NLP

Functionalities

- Tokenization.
- Part Of Speech tagging (POS).
- Named Entity Recognition (NER).
- Classification.
- Sentiment analysis.
- Packages of chatbots.

Applications

- Recommendation systems.
- Sentiment analysis.
- Building chatbots.

For more information, check official documentation: Link

DataCleaning

What is Tokenization?

Tokenization is a process of splitting a text object into smaller units which are also called tokens

- White-space Tokenization
- Regular Expression Tokenization

White-space Tokenization

For Example, Consider the following sentence-

```
Sentence: I went to New-York to play football
Tokens generated are:  “I”, “went”, “to”, “New-York”, “to”, “play”,
“football”
```

Regular Expression Tokenization

For Example, consider the following string containing multiple delimiters such as comma, semi-colon, and white space.

```
Sentence:=  “Basketball, Hockey; Golf Tennis"
re.split(r’[;,s]’, Sentence
Tokens generated are:  “Basketball”, ”Hockey”, “Golf”, “Tennis”
```

Tokenization can be performed at the sentence level or at the word level or even at the character level. Based on it we discuss the following two types of Tokenization:

- Sentence Tokenization
- Word Tokenization

Sentence Tokenization

```
from nltk.tokenize import sent_tokenize
text="Hello friends!. Good Morning! Today we will learn Natural Language Processing. It is very interesting"
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Hello friends!.', 'Good Morning!', 'Today we will learn Natural Language Processing.', 'It is very interesting']
```

Word Tokenization

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Hello', 'friends', '!', '.', 'Good', 'Morning', '!', 'Today', 'we', 'will', 'learn', 'Natural', 'Language', 'Processing',
'.', 'It', 'is', 'very', 'interesting']
```

Noise Entities Removal

Language stopwords (commonly used words of a language – is, am, the, of, in, etc)

`nltk. download("stopwords")`

```
from nltk.corpus import stopwords

#List of stopwords
stopwords = stopwords.words("english")
print(stopwords)
```

Removal of Stopwords

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "India is my country"
text_tokens = word_tokenize(text)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]

print(tokens_without_sw)
```

```
['India', 'country']
```

Parts of Speech (POS) Tagging

For Example, Let's consider the following sentence:

```
Sentence: David has purchased a new laptop from the Apple store
```

In the below sentence, every word is associated with a part of the speech tag which defines its functions.


```
from nltk import word_tokenize, pos_tag
s = "The name of our country is India"
print(pos_tag(word_tokenize(s)))
```

```
[('The', 'DT'), ('name', 'NN'), ('of', 'IN'), ('our', 'PRP$'), ('country', 'NN'), ('is', 'VBZ'), ('India', 'NNP')]
```

Text Cleaning Techniques

1. More Text Cleaning Techniques

- Converting Text to Lowercase
- Removing HTML tags
- Removing Unaccented Characters
- Expanding Contractions
- Removing Special Characters
- Correction of Typos
- Standardization

What is Normalization?

Another type of textual noise happens when there are multiple representations exhibited by a single word.

Due to grammatical reasons, a document can contain:

- Different forms of a particular word such as drive, drives, driving.
- Related words having a similar meaning, such as nation, national, nationality

For Examples,

```
am, are, is => be
dog, dogs, dog's, dogs' => dog
```

If we apply the above mapping to the below sentence, then the text will be look something like this:

```
the boy's dogs are different sizes => the boy dog be a different size
```

Important Points about Normalization

Normalization is useful in many ways. Some of them are as follows:

- Reduces the number of unique tokens present in the text,
- Removes the variations of a word in the text, and
- Removes the redundant information too.

The popular methods which are used for normalization are Stemming and Lemmatization. However, they are different from each other.

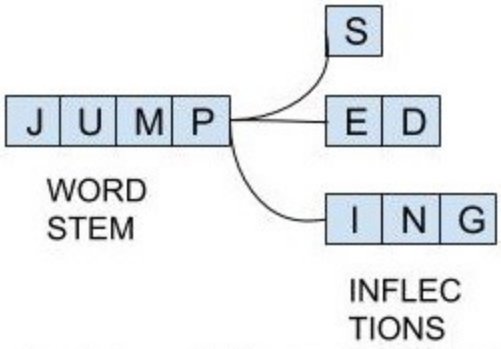
In the Feature Engineering step for text data, Normalization becomes a crucial step as it converts the high dimensional features to the low dimensional space, which is an ideal task for any Machine learning model.

Stemming

To understand stemming, firstly we have to understand the meaning of the word stem. **Word stems,** also known as the base form of a word, and we can create new words by attaching affixes to them in a process known as inflection.

For Example, for the stem word JUMP, we can add affixes to it and form new words like JUMPS, JUMPED, and JUMPING.

```
JUMP -> JUMPS, JUMPED, JUMPING
```



Word stem and its inflections (Source: Text Analytics with Python, Apress/Springer 2016)

For Example, Consider the following words,

```
Words: “laughing”, “laughed“, “laughs”, “laugh”
```

All the above words will become “laugh”, which is their stem because their inflection form will be removed.

“laughing”, “laughed”, “laughs”, “laugh” >>> “laugh”

Here is one quick example using Porter Stemmer in NLTK.

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('studies'))
```

```
studi
```

Lemmatization

The output of lemmatization is the root word called **lemma**. **For Example,**

```
Am, Are, Is >> Be
Running, Ran, Run >> Run
```

Important points about Lemmatization

1. Since Lemmatization is a systematic process so while performing lemmatization we can specify the part of the speech tag for the desired term and lemmatization will only be performed when the given word has a proper part of the speech tag associated with it.

Based on the applicability you can choose any of the below lemmatizers

- Wordnet Lemmatizer
- Spacy Lemmatizer
- TextBlob
- CLiPS Pattern

- Stanford CoreNLP
- Gensim Lemmatizer
- TreeTagger

Here is one quick example using Wordnet lemmatizer in NLTK.

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize('studies'))
```

```
study
```

Important Step 1 Process

Text Cleaning

Sentence Tokenization

```
from nltk.tokenize import sent_tokenize
text="Hello friends!. Good Morning! Today we will learn Natural Language Processing. It is very interesting"
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Hello friends!.', 'Good Morning!', 'Today we will learn Natural Language Processing.', 'It is very interesting']
```

Word Tokenization

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Hello', 'friends', '!', '.', 'Good', 'Morning', '!', 'Today', 'we', 'will', 'learn', 'Natural', 'Language', 'Processing',
'.', 'It', 'is', 'very', 'interesting']
```

Removal of Stopwords

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "India is my country"
text_tokens = word_tokenize(text)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]

print(tokens_without_sw)
```

```
['India', 'country']
```

Normalization & lemmatization

Normalization

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('studies'))
```

```
studi
```

Lemmatization

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize('studies'))
```

```
study
```

Word Embedding and Text Vectorization

This article is part of an ongoing blog series on Natural Language Processing (NLP). Up to the previous part of this article series, we almost completed the necessary steps involved in text cleaning and normalization pre-processing. After that, we will convert the processed text to numeric feature vectors so that we can feed it to computers for Machine Learning applications

What is Word Embedding?

In very simple terms, Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text. But before we dive into the details of Word Embeddings, the following question should come to mind

One-Hot Encoding (OHE)

```
Sentence: I am teaching NLP in Python

Dictionary: [‘I’, ’am’, ’teaching’,’ NLP’,’ in’, ’Python’]
```

```
Vector for NLP: [0,0,0,1,0,0]
Vector for Python:  [0,0,0,0,0,1]
```

Matrix Formulation

Let’s consider the following example:

```
Document-1: He is a smart boy. She is also smart.
Document-2: Chirag is a smart person.
```

The dictionary created contains the list of unique tokens(words) present in the corpus

```
Unique Words: [‘He’, ’She’, ’smart’, ’boy’, ’Chirag’, ’person’]
```

Here, D=2, N=6

So, the count matrix M of size 2 X 6 will be represented as –

	He	She	smart	boy	Chirag	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

Bag-of-Words(BoW)

This vectorization technique converts the text content to numerical feature vectors. Bag of Words takes a document from a corpus and converts it into a numeric vector by mapping each document word to a feature vector for the machine learning model

- Tokenization
- Vectors Creation

Tokenization

It is the process of dividing each sentence into words or smaller parts, which are known as tokens. After the completion of tokenization, we will extract all the unique words from the corpus. Here corpus represents the tokens we get from all the documents and used for the bag of words creation.

```
this burger is very tasty and affordable.
this burger is not tasty and is affordable.
this burger is very very delicious.
```

`Unique words: [“and”, “affordable.”, “delicious.”, “is”, “not”, “burger”, “tasty”, “this”, “very”]`

sparse matrix of example sentences.

Now, the implementation of the above example in Python is given below:

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ["This burger is very tasty and affordable."," This burger is not tasty and is affordable.","This burger is
countvectorizer = CountVectorizer()
X = countvectorizer.fit_transform(corpus)
result = X.toarray()
print(result)
```

```
[[1 1 1 0 1 0 1 1 1]
 [1 1 1 0 2 1 1 1 0]
 [0 0 1 1 1 0 0 1 2]]
```

Now, the implementation of the example discussed in BOW in Python is given below:

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
corpus = ["This burger is very tasty and affordable."," This burger is not tasty and is affordable.","This burger is
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(corpus)
feature_names = vectorizer.get_feature_names()
print(f"Feature names \n{feature_names}")
matrix = vectors.todense()
list_dense = matrix.tolist()
df=pd.DataFrame(list_dense, columns=feature_names)
print(df)
```

```
Feature names
['affordable', 'and', 'burger', 'delicious', 'is', 'not', 'tasty', 'this', 'very']
   affordable       and    burger  delicious        is       not     tasty  \
0    0.414896  0.414896  0.322204   0.000000  0.322204  0.000000  0.414896
1    0.346117  0.346117  0.268791   0.000000  0.537582  0.455102  0.346117
2    0.000000  0.000000  0.282851   0.478909  0.282851  0.000000  0.000000

       this      very
0  0.322204  0.414896
1  0.268791  0.000000
2  0.282851  0.728445
```

Word2Vec

Prediction-based Word Embedding

So far, we have discussed the deterministic methods to determine vector representation of the words but these methods proved to be limited in their word representations until the new word embedding technique named **word2vec** comes to the NLP community.

The popular pre-trained models to create word embedding of a text are as follows:

- Word2Vec — From Google
- Fast text — From Facebook
- **Glove** — From Stanford

Different Model Architectures for Word representation

The following model architectures are used for word representations with an objective to maximize the accuracy and minimize the computation complexity:

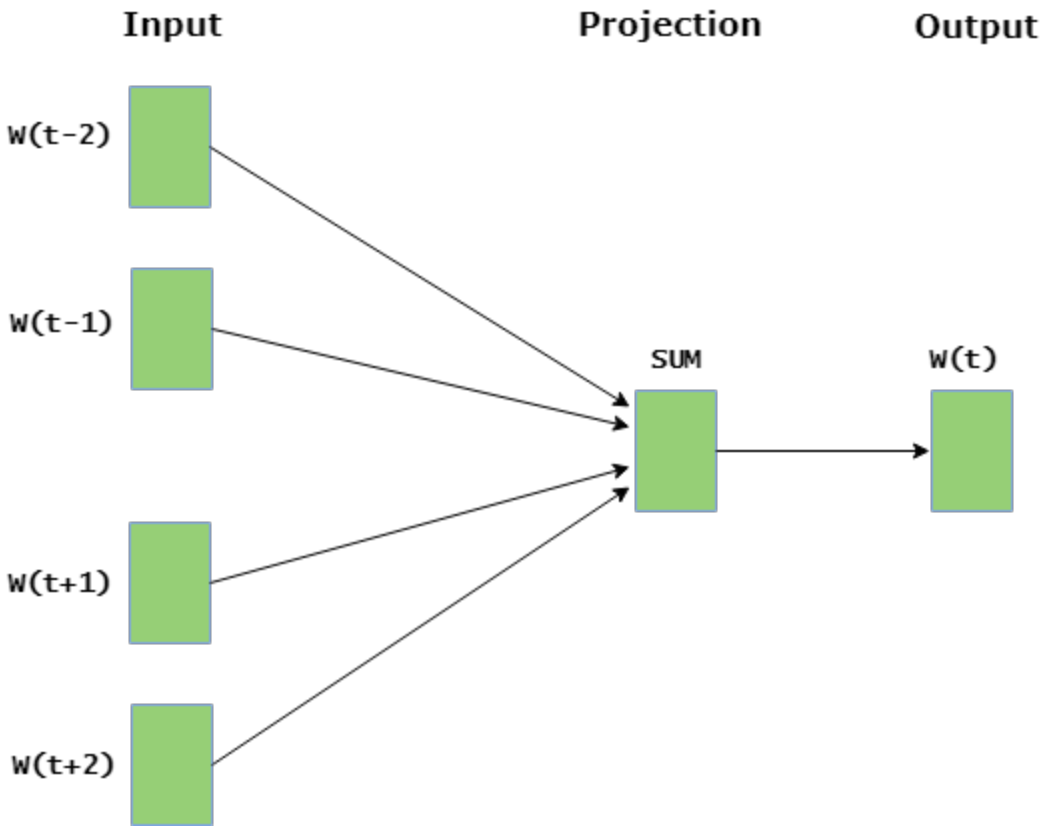
- FeedForward Neural Net Language Model (NNLM)
- Recurrent Neural Net Language Model (RNNLM)

For training of the above-mentioned models, we use Stochastic gradient descent as an optimizer and backpropagation.

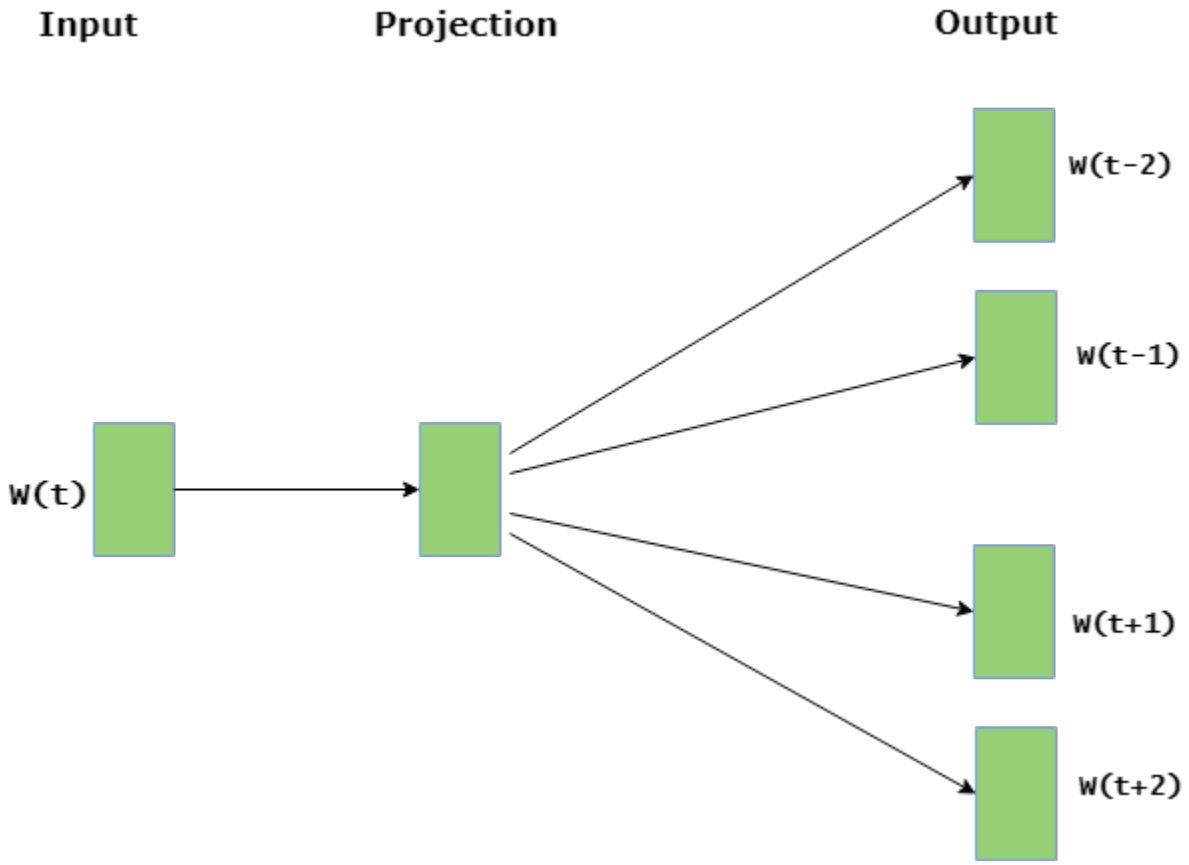
Word Embedding is a language modeling technique used for mapping words to vectors of real numbers. It represents words or phrases in vector space with several dimensions. Word embeddings can be generated using various methods like neural networks, co-occurrence matrix, probabilistic models, etc.

Word2Vec consists of models for generating word embedding. These models are shallow two layer neural networks having one input layer, one hidden layer and one output layer. Word2Vec utilizes two architectures

CBOW (Continuous Bag of Words) : CBOW model predicts the current word given context words within specific window. The input layer contains the context words and the output layer contains the current word. The hidden layer contains the number of dimensions in which we want to represent current word present at the output layer.



Skip Gram : Skip gram predicts the surrounding context words within specific window given current word. The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.



Text Data link here

https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/

Sample

```
pip install nltk
pip install gensim
```

```
import gensim
from gensim.models import Word2Vec

# Create CBOW model
model1 = gensim.models.Word2Vec(data, min_count = 1,
                              size = 100, window = 5)

# Create Skip Gram model
model2 = gensim.models.Word2Vec(data, min_count = 1, size = 100,
                                             window = 5, sg = 1)
print(model1.similarity('alice',
'wonderland'))
print(model2.similarity('alice',
'wonderland'))
```

To Find the degree of similarity between two words

```
model.similarity('woman','man')
#Output
0.73723527
```

To Find the odd one out from a set of words

```
model.doesnt_match('breakfast cereal dinner lunch';.split())
#Output
'cereal'
```

Doing algebraic manipulations using the word (like Woman+King-Man =Queen)

```
model.most_similar(positive=['woman','king'],negative=['man'],topn=1)
#Output
queen: 0.508
```

To find the Probability of a text under the model

```
model.score(['The fox jumped over the lazy dog'.split()])
#Output
0.21
```

Word Embedding Using pre-trained Word Vectors

```
#Import Word2Vec from Gensim Library
from gensim.models import Word2Vec
#Loading the downloaded model
model = Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.
bin', binary=True, norm_only=True)
#Getting word vectors of a word
dog = model['dog']
print(model.most_similar(positive=['woman', 'king'], negative=['man']))
print(model.doesnt_match("breakfast cereal dinner lunch".split()))
print(model.similarity('woman', 'man'))

#Training your own Word Vectors
sentence: [ [‘Chirag’, ’ Boy’], [‘Kshitiz', ’ is’], [‘good’, ’ boy’]]
model = gensim.models.Word2Vec(sentence, min_count=1,size=300,workers=4)
print(model.similarity('woman', 'man'))
```

https://www.geeksforgeeks.org/chat-bot-in-python-with-chatterbot-module/

https://www.datacamp.com/community/tutorials/building-a-chatbot-using-chatterbot

https://www.javatpoint.com/chatbot-in-python

https://www.youtube.com/watch?v=c_gXrw1RoKo

https://www.youtube.com/watch?v=3Qjq1uqtvxI

https://towardsdatascience.com/how-to-build-your-own-chatbot-using-deep-learning-bb41f970e281

Indian Language.(NLP Libraries for Indian Languages)

https://medium.datadriveninvestor.com/natural-langauge-processing-nlp-for-indian-language-hindi-on-web-64d83f16544a

https://www.analyticsvidhya.com/blog/2020/01/3-important-nlp-libraries-indian-languages-python/

how to install INLTK

step1

pip install torch===1.5.0 torchvision===0.6.0 -f https://download.pytorch.org/whl/torch_stable.html

step2

pip install inltk

step3

pip install sklearn

step 4

basic code

```
from inltk.inltk import setup
setup('hi')
```

DTAT CLEANING

https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b

```
import string
print(string.punctuation)
```

```
import re
def tokenize(text):
    tokens = re.split('\W+', text) #W+ means that either a word
character (A-Za-ze-9_) or a dash (-) can go there.    return tokens

print(tokenize("play aaluma doluma song in youtube"))
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "play aaluma doluma song in youtube"text_tokens = word_tokenize
(text)
tokens_without_sw = [word for word in text_tokens if not word in
stopwords.words ()]
print(tokens_without_sw)
```