DSA0310 – Natural language processing (practical programs with output)

Name: R.Rithvik Roshan

192124021

1.

import re

text="new dog1233"

word=r'\bdog[0-9]\*\b'

match1=re.findall(word,text)

if match1:

  print("match found")

  for a in match1:

    print(a)

output:

match found

dog1233

2.

strings\_to\_test = ["ab", "abc", "defab", "abab", "ababab","abab"]

for string in strings\_to\_test:

    state = 0

    for char in string:

        if state == 0 and char == 'a':

            state = 1

        elif state == 1 and char == 'b':

            state = 2

        else:

            state = 0

    if state == 2:

        print(f"'{string}' matches the pattern.")

    else:

        print(f"'{string}' does not match the pattern.")

output:

'ab' matches the pattern.

'abc' does not match the pattern.

'defab' matches the pattern.

'abab' does not match the pattern.

'ababab' matches the pattern.

'abab' does not match the pattern.

3.

from nltk import PorterStemmer

from nltk.tokenize import word\_tokenize

import nltk

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('words')

nltk.download('stopwords')

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

text = "you must be takecare while crossing road and also lookup the magically"

port = PorterStemmer()

words = word\_tokenize(text)

print(words)

stem1 = [port.stem(word) for word in words]

print(stem1)

output:

['you', 'must', 'be', 'takecare', 'while', 'crossing', 'road', 'and', 'also', 'lookup', 'the', 'magically']

['you', 'must', 'be', 'takecar', 'while', 'cross', 'road', 'and', 'also', 'lookup', 'the', 'magic']

4.

irregular\_nouns = {'man': 'men', 'woman': 'women', 'child': 'children'}

words\_to\_pluralize = ['cat', 'dog', 'man', 'woman', 'bus', 'box', 'church']

for word in words\_to\_pluralize:

    if word in irregular\_nouns:

        plural\_form = irregular\_nouns[word]

    elif word.endswith('s') or word.endswith('x') or word.endswith('z'):

        plural\_form = word + 'es'

    else:

        plural\_form = word + 's'

    print(f'Plural of {word}: {plural\_form}')

output:

Plural of cat: cats

Plural of dog: dogs

Plural of man: men

Plural of woman: women

Plural of bus: buses

Plural of box: boxes

Plural of church: churchs

5.

from nltk import PorterStemmer

from nltk.tokenize import word\_tokenize

import nltk

nltk.download('wordnet')

nltk.download('words')

nltk.download('stopwords')

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

text="you must be takecare  while crossing road and also lookup the magically"

port= PorterStemmer()

words=word\_tokenize(text)

print(words)

stem1=[port.stem(word) for word in words]

print(stem1)

output:

['you', 'must', 'be', 'takecare', 'while', 'crossing', 'road', 'and', 'also', 'lookup', 'the', 'magically']

['you', 'must', 'be', 'takecar', 'while', 'cross', 'road', 'and', 'also', 'lookup', 'the', 'magic']

6.

import random

sample\_text = "This is a sample text "

words = sample\_text.split()

bigrams = [(words[i], words[i+1]) for i in range(len(words)- 1)]

generated\_text = []

current\_word = random.choice(words)

for \_ in range(50):

    next\_candidates = [b[1] for b in bigrams if b[0] == current\_word]

    if next\_candidates:

        next\_word = random.choice(next\_candidates)

    else:

        next\_word = random.choice(words)

    generated\_text.append(current\_word)

    current\_word = next\_word

generated\_text = ' '.join(generated\_text)

print("\n",generated\_text)

output:

sample text a sample text a sample text This is a sample text is a sample text sample text text is a sample text This is a sample text text is a sample text text a sample text sample text a sample text This is a sample text sample text

7.

import nltk

from nltk.tokenize import word\_tokenize

from nltk import pos\_tag

nltk.download('punkt')

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

text = "this is my pen"

words = word\_tokenize(text)

print(words)

tag1 = pos\_tag(words)

print(tag1)

output:

['this', 'is', 'my', 'pen']

[('this', 'DT'), ('is', 'VBZ'), ('my', 'PRP$'), ('pen', 'JJ')]

8.

import random

sentence = "The quick brown fox jumps over the lazy dog"

words = sentence.split()

pos\_model = {

    'The': 'DT',

    'quick': 'JJ',

    'brown': 'JJ',

    'fox': 'NN',

    'jumps': 'VB',

    'over': 'IN',

    'the': 'DT',

    'lazy': 'JJ',

    'dog': 'NN'

