# Practical No. 01: Convert the text into tokens. Find the word frequency. Date: 13/8/2024

**Program:**

import nltk

from collections import Counter

text = """This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used."""

print("\nThe generated tokenized words are: ") words = nltk.word\_tokenize(text)

print(words, "\n")

word\_freq = Counter(words) print(word\_freq)

# Output:

The generated tokenized words are:

['This', 'tokenizer', 'divides', 'a', 'text', 'into', 'a', 'list', 'of', 'sentences', 'by', 'using', 'an', 'unsupervised', 'algorithm', 'to', 'build',

'a', 'model', 'for', 'abbreviation', 'words', ',', 'collocations', ',', 'and', 'words', 'that', 'start', 'sentences', '.', 'It', 'must', 'be', 'trained',

'on', 'a', 'large', 'collection', 'of', 'plaintext', 'in', 'the', 'target', 'language', 'before', 'it', 'can', 'be', 'used', '.']

Counter({'a': 4, 'of': 2, 'sentences': 2, 'words': 2, ',': 2, '.': 2, 'be': 2, 'This': 1, 'tokenizer': 1, 'divides': 1, 'text': 1, 'into': 1, 'list':

1, 'by': 1, 'using': 1, 'an': 1, 'unsupervised': 1, 'algorithm': 1, 'to': 1, 'build': 1, 'model': 1, 'for': 1, 'abbreviation': 1,

'collocations': 1, 'and': 1, 'that': 1, 'start': 1, 'It': 1, 'must': 1, 'trained': 1, 'on': 1, 'large': 1, 'collection': 1, 'plaintext': 1, 'in':

1, 'the': 1, 'target': 1, 'language': 1, 'before': 1, 'it': 1, 'can': 1, 'used': 1})

# Practical No. 02: Find the synonym /antonym of a word using WordNet. Date: 03/09/2024

**Program:**

import nltk

from nltk.corpus import wordnet

def find\_synonyms\_antonyms(word):

synonyms = [] antonyms = []

# Retrieve all synsets for the word for syn in wordnet.synsets(word):

for lemma in syn.lemmas():

synonyms.append(lemma.name()) # Add the synonym if lemma.antonyms(): # Check if antonyms exist

antonyms.append(lemma.antonyms()[0].name()) # Add the antonym

print(f"Synonyms of {word}: {set(synonyms)}") print(f"Antonyms of {word}: {set(antonyms)}")

# Example usage

# find\_synonyms\_antonyms("Bad")

text = str(input("Enter a word to find its synonyms and antonyms: ")) find\_synonyms\_antonyms(text)

# Output:

Enter a word to find its synonyms and antonyms: good

Synonyms of good: {'unspoiled', 'adept', 'goodness', 'commodity', 'respectable', 'upright', 'effective', 'ripe', 'honest', 'near', 'practiced', 'undecomposed', 'unspoilt', 'good', 'soundly', 'thoroughly', 'proficient', 'just', 'safe', 'beneficial', 'honorable', 'dear', 'secure', 'dependable', 'salutary', 'trade\_good', 'right', 'well', 'in\_effect', 'expert', 'full', 'serious', 'skilful', 'estimable', 'in\_force', 'sound', 'skillful'}

Antonyms of good: {'evilness', 'evil', 'bad', 'ill', 'badness'}

# Practical No. 03: Demonstrate a bigram / trigram language model. Generate regular expression for a given text. Date: 10/09/2024

**Program:**

import nltk

nltk.download('punkt') # nltk.download('punkt\_tab')

from nltk import ngrams

from nltk.tokenize import word\_tokenize

sentence = "N-grams enhance language processing tasks." tokens = word\_tokenize(sentence)

bigrams = list(ngrams(tokens, 2)) trigrams = list(ngrams(tokens, 3))

print("Bigrams: ",bigrams) print("Trigrams: ",trigrams)

# Output:

Bigrams: [('N-grams', 'enhance'), ('enhance', 'language'), ('language', 'processing'), ('processing', 'tasks'), ('tasks', '.')]

Trigrams: [('N-grams', 'enhance', 'language'), ('enhance', 'language', 'processing'), ('language', 'processing', 'tasks'), ('processing', 'tasks', '.')]

