**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL804** |

|  |  |
| --- | --- |
| **Practical No:** | 1 |
| **Title:** | Word Analysis |
| **Date of Performance:** | 23/02/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

**EXPERIMENT 1 – Word Analysis**

**AIM:** To understand and implement Word Analysis

# THEORY:

Word analysis is a process of learning more about word meanings by studying their origins and parts. A “morpheme” is the smallest meaningful part of a word.

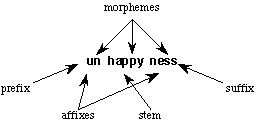
# How is Word Analysis Important?

Words are more than a collection of letters: wonderful, review, disorder. Students need to understand that words include more than one morpheme, each one with a story. Analyzing words helps students see how words connect to cultures and linguistic patterns.

Linguistics sets out to describe language. Any description needs some terminology with which to set out its description. We can think of this as the technical vocabulary of the discipline. Natural languages have their terms to describe themselves. For instance, we colloquially talk about "words", "phrases", "sentences" and "paragraphs".

We'll look at just the definition of the word. In-text like this, we can easily spot "words" because they are separated from each other by spaces or by punctuation. However, if you record ordinary, conversational speech, you will find that there are no breaks between words. Despite this, we could isolate units that we use in speech again and again, but in different combinations. This suggests that there is a small unit something like a word. But just how do we define a "word"? We will all agree that *black* and *bird* are words. Is *blackbird* one word or two words? Is *blackbirds* the same word as *blackbird* or a separate word?

### What is Morphology?

Morphology is the study of the structure and formation of words. Its most important unit is the *morpheme*, which is defined as the "minimal unit of meaning". (Linguistics textbooks usually define it slightly differently as "the minimal unit of grammatical analysis".) Consider a word like "unhappiness". This has three parts:

There are three morphemes, each carrying a certain amount of meaning. *un* means "not", while

*ness* means "being in a state or condition". *Happy* is a *free morpheme* because it can appear on its own (as a "word" in its own right). *Bound morphemes* have to be attached to a free morpheme, and so cannot be words in their own right. Thus, you can't have sentences in English such as "Jason feels very un ness today".

Morphological processes

There are several morphological processes of which some are more important than others for NLP. The account given here is selective and unusual in that it points out the practical aspects of the processes selected.

*Inflection*

Inflection is the process of changing the form of a word so that it expresses information such as number, person, case, gender, tense, mood and aspect, but the syntactic category of the word remains unchanged. As an example, the plural form of the noun in English is usually formed from the singular form by adding an *s.*

* car / cars
* table / tables
* dog / dogs

In each of these cases, the syntactic category of the word remains unchanged.

It doesn't take long to find examples where the simple rule given above doesn't fit. So there are smaller groups of nouns that form plurals in different ways:

* wolf/wolves
* knife/knives
* switch/switches.

A little more thought and we can think of apparently completely irregular plural forms, such as:

* foot / feet
* child/children.

English verbs are relatively simple (especially compared with languages like Finnish which has over 12,000 verb inflections).

* mow - stem
* mows - the third person singular, present tense
* mowed - past tense and past participle
* mowing - present continuous tense

*NLP aspects*

NLP systems for English often don't include any morphological process, especially if they are small-scale systems. Where English-based systems do include analysis of inflection, the regular forms of words are analysed using one of the standard techniques (for instance, Finite State

Automata), while the exceptions (the irregular words) are each listed individually. This means that regular forms have to be entered in the dictionary only once, which can save a lot of space and data entry if the dictionary holds a lot of syntactic and semantic information.

# Derivation

As was seen above, inflection doesn't change the syntactic category of a word. Derivation *does* change the category. Linguists classify derivation in English according to whether or not it induces a change of pronunciation. For instance, adding the suffix *it* changes the pronunciation of the root of *active* so the stress is on the second syllable: *activity.* The addition of the suffix *al* to *approve* doesn't change the pronunciation of the root: *approval.*

## NLP aspects

The obvious use of derivational morphology in NLP systems is to reduce the number of forms of words to be stored. So, if there is already an entry for the base form of the verb *sing,* then it should be possible to add rules to map the nouns *singer* and *singers* onto the same entry. The problem here is that the detection of the derivation of the *singer* from *sing* must allow also allow the morphological analyser to contribute the information that is special to the *singer.* This seems a little obscure, but an example will make it clearer. The addition of *er* to the word indicates that it is a person who is undertaking the action. This semantic information must be added to the stored information from the dictionary entry for the root form *sing* so that the correct meaning of the sentence can be found. This seems fine, but suppose the next two words are: *recorder* and *dragster.* The use of the *er* morpheme can't be taken to necessarily mean someone who undertakes the action represented by the root form. Derivational morphological analysers can do this quite easily, because they are always attempting to reduce words to smaller units. Derivational morphology is particularly useful for Machine Translation. Successful MT has to process large quantities of text which can contain many previously unseen words.

# Various methods of morphological analysis

Various NLP (Natural Language Processing) research groups have developed different methods and algorithms for morphological analysis. Some of the algorithms are language-dependent and some of them are language independent. Finite State Automata (FSA). Two Level Morphology Finite State Transducer (FST).

* Stemmer Algorithm.
* Corpus-Based Approach
* DAWG (Directed Acrylic Word Graph)
* Paradigm Based Approach.

# CODE:

**1:**

>>> import nltk

>>> from nltk.stem import PorterStemmer

>>> stemmerporter = PorterStemmer()

>>> stemmerporter.stem('working') 'work'

>>> stemmerporter.stem('happiness') 'Happi'

**2:**

from polyglot.text import Text, Word

words = ["preprocessing", "processor", "invaluable", "thankful", "crossed"]

for w in words:

w = Word(w, language="en")

print("{:<20}{}".format(w, w.morphemes)) preprocessing ['pre', 'process', 'ing'] processor ['process', 'or'] invaluable ['in', 'valuable'] thankful ['thank', 'ful']

crossed ['cross', 'ed']

# CONCLUSION:

Thus, to conclude, we have understood fundamental concepts like, Morphology which is the study of the structure and formation of words. At the same time, we have understood the various elements and aspects of this process. Finally, we also understand the various types of Morphology and their way of distinguishing from each other.

**POST LAB:**

1]Discuss various application domains of Morphological Analysis.

Ans: -

Applications of morphological analysis:

1. Text to Speech Synthesis

Morphological information plays an important role in determining the pronunciation of a homograph. Morphological analysis can be used to reduce the size of the lexicon also. Because here one needs to store only the root word and various inflexions need not be stored.

1. Machine Translation

The morphological analysis provides the information of the word such as number, gender etc. This information can be used in the target language to generate the correct form of the word.

1. Spell Checker

The advantage associated with a morphology-based spell checker is that it can handle the Name Entity problem. If any new word is found which is not stored already in the lexicon, then that can be included in the particular paradigm of the lexicon.

1. Search Engine

Morphological Analysis and Generation improves the result of the search engine.

1. Information Retrieval

Morphological Analysis can be applied to the words which are not present in the dictionary, after getting the root word as an output of Morphological Analysis, that root word is searched in the dictionary.

2] How a word is different from a morpheme?

Ans: -

A morpheme is the smallest meaningful unit in a language. A morpheme is not necessarily the same as a [word](https://en.wikipedia.org/wiki/Word). The main difference between a morpheme and a word is that a morpheme [sometimes does not](https://en.wikipedia.org/wiki/Bound_and_free_morphemes) stand alone, but a word, by definition, always stands alone. The field of [linguistic](https://en.wikipedia.org/wiki/Linguistics) study dedicated to morphemes is called [morphology](https://en.wikipedia.org/wiki/Morphology_(linguistics)).

