Natural Language Processing - 2nd Semester (2024-2025)
1038141

1.8 - Part-of-Speech tagging

SAPIENZA
UNIVERSITÀ DI ROMA

Prof. Stefano Faralli
faralli@di.uniroma1.it

Prof. Iacopo Masi
masi@di.uniroma1.it

**credits are reported in the last slide

SAPIENZA
UNIVERSITÀ DI ROMA

1.8 - Part-of-Speech tagging

- PoS tag, Closed classes and Open classes, Universal PoS tagset, mappings
- PoS taggers, ambiguity
- Rule-based PoS taggers, Stochastic PoS taggers
- Markov Models vs. Hidden Markov Models
- Training a HMM tagger, decoding a sequence with the Viterbi Algorithm
- Maximum entropy models, Logistic Regression models
- Exercises
- Q&A

Milestones in NLP

today topic is fundamental for the three areas.

rule-based systems

statistical classical machine learning models

deep learning models

What are Part-of-Speech tags

definition:

Part-of-Speech (PoS) is a historical classification of words into classes (since Thrax of Alexandria, ~100 b.C.) included:

-noun, verb, pronoun, preposition, adverb, conjunction, participle and article

Does it hold for all languages?

Does it hold for all languages?

- Some languages do not have adjectives
  - e.g., the American language Lakhota
- Some languages have no articles
  - e.g., Latin, Sanskrit, Persian

Word classes / Parts Of Speech (POS) / Lexical Tags

Q: Why do we need **word classes**?

A1: They give important information about the word and its neighbours

He is running the race

He decided to race for the job

Word classes / Parts Of Speech (POS) / Lexical Tags

Q: Why do we need **word classes**?

A2: Useful for recognizing speech and correcting spelling errors:

what is likely to come after an adjective?

a verb? a preposition? a noun?

Word classes / Parts Of Speech (POS) / Lexical Tags

Q: Why do we need **word classes**?

A3: Useful for understanding text, a.k.a. Natural Language Understanding (NLU):

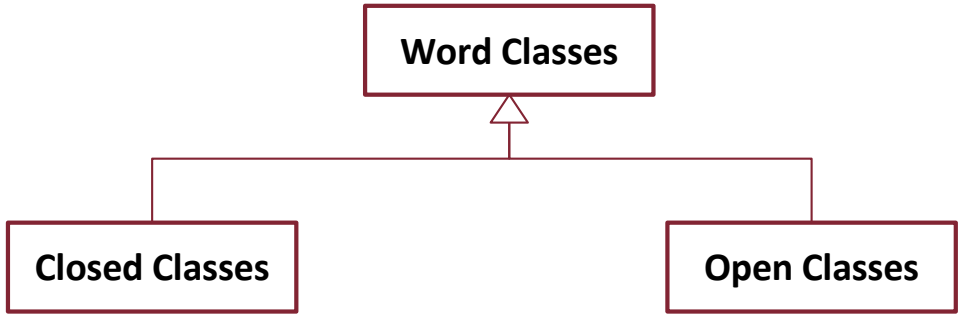
old plane

existential plane

inclined plane

two-dimensional plane

Closed classes vs. Open classes



- Closed classes: a fixed set of words,very unlikely that new words of this kind are coined
  – pronouns, prepositions, conjunctions, interjections, articles

- Open classes: open to neologisms, newly coined words, borrowed from other languages, etc.
  – nouns, verbs, adjectives, adverbs

How many word classes?

The Penn Treebank part-of-speech tags has 45 classes:

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two, three*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb, base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb, past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb, gerund	*eating*
JJ	adjective	*yellow*	VBN	verb, past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb, non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb, 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, singular	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

[Santorini, 1990]

Example of PoS tags

The first story is about connecting the dots .

Example of PoS tags

The first story is about connecting the dots .

With the classical 8 word classes:

*The***/art** *first***/a** *story***/n** *is***/v** *about***/p** *connecting***/v** *the***/art** *dots***/n** .

Example of PoS tags

The first story is about connecting the dots .

With the classical 8 word classes:

The/**art** *first*/**a** *story*/**n** *is*/**v** *about*/**p** *connecting*/**v** *the*/**art** *dots*/**n** .

With the Penn Treebank dataset:

The/**DT** *first*/**JJ** *story*/**NN** *is*/**VBZ** *about*/**IN** *connecting*/**VBG** *the*/**DT** *dots*/**NNS** .

A Universal PoS tagset

Developed by Google Research

Best paper (ACL 2011)

https://github.com/slavpetrov/universal-pos-tags

[Das and Petrov, 2011]

A Universal PoS tagset

The 17 **universal tags** are:

- **VERB** -verbs (all tenses and modes)
- **NOUN** -nouns (common and proper)
- **PROPN** –proper nouns
- **PRON** –pronouns
- **AUX** -auxiliaries
- **ADJ** -adjectives
- **ADV** -adverbs
- **ADP** -adpositions (prepositions and postpositions)
- **INTJ** –interjection (exclamations)
- **CCONJ** –coordinating conjunctions (and, or, etc.)
- **SCONJ** –subordinating conjunctions (that, if, when, since)
- **DET** -determiners
- **NUM** -cardinal numbers
- **PART** -particles or other function words
- **PUNCT** –punctuation marks
- **SYM** -$ (substitutable by "dollar")
- **X** -other: foreign words, typos, abbreviations

"...this set of coarse-grained POS categories is defined operationally, by collapsing language (or treebank) specific distinctions to a set of categories that exists across all languages...."

[Das and Petrov, 2011]

Example of PoS tagged English sentence

sentence:	The	oboist	Heinz	Holliger	has	taken	a	hard	line	about	the	problems	.
original:	DT	NN	NNP	NNP	VBZ	DT	JJ	NN	NN	IN	DT	NNS	.
universal:	DET	NOUN	NOUN	NOUN	VERB	DET	ADJ	NOUN	NOUN	ADP	DET	NOUN	.

[Das and Petrov, 2011]

Mapping between Italian and Universal Tags

A	ADJ	DR	DET	PI	PRON	RI	DET
AP	ADJ	DT	DET	PP	PRON	S	NOUN
B	ADV	E	ADP	PQ	PRON	SA	X
C	CONJ	I	X	PR	PRON	SP	NOUN
DD	DET	N	NUM	PT	PRON	SW	NOUN
DE	DET	NO	NUM	PU	.	V	VERB
DI	DET	PD	PRON	RD	DET	X	X

Other mappings are available:

https://github.com/slavpetrov/universal-pos-tags

[Das and Petrov, 2011]

Part-of-Speech Tagger

It consists of assigning a part-of-speech tag to each word in a text automatically:

Ambiguity

- Unfortunately, the same word can be tagged with different POS tags:
  - How to increase the water pressure from a **well**?
  - Tears **well** in her eyes
  - The wound is nearly **well**
  - The party went **well**
- Part-of-Speech tagging is a disambiguation task!

Ambiguity - the Brown corpus

The Brown corpus is a million-word corpus of modern American texts

http://icame.uib.no/brown/bcm.html

# ambiguous tags	# word types
Unambiguous (1-tag)	35340
Ambiguous (2-7 tags)	4,100
2 tags	3760
3 tags	264
4 tags	61
5 tags	12
6 tags	2
7 tags	1

11.5% of ambiguous word types, **BUT**: 40% of ambiguous tokens

Rule-based PoS tagging (since 1960)

1) Select a lexicon which provides a list of parts of speech for each word

1) Use hand-written disambiguation rules

example:

RULE: NON-VERB-RULE
INPUT: a word $\mathbf{w_i}$ with it's list of possible tags
// if previous word $\mathbf{w_{i-1}}$ is an article or an adjective
if(PoS($\mathbf{w_{i-1}}$)== ART || PoS($\mathbf{w_{i-1}}$)==ADJ)
then remove VERB from the list of possible tags for the word

Stochastic Part-of-Speech Tagging (since 1970s)

Stochastic POS tagging uses probabilities to tag:

- **Idea:** use Hidden Markov Models to select the most-likely tag sequence for a given sequence of words:

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max} P(t_1^n \mid w_1^n)$$

- But how can we calculate these probabilities?

Stochastic Part-of-Speech Tagging (since 1970s)

Do you remember the Bayes' theorem?

$$P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)}$$

$$\hat{t}_1^n = \arg\max_{t_1^n \in Tagset^n} P(t_1^n \mid w_1^n)$$

$$\hat{t}_1^n = \arg\max_{t_1^n \in Tagset^n} \frac{P(w_1^n \mid t_1^n)P(t_1^n)}{P(w_1^n)} \approx \arg\max_{t_1^n \in Tagset^n} P(w_1^n \mid t_1^n)P(t_1^n)$$

likelihood prior

by dropping the denominator

Stochastic Part-of-Speech Tagging (since 1970s)

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max}\, P(w_1^n \mid t_1^n) P(t_1^n)$$

likelihood

prior

still hard to compute

Stochastic Part-of-Speech Tagging (since 1970s)

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max}\, P(w_1^n \mid t_1^n) P(t_1^n)$$

likelihood prior

First assumption:

The probability of a word **depends only on its own part-of-speech tag**

$$P(w_1^n \mid t_1^n) = P(w_1 \mid t_1^n) P(w_2 \mid w_1, t_1^n) \ldots P(w_n \mid w_1^{n-1}, t_1^n) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

chain rule

$$P(w_1^n \mid t_1^n) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

Stochastic Part-of-Speech Tagging (since 1970s)

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max}\, P(w_1^n \mid t_1^n) P(t_1^n)$$

likelihood prior

Second assumption:

The probability of a tag appearing depends only on the previous tag (**bigram assumption**)

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

Stochastic Part-of-Speech Tagging (since 1970s)

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max}\, P(w_1^n \mid t_1^n) P(t_1^n)$$

likelihood

prior

first assumption

$$P(w_1^n \mid t_1^n) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

second assumption

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

Stochastic Part-of-Speech Tagging (since 1970s)

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

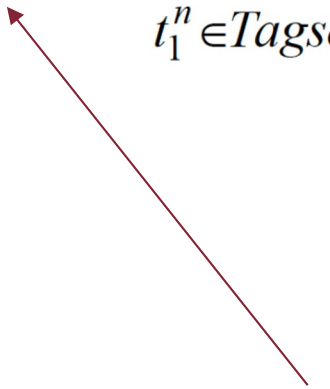
Now we can easily estimate these two probabilities from a part-of-speech tagged corpus with:

$$P(t_i \mid t_{i-1}) = \frac{c(t_{i-1}, t_i)}{c(t_{i-1})}$$

$$P(w_i \mid t_i) = \frac{c(t_i, w_i)}{c(t_i)}$$

Stochastic Part-of-Speech Tagging (since 1970s)

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$



We know how to estimate the likelihood and the prior.

How to find the global PoS tag sequence which maximizes the product of likelihood by prior?

Can we use a Markov Models ?

Markov Chains are probabilistic models for which the markov assumption is valid:

$$P(x_n)=P(x_n \mid x_1\, x_2 \ldots x_{n-1}) = P(x_n \mid x_{n-1})$$

in NLP is referred as the bigram assumption

Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	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, $\sum_{i=1}^{n} \pi_i = 1$

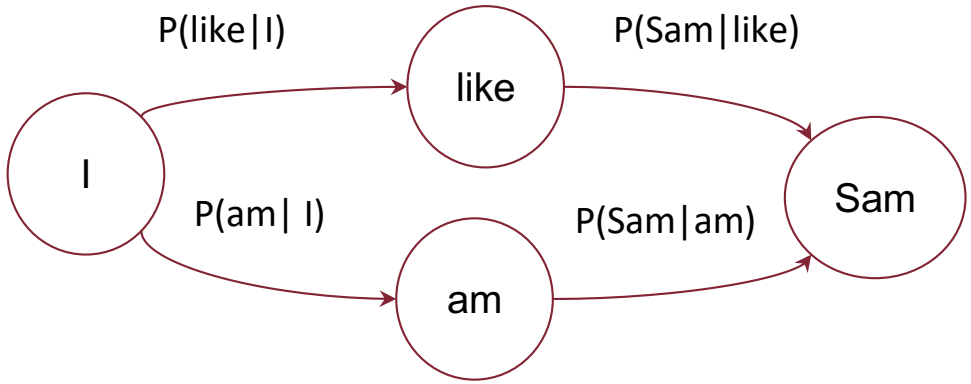
Can we use a Markov Models ?

Markov Chains are probabilistic models for which the markov assumption is valid:

$$P(x_n)=P(x_n \mid x_1\ x_2 \ldots x_{n-1}) = P(x_n \mid x_{n-1})$$

in NLP is referred as the bigram assumption

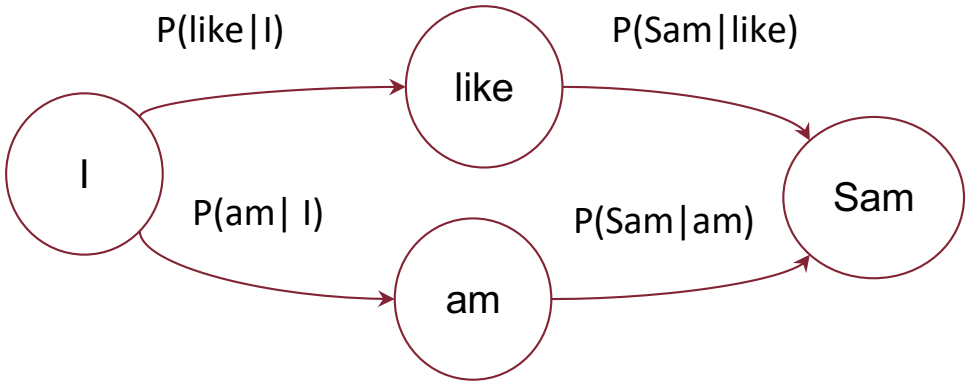
One can visually represent a Markov chain as:



one can estimate the most probable next word following the outgoing edge with the highest conditional probability

Can we use a Markov Models ?

One can visually represent a Markov chain as:



one can estimate the most probable next word following the outgoing edge with the highest conditional probability

We need something different:

1) we need a model capable of handling hidden information such as the PoS tags

likelihood prior

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max}\, P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

Can we use a Markov Models ?

We need something different:

1) we need a model capable of handling hidden information such as the PoS tags

likelihood prior

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max} P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

2) we observe words but we need hidden PoS tags

Hidden Markov Models:

The Jason Eisner task (2002)

- You keep a diary listing how many ice creams you ate every day last summer
- Using these observations you want to estimate the temperature every day (say, cold **C** or hot **H**)

Formally:

"given a sequence of observations **O**, each observation being an **integer** corresponding to the **number of ice creams** eaten on a given day, figure out the correct hidden sequence **Q** of weather states (**H** or **C**) which caused Jason to eat the ice cream"

Hidden Markov Models:

$Q = q_1 q_2 \ldots q_N$	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. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_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, $\sum_{i=1}^{n} \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions:

Markov Assumption: $P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1})$

Output Independence: $P(o_i|q_1, \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i)$

Hidden Markov Models

$Q = q_1 q_2 \ldots q_N$	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. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_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, $\sum_{i=1}^{n} \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions:

Markov Assumption: $P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1})$

Output Independence: $P(o_i|q_1, \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i)$

it looks like …

Hidden Markov Models: training a PoS tagger

$Q = q_1 q_2 \ldots q_N$	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. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_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, $\sum_{i=1}^{n} \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions:

Markov Assumption: $P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1})$

Output Independence: $P(o_i|q_1, \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i)$

likelihood prior

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max} P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

Hidden Markov Models: training a PoS tagger

$Q = q_1 q_2 \ldots q_N$	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. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_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, $\sum_{i=1}^{n} \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions:

Markov Assumption: $P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1})$

Output Independence: $P(o_i|q_1, \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i)$

emission **transition**

likelihood prior

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max} P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

Hidden Markov Models: training a PoS tagger

$Q = q_1 q_2 \ldots q_N$	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. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_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, $\sum_{i=1}^{n} \pi_i = 1$

emission likelihood **transition** prior

$$\hat{t}_1^n = \underset{t_1^n \in Tagset^n}{\arg\max} P(t_1^n \mid w_1^n) \approx \underset{t_1^n \in Tagset^n}{\arg\max} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1})$$

1) we define Q as the Tagset, O as the set of words;
2) given a training dataset (a PoS-tagged corpus):
   1: we compute the transition probability matrix as prior probabilities
   2: we compute the emission probabilities as likelihood probabilities
   3: we can compute the probabilities π counting how many times a PoS tag occurs at the beginning of a sentence of the training corpus.

$$P(t_i \mid t_{i-1}) = \frac{c(t_{i-1}, t_i)}{c(t_{i-1})}$$

$$P(w_i \mid t_i) = \frac{c(t_i, w_i)}{c(t_i)}$$

Hidden Markov Models: training a PoS tagger

$Q = q_1 q_2 \ldots q_N$	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. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_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, $\sum_{i=1}^{n} \pi_i = 1$

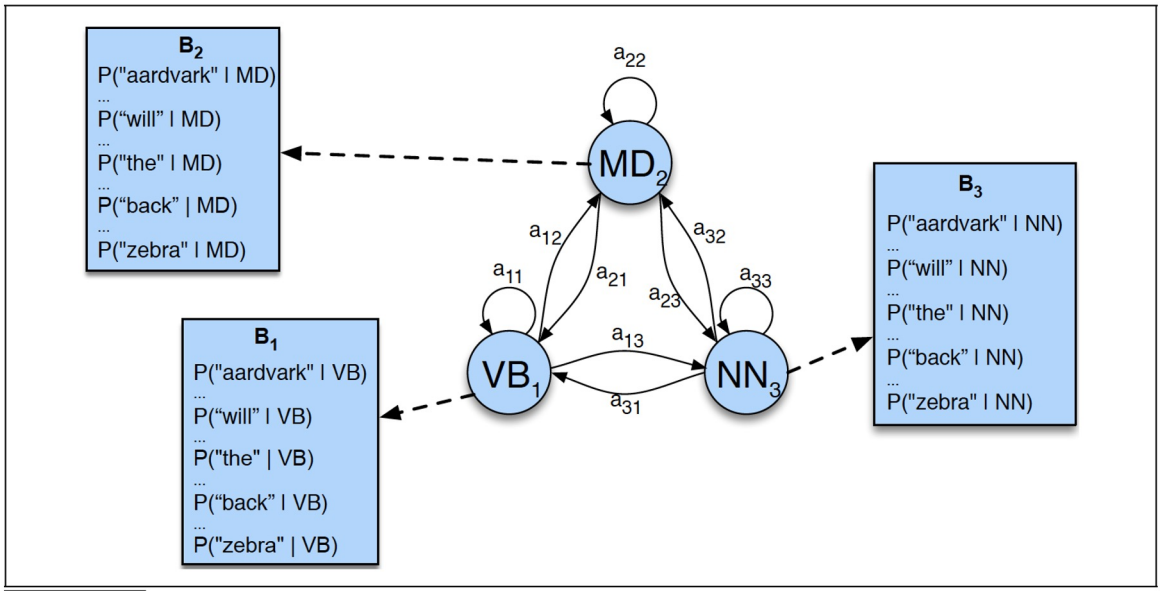


Figure 8.9 An illustration of the two parts of an HMM representation: the A transition probabilities used to compute the prior probability, and the B observation likelihoods that are associated with each state, one likelihood for each possible observation word.

Hidden Markov Models: PoS tagger - Decoding

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

In PoS tagging, since Observations are words and States are PoS tags…

Decoding (finding the most probable sequence of PoS tags) is performed with the Viterbi Algorithms

Hidden Markov Models: PoS tagger - Decoding

The 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*

Figure 8.10 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Hidden Markov Models: PoS tagger - Decoding

The Viterbi Algorithm

observations are words

states are PoS tags

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*

Figure 8.10 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Hidden Markov Models: PoS tagger - Decoding

The Viterbi Algorithm

Given an hidden markov model HHM=(A, B):

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.12 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

$Q=\{q_1,q_2, \dots , q_N\} = \{NNP,MD,VB,JJ,NN,RB,DT\}$

$a_{i,j}$ = transition probability = $P(q_j | q_i)$

$a_{0,j}= \pi_j = P(q_j | <s>)$

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.13 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

$O=\{o_1, o_2, \dots , o_T\} =\{Janet, will, back, the, bill\}$

$b_{i,j} = b_i(o_j)$ = emission probability $P(o_j | q_i)$

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Initialization:

create a path probability matrix *viterbi[N,T]*
for each state *s* **from** 1 **to** *N* **do** ; initialization step
viterbi[*s*,1] $\leftarrow \pi_s * b_s(o_1)$ **s=1**
backpointer[s,1] $\leftarrow 0$

	NNP	MD	VB	JJ	NN	RB	DT
<*s*>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT					
q_6=RB					
q_5=NN					
q_4=JJ					
q_3=VB					
q_2=MD					
q_1=NNP					
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Initialization:

create a path probability matrix *viterbi[N,T]*
for each state *s* **from** 1 **to** *N* **do** ; initialization step
viterbi[*s*,1] $\leftarrow \pi_s * b_s(o_1)$ s=1
backpointer[s,1] $\leftarrow 0$

	NNP	MD	VB	JJ	NN	RB	DT
<*s*>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT					
q_6=RB					
q_5=NN					
q_4=JJ					
q_3=VB					
q_2=MD					
q_1=NNP	**$\pi_1 * b_1(o_1)$ =.28*.000032**				
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Initialization:

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] $\leftarrow 0$

s=1

	NNP	MD	VB	JJ	NN	RB	DT
<*s*>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT					
q_6=RB					
q_5=NN					
q_4=JJ					
q_3=VB					
q_2=MD					
q_1=NNP	**$\pi_1 * b_1(o_1)$ =.000009**				
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Initialization:

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] $\leftarrow 0$

s=2

	NNP	MD	VB	JJ	NN	RB	DT
<*s*>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT					
q_6=RB					
q_5=NN					
q_4=JJ					
q_3=VB					
q_2=MD	**$\pi_2 * b_2(o_1)$**				
q_1=NNP	**$\pi_1 * b_1(o_1)$ =.000009**				
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Initialization:

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] $\leftarrow 0$

s=7

	NNP	MD	VB	JJ	NN	RB	DT
<*s*>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	$\pi_7 * b_7(o_1)$				
q_6=RB	$\pi_6 * b_6(o_1)$				
q_5=NN	$\pi_5 * b_5(o_1)$				
q_4=JJ	$\pi_4 * b_4(o_1)$				
q_3=VB	$\pi_3 * b_3(o_1)$				
q_2=MD	$\pi_2 * b_2(o_1)$				
q_1=NNP	$\pi_1 * b_1(o_1)$ =.000009				
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Recursion step:

for each time step t **from** 2 **to** T **do** ; recursion step

  for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	.0				
q_6=RB	.0				
q_5=NN	.0				
q_4=JJ	.0				
q_3=VB	.0				
q_2=MD	.0				
q_1=NNP	.000009				
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Recursion step:

for each time step t **from** 2 **to** T **do** ; recursion step

t=2
s=1

for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	.0 $*a_{7,1}*b_1(o_2)$				
q_6=RB	.0				
q_5=NN	.0				
q_4=JJ	.0				
q_3=VB	.0				
q_2=MD	.0 $*a_{2,1}*b_1(o_2)$				
q_1=NNP	.000009 $*a_{1,1}*b_1(o_2)$				
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Recursion step:

for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

t=2
s=1

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	.0	0			
q_6=RB	.0				
q_5=NN	.0				
q_4=JJ	.0				
q_3=VB	.0				
q_2=MD	.0	0			
q_1=NNP	.000009	*0.3777*0	.0		
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Recursion step:

for each time step t **from** 2 **to** T **do** ; recursion step

t=2
s=1

for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	.0				
q_6=RB	.0				
q_5=NN	.0				
q_4=JJ	.0				
q_3=VB	.0				
q_2=MD	.0				
q_1=NNP	.000009	.0			
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Recursion step:

for each time step t **from** 2 **to** T **do** ; recursion step

t=2
s=2

for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	.0				
q_6=RB	.0				
q_5=NN	.0				
q_4=JJ	.0				
q_3=VB	.0				
q_2=MD	.0	2.772e-8			
q_1=NNP	.000009	.0			
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Dec

The Viterbi Algorithm

Recursion step:

for each time step t **from** 2 **to** T **do** ; recursion step

  for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

q_7=DT	.0	.0		>0	
q_6=RB	.0	.0			
q_5=NN	.0	.0000000001			>0
q_4=JJ	.0	.0			
q_3=VB	.0	2.5e-13	>0		
q_2=MD	.0	2.772e-8			
q_1=NNP	.000009	.0			
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Decoding

The Viterbi Algorithm

termination step:

$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step

$bestpathpointer \leftarrow \text{argmax}_{s=1}^{N} viterbi[s,T]$; termination step

bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time

return *bestpath*, *bestpathprob*

we search for the max value in the last column e.g., $v_{5,5}$

q_7=DT	.0	.0		>0	
q_6=RB	.0	.0			
q_5=NN	.0	.0000000001			>0
q_4=JJ	.0	.0			
q_3=VB	.0	2.5e-13	>0		
q_2=MD	.0	2.772e-8			
q_1=NNP	.000009	.0			
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Decoding

The Viterbi Algorithm

termination step:

$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step

$bestpathpointer \leftarrow \text{argmax}_{s=1}^{N} viterbi[s,T]$; termination step

bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time

return *bestpath*, *bestpathprob*

we backtrack until we reach π

q_7=DT	.0	.0		>0	
q_6=RB	.0	.0			
q_5=NN	.0	.0000000001			>0
q_4=JJ	.0	.0			
q_3=VB	.0	2.5e-13	>0		
q_2=MD	.0	2.772e-8			
q_1=NNP	.000009	.0			
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Hidden Markov Models: PoS tagger - Decoding

The Viterbi Algorithm

q_7=DT	.0	.0		>0	
q_6=RB	.0	.0			
q_5=NN	.0	.0000000001			>0
q_4=JJ	.0	.0			
q_3=VB	.0	2.5e-13	>0		
q_2=MD	.0	2.772e-8			
q_1=NNP	.000009	.0			
π	o_1=Janet	o_2=will	o_3=back	o_4=the	o_5=bill

Janet/**NNP** will/**MD** back/**VB** the/**DT** bill/**NN**

Maximum Entropy models: intuition

MaxEnt belongs to the family of classifiers known as the exponential or log-linear classifiers

- Extract a set of features f_1, …, f_n from the input and combine them linearly
- The probability

$$P(c \mid x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i\right)$$

Feature weight

Normalizing factor

Feature

We choose the class which maximizes the class conditional probability:

$$\hat{c} = \arg\max_{c \in C} P(c \mid x)$$

Maximum Entropy models: intuition

examples of MaxEnt features

$$f_1(c,x) = \begin{cases} 1 & \text{if } word_i = \text{“race”} \ \& \ c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(c,x) = \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \ \& \ c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(c,x) = \begin{cases} 1 & \text{if } \text{suffix}(word_i) = \text{“ing”} \ \& \ c = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(c,x) = \begin{cases} 1 & \text{if } \text{is\_lower\_case}(word_i) \ \& \ c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

Harder in **HMM**s!

$$f_5(c,x) = \begin{cases} 1 & \text{if } word_i = \text{”race”} \ \& \ c = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_6(c,x) = \begin{cases} 1 & \text{if } t_{i-1} = \text{TO} \ \& \ c = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression models: intuition

Binary classification problem with the goal:
Build a "model" that can estimate **P(Y=1|X=?)**
i.e. given X, yield (or "predict") the probability that Y=1
In machine learning, tradition is to use Y for the variable being predicted and X for the features use to make the prediction.

Example:

Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

sentence and target	**X**	**Y**
They attend **Stony** Brook University.	2	1
The trail was **very** stony.	0	0
Next to the **brook** Gandalf lay thinking.	1	0
Her degree is from SUNY **Stony** Brook.	6	1
They attend **Binghamton**.	1	1

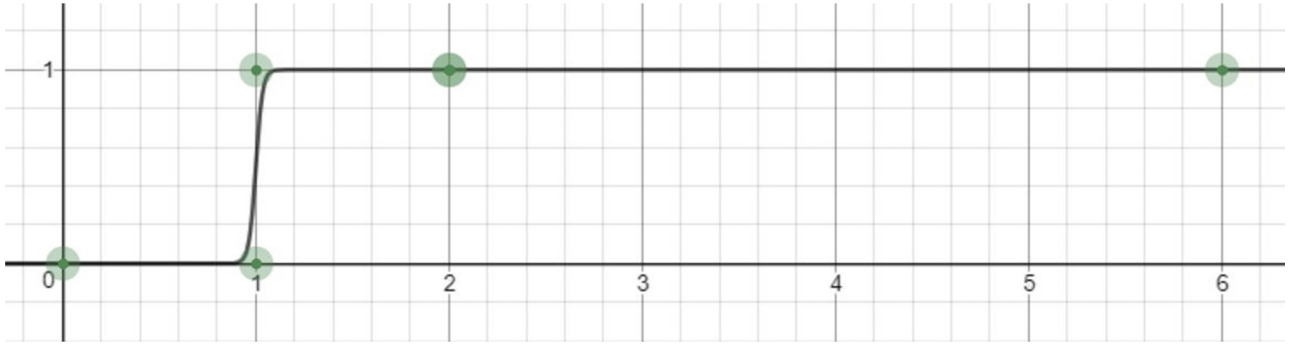
Logistic Regression models: intuition

Example:

Y: 1 if target is a part of a proper noun, 0 otherwise;

X: number of capital letters in target and surrounding words.

