NLP UNIT 1 and 2
PART A
1.Define Natural language processing.

Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

2.List out the challenges of NLP.

NLP is a powerful tool with huge benefits, but there are still a number of Natural Language Processing limitations and problems:
1. Contextual words and phrases and homonyms
2. Synonyms
3. Irony and sarcasm
4. Ambiguity
5. Errors in text or speech
6. Colloquialisms and slang
7. Domain-specific language
8. Low-resource languages
9. Lack of research and development

3.What is NLTK? List few text processing operations using NLTK.

NLTK (Natural Language Toolkit) is a toolkit build for working with NLP in Python. It provides us various text processing libraries with a lot of test datasets. A variety of tasks can be performed using NLTK such as tokenizing, parse tree visualization etc.

4.Classify various N-gram language models.

An n-gram is a sequence of n words:
A **1-gram (unigram)** is a single word sequence of words like “please” or “ turn”.
A **2-gram or Bi-gram** is a two-word sequence of words like “please turn”, “turn your”, or “your homework”, and
A **3-gram (a trigram)** is a three-word sequence of words like “please turn your”, or “turn your homework”.

5. Differentiate Extrinsic Evaluation and Intrinsic Evaluation.

Extrinsic Evaluation
Extrinsic Evaluation of a N-gram language model is to use it in an application and measure how much the application improves.

Intrinsic Evaluation
An intrinsic evaluation metric is one that measures the quality of a model independent of any application.

6.Mention the applications of NLP.

- Translation Tools: Tools such as Google Translate, Amazon Translate, etc. translate sentences from one language to another using NLP.
- Chatbots: Chatbots can be found on most websites and are a way for companies to deal with common queries quickly.
- Virtual Assistants: Virtual Assistants like Siri, Cortana, Google Home, Alexa, etc can not only talk to you but understand commands given to them.

Other applications include : Targeted Advertising, Autocorrect, Speech Recognition, Sentiment Analysis.

7.Define Levenshtein Distance.

Weighted Edit Distance - Levenshtein Distance

The Levenshtein distance (a.k.a edit distance) is a measure of similarity between two strings. It is defined as the minimum number of changes required to convert string a into string b. Given two character strings s1 and s2, the *edit distance* between them is the minimum number of *edit operations* required to transform s1 into s2.

8.Compare the two types of spelling errors.

Various techniques that were designed on the basis of spelling errors and trends also called error patterns, and most notable studies on these were performed by Damerau. According to these studies spelling errors are generally divided into two types

- Typographic errors and
- Cognitive errors.

9.What is smoothing? List its types.

To keep a language model from assigning zero probability to unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen. This modification is called smoothing (or discounting).

There are many ways to do smoothing, and some of them are:

- Add-1 smoothing (Laplace Smoothing)
- Add-k smoothing,
- Backoff
- Kneser-Ney smoothing.

10.Write about backoff and interpolation.

In backoff, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram.

By contrast, in interpolation, we always mix the probability estimates from all the n-gram estimators, weighting and combining the trigram, bigram, and unigram counts.

PART B

1. Describe in detail various challenges and limitations of Natural language processing.

NLP is a powerful tool with huge benefits, but there are still a number of Natural Language Processing limitations and problems:

1. Contextual words and phrases and homonyms
2. Synonyms
3. Irony and sarcasm
4. Ambiguity
5. Errors in text or speech
6. Colloquialisms and slang
7. Domain-specific language
8. Low-resource languages
9. Lack of research and development

1.Contextual words and phrases and homonyms

The same words and phrases can have different meanings according the context of a sentence and many words – especially in English – have the exact same pronunciation but totally different meanings.

2. Synonyms

Synonyms can lead to issues similar to contextual understanding because we use many different words to express the same idea.

3. Irony and sarcasm

Irony and sarcasm present problems for machine learning models because they generally use words and phrases that, strictly by definition, may be positive or negative, but actually connote the opposite.

4. Ambiguity

Ambiguity in NLP refers to sentences and phrases that potentially have two or more possible interpretations. NLP has the following types of ambiguities

5. Spelling Errors in text

Misspelled or misused words can create problems for text analysis. Spelling mistakes can occur for a variety of reasons, from typing errors to extra spaces between letters or missing letters.

6. Colloquialisms and slang

Informal phrases, expressions, idioms, and culture-specific lingo present a number of problems for NLP – especially for models intended for broad use.

