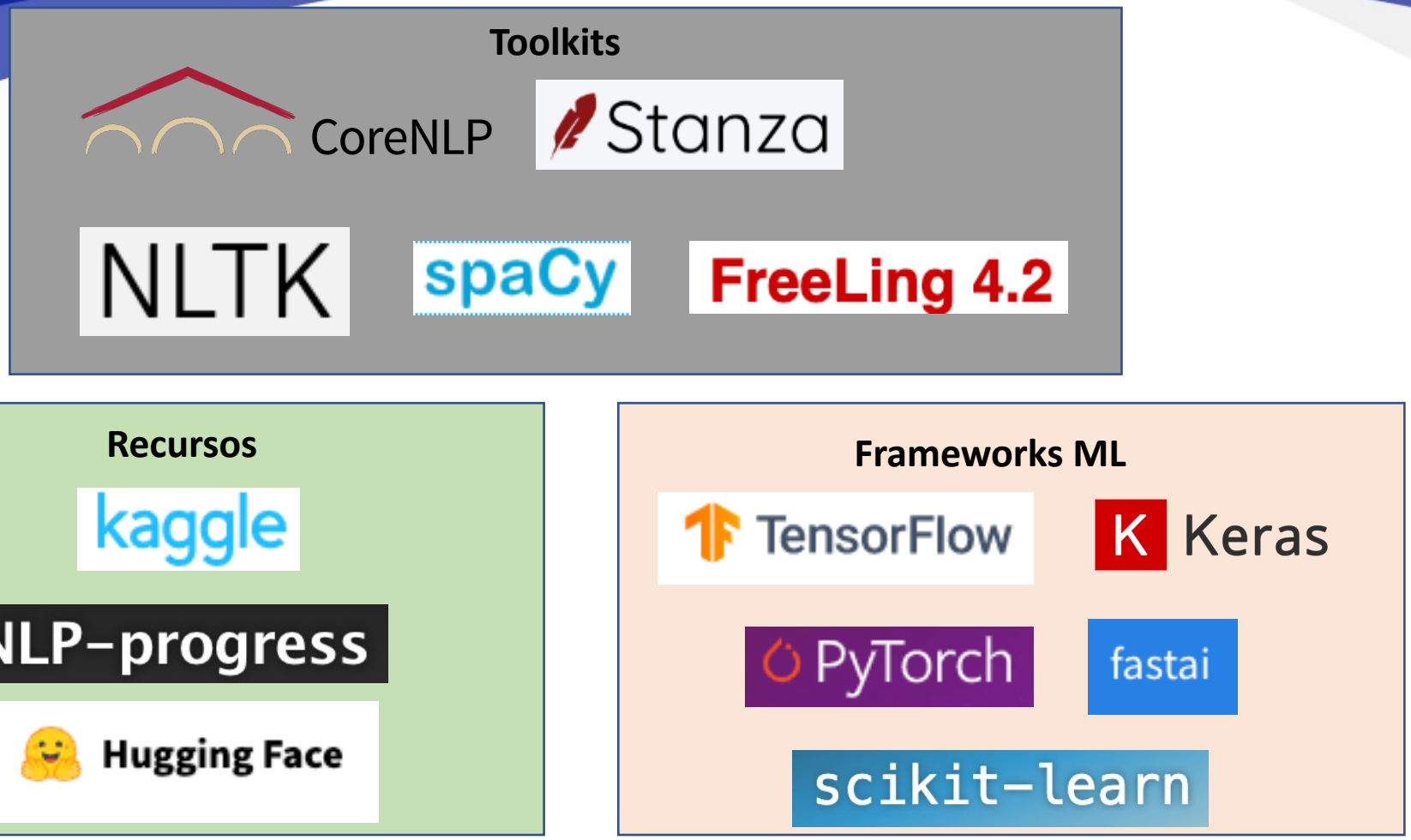
2.3. Lemmatization and Stemming

2.4. POS Tagging

2.5. Integer Encoding

2.6. Padding

2.7. One-Hot Encoding


Toolkits
CoreNLP
Stanza
NLTK
spaCy
FreeLing 4.2
Recursos
kaggle
NLP-progress
Hugging Face
Frameworks ML
TensorFlow
Keras
PyTorch
fastai
scikit-learn

Pre-processing

Pre-processing:

- Tokenization: break down the raw text into words or sentences.
- Cleaning:
  - Remove punctuation marks, excess spaces, special characters, etc. (*)
  - Also remove excessively short or infrequent words, etc.
- Normalization:
  - Conversion to the lowercase.
  - Remove the stop words.
  - Stemming and Lemmatization.
  - Expansion of abbreviations. (*)
- (*) Regular expressions can be useful.

