Concepts and Applications in NLP

Sequence Labeling for Parts of Speech and Named Entities

Marion Di Marco

November 5, 2024

Credits

Based on Chapter 17 of *Speech and Language Processing*:

"Sequence Labeling for Parts of Speech and Named Entities"

Jurafsky and Martin (2024)

Overview

- **Parts of Speech (POS)**: useful clues to sentence structure and meaning
  - syntactic structure: POS-tagging is an essential part in parsing
  - "patterns": English nouns are preceded by determiners and adjectives
- **Named Entities**: useful for tasks like question answering or information extraction.
  - *Washington*: name of a person, a place, or a university?

- **POS tagging**: assigning each word in a sequence a part of speech like noun or verb
- **Named Entity Recognition (NER)**: assigning words or phrases tags like person location or organization

Outline

Word Classes

Part-of-Speech Tagging

Named Entities and Named Entity Tagging

HMM Part-of-Speech Tagging

Conditional Random Fields (CRFs)

POS tagging in Morphologically Rich Languages

Summary

Word Classes and Parts of Speech

- **Word classes**: loosely correspond to semantic properties
  - adjectives → properties
  - nouns → people, things
  - verbs → activities
- **Parts of speech (POS)**: defined on the grammatical relationships with neighbouring words and morphological properties

- **Closed Class**: finite set of words
  - mostly function words, for example pronouns and prepositions
  - occur frequently and contribute to the structure of a sentence

- **Open Class**: infinite amount of words providing lexical content
  - nouns, verbs, adjectives, adverbs
  - new words are coined frequently (e.g. *barbiecore, greedflation*)

English Word Classes

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions and processes	*draw, provide, go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by, under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, 2026, 11:00, hundred*
	PART	Particle: a function word that must be associated with another word	*'s, not, (infinitive) to*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*whether, because*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

Figure 8.1 The 17 parts of speech in the Universal Dependencies tagset (de Marneffe et al., 2021). Features can be added to make finer-grained distinctions (with properties like number, case, definiteness, and so on).

English POS: Nouns

- Nouns: commonly used for people, places, things and other
- Common nouns
  - concrete terms: *mango, cat*
  - abstractions: *algorithm, beauty*
  - nominalizations: *(his) pacing*
- Some properties of common nouns
  - count nouns can occur in singular and plural and can be counted (*one dog, two dogs*)
  - mass nouns: something is conceptualized as a homogeneous group (*snow, *two snows*)
- Proper nouns: names of specific persons or entities
  - *Bob, IBM, Italy*

English POS: Verbs

- Verbs refer to actions and processes
- Main verbs: *eat, run, laugh*
- Auxiliary verbs: mark semantic features of a main verb such as its tense
  - *has done, was written*
- Modal verbs: mark the mood associated with the event depicted by the main verb
  - *can* → ability or possibility
  - *may* → permission or possibility
  - *must* → necessity
- Phrasal verbs: verb and a particle acting as a single unit *turn down*

English POS tags

- English-specific POS tags from the Penn Treebank

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	infinitive to	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+, %, &*	WRB	wh-adverb	*how, where*

Figure 8.2 Penn Treebank part-of-speech tags.

English POS Tagging: Example

(8.1) There/PRO/EX are/VERB/VBP 70/NUM/CD children/NOUN/NNS there/ADV/RB ./PUNC/.

(8.2) Preliminary/ADJ/JJ findings/NOUN/NNS were/AUX/VBD reported/VERB/VBN in/ADP/IN today/NOUN/NN 's/PART/POS London/PROPN/NNP Journal/PROPN/NNP of/ADP/IN Medicine/PROPN/NNP

- Tagged according to Universal Dependency (UD) and the Penn tagsets
- Penn tagset is more fine-grained
  - tense and participles on verbs,
  - number on nouns
- *London Journal of Medicine*: proper noun
  - all parts are marked as PROP/NNP

Outline

POS Tagging

- **Part of speech tagging**: assigning a POS tag to every word in a tokenized sentence

- Input: sentence $x_1, x_2, ..., x_n$ and a tagset
- Output: a corresponding sequence of tags $y_1, y_2, ..., y_n$

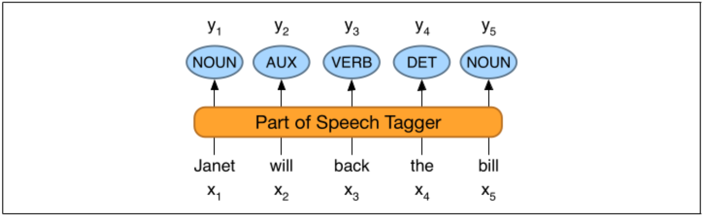


Figure 8.3 The task of part-of-speech tagging: mapping from input words $x_1, x_2, ..., x_n$ to output POS tags $y_1, y_2, ..., y_n$.

