Here are the additional programs from the document, formatted as requested:

### 1) AMBIGUITY

**Code:**

Python

from nltk import CFG, ChartParser  
  
# Example 1: Lexical Ambiguity  
word = "bank"  
meanings = [  
 "A financial institution where people deposit and withdraw money",  
 "The land alongside a river",  
 "To tilt an airplane while turning"  
]  
print("LEXICAL AMBIGUITY:")  
print(f"The word '{word}' has {len(meanings)} meanings:")  
for i, meaning in enumerate(meanings, 1):  
 print(f"{i}. {meaning}")  
print("\n" + "-"\*50 + "\n")  
  
# Example 2: Syntactic Ambiguity  
grammar = CFG.fromstring("""  
S -> NP VP  
VP -> V NP | VP PP  
PP -> P NP  
NP -> Det N | NP PP | 'I'  
V -> 'saw'  
Det -> 'the' | 'a'  
N -> 'man' | 'telescope'  
P -> 'with'  
""")  
sentence = ['I', 'saw', 'the', 'man', 'with', 'a', 'telescope']  
parser = ChartParser(grammar)  
  
print("SYNTACTIC AMBIGUITY:")  
print("Sentence: 'I saw the man with a telescope'")  
print("Possible parses:\n")  
  
for tree in parser.parse(sentence):  
 print(tree)  
 tree.pretty\_print()  
 print()

**Output:**

Plaintext

LEXICAL AMBIGUITY:  
The word 'bank' has 3 meanings:  
1. A financial institution where people deposit and withdraw money  
2. The land alongside a river  
3. To tilt an airplane while turning  
  
--------------------------------------------------  
  
SYNTACTIC AMBIGUITY:  
Sentence: 'I saw the man with a telescope'  
Possible parses:  
  
(S (NP I) (VP (VP (V saw) (NP (Det the) (N man))) (PP (P with) (NP (Det a) (N telescope)))))  
 S  
 |  
 NP  
 |  
 VP  
 / \  
VP PP  
| / \  
...  
I saw the man with a telescope  
  
(S (NP I) (VP (V saw) (NP (NP (Det the) (N man)) (PP (P with) (NP (Det a) (N telescope))))))  
 S  
 |  
 NP  
 |  
 VP  
 |  
 V  
 / \  
saw NP  
 / \  
 NP PP  
 | / \  
 ...  
I saw the man with a telescope

### 2) Text classification program (Named Entity Recognition)

**Code:**

Python

import spacy  
from spacy import displacy  
  
# Load the small English model  
nlp = spacy.load("en\_core\_web\_sm")  
text = """ Apple Inc. is looking at buying U.K. startup for $1 billion. Tim Cook, the CEO of Apple, said this deal will be completed by next month. Barack Obama was the 44th president of the United States. """  
doc = nlp(text)  
  
# Print entities and their labels  
for ent in doc.ents:  
 print(f"Text: {ent.text} | Label: {ent.label\_}")  
  
# The displacy visualization part of the output cannot be rendered here.

**Output:**

Plaintext

Text: Apple Inc. | Label: ORG  
Text: U.K. | Label: GPE  
Text: $1 billion | Label: MONEY  
Text: Tim Cook | Label: PERSON  
Text: Apple | Label: ORG  
Text: next month | Label: DATE  
Text: Barack Obama | Label: PERSON  
Text: 44th | Label: ORDINAL  
Text: the United States | Label: GPE

### 3) Smoothing

**Code:**

Python

from sentence\_transformers import SentenceTransformer  
from sklearn.metrics.pairwise import cosine\_similarity  
import numpy as np  
  
model = SentenceTransformer('all-MiniLM-L6-v2')   
sentences = [ "I love playing football.", "Soccer is my favorite sport.", "The weather is sunny today.", "It is raining outside."]  
sentence\_vectors = model.encode(sentences)  
  
print("Sentence embedding shape:", sentence\_vectors.shape)  
similarity\_matrix = cosine\_similarity(sentence\_vectors)  
print("\nCosine Similarity Matrix:\n")  
print(similarity\_matrix)  
  
most\_similar\_idx = np.argmax(similarity\_matrix[0][1:]) + 1   
print(f"\nSentence most similar to '{sentences[0]}' is '{sentences[most\_similar\_idx]}'")