Tokenization

Often the first step in the data pre-processing.

- a) Tokenization into sentences:
  - Usually the sentence boundary is marked by a period, exclamation mark, question mark, etc.
  - But, sometimes a period does not mark the end of a sentence: abbreviations, for example.

    Ex "He got his *M.D.* from the university of Wisconsin."
  - A good sentence tokenizer should recognize these exceptions.
- b) Tokenization into words:
  - Usually splitting by white space or punctuation mark works.
  - In some cases, there is ambiguity: apostrophe, for example.

    Ex "*Don't* lose your hope, *everything's* possible."

    ⇒ How do we tokenize "*don't*" and "*everything's*"? ⇒ It depends on the tokenizer.

EjemplosTextProcessing.ipynb

In [ ]:
```
# NLTK offers tools necessary for tokenization of English corpus.
```

In [1]:
```
from nltk.tokenize import word_tokenize
print(word_tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a past
```

```
['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'as',
'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

In [ ]:
```
# word_tokenize separated Don't into Do and n't, but Jone's into Jone and 's
```

In [2]:
```
from nltk.tokenize import WordPunctTokenizer
print(WordPunctTokenizer().tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery
```

```
['Don', "'", 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone', "'", 's', 'Orphanage',
'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

In [ ]:
```
# Keras, also has a tokenization tool, supports text_to_word_sequence
```

In [3]:
```
from tensorflow.keras.preprocessing.text import text_to_word_sequence
print(text_to_word_sequence("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes fo
```

```
["don't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'mr', "jone's", 'orphanage', 'is', 'as', 'cheery', 'a
s', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

```
# regexp_tokenizer tokenizes a text given a regular expression
```

```
import nltk
nltk.download('punkt')
from nltk import regexp_tokenize
from nltk import word_tokenize

pattern = r'''(?x)     # set flag to allow verbose regexps
    (?:[A-Z]\.)+       # abbreviations, e.g. U.S.A.
    | \w+(?:-\w+)*       # words with optional internal hyphens
    | \$?\d+(?:\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
    | \.\.\.             # ellipsis
    | [][.,;"'?():-_`]   # these are separate tokens; includes ], [
'''

```

```
text = 'That U.S.A. poster-print costs $12.40...'
```

```
print(text.split())
print(word_tokenize(text))
print(regexp_tokenize(text, pattern))
```

```
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40...']
['That', 'U.S.A.', 'poster-print', 'costs', '$', '12.40', '...']
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

 # regexp_tokenizer tokenize a text given a regular expression

```
import nltk
nltk.download('punkt')
from nltk import regexp_tokenize
from nltk import word_tokenize

pattern = r'''(?x)     # set flag to allow verbose regexps
    (?:[A-Z]\.)+       # abbreviations, e.g. U.S.A.
    | \w+(?:-\w+)*       # words with optional internal hyphens
    | \$?\d+(?:\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
    | \.\.\.             # ellipsis
    | [][.,;"'?():-_`]   # these are separate tokens; includes ], [
'''

```

```
text2 = "Don't be fool by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry sho
```

```
print(text2.split())
print(word_tokenize(text2))
print(regexp_tokenize(text2, pattern))
```

```
["Don't", 'be', 'fool', 'by', 'the', 'dark', 'sounding', 'name,', 'Mr.', "Jone's", 'Orphanage', 'is', 'as', 'cheery'
['Do', "n't", 'be', 'fool', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'a
['Don', "'", 't', 'be', 'fool', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone', "'", 's', 'Orphanag
```

For non-normative text, such as tweets, we need to clean and normalize the text and decide which elements are of interest

```
"@pepe: yooooooo 😂😻😍SOY el #mejor y tú no :-))))!!!! lol!. 🙂😍
http://www.micasa.com, tucasa.org/comedor/sillon.html"
```

- **Mentions**: @Pepe
- **Repetitions**: yooooooo
- **Emojis:** 😂😻😍
- **Uppercase**: SOY
- **Hashtags**: #mejor
- **Urls**: http://www.micasa.com
- **Other**: :-))))

