Experiment 6

NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
SUBJECT	**NLP Lab**

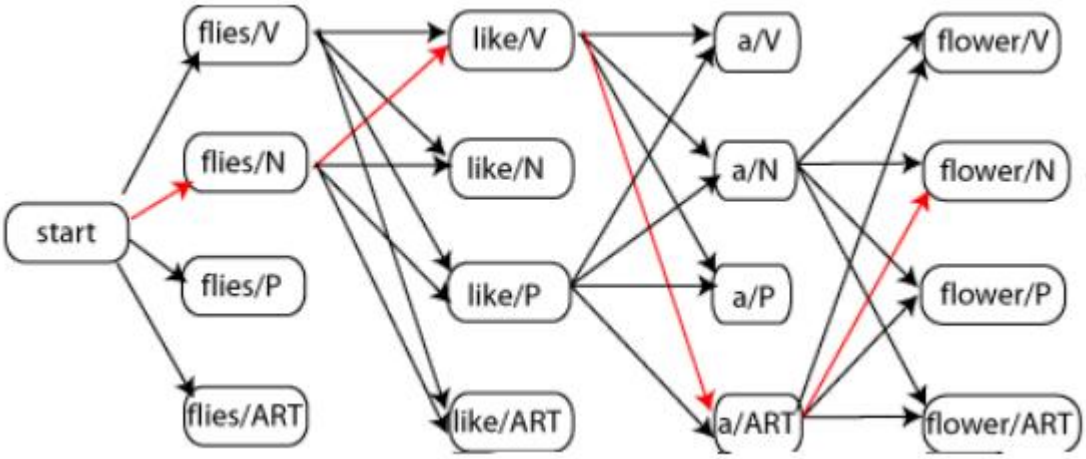
AIM: Find POS tags of words in a sentence using Viterbi decoding

THEORY:

In the previous experiment you calculated the transition and emission matrix, and now in this experiment it will be used to find the POS tag sequence for a given sentence. When we have an emission and transition matrix, various algorithms can be applied to find out the POS tags for words. Some of the possible algorithms are:

- Backward algorithm
- Forward algorithm
- Viterbi algorithm

Here, in this experiment, you can get familiar with Viterbi Decoding



What is Viterbi Decoding?

Viterbi Decoding is based on dynamic programming. This algorithm takes the emission and transmission matrix as the input. Emission matrix gives us information about probabilities of a POS tag for a given word and transmission matrix gives the probability of transition from one POS tag to another POS tag. It observes sequence of words and returns the state sequences of POS tags along with its probability.

The Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states called the Viterbi path. This results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models (HMM).

Viterbi decoding efficiently determines the most probable path from the exponentially many possibilities. It finds the highest probability given for a word against all our tags by looking through our transmission and emission probabilities, multiplying the probabilities, and then finding the max probability.

function VITERBI(*observations* of len T,*state-graph*) **returns** *best-path*

num-states ← NUM-OF-STATES(*state-graph*)
Create a path probability matrix *viterbi[num-states+2,T+2]*
viterbi[0,0] ← 1.0
for each time step t **from** 0 **to** T **do**
  for each state s **from** 0 **to** *num-states* **do**
    for each transition s' from s specified by *state-graph*
      new-score ← $viterbi[s, t] * a[s,s'] * b_{s'}(o_t)$
      if $((viterbi[s',t+1] = 0) \mid\mid (new\text{-}score > viterbi[s', t+1]))$
        then
          $viterbi[s', t+1] \leftarrow new\text{-}score$
          $back\text{-}pointer[s', t+1] \leftarrow s$
  Backtrace from highest probability state in the final column of *viterbi[]* and return path

Here "s" denotes words and "t" denotes tags. "a" is transmission matrix and "b" is emission matrix. Using the above algorithm, we have to fill the viterbi table column by column.

Example:

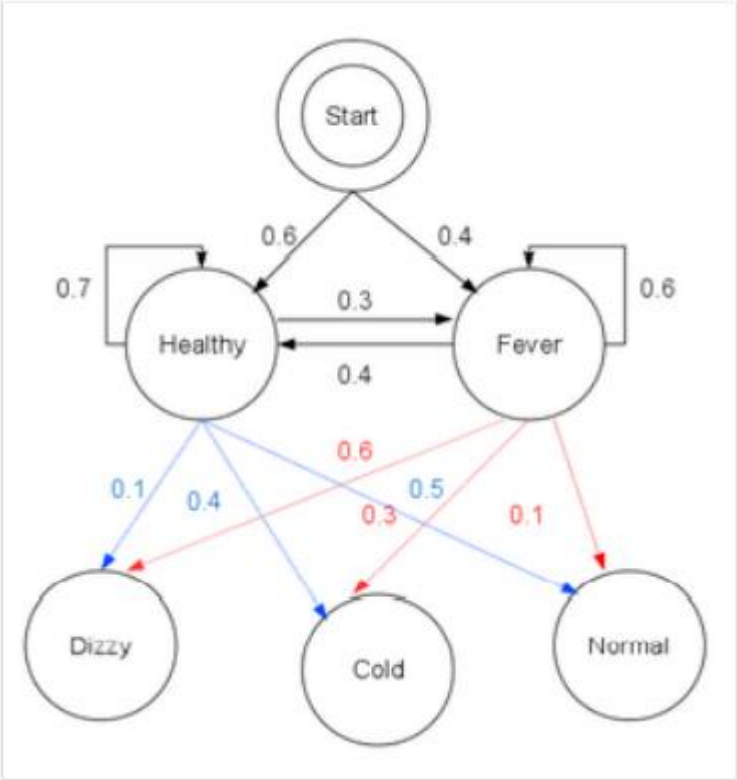
Consider a village where all villagers are either healthy or have a fever and only the village doctor can determine whether each has a fever. The doctor diagnoses fever by asking patients how they feel. The villagers may only answer that they feel normal, dizzy, or cold.

The doctor believes that the health condition of his patients operates as a discrete Markov chain. There are two states, "Healthy" and "Fever", but the doctor cannot observe them directly; they are hidden from him. On each day, there is a certain chance that the patient will tell the doctor he is "normal", "cold", or "dizzy", depending on his health condition.

The observations (normal, cold, dizzy) along with a hidden state (healthy, fever) form a hidden Markov model (HMM), and can be represented as follows in the Python programming language:

```
obs = ("normal", "cold", "dizzy")
states = ("Healthy", "Fever")
start_p = {"Healthy": 0.6, "Fever": 0.4}
trans_p = {
    "Healthy": {"Healthy": 0.7, "Fever": 0.3},
    "Fever": {"Healthy": 0.4, "Fever": 0.6},
}
emit_p = {
    "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1},
    "Fever": {"normal": 0.1, "cold": 0.3, "dizzy": 0.6},
}
```

In this piece of code, start_p represents the doctor's belief about which state the HMM is in when the patient first visits (all he knows is that the patient tends to be healthy). The particular probability distribution used here is not the equilibrium one, which is (given the transition probabilities) approximately {'Healthy': 0.57, 'Fever': 0.43}. The transition_p represents the change of the health condition in the underlying Markov chain. In this example, there is only a 30% chance that tomorrow the patient will have a fever if he is healthy today. The emit_p represents how likely each possible observation, normal, cold, or dizzy is given their underlying condition, healthy or fever. If the patient is healthy, there is a 50% chance that he feels normal; if he has a fever, there is a 60% chance that he feels dizzy.



The patient visits three days in a row and the doctor discovers that on the first day he feels normal, on the second day he feels cold, on the third day he feels dizzy. The doctor has a question: what is the most likely sequence of health conditions of the patient that would explain these observations? This is answered by the Viterbi algorithm.

IDE USED: Jupyter Notebook

CODE:

```
def viterbi(obs, states, start_p, trans_p, emit_p):
    V = [{}]
    for st in states:
        V[0][st] = {"prob": start_p[st] * emit_p[st] [obs[0]], "prev": None}

    for t in range(1, len(obs)):
        V.append({})
        for st in states:
            max_tr_prob = V[t - 1][states[0]] ["prob"] * trans_p[states[0]] [st]
            prev_st_selected = states[0]
            for prev_st in states[1:]:
                tr_prob = V[t - 1][prev_st] ["prob"] * trans_p[prev_st] [st]
                if tr_prob > max_tr_prob:
                    max_tr_prob = tr_prob
                    prev_st_selected = prev_st

            max_prob = max_tr_prob * emit_p[st] [obs[t]]
            V[t][st] = {"prob": max_prob, "prev": prev_st_selected}

    for line in table(V):
        print(line)

    opt = []
    max_prob = 0.0
    best_st = None


     # Get most probable state and its backtrack
     for st, data in V[-1].items():
         if data["prob"] > max_prob:
             max_prob = data["prob"]
             best_st = st
     opt.append(best_st)
     previous = best_st

     # Follow the backtrack till the first observation
     for t in range(len(V) - 2, -1, -1):
         opt.insert(0, V[t + 1] [previous] ["prev"])
         previous = V[t + 1] [previous] ["prev"]

     print ("\nThe steps of states with highest probability of %s" % max_prob + " are : " + " ".join(opt))

 def table(V):
     # Print a table of steps from dictionary
     yield "    " * 5 + "\t".join(("%3d" % i) for i in range(len(V)))
     for state in V[0]:
         yield "%.7s:  \t" % state + " ".join("%.7s" % ("%lf" % v[state]["prob"]) for v in V)
```

```
# INPUT
obs= ("normal", "cold", "dizzy")
states = ("Healthy", "Fever")
start_p = {"Healthy": 0.6, "Fever": 0.4}
trans_p = {
    "Healthy": {"Healthy": 0.7, "Fever": 0.3},
    "Fever": {"Healthy": 0.4, "Fever": 0.6},
}
emit_p = {
    "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1},
    "Fever": {"normal": 0.1, "cold": 0.3, "dizzy": 0.6},
}

viterbi(obs, states, start_p, trans_p, emit_p)
```

OUTPUT:

```
                 0        1       2
 Healthy:        0.30000 0.08400 0.00588
 Fever:          0.04000 0.02700 0.01512

 The steps of states with highest probability of 0.01512 are : Healthy Healthy Fever
```

REFERENCES:

1. https://en.wikipedia.org/wiki/Viterbi_algorithm
2. https://en.wikipedia.org/wiki/Hidden_Markov_model
3. https://www.youtube.com/watch?v=IqXdjdOgXPM&t=498s
4. https://www.youtube.com/watch?v=xejm-z3sbWA