}

pos\_tags = [pos\_model.get(word, random.choice(['NN', 'VB', 'JJ', 'DT', 'IN'])) for word in words]

tagged\_sentence = list(zip(words, pos\_tags))

print(tagged\_sentence)

output:

[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VB'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]

9.

import re

sentence = "The quick brown fox jumps over the lazy dog"

pos\_rules = [

    (r'\bThe\b', 'DT'),

    (r'\bquick\b', 'JJ'),

    (r'\bbrown\b', 'JJ'),

    (r'\bfox\b', 'NN'),

    (r'\bjumps\b', 'VB'),

    (r'\bover\b', 'IN'),

    (r'\bthe\b', 'DT'),

    (r'\blazy\b', 'JJ'),

    (r'\bdog\b', 'NN')

]

words = sentence.split()

pos\_tags = []

for word in words:

    for pattern, pos\_tag in pos\_rules:

        if re.search(pattern, word, re.IGNORECASE):

            pos\_tags.append((word, pos\_tag))

            break

    else:

        pos\_tags.append((word, 'NN'))

print(pos\_tags)

output:

[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VB'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]

10.

import re

sentence = "The quick brown fox jumps over the lazy dog"

initial\_tagging = {

    'The': 'DT',

    'quick': 'NN',

    'brown': 'NN',

    'fox': 'NN',

    'jumps': 'NN',

    'over': 'IN',

    'the': 'DT',

    'lazy': 'NN',

    'dog': 'NN'

}

transformation\_rules = [

    (r'(\w+) DT (\w+)', r'\1 NN \2'),

    (r'(\w+) IN (\w+)', r'\1 NN \2'),

    (r'(\w+) NN (\w+)', r'\1 VB \2'),

    (r'(\w+) NN (\w+)', r'\1 JJ \2'),

]

improvement = True

while improvement:

    improvement = False

    previous\_tagging = initial\_tagging.copy()

    for rule in transformation\_rules:

        pattern, replacement = rule

        for word in initial\_tagging:

            initial\_tagging[word] = re.sub(pattern, replacement, initial\_tagging[word])

    if initial\_tagging != previous\_tagging:

        improvement = True

tagged\_sentence = [(word, tag) for word, tag in initial\_tagging.items()]

print(tagged\_sentence)

output:

[('The', 'DT'), ('quick', 'NN'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'NN'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'NN'), ('dog', 'NN')]

11.

class SimpleParser:

    def \_\_init\_\_ (self, grammar):

        self.grammar = grammar

    def parse (self, input\_string):

        self.input = input\_string

        self.index = 0

        self.result = True

        if self.expression ():

            if self.index == len (self.input):

                print (f'Parsing successful for input: {input\_string}')

                return

        print (f'Parsing failed for input: {input\_string}')

    def expression (self):

        return self.term () and self.expression\_tail ()

    def expression\_tail (self):

        current\_index = self.index

        if self.match ('+'):

            return self.term () and self.expression\_tail ()

        self.index = current\_index

        return True

    def term (self):

        return self.factor () and self.term\_tail ()

    def term\_tail (self):

        current\_index = self.index

        if self.match ('\*'):

            return self.factor () and self.term\_tail ()

        self.index = current\_index

        return True

    def factor (self):

        if self.match ('('):

            if self.expression () and self.match (')'):

                return True

            return False

        return self.match ('number')

    def match (self, expected):

        if self.index < len (self.input) and (expected == self.input [self.index] or expected == 'number' and self.input [self.index].isdigit ()):

            self.index += 1

            return True

        return False

grammar = {

    'start': 'Expression',

}

parser = SimpleParser (grammar)

parser.parse ('3\* (2+1)')

parser.parse ('2+1\*3')

parser.parse ('2+ (1\*1)')

output:

Parsing failed for input: 3\* (2+1)

Parsing successful for input: 2+1\*3

Parsing failed for input: 2+ (1\*1)

12.

