Morphology – Getting Our Feet Wet

Morphology may be defined as the study of the composition of words using morphemes. A morpheme is the smallest unit of language that has meaning. In this chapter, we will discuss stemming and lemmatizing, stemmer and lemmatizer for non-English languages, developing a morphological analyzer and morphological generator using machine learning tools, search engines, and many such concepts.

In brief, this chapter will include the following topics: •Introducing morphology •Understanding stemmer •Understanding lemmatization •Developing a stemmer for non-English languages •Morphological analyzer •Morphological generator •Search engine

Introducing morphology

Morphology may be defined as the study of the production of tokens with the help of morphemes. A morpheme is the basic unit of language carrying meaning. There are two types of morpheme: stems and affixes (suffixes, prefixes, infixes, and circumfixes).

Stems are also referred to as free morphemes, since they can even exist without adding affixes. Affixes are referred to as bound morphemes, since they cannot exist in a free form and they always exist along with free morphemes. Consider the word unbelievable. Here, believe is a stem or a free morpheme. It can exist on its own. The morphemes un and able are affixes or bound morphemes. They cannot exist in a free form, but they exist together with stem.

There are three kinds of language, namely isolating languages, agglutinative languages, and inflecting languages. Morphology has a different meaning in all these languages. Isolating languages are those languages in which words are merely free morphemes and they do not carry any tense (past, present, and future) and number (singular or plural) information. Mandarin Chinese is an example of an isolating language.

Agglutinative languages are those in which small words combine together to convey compound information. Turkish is an example of an agglutinative language.

Inflecting languages are those in which words are broken down into simpler units, but all these simpler units exhibit different meanings. Latin is an example of an inflecting language

Morphological processes are of the following types: inflection, derivation, semiaffixes and combining forms, and cliticization. Inflection means transforming the word into a form so that it represents person, number, tense, gender, case, aspect, and mood. Here, the syntactic category of a token remains the same. In derivation, the syntactic category of a word is also changed. Semiaffixes are bound morphemes that exhibit words, such as quality, for example, noteworthy, antisocial, anticlockwise, and so on.

Understanding stemmer

Stemming may be defined as the process of obtaining a stem from a word by eliminating the affixes from a word. For example, in the case of the word raining, stemmer would return the root word or stem word rain by removing the affix from raining. In order to increase the accuracy of information retrieval, search engines mostly use stemming to get the stems and store them as indexed words. Search engines call words with the same meaning synonyms, which may be a kind of query expansion known as conflation. Martin Porter has designed a well-known stemming algorithm known as the Porter stemming algorithm. This algorithm is basically designed to replace and eliminate some well-known suffices present in English words. To perform stemming in NLTK, we can simply do an instantiation of the PorterStemmer class and then perform stemming by calling the stem method.

In [2]:
```
import nltk
from nltk.stem import PorterStemmer
stemmerporter = PorterStemmer()
stemmerporter.stem('working')
```

Out[2]: 'work'

In [3]:
```
stemmerporter.stem('happiness')
```

Out[3]: 'happi'

Types of Stemmers: i)PorterStemmer ii)LancasterStemmer iii)RegExp Stemmer iv)SnowballStemmer

LancasterStemmer

Lancaster stemming algorithm was introduced at Lancaster University. Similar to the PorterStemmer class, the LancasterStemmer class is used in NLTK to implement Lancaster stemming. However, one of the major differences between the two algorithms is that Lancaster stemming involves the use of more words of different sentiments as compared to Porter Stemming.

In [5]:
```
from nltk.stem import LancasterStemmer
stemmer_lan=LancasterStemmer()
stemmer_lan.stem('working')
```

Out[5]: 'work'

In [8]:
```
stemmer_lan.stem('happiness')
```

Out[8]:
```
'happy'
```

RegexStemmer

We can also build our own stemmer in NLTK using RegexpStemmer. It works by accepting a string and eliminating the string from the prefix or suffix of a word when a match is found.

In [9]:
```
from nltk.stem import RegexpStemmer
stemmer_regexp=RegexpStemmer('ing')
stemmer_regexp.stem('working')
```

Out[9]:
```
'work'
```

In [10]:
```
 stemmer_regexp.stem('happiness')
```

Out[10]:
```
'happiness'
```

In [11]:
```
 stemmer_regexp.stem('pairing')
```

Out[11]:
```
'pair'
```

SnowballStemmer

SnowballStemmer is used to perform stemming in 13 languages other than English. In order to perform stemming using SnowballStemmer, firstly, an instance is created in the language in which stemming needs to be performed. Then, using the stem() method, stemming is performed.

In [12]:
```
from nltk.stem import SnowballStemmer
SnowballStemmer.languages
```

Out[12]:
```
('arabic',
 'danish',
 'dutch',
 'english',
 'finnish',
 'french',
 'german',
 'hungarian',
 'italian',
 'norwegian',
 'porter',
 'portuguese',
 'romanian',
 'russian',
 'spanish',
 'swedish')
```

In [14]:
```
spanishstemmer=SnowballStemmer('spanish')
spanishstemmer.stem('comiendo')
```

Out[14]: 'com'

In [15]:
```
frenchstemmer=SnowballStemmer('french')
frenchstemmer.stem('manger')
```

Out[15]: 'mang'

Understanding lemmatization

Lemmatization is the process in which we transform the word into a form with a different word category. The word formed after lemmatization is entirely different. The built-in morphy() function is used for lemmatization in WordNetLemmatizer. The inputted word is left unchanged if it is not found in WordNet. In the argument, pos refers to the part of speech category of the inputted word.

Consider an example of lemmatization in NLTK:

In [18]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer_output=WordNetLemmatizer()
lemmatizer_output.lemmatize('working')
```

Out[18]: 'working'

In [19]:
```
lemmatizer_output.lemmatize('working',pos='v')
```

Out[19]: 'work'

In [20]:
```
lemmatizer_output.lemmatize('works')
```

Out[20]: 'work'

Developing a stemmer for non-English language

Polyglot is a software that is used to provide models called morfessor models that are used to obtain morphemes from tokens. The Morpho project's goal is to create unsupervised data-driven processes. The main aim of the Morpho project is to focus on the creation of morphemes, which is the smallest unit of syntax. Morphemes play an important role in natural language processing. Morphemes are useful in automatic recognition and the creation of language. With the help of the vocabulary dictionaries of Polyglot, morfessor models on the 50,000 tokens of different languages were used.

Let's see the code for obtaining the language table using polyglot:

Refer to Polyglot_Package_Usage

Morphological analyzer

Morphological analysis may be defined as the process of obtaining grammatical information from tokens, given their suffix information. Morphological analysis can be performed in three ways: morpheme-based morphology (or anitem and arrangement approach), lexeme-based morphology (or an item and process approach), and wordbased morphology (or a word and paradigm approach). A morphological analyzer may be defined as a program that is responsible for the analysis of the morphology of a given input token. It analyzes a given token and generates morphological information, such as gender, number, class, and so on, as an output.

We can determine the category of the word with the help of the following points: • Morphological hints: The suffix's information helps us detect the category of a word. For example, the -ness and –ment suffixes exist with nouns. • Syntactic hints: Contextual information is conducive to determine the category of a word. For example, if we have found the word that has the noun category, then syntactic hints will be useful for determining whether an adjective would appear before the noun or after the noun category. • Semantic hints: A semantic hint is also useful for determining the word's category. For example, if we already know that a word represents the name of a location, then it will fall under the noun category. • Open class: This is class of words that are not fixed, and their number keeps on increasing every day, whenever a new word is added to their list. Words in the open class are usually nouns. Prepositions are mostly in a closed class. For example, there can be an unlimited number of words in the of Persons list. So, it is an open class. • Morphology captured by the Part of Speech tagset: The Part of Speech tagset captures information that helps us perform morphology. For example, the word plays would appear with the third person and a singular noun. • Omorfi:Omorfi (Open morphology of Finnish) is a package that has been licensed by GNU GPL version 3. It is used for performing numerous tasks, such as language modeling, morphological analysis, rule-based machine translation, information retrieval, statistical machine translation, morphological segmentation, ontologies, and spell checking and correction

Morphological generator

A morphological generator is a program that performs the task of morphological generation. Morphological generation may be considered an opposite task of morphological analysis. Here, given the description of a word in terms of number, category, stem, and so on, the original word is retrieved. For example, if root = go, part of speech = verb, tense= present, and if it occurs along with a third person and singular subject, then a morphological generator would generate its surface form, goes.

There is a lot of Python-based software that performs morphological analysis and generation. Some of them are as follows:

• ParaMorfo: It is used to perform morphological generation and analysis of Spanish and Guarani nouns, adjectives, and verbs. • HornMorpho: It is used for the morphological generation and analysis of Oromo and Amharic nouns and verbs, as well as Tigrinya verbs. • AntiMorfo: It is used for the morphological generation and analysis of Quechua

adjectives, verbs, and nouns, as well as Spanish verbs. • MorfoMelayu: It is used for the morphological analysis of Malay words.

Other examples of software that is used to perform morphological analysis and generation are as follows:

• Morph is a morphological generator and analyzer for English for the RASP system • Morphy is a morphological generator, analyzer, and POS tagger for German • Morphisto is a morphological generator and analyzer for German • Morfette performs supervised learning (inflectional morphology) for Spanish and French

Search engine

In [73]:
```
def eliminatestopwords(self,list):
    return[ word for word in list if word not in self.stopwords ]
```

In [74]:
```
def tokenize(self,string):
    Str=self.clean(str)
    Words=str.split(" ")
    return [self.stemmer.stem(word,0,len(word)-1) for word in words]
```

In [75]:
```
def obtainvectorkeywordindex(self, documentList):
    vocabstring = "".join(documentList)
    vocablist = self.parser.tokenise(vocabstring)
    vocablist = self.parser.eliminatestopwords(vocablist)
    uniqueVocablist = util.removeDuplicates(vocablist)
    vectorIndex={}
    offset=0
    for word in uniqueVocablist:
        vectorIndex[word]=offset
        offset+=1
    return vectorIndex
```

In [76]:
```
def constructVector(self, wordString):
    Vector_val = [0] * len(self.vectorKeywordIndex)
    tokList = self.parser.tokenize(tokString)
    tokList = self.parser.eliminatestopwords(tokList)
    for word in toklist:
        vector[self.vectorKeywordIndex[word]] += 1;
    return vector
```

In [77]:
```
def cosine(vec1, vec2):
    return float(dot(vec1,vec2) / (norm(vec1) * norm(vec2)))
```

In [78]:
```
def searching(self,searchinglist):
    askVector = self.buildQueryVector(searchinglist)
    ratings = [util.cosine(askVector, textVector) for textVector in self.docum
entVectors]
    ratings.sort(reverse=True)
    return ratings
```

In [79]:
```
import sys
```

In [80]:
```
try:
    from nltk import wordpunct_tokenize
    from nltk.corpus import stopwords
except ImportError:
    print( 'Error has occured')
```

In [81]:
```
def _calculate_languages_ratios(text):
    languages_ratios = {}
    tok = wordpunct_tokenize(text)
    wor = [word.lower() for word in tok]
    for language in stopwords.fileids():
        stopwords_set = set(stopwords.words(language))
        words_set = set(wor)
        common_elements = words_set.intersection(stopwords_set)
        languages_ratios[language] = len(common_elements)
        return languages_ratios
```

In [82]:
```
def detect_language(text):
    ratios = _calculate_languages_ratios(text)
    most_rated_language = max(ratios, key=ratios.get)
    return most_rated_language
```

In [93]:
```
text = """
الثالث على التوالي عن التقدم الى داخل مدينة تكريت بسبب
انتشار قناصي التنظيم الذي ي"""
```

In [94]:
```
language = detect_language(text)
```

In [95]:
```
print(language)
```

```
arabic
```