[nltk\_data] Downloading package punkt to /root/nltk\_data... [nltk\_data] Package punkt is already up-to-date!

# Practical No. 04: Perform Lemmatization and Stemming. Identify parts-of Speech using Penn Treebank tag set. Date: 21/10/2024

**Program:**

import nltk

nltk.download('punkt')

from nltk.stem import PorterStemmer, WordNetLemmatizer from nltk import pos\_tag, word\_tokenize

# Initialize stemmer and lemmatizer stemmer = PorterStemmer() lemmatizer = WordNetLemmatizer()

# Sample sentence

sentence = "The cats are running quickly." tokens = word\_tokenize(sentence)

# Perform stemming

stemmed\_words = [stemmer.stem(word) for word in tokens] print("Stemmed words:", stemmed\_words)

# Perform lemmatization

lemmatized\_words = [lemmatizer.lemmatize(word, pos='v') for word in tokens] print("Lemmatized words:", lemmatized\_words)

# Output:

Stemmed words: ['the', 'cat', 'are', 'run', 'quickli', '.']

Lemmatized words: ['The', 'cat', 'be', 'run', 'quickly', '.']

[nltk\_data] Downloading package punkt to /root/nltk\_data... [nltk\_data] Package punkt is already up-to-date!

# Program:

import nltk

# Download the specific resource 'averaged\_perceptron\_tagger\_eng'

nltk.download('averaged\_perceptron\_tagger\_eng')

pos\_tags = pos\_tag(tokens) print("POS tags:", pos\_tags)

# Output:

[nltk\_data] Downloading package averaged\_perceptron\_tagger\_eng to [nltk\_data] /root/nltk\_data...

[nltk\_data] Unzipping taggers/averaged\_perceptron\_tagger\_eng.zip.

POS tags: [('The', 'DT'), ('cats', 'NNS'), ('are', 'VBP'), ('running', 'VBG'), ('quickly', 'RB'), ('.', '.')]

# Program:

import nltk

from nltk.stem import WordNetLemmatizer, PorterStemmer from nltk import pos\_tag, word\_tokenize

from nltk.corpus import wordnet

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger') nltk.download('punkt')

# Output:

[nltk\_data] Downloading package wordnet to /root/nltk\_data... [nltk\_data] Package wordnet is already up-to-date!

[nltk\_data] Downloading package averaged\_perceptron\_tagger to [nltk\_data] /root/nltk\_data...

[nltk\_data] Package averaged\_perceptron\_tagger is already up-to- [nltk\_data] date!

[nltk\_data] Downloading package punkt to /root/nltk\_data... [nltk\_data] Package punkt is already up-to-date!

True

# Program:

text = "Perform Lemmatization and Stemming. Identify parts-of Speech using Penn Treebank tag set."

tokens = word\_tokenize(text.lower())

lemmatizer = WordNetLemmatizer() stemmer = PorterStemmer()

lemmatized\_words = [lemmatizer.lemmatize(token) for token in tokens] print("Lemmatized words:", lemmatized\_words)

# Output:

Lemmatized words: ['perform', 'lemmatization', 'and', 'stemming', '.', 'identify', 'parts-of', 'speech', 'using', 'penn', 'treebank', 'tag', 'set', '.']

# Program:

stemmed\_words = [stemmer.stem(token) for token in tokens] print("Stemmed words:", stemmed\_words)

# Output:

Stemmed words: ['perform', 'lemmat', 'and', 'stem', '.', 'identifi', 'parts-of', 'speech', 'use', 'penn', 'treebank', 'tag', 'set', '.']

# Program:

pos\_tags = pos\_tag(tokens) print("POS Tags:", pos\_tags)

# Output:

POS Tags: [('perform', 'NN'), ('lemmatization', 'NN'), ('and', 'CC'), ('stemming', 'NN'), ('.', '.'), ('identify', 'VB'), ('parts-of', 'JJ'),

('speech', 'NN'), ('using', 'VBG'), ('penn', 'JJ'), ('treebank', 'NN'), ('tag', 'NN'), ('set', 'NN'), ('.', '.')]

# Practical No. 05: Implement HMM for POS tagging. Build a Chunker Date: /1 /2024

**Program:**

import nltk

from nltk.corpus import treebank from nltk.tag import hmm

nltk.download('treebank')

train\_data = treebank.tagged\_sents()

trainer = hmm.HiddenMarkovModelTrainer() hmm\_tagger = trainer.train(train\_data)

sentence = "The quick brown fox jumps over the lazy dog".split()

pos\_tags = hmm\_tagger.tag(sentence) print("POS Tags:", pos\_tags)

# Output:

[nltk\_data] Downloading package treebank to /root/nltk\_data... [nltk\_data] Unzipping corpora/treebank.zip.

/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:333: RuntimeWarning: overflow encountered in cast X[i, j] = self.\_transitions[si].logprob(self.\_states[j])

/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:335: RuntimeWarning: overflow encountered in cast O[i, k] = self.\_output\_logprob(si, self.\_symbols[k])

/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:331: RuntimeWarning: overflow encountered in cast P[i] = self.\_priors.logprob(si)

POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NNP'), ('fox', 'NNP'), ('jumps', 'NNP'), ('over', 'NNP'), ('the', 'NNP'), ('lazy', 'NNP'), ('dog', 'NNP')]

/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:363: RuntimeWarning: overflow encountered in cast O[i, k] = self.\_output\_logprob(si, self.\_symbols[k])

# Practical No. 06: Implement Named Entity Recognizer. Date: /1 /2024

**Program:**

import nltk

# Download necessary NLTK data (if not already downloaded) nltk.download('maxent\_ne\_chunker')

nltk.download('words')

nltk.download('averaged\_perceptron\_tagger') nltk.download('punkt')

nltk.download('maxent\_ne\_chunker\_tab')

def named\_entity\_recognizer(text):

# Tokenize the text

tokens = nltk.word\_tokenize(text)

# Part-of-speech tagging pos\_tags = nltk.pos\_tag(tokens)

# Named Entity Recognition using ne\_chunk named\_entities = nltk.ne\_chunk(pos\_tags)

# Print the results (you can modify this to return the results in a different format) print(named\_entities)

# Example usage

text = "Barack Obama was born in Honolulu, Hawaii." named\_entity\_recognizer(text)

# Output:

[nltk\_data] Downloading package maxent\_ne\_chunker to [nltk\_data] /root/nltk\_data...

[nltk\_data] Package maxent\_ne\_chunker is already up-to-date! [nltk\_data] Downloading package words to /root/nltk\_data...

[nltk\_data] Package words is already up-to-date!

[nltk\_data] Downloading package averaged\_perceptron\_tagger to [nltk\_data] /root/nltk\_data...

[nltk\_data] Package averaged\_perceptron\_tagger is already up-to- [nltk\_data] date!

[nltk\_data] Downloading package punkt to /root/nltk\_data... [nltk\_data] Package punkt is already up-to-date!

[nltk\_data] Downloading package maxent\_ne\_chunker\_tab to [nltk\_data] /root/nltk\_data...

[nltk\_data] Package maxent\_ne\_chunker\_tab is already up-to-date! (S

(PERSON Barack/NNP) (PERSON Obama/NNP)

was/VBD born/VBN in/IN

(GPE Honolulu/NNP)

,/,

(GPE Hawaii/NNP)

./.)

# Practical No. 07: Implement Semantic Role Labeling (SRL) to Identify Named Entities Date: /1 /2024

**Program:**

import nltk

# Download necessary NLTK data (if not already downloaded) nltk.download('maxent\_ne\_chunker')

nltk.download('words')

nltk.download('averaged\_perceptron\_tagger') nltk.download('punkt')

nltk.download('conll2000')

from nltk.chunk import conlltags2tree, tree2conlltags from pprint import pprint

def named\_entity\_recognizer(text):

# Tokenize the text

tokens = nltk.word\_tokenize(text)

# Part-of-speech tagging pos\_tags = nltk.pos\_tag(tokens)

# Named Entity Recognition using ne\_chunk

named\_entities = nltk.ne\_chunk(pos\_tags, binary=True) # Use binary=True for simpler output iob\_tagged = tree2conlltags(named\_entities)

pprint(iob\_tagged)

# Print the results (you can modify this to return the results in a different format) #print(named\_entities)

# Example usage

text = "Barack Obama was born in Honolulu, Hawaii. He studied at Columbia University." named\_entity\_recognizer(text)

# Output:

[nltk\_data] Downloading package maxent\_ne\_chunker to [nltk\_data] /root/nltk\_data...