class EarleyParser:

    def \_\_init\_\_(self, grammar):

        self.grammar = grammar

    def parse(self, input\_string):

        self.chart = [[] for \_ in range(len(input\_string) + 1)]

        self.chart[0].append(('start', '', 0))

        for i in range(len(input\_string) + 1):

            for state in self.chart[i]:

                self.predictor(state, i)

                if i < len(input\_string):

                    self.scanner(state, input\_string[i], i)

                else:

                    self.completer(state, i)

        if ('start', self.grammar['start'], 0) in self.chart[len(input\_string)]:

            print(f'Parsing successful for input: {input\_string}')

        else:

            print(f'Parsing failed for input: {input\_string}')

    def predictor(self, state, index):

        if state[1] in self.grammar:

            for production in self.grammar[state[1]]:

                self.chart[index].append((state[1], production, index))

    def scanner(self, state, token, index):

        if state[1] == '' or state[1][0] != token:

            return

        self.chart[index + 1].append((state[0], state[1][1:], state[2]))

    def completer(self, state, index):

        for st in self.chart[state[2]]:

            if st[1] == '' or st[1][0] != state[0]:

                continue

            self.chart[index].append((st[0], st[1][1:], st[2]))

grammar = {

    'start': 'Expression',

    'Expression': ['Term + Expression', 'Term'],

    'Term': ['Factor \* Term', 'Factor'],

    'Factor': ['( Expression )', 'number']

}

parser = EarleyParser(grammar)

parser.parse('3\* (2+1)')

parser.parse('2+1\*3')

parser.parse('2+ (1\*3)')

output:

Parsing failed for input: 3\* (2+1)

Parsing failed for input: 2+1\*3

Parsing failed for input: 2+ (1\*3)

13.

import nltk

from nltk import CFG

from nltk import parse

grammar = CFG.fromstring("""

    S -> NP VP

    NP -> Det N

    VP -> V NP

    Det -> 'the' | 'a'

    N -> 'cat' | 'dog'

    V -> 'chased' | 'ate'

""")

parser = parse.ChartParser(grammar)

def generate\_parse\_tree(sentence):

    words = nltk.word\_tokenize(sentence)

    chart = parser.chart\_parse(words)

    parse\_tree = list(chart.parses(grammar.start()))[0]

    parse\_tree.pretty\_print()

sentence = "the cat chased the dog"

generate\_parse\_tree(sentence)

output:

S

\_\_\_\_\_\_\_\_|\_\_\_\_\_

| VP

| \_\_\_\_\_|\_\_\_

NP | NP

\_\_\_|\_\_\_ | \_\_\_|\_\_\_

Det N V Det N

| | | | |

the cat chased the dog

14.

import nltk

from nltk import CFG, ChartParser

grammar = CFG.fromstring("""

    S -> NP\_SG VP\_SG | NP\_PL VP\_PL

    NP\_SG -> 'the' 'cat'

    NP\_PL -> 'the' 'cats'

    VP\_SG -> 'chases'

    VP\_PL -> 'chase'

""")

parser = ChartParser(grammar)

def check\_subject\_verb\_agreement(sentence):

    words = sentence.split()

    parse\_trees = list(parser.parse(words))

    if not parse\_trees:

        return "No valid parse tree found for the sentence."

    return "Subject and verb agree in the sentence."

sentences = [

    "the cat chases the cat",

    "the cats chase the cats",

    "the cat chases the cats",

    "the cats chase the cat",

]

for sentence in sentences:

    agreement\_result = check\_subject\_verb\_agreement(sentence)

    print(f"Sentence: {sentence}")

    print(f"Agreement: {agreement\_result}")

    print()

output:

Sentence: the cat chases the cat

Agreement: No valid parse tree found for the sentence.

Sentence: the cats chase the cats

Agreement: No valid parse tree found for the sentence.

Sentence: the cat chases the cats

Agreement: No valid parse tree found for the sentence.

Sentence: the cats chase the cat

Agreement: No valid parse tree found for the sentence.

15.

import nltk

from nltk import PCFG

from nltk import ChartParser

pcfg\_grammar = PCFG.fromstring("""

    S -> NP VP [1.0]

    NP -> Det N [0.7] | NP PP [0.3]

    VP -> V NP [0.4] | VP PP [0.3] | V [0.3]

    PP -> P NP [1.0]

    Det -> 'the' [0.6] | 'a' [0.4]

    N -> 'cat' [0.2] | 'dog' [0.2] | 'bat' [0.2] | 'rat' [0.2] | 'hat' [0.2]

    V -> 'chased' [0.5] | 'saw' [0.5]

    P -> 'in' [0.6] | 'on' [0.4]

""")

parser = ChartParser(pcfg\_grammar)

sentence = "the cat chased the dog".split()

for tree in parser.parse(sentence):

    tree.pretty\_print()

output:

S

\_\_\_\_\_\_\_\_|\_\_\_\_\_

| VP

| \_\_\_\_\_|\_\_\_

NP | NP

\_\_\_|\_\_\_ | \_\_\_|\_\_\_

Det N V Det N

| | | | |

the cat chased the dog

16.

import spacy

nlp = spacy.load("en\_core\_web\_sm")

text = "Apple Inc. is a technology company based in Cupertino, California. Tim Cook is the CEO."

doc = nlp(text)

for ent in doc.ents:

    print(f"Entity: {ent.text}, Label: {ent.label\_}")

organization\_entities = [ent.text for ent in doc.ents if ent.label\_ == "ORG"]

print("Organizations:", organization\_entities)

output:

Entity: Apple Inc., Label: ORG

Entity: Cupertino, Label: GPE

Entity: California, Label: GPE

Entity: Tim Cook, Label: PERSON

Organizations: ['Apple Inc.']

17.

import nltk

from nltk.corpus import wordnet

nltk.download('wordnet')

word\_synsets = wordnet.synsets("example")

for synset in word\_synsets:

    print(f"Synset: {synset.name()}")

    print(f"Definition: {synset.definition()}")

    print(f"Examples: {synset.examples()}")

    print()

word = "happy"

synonyms = []

for syn in wordnet.synsets(word):

    for lemma in syn.lemmas():

        synonyms.append(lemma.name())

print(f"Synonyms for '{word}': {synonyms}")

output:

Synset: example.n.01

Definition: an item of information that is typical of a class or group

Examples: ['this patient provides a typical example of the syndrome', 'there is an example on page 10']

Synset: model.n.07

Definition: a representative form or pattern

Examples: ['I profited from his example']

Synset: exemplar.n.01

Definition: something to be imitated

Examples: ['an exemplar of success', 'a model of clarity', 'he is the very model of a modern major general']

Synset: example.n.04

Definition: punishment intended as a warning to others

Examples: ['they decided to make an example of him']

Synset: case.n.01

Definition: an occurrence of something

Examples: ['it was a case of bad judgment', 'another instance occurred yesterday', 'but there is always the famous example of the Smiths']

Synset: exercise.n.04

Definition: a task performed or problem solved in order to develop skill or understanding

Examples: ['you must work the examples at the end of each chapter in the textbook']

Synonyms for 'happy': ['happy', 'felicitous', 'happy', 'glad', 'happy', 'happy', 'well-chosen']

18.

facts = {("R", "apple", "banana"), ("R", "banana", "cherry"), ("R", "apple", "cherry")}

expressions = ["R(apple, banana)", "R(banana, cherry)", "R(apple, cherry)", "R(pear, orange)"]

for expression in expressions:

    predicate, args = expression.split('(')

    args = args.rstrip(')').split(',')

    if (predicate, args[0], args[1]) in facts:

        result = True

    else:

        result = False

    print(f"{expression}: {result}")

output:

R(apple, banana): False

R(banana, cherry): False

R(apple, cherry): False

R(pear, orange): False

19.

import nltk

from nltk.corpus import wordnet

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

nltk.download('wordnet')

nltk.download('stopwords')

nltk.download('punkt')

def lesk\_algorithm(context, target\_word):

    context\_tokens = word\_tokenize(context)

    context\_words = [word.lower() for word in context\_tokens if word.lower() not in stopwords.words('english')]

    target\_synsets = wordnet.synsets(target\_word)

    if not target\_synsets:

        return None

    best\_sense = None

    max\_overlap = 0

    for sense in target\_synsets:

        sense\_definition = word\_tokenize(sense.definition())

        sense\_examples = [word.lower() for word in word\_tokenize(' '.join(sense.examples()))]

        overlap = len(set(context\_words).intersection(set(sense\_definition + sense\_examples)))

        if overlap > max\_overlap:

            max\_overlap = overlap

            best\_sense = sense

    return best\_sense

context = "He saw the bat fly over the baseball field."

target\_word = "bat"

result = lesk\_algorithm(context, target\_word)

if result:

    print(f"Word sense of '{target\_word}' in the context: {result.name()} - {result.definition()}")

else:

    print(f"Unable to disambiguate the word '{target\_word}' in the context.")

output:

Word sense of 'bat' in the context: bat.n.02 - (baseball) a turn trying to get a hit

20.

import math

from collections import Counter

documents = [

    "This is the first document. It is a simple document.",

    "This document is the second one. It has more words.",

    "And this is the third document. It has even more words."

]

query = "simple document"

stopwords = set(["this", "is", "the", "and", "it", "has"])

tokenized\_documents = []

for doc in documents:

    doc = doc.lower()

    doc = doc.split()

    doc = [word for word in doc if word not in stopwords and word.isalpha()]

    tokenized\_documents.append(doc)

N = len(documents)

vocabulary = set(word for doc in tokenized\_documents for word in doc)

tfidf\_scores = []

for doc in tokenized\_documents:

    tfidf\_doc = {}

    for term in vocabulary:

        tf = doc.count(term)

        df = sum(1 for d in tokenized\_documents if term in d)

        idf = math.log(N / (df + 1))

        tfidf = tf \* idf

        tfidf\_doc[term] = tfidf

    tfidf\_scores.append(tfidf\_doc)

query = query.lower().split()

query\_vector = Counter(query)

ranked\_documents = []

for i, doc in enumerate(tfidf\_scores):

    dot\_product = sum(doc[term] \* query\_vector[term] for term in query if term in doc)

    doc\_length = math.sqrt(sum(score \*\* 2 for score in doc.values()))

    query\_length = math.sqrt(sum(score \*\* 2 for score in query\_vector.values()))

    if query\_length == 0:

        similarity = 0

    else:

        similarity = dot\_product / (doc\_length \* query\_length)

    ranked\_documents.append((i, similarity))

ranked\_documents.sort(key=lambda x: x[1], reverse=True)

for i, similarity in ranked\_documents:

    print(f"Document {i + 1} - Similarity: {similarity:.4f}")

output:

Document 2 - Similarity: 0.5000

Document 1 - Similarity: 0.4082

Document 3 - Similarity: 0.0000

21.

import nltk

from nltk.corpus import wordnet

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

# Sample sentence

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

# Tokenize the sentence and perform part-of-speech tagging

tokens = nltk.word\_tokenize(sentence)

pos\_tags = nltk.pos\_tag(tokens)

# Noun phrase extraction using part-of-speech tags

noun\_phrases = []

current\_noun\_phrase = []

for word, pos in pos\_tags:

    if pos.startswith('N'):  # Nouns and noun phrases typically start with 'N'

        current\_noun\_phrase.append(word)

    else:

        if current\_noun\_phrase:

            noun\_phrases.append(' '.join(current\_noun\_phrase))

        current\_noun\_phrase = []

# Add the last noun phrase if it exists

if current\_noun\_phrase:

    noun\_phrases.append(' '.join(current\_noun\_phrase))

# Extract meanings of noun phrases using WordNet

noun\_phrase\_meanings = {}

for np in noun\_phrases:

    synsets = wordnet.synsets(np)

    if synsets:

        meanings = [synset.definition() for synset in synsets]

        noun\_phrase\_meanings[np] = meanings

# Display extracted noun phrases and their meanings

print("Noun Phrases:")

for np in noun\_phrases:

    print("- " + np)

print("\nMeanings of Noun Phrases:")

for np, meanings in noun\_phrase\_meanings.items():

    print("- " + np + ":")

    for meaning in meanings:

        print("  - " + meaning)

22.

import spacy

def resolve\_references(text):

    nlp = spacy.load("en\_core\_web\_sm")

    doc = nlp(text)

    resolved\_text = []

    current\_reference = None

    for token in doc:

        if token.text.lower() == 'it' and current\_reference:

            resolved\_text.append(current\_reference.text)

        else:

            resolved\_text.append(token.text)

            if token.dep\_ == 'nsubj':

                current\_reference = token

    return ' '.join(resolved\_text)

# Example usage

text = "The cat is on the mat. It is sleeping."

resolved\_text = resolve\_references(text)

print("Resolved Text:", resolved\_text)

output:

Resolved Text: The cat is on the mat . cat is sleeping .

23.

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

def calculate\_coherence(text):

    # Tokenize the text into sentences

    sentences = sent\_tokenize(text)

    # Tokenize each sentence into words

    tokenized\_sentences = [word\_tokenize(sentence) for sentence in sentences]

    # Create a Bag-of-Words representation of the sentences

    vectorizer = CountVectorizer()

    sentence\_vectors = vectorizer.fit\_transform([" ".join(sentence) for sentence in tokenized\_sentences]).toarray()

    # Calculate cosine similarity between sentence vectors

    similarity\_matrix = cosine\_similarity(sentence\_vectors)

    # Calculate average similarity (coherence) excluding self-similarity

    total\_similarity = 0

    for i in range(len(similarity\_matrix)):

        for j in range(len(similarity\_matrix[i])):

            if i != j:

                total\_similarity += similarity\_matrix[i][j]

    coherence = total\_similarity / ((len(sentences) \* (len(sentences) - 1)) / 2)

    return coherence

if \_\_name\_\_ == "\_\_main\_\_":

    input\_text = """

    John loves to play video games. He often spends hours exploring virtual worlds and completing challenging missions.

    The thrill of victory keeps him engaged. Besides gaming, he enjoys hiking in the mountains.

    Nature provides a welcome contrast to the digital adventures. Last weekend, he hiked to the summit of a breathtaking peak and captured stunning photos.

    John's dual passion for gaming and nature showcases the diverse aspects of his personality.

    """

    coherence\_score = calculate\_coherence(input\_text)

    print(f"Coherence Score: {coherence\_score}")

output:

Coherence Score: 0.23594663094821858

24.

import nltk

from nltk import pos\_tag, ne\_chunk

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

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

nltk.download('words')

def recognize\_dialog\_acts(utterance):

    tagged\_words = pos\_tag(word\_tokenize(utterance))

    chunked\_tree = ne\_chunk(tagged\_words)

    dialog\_acts = []

    for subtree in chunked\_tree:

        if isinstance(subtree, nltk.Tree):

            label, mention = subtree.label(), subtree.leaves()[0][0]

            response = {'PERSON': f'Hello! How can I help you, {mention}?',

                        'GPE': f'I see you mentioned {mention}. What can I assist you with related to that?'}.get(label, "I'm not sure how to respond.")

            dialog\_acts.append((label, response))

        else:

            dialog\_acts.append(('UNKNOWN', "I'm not sure how to respond."))

    return dialog\_acts

# Example usage

conversation = ["Hi there!", "Planning a trip to Paris.", "What's the weather like there?",

                "Not sure about the weather, but I can help with travel tips.", "Tell me more about Paris."]

for utterance in conversation:

    acts = recognize\_dialog\_acts(utterance)

    print("Dialog Acts:", acts)

output:

Dialog Acts: [('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond.")]

Dialog Acts: [('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('GPE', 'I see you mentioned Paris. What can I assist you with related to that?'), ('UNKNOWN', "I'm not sure how to respond.")]

Dialog Acts: [('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond.")]

Dialog Acts: [('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond."), ('UNKNOWN', "I'm not sure how to respond.")

25.

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

nltk.download('punkt')

# Sample conversation

conversation = [

"Hello there! How are you doing today?",

"I'm doing great, thanks for asking.",

"Can you help me with my homework?",

"Sure, I'd be happy to help. What do you need assistance with?",

"I'm stuck on this math problem.",

"Alright, let me take a look.",

]

# Iterate through the conversation and recognize dialog acts

for i, utterance in enumerate(conversation):

# Tokenize the utterance into sentences

sentences = sent\_tokenize(utterance)

for j, sentence in enumerate(sentences):

words = word\_tokenize(sentence)

if "?" in words:

dialog\_act = "question"

else:

dialog\_act = "statement"

print(f"Utterance {i + 1}, Sentence {j + 1}: {dialog\_act.capitalize()} - {sentence}")

output:

Utterance 1, Sentence 1: Statement - Hello there!

Utterance 1, Sentence 2: Question - How are you doing today?

Utterance 2, Sentence 1: Statement - I'm doing great, thanks for asking.

Utterance 3, Sentence 1: Question - Can you help me with my homework?

Utterance 4, Sentence 1: Statement - Sure, I'd be happy to help.

Utterance 4, Sentence 2: Question - What do you need assistance with?

Utterance 5, Sentence 1: Statement - I'm stuck on this math problem.

Utterance 6, Sentence 1: Statement - Alright, let me take a look.

26.

from transformers import pipeline

def translate\_english\_to\_french(text):

    translator = pipeline(task="translation", model="Helsinki-NLP/opus-mt-en-fr")

    translated\_text = translator(text, max\_length=50)[0]['translation\_text']

    return translated\_text

# Example usage

english\_text = "Hello, how are you?"

french\_translation = translate\_english\_to\_french(english\_text)

print("English:", english\_text)

print("French Translation:", french\_translation)

output:

