The Viterbi Heuristic

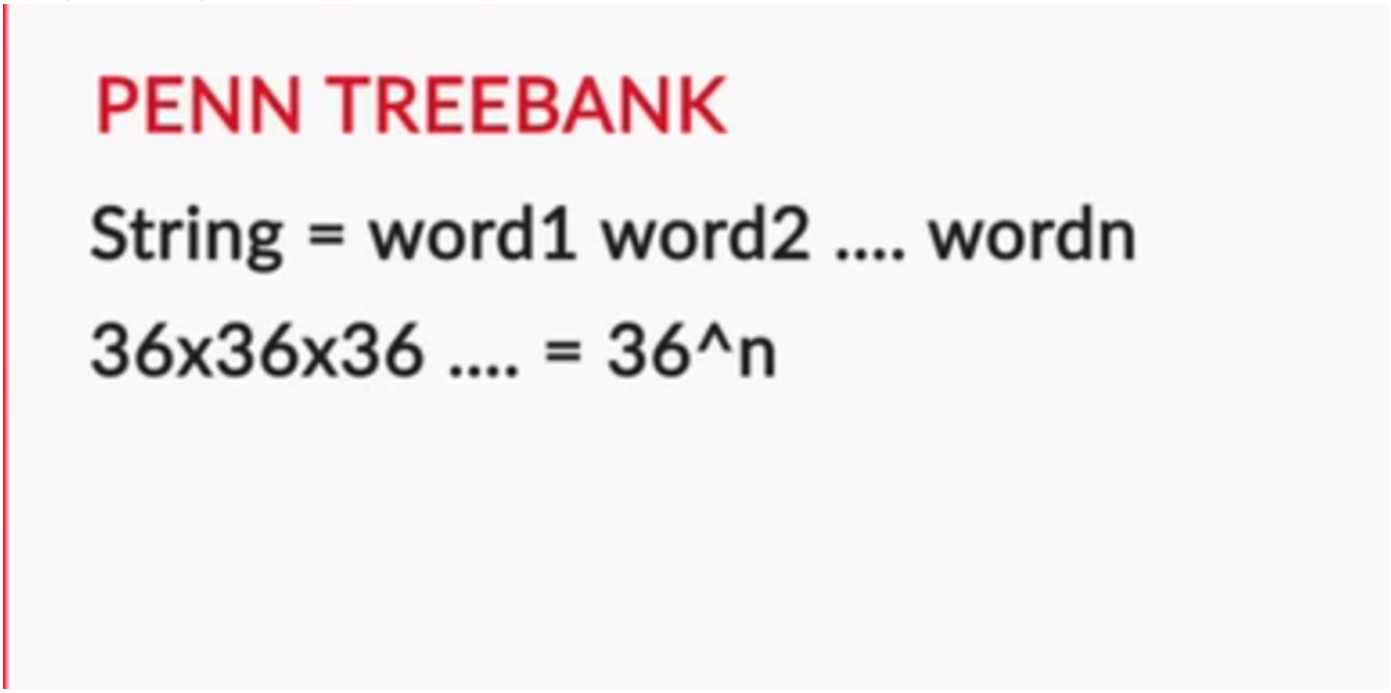
In the previous segment, you learnt how to calculate the probability of a tag sequence given a sequence of words. The idea is to compute the probability of all possible tag sequences and assign the sequence having the maximum probability.

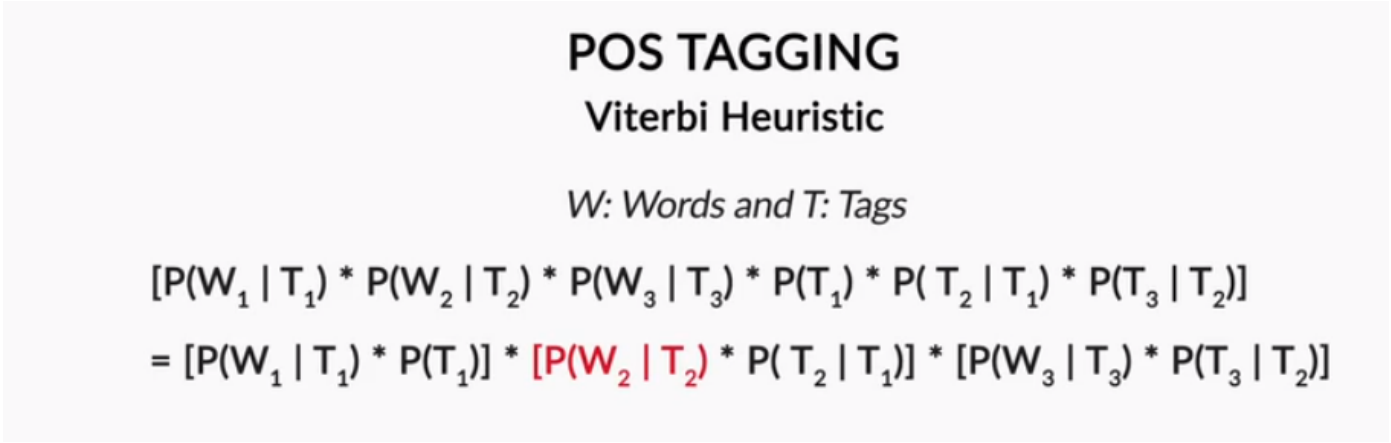
Although this approach can work in principle, it is computationally very expensive. For e.g. if you have just three POS tags - DT, JJ, NN, and you want to tag the sentence "The high cost", there are 3 (raised 3) = 27 possible tag sequences (each word can have three possible tags).