[nltk\_data] Package maxent\_ne\_chunker is already up-to-date! [nltk\_data] Downloading package words to /root/nltk\_data... [nltk\_data] Package words is already up-to-date!

[nltk\_data] Downloading package averaged\_perceptron\_tagger to [nltk\_data] /root/nltk\_data...

[nltk\_data] Package averaged\_perceptron\_tagger is already up-to- [nltk\_data] date!

[nltk\_data] Downloading package punkt to /root/nltk\_data... [nltk\_data] Package punkt is already up-to-date!

[nltk\_data] Downloading package conll2000 to /root/nltk\_data... [nltk\_data] Package conll2000 is already up-to-date!

[('Barack', 'NNP', 'B-NE'),

('Obama', 'NNP', 'I-NE'),

('was', 'VBD', 'O'),

('born', 'VBN', 'O'),

('in', 'IN', 'O'),

('Honolulu', 'NNP', 'B-NE'),

(',', ',', 'O'),

('Hawaii', 'NNP', 'B-NE'),

('.', '.', 'O'),

('He', 'PRP', 'O'),

('studied', 'VBD', 'O'),

('at', 'IN', 'O'),

('Columbia', 'NNP', 'B-NE'),

('University', 'NNP', 'I-NE'),

('.', '.', 'O')]

# Practical No. 08: Implement text classifier using logistic regression model. Date: /1 /2024

**Program:**

# prompt: Implement text classifier using logistic regression model import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Sample data (replace with your actual data)

data = {'text': ['This is a positive sentence.', 'This is a negative sentence.', 'Another positive example.', 'A negative one.'], 'label': [1, 0, 1, 0]} # 1 for positive, 0 for negative

df = pd.DataFrame(data)

# Create TF-IDF features vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(df['text']) y = df['label']

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a logistic regression model model = LogisticRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy: {accuracy}")

# Example prediction

new\_text = ['This is a new positive sentence.'] new\_text\_vectorized = vectorizer.transform(new\_text) prediction = model.predict(new\_text\_vectorized) print(f"Prediction for '{new\_text[0]}': {prediction[0]}")

# Output:

Accuracy: 0.0

Prediction for 'This is a new positive sentence.': 1

# Practical No. 09: Implement a movie reviews sentiment classifier Date: /1 /2024

**Program:**

# prompt: Implement a movie reviews sentiment classifier

import nltk

import random

import numpy as np import pandas as pd

from nltk.corpus import movie\_reviews

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Download necessary NLTK data (if not already downloaded) nltk.download('movie\_reviews')

nltk.download('punkt')

# Prepare the data

documents = [(list(movie\_reviews.words(fileid)), category) for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

random.shuffle(documents)

all\_words = []

for w in movie\_reviews.words(): all\_words.append(w.lower())

all\_words = nltk.FreqDist(all\_words)

word\_features = list(all\_words.keys())[:3000]

def find\_features(document):

words = set(document) features = {}

for w in word\_features:

features[w] = (w in words) return features

featuresets = [(find\_features(rev), category) for (rev, category) in documents]

training\_set = featuresets[:1900] testing\_set = featuresets[1900:]

# Train the classifier

classifier = nltk.NaiveBayesClassifier.train(training\_set)

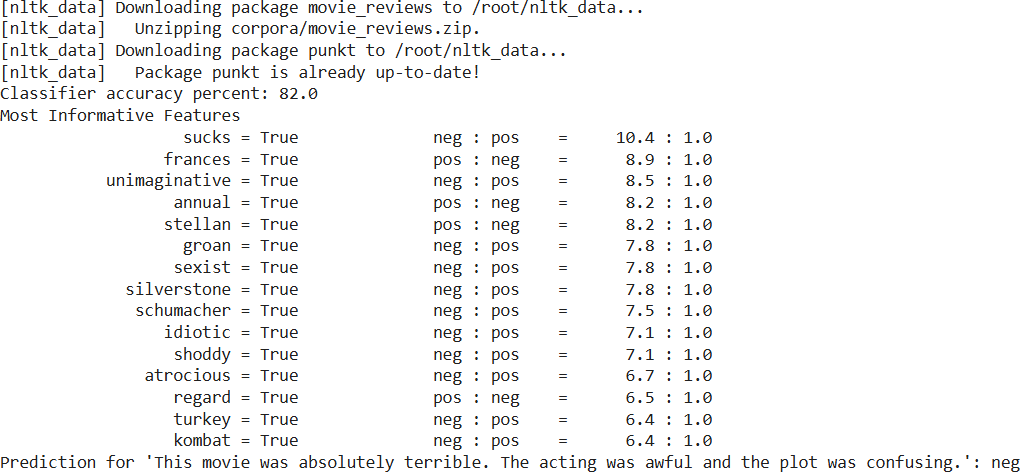
print("Classifier accuracy percent:", (nltk.classify.accuracy(classifier, testing\_set))\*100) classifier.show\_most\_informative\_features(15)

# Example usage

example\_text = "This movie was absolutely terrible. The acting was awful and the plot was confusing." example\_features = find\_features(word\_tokenize(example\_text.lower()))

prediction = classifier.classify(example\_features) print(f"Prediction for '{example\_text}': {prediction}")

# Output:



**Practical No. 10: Implement RNN for sequence labelling and show some output Date: /1 /2024**

# Program:

import numpy as np

# Sample data (replace with your actual sequence labeling data) # Sequences: [['word1', 'word2', ...], ...]

# Labels: [['label1', 'label2', ...], ...]

sequences = [['The', 'quick', 'brown', 'fox'], ['jumps', 'over', 'the', 'lazy', 'dog']]

labels = [['DET', 'ADJ', 'ADJ', 'NOUN'], ['VERB', 'ADP', 'DET', 'ADJ', 'NOUN']]

# Create vocabulary and label dictionaries word\_to\_index = {}

label\_to\_index = {} index\_to\_label = {}

for seq, lab in zip(sequences, labels):

for word in seq:

if word not in word\_to\_index:

word\_to\_index[word] = len(word\_to\_index) for label in lab:

if label not in label\_to\_index:

label\_to\_index[label] = len(label\_to\_index) index\_to\_label[len(label\_to\_index)-1] = label

# Convert data to numerical representations

X = [[word\_to\_index[word] for word in seq] for seq in sequences] y = [[label\_to\_index[label] for label in lab] for lab in labels]

# Pad sequences to ensure uniform length max\_len = max(len(seq) for seq in X)

X = [seq + [0] \* (max\_len - len(seq)) for seq in X] y = [lab + [0] \* (max\_len - len(lab)) for lab in y]

import numpy as np

# Simulate RNN calculations (replace with actual RNN implementation) # Example: a basic RNN using NumPy

def simple\_rnn(input\_seq, weights, bias):

hidden\_state\_size = weights[1].shape[0] # Get the size of the hidden state from the recurrent weight matrix hidden\_state = np.zeros(hidden\_state\_size) # Initialize hidden state with the correct size

outputs = []

for word\_index in input\_seq:

input\_vector = np.zeros(len(word\_to\_index))

input\_vector[word\_index] = 1 #one-hot encoding

hidden\_state = np.tanh(np.dot(input\_vector, weights[0]) + np.dot(hidden\_state, weights[1]) + bias[0])

# Predict label using the hidden state (replace with more appropriate prediction method) output\_probs = np.dot(hidden\_state, weights[2]) + bias[1]

predicted\_label\_index = np.argmax(output\_probs) outputs.append(predicted\_label\_index)

return outputs

# Randomly initialize weights and biases (replace with training)

hidden\_state\_size = 10 # Define the desired size of the hidden state

weights = [np.random.rand(len(word\_to\_index), hidden\_state\_size), np.random.rand(hidden\_state\_size,

hidden\_state\_size), np.random.rand(hidden\_state\_size, len(label\_to\_index))] # Ensure weights are compatible with the hidden state size

bias = [np.random.rand(hidden\_state\_size), np.random.rand(len(label\_to\_index))]

# Example usage: predict labels for a sequence predicted\_labels = simple\_rnn(X[0], weights, bias) print(predicted\_labels) # output as indexes

#convert prediction to label

print([index\_to\_label[pred] for pred in predicted\_labels])

# Output:

[1, 1, 1, 1, 1]

['ADJ', 'ADJ', 'ADJ', 'ADJ', 'ADJ']