POS Tagging: Disambiguation

- Words are ambiguous and can have more than one part of speech
  - verb $\leftrightarrow$ noun:

    book that flight $\leftrightarrow$ *hand me the book*
  - determiner $\leftrightarrow$ conjunction:

    does that flight serve dinner? $\leftrightarrow$ *I thought that your flight was earlier*
- Goal of POS-tagging: resolve these ambiguities
  and find the correct tag for the context

Ambiguous Words (English)

- Overview of ambiguous words in English:

Types:		WSJ		Brown	
Unambiguous	(1 tag)	44,432	(**86%**)	45,799	(**85%**)
Ambiguous	(2+ tags)	7,025	(**14%**)	8,050	(**15%**)
Tokens:					
Unambiguous	(1 tag)	577,421	(**45%**)	384,349	(**33%**)
Ambiguous	(2+ tags)	711,780	(**55%**)	786,646	(**67%**)

Figure 8.4 Tag ambiguity in the Brown and WSJ corpora (Treebank-3 45-tag tagset).

- Most word *types* are not ambiguous:
  for example: *hesitantly* is always an adverb
- Only 14-15 % words of the vocabulary are ambiguous,
  but they are very common and account for 55-67 % word *tokens*
  for example: *that, back, down, put, set*

Ambiguous Words: Examples with *back*

- earnings growth took a **back/JJ** seat
- a small building in the **back/NN**
- a clear majority of senators **back/VBP** the bill
- Dave began to **back/VB** toward the door
- enable the country to buy **back/RP** debt
- I was twenty-one **back/RB** then

English Tagging: Some Statisctics

- Different tags are not equally likely:
  - *a* can be the letter 'a' or a determiner → determiner sense is more likely
  - *can* can be an auxiliary or a noun → more frequently used as auxiliary

- Baseline: choose the tag which is most frequent in the training corpus

> **Most Frequent Class Baseline:** Always compare a classifier against a baseline at least as good as the most frequent class baseline (assigning each token to the class it occurred in most often in the training set).

- The most-frequent-tag baseline has an accuracy of about 92 %[1]

- For comparison: accuracies on various English treebanks are 97 % (no matter the algorithm; HMMs, CRFs, BERT perform similarly)

- Human performance: about 97 % (English)

[1] In English, on the WSJ corpus, tested on sections 22-24.

Outline

Named Entities and Named Entity Tagging

- Proper nouns refer to different kinds of entities:
  - *Janet*: person
  - *Stanford University*: organization
  - *Colorado*: location

- **Named entities**: anything that can be referred to with a proper name

- **Named Entity Recognition (NER)**: find spans of text that constitute proper names and tag the type of the entity
  - PER (person)
  - LOC (location)
  - ORG (organization)
  - GPE (geo-political entity)

- NEs commonly also extend to *dates*, *times*, other kinds of *temporal expressions*, and even *numerical expressions*

Named Entities: Examples

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

Figure 8.5 A list of generic named entity types with the kinds of entities they refer to.

Citing high fuel prices, [$_{\text{ORG}}$ **United Airlines**] said [$_{\text{TIME}}$ **Friday**] it has increased fares by [$_{\text{MONEY}}$ **\$6**] per round trip on flights to some cities also served by lower-cost carriers. [$_{\text{ORG}}$ **American Airlines**], a unit of [$_{\text{ORG}}$ **AMR Corp.**], immediately matched the move, spokesman [$_{\text{PER}}$ **Tim Wagner**] said. [$_{\text{ORG}}$ **United**], a unit of [$_{\text{ORG}}$ **UAL Corp.**], said the increase took effect [$_{\text{TIME}}$ **Thursday**] and applies to most routes where it competes against discount carriers, such as [$_{\text{LOC}}$ **Chicago**] to [$_{\text{LOC}}$ **Dallas**] and [$_{\text{LOC}}$ **Denver**] to [$_{\text{LOC}}$ **San Francisco**].