7. Domain-specific language

Different businesses and industries often use very different language. An NLP processing model needed for healthcare, for example, would be very different than one used to process legal documents.

8. Low-resource languages

AI machine learning NLP applications have been largely built for the most common, widely used languages.

9. Lack of research and development

Machine learning requires A LOT of data to function to its outer limits – billions of pieces of training data.

2. Summarize various word segmentation and sentence segmentation methods in NLP with appropriate example program.

Tokenization is splitting the raw text into small chunks of words or sentences, called tokens. If the text is split into words, then its called as 'Word Tokenization' and if it's split into sentences then its called as 'Sentence Tokenization'.

Word Tokenization
Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed.

NLTK Word Tokenize
Natural Language Toolkit (NLTK) is library written in python for natural language processing. NLTK has module word_tokenize() for word tokenization and sent_tokenize() for sentence tokenization.

Example
```
from nltk.tokenize import word_tokenize
text = """There are multiple ways we can perform tokenization on given text data. We can choose any method based on langauge, library and purpose of modeling."""
tokens = word_tokenize(text)
print(tokens)
```

Tokenization Using Regular Expressions(RegEx)
A regular expression is a sequence of characters that define a search pattern.Using RegEx we can match character combinations in string and perform word/sentence tokenization. We can use Python's re library for RegeEx related operations.

Example
```
import re
text = """There are multiple ways we can perform tokenization on given text data. We can choose any method based on langauge, library and purpose of modeling."""
tokens = re.findall("[\w]+", text)
print(tokens)
```

Sentence Segmentation
Sentence segmentation is another important step in text processing. The most use ful cues for segmenting a text into sentences are punctuation, like periods, question marks, and exclamation points. Question marks and exclamation points are relatively unambiguous markers of sentence boundaries. Periods, on the other hand, are more ambiguous

Sentence Tokenization using NLTK
sent_tokenize() module is used for sentence tokenization.
Example
```
from nltk.tokenize import sent_tokenize
text = """Characters like periods, exclamation point and newline char are used to separate the sentences. But one drawback with split() method, that we can only use one separator at a time! So sentence tonenization wont be foolproof with split() method."""
sent_tokenize(text)
```

Sentence Tokenization using RegEx
Example
```
import re
text = """Characters like periods, exclamation point and newline char are used to separate the sentences. But one drawback with split() method, that we can only use one separator at a time! So sentence tonenization wont be foolproof with split() method."""
```

```
tokens_sent = re.compile('[.!?] ').split(text)
tokens_sent
```

3. Explain with suitable example the minimum edit distance algorithm using dynamic programming.

Edit distance gives us a way to quantify both of these intuitions about string similarity. More formally, the minimum edit distance between two strings is defined as the minimum number of editing operations (operations like insertion, deletion, substitution) needed to transform one string into another.

Dynamic Programming Edit Distance - The Minimum Edit Distance Algorithm

Dynamic programming is the name for a class of algorithms, first introduced by Bellman (1957), that apply a table-driven method to solve problems by combining solutions to subproblems. Some of the most commonly used algorithms in natural language processing make use of dynamic programming.

The intuition of a dynamic programming problem is that a large problem can be solved by properly combining the solutions to various subproblems.

The minimum edit distance algorithm was named by Wagner and Fischer but independently discovered by many people.

Let's first define the minimum edit distance between two strings. Given two strings, the source string X of length n, and target string Y of length m, we'll define D[i; j] as the edit distance between X[1::i] and Y[1:: j], i.e., the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus D[n;m].

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

If we assume the version of Levenshtein distance in which the insertions and deletions each have a cost of 1 (ins-cost(.) = del-cost(.) = 1), and substitutions have a cost of 2 (except substitution of identical letters have zero cost), the computation for D[i; j] becomes:

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \\ D[i,j-1] + 1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
  D[0,0] = 0
  for each row i from 1 to n do
    D[i,0] ← D[i-1,0] + del-cost(source[i])
  for each column j from 1 to m do
    D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
        D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                      D[i−1,j−1] + sub-cost(source[i], target[j]),
                      D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

Figure 2.17 The minimum edit distance algorithm, an example of the class of dynamic programming algorithms. The various costs can either be fixed (e.g., $\forall x, \text{ins-cost}(x) = 1$) or can be specific to the letter (to model the fact that some letters are more likely to be inserted than others). We assume that there is no cost for substituting a letter for itself (i.e., $\text{sub-cost}(x,x) = 0$).

4. Elaborate various ways to perform smoothing.

To keep a language model from assigning zero probability to unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen. This modification is called smoothing (or discounting).

There are many ways to do smoothing, and some of them are:

- Add-1 smoothing (Laplace Smoothing)
- Add-k smoothing,
- Backoff
- Kneser-Ney smoothing.

Add-1 smoothing (Laplace Smoothing)

The simplest way to do smoothing is to add one to all the n-gram counts, before we normalize them into probabilities. All the counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on. This algorithm is called Laplace smoothing.

Laplace smoothing to unigram probabilities

The unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N

$$P(w_i) = \frac{c_i}{N}$$

Add-k smoothing

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k (.5? .05? .01?). This algorithm is called add-k smoothing.

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

Backoff and Interpolation

In backoff, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram.

By contrast, in interpolation, we always mix the probability estimates from all the n-gram estimators, weighting and combining the trigram, bigram, and unigram counts.

In simple linear interpolation, we combine different order n-grams by linearly interpolating them. Thus, we estimate the trigram probability $P(w_n|w_n\text{-}2w_n\text{-}1)$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n) \\ &\quad +\lambda_2 P(w_n|w_{n-1}) \\ &\quad +\lambda_3 P(w_n|w_{n-2}w_{n-1})\end{aligned}$$

PART C

5. Evaluate the different text processing methods used in NLTK and suggest the best method with proper justification.

NLTK is a standard python library with prebuilt functions and utilities for the ease of use and implementation. It is one of the most used libraries for natural language processing and computational linguistics.

By using NLTK various text processing can be done which include the following

- Tokenization
- Lower case conversion
- Stop Words removal
- Stemming
- Lemmatization
- Parse tree or Syntax Tree generation
- POS Tagging

Tokenization

Tokenization is the process of breaking text up into smaller chunks as per our requirements.

Word tokenization is the process of breaking a sentence into words. word_tokenize function has been used, which returns a list of words as output.

Sentence tokenization is the process of breaking a corpus into sentence level tokens. It's essentially used when the corps consists of multiple paragraphs. Each paragraph is broken down into sentences.

Stop Words Removal

Stop words are words which occur frequently in a corpus. e.g a, an, the, in. Frequently occurring words are removed from the corpus for the sake of text-normalization.

Stemming
It is reduction of inflection from words. Words with same origin will get reduced to a form which may or may not be a word. In our text we may find many words like playing, played, playfully, etc… which have a root word, play all of these convey the same meaning. So we can just extract the root word and remove the rest. Here the root word formed is called 'stem' and it is not necessarily that stem needs to exist and have a meaning. Just by committing the suffix and prefix, we generate the stems.
NLTK provides us with PorterStemmer LancasterStemmer and SnowballStemmer packages

Lemmatization
Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings to one word.

POS Tagging:
Part of Speech tagging is used in text processing to avoid confusion between two same words that have different meanings. With respect to the definition and context, we give each word a particular tag and process them. Two Steps are used here:
Tokenize text (word_tokenize).
Apply the pos_tag from NLTK to the above step.

6. b) Analyze the performance of N-gram models and justify the model which has the highest accuracy.
The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves.

Extrinsic Evaluation
Extrinsic Evaluation of a N-gram language model is to use it in an application and measure how much the application improves.
To compare two language models A and B:

- Use each of language model in a task such as spelling corrector, MT system.
- Get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly

Compare accuracy for A and B. The model produces the better accuracy is the better model.
Extrinsic evaluation can be time-consuming.

Intrinsic Evaluation
An intrinsic evaluation metric is one that measures the quality of a model independent of any application.
When a corpus of text is given and to compare two different n-gram models,

- Divide the data into training and test sets,
- Train the parameters of both models on the training set, and
- Compare how well the two trained models fit the test set.
    - Whichever model assigns a higher probability to the test set

Perplexity
The perplexity (sometimes called PPL for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words.

Minimizing perplexity is the same as maximizing probability

• The perplexity PP for a test set $W=w_1w_2...w_n$ is

$$\begin{aligned} \text{perplexity}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

We can use the chain rule to expand the probability of W:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

Perplexity of Unigram

The perplexity of a test set W depends on which language model we use. Here's the perplexity of W with a unigram language model (just the geometric mean of the unigram probabilities):

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}}$$

Perplexity of Bi-gram

The perplexity of W computed with a bigram language model is still a geometric mean, but now of the bigram probabilities,

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$