|  |
| --- |
| LAB PROGRAMS -1Tokenization using UNIX commands  1. Create a text file text1.txt containing text of your choice Ans : echo "This is an example text file." > text1.txt 2. Use tr utility to replace every occurrence of a given character to another given character Ans: tr 'o' 'x' < example.txt (Hello World!) > Hellx Wxrld! 3. What are s and c options of tr command? Ans: echo "aaabbbccc" | tr -s 'a' Output: abbbc   **-s option (squeeze-repeats):** This option is used to squeeze (reduce) repeated characters in the input to a single character.  **-c option (complement):** This option is used to complement the set of characters. When used, it replaces characters not listed in the first set with the corresponding character from the second set  echo "abc123" | tr -c 'a-z' 'X' > Output: XXX123   1. What is the output of the following command? tr a-z A-Z < text1.txt   Ans: The command tr a-z A-Z < text1.txt will transform all lowercase letters to uppercase in the content of the file text1.txt. The output will be displayed on the terminal.   1. Create a text file text2.txt containing any text (one word per line) sort the words in text2 file using “sort” command   Ans: echo -e "banana\napple\norange\nkiwi\ngrape" > text2.txt sort text2.txt   1. Sort and display unique lines in the text2.txt file Ans: sort -u text2.txt   **Output**: apple banana grape  kiwi orange   1. Sort and display unique lines in text2.txt file such that each word is preceded with frequency count of the word in text2.txt file   Ans: sort text2.txt | uniq -c Output:  * 1. apple   2. banana   1 grape  3 kiwi  1 orange   1. Obtain and display the tokens in text1.txt file   Ans: tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' (input: This is an example text file.) Output: This is an  example text  file   1. Display the tokens in sorted order   Ans: tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' | sort Output: This an  example file  is text   1. Display the unique tokens in sorted order   Ans: tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' | sort | uniq Output: This an  example file  is text LAB PROGRAMS - 2Implement a word tokenization using regular expressions from nltk.tokenize import RegexpTokenizer  s = "Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\n\nThanks." tokenizer = RegexpTokenizer(r'\w+|\$[\d\.]+|\S+')  print(tokenizer.tokenize(s)) Output: import nltk nltk.download('punkt')  nltk.download('averaged\_perceptron\_tagger') nltk.download('maxent\_ne\_chunker') nltk.download('words')  sentence = """At eight o'clock on Thursday morning... Arthur didn't feel very good.""" tokens = nltk.word\_tokenize(sentence)  print(tokens)  **Output**:    text = "This is V.C.E C.S.E I am a Student . I paid a fees of 13.0"  print(re.findall("(?:[A-Z]\.)+[A-Z]",text)) # ([A-Z]\.)+ add "?:" before this for concatenation text = "That U.S.A poster-print costs $12.40... which is 3.45."  print(re.split(" ",text)) #regex for hypened-words  print(re.findall("\w+-\w+",text)) print(re.findall("(?:\w+-\w+)",text)) from nltk.corpus import stopwords print(word\_tokenize(text))  li = []  for w in word\_tokenize(text):  if w not in stopwords.words('english'): li.append(w)  li Output: ['V.C.E', 'C.S.E']  ['That', '', 'U.S.A', 'poster-print', 'costs', '$12.40...', 'which', 'is', '3.45.'] ['poster-print']  ['poster-print']  ['That', 'U.S.A', 'poster-print', 'costs', '$', '12.40', '...', 'which', 'is', '3.45', '.']  ['That', 'U.S.A', 'poster-print', 'costs', '$', '12.40', '...', '3.45', '.'] |