Depending on the application: extend the tag set (*genes, commercial products, works of art, ...*)

Named Entities: Challenges

- Named entity tagging: useful first step in many NLP tasks
- NER has several challenges
- Segmentation
  - POS-tagging: assume tokenized text → one word gets one tag
  - NER: find and label *spans* of text → identify boundaries of NEs
- Ambiguities: NEs can belong to different categories
  JFK → a person, the airport in New York, or any number of schools, bridges, and streets in the United States.

[$_{\text{PER}}$ Washington] was born into slavery on the farm of James Burroughs.
[$_{\text{ORG}}$ Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [$_{\text{LOC}}$ Washington] for what may well be his last state visit.
In June, [$_{\text{GPE}}$ Washington] passed a primary seatbelt law.

Figure 8.6 Examples of type ambiguities in the use of the name *Washington*.

BIO Tagging

- **BIO tagging**: standard approach to sequence labeling for a span-recognition problem
- Treat NER like a word-by-word sequence labeling task
  - **B**: token that *begins* a span of interest
  - **I**: tokens that occur *inside* a span
  - **O**: tokens *outside* of any span of interest
- Distinct B and I tags for the different NE classes
- Variants:
  - **E**: tag for the *end* of a span
  - **S**: or a span consisting of a *single* word

BIO Tagging: Example

[$_{\text{PER}}$ **Jane Villanueva**] of [$_{\text{ORG}}$ **United**] , a unit of [$_{\text{ORG}}$ **United Airlines Holding**] , said the fare applies to the [$_{\text{LOC}}$ **Chicago**] route.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Figure 8.7 NER as a sequence model, showing IO, BIO, and BIOES taggings.

Outline

Sequence Labeling

- Sequence Labeling: assign a label to each token in a sentence
- **Hidden Markov Models (HMM)** are probabilistic sequence models:

  given a sequence of units, it computes a probability distribution over possible label sequences and then chooses the best one
- HMMs are based on Markov chains
- **Markov Chain**: models the probabilities of sequences of random variables, *states*, which can take values from some set (e.g. words)
- Assumption: to predict the future, only the current state matters

Markov Model

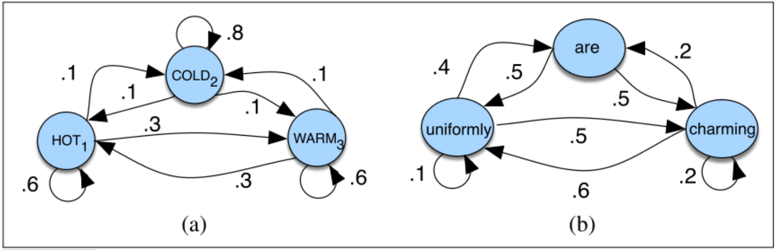


Figure 8.8 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

- **Markov Assumption:** $P(q_i = a | q_1 ... q_{i-1}) = P(q_i = a | q_{i-1})$
- Nodes: states
- Edges: transitions with probabilities
  (values of arcs leaving a state must sum to 1)

Markov Model: Definition

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, $\sum_{i=1}^{n} \pi_i = 1$

Hidden Markov Models

- Markov chains are useful to compute the probability of a sequence of *observable* events
- Some events are not observable, but *hidden*:
  for example, POS tags in a sentence

$\Rightarrow$ we see words and must infer the tags from the word sequence

- **Hidden Markov Model (HMM)**: combines *observed* events (words in text) and *hidden* events (POS tags)

Hidden Markov Model: Definition

$Q = q_1 q_2 \dots q_N$	a set of N **states**
$A = a_{11} \dots a_{ij} \dots a_{NN}$	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 $V = v_1, v_2, \dots, v_V$) being generated from a state q_i
$\pi = \pi_1, \pi_2, \dots, \pi_N$	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, $\sum_{i=1}^{n} \pi_i = 1$

- Input $O = o_1 o_2 \dots o_T$: sequence of T observations (from vocabulary V)
- Simplifying assumptions (first-order HMM):
  Markov Assumption: $P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$
  Output Independence: $P(o_i | q_1 \dots q_i \dots q_T, o_1 \dots o_i \dots o_T) = P(o_i | q_i)$

Components of a Hidden Markov Model

- HMMs use two sets of probabilities (A and B)
- Probabilities are computed by maximum likelihood estimates based on counts in a corpus
- **Tag transition probabilities** $P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$

  probability of a tag occurring given the previous tag (*A* probabilities)
- **Emission probabilities** $P(w_i|t_i) = \frac{C(t_i,w_i)}{C(t_i)}$

  probability, that a given tag is associated with a given word (*B* probabilities)

  Note: we are not asking "what is the most likely tag for word *w*?"