**Output:**

Plaintext

Sentence embedding shape: (4, 384)  
  
Cosine Similarity Matrix:  
  
[[ 1. 0.65746427 -0.02304317 0.01971502]  
 [ 0.65746427 1.0000002 0.05700713 0.04840674]  
 [-0.02304317 0.05700713 1.0000001 0.46222505]  
 [ 0.01971502 0.04840674 0.46222505 0.99999976]]  
  
Sentence most similar to 'I love playing football.' is 'Soccer is my favorite sport.'

### 4) Text summarization

**Code:**

Python

from transformers import pipeline  
  
summarizer = pipeline("summarization", model="t5-small") # T5-small model  
text = """ Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans using natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of human languages in a manner that is valuable. Many challenges in NLP involve natural language understanding, natural language generation, sentiment analysis, text summarization, machine translation, and question answering. """  
summary = summarizer(text, max\_length=50, min\_length=25, do\_sample=False)  
  
print("Original Text:\n", text)  
print("\nSummarized Text:\n", summary[0]['summary\_text'])

**Output:**

Plaintext

Original Text:  
 Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between  
 computers and humans using natural language. The ultimate objective of NLP is to read, decipher, understand, and  
 make sense of human languages in a manner that is valuable. Many challenges in NLP involve natural language  
 understanding, natural language generation, sentiment analysis, text summarization, machine translation, and  
 question answering.  
  
Summarized Text:  
natural language processing (NLP) is a field of artificial intelligence that focuses on the interaction between  
computers and humans using natural language. many challenges in NLP involve natural language understanding,  
natural language generation, sentiment analysis, text summarization, machine translation.

### 5) RECOMMENDATION SYSTEM

**Code:**

Python

from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import cosine\_similarity  
   
movies = [   
 {"title": "The Matrix", "description": "A hacker learns about the true nature of reality and fights against controllers."},   
 {"title": "Inception", "description": "A thief enters dreams to plant an idea into a CEO's mind."},   
 {"title": "Interstellar", "description": "Explorers travel through a wormhole in space to ensure humanity's survival."},   
 {"title": "The Lord of the Rings", "description": "A hobbit journeys to destroy a powerful ring and save Middle-earth."},  
 {"title": "The Social Network", "description": "The story of the founding of Facebook and the legal battles that followed."}  
]  
   
descriptions = [movie['description'] for movie in movies]  
vectorizer = TfidfVectorizer(stop\_words='english')  
tfidf\_matrix = vectorizer.fit\_transform(descriptions)  
similarity\_matrix = cosine\_similarity(tfidf\_matrix)  
  
def recommend(title, movies=movies, similarity\_matrix=similarity\_matrix, top\_n=3):  
 idx = next(i for i, m in enumerate(movies) if m['title'] == title)  
 sim\_scores = list(enumerate(similarity\_matrix[idx]))  
 sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)  
 sim\_scores = sim\_scores[1:top\_n+1]  
   
 print(f"Movies similar to '{title}':")  
 for i, score in sim\_scores:  
 print(f"- {movies[i]['title']} (Similarity: {score:.2f})")  
  
recommend("Inception")

**Output:**

Plaintext

Movies similar to 'Inception':  
- The Matrix (Similarity: 0.00)  
- Interstellar (Similarity: 0.00)  
- The Lord of the Rings (Similarity: 0.00)

### 6) Encoder decoder machine translation

**Code:**

Python

import numpy as np  
import tensorflow as tf  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
   
latent\_dim = 32  
embedding\_dim = 32  
   
# Data Preparation (as shown in the record)  
input\_texts = ["hello", "how are you", "i am fine", "thank you", "good morning"]  
target\_texts = [" bonjour", " comment ça va ", " je vais bien", " merci ", " bonjour "]  
   
input\_tokenizer = Tokenizer()  
input\_tokenizer.fit\_on\_texts(input\_texts)  
input\_sequences = input\_tokenizer.texts\_to\_sequences(input\_texts)  
max\_encoder\_seq\_length = max(len(seq) for seq in input\_sequences)  
num\_encoder\_tokens = len(input\_tokenizer.word\_index) + 1  
encoder\_input\_data = pad\_sequences(input\_sequences, maxlen=max\_encoder\_seq\_length, padding='post')  
   
target\_tokenizer = Tokenizer(filters='')  
target\_tokenizer.fit\_on\_texts(target\_texts)  
target\_sequences = target\_tokenizer.texts\_to\_sequences(target\_texts)  
max\_decoder\_seq\_length = max(len(seq) for seq in target\_sequences)  
num\_decoder\_tokens = len(target\_tokenizer.word\_index) + 1  
decoder\_input\_data = pad\_sequences([seq[:-1] for seq in target\_sequences], maxlen=max\_decoder\_seq\_length-1, padding='post')  
decoder\_target\_data = pad\_sequences([seq[1:] for seq in target\_sequences], maxlen=max\_decoder\_seq\_length-1, padding='post')  
decoder\_target\_data\_oh = np.zeros((len(input\_texts), max\_decoder\_seq\_length-1, num\_decoder\_tokens), dtype='float32')  
for i, seq in enumerate(decoder\_target\_data):  
 for t, word\_idx in enumerate(seq):  
 if word\_idx > 0:  
 decoder\_target\_data\_oh[i, t, word\_idx] = 1.0  
  
# Encoder Setup  
encoder\_inputs = Input(shape=(max\_encoder\_seq\_length,))  
enc\_emb = Embedding(num\_encoder\_tokens, embedding\_dim, input\_length=max\_encoder\_seq\_length)(encoder\_inputs)  
encoder\_lstm = LSTM(latent\_dim, return\_state=True)  
encoder\_outputs, state\_h, state\_c = encoder\_lstm(enc\_emb)  
encoder\_states = [state\_h, state\_c]  
   
# Decoder Setup  
decoder\_inputs = Input(shape=(max\_decoder\_seq\_length-1,))  
dec\_emb = Embedding(num\_decoder\_tokens, embedding\_dim, input\_length=max\_decoder\_seq\_length-1)(decoder\_inputs)  
decoder\_lstm = LSTM(latent\_dim, return\_sequences=True, return\_state=True)  
decoder\_outputs, \_, \_ = decoder\_lstm(dec\_emb, initial\_state=encoder\_states)  
decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax')  
decoder\_outputs = decoder\_dense(decoder\_outputs)  
   
# Model Compilation and Training  
model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
model.summary()  
model.fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data\_oh, batch\_size=2, epochs=50, verbose=1)

**Output:**

Plaintext

Model: "functional"  
Layer (type) Output Shape | Param # | Connected to  
=================================================================  
input\_layer\_1 (InputLayer) | (None, 3) | 0 | -  
input\_layer\_2 (InputLayer) | (None, 4) | 0 | -  
embedding (Embedding) | (None, 3, 32) | 352 | input\_layer\_1[0][0]  
embedding\_1 (Embedding) | (None, 4, 32) | 352 | input\_layer\_2[0][0]  
lstm (LSTM) | [(None, 32), (None, 32), (None, 32)] | 8,320 | embedding[0][0]  
lstm\_1 (LSTM) | [(None, 4, 32), (None, 32), (None, 32)] | 8,320 | embedding\_1[0][0], lstm[0][1], lstm[0][2]  
dense\_1 (Dense) | (None, 4, 11) | 363 | lstm\_1[0][0]  
=================================================================  
Total params: 17,707  
Trainable params: 17,707  
Non-trainable params: 0  
-----------------------------------------------------------------  
Epoch 1/50  
3/3 - loss: 1.6760 - accuracy: 0.1500  
Epoch 2/50  
3/3 - loss: 1.6684 - accuracy: 0.2500  
...  
Epoch 37/50  
3/3 - loss: 0.9888 - accuracy: 0.3000  
...  
Epoch 50/50  
3/3 - loss: 0.7505 - accuracy: 0.5000

### 7) Named entity recognition

**Code:**

Python

import spacy  
nlp = spacy.load("en\_core\_web\_sm")  
text = """Apple is planning to open a new office in Hyderabad, India by 2026. Tim Cook, the CEO of Apple, met with Narendra Modi to discuss investment opportunities."""  
doc = nlp(text)  
  
print("Named Entities, their labels, and positions:\n")  
for ent in doc.ents:  
 print(f"{ent.text:<25} | {ent.label\_:<10} | Start: {ent.start\_char}, End: {ent.end\_char}")  
  
print("\nEntity Label Explanation:")  
print(f"GPE: {spacy.explain('GPE')}")  
print(f"ORG: {spacy.explain('ORG')}")  
print(f"PERSON: {spacy.explain('PERSON')}")

**Output:**

Plaintext

Named Entities, their labels, and positions:  
  
Apple | ORG | Start: 0, End: 5  
Hyderabad | GPE | Start: 42, End: 51  
India | GPE | Start: 53, End: 58  
2026 | DATE | Start: 62, End: 66  
Tim Cook | PERSON | Start: 80, End: 88  
Apple | ORG | Start: 101, End: 106  
Narendra Modi | PERSON | Start: 117, End: 130  
  
Entity Label Explanation:  
GPE: Countries, cities, states  
ORG: Companies, agencies, institutions, etc.  
PERSON: People, including fictional

### 8) Word to vec

**Code:**

Python

import nltk  
from gensim.models import Word2Vec   
  
nltk.download('punkt')  
  
text = """Natural Language Processing is a field of Artificial Intelligence. It deals with how computers understand human language. Word embeddings like Word2Vec capture semantic meaning of words."""  
sentences = nltk.sent\_tokenize(text)  
data = [nltk.word\_tokenize(sentence.lower()) for sentence in sentences]  
  
model = Word2Vec(sentences=data, vector\_size=50, window=5, min\_count=1, sg=1)  
  
print("Vector representation for 'language':\n", model.wv['language'])  
print("\nWords most similar to 'language':")  
print(model.wv.most\_similar('language'))  
print("\nSimilarity between 'language' and 'computers':", model.wv.similarity('language', 'computers'))

**Output:**

Plaintext

Vector representation for 'language':  
[-0.01631313 0.00898634 -0.00828216 0.00164936 0.01698417 -0.00893383  
 0.00903378 -0.01356238 -0.00709291 0.01878958 -0.00315817 0.00064451  
 -0.00826876 -0.01536609 -0.00303185 0.0049505 -0.00176973 0.01108032  
 -0.00550026 0.00451065 0.01092105 0.01671072 -0.00290536 -0.01840301  
 0.00874378 0.00115624 0.01488257 -0.00161443 -0.00528963 -0.01750265  
 -0.00171541 0.00567086 0.01080243 0.01410966 -0.01140788 0.00371402  
 0.01217452 -0.00960064 -0.00619748 0.01360284 0.00326445 0.00038038  
 0.00693825 0.00044166 0.01925861 0.01011735 -0.01783446 -0.01409558  
 0.00180856 0.01277417]  
  