In general, for a sequence of n words and t tags, a total of t(raised)n tag sequences are possible. The Penn Treebank dataset in NLTK itself has 36 POS tags, so for a sentence of length say 10, there are 36(raised) 10 possible tag sequences (that's about three thousand trillion!).

Clearly, computing trillions of probabilities to tag a 10-word sentence is impractical. Thus, we need to find a much more efficient approach to tagging.

String = "The high cost"





VITERBI HEURISTIC

P(DT|start) = 0.4
P(VB|start) = 0.3
P(NN|start) = 0.1
P(JJ|start) = 0.2

Tag For "The"

P(DT|start)*P("The"|DT) = 0.4*0.6 = 0.24
P(NN|start)*P("The"|NN) = 0.1*0.2 = 0.02
P(JJ|start)*P("The"|JJ) = 0.2*0.1 = 0.02
P(VB|start)*P("The"|VB) = 0.3*0.1 = 0.03

P("The"|JJ) = 0.1, P("high"|JJ) = 0.4, P("cost"|JJ) = 0.2,
P("The"|NN) = 0.2, P("high"|NN) = 0.2, P("cost"|NN) = 0.3,
P("The"|DT) = 0.6, P("high"|DT) = 0.2, P("cost"|DT) = 0.2,
P("The"|VB) = 0.1, P("high"|VB) = 0.2, P("cost"|VB) = 0.3

Tag For "High"

P(DT|DT)*P("high"|DT) = 0.1*0.2 = 0.02
P(NN|DT)*P("high"|NN) = 0.4*0.2 = 0.08
P(JJ|DT)*P("high"|JJ) = 0.3*0.4 = 0.12
P(VB|DT)*P("high"|VB) = 0.2*0.1 = 0.02

Tag For "Cost"

P(DT|JJ)*P("cost"|DT) = 0.2*0.2 = 0.04
P(NN|JJ)*P("cost"|NN) = 0.5*0.3 = 0.15
P(JJ|JJ)*P("cost"|JJ) = 0.2*0.2 = 0.04
P(VB|JJ)*P("cost"|VB) = 0.1*0.3 = 0.03

To summarise, the basic idea of the Viterbi algorithm is as follows - given a list of observations (words) O1,O2....On to be tagged, rather than computing the probabilities of all possible tag sequences, you assign tags sequentially, i.e. assign the most likely tag to each word using the previous tag.

More formally, you assign the tag Tj to each word Oi such that it maximises the likelihood:

$$P(T_j|O_i) = P(Oi|T_j) * P(T_j|T_{j-1}) \,,$$

where T_{j-1} is the tag assigned to the previous word. Recall that according to the Markov assumption, the probability of a tag T_j is assumed to be **dependent only on the previous tag** T_{j-1}, and hence the term $P(T_j|T_{j-1})$.

In other words, you assign tags sequentially such that the tag assigned to every word O_i maximises the likelihood $P(T_j|O_i)$ locally, i.e. it is assumed to be dependent only on the current word and the previous tag. This algorithm does not guarantee that the resulting tag sequence will be the most likely sequence, but by making these simplifying assumptions, it reduces the computational time manifold (you'll see how much in the next lecture).

This is also why it is called a **greedy algorithm** - it assigns the tag that is most likely *at every word*, rather than looking for the overall most likely sequence. In the following video, Prof. Srinath will compare the computational costs of the two tagging algorithms - the brute force algorithm and the Viterbi algorithm.

In [ ]:

```
```