03_tokenizar_tweets.ipynb

`TweetTokenizer` is a tweet-specific tokenizer that can be configured to:

- *preserve_case*: convert all text to lowercase or keep uppercase
- *reduce_len*: remove repeating characters
- *strip_handles*: remove mentions

```
tweet = "@pepe: yooooooooo 😂😻😍SOY el #mejor y tú no :-))))!!!! lol!. 🙂😍
http://www.micasa.com, tucasa.org/comedor/sillon.html"
```

```
import nltk.tokenize

tknzr = TweetTokenizer(preserve_case=False, reduce_len=True,
strip_handles=True)

tknzr.tokenize(tweet)
```

```
[':', 'yooo', '😂', '😻', '😍', 'soy', 'el', '#mejor', 'y', 'tú', 'no',
':-)', ')', ')', '!', '!', '!', 'lol', '!', '.', '🙂', '😍',
'http://www.micasa.com', ',', 'tucasa.org/comedor/sillon.html']
```

Other useful tools:

`twitter-text-parser`: *to parse* a text and to extract elements such as urls or emojis

https://twitter-text-python.readthedocs.io/en/latest/#

`emoji`: to convert emoji to text (`emojize`) or text to emoji (`demojize`)

https://github.com/carpedm20/emoji/

Unit 2.

Text Preprocessing

- 2.1. Tokenization
- 2.2. Stop Words
- 2.3. Lemmatization and Stemming
- 2.4. POS Tagging
- 2.5. Integer Encoding
- 2.6. Padding
- 2.7. One-Hot Encoding

Remove *stop words*

En un lugar de La_Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.

|tokens|=37, |vocab|=29

_ _ lugar _ La_Mancha , _ cuyo nombre _ quiero acordarme, _ _ _ tiempo _ vivía _ hidalgo _ _ _ lanza _ astillero , adarga antigua , rocín flaco _ galgo corredor.

|tokens|=22, |vocab|=19

Lemmatization

En un lugar de La_Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.

En un lugar de La_Mancha, de cuyo nombre no querer acordarme, no ha mucho tiempo que vivir un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.

Stemming

En un lugar de La_Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.

En un lug de La_Manch , de cuy nombr no quier acord, no ha much tiemp que viv un hidalg de los de lanz en astiller, adarg antigu, rocín flac y galg corredor.

Stop Words

Stop words and keywords:

- Stop words are commonly used words that do not contain semantic information:

  Ex Definite and indefinite articles: “the”, “a”, “an”.

  Ex Prepositions: “on”, “with”, “into”, “upon”, etc.

- Stop words are removed during the “data normalization” process.
- Keywords are selected among the available words after removing the stop words.
  - Often the most frequent words can be selected as keywords.
  - Sometimes we should also take into account the purpose and context.

2.2. Stop Words

Stop words and keywords:

- In the NLTK library, there are 179 English stop words:

i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, him, his, himself, she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, that'll, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn, mightn't, mustn, mustn't, needn, needn't, shan, shan't, shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't

2.2. Stop Words

51_ex_0506.ipynb

Stop words and keywords:

In [9]:
```
from nltk.corpus import stopwords
stopwords.words('english')[:10]
```

Out[9]:
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

In [11]:
```
    # NLTK defines words such as 'i', 'me', 'my' as stop words
```

In [10]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for w in word_tokens:
    if w not in stop_words:
        result.append(w)

print(word_tokens)
print(result)
```

```
['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
```

Unit 2.

Text Preprocessing

- 2.1. Tokenization
- 2.2. Stop Words
- 2.3. Lemmatization and Stemming
- 2.4. POS Tagging
- 2.5. Integer Encoding
- 2.6. Padding
- 2.7. One-Hot Encoding

Lemmatization : returns the base or canonical form of the word that would match a dictionary entry (lemma).

Nouns and adjectives: masculine singular form

Verbs: infinitive form

```
In [12]: from nltk.stem import WordNetLemmatizer
         n=WordNetLemmatizer()
         words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
         print([n.lemmatize(w) for w in words])

         ['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 'starting']

In [13]: n.lemmatize('dies', 'v')
Out[13]: 'die'

In [14]: n.lemmatize('watched', 'v')
Out[14]: 'watch'