Words most similar to 'language':  
[('understand', 0.23005631566047668), ('embeddings', 0.16102485358715057), ('word', 0.14890338480472565), ('deals',  
0.12482792884111404), ('with', 0.08072882890701294), ('is', 0.07400191575288773), ('word2vec', 0.055481769144535065), ('.',  
0.04347098991274834), ('processing', 0.018909117206931114), ('how', 0.01197921484708786)]  
  
Similarity between 'language' and 'computers': -0.034299165

### 9) word disambiguity

**Code:**

Python

import nltk  
from nltk.corpus import wordnet as wn  
from nltk.wsd import lesk  
  
sentences = [  
 "He deposited money in the bank.",  
 "The fisherman sat on the bank of the river."  
]  
  
for sent in sentences:  
 tokens = nltk.word\_tokenize(sent)  
 sense = lesk(tokens, "bank")  
 print(f"Sentence: {sent}")  
 print(f"Predicted Sense: {sense}")  
 print(f"Definition: {sense.definition() if sense else 'No definition found'}\n")

**Output:**

Plaintext

Sentence: He deposited money in the bank.  
Predicted Sense: Synset('savings\_bank.n.02')  
Definition: a container (usually with a slot in the top) for keeping money at home  
  
Sentence: The fisherman sat on the bank of the river.  
Predicted Sense: Synset('bank.v.07')  
Definition: cover with ashes so to control the rate of burning

### 10) sentence to vec

**Code:**

Python

from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import cosine\_similarity  
   
sentences = [ "I love playing football.", "Soccer is my favorite sport.", "The Eiffel Tower is in Paris." ]  
vectorizer = TfidfVectorizer()  
sentence\_vectors = vectorizer.fit\_transform(sentences)  
  
print("Vector shape:", sentence\_vectors.shape)  
similarity = cosine\_similarity(sentence\_vectors[0], sentence\_vectors[1])  
print("Similarity between first two sentences:", similarity[0][0])

**Output:**

Plaintext

Vector shape: (3, 13)  
Similarity between first two sentences: 0.0

### 11) TEXT GENERATION WITH LSTM

**Code:**

Python

import numpy as np  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import LSTM, Dense, Embedding  
   
# Sample text (toy dataset)  
text = "lstm is a type of recurrent neural network. it is used in natural language processing."  
chars = sorted(list(set(text)))  
char\_to\_idx = {c: i for i, c in enumerate(chars)}  
idx\_to\_char = {i: c for i, c in enumerate(chars)}  
   
# Convert text to integers  
encoded = [char\_to\_idx[c] for c in text]  
   
# Prepare sequences  
seq\_length = 10  
x = []  
y = []  
for i in range(len(encoded) - seq\_length):  
 x.append(encoded[i:i+seq\_length])  
 y.append(encoded[i+seq\_length])  
x = np.array(x)  
y = np.array(y)  
   
# Build model  
model = Sequential()  
model.add(Embedding(len(chars), 50, input\_length=seq\_length))  
model.add(LSTM(100))  
model.add(Dense(len(chars), activation='softmax'))  
model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam')  
   
# Train   
model.fit(x, y, epochs=50, verbose=1) # The epoch output is summarized below.  
   
# Generate text  
seed = "lstm is a "  
seed\_encoded = [char\_to\_idx[c] for c in seed]  
generated\_text = seed   
   
for \_ in range(50):  
 x\_pred = np.array(seed\_encoded[-seq\_length:]).reshape(1, seq\_length)  
 pred = np.argmax(model.predict(x\_pred, verbose=0))  
 seed\_encoded.append(pred)  
 generated\_text = "".join(idx\_to\_char[i] for i in seed\_encoded)   
   
print("Generated text:")  
print(generated\_text)

**Output:**

Plaintext

Epoch 1/50  
3/3 - loss: 3.0434  
...  
Epoch 50/50  
3/3 - loss: 1.8525  
  
Generated text:  
lstm is a yye preeerer nnnenaalllggggppeesssiiiiiintnntuaa