Foundations of NLP

CS3126

Lecture-14

POS Tagging, Sequence Labeling, Markov Chains, Hidden Markov Model, Viterbi Algorithm

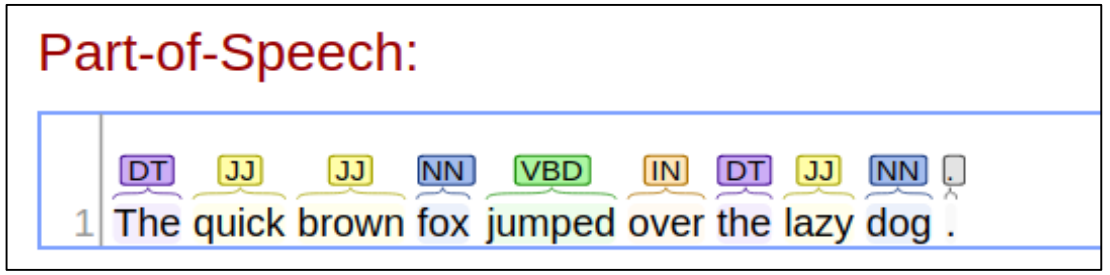
Recap

- NLP
- Applications
- Regular expressions
- Tokenization
- Stemming
    - Porter Stemmer
- Lemmatization
- Normalization
- Stopwords
- Bag-of-Words
- TF-IDF
- Probabilistic Classifiers (Naïve Bayes)
- **NER**
- **POS tagging**

POS-tagging

- Part-of-Speech (POS) tagging involves assigning specific grammatical categories or labels (such as nouns, verbs, adjectives, adverbs, pronouns, etc.) to individual words within a sentence.
- Advantages:
  - Provides insights into the syntactic structure of the text
  - Aiding in understanding word relationships
  - Disambiguating word meanings
  - Facilitating various linguistic and computational analyses of textual data

Demo - CoreNLP (stanfordnlp.github.io)



Types of POS tags

- Universal POS Tags
- Detailed POS Tags

Tag	Description
ADJ	Adjective
ADV	Adposition
ADP	Adverb
AUX	Auxiliary
CCONJ	Coordinating Conjunction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numeral
PART	Particle
PRON	Pronoun
PROPN	Proper Noun
PUNCT	Punctuation
SCONJ	Subordinating Conjunction
SYM	Symbol
VERB	Verb
X	Other

Detailed POS Tags

These tags are the result of the division of universal POS tags into various tags, like NNS for common plural nouns and NN for the singular common noun compared to NOUN for common nouns in English. These tags are language-specific.

```
1  import spacy
2  nlp=spacy.load('en_core_web_sm')
3
4  text='It took me more than two hours to translate a few pages of English.'
5
6  for token in nlp(text):
7   print(token.text, '=>',token.pos_,'=>',token.tag_)
```

```
It => PRON => PRP
took => VERB => VBD
me => PRON => PRP
more => ADJ => JJR
than => SCONJ => IN
two => NUM => CD
hours => NOUN => NNS
to => PART => TO
translate => VERB => VB
a => DET => DT
few => ADJ => JJ
pages => NOUN => NNS
of => ADP => IN
English => PROPN => NNP
. => PUNCT => .
```

In the above code sample, Load **spacy's en_web_core_sm** model and used it to get the POS tags.
You can see that the **pos_ returns** the universal POS tags, and **tag_ returns** detailed POS tags for words in the sentence.

But why do we need POS Tagging?

- These tags reveal a lot about a **word and its neighbors.**
- Gives an idea about syntactic structure (nouns are generally part of noun phrases), hence helping in **text parsing.**
- Parts of speech are useful features for labeling **named entities** like people or organizations (Mumbai in ‘Mumbai is the city of dreams)in information extraction, or for **coreference resolution.**
- A word’s part of speech can even play a role in **speech recognition** or synthesis, e.g., the word content is pronounced CONtent when it is a noun and conTENT when it is an adjective.

Markov Assumption

- More formally, consider a sequence of state variables $q_1,q_2,...,q_i$. A Markov model embodies the **Markov assumption** on the probabilities of this sequence: that Markov assumption when predicting the future, the past doesn't matter, only the present.

Markov Assumption: $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$

Markov Chains

A Markov chain is a model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set.
In the first chain, we have HOT, COLD & WARM as states & th**e decimal numbers represent the state transition (State1 →State2) probability** i.e there is a 0.1 probability of it being COLD tomorrow if today it is HOT.

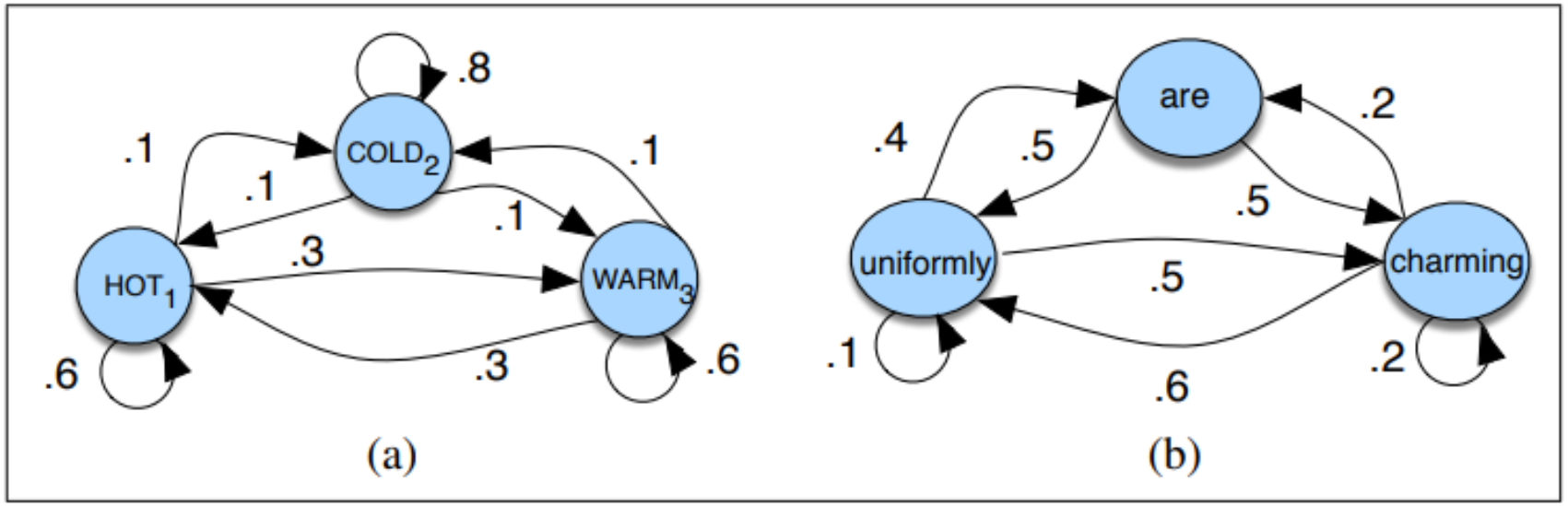
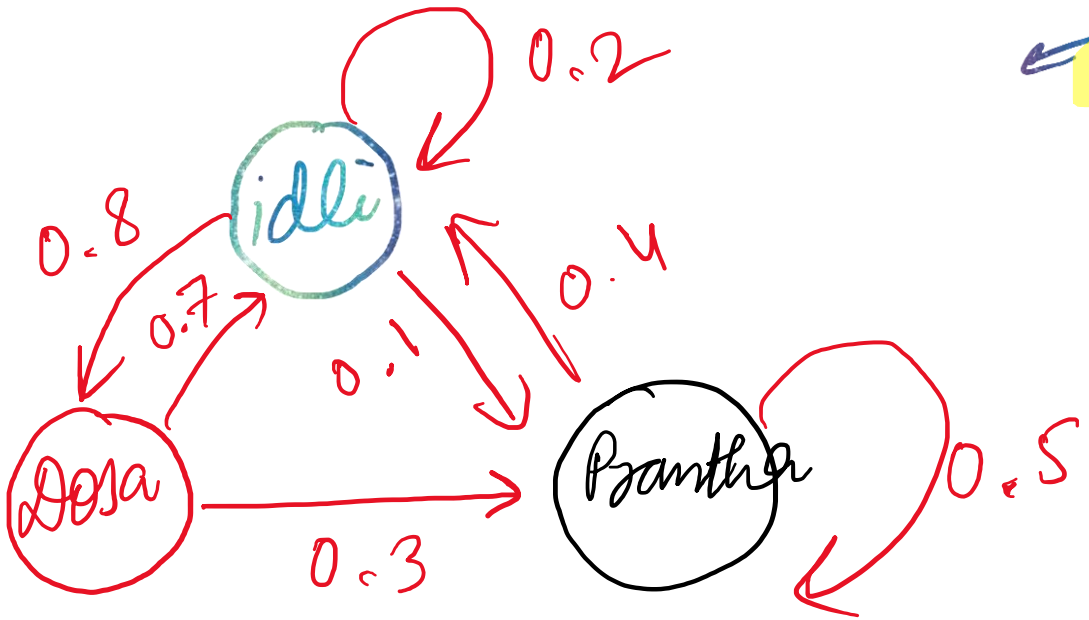


Figure A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Markov chains

MU mess

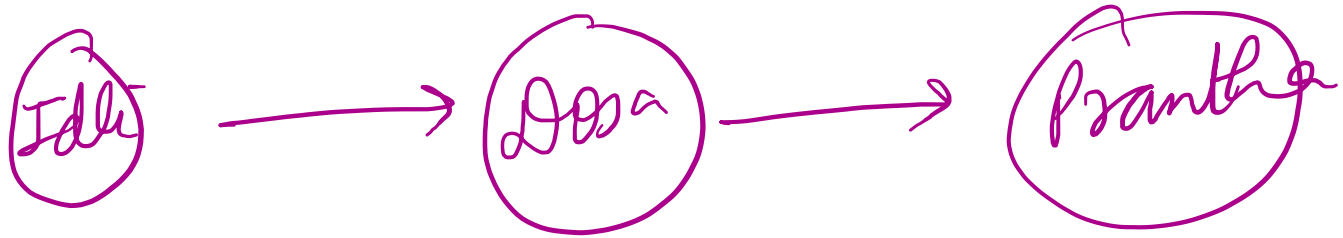


80% chance that tomorrow is dosa day, given today, idli day

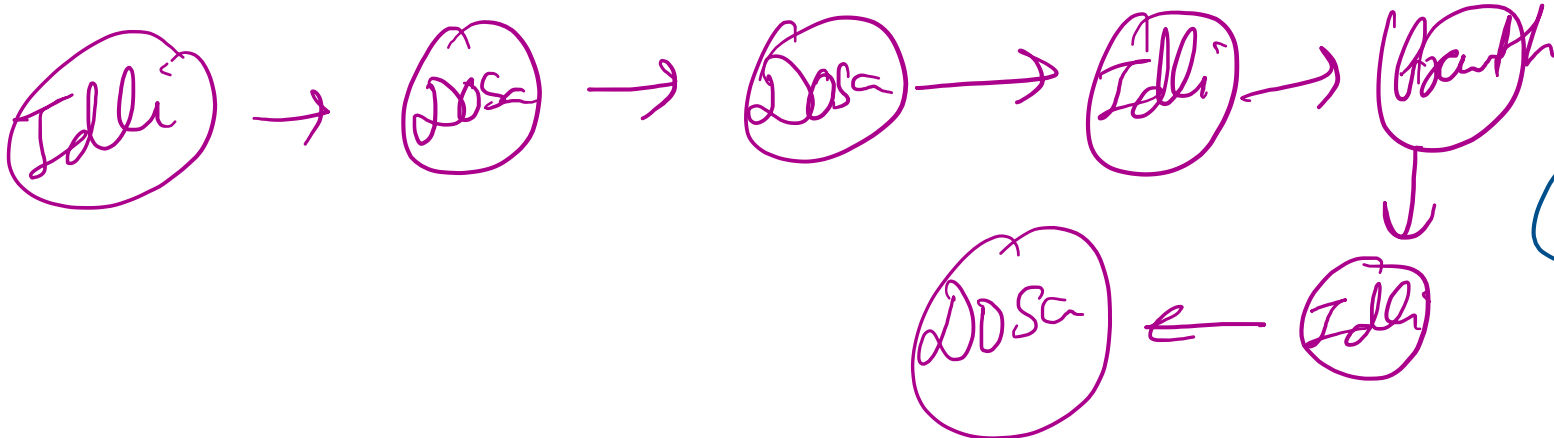
Any day will serve any of these three items!

Each Arrow is a transition from one state to another

Markov chains



Random
After 10 steps



1. State
2. Transition Probability
3. Depends only on previous day

(1) Find the probability of having Pranthas, Idli, Dosa's for this Markov chain

Sol^n Directed Graph

Create a adjacency matrix

Transition Matrix ⟶

	I	D	P
I	0.2	0.8	0.1
D	0.7	0	0.3
P	0.4	0	0.5

Π → initial state

MArkov Assumption

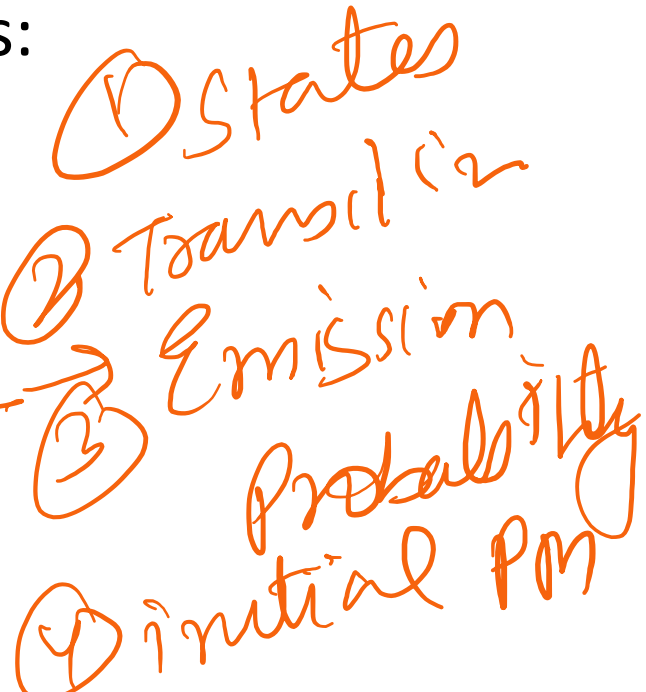
- Below are specified all the components of Markov Chains :

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

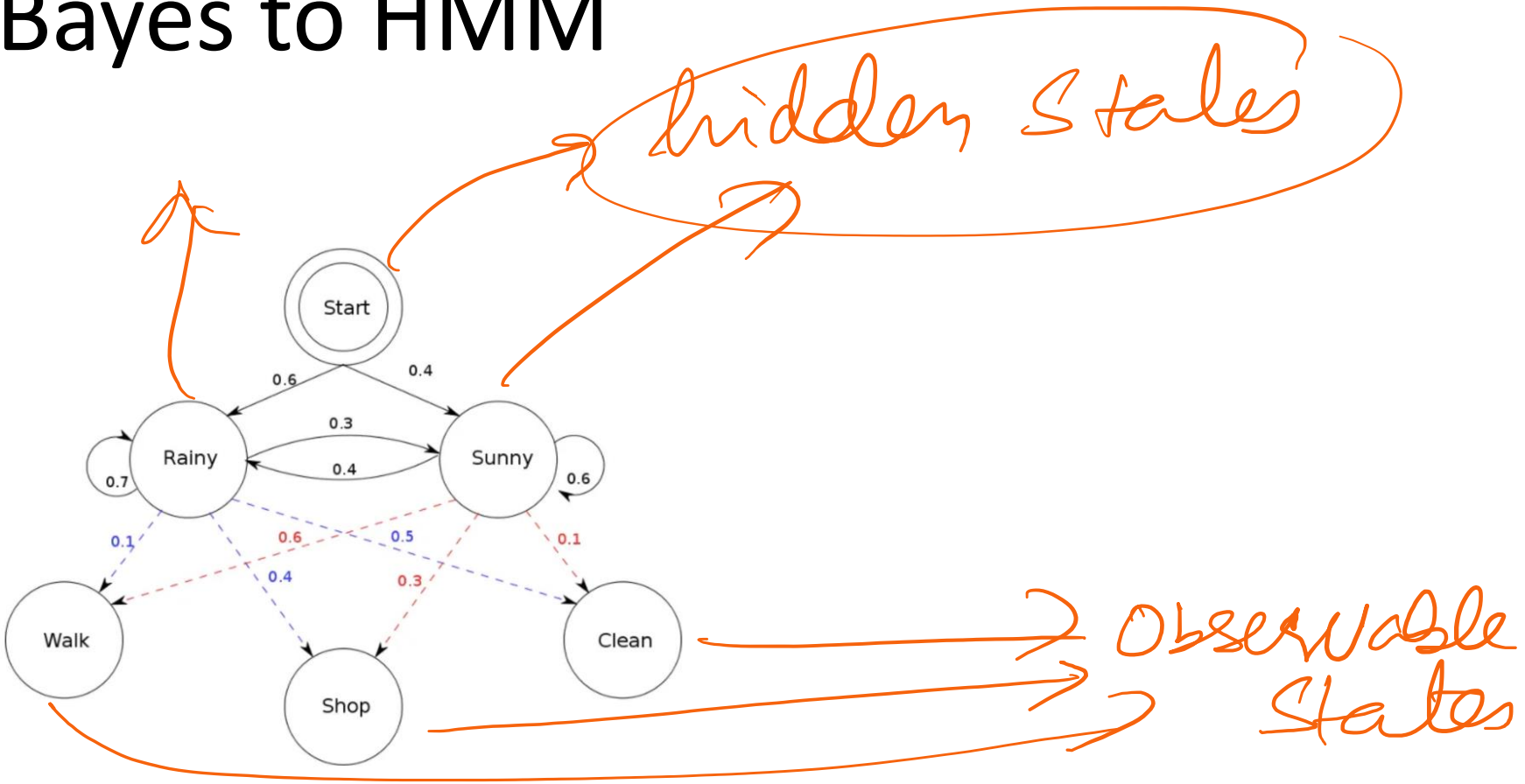
Hidden Markov Model

- We can have the **words in a sentence as Observable States (given to us in the data) but their POS Tags as Hidden states and hence we use HMM for estimating POS tags.** It must be noted that we call **Observable states 'Observation' & Hidden states 'States'.**
- A Hidden Markov Model has the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t (drawn from a vocabulary $V = v_1, v_2, ..., v_V$) being generated from a state q_i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$



From Naïve Bayes to HMM



In the above HMM, we are given Walk, Shop & Clean as observable states. But we are more interested in tracing the sequence of the hidden states that will be followed which are Rainy & Sunny.