In [15]: n.lemmatize('has', 'v')
Out[15]: 'have'
```

Stemming: removes affixes from a word returning only the root or stem

Porter Stemmer (English). Ordered set of rules: (condition) suffix_1 → suffix_2

Step 1a

SSES	→SS	caresses	→caress
IES	→I	ponies	→poni
		ties	→ti
SS	→SS	caress	→caress
S	→	cats	→cat

Step 1b

(m>0) EED	→EE	feed	→feed
		agreed	→agree
(*v*) ED	→	plastered	→plaster
		bled	→bled
(*v*) ING	→	motoring	→motor
		sing	→sing

Step 2

(m>0) ATIONAL	→ATE	relational	→relate
(m>0) TIONAL	→TION	conditional	→condition

Step 3

(m>0) ICATE	→IC	triplicate	→triplic
(m>0) ATIVE	→	formative	→form

A word can be represented as **[C](VC)^m[V]**
C = arbitrary number of consonants
V = arbitrary number of vocals

(m>0): the stem contains at least 1 sequence VC

(*v*): the stem contains at least 1 vocal

https://snowballstem.org/

Stemming:

In [16]:
```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
s = PorterStemmer()
text="This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights
words=word_tokenize(text)
print(words)
```

```
['This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate',
'copy', ',', 'complete', 'in', 'all', 'things', '--', 'names', 'and', 'heights', 'and', 'soundings', '--', 'with', 'the',
'single', 'exception', 'of', 'the', 'red', 'crosses', 'and', 'the', 'written', 'notes', '.']
```

In [17]:
```
print([s.stem(w) for w in words])
```

```
['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi',
',', 'complet', 'in', 'all', 'thing', '--', 'name', 'and', 'height', 'and', 'sound', '--', 'with', 'the', 'singl', 'excep
t', 'of', 'the', 'red', 'cross', 'and', 'the', 'written', 'note', '.']
```

Stemming:

In [18]:
```
   #Stemmization of porter algorithm has rules as below.
#ALIZE → AL
#ANCE → Delete
#ICAL → IC
```

In [19]:
```
words=['formalize', 'allowance', 'electricical']
print([s.stem(w) for w in words])
```

```
['formal', 'allow', 'electric']
```

In [20]:
```
from nltk.stem import PorterStemmer
s=PorterStemmer()
words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
print([s.stem(w) for w in words])
```

```
['polici', 'do', 'organ', 'have', 'go', 'love', 'live', 'fli', 'die', 'watch', 'ha', 'start']
```

In [21]:
```
from nltk.stem import LancasterStemmer
l=LancasterStemmer()
words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
print([l.stem(w) for w in words])
```

```
['policy', 'doing', 'org', 'hav', 'going', 'lov', 'liv', 'fly', 'die', 'watch', 'has', 'start']
```

Unit 2.

Text Preprocessing

- 2.1. Tokenization
- 2.2. Stop Words
- 2.3. Lemmatization and Stemming
- 2.4. POS Tagging
- 2.5. Integer Encoding
- 2.6. Padding
- 2.7. One-Hot Encoding

POS Tagging

Part of Speech (POS) tag set in English:

Tag Set	Description	Example
CC	Coordinating conjunction	
CD	Cardinal digit	
DT	Determiner	
EX	Existential 'there'	there is
FW	Foreign word	
IN	Preposition/subordinating conjunction	
JJ	Adjective	Big
JJR	Adjective, comparative	Bigger
JJS	Adjective, superlative	Biggest
LS	List marker	1)
MD	Modal	could, will
NN	Noun, singular	Desk
NNS	Noun, plural	Desks
NNP	Proper noun, singular	Harrison
NNPS	Proper noun, plural	Americans
PDT	Predeterminer	'all' the kids
POS	Possessive ending	parent's

Part of Speech (POS) tag set in English:

Tag Set	Description	Example
PRP	Personal pronoun	I, he, she
PRP$	Possessive pronoun	my, his, hers
RB	Adverb	very, silently
RBR	Adverb, comparative	better
RBS	Adverb, superlative	best
RP	Particle	give up
TO	To	go 'to' the store
UH	Interjection	errrrrrrrm
VB	Verb, base form	take
VBD	Verb, past tense	took
VBG	Verb, gerund/present participle	taking
VBN	Verb, past participle	taken
VBP	Verb, sing. Present, non-3d	take
VBZ	Verb, 3rd person sing. Present	takes
WDT	Wh-determiner	which
WP	Wh-pronoun	who, what
WP$	Possessive wh-pronoun	whose
WRB	Wh-abverb	where, when

51_ex_0507.ipynb

Part of Speech (POS) tag set in English:

In [22]:
```
# Test sentence.
my_sentence = "The Colosseum was built by the emperor Vespassian"
```

In [24]:
```
import nltk
```

In [25]:
```
# Simple pre-processing.
my_words = nltk.word_tokenize(my_sentence)
for i in range(len(my_words)):
    my_words[i] = my_words[i].lower()
my_words
```

Out[25]:
```
['the', 'colosseum', 'was', 'built', 'by', 'the', 'emperor', 'vespassian']
```

In [26]:
```
# POS tagging.
# OUTPUT: A list of tuples.
my_words_tagged = nltk.pos_tag(my_words)
my_words_tagged
```

Out[26]:
```
[('the', 'DT'),
 ('colosseum', 'NN'),
 ('was', 'VBD'),
 ('built', 'VBN'),
 ('by', 'IN'),
 ('the', 'DT'),
 ('emperor', 'NN'),
 ('vespassian', 'NN')]
```

NLTK Library

NLTK (Natural Language Toolkit):

- One of the most used Python libraries for the NLP.

a) Tokenization: sent_tokenize, word_tokenize, etc.

b) Stop words: 179 in total. The list of stop words needs to be downloaded.

```
nltk.download('stopwords')
```

c) Stemming: PorterStemmer, LancasterStemmer, SnowballStemmer, etc.

d) Lemmatization: WordNetLemmatizer.

e) POS tagging: Uses the “Penn Treebank tag set”.

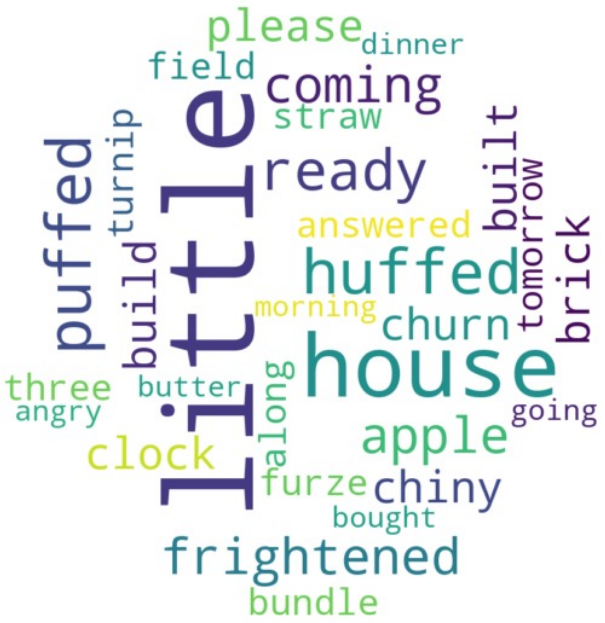
f) Lexical resource: WordNet, SentiWordNet, etc. Useful for sentiment analysis.

Visualization

52_ex_0508.ipynb

Word Cloud:

- Visualization of the keywords where the size is related to the frequency.



Word Cloud:

- We can create a word cloud following the steps below:

1) Tokenize by words.

2) Apply cleaning and normalization.

3) Make a frequency table of the words.

4) Sort by frequency and keep only the top keywords. The number of keywords is adjustable.

5) Customize the arguments of the WordCloud() function.

Coding Exercise #0501~0508

Follow practice steps on 'ex_0506~8.ipynb' file

Unit 2.

Text Preprocessing

- 2.1. Tokenization
- 2.2. Stop Words
- 2.3. Lemmatization and Stemming
- 2.4. POS Tagging
- 2.5. Integer Encoding
- 2.6. Padding
- 2.7. One-Hot Encoding

Integer Encoding

Consists in replacing the categories (tokens) by numbers

Numbers can be assigned arbitrarily

Preserves the category order in the sentence

Advantages:

- Easy to implement
- Does not expand the feature space (substitutes a categorical value by a numeric value)

Limitations:

- Does not capture any information about categories and its relationships
- Not suitable for linear models

```
['a barber is a person.',              [1,2,3,1,4],
 'a barber is good person.',    ⟶      [1,2,3,5,4],
 'a barber is huge person.']            [1,2,3,6,4],
```

Index: { 'a' : 1 ; 'barber' : 2 ; 'is' : 3 ; 'person' : 4 ; 'good' : 5 ; 'huge' : 6 }

Integer Encoding

Exercise

In [1]:
```
import nltk
nltk.download('punkt')
```

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

Out[1]:
```
True
```

In [2]:
```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

In [3]:
```
text = "A barber is a person. a barber is a good person. a barber is a huge person. he Knew A Secret!. The Secret He Kept i
```

In [4]:
```
text = sent_tokenize(text)
print(text)
```

```
['A barber is a person.', 'a barber is a good person.', 'a barber is a huge person.', 'he Knew A Secret!.', 'The Secret He
Kept is huge secret.', 'Huge secret.', 'His barber kept his word.', 'a barber kept his word.', 'His barber kept his secre
t.', 'But keeping and keeping such a huge secret to himself was driving the barber crazy.', 'the barber went up a huge mou
ntain.']
```

Integer Encoding:

In [4]:
```
# Cleaning and word tokenization
vocab = {}  # Python's dictionary datatype
sentences = []
stop_words = set(stopwords.words('english'))

for i in text:
    sentence = word_tokenize(i)  # executes word tokenization
    result = []

    for word in sentence:
        word = word.lower()  # Reduce the number of words by lowercasing all words
        if word not in stop_words:  # Remove stop words in case of word tokenization.
            if len(word) > 2:    # Remove additional words if the length of the word is lower or equal to 2
                result.append(word)
                if word not in vocab:
                    vocab[word] = 0
                vocab[word] += 1
    sentences.append(result)
print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept',
'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secre
t'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', 'huge', 'mountain']]
```

In [5]:
```
print(vocab)
```

```
{'barber': 8, 'person': 3, 'good': 1, 'huge': 5, 'knew': 1, 'secret': 6, 'kept': 4, 'word': 2, 'keeping': 2, 'driving': 1,
'crazy': 1, 'went': 1, 'mountain': 1}
```

In [6]:
```
print(vocab["barber"])
```

```
8
```

Align in the order of frequency

In [7]:
```
vocab_sorted = sorted(vocab.items(), key = lambda x:x[1], reverse = True)
print(vocab_sorted)
```

```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3), ('word', 2), ('keeping', 2), ('good', 1), ('knew',
1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
```

2.5. Integer Encoding

Assign low integer index for words of high frequency

In [8]:
```
word_to_index = {}
i=0
for (word, frequency) in vocab_sorted :
    if frequency > 1 :     # Low frequency words are deleted. (It was learned in the Cleaning section.)
        i=i+1
        word_to_index[word] = i
print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

Use top 5 words

In [9]:
```
vocab_size = 5
words_frequency = [w for w,c in word_to_index.items() if c >= vocab_size + 1] # Remove words of index higher than 5
for w in words_frequency:
    del word_to_index[w] # Remove index information from the corresponding word
print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

Changing each word from sentences stored after tokenization to an integer by using word_to_index

In [11]:
```
word_to_index['OOV'] = len(word_to_index) + 1
```

In [12]:
```
encoded = []
for s in sentences:
    temp = []
    for w in s:
        try:
            temp.append(word_to_index[w])
        except KeyError:
            temp.append(word_to_index['OOV'])
    encoded.append(temp)
print(encoded)
```

```
[[1, 5], [1, 6, 5], [1, 3, 5], [6, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [6, 6, 3, 2, 6, 1, 6], [1,
6, 3, 6]]
```

Using counter

In [13]:
```
from collections import Counter
```

In [14]:
```
print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept',
'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secre
t'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', 'huge', 'mountain']]
```

In [15]:
```
words = sum(sentences, [])
 # Above task can be executed by words=np.hstack(sentences)
print(words)
```

```
['barber', 'person', 'barber', 'good', 'person', 'barber', 'huge', 'person', 'knew', 'secret', 'secret', 'kept', 'huge',
'secret', 'huge', 'secret', 'barber', 'kept', 'word', 'barber', 'kept', 'word', 'barber', 'kept', 'secret', 'keeping', 'ke
eping', 'huge', 'secret', 'driving', 'barber', 'crazy', 'barber', 'went', 'huge', 'mountain']
```

In [16]:
```
vocab = Counter(words)    # You can easily count all frequency of the words by using Python's Counter module.
print(vocab)
```

```
Counter({'barber': 8, 'secret': 6, 'huge': 5, 'kept': 4, 'person': 3, 'word': 2, 'keeping': 2, 'good': 1, 'knew': 1, 'driv
ing': 1, 'crazy': 1, 'went': 1, 'mountain': 1})
```

In [17]:
```
print(vocab["barber"])    # Print the frequency of the word 'barber'
```

```
8
```

2.5. Integer Encoding

Assign low integer index for words of high frequency

In [18]:
```
vocab_size = 5
vocab = vocab.most_common(vocab_size)  # Save top 5 words with the highest frequency
vocab
```

Out[18]:
```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3)]
```

In [19]:
```
word_to_index = {}
i = 0
for (word, frequency) in vocab :
    i = i+1
    word_to_index[word] = i
print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

Use NLTK's FreqDist

In [20]:
```
from nltk import FreqDist
import numpy as np
```

In [21]:
```
# Remove sentence section with np.hstack and use it as input. Ex) 'barber', 'person', 'barber', 'good' ···
vocab = FreqDist(np.hstack(sentences))
```

In [22]:
```
print(vocab["barber"])
```

```
8
```

In [23]:
```
vocab_size = 5
vocab = vocab.most_common(vocab_size) # Save top 5 words with the highest frequency
vocab
```

Out[23]:
```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3)]
```

In [24]:
```
word_to_index = {word[0] : index + 1 for index, word in enumerate(vocab)}
print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

Use Enumerate

In [25]:
```
test=['a', 'b', 'c', 'd', 'e']
for index, value in enumerate(test): # Assign index starting from 0 in the order of input
  print("value : {}, index: {}".format(value, index))
```

```
value : a, index: 0
value : b, index: 1
value : c, index: 2
value : d, index: 3
value : e, index: 4
```

Use Keras

In [27]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [28]:
```
sentences=[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret',
```

In [29]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
# fit_on_texts(), with corpus as input,  generates vocabulary based on word frequency
```

In [30]:
```
print(tokenizer.word_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7, 'good': 8, 'knew': 9, 'driving': 1
0, 'crazy': 11, 'went': 12, 'mountain': 13}
```

In [34]:
```
print(tokenizer.word_counts)
```

```
OrderedDict([('barber', 8), ('person', 3), ('good', 1), ('huge', 5), ('knew', 1), ('secret', 6), ('kept', 4), ('word', 2),
('keeping', 2), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)])
```

In [39]:
```
print(tokenizer.texts_to_sequences(sentences))
```

```
[[1, 5], [1, 5], [1, 3, 5], [2], [2, 4, 3, 2], [3, 2], [1, 4], [1, 4], [1, 4, 2], [3, 2, 1], [1, 3]]
```

Use Keras

```
[27]  1 from tensorflow.keras.preprocessing.text import Tokenizer
```

```
[28]  1 sentences=[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept', 'huge', 'secr
```

```
[29]  1 tokenizer = Tokenizer(num_words=7,oov_token='OOV')
      2 tokenizer.fit_on_texts(sentences)
      3 # When putting corpus in fit_on_texts(), it is possible to generate the group of words based on frequency.
```

```
[30]  1 print(tokenizer.word_index)
```

```
{'OOV': 1, 'barber': 2, 'secret': 3, 'huge': 4, 'kept': 5, 'person': 6, 'word': 7, 'keeping': 8, 'good': 9, 'knew': 10, 'driving': 11, 'crazy': 12,
```

```
[31]  1 print(tokenizer.word_counts)
```

```
OrderedDict([('barber', 8), ('person', 3), ('good', 1), ('huge', 5), ('knew', 1), ('secret', 6), ('kept', 4), ('word', 2), ('keeping', 2), ('driving
```

```
[32]  1 print(tokenizer.texts_to_sequences(sentences))
```

```
[[2, 6], [2, 1, 6], [2, 4, 6], [1, 3], [3, 5, 4, 3], [4, 3], [2, 5, 1], [2, 5, 1], [2, 5, 3], [1, 1, 4, 3, 1, 2, 1], [2, 1, 4, 1]]
```

Coding Exercise

Follow practice steps on 'Integer Encoding.ipynb' file

Unit 2.

Text Preprocessing

- 2.1. Tokenization
- 2.2. Stop Words
- 2.3. Lemmatization and Stemming
- 2.4. POS Tagging
- 2.5. Integer Encoding
- 2.6. Padding
- 2.7. One-Hot Encoding

Padding

Padding using NumPy

In [1]:
```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [2]:
```
sentences = [['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'],
             ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'],
             ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secret'],
             ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'],
             ['barber', 'went', 'huge', 'mountain']]
```

In [3]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences) # fit_on_texts(), with corpus as input, generates vocabulary based on word frequency
```

In [4]:
```
encoded = tokenizer.texts_to_sequences(sentences)
print(encoded)
```

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1,
12, 3, 13]]
```

In [5]:
```
max_len = max(len(item) for item in encoded)
print(max_len)
```

```
7
```

Padding using NumPy

In [6]:
```
for item in encoded: # For each sentence
    while len(item) < max_len: # If smaller than max_len
        item.append(0)

padded_np = np.array(encoded)
padded_np
```

Out[6]:
```
array([[ 1,  5,  0,  0,  0,  0,  0],
       [ 1,  8,  5,  0,  0,  0,  0],
       [ 1,  3,  5,  0,  0,  0,  0],
       [ 9,  2,  0,  0,  0,  0,  0],
       [ 2,  4,  3,  2,  0,  0,  0],
       [ 3,  2,  0,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  2,  0,  0,  0,  0],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0,  0,  0]])
```

2.6. Padding

Padding using Keras preprocessing tool

In [8]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [10]:
```
encoded = tokenizer.texts_to_sequences(sentences)
print(encoded)
```

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1,
12, 3, 13]]
```

In [11]:
```
padded = pad_sequences(encoded)
padded
```

Out[11]:
```
array([[ 0,  0,  0,  0,  0,  1,  5],
       [ 0,  0,  0,  0,  1,  8,  5],
       [ 0,  0,  0,  0,  1,  3,  5],
       [ 0,  0,  0,  0,  0,  9,  2],
       [ 0,  0,  0,  2,  4,  3,  2],
       [ 0,  0,  0,  0,  0,  3,  2],
       [ 0,  0,  0,  0,  1,  4,  6],
       [ 0,  0,  0,  0,  1,  4,  6],
       [ 0,  0,  0,  0,  1,  4,  2],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 0,  0,  0,  1, 12,  3, 13]])
```

Coding Exercise

Follow practice steps on 'Padding.ipynb' file

Unit 2.

Text Preprocessing

2.1. Tokenization

2.2. Stop Words

2.3. Lemmatization and Stemming

2.4. POS Tagging

2.5. Integer Encoding

2.6. Padding

2.7. One-Hot Encoding

One-Hot Encoding

Represents categorical variables (tokens) as binary vectors

Creates a column for each category, indicating the presence (1) o absence of the category (0)

Advantages:

- Easy to implement
- Does not establish any order between the categories
- Can be more expressive
- Can manage unseen categories

Limitations:

- Increases the dimensionality
- Creates sparse data

Each token is a new feature (column)

```
['a barber is a person.']
```

```
[1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ],
[1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],
[0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],
[0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ],
[1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ],
[0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]
```

Index: { 'a' : 1 ; 'barber' : 2 ; 'is' : 3 ; 'person' : 4 ; 'good' : 5 ; 'huge' : 6 ; 'he' : 7 ; 'knew' : 8 ; 'secret' : 9}

One-Hot Encoding

One-hot encoding using Keras preprocessing tool

```
[ ]  1 text = "A barber is a person. a barber is good person. a barber is huge person."
```

```
[ ]  1 from tensorflow.keras.preprocessing.text import Tokenizer
     2 from tensorflow.keras.utils import to_categorical
```

```
[ ]  1 t = Tokenizer()
     2 t.fit_on_texts([text])
     3 print(t.word_index) # Print the encoding result of each word
```

```
{'a': 1, 'barber': 2, 'is': 3, 'person': 4, 'good': 5, 'huge': 6}
```

```
[ ]  1 encoded=t.texts_to_sequences([text])
     2 print(encoded)
```

```
[[1, 2, 3, 1, 4, 1, 2, 3, 5, 4, 1, 2, 3, 6, 4]]
```

2.7. One-hot Encoding

One-hot encoding using Keras preprocessing tool

```
[ ]  1 encoded=t.texts_to_sequences([text])
     2 print(encoded)
```

```
[[1, 2, 3, 1, 4, 1, 2, 3, 5, 4, 1, 2, 3, 6, 4]]
```

```
[ ]  1 one_hot = to_categorical(encoded)
     2 print(one_hot)
```

```
[[[0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0.]
  [0. 0. 0. 0. 1. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 1.]
  [0. 0. 0. 0. 1. 0. 0.]]]
```

Coding Exercise

Follow practice steps on 'One-hot Encoding.ipynb' file