NLP Complete Document

Roadmap to Master NLP

What is Tokenization? == 1

https://www.guru99.com/tokenize-words-sentences-nltk.html

Tokenization is the process by which a large quantity of text is divided into smaller parts called tokens

word tokenize

```
from nltk.tokenize import word_tokenize
text = "God is Great! I won a lottery."
print(word_tokenize(text))

Output: ['God', 'is', 'Great', '!', 'I', 'won', 'a', 'lottery', '.']
```

sentence tokenize

```
from nltk.tokenize import sent_tokenize
text = "God is Great! I won a lottery."
print(sent_tokenize(text))

Output: ['God is Great!', 'I won a lottery ']
```

What is Stemming? == 2

Stemming and Lemmatization Link. == https://www.guru99.com/stemming-lemmatization-python-nltk.html#3

```
from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
    rootWord=ps.stem(w)
    print(rootWord)

Output
wait
wait
wait
wait
```

What is Lemmatization? == 3 **Normalization & lemmatization both are same**

https://www.guru99.com/stemming-lemmatization-python-nltk.html#3

```
import nltk
        from nltk.stem import         WordNetLemmatizer
        wordnet_lemmatizer = WordNetLemmatizer()
        text = "studies studying cries cry"
        tokenization = nltk.word_tokenize(text)
         for w in tokenization:
                print("Lemma for {} is {}".format(w, wordnet_lemmatizer.
lemmatize(w)))
Output
Lemma for studies is study
Lemma for studying is studying
Lemma for cries is cry
Lemma for cry is cry
```

What is POS Tagging? == 4

https://www.guru99.com/pos-tagging-chunking-nltk.html

```
from collections import Counter
import nltk
text = "Guru99 is one of the best sites to learn WEB, SAP, Ethical
Hacking and much more online."
lower_case = text.lower()
tokens = nltk.word_tokenize(lower_case)
tags = nltk.pos_tag(tokens)
counts = Counter( tag for word,  tag in tags)
print(counts)

Output = [('guru99', 'NN'), ('is', 'VBZ'), ('one', 'CD'), ('of', 'IN'),
('the', 'DT'), ('best', 'JJS'), ('site', 'NN'), ('to', 'TO'), ('learn',
'VB'), ('web', 'NN'), (',', ','), ('sap', 'NN'), (',', ','),
('ethical', 'JJ'), ('hacking', 'NN'), ('and', 'CC'), ('much', 'RB'),
('more', 'JJR'), ('online', 'JJ')]
```

What is Stopwords removal? == 5

https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

```
Stop-words-list
*****************
import nltk
from nltk.corpus import stopwords
print(stopwords.words('english'))
**************************************************************************
***************
how to clear stop-words
************************
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
example_sent = """This is a sample sentence,
                                showing off the stop words
filtration."""
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
filtered_sentence = [w for w in word_tokens if not w.lower() in
stop_words]
filtered_sentence = []
for w in word_tokens:
        if w not in stop_words:
                filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing',
'off', 'the', 'stop', 'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop',
'words', 'filtration', '.']
```

What is Remove Punctuations ? == 6

https://www.programiz.com/python-programming/examples/remove-punctuation

```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

my_str = "Hello!!!, he said ---and went."

# To take input from the user
# my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
   if char not in punctuations:
       no_punct = no_punct + char

# display the unpunctuated string
print(no_punct)
OUTPUT
Hello he said and went
```

Step 2

Advanced level Text Cleaning

Correction of Typos ?

https://www.geeksforgeeks.org/correcting-words-using-nltk-in-python/

```
# importing the nltk suite
import nltk

# importing jaccard distance
# and ngrams from nltk.util
from nltk.metrics.distance import jaccard_distance
from nltk.util import ngrams

# Downloading and importing
# package 'words' from nltk corpus
nltk.download('words')
from nltk.corpus import words
correct_words = words.words()
# list of incorrect spellings
# that need to be corrected
incorrect_words=['happpy', 'azmaing', 'intelliengt']
# loop for finding correct spellings
# based on jaccard distance
# and printing the correct word
for word in incorrect_words:
        temp = [(jaccard_distance(set(ngrams(word, 2)),
                                                        set(ngrams(w,
2))),w)
                        for w in correct_words if w[0]==word[0]]
        print(sorted(temp, key = lambda val:val[0])[0][1])

OUTPUT
happy, amazing, intellingent
```

Step 3

Text preprocessing Level-2

https://www.analyticsvidhya.com/blog/2021/08/a-friendly-guide-to-nlp-bag-of-words-with-python-example/

Bag of words (BOW) === 1

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:

Term frequency Inverse Document Frequency (TFIDF) === 2

https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/

```
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
```

All these are advanced techniques to convert words into vectors

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:

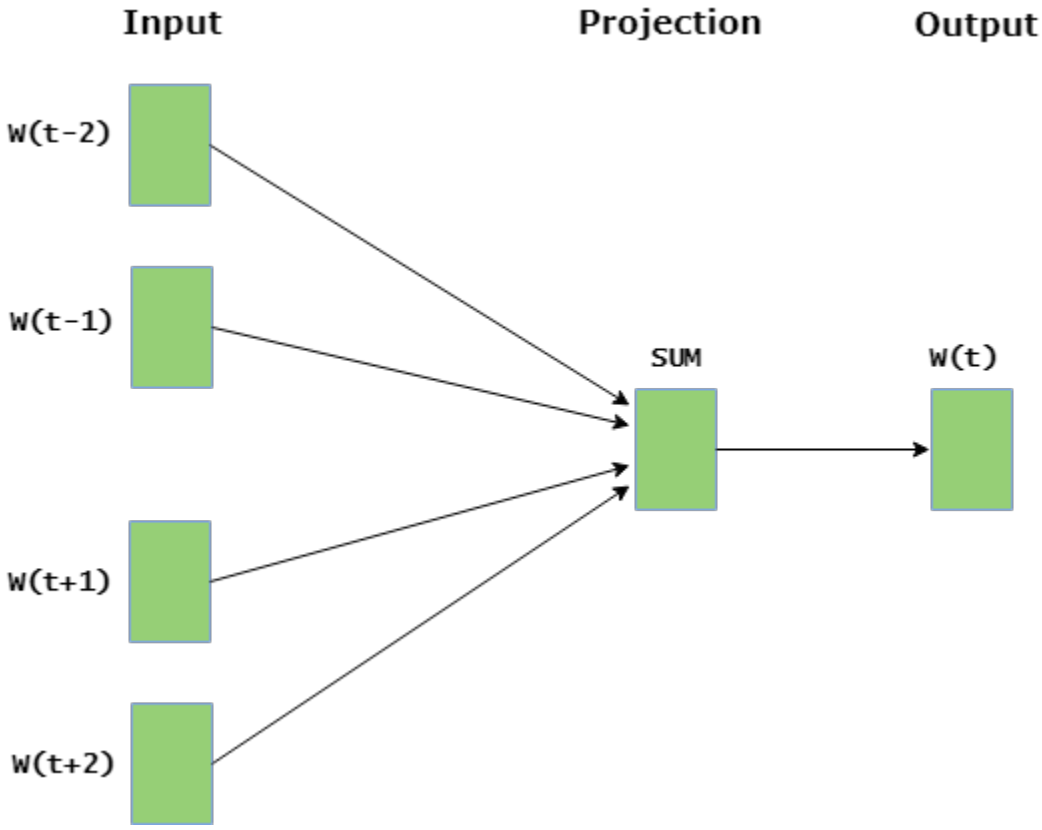
- FeedForward Neural Net Language Model (NNLM)
- Recurrent Neural Net Language Model (RNNLM)

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

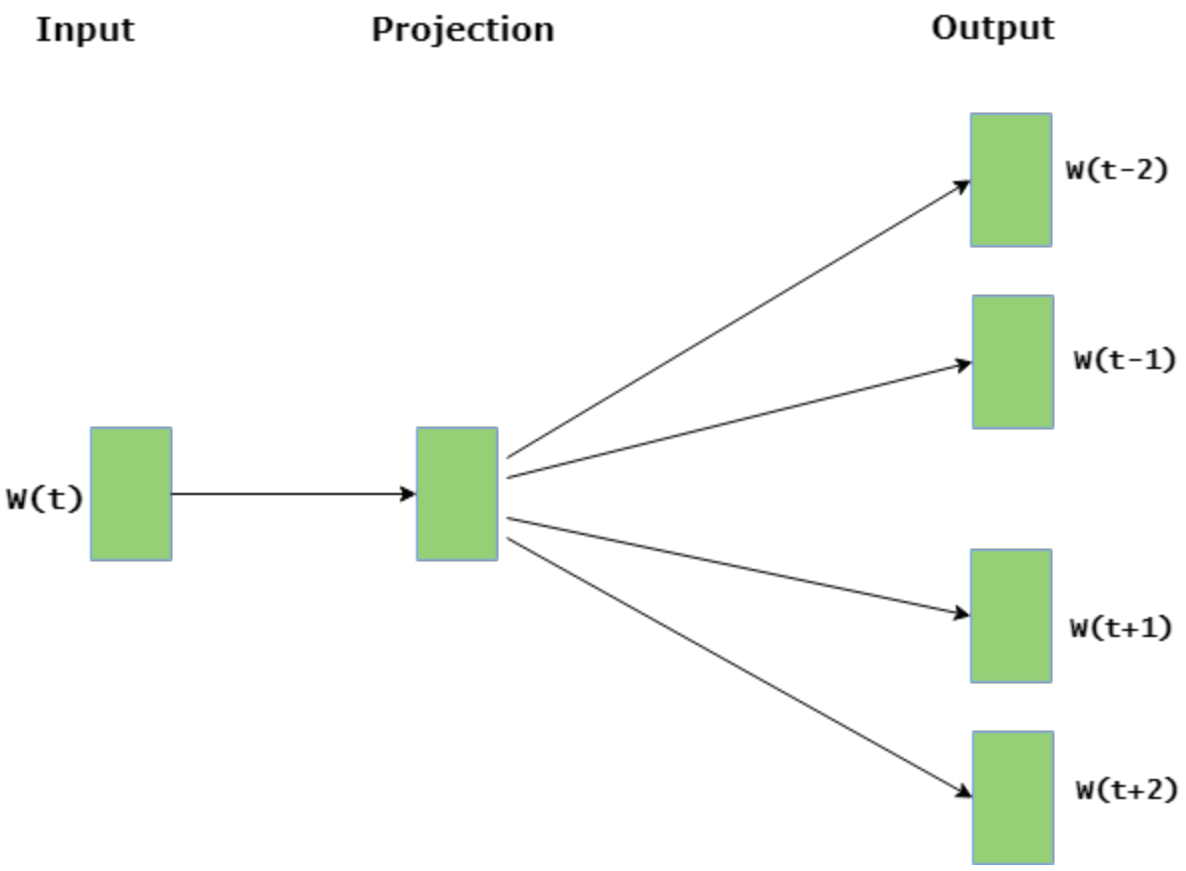
Word Embedding is a language modeling technique used for mapping words to vectors of real numbers. It represents words or phrases in vector space with several dimensions. Word embeddings can be generated using various methods like neural networks, co-occurrence matrix, probabilistic models, etc.

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'))
```

Now we need build a model

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import SGD
import random
import nltk
from nltk.stem import WordNetLemmatizer
import json
import pickle

lemmatizer = WordNetLemmatizer()

words=[]
classes = []
documents = []
ignore_letters = ['!', '?', ',', '.']
intents_file = open('intents.json').read()
intents = json.loads(intents_file)

for intent in intents['intents']:
    for pattern in intent['patterns']:
```

```
        #tokenize each word
        word = nltk.word_tokenize(pattern)
        words.extend(word)
        #add documents in the corpus
        documents.append((word, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
print(documents)
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in
ignore_letters]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)

pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))

# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence

for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent
related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in
pattern_words]
    # create our bag of words array with 1, if word match found in
current pattern
    for word in words:
        bag.append(1) if word in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each
pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
```

```
    training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")

# Create model - 3 layers. First layer 128 neurons, second layer 64
neurons and 3rd output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated
gradient gives good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=
['accuracy'])

#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200,
batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)

print("model created")
```

Final Test

```
from fastapi import FastAPI, File, UploadFile,APIRouter
import uvicorn
from pydantic import BaseModel
import nltk
from nltk.stem import WordNetLemmatizer
import pickle
import numpy as np
import json
import random
from tensorflow.keras.models import load_model

class Item(BaseModel):
    my_value: str# app = FastAPI()bookroute = APIRouter()
```

```
model = load_model('chatbot_model.h5')

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl', 'rb'))
classes = pickle.load(open('classes.pkl', 'rb'))


def clean_up_sentence(sentence):
    # tokenize the pattern - splitting words into array
sentence_words = nltk.word_tokenize(sentence)
    # stemming every word - reducing to base form    sentence_words =
[lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words


# return bag of words array: 0 or 1 for words that exist in sentencedef
bag_of_words(sentence, words, show_details=False):
    # tokenizing patterns    sentence_words = clean_up_sentence
(sentence)
    # bag of words - vocabulary matrix    bag = [0]*len(words)
    for s in sentence_words:
        # print(s)        for i,word in enumerate(words):
            if word == s:
                # assign 1 if current word is in the vocabulary
position                bag[i] = 1                if show_details:
                    print ("found in bag: %s" % word)
    # print(np.array((bag)))    return(np.array(bag))

def predict_class(sentence):
    # filter below  threshold predictions    p = bag_of_words(sentence,
words,show_details=True)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25    results = [[i,r] for i,r in enumerate
(res) if r>ERROR_THRESHOLD]
    # sorting strength probability    results.sort(key=lambda x: x[1],
reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str
(r[1])})
    # print(return_list)    return return_list

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
```

```
            break    return result

@bookroute.post("/english_chatbot_text")
async def analyze_route(input:Item):
    try:
        res = input.my_value
        ints = predict_class(res)
        res = getResponse(ints, intents)

        return {"result":res}
    except Exception as e:
        return {"Success": "false", "Result":str(e) }
```