  Instead we ask: "if I were to generate a tag=*t*, how likely is it that the word would be *w*"?

Components of a Hidden Markov Model: Example

- Transition probabilities:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \tag{17.8}$$

In the WSJ corpus, for example, MD occurs 13124 times of which it is followed by VB 10471, for an MLE estimate of

$$P(VB|MD) = \frac{C(MD,VB)}{C(MD)} = \frac{10471}{13124} = .80 \tag{17.9}$$

- Emission probabilities:

The B emission probabilities, $P(w_i|t_i)$, represent the probability, given a tag (say MD), that it will be associated with a given word (say *will*). The MLE of the emission probability is

$$P(w_i|t_i) = \frac{C(t_i,w_i)}{C(t_i)} \tag{17.10}$$

Of the 13124 occurrences of MD in the WSJ corpus, it is associated with *will* 4046 times:

$$P(will|MD) = \frac{C(MD,will)}{C(MD)} = \frac{4046}{13124} = .31 \tag{17.11}$$

HMM: Illustration

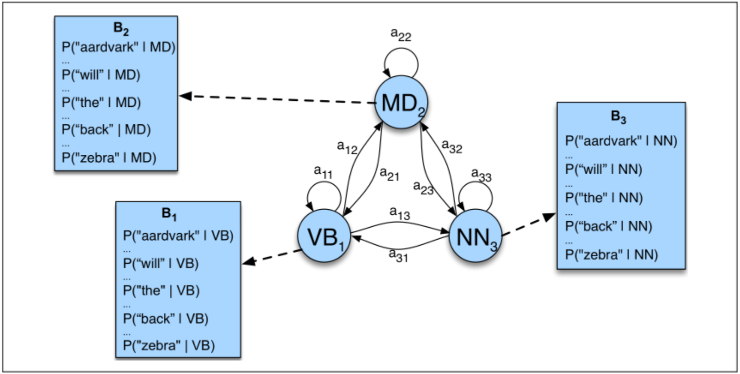


Figure 8.9 An illustration of the two parts of an HMM representation: the *A* transition probabilities used to compute the prior probability, and the *B* observation likelihoods that are associated with each state, one likelihood for each possible observation word.

- Illustration for three states of an HMM
- The full tagger has a state for each tag

HMM Tagging as Decoding

- **Decoding**: determining the sequence of hidden variables corresponding to the sequence of observations:

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

- POS-tagging: choose tag sequence $t_1 ... t_n$ that is most probable given the observation sequence o1 n words $w_1 ... w_n$

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n | w_1 \ldots w_n) \tag{17.12}$$

- Bayes's rule:

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} \frac{P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)}{P(w_1 \ldots w_n)} \tag{17.13}$$

- Simplify by dropping the denominator:

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n) \tag{17.14}$$

HMM Tagging as Decoding

- HMM simplifying assumption 1: output independence
  probability of a word appearing depends only on its own tag, independent of neighboring words and tags

$$P(w_1 \dots w_n | t_1 \dots t_n) \approx \prod_{i=1}^{n} P(w_i | t_i) \quad (17.15)$$

- HMM simplifying assumption 2: Markov assumption
  probability of a tag is dependent only on the previous tag

$$P(t_1 \dots t_n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1}) \quad (17.16)$$

- Apply the simplifying assumptions:

$$\hat{t}_{1:n} = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(t_1 \dots t_n | w_1 \dots w_n) \approx \underset{t_1 \dots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i | t_i)}^{\text{emission}} \overbrace{P(t_i | t_{i-1})}^{\text{transition}} \quad (17.17)$$

- The two parts correspond neatly to the B **emission probability** and A **transition probability**

The Viterbi Algorithm

- Decoding algorithm for HMMs
- Idea: recursively compute an optimal sequence from optimal solutions for sub-problems (dynamic programming)
- Probability matrix or lattice
  - one column for each observation o_t
  - one row for each state q_i

  → each column has a cell for each state q_i
- Cells $v_t(j)$ represent the probability that the HMM is in state j
  - after seeing the first t observations
  - passing through the most probable state sequence $q_1 ... q_{t-1}$
  - given the HMM λ .
- Cells $v_t(j)$: recursively taking the most probable path to this cell

$$v_t(j) = \max_{q_1,\ldots,q_{t-1}} P(q_1 ... q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda) \qquad (17.18)$$