sentence and target	**X**	**Y**
They attend **Stony** Brook University.	2	1
The trail was **very** stony.	0	0
Next to the **brook** Gandalf lay thinking.	1	0
Her degree is from SUNY **Stony** Brook.	6	1
They attend **Binghamton**.	1	1



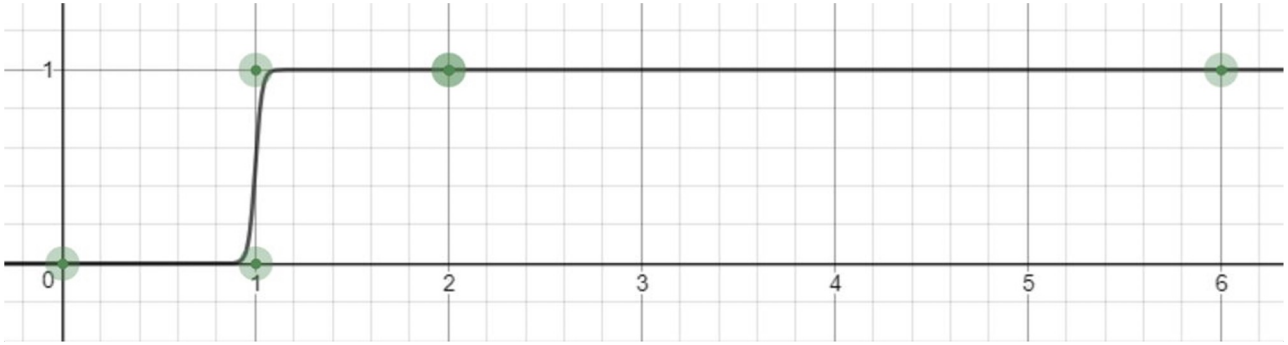
we draw the points and try to fit a logit function to better separate the points

Logistic Regression models: intuition

Example:

Y: 1 if target is a part of a proper noun, 0 otherwise;

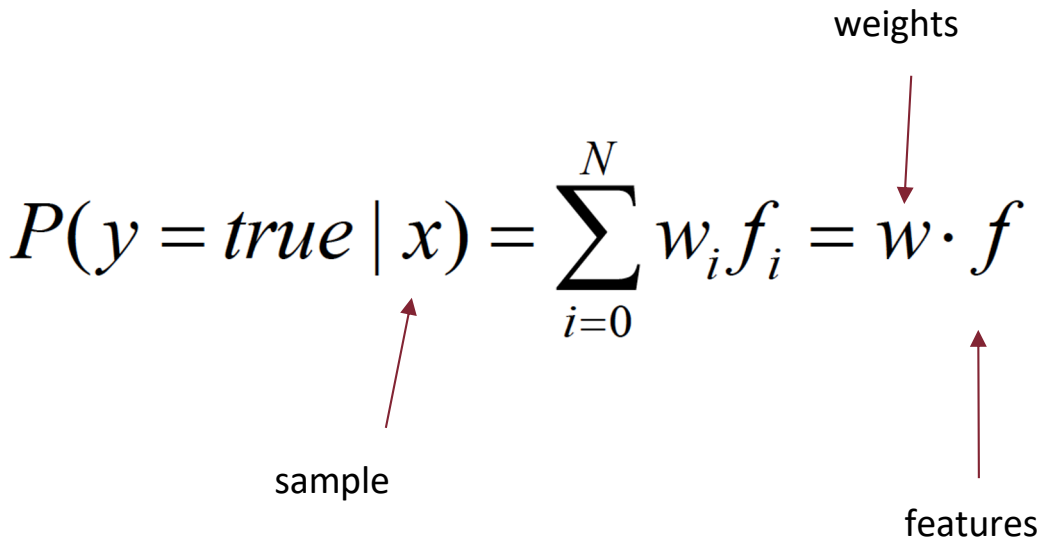
X: number of capital letters in target and surrounding words.



we draw the points and try to fit a logistic function to better separate the points, how to fit the function? is a ML problem.

From Linear to Logistic Regression models

In general



- We could **train** a model to learn those **weights.**
- **Problem**: the output of the **linear regression** is not forced to lie between 0 and 1 (it's actually between -∞ and +∞ !)
- We can calculate a ratio of two probabilities and use the linear model to predict that ratio:

$$\frac{P(y = true \mid x)}{1 - P(y = true \mid x)} = w \cdot f$$

From Linear to Logistic Regression models

•We can calculate a ratio of two probabilities and use the linear model to predict that ratio:

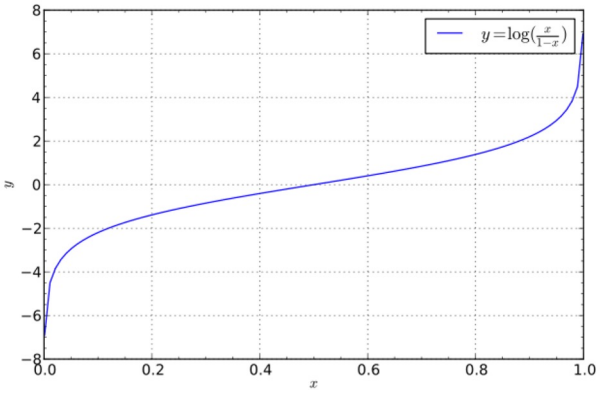
$$\frac{P(y=true\,|\,x)}{1-P(y=true\,|\,x)}=w\cdot f$$

•we take the natural log:

$$\ln\frac{P(y=true\,|\,x)}{1-P(y=true\,|\,x)}=w\cdot f$$

if we rename *P(y=true | x)=p(c),* we obtain the logit function:

$$\text{logit}(p(x))=\ln\left(\frac{p(x)}{1-p(x)}\right)$$



From Linear to Logistic Regression models

- The model which uses a linear function to estimate the logit of a probability (instead of the probability) is known as **logistic regression**:

$$\ln \frac{P(y = true \mid x)}{1 - P(y = true \mid x)} = w \cdot f$$

From Linear to Logistic Regression models

- Now, how to calculate the real probability P(y=*true*|x)?

$$\ln \frac{P(y=true \mid x)}{1-P(y=true \mid x)} = w \cdot f$$

$$\frac{P(y=true \mid x)}{1-P(y=true \mid x)} = e^{w \cdot f}$$

$$P(y=true \mid x) = (1-P(y=true \mid x))e^{w \cdot f}$$

$$P(y=true \mid x) = e^{w \cdot f} - e^{w \cdot f} P(y=true \mid x)$$

$$P(y=true \mid x) + e^{w \cdot f} P(y=true \mid x) = e^{w \cdot f}$$

$$P(y=true \mid x)(1+e^{w \cdot f}) = e^{w \cdot f}$$

$$P(y=true \mid x) = \frac{e^{w \cdot f}}{1+e^{w \cdot f}}; P(y=false \mid x) = \frac{1}{1+e^{w \cdot f}}$$

From Linear to Logistic Regression models

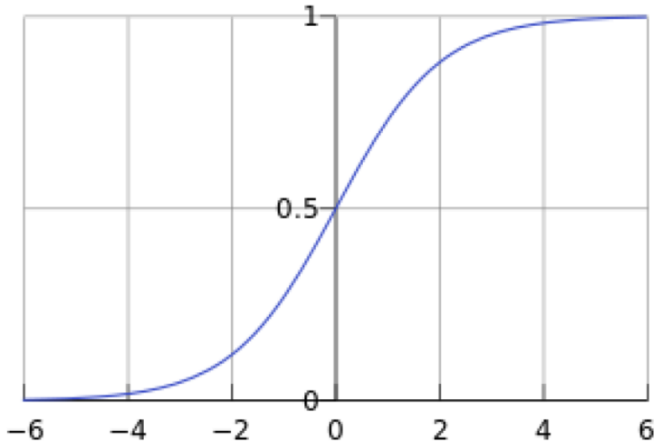
The logistic function!

$$\frac{1}{1+e^{-x}}$$

$$P(y=true \mid x) = \frac{e^{w\cdot f}}{1+e^{w\cdot f}} = \frac{e^{w\cdot f}e^{-w\cdot f}}{(1+e^{w\cdot f})e^{-w\cdot f}} = \frac{1}{1+e^{-w\cdot f}}$$

- The logistic function maps all values from -∞ and +∞ to lie between 0 and 1

$$\frac{1}{1+e^{-x}}$$



From Linear to Logistic Regression models

- The best class is the one with higher probability:

$$P(y = true \mid x) > P(y = false \mid x) = 1 - P(y = true \mid x)$$

$$\frac{P(y = true \mid x)}{1 - P(y = true \mid x)} > 1$$

$$e^{w \cdot f} > 1$$

$$w \cdot f > 0$$

Remember:

- So we just need to compute the linear combination of features and see whether the function value is positive!

$$\frac{P(y = true \mid x)}{1 - P(y = true \mid x)} = e^{w \cdot f}$$

Q&A

Resources and References

[Jurafsky&Martin, 2022] Jurafsky and Martin. Speech and Language Processing, Prentice Hall, third edition https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

[Santorini, 1990]
Santorini, B. (1990). Part-of-speech tagging guidelines for the Penn Treebank Project. https://www.nilsreiter.de/assets/2019-09-06-reflected-text-analysis/Penn-Treebank-Tagset.pdf

[Das and Petrov, 2011]
Dipanjan Das and Slav Petrov. (2011) Unsupervised Part-of-Speech Tagging with Bilingual Graph-Based Projections. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 600–609, Portland, Oregon, USA. Association for Computational Linguistics.
https://aclanthology.org/P11-1061.pdf

**Credits

The slides of this part of the course are the result of a personal reworking of the slides and of the course material from different sources:

1. The NLP course of Prof. Roberto Navigli, Sapienza University of Rome
2. The NLP course of Prof. Simone Paolo Ponzetto, University of Mannheim, Germany
3. The NLP course of Prof. Chris Biemann, University of Hamburg, Germany
4. The NLP course of Prof. Dan Jurafsky, Stanford University, USA