| LAB PROGRAMS- 3Implement Minimum Edit Distance (MED) algorithm for spelling correction from re import M  #source = input('Enter the string given: ') source = ‘kitten’  #target = input('Enter the string to convert to: ') target = ‘sitting’  m = len(source) n n= len(target)  dp = [[0 for i in range(m+1)] for j in range(n+1)] print(dp)  for i in range(m+1):  dp[0][i] = i  for j in range(n+1):  dp[j][0] = j print(dp)  for i in range(1, n+1):  for j in range(1, m+1):  if source[j-1] == target[i-1]:  cost = 0 else:cost = 1  dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+cost) print(dp)  print('Edit Distance: ', dp[n][m])  **Output:**    import nltk  def find\_minimum\_edit\_distance(word1, word2): distance = nltk.edit\_distance(word1, word2) return distance  # Example usage word1 = "kitten" word2 = "sitting"  min\_edit\_distance = find\_minimum\_edit\_distance(word1, word2)  print(f"The minimum edit distance between '{word1}' and '{word2}' is: {min\_edit\_distance}") Output: The minimum edit distance between Kitten and Sitting is: 3 |
| LAB PROGRAMS - 4Implement n-gram language model import nltk nltk.download('brown')  from nltk.corpus import brown  from nltk.lm.preprocessing import pad\_both\_ends from nltk.util import bigrams  corpus = brown.sents(categories = "news")  test\_sentence = ['There', "wasn't", 'a', 'bit', 'of', 'trouble', 'in', 'Texas'] test\_sentence\_bigrams = list(bigrams(pad\_both\_ends(test\_sentence, n=2))) print(test\_sentence\_bigrams)  from nltk.lm.preprocessing import padded\_everygram\_pipeline train, vocab = padded\_everygram\_pipeline(2, corpus)  from nltk.lm import MLE lm = MLE(2)  lm.fit(train, vocab)  print('Number of words in vocabulary is:', len(lm.vocab)) print(lm.counts)  print( lm.score("There",["<s>"]) ) prob = 1  for t in test\_sentence\_bigrams: print(lm.score(t[1],[t[0]]))  prob = prob\*lm.score(t[1],[t[0]]) print(prob)  **Output**: |
| LAB PROGRAMS - 5Implement Naïve Bayes classification for sentiment analysis from nltk.corpus import movie\_reviews import random  import nltk  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 = nltk.FreqDist(w.lower() for w in movie\_reviews.words()) word\_features = list(all\_words)[:2000]  def document\_features(document): document\_words = set(document) features = {}  for word in word\_features:  features['contains({})'.format(word)] = (word in document\_words) return features  #print(document\_features(movie\_reviews.words('pos/cv957\_8737.txt'))) featuresets = [(document\_features(d), c) for (d,c) in documents] train\_set, test\_set = featuresets[100:], featuresets[:100]  classifier = nltk.NaiveBayesClassifier.train(train\_set) #print(test\_set[0][0].values())  for i in range(100): print(classifier.classify(test\_set[i][0]), test\_set[i][1] )  for i in range(100):  predicted = classifier.classify(test\_set[i][0])  actual = test\_set[i][1] print(predicted, actual)  if predicted == 'pos' and actual == 'pos': confusion\_matrix['tp'] += 1  elif predicted == 'neg' and actual == 'neg': confusion\_matrix['tn'] += 1  elif predicted == 'pos' and actual == 'neg': confusion\_matrix['fp'] += 1  else:  confusion\_matrix['fn'] += 1  print('Confusion Matrix:')  print('True Positives (TP): ' ,confusion\_matrix["tp"]) print('True Negatives (TN): ' ,confusion\_matrix["tn"]) print('False Positives (FP): ' ,confusion\_matrix["fp"]) print('False Negatives (FN): ' ,confusion\_matrix["fn"])  **Output**: |
| LAB PROGRAMS - 6Implement POS tagging using HMM import nltk  from nltk.corpus import brown from nltk.tag import hmm nltk.download('punkt')  sentences = brown.tagged\_sents()  trainer=hmm.HiddenMarkovModelTrainer() tagger = trainer.train(sentences)  text= " this is a sample sentence for POS tagging in python"  words=nltk.word\_tokenize(text)  tags=tagger.tag(words) for word,tag in tags:  print(f"{word}: {tag}") Output: |
| import nltk  from nltk.corpus import brown from nltk.tag import hmm  nltk.download('brown')  brown\_tagged\_sentences = brown.tagged\_sents(categories='news')  size = int(len(brown\_tagged\_sentences)\*0.9) train\_sentences = brown\_tagged\_sentences[:size] test\_sentences = brown\_tagged\_sentences[:size]  trainer = hmm.HiddenMarkovModelTrainer() tagger=trainer.train(train\_sentences)  print(tagger.accuracy(test\_sentences))  **Output**: |
| LAB PROGRAMS - 7Implement CKY parsing algorithm def print\_chart(chart,n): for p in range(n+1):  for q in range(n+1): print(chart[p][q],end="\t")  print()  def CKY\_PARSE(words, grammar): n = len(words) print(words,end="\n")  # Initialize parse table  table = [[set() for i in range(n+1)] for j in range(n+1)] for i in range(1,n+1):  word = words[i-1]  for lhs,rhs in grammar:  if rhs == (word,):  table[i-1][i].add(lhs) for j in range(2,n+1):  for i in range(j-2,-1,-1):  for k in range(i+1,j):  for lhs,rhs in grammar: if len(rhs) == 2:  if (rhs[0] in table[i][k] ) and ( rhs[1] in table[k][j] ): table[i][j].add(lhs)  return table  sentence = "the dog chased the cat" words = sentence.split()  n = len(words)  grammar = [('S', ('NP','VP')),  ('NP',('DET','NOMINAL')),  ('VP',('VERB','NP')),  ('NOMINAL',('cat',) ),  ('NOMINAL',('dog',) ),  ('VERB',('chased',)),  ('DET',('the',)) ]  # Perform CKY parsing  chart = CKY\_PARSE(words, grammar) print\_chart(chart,n)  # Check if the start symbol is in the final cell and output the result start\_symbol = 'S'  if start\_symbol in chart[0][n]:  print("The sentence is grammatically correct.") else:  print("The sentence is not grammatically correct.")  **Output:** |
| LAB PROGRAMS - 8 **Implement PCKY parsing algorithm**  def print\_chart(chart,n): for p in range(n+1):  for q in range(n+1): print('[',p,',',q,']',':',end = " ") for nt in non\_terminals:  if chart[p][q][nt]>0: print('{',nt,':',chart[p][q][nt],'}',end=" ")  print() print()  def PCKY\_PARSE(words, grammar): n = len(words) print(words,end="\n")  # Initialize parse table  table = [[dict() for i in range(n+1)] for j in range(n+1)] for i in range(n+1):  for j in range(n+1):  for nt in non\_terminals: table[i][j][nt]= 0.0  #Fill table cells  for j in range(1,n+1):  for lhs,rhs,pr in grammar: if rhs == (words[j-1],):  table[j-1][j][lhs] = pr  for i in range(j-2,-1,-1): for k in range(i+1,j):  for lhs,rhs,pr in grammar:  if len(rhs) == 2 and table[i][k][rhs[0]] > 0 and table[k][j][rhs[1]] > 0 :  if( table[i][j][lhs] < pr\*table[i][k][rhs[0]]\* table[k][j][rhs[1]] ):  table[i][j][lhs] = pr\*table[i][k][rhs[0]]\* table[k][j][rhs[1]]  return table  sentence = "the flight includes a meal" words = sentence.split()  n = len(words)grammar = [('S', ('NP','VP'),.80),  ('NP',('DET','NOMINAL'),.30),  ('VP',('VERB','NP'),.20),  ('NOMINAL',('meal',),.01 ),  ('NOMINAL',('flight',), .02),  ('VERB',('includes',),.05),  ('DET',('the',),.40),  ('DET',('a',),.40) ]  non\_terminals = ['S', 'NP','VP','DET','NOMINAL','VERB']  # Perform CKY parsing  chart = PCKY\_PARSE(words, grammar) print\_chart(chart,n)  # Check if the start symbol is in the final cell and output the result start\_symbol = 'S'  if chart[0][n]['S']>0:  print("The sentence is grammatically correct.") else:  print("The sentence is not grammatically correct.") OUTPUT: ['the', 'flight', 'includes', 'a', 'meal'] [ 0 , 0 ] :  [ 0 , 1 ] : { DET : 0.4 }  [ 0 , 2 ] : { NP : 0.0024 } [ 0 , 3 ] :  [ 0 , 4 ] :  [ 0 , 5 ] : { S : 2.3040000000000003e-08 }  [ 1 , 0 ] :  [ 1 , 1 ] :  [ 1 , 2 ] : { NOMINAL : 0.02 } [ 1 , 3 ] :  [ 1 , 4 ] :  [ 1 , 5 ] :  [ 2 , 0 ] :  [ 2 , 1 ] :  [ 2 , 2 ] :[ 2 , 3 ] : { VERB : 0.05 } [ 2 , 4 ] :  [ 2 , 5 ] : { VP : 1.2000000000000002e-05 } [ 3 , 0 ] :  [ 3 , 1 ] :  [ 3 , 2 ] :  [ 3 , 3 ] :  [ 3 , 4 ] : { DET : 0.4 }  [ 3 , 5 ] : { NP : 0.0012 } [ 4 , 0 ] :  [ 4 , 1 ] :  [ 4 , 2 ] :  [ 4 , 3 ] :  [ 4 , 4 ] :  [ 4 , 5 ] : { NOMINAL : 0.01 } [ 5 , 0 ] :  [ 5 , 1 ] :  [ 5 , 2 ] :  [ 5 , 3 ] :  [ 5 , 4 ] :  [ 5 , 5 ] :  The sentence is grammatically correct. LAB PROGRAMS - 9 **Compute cosine similarity between the words using termdocument matrix and term-term matrix**  import nltk import random import string import math  from nltk.corpus import brown  def freq(word,document):  d\_terms = extract\_words(document) fdist = nltk.FreqDist(w for w in d\_terms) returnfdist[word]  def extract\_words(document):  all\_terms\_list = brown.words(fileids=document)  only\_words\_list = [w.lower() for w in all\_terms\_list if w.isalpha()] stopwords\_list = nltk.corpus.stopwords.words('english')  final\_terms\_list = [w for w in only\_words\_list if w not in stopwords\_list ] return final\_terms\_list  doc\_names=['ca01','ca02','ca03','ca04'] vocab = set( extract\_words(doc\_names) ) vocab\_lenlen(vocab)  print("length of vocabulary = ", vocab\_len) #randomly selecting two words from vocabulary word1 = list(vocab)[random.randint(0,len(vocab))] word2 = list(vocab)[random.randint(0,len(vocab))] print("word1:",word1)  print("word-2:",word2)  #constructing vectors for the two vectors  word1\_vector = [ freq(word1,doc) for doc in doc\_names] word2\_vector = [ freq(word2,doc) for doc in doc\_names]  print("word-1-vector:",word1\_vector) print("word-2-vector:",word2\_vector)  #computing cosine similarity between word1 and word2 dot\_product = 0  for i in range(0,len(doc\_names)):  dot\_product = dot\_product + (word1\_vector[i]\*word2\_vector[i])  sum1 = sum2 = 0  for i in range(0,len(doc\_names)):  sum1 = sum1 + ( word1\_vector[i]\*word1\_vector[i] ) sum2 = sum2 + ( word2\_vector[i]\*word2\_vector[i] )  vector1\_len = math.sqrt(sum1) vector2\_len = math.sqrt(sum2)  cos\_theta = dot\_product / (vector1\_len\*vector2\_len)  print( "cos\_theta(",word1,",",word2,") = ",cos\_theta)  **OUTPUT:**  length of vocabulary = 2023 word-1: ignored  word-2: interest  word-1-vector: [0, 0, 1, 0]  word-2-vector: [2, 0, 0, 0] cos\_theta( ignored , interest ) = 0.0 |
| LAB PROGRAMS - 10 **Compute tf-idf matrix for the given document set**  import nltk import random import string import math  from nltk.corpus import brown def term\_freq(word,document):  d\_terms = extract\_words(document) fdist = nltk.FreqDist(w for w in d\_terms) return math.log10(fdist[word]+1)  def extract\_words(document):  all\_terms\_list = brown.words(fileids=document)  only\_words\_list = [w.lower() for w in all\_terms\_list if w.isalpha()] stopwords\_list = nltk.corpus.stopwords.words('english')  final\_terms\_list = [w for w in only\_words\_list if w not in stopwords\_list ] return final\_terms\_list  def idf(word): df = 0  for doc in doc\_names:  if word in extract\_words(doc): df = df+1  return math.log10( len(doc\_names)/df )  doc\_names=['ca01','ca02','ca03','ca04'] vocab = set( extract\_words(doc\_names) ) vocab\_len = len(vocab)  print("length of vocabulary = ", vocab\_len) #randomly selecting two words from vocabulary word1 = list(vocab)[random.randint(0,len(vocab))] word2 = list(vocab)[random.randint(0,len(vocab))] print("word1:",word1)  print("word-2:",word2)  #constructing vectors for the two vectors  word1\_vector = [ term\_freq(word1,doc)\*idf(word1) for doc in doc\_names] word2\_vector=[ term\_freq(word2,doc)\*idf(word2) for doc in doc\_names]  print("word-1-vector:",word1\_vector) print("word-2-vector:",word2\_vector)  #computing cosine similarity between word1 and word2 dot\_product = 0  for i in range(0,len(doc\_names)):  dot\_product = dot\_product + (word1\_vector[i]\*word2\_vector[i])  sum1 = sum2 = 0  for i in range(0,len(doc\_names)):  sum1 = sum1 + ( word1\_vector[i]\*word1\_vector[i] ) sum2= sum2 + ( word2\_vector[i]\*word2\_vector[i] )  vector1\_len = math.sqrt(sum1) vector2\_len = math.sqrt(sum2)  cos\_theta = dot\_product / (vector1\_len\*vector2\_len) print( "cos\_theta(",word1,",",word2,") = ",cos\_theta) **OUTPUT:**  length of vocabulary = 2023 word-1: speaker  word-2: problem  word-1-vector: [0.03761030733945982, 0.03761030733945982, 0.03761030733945982, 0.0]  word-2-vector: [0.03761030733945982, 0.03761030733945982, 0.0, 0.05961092677364148]  cos\_theta( speaker , problem ) = 0.5436003344535152 |
| LAB PROGRAMS - 11IMPLEMENT LANGUAGE MODEL USING FEEDFORWARD NEURAL NETWORK import tensorflow as tf  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences import numpy as np  # Toy dataset corpus = [  'This is a simple example', 'Language modeling is interesting', 'Neural networks are powerful',  'Feed-forward networks are common in NLP']  # Tokenize the text tokenizer = Tokenizer()  tokenizer.fit\_on\_texts(corpus)  total\_words = len(tokenizer.word\_index) + 1  # Create input sequences and labels input\_sequences = []  for line in corpus:  token\_list = tokenizer.texts\_to\_sequences([line])[0] for i in range(1, len(token\_list)):  n\_gram\_sequence = token\_list[:i+1] input\_sequences.append(n\_gram\_sequence)  max\_sequence\_length = max([len(x) for x in input\_sequences])  input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_length, padding='pre') X, y = input\_sequences[:, :-1], input\_sequences[:, -1]  y = tf.keras.utils.to\_categorical(y, num\_classes=total\_words)  # Build the model  model = tf.keras.Sequential([  tf.keras.layers.Embedding(total\_words, 50, input\_length=max\_sequence\_length-1), tf.keras.layers.Flatten(),  tf.keras.layers.Dense(100, activation='relu'), tf.keras.layers.Dense(total\_words, activation='softmax')])  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) # Train the model  model.fit(X, y, epochs=100, verbose=1)  # Generate text using the trained model seed\_text = "Neural networks" next\_words 5  for \_ in range(next\_words):  token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]  token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_length-1, padding='pre')  predicted = np.argmax(model.predict(token\_list), axis=-1) output\_word = ""  for word, index in tokenizer.word\_index.items(): if index == predicted:  output\_word = word break  seed\_text += " " + output\_word print(seed\_text)  **OUTPUT:**  Epoch 1/1001/1 [==============================] - ETA: 0s - loss: 2.8825 -  accuracy:0.1250  1/1 [========================] - 1s 897ms/step - loss: 2.8825 - accuracy: 0.1250 Epoch 100/100 1/1 [==============================] - ETA: 0s - loss: 0.0587 -  accuracy:1.0000  1/1 [==========================] - 0s 8ms/step - loss: 0.0587 - accuracy: 1.0000 Neural networks are powerful in nlp a |
| LAB PROGRAMS - 12IMPLEMENT LANGUAGE MODEL USING RNN. import tensorflow as tf  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences import numpy as np  # Toy dataset corpus = [  'This is a simple example', 'Language modeling is interesting', 'Neural networks are powerful',  'Recurrent neural networks capture sequences well'] # Tokenize the text  tokenizer = Tokenizer() tokenizer.fit\_on\_texts(corpus)  total\_words = len(tokenizer.word\_index) + 1 # Create input sequences and labels input\_sequences = []  for line in corpus:  token\_list = tokenizer.texts\_to\_sequences([line])[0] for i in range(1, len(token\_list)):  n\_gram\_sequence = token\_list[:i+1] input\_sequences.append(n\_gram\_sequence)  max\_sequence\_length = max([len(x) for x in input\_sequences])  input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_length, padding='pre')  X, y = input\_sequences[:, :-1], input\_sequences[:, -1]  y = tf.keras.utils.to\_categorical(y, num\_classes=total\_words) # Build the model  model = tf.keras.Sequential([  tf.keras.layers.Embedding(total\_words, 50, input\_length=max\_sequence\_length-1), tf.keras.layers.LSTM(100),  tf.keras.layers.Dense(total\_words, activation='softmax')])  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) # Train the model  model.fit(X, y, epochs=100, verbose=1)  # Generate text using the trained model seed\_text = "Recurrent neural networks" next\_words = 5  for \_ in range(next\_words):  token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]  token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_length-1, padding='pre') predicted = np.argmax(model.predict(token\_list), axis=-1)  output\_word = ""  for word, index in tokenizer.word\_index.items(): ifindex== predicted:  output\_word = word break  seed\_text += " " + output\_word print(seed\_text) OUTPUT: Epoch 1/100 1/1 [==============================] - ETA: 0s - loss: 2.8342 -  accuracy:0.0667  1/1 [==============================] - 1s 1s/step - loss: 2.8342 - accuracy: 0.0667 Epoch 100/100 1/1 [==============================] - ETA: 0s - loss: 0.3300 -  accuracy:1.0000  1/1 [==============================] - 0s 0s/step - loss: 0.3300 - accuracy: 1.0000  1/1 [==============================] - 0s 0s/step - loss: 0.3300 - accuracy: 1.0000  1/1 [==============================] - ETA:0s  1/1 [==============================] - 0s 336ms/step  1/1 [==============================] - ETA:  0s  1/1 [==============================] - 0s 16ms/step  1/1 [==============================] - ETA:  0s  1/1 [==============================] - 0s 8ms/step  1/1 [==============================] - ETA:  0s  1/1 [==============================] - 0s 16ms/step  Recurrent neural networks capture sequences well well well |
|  |