1Q)

Aim: POS Tagging: Hidden Markov Model: experiment is to calculate emission and transition matrix which will be helpful for tagging Parts of Speech using Hidden Markov Model.

Theory:

It is a process of converting a sentence to forms –list of words, list of tuples where each tuple is having a form(word, tag)).The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on. Default tagging is a basic step for the part-of-speech tagging. It is performed using the DefaultTagger class. The Default Tagger class takes 'tag' as a single argument. NN is the tag for a singular noun. DefaultTagger is most useful when it gets to work with most common part-of-speech tag. That's why a noun tag is recommended.

Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of- speech, semantic information and so on. Now, if we talk about Part-of-Speech (PoS) tagging, then it may be defined as the process of assigning one of the parts of speech to the given word. It is generally called POS tagging. In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub- categories.

Rule-based POS Tagging:

One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag. Disambiguation canal so be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words. For example, suppose if the preceding word of a word is article, then word must be a noun.

Stochastic POS Tagging:

Another technique of tagging is Stochastic POS Tagging. Now, the question that arises here is which model can be stochastic. The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger. The simplest stochastic tagger applies the following approaches for POS tagging –

Word Frequency Approach:

In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

Tag Sequence Probabilities:

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

Transformation-based Tagging:

Transformation based tagging is also called Brill tagging. It is an instance of the transformation- based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

It draws the inspiration from both the previous explained taggers – rule-based and stochastic. If we see similarity between rule-based and transformation tagger, then like rule-based, it is also based on the rules that specify what tags need to be assigned to what words. On the other hand, if we see similarity between stochastic and transformation tagger then like stochastic, it is machine learning technique in which rules are automatically induced from data.

HMM for POS Tagging:

The POS tagging process is the process of finding the sequence of tags which is most likely to have generated a given word sequence. We can model this POS process by using a Hidden 16Markov Model (HMM), where tags are the hidden states that produced the observable output, i.e., the words.

Code:

```
import nltk
nltk.download("averaged_perceptron_tagger")
nltk.download("punkt")
text=nltk.word_tokenize("And now for everything completely same")
nltk.pos_tag(text)
```

Output:

```
[('And', 'CC'),
('now', 'RB'),
('for', 'IN'),
('Everything', 'VBG'),
('completely', 'RB'),
('Same', 'JJ')]
```

Conclusion:

Thus, we have studied POS Tagging in the above experiment also learned regarding different types of POS Tagging and tried to implement the code for POS Tagging and successfully executed it.

2Q)

```
#exp 1

import nltk
from nltk.corpus import treebank
nltk.download('treebank')
ptb=treebank.parsed_sents()
print('Total sentences in penn treeebank corpus:',len(ptb))
first_sent_tree=ptb[0]
print('\n Parse tree of first sentence:',first_sent_tree)
subtree=first_sent_tree[0]
print('\n Subtree:',subtree)


#exp 2

import nltk
import random
import re
from nltk.tokenize import word_tokenize

patterns = [
  {
    "pattern": r"hi|hello|hey",
    "responses": ["Hello!", "Hi there!", "Hey!"]
  },
  {
    "pattern": r"how are you",
    "responses": ["I'm good, thank you!", "Feeling great, thank you!"]
  },
  {
```

```
        "pattern": r"bye|good bye",
        "responses": ["Goodbye!", "See you later, bye!"]
    }
]

def get_intent(text):
    for pattern in patterns:
        if re.search(pattern['pattern'], text.lower()):
            return pattern['responses']
    return ["I'm sorry, I don't understand that."]

print("Bot: Hi there! How can I help you today?")
while True:
    user_input = input("You: ")
    if user_input.lower() == 'exit':
        print('Bot: Goodbye!')
        break
    responses = get_intent(user_input)
    bot_response = random.choice(responses)
    print("Bot:", bot_response)
```

Output:

```
Total sentences in penn treeebank corpus: 3914

 Parse tree of first sentence: (S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))

 Subtree: (NP-SBJ
  (NP (NNP Pierre) (NNP Vinken))
  (, ,)
  (ADJP (NP (CD 61) (NNS years)) (JJ old))
  (, ,))
Bot: Hi there! How can I help you today?
You: hi
Bot: Hey!
You: exit
Bot: Goodbye!
```

3Q)

```
import re
text = "The quick brown fox jumps over the lazy dog"
variables = {
    "animals": r"(fox|dog)",
```

```
    "action": r"(jumps|runs)",
    "adjective": r"(quick|lazy|brown)"
}
variable_values = {}
for var, pattern in variables.items():
    match = re.search(pattern, text)
    if match:
        variable_values[var] = match.group()
print("Variable values extracted from text:")
for var, value in variable_values.items():
    print(f"{var}: {value}")
```