When a morpheme can stand alone, it is considered a [root](https://en.wikipedia.org/wiki/Root_(linguistics)) because it has a meaning of its own (such as the morpheme *cat*). When it depends on another morpheme to express an idea, it is an [affix](https://en.wikipedia.org/wiki/Affix) because it has a grammatical function (such as the *–s* in *cats* to indicate plurality).[[1]](https://en.wikipedia.org/wiki/Morpheme#cite_note-1) Every word is composed of one or more morphemes.

|  |
| --- |
| **Examples** |
| * "Unbreakable" is composed of three morphemes: *un-* (a bound morpheme signifying "not"), *-break-* (the [root](https://en.wikipedia.org/wiki/Root_(linguistics)), a free morpheme), and *-able* (a free morpheme signifying "can be done"). * [Allomorphs](https://en.wikipedia.org/wiki/Allomorph) of the plural morpheme for regular nouns: /s/ (e.g., in *cats* [/kæts/](https://en.wikipedia.org/wiki/Help:IPA/English)), /ɪz, əz/ (e.g., in *dishes* [/dɪʃɪz/](https://en.wikipedia.org/wiki/Help:IPA/English)), and /z/ (e.g., in *dogs* [/dɒɡz/](https://en.wikipedia.org/wiki/Help:IPA/English)). |

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL804** |

|  |  |
| --- | --- |
| **Practical No:** | 2 |
| **Title:** | Word Generation |
| **Date of Performance:** | 23/02/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

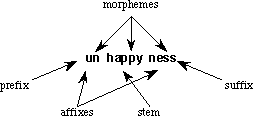
**Experiment No. – 2: Word Generation**

**AIM:** To understand and implement Word Generation

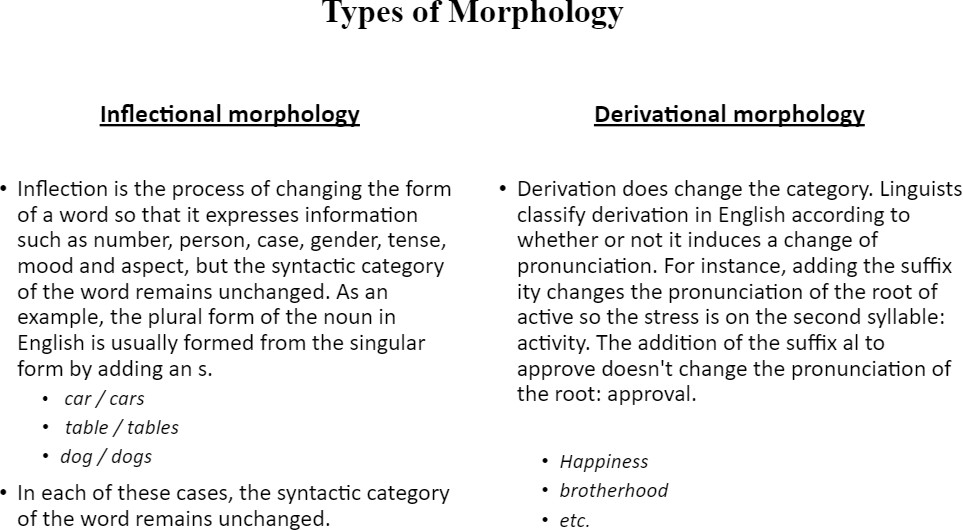
**THEORY:**

Morphological rules are exceptions to the orthographic rules used when breaking a word into its stem and modifiers. An example would be while one normally pluralizes a word in English by adding 's' as a suffix, the word 'fish' does not change when pluralized. Contrast this to orthographic rules which contain general rules. Both of these types of rules are used to construct systems that can do morphological parsing. Applications of morphological processing include machine translation, spell checker, and information retrieval.

Morphology is the study of the structure and formation of words. Its most important unit is the morpheme, which is defined as the "minimal unit of meaning". (Linguistics textbooks usually define it slightly differently as "the minimal unit of grammatical analysis".) Consider a word like "unhappiness". This has three parts:



There are three morphemes, each carrying a certain amount of meaning. un means "not", while ness means "being in a state or condition". Happy is a free morpheme because it can appear on its own (as a "word" in its own right). Bound morphemes have to be attached to a free morpheme, and so cannot be words in their own right. Thus, you can't have sentences in English such as "Jason feels very un ness today".



# Approaches of Morphology:

Morpheme based morphology: Word forms are analyzed as arrangements of morphemes. Morphemes- smallest linguistic unit with a grammatical function.

Lexeme based Morphology:

Lexeme-based morphology usually takes what is called an "item-and-process" approach. Instead of analyzing a word form as a set of morphemes arranged in sequence, a word form is said to be the result of applying rules that alter a word form or stem to produce a new one.

Word-based Morphology:

Word-based morphology is (usually) a word and paradigm approach. Instead of stating rules to combine morphemes into word forms, or to generate word forms from stems, word-based morphology states generalizations that hold between the forms of inflectional paradigms.

# Algorithm:

* Is the inverse process of morphological analysis.
* Word generation is a deterministic process.
* We take into consideration the root, category, gender, number, tense, person and case (in the case of Hindi) to generate the word based on the analysis done, i.e. using root and suffix information to generate the word.

# CODE:

#word generation

wordList = [

{ "word":"boy", "rt":"boy", "cat":"n", "gen":"m", "num":"sg", "per":"1","tense":"","case":""},

{ "word":"boys", "rt":"boy", "cat":"n", "gen":"m", "num":"pl", "per":"","tense":"","case":""},

{ "word":"plays", "rt":"play", "cat":"v", "gen":"", "num":"pl", "per":"3","tense":"pr","case":""},

{ "word":"play", "rt":"play", "cat":"v", "gen":"", "num":"sg", "per":"1","tense":"pr","case":""},

{"word":"पढTई","rt": "पढ", "cat": "v" ,"gen": "f" ,"num":"sg", "per":"1", "tense":"fu","case":"dir"},

]

def process(root, num, per, tense,case): for entry in wordList:

if (root == entry["rt"] and num == entry["num"] and per == entry["per"] and tense==entry["tense"] and case==entry["case"]):

print("ANS = " + entry["word"]) else:

continue

process("play","pl","3","pr","")

process("पढ","sg","1","fu","dir")

# CONCLUSION:

Thus, we have successfully implemented word generation.

# POST LAB:

Q. Discuss and Compare different techniques in word generation in natural language processing

Ans: -

1. Named Entity Recognition (NER)
2. Tokenization
3. Stemming and Lemmatization
4. Bag of Words
5. Natural language generation
6. Sentiment Analysis
7. Sentence Segmentation
8. Named Entity Recognition (NER)

This technique is one of the most popular and advantageous techniques in Semantic analysis, Semantics is something conveyed by the text. Under this technique, the algorithm takes a phrase or paragraph as input and identifies all the nouns or names present in that input.

There are many popular use cases of this algorithm below we are mentioning some of the daily use cases;

1. News Categorization:> This algorithm automatically scans all the news articles and extracts all sorts of information, like individuals, companies, organizations, people, celebrities name, places from that article. Using this algorithm, we can easily classify news content into different categories.
2. Efficient Search Engine:> The Named entity recognition algorithm applies to all the articles, results, news to extract relevant tags and stores them separately. These will boost up the searching process and make an efficient search engine.
3. Customer Support :> You must have read out thousands of feedbacks provided by people concerning heavy traffic areas on Twitter daily. If Named Entity Recognition API is used then we can easily pull out all the keywords (or tags) to inform concerned traffic police departments.
4. Tokenization

First of all, understanding the meaning of Tokenization is splitting the whole text into a list of tokens, lists can be anything such as words, sentences, characters, numbers, punctuation, etc. Tokenization has two main advantages, one is to reduce search to a significant degree, and the second is to be effective in the use of storage space. The process of mapping sentences from character to strings and strings into words are initially the basic steps of any NLP problem because to understand any text or document we need to understand the meaning of the text by interpreting words/sentences present in the text. Tokenization is an integral part of any Information Retrieval (IR) system, it not only involves the pre-process of text but also generates tokens respectively that are used in the indexing/ranking process. There are various tokenization’ techniques available among which Porter’s Algorithm is one of the most prominent techniques.

1. Stemming and Lemmatization

The increasing size of data and information on the web is an all-time high from the past couple of years. This huge data and information demand the necessary tools and techniques to extract inferences with much ease. “Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form - generally a written form of the word.” For example, what stemming does, basically it cuts off all the suffixes. So after applying a step of stemming on the word “playing”, it becomes “play”, or like, “asked” becomes “ask”. Lemmatization usually refers to do things with the proper use of vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. In simple words, Lemmatization deals with the lemma of a word that involves reducing the word form after understanding the part of speech (POS) or context of the word in any document.

1. Bag of Words

The bag of words technique is used to pre-process text and to extract all the features from a text document to use in Machine Learning modelling. It is also a representation of any text that elaborates/explains the occurrence of the words within a corpus (document). It is also called “Bag” due to its mechanism, i.e., it is only concerned with whether known words occur in the document, not the location of the words. Let’s take an example to understand bag-of-words in more detail. Like below, we are taking 2 text documents:

“Neha was angry with Sunil and he was angry on Ramesh.” “Neha love animals.”

Above you see two corpora as documents, we treat both documents as a different entity and make a list of all the words present in both documents except punctuations as here,

“Neha”, “was”, “angry”, “on”, “Sunil”, “and”, “he”, “Ramesh”, “love”, “animals”

Then we create these documents into vectors (or we can say, creating a text into numbers is called vectorization in ML) for further modelling. Presentation of “Neha was angry on Sunil and he was angry on Ramesh” into vector form as [1,1,1,1,1,1,1,0,0] , and the same as in, “Neha love animals” having vector form as [1,0,0,0,0,0,0,0,1,1]. So, the bag-of-words technique is mainly used for featuring generation from text data.

1. Natural Language Generation

Natural language generation (NLG) is a technique that uses raw structured data to convert it into plain English (or any other) language. We also call it data storytelling. This technique is very helpful in many organizations where a large amount of data is used, it converts structured data into natural languages for a better understanding of patterns or detailed insights into any business. This can be viewed opposite of Natural Language Understanding (NLU) that we have already explained above. NLG makes data understandable to all by making reports that are mainly data-driven, like, stock-market and financial reports, meeting memos, reports on product requirements, etc.

There are many stages of any NLG;

1. Content Determination: Deciding what are the main content to be represented in text or information provided in the text.
2. Document Clustering: Deciding the overall structure of the information to convey.
3. Aggregation: Merging of sentences to improve sentence understanding and readability.
4. Lexical Choice: Putting appropriate words to convey the meaning of the sentence more clearly.
5. Referring Expression Generation: Creating references to identify main objects and regions of the text properly.
6. Realization: Creating and optimizing text that should follow all the norms of grammar (like syntax, morphology, orthography).
7. Sentiment Analysis

It is one of the most common natural language processing techniques. With sentiment analysis, we can understand the emotion/feeling of the written text. Sentiment analysis is also known as Emotion AI or Opinion Mining. The basic task of Sentiment analysis is to find whether expressed opinions in any document, sentence, text, social media, reviews are positive, negative, or neutral, it is also called finding the Polarity of Text. Sentiment analysis usually works best on subjective text data rather than objective test data. Generally, objective text data are either statements or facts which does not represent any emotion or feeling. On the other hand, the subjective text is usually written by humans showing emotions and feelings. For example, Twitter is all filled up with sentiments, users are addressing their reactions or expressing their opinions on each topic whichever or wherever possible. So, to access tweets of users in a real-time scenario, there is a powerful python library called “twippy”.

1. Sentence Segmentation

The most fundamental task of this technique is to divide all text into meaningful sentences or phrases. This task involves identifying sentence boundaries between words in text documents. We all know that almost all languages have punctuation marks that are presented at sentence boundaries, so sentence segmentation also referred to as sentence boundary detection, sentence boundary disambiguation or sentence boundary recognition.

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 3 |
| **Title:** | Stop Word Removal |
| **Date of Performance:** | 23/02/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **PostLab(2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

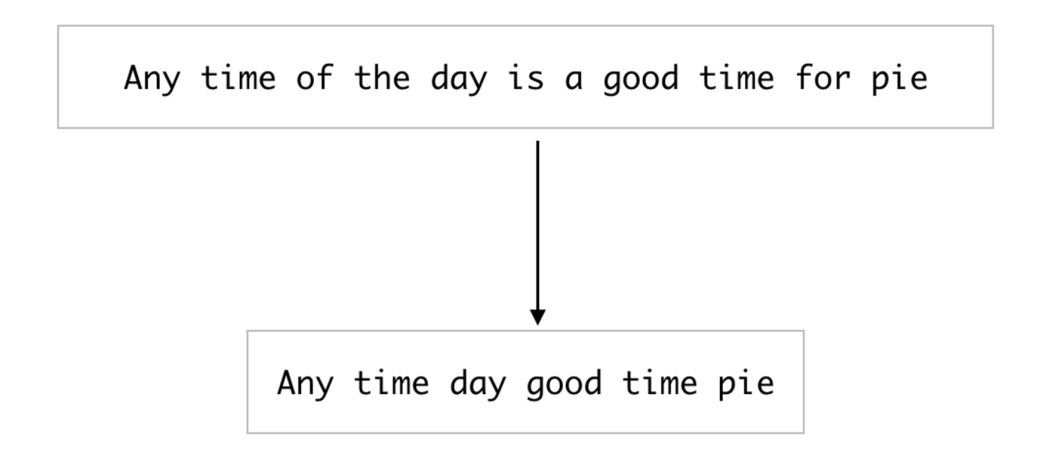
**EXPERIMENT 3 – Stop word removal**

**AIM:** To understand and implement Stop word removal

**THEORY:**

The process of converting data to something a computer can understand is referred to as pre-processing. One of the primary forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words. There is also a corpus of stop words, that is, high-frequency of words like “the, to and also” that we sometimes want to filter out of a document before further processing. In computing, stop words are words that are filtered out before or after the natural language data (text) are processed. While “stop words” typically refers to the most common words in a language, all-natural language processing tools don’t use a single universal list of stop words. Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For some search engines, these are some of the most common, short function words, such as “the”, “a”, “an”, “in” that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. Stop words usually have little lexical content, and their presence in a text fails to distinguish it from other texts

Example:



Different Methods to remove stop words are as follows:

* Using NLTK library:

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language. It contains text processing libraries for tokenization, parsing, classification, stemming, tagging, and semantic reasoning.

* Using SpaCy Library:

spaCy is an open-source software library for advanced natural language processing. spaCy is designed specifically for production use and helps you build applications that process and “understand” large volumes of text. It can be used to build information extraction or natural language understanding systems or to pre-process text for deep learning.

* Using Gensim Library:

Gensim is an open-source library for unsupervised topic modelling and natural language processing, using modern statistical machine learning. Gensim is designed to handle large text collections using data streaming and incremental online algorithms, which differentiates it from most other machine learning software packages that target only in-memory processing. For more detail’s check out [Gensim](https://pypi.org/project/gensim/) documentation. Using Gensim we can directly call remove\_stopwords(), which is a method of gensim.parsing.preprocessing. Next, we need to pass our sentence from which you want to remove stop words, to the remove\_stopwords() method which returns the text string without the stop words. We can then tokenize the returned sentences.

* Custom stop words:

If you feel that the default stop words in any python NLP language tool are too many and are causing loss of information, or are too few to remove all unnecessary words in your corpus, then we can opt for a custom stop words list. For this, you can simply obtain the default stop words to a list and append or delete the required words from the list as per the requirement.

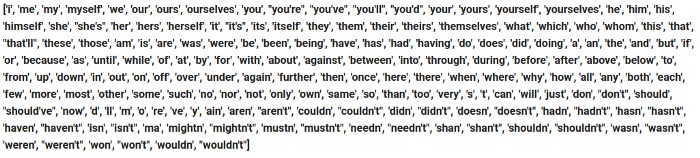
**CODE:**

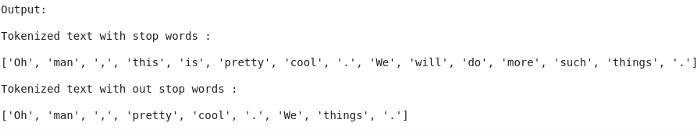
Link to Colab Notebook:

<https://colab.research.google.com/drive/1TKw8h2tWNng5g0zteUB78zbX6_exuwzf?usp=sharing>

|  |
| --- |
| Remove stop words using the NLTK python library  # importing NLTK libarary stopwords import nltk from nltk.corpus import stopwords nltk.download('stopwords') nltk.download('punkt')  from nltk.tokenize import word\_tokenize    print(stopwords.words('english'))  # random sentecnce with lot of stop words  sample\_text = "Oh man, this is pretty cool. We will do more such things." text\_tokens = word\_tokenize(sample\_text)    tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words('english')]    print(text\_tokens) print(tokens\_without\_sw) |

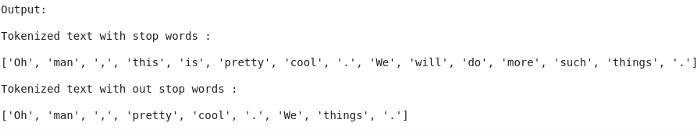
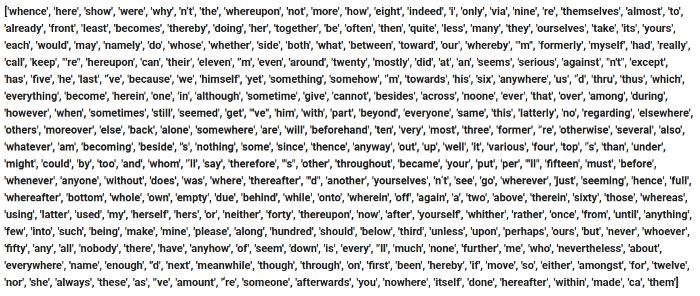
NLTK Stop word list:





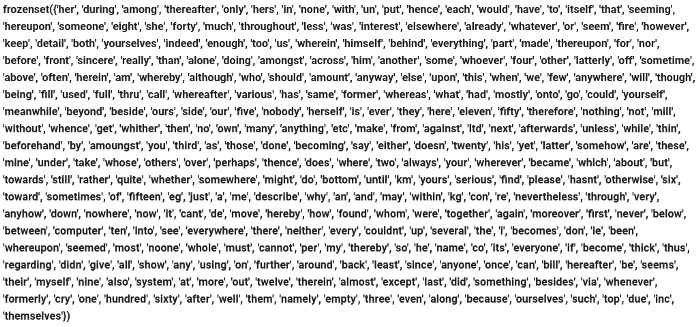
|  |
| --- |
| Remove stop words using spacy library  import spacy  from nltk.tokenize import word\_tokenize # loading english language model of spaCy en\_model = spacy.load('en\_core\_web\_sm')  # gettign the list of default stop words in spaCy english model stopwords = en\_model.Defaults.stop\_words  sample\_text = "Oh man, this is pretty cool. We will do more such things." text\_tokens = word\_tokenize(sample\_text)  tokens\_without\_sw= [word for word in text\_tokens if not word in stopwords]  print(text\_tokens)  print(tokens\_without\_sw) |

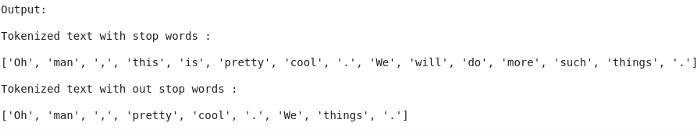
Spacy Stop word list:



|  |
| --- |
| Remove stop words using the Gensim library  from gensim.parsing.preprocessing import remove\_stopwords  sample\_text = "Oh man, this is pretty cool. We will do more such things." sample\_text\_NSW = remove\_stopwords(text)  print(word\_tokenize(sample\_text))  print(word\_tokenize(sample\_text\_NSW)) |

Gensim Stop word list**:**





|  |
| --- |
| Remove Custom stop words  # importing NLTK libarary stopwords import nltk from nltk.corpus import stopwords nltk.download('stopwords')    stopwords\_default = stopwords.words('english') print(len(stopwords\_defaut))    stopwords\_default.append('like')  , 'marvel', 'ghost'])  print(len(stopwords\_default))    # for adding multiple words  stopwords\_default.extend(['marvel', 'ghost']) print(len(stopwords\_default)) |

Custom Stop word list:

['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", 'like', 'marvel', 'ghost']

Output

['Oh', 'man', ',', 'this', 'is', 'pretty', 'cool', '.', 'We', 'will', 'do', 'more', 'such', 'things', 'like', 'this', '.'] ['Oh', 'man', ',', 'pretty', 'cool', '.', 'We', 'things', '.']

**CONCLUSION:**

For tasks like text classification, where the text is to be classified into different categories, stopwords are removed or excluded from the given text so that more focus can be given to those words which define the meaning of the text. If we have a task of text classification or sentiment analysis then we should remove stop words as they do not provide any information to our model, i.e keeping out unwanted words out of our corpus, but if we have the task of language translation then stopwords are useful, as they have to be translated along with other words.

**POST LAB**

**1]** Explain the importance of Stop Word Removal in NLP

* Stop words are often removed from the text before training deep learning and machine learning models since stop words occur in abundance, hence providing little to no unique information that can be used for classification or clustering.
* On removing stopwords, dataset size decreases, and the time to train the model also decreases without a huge impact on the accuracy of the model.
* Stopword removal can potentially help in improving performance, as there are fewer and only significant tokens left. Thus, the classification accuracy could be improved

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 4 |
| **Title:** | Stemming |
| **Date of Performance:** | 23/02/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

**EXPERIMENT 4 – Stemming**

**AIM:** To understand and implement Stemming

**THEORY:**

Stemming refers to the idea of reducing the different forms of words to a core root. Words that are derived from one another can be mapped to a central word or symbol especially if they have the same core meaning. For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

For instance:

am, are, is => be car, cars, car's, cars => car

The result of this mapping of text will be something like:

the boy's cars are different colours => the boy car be a different colour

However, the two words differ in their flavour. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Stemming Algorithms:

1. Porter Stemmer: The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is Porter's algorithm. The entire algorithm is too long and intricate to present here, but we will indicate its general nature.

Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase, there are various conventions to select rules, such as selecting the rule from each ruling group that applies to the longest suffix.

In the first phase, this convention is used with the following rule group:



1. Snowball Stemmer: It is also known as Porter2 stemming algorithm. It is almost universally accepted as better than Porter Stemmer as is even acknowledged by the individual who created Porter Stemmer. With this being said it is more aggressive. A lot of things were added new to the Snowball stemmer because of the issues noticed with the Porter Stemmer. One advantage of the snowball stemmer is that it is computationally faster.
2. Lancaster Stemmer: It is the most aggressive stemming algorithm from the 3. It is also computationally faster compared to the rest. However, one complaint around this stemming algorithm is that it is overly aggressive and sometimes transforms words into strange stems. It is not the most suitable tool when distinction and clarity are required in the application.

Two main issues with Stemming

1. Over Stemming:

Over-stemming is when two words with different stems are stemmed to the same root. This is also known as a false positive.

universal, university, universe

All the above 3 words are stemmed from univers which is wrong behaviour.

1. Under Stemming

Under-stemming is when two words that should be stemmed to the same root are not. This is also known as a false negative. Below is an example of the same.

alumnus, alumni, alumnae

**CODE:**

Link to the collab notebook:

<https://colab.research.google.com/drive/1nRSIIToycWgjZ8oe7I8Ow8_v9lGFA9yK?usp=sharing>

|  |
| --- |
| import nltk nltk.download('punkt')  from nltk.tokenize import sent\_tokenize, word\_tokenize  # Porter, Snowball and Lancaster Stemmer  from nltk.stem import PorterStemmer  from nltk.stem import SnowballStemmer  from nltk.stem import LancasterStemmer  example\_words = ['python', 'pythoner', 'pythoning', 'pythoned', 'pythonly', 'fairly', 'destablize', 'friendships', 'anxious'] |
| # Porter Stemmer  ps = PorterStemmer()  for word in example\_words:  print(ps.stem(word))  """ Output:  python  python  python  python  pythonli  fairli  destabl  friendship  anxiou""" |
| # Snowball Stemmer  sb = SnowballStemmer(language='english')  for word in example\_words:  print(sb.stem(word))    """ Output:  python  python  python  python  python  fair  destabl  friendship  anxious """ |
| # Lancaster Stemmer  ls = LancasterStemmer()  for word in example\_words:  print(ls.stem(word))  """ Output:  python  python  python  python  python  fair  dest  friend  anxy  """    # Overstemming words = ['universal', 'university', 'universe']  for word in words:  print(ps.stem(word))    """ Output:  univers  univers  univers  """    # Understemming words = ['alumnus', 'alumni', 'alumnae'] for word in words:  print(ps.stem(word))    """ Output:  alumnu  alumni  alumna  """ |

**CONCLUSION:**

* Porter stemmer algorithm is the simplest of them all. However, it does not cover all conditions leaving room for improvement.
* Snowball stemmer algorithm is a much detailed algorithm covering the issues present in the Porter algorithm. It is also computationally faster.
* Finally, the Lancaster algorithm reduces the words drastically to their stem and may not be a suitable algorithm for applications that require clarity and distinction.

**POST LAB:**

1. What is Stemming?

Ans: -

* 1. Stemming is the process of reducing the inflected word to its original word
  2. Some stemmers don't have meaning, yet it is faster than Lemmatization
  3. Example
     + - 1. history, historical => histori
         2. finally, final, finalized => fina
         3. automation, automatic => automat
  4. Stemming is a crude heuristic process that chops off the ends of words in the hope to achieve this goal correctly most of the time.
  5. Search engines use stemming for indexing words. Instead of storing all forms of the words, search engines only store the stems
  6. Stemming algorithms are rule-based. And therefore, can result in non-sensical stems where the meanings of the word are lost.
  7. Example: data and datum => dat and datu which makes no sense.
  8. Porter Stemmer
  9. Porter Algorithm consists of 5 phases of word reductions applied sequentially
     1. Rule Example

sses => ss caresses => caress ies => I ponies => poni ss => ss caress => caress s => cats => cat

* + 1. Many of later rules use a concept of measure of word
    2. It loosely checks the number of syllables to see whether the word is long enough that is reasonable to regard the matching portion of a rule as a suffix rather than as a part of the stem
    3. (m>1) ement => would map replacement to replac but not cement to c
    4. However, it is possible that we would lose considerable precision on queries
       - 1. Operational and research
         2. Operating and system
         3. Operative and dentistry

1. Explain OverStemming and Understemming.

Ans: -

* 1. Overstemming:
     1. Here too much from the word is cut off
     2. This results in nonsensical stems, where all the meaning of the word is lost or muddled. iii. It can also resolve the words to the same stem even though they probably should not.

For example, university, universal, universities, universe stem to “univers”. While it might be nice to have universal and universe stemmed together and university and universities should stem together not all 4 together. But the enforcing rules result in these issues.

* 1. Understemming:
     1. It faces the opposite issue.
     2. Here, we have several words that are forms of one another.
     3. They should all resolve to the same stem, but unfortunately, they do not.

For example, words like data and datum so stem from “data” but they stem to dat and datu respectively.

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 5 |
| **Title:** | Morphology |
| **Date of Performance:** | 23/02/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

**EXPERIMENT 5: MORPHOLOGY**

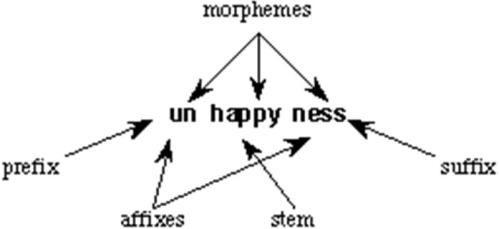
**AIM:** Write a program to perform Morphological analysis

**THEORY:**

Morphology is the study of the ways words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is the smallest meaningful linguistic unit. Morphemes are considered the smallest meaningful units of language. These morphemes can either be a root word(play) or affix(ed). The combination of these morphemes is called the morphological process. So, the word “played” is made out of two morphemes: “play” and “-ed”. Thus, finding all parts of a word(morphemes) and thus describing properties of a word is called “Morphological Analysis”. For example, “played” has information verb “play” and “past tense”, so the given word is past tense form of the verb “play”

Morphology is the study of the structure and formation of words. Its most important unit is the *morpheme*, which is defined as the "minimal unit of meaning". (Linguistics textbooks usually define it slightly differently as "the minimal unit of grammatical analysis".) Consider a word like:

"unhappiness". This has three parts:



There are three morphemes, each carrying a certain amount of meaning. *un* means "not", while *ness* means "being in a state or condition". *Happy* is a *free morpheme* because it can appear on its own (as a "word" in its own right). *Bound morphemes* have to be attached to a free morpheme, and so cannot be words in their own right. Thus you can't have sentences in English such as "Jason feels very un ness today".

Armed with these definitions, we can look at ways used to classify languages according to their morphological structure.

# Morphological analysis

This section has three parts. In the first part, some basic terms in morphology are introduced, in particular, *morpheme*, affix, *prefix*, *suffix*, *bound and free* forms. The second reviews conventional ways of grouping languages, such as *isolating*, *agglutinative* and *inflecting*. The final section looks at some morphological processes, concentrating only on those of greater relevance to natural language engineering.

Analysis of a word :

ब€ो ो (bachchoM) = ब€ो (bachchaa)(root) + ओ

(oM)(suffix) (ओो =3 plural oblique)

A linguistic paradigm is the complete set of variants of a given lexeme. These variants can be classified according to shared inflectional categories (eg: number, case etc) and arranged into tables.

A paradigm for ब ा

|  |  |  |
| --- | --- | --- |
| case/num | singular | plural |
| direct | ब€ो (bachcha  a) | ब€ो (bachche) |
| oblique | ब€ो (bachche  ) | ब€ो ो  (bachchoM) |

Algorithmto get ब ा (ा bachchoM) from ब ा (bachchaa)

1. Take Root ब€(bachch)आ(aa)
2. Delete आ(aa)
3. output ब€(bachch)
4. Add ओ (oM) to output
5. Return ब€ो ो (bachchoM)

Therefore आ is deleted and ओ is added to get ब€ो ो

Add-Delete table for ब ा

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Delete | Add | Nubmer | Case | Variants |
| आ(aa) | आ(aa) | sing | dr | ब€ो (bachchaa) |
| आ(aa) | ए(e) | plu | dr | ब€ो (bachche) |
| आ(aa) | ए(e) | sing | ob | ब€ो (bachche) |
| आ(aa) | ो ओ(oM  ) | plu | ob | ब€ो (ो bachchoM) |

Paradigm Class Words in the same paradigm class behave similarly, for Example

लड़क is in the same paradigm

class as ब€, so लड़क would behave similarly as ब€ो as they share the same paradigm class

**CODE:**

<https://colab.research.google.com/drive/1iHyXlWdBKmtrWd7irVNiiyRW6DWkqwwT>

# -\*- coding: utf-8 -\*-

"""Exp5- Morphology .ipynb - """

!pip install polyglot

!pip install PyICU

!pip install pycld2 !pip install morfessor

import polyglot from polyglot.text import Text, Word

"""Downloading Required models"""

# Commented out IPython magic to ensure Python compatibility.

# %%bash

# polyglot download morph2.en morph2.ar

word = Text("preprocessor").words[0] print(word.morphemes)

from polyglot.downloader import downloader print(downloader.supported\_languages\_table("morph2"))

print("words:-")

print("1-लड़क ; 2-ग ली ; 3-बच् च ") i= int(input("enter your word choice ")) dele=[] addi=[] for j in range(0,4):

print("enter what you want to delete") d=int(input("1-आ ; 2-ई :- ")) dele.append(d)

print("enter what you want to add")

a=int(input("1-आ ;2-ओ ;3-आयें ;4-इय ाँ ;5-ई ;6-इय ;7-ए :-

")) addi.append(a) if i == 1 or 3 :

if dele ==[1, 1, 1, 1] and addi ==[1, 7, 7, 2] : print("correct")

else : print("correct answer is :-" ) आ - आ\nआ - ए\nआ - ए\nआ - ओ \n ")

ई - ई\nई - इय ाँ\nई - ई\nई - इय \n ")

**CONCLUSION:**

Thus we have successfully studied and implemented Morphology

**POST LAB:**

1] What is Morphology?

Ans: -

Morphology is the study of the structure and formation of words. Its most important unit is the morpheme, which is defined as the "minimal unit of meaning". (Linguistics textbooks usually define it slightly differently as "the minimal unit of grammatical analysis".) The term morphology is Greek and is a makeup of morph- meaning ‘shape, form’, and - ology which means ‘the study of something’. Morphology as a sub-discipline of linguistics was named for the first time in 1859 by the German linguist August Schleicher who used the term for the study of the form of words.

2) What are Morphological Processes?

Ans: -

Morphology is the study of the rules governing the formation of words.' Morphological processes can be by affixation or other words formation. Affixation can be inflexion or derivation while in other words, the formation can be compound, reduplication, suppletion, internal change, clipping, conversion.

A. AFFIXATION

Affixation is the process in which a free morpheme (root) is added with bound morphemes (affixes). There are two kinds of affixation, they are inflexion and derivation.

I. INFLECTION

Inflection is a word-formation process that changes the morphological form of a word to fit a syntactic context. Example:

walk vs. walked

cat vs. cats

There are some characteristics of inflexion: inflexion does not change the grammatical category of the base; inflexion

* does not affect the meaning of the word;
* inflectional processes take place after derivational ones;

Example neighbourhoods vs. \*neighbourhood inflectional affixes have few exceptions (they are almost fully productive), while

* derivational affixes usually attach to a limited class of words;
* English inflectional affixes are all suffixes.

Example

plural -s: cat - cats

possessive/genitive ’s: John’s

3rd person sg. non-past -s: sing-sings

progressive -ing: sing-singing

past tense -ed: talk-talked

past participle -en/-ed: eat-eaten/study-studied

comparative -er: happy-happier

superlative -est: happy-happiest

There are two types of inflexion. They are: regular inflexion = rule-based; walk-walked

* Irregular inflection = stored in the lexicon; come-came;goose-geese
* Evidence for distinction for irregular verbs, response time is linked to the frequency of the verb
* for regular verb, no such difference is found since the past tense is formed by a regular rule

## II. DERIVATION

Derivational affixes are affixes (suffixes) which change the meaning of the base in some important ways or change it into a different word class. They turn nouns into adjectives, adjectives into verbs, nouns of one type into nouns to the other type, and so on. They add new meanings to the base. They are readily followed by inflectional suffixes, and in many cases, more than one derivational suffix can be found in some word. For instance, let us start with the verb Establish in its rather specialized meaning of ‘grant special state privileges to a church’. We can derive the verb disestablish, meaning ‘take away special privileges. Then we can form the noun disestablishment meaning ‘the act of taking away privileges’, then the noun disestablishmentarian meaning ‘one who advocates disestablishment’, then the noun disestablishmentarianism meaning ‘the doctrine of disestablishment’, and finally antidisestablishmentarianism, meaning ‘opposite to the disestablishing the church. The latter word is often cited as ‘the longest word in English Language’ Brockman

## B. OTHER WORD FORMATION

## Compounding

A compound word contains at least two bases which are both words, or at any rate , root morphemes. examples: -

n+n))(Tea) +( pot ) => teapot

Hair) + (dress) + er => hairdresser(n+v) Blue) + (bird)

=> bluebird (a+n)

Over) + (lord) => overlord (pre+n)

## Conversion

Conversion is a process that assigns an already existing word to a new syntactic category.

Examples: -

=>V derived from n

e.g button (the shirt)

=>N

deri ved

fro

m v

(a

long

)

wal k

=>V derived

from A

Open (a door)

* Clipping

Clipping is a process that shortens a polysyllabic word by deleting one or more syllables. It is especially popular among students. Examples: -

Prof => for professor

Poli – sci => for political science

Zoo for=> zoological garden

## Blends

Blends are words created from non-morphemic parts of two already existing items. A blend is usually formed from the first part of one word and the final part of the second one.

Examples: -

brunch=>from breakfast and lunch

Smog=>from smoke and fog.

Spam => from spiced and ham.

## Internal change

Internal change is a process that substitutes' one non-morphemic segment for another.

Examples: -

sing(present) =>sang(past)

Sink(present) =>sank(past)

Foot (singular) => feet(plural)

Goose(singular) => geese(plural)

## Suppletion

Suppletion is a morphological process whereby a root morpheme is replaced by a phonologically unrelated form to indicate a grammatical contrast.

Examples: -

have => had

Go => went

good= > better

## Acronym

They are formed from the initial letters of a set of other words. They are usually pronounced as single words *(e.g. NATO, PIN, etc.)* Or as a set of letters *(e.g. CD, VIP, etc.)*

## Back Formation

A word of one type (usually a noun) is reduced to a word of a different type (usually a verb) through widespread use.

* to donate from donation
* to opt from option
* Other examples: pronunciation (< pronunciation), resurrect (< resurrection), enthuse (< enthusiasm).

## Borrowing

Taking over words from other languages.

Examples from Italian

*pasta*

*piano*

## Coinage

Coinage is the invention of totally new terms. Often a brand name becomes the name for the item or process associated with the brand name

Examples:

1. hoover
2. Kleenex
3. Xerox
4. Kodak

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 6 |
| **Title:** | POS Tagging |
| **Date of Performance:** | 10/04/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

**EXPERIMENT 6 – POS Tagging**

**AIM:** Write a program to perform POS Tagging

**THEORY:**

POS Tagging or Part-of-speech Tagging is the procedure of assigning a grammar category like noun, verb, adjective etc. to a word. In this process, both the information and context play an important role as the same lexical form can be used differently in a different context. Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms that associate discrete terms, as well as hidden parts of speech, under a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic.

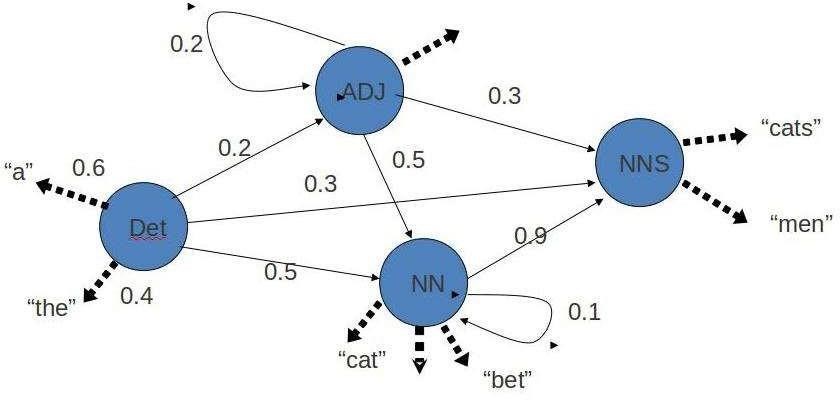
Types of POS taggers:

POS-Tagging algorithms fall into two distinct categories:

1. Rule-based POS Taggers
2. Stochastic POS Taggers

POS tagging or part-of-speech tagging is the procedure of assigning a grammatical category like noun, verb, adjective etc. to a word. In this process, both the lexical information and the context play an important role as the same lexical form can behave differently in a different context. For example, the word "Park" can have two different lexical categories based on the context.

1. The boy is playing in the park. ('Park' is Noun)
2. Park the car. ('Park' is Verb)



|  |  |
| --- | --- |
| **Abbreviation** | **Meaning** |
| CC | coordinating conjunction |
| CD | cardinal digit |
| DT | determiner |
| EX | existential there |
| FW | foreign word |
| IN | preposition/subordinating conjunction |
| JJ | This NLTK POS Tag is an adjective (large) |
| JJR | adjective, comparative (larger) |
| JJS | adjective, superlative (largest) |
| LS | list market |
| MD | modal (could, will) |
| NN | noun, singular (cat, tree) |
| NNS | noun plural (desks) |
| NNP | a proper noun, singular (Sarah) |
| NNPS | a proper noun, plural (Indians or Americans) |
| PDT | predeterminer (all, both, half) |
| POS | possessive ending (parent\ 's) |
| PRP | personal pronoun (hers, herself, him, himself) |
| PRP$ | possessive pronoun (her, his, mine, my, our ) |
| RB | adverb (occasionally, swiftly) |
| RBR | adverb, comparative (greater) |
| RBS | adverb, superlative (biggest) |

Assigning part of speech to words by hand is a common exercise one can find in an elementary grammar class. But here we wish to build an automated tool that can assign the appropriate part-of-speech tag to the words of a given sentence. One can think of creating handcrafted rules by observing patterns in the language, but this would limit the system's performance to the quality and number of patterns identified by the rule crafter. Thus, this approach is not practically adopted for building POS Taggers. Instead, a large corpus annotated with correct POS tags for each word is given to the computer and algorithms then learn the patterns automatically from the data and store them in the form of a trained model. Later this model can be used to POS tag new sentences.

Pos-tag abbreviation vs their meaning is given below.

|  |  |
| --- | --- |
| TO | infinite marker (to) |
| UH | interjection (goodbye) |
| VB | verb (ask) |
| VBG | verb gerund (judging) |
| VBD | verb past tense (pleaded) |
| VBN | verb past participle (reunified) |
| VBP | verb, present tense not 3rd person singular(wrap) |
| VBZ | verb, present tense with 3rd person singular (bases) |
| WDT | wh-determiner (that, what) |
| WP | wh- pronoun (who) |
| WRB | wh- adverb (how) |

**CODE:**

Link to colab notebook : https://colab.research.google.com/drive/1F5cCwOdjZjIoUOoKyf6xhdrhFi-s1YE#scrollTo=AoM5taAiTXPt

|  |
| --- |
| Pos tagging using Spacy:    import spacy  sp = spacy.load('en\_core\_web\_sm')  sen = sp(u"Park the car.")  for word in sen:  print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}} {spacy.expla in(word.tag\_)}')    Output :  Park VERB VB verb, base form the DET DT determiner  car NOUN NN noun, singular or mass  . PUNCT . punctuation mark, sentence closer |
| Pos Tagging using NLTK:    import nltk  from nltk import pos\_tag  from nltk import RegexpParser  text ="Park the car".split()  print("After Split:",text) t  okens\_tag = pos\_tag(text)  print("After Token:",tokens\_tag)  patterns= """mychunk:{<NN.?>\*<VBD.?>\*<JJ.?>\*<CC>?}""" chunker = RegexpParser(patterns) print("After Regex:",chunker) output = chunker.parse(tokens\_tag)  print("After Chunking",output) |
| Output :  After Split: ['Park', 'the', 'car']  After Token: [('Park', 'NNP'), ('the', 'DT'), ('car', 'NN')] After Regex: chunk.RegexpParser with 1 stages:  RegexpChunkParser with 1 rules:  <ChunkRule: '<NN.?>\*<VBD.?>\*<JJ.?>\*<CC>?'>  After Chunking (S (mychunk Park/NNP) the/DT (mychunk car/NN)) |

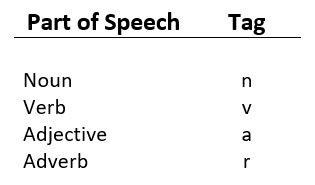
**POST LAB:**

1. What is Default Tagging in POS Tagging?

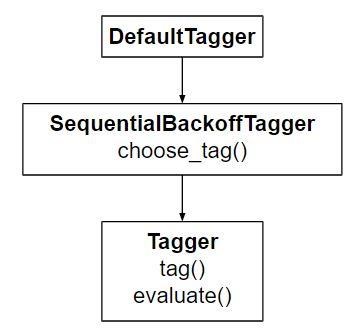
Ans: -

1)What is default tagging in pos tagging?

It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form *(word, tag)*). The tag in case of is a part-of-speech tag and signifies whether the word is a noun, adjective, verb, and so on.



Default tagging is a basic step for part-of-speech tagging. It is performed using the DefaultTagger class. The DefaultTagger class takes ‘tag’ as a single argument. NN is the tag for a singular noun. DefaultTagger is most useful when it gets to work with the most common part-of-speech tag. that’s why a noun tag is recommended.



2] Different pos tagging techniques?

Ans: -

1. Rule-based POS Tagging

One of the oldest techniques of tagging is rule-based POS tagging. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag. As the name suggests, all such kind of information in rule-based POS tagging is coded in the form of rules. These rules may be either −

* + Context-pattern rules
  + Or, as Regular expression compiled into finite-state automata, intersected with lexically ambiguous sentence representation.

We can also understand Rule-based POS tagging by its two-stage architecture −

* + The first stage − In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.
  + The second stage − In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

1. Stochastic POS Tagging

The model that includes frequency or probability (statistics) can be called stochastic.

* + Word Frequency Approach

In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield an inadmissible sequence of tags.

* + Tag Sequence Probabilities

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called the n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 7 |
| **Title:** | Chunking |
| **Date of Performance:** | 10/04/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

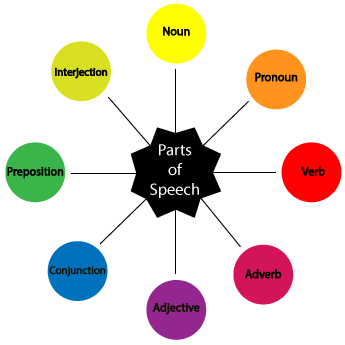
Signature of the Teacher :

**EXPERIMENT 7 – Chunking**

**AIM:**  To understand and implement Chunking

**THEORY:**

Chunking is a process of extracting phrases from unstructured text. Instead of just simple tokens which may not represent the actual meaning of the text, it’s advisable to use phrases such as “South Africa” as a single word instead of ‘South’ and ‘Africa’ separate words. Chunking works on top of POS tagging, it uses pos-tags as input and provides chunks as output. Similar to POS tags, there are a standard set of Chunk tags like Noun Phrase (NP), Verb Phrase (VP), etc. Chunking is very important when you want to extract information from text such as Locations, Person Names etc. In NLP called Named Entity Extraction. The part of speech explains how a word is used in a sentence. There are eight main parts of speech - nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions and interjections.



Noun (N)- Daniel, London, table, dog, teacher, pen, city, happiness, hope

Verb (V)- go, speak, run, eat, play, live, walk, have, like, are, is

Adjective (Adj)- big, happy, green, young, fun, crazy, three

Adverb (Adv)- slowly, quietly, very, always, never, too, well, tomorrow

Preposition (P)- at, on, in, from, with, near, between, about, under

Conjunction (CON)- and, or, but, because, so, yet, unless, since, if

Pronoun (PRO)- I, you, we, they, he, she, it, me, us, them, him, her, this

Interjection (INT)- Ouch! Wow! Great! Help! Oh! Hey! Hi!

**CODE:**

import nltk

text = "the little yellow dog barked at the cat"

tokens = nltk.word\_tokenize(text)

print("Tokens: "+str(tokens))

tag = nltk.pos\_tag(tokens)

print("POS Tags: "+str(tag))

grammar = "NP: {<DT>?<JJ>\*<NN>}"

cp =nltk.RegexpParser(grammar)

result = cp.parse(tag)

print("------Chunks-----")

print(result)

result.draw()

Output:

Tokens: ['the', 'little', 'yellow', 'dog', 'barked', 'at', 'the', 'cat']

POS Tags: [('the', 'DT'), ('little', 'JJ'), ('yellow', 'JJ'), ('dog', 'NN'), ('barked', 'VBD'), ('at', 'IN'), ('the', 'DT'), ('cat', 'NN')]

------Chunks-----

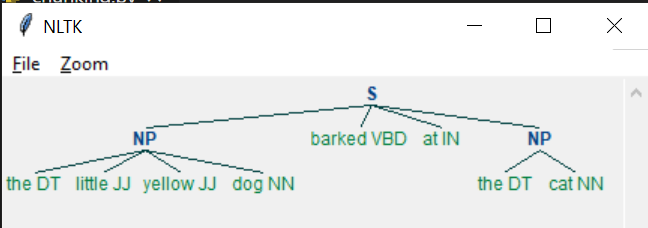
(S

(NP the/DT little/JJ yellow/JJ dog/NN)

barked/VBD

at/IN

(NP the/DT cat/NN))

****

**CONCLUSION:**

Rather than making use of simple tokens which may not represent the actual meaning of the text, chunking focuses on extracting phrases from the text. The experiment enables us to understand the importance of chunking. Chunking is very important when we wish to extract information such as the name of the location, or a person’s name, date etc.

**POST LAB:**

1] What is Chunking?

Ans: -

1. Chunking is the process of extracting phrases from the unstructured text.
2. It is also known as Partial or Shallow parsing.
3. Chunking is based on POS tagging. It accepts POS tags as the input and generates chunks as the output.
4. Rather than using simple tokens which may not represent the actual meaning of the text, chunking focuses on extracting phrases from the text. Eg: The term “South Korea” will be treated as a single word instead of two separate words “South” and “Korea”.
5. The chunks to be extracted are defined using Regex along with POS Tags.
6. Chunking is important when we wish to extract useful information from the text such as the name of the location, or a person’s name, date etc.

2] Apply Chunking to the string “The Book has many chapters”. Explain with neatly labelled trees.

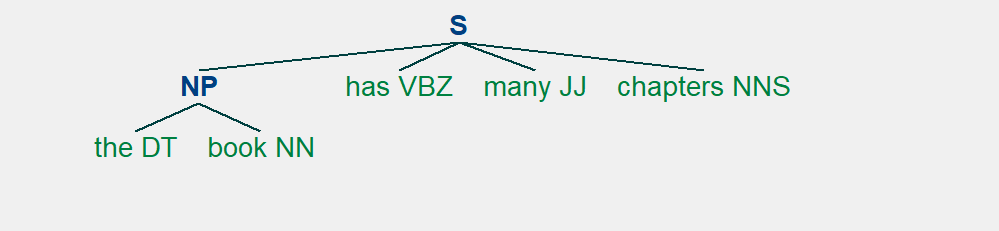
Statement: “The book has many chapters”

Ans: -

Grammar: ('''NP: {<DT>?<JJ>\*<NN>} ''')

POS Tags: [('the', 'DT'), ('book', 'NN'), ('has', 'VBZ'), ('many', 'JJ'), ('chapters', 'NNS')]

Tree:



**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 8 |
| **Title:** | N-Gram Model |
| **Date of Performance:** | 9/04/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **Post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

**EXPERIMENT 8 – N- GRAM MODEL**

**AIM:** To understand and implement N- GRAM MODEL

**THEORY:**

N-GRAMS: The probability of a sentence can be calculated by the probability of a sequence of words occurring in it. We can use the Markov assumption, that the probability of a word in a sentence depends on the probability of the word occurring just before it. Such a model is called the first-order Markov model or the bigram model.



Here, Wn refers to the word token corresponding to the nth word in a sequence.

A combination of words forms a sentence. However, such a formation is meaningful only when the words are arranged in some order.

E.g.: Sit I car in the

Such a sentence is not grammatically acceptable. However, some perfectly grammatical sentences can be nonsensical too!

e.g.: Colorless green ideas sleep furiously

One easy way to handle such unacceptable sentences is by assigning probabilities to the strings of words i.e., how likely the sentence is.

# Probability of a sentence

If we consider each word occurring in its correct location as an independent event, the probability of the sentences is : P(w(1), w(2)..., w(n-1), w(n))

Using chain rule:

=P(w(1)) \* P(w(2) | w(1)) \* P(w(3) | w(1)w(2)) ... P(w(n) | w(1)w(2)…w(n-1))

Bigrams

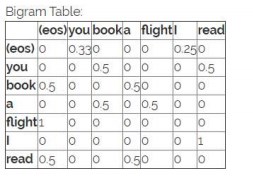
We can avoid this very long calculation by approximating that the probability of a given word depends only on the probability of its previous words. This assumption is called the Markov assumption and such a model is called the Markov model- bigrams. Bigrams can be generalized to the n-gram which looks at (n-1) words in the past. A bigram is a first-order Markov model. Therefore,

P(w(1), w(2)..., w(n-1), w(n))= P(w(2)|w(1)) P(w(3)|w(2)) …. P(w(n)|w(n-1))

We use (eos) tag to mark the beginning and end of a sentence.

A bigram table for a given corpus can be generated and used as a lookup table for calculating the probability of sentences.

Eg: Corpus – (eos) You book a flight (eos) I read a book (eos) You read (eos)



P((eos) you read a book (eos))

= P(you|eos) \* P(read|you) \* P(a|read) \* P(book|a) \* P(eos|book)

= 0.33 \* 0.5 \* 0.5 \* 0.5 \* 0.5

**CODE:**

|  |
| --- |
| import re  from nltk.util import ngrams  s = "Natural-language processing (NLP) is an area of computer science " \ "and artificial intelligence  concerned with the interactions " \ "between computers and human (natural) languages."  s = s.lower() s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)  tokens = [token for token in s.split(" ") if token != ""] output = list(ngrams(tokens, 5)) print( output )  Output:  [('natural', 'language', 'processing', 'nlp', 'is'),  ('language', 'processing', 'nlp', 'is', 'an'),  ('processing', 'nlp', 'is', 'an', 'area'),  ('nlp', 'is', 'an', 'area', 'of'),  ('is', 'an', 'area', 'of', 'computer'),  ('an', 'area', 'of', 'computer', 'science'),  ('area', 'of', 'computer', 'science', 'and'),  ('of', 'computer', 'science', 'and', 'artificial'),  ('computer', 'science', 'and', 'artificial', 'intelligence'),  ('science', 'and', 'artificial', 'intelligence', 'concerned'),  ('and', 'artificial', 'intelligence', 'concerned', 'with'),  ('artificial', 'intelligence', 'concerned', 'with', 'the'),  \('intelligence', 'concerned', 'with', 'the', 'interactions'),  ('concerned', 'with', 'the', 'interactions', 'between'),  ('with', 'the', 'interactions', 'between', 'computers'),  ('the', 'interactions', 'between', 'computers', 'and'),  ('interactions', 'between', 'computers', 'and', 'human'),  ('between', 'computers', 'and', 'human', 'natural'), ('computers', 'and', 'human', 'natural', 'languages')] |

**CONCLUSION:**

N-gram language models can be used to offer predictions, suggestions, and corrections. There are 3 types of N-gram models

i) Unigram ii) Bigram iii) Trigram

The accuracy and feasibility of an N-gram based approach depend on the task we would like to achieve. But for most tasks, we can get away with ignoring the long-distance dependencies and using N-grams.

**POST LAB:**

1] Explain the N-gram model with an example.

Ans: -

Language modelling i.e., predicting the probability of a word in a sentence, is a fundamental task in natural language processing. This is achieved by using N-grams. These models assign probabilities to sequences of words. In natural language processing, an n-gram is a sequence of n words. The essential idea behind chunking and n-grams is that similar words tend to follow each other. It is used in many NLP applications such as autocomplete, spelling correction, or text generation. An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. If we have a good N-gram model, we can predict p(w | h) – what is the probability of seeing the word *w* given a history of previous words *h* – where the history contains n-1 words.

There are 3 types of N-gram models

i) Unigram - only consider the current word to find the probability of next word ii) Bigram - considers a group of 2 preceding words to find the next word iii) Trigram - considers a group of 3 preceding words to find the next word

Examples of outputs generated by an N-gram model are as follows

Months the May and issue of year foreign new exchange's September were recession exchange new endorsed a acquire to six executives

## ii) Bigram -

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five per cent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

## iii) Trigram -

They also point to ninety-nine point six billion dollars from two hundred four-oh six three per cent of the rates of interest stores as Mexico and Brazil on market conditions as can be seen, the higher the N the better is the model usually. But this leads to lots of computation overhead that requires large computation power in terms of RAM. So, it won’t always be feasible.

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

# Rubrics for Lab Experiments

|  |  |  |
| --- | --- | --- |
| **Class** | **: B*.E. Computer*** | **Subject Name: NLP** |
| **Semester** | **: VIII** | **Subject Code: CSL*804*** |

|  |  |
| --- | --- |
| **Practical No:** | 9 |
| **Title:** | Regular Expressions |
| **Date of Performance:** | 17/02/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Efforts (3)** | N/A | N/A | Not Completed (1) | Partially Completed (2) | Completed (3) |
| **Legibility (3)** | N/A | N/A | Poor (1) | Good (2) | Very Good (3) |
| **post Lab (2)** | N/A | N/A | N/A | Partially Correct (1) | All Correct (2) |

Total Marks :

Signature of the Teacher :

**Experiment 9: Regular Expressions**

**AIM:** The objective of this experiment is to understand the concept of Regular Expressions and get familiar with commonly used Regular Expressions.

**THEORY:** Regular Expressions

Regular Expressions are used to denote regular languages. An expression is regular if:

* ɸ is a regular expression for regular language ɸ.
* ɛ is a regular expression for regular language {ɛ}.
* If a ∈ Σ (Σ represents the input alphabet), a is a regular expression with language {a}.
* If a and b are regular expressions, a + b is also a regular expression with language {a,b}.
* If a and b are regular expressions, ab (concatenation of a and b) is also regular.
* If a is a regular expression, a\* (0 or more times a) is also regular.

Some RE Examples:

|  |  |
| --- | --- |
| Regular Expressions | Regular Set |
| (0 + 10\*) | L = {0, 1, 10, 100, 1000, 10000, …} |
| (0\*10\*) | L = {1, 01, 10, 010, 0010, …} |
| (0 + ε) (1 + ε) | L = {ε, 0, 1, 01} |
| (a+b) \* | Set of strings of a’s and b’s of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa…….} |
| (a+b) \*abb | Set of strings of a’s and b’s ending with the string abb. So, L = {abb, aabb, babb, aaabb, ababb, ………….} |
| (11) \* | Set consisting of even number of 1’s including empty string, So L= {ε, 11, 1111, 111111, ……….} |
| (aa)\*(bb)\*b | Set of strings consisting of even number of a’s followed by odd number of b’s, so L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, …………...} |
| (aa + ab + ba + bb) \* | String of a’s and b’s of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so L = {aa, ab, ba, bb, aaab, aaba, ………….} |

Advantages of using Regular Expressions: -

* Wide range of usage possibility, you may create one regular expression to validate any kind of input.
* Supported by almost any language, there are only a few programming languages which do not understand regular expressions.
* Do more with less, keep your code cleaner.
* Faster validations, instead of having many IF and ELSE operators you may validate only once with a regular expression.

**CODE:**

Link to Github Repository:<https://github.com/barbozadevin/regular-expressions>

|  |
| --- |
| #Match any alphanumeric text  import re  text = "This is a demo for regular expressions in python"  result = re.match(".\*", text)  print(result) |
| #Check if text starts with an alphabetic character  text = "This is a demo for regular expressions in python"  result = re.match(r"[a-zA-z]+", text)  print(result) |
| #Check if text starts with an alphabetic character  text = "2021 This is a demo for regular expressions in python"  result = re.match("[a-zA-z]+", text)  print(result) |
| #Check if text starts with “2021”  result = re.search("^2021", text)  type(result) |
| #Find Numeric characters  text = "The Corona virus has caused a pandemic since 2020 and is still around in 2021"  result = re.search("\d+", text)  print(result.group(0)) |
| #Substitute “2020” with “2021”  text = "2020 This is a demo for regular expressions in python"  result = re.sub("2020", "2021", text)  print(result) |
| #Remove stray symbols  text = "The is, '@a demo' for ? regular \_ expressions % $ in python."  result = re.sub(r"[,@\'?\.$%\_]", "", text, flags=re.I)  print(result) |
| #Remove stray spaces  text = "This is a demo for regular expressions in python"  result = re.sub(r"\s+"," ", text, flags = re.I)  print(result) |
| #Matches all occurrences of decimal digits  result = re.findall("\d+", text)  print(result) |
| #Creating a pattern and using it to check for an email id  pattern = re.compile("^[a-zA-Z0-9+.-\_]+@[a-zA-Z0-9+.-\_]+$")  result = pattern.search("abc\_c@gmail.com")  print(result) |

**CONCLUSION:**

Thus, we have successfully studied and implemented regular expressions.

**POST LAB:**

Q 1. Which one of the following languages over the alphabet {0,1} is described by the regular expression?

(0+1) \*0(0+1) \*0(0+1) \*

(A) The set of all strings containing the substring 00.

(B) The set of all strings containing at most two 0’s.

(C) The set of all strings containing at least two 0’s.

(D) The set of all strings that begin and end with either 0 or 1.

Ans: -

(C) The set of all strings containing at least two 0’s.

Q 2. Which of the following languages is generated by given grammar?

S -> aS | bS | ∊

(A) {an bm | n,m ≥ 0}

(B) {w ∈ {a,b}\* | w has equal number of a’s and b’s}

(C) {an | n ≥ 0} ∪ {bn | n ≥ 0} ∪ {an bn | n ≥ 0}

Ans: -

(D) {a,b}\*

Q 3. The regular expression 0\*(10\*) \* denotes the same set as

(A) (1\*0) \*1\*

(B) 0 + (0 + 10) \*

(C) (0 + 1) \* 10(0 + 1) \*

(D) none of these

Ans: -

1. (1\*0) \*1\*

Q 4. The regular expression for the language having input alphabets a and b, in which two a’s do not come together:

(A) (b + ab) \* + (b +ab) \*a

(B) a(b + ba)\* + (b + ba)\*

(C) both options (A) and (B)

(D) none of the above

Ans: -

(C) The set of all strings containing at least two 0’s.