A sample HMM with both ‘A’ & ‘B’ matrices will look like this :

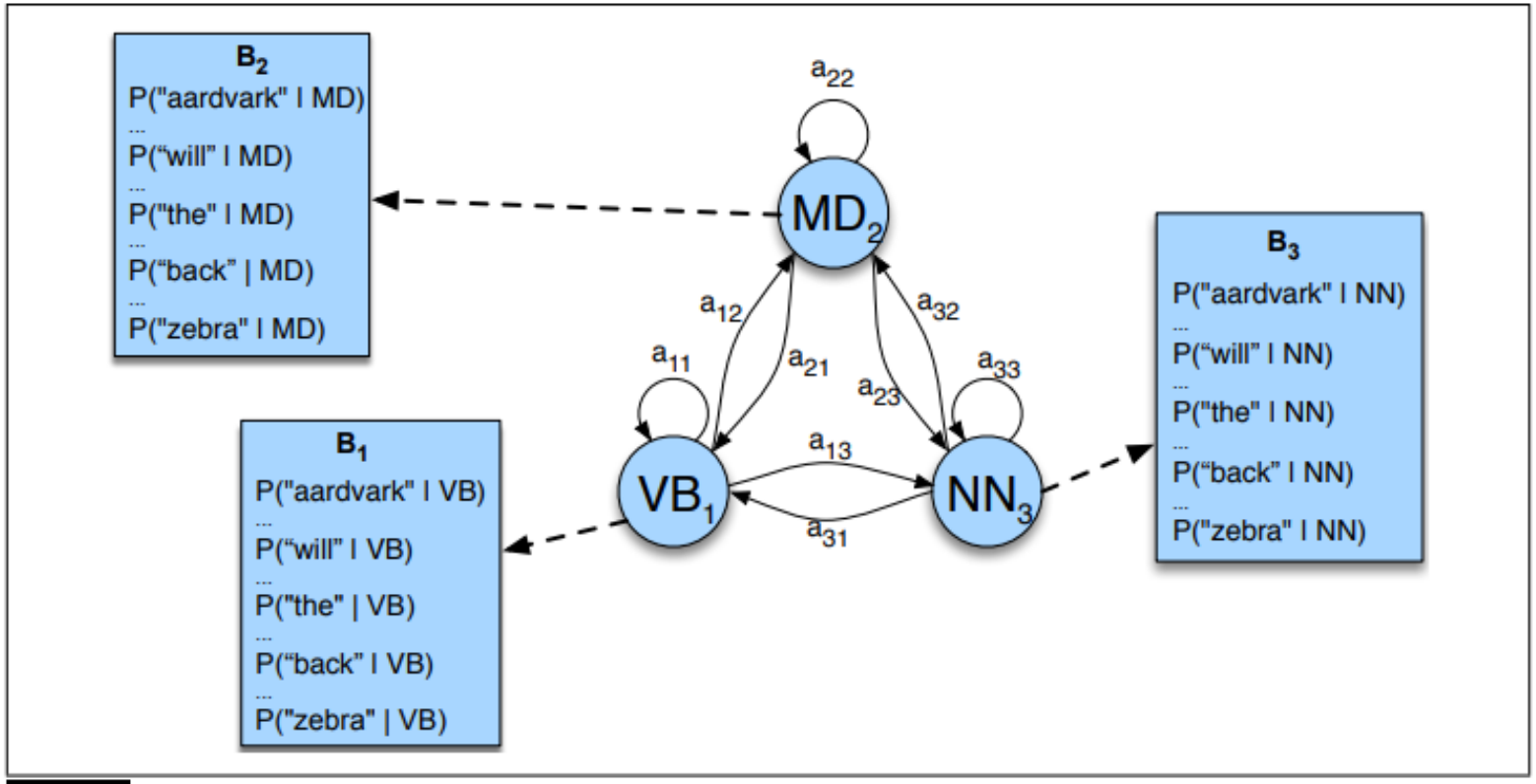


Figure An illustration of the two parts of an HMM representation: the *A* transition probabilities used to compute the prior probability, and the *B* observation likelihoods that are associated with each state, one likelihood for each possible observation word.

Probability Matrix

Q: Set of possible Tags

A: The *A* matrix contains the tag transition probabilities $P(ti|ti-1)$ which represent the probability of a tag occurring given the previous tag. Example: Calculating A[Verb][Noun]:

P (Noun|Verb): Count(Noun & Verb)/Count(Verb)

Q: Sequence of observation (words in the sentence)

B: The *B* emission probabilities, $P(wi|ti)$, represent the probability, given a tag (say Verb), that it will be associated with a given word (say Playing). The emission probability B[Verb][Playing] is calculated using:

P(Playing | Verb): Count (Playing & Verb)/ Count (Verb)

It must be noted that we get all these Count() from the corpus itself used for training.

Probability Matrix

- We need to set up a **probability matrix called lattice** where we have **columns as our observables** (words of a sentence in the same sequence as in sentence) & **rows as hidden states**(all possible POS Tags are known). For the sentence :
- **'Janet will back the bill'** has the below lattice:

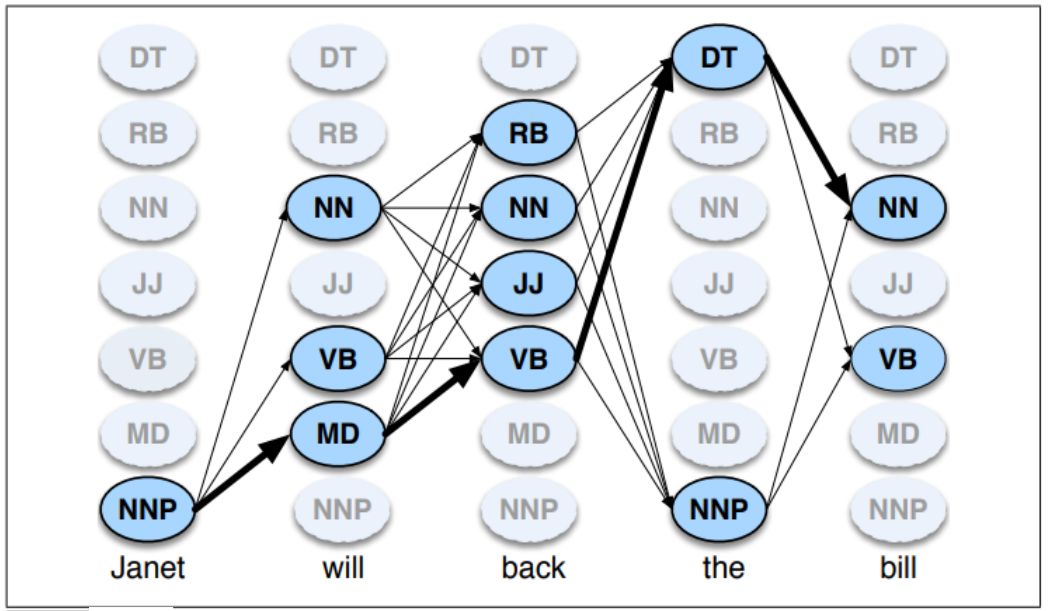


Figure A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Viterbi Algorithm

The decoding algorithm for HMMs is the **Viterbi algorithm.**

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

Each cell of the lattice, $v_t(j)$, represents the probability that the HMM is in state *j* after seeing the first *t* observations and passing through the most probable state sequence $q_1,\ldots,q_{t-1}$, given the HMM λ.
The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell.

(Read example from Book)

Viterbi Algorithm

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

create a path probability matrix *viterbi*$[N,T]$
for each state s **from** 1 **to** N **do** ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

	DT	NN	VB
START	0.8	0.2	0
DT	0	0.9	0.1
NN	0	0.5	0.5
VB	0.5	0.5	0

A = → Transition

B = ↗ Emission.

	THE	BANK	GIVES	LOAN
DT	0.2	0	0	0
NN	0	0.1	0.3	0.1
VB	0	0.2	0.15	0.8

"The Bank gives loan". What is the most likely sequence of POS for this sentence using Viterbi Algo^m.

	DT	NN	VB
START	0.8	0.2	0
DT	0	0.9	0.1
NN	0	0.5	0.5
VB	0.5	0.5	0

→ Transition

A =

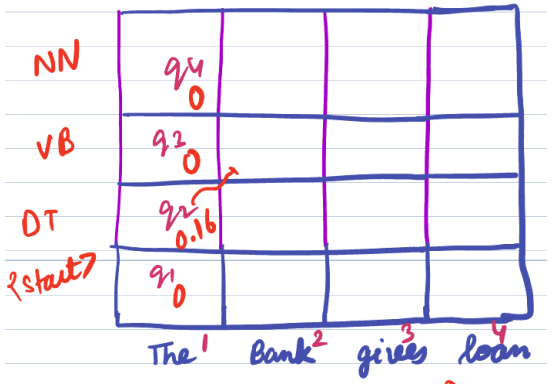
↗ Emission.

B =

	THE	BANK	GIVES	LOAN
DT	0.2	0	0	0
NN	0	0.1	0.3	0.1
VB	0	0.2	0.15	0.8

"The Bank gives loan". What is the most likely sequence of POS for this sentence using Viterbi Algo^m.

π = [DT 0.8, NN 0.2, VB 0]



Each state (DT, VB, NN)

$V_{The}(i, \text{state}) = \text{Transition Prob} \times \text{Emission Prob.}$

$V_{The}(1, DT) = P(\text{start} \to DT) \times P(The \mid DT)$

$= 0.8 \times 0.2$

$= 0.16$

$V_{The}(1, NN) = P(\text{start} \to NN) \times P(The \mid NN)$

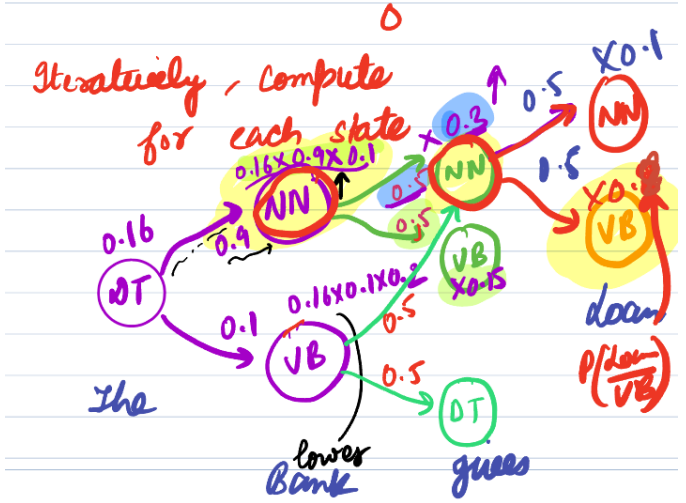
0.2×0

$= 0$

$V_{The}(1, VB) = P(\text{start} \to VB) \times P(The \mid VB)$

0×0

$=$



DT → NN → NN → VB

Yellow highlighted is solution / most likely sequence!

DT	0.16	0	0	0
NN	0	0.0144	0.00216	0.000108
VB	0	0.0032	0.00108	0.0008064

The Bank gives loan

Viterbi Algorithm

We represent the most probable path by taking the maximum over all possible previous state sequences max $q_1,...,q_{t-1}$.
Like other dynamic programming algorithms, Viterbi fills each cell recursively!!

$$v_t(j) = \max_{q_1,\cdots,q_{t-1}} P(q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda)$$

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

Complexity of Viterbi Algorithm

- K states y_1,……,y_K, observation sequence of length T
- $O(K^2T)$

Limitations

- Unknown words
- It’s hard for generative models like HMMs to add arbitrary features directly into the model in a clean way

Acknowledgments

- These slides were adapted from the book SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition
- Many of these slides are derived from Tom Mitchell, William Cohen, Eric Xing and Seyoung Kim. Thanks!