The Viterbi Algorithm

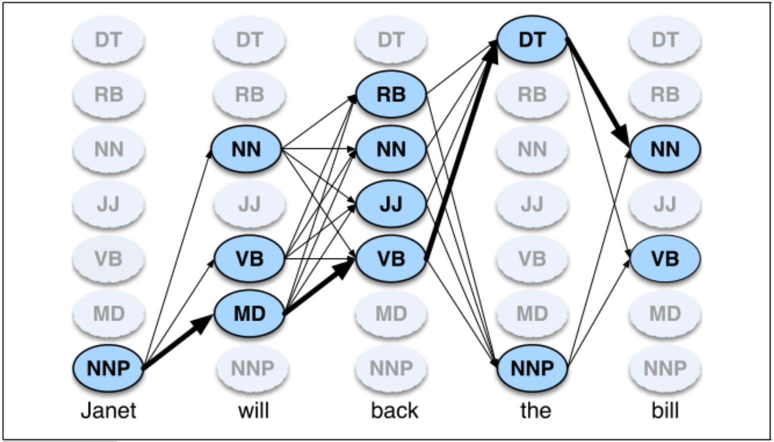


Figure 17.11 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

The Viterbi Algorithm

- Cells are filled recursively
- Probability of being in every state at time $t-1$ already computed: take most probable extension of the paths that lead to current cell
- For given state $q-j$ at time t: $v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$

$v_{t-1}(i)$	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

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

Figure 17.10 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.

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**
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
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*

Initialization

Figure 17.10 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.

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

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

Figure 17.10 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.

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

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

Figure 17.10 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.

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
    $viterbi[s,1] \leftarrow \pi_s \ast b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} \; viterbi[s',t-1] \ast a_{s',s} \ast b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} \; viterbi[s',t-1] \ast a_{s',s} \ast b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} \; viterbi[s,T]$

Termination

$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} \; viterbi[s,T]$; termination step
bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure 17.10 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.

The Viterbi Algorithm: Example

- Input sentence: *Janet will back the bill*

DT					
RB					
NN					
JJ					
VB					
MD					
NNP					
	Janet	will	back	the	bill

The Viterbi Algorithm: Example

	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 17.12 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 0.796 <s> is the start token.

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

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

The Viterbi Algorithm: Example

- Input sentence: *Janet will back the bill*

DT	...				
RB	...				
NN	...				
JJ	...				
VB	...				
MD	$P(MD \mid <s>) * P(Janet \mid MD)$				
NNP	$P(NNP \mid <s>) * P(Janet \mid NNP)$				
	Janet	**will**	**back**	**the**	**bill**

Column 1 (*Janet*): product of the π transition probability (start probability from $<s>$) and the observation likelihood of *Janet*

The Viterbi Algorithm: Example

- Input sentence: *Janet will back the bill*

DT	...				
RB	...				
NN	...				
JJ	...				
VB	...				
MD	$0.0006 * 0$				
NNP	$0.02767 * 0.000032$				
	Janet	**will**	**back**	**the**	**bill**

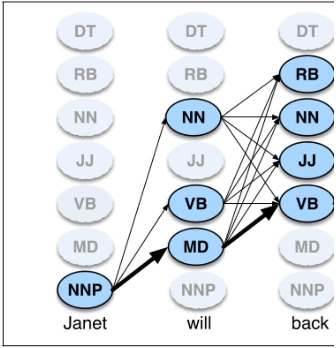
The Viterbi Algorithm: Example

- Input sentence: *Janet will back the bill*

DT	0	
RB	0	
NN	0	$v(NNP, Janet) * P(NN\|NNP) * P(will\|NN)$
JJ	0	
VB	0	$v(NNP, Janet) * P(VB\|NNP) * P(will\|VB)$
MD	0	$v(NNP, Janet) * P(MD\|NNP) * P(will\|MD)$
NNP	$8.85x10-6$	
	Janet	**will**

...

The Viterbi Algorithm: Example



v[RB,back] = max{v[NN,will]*P(RB|NN)*P(back|RB),
v[VB,will]*P(RB|VB)*P(back|RB),
v[MD,will]*P(RB|MD)*P(back|RB)}

v[NN,back] = max{v[NN,will]*P(NN|NN)*P(back|NN),
v[VB,will]*P(NN|VB)*P(back|NN),
v[MD,will]*P(NN|MD)*P(back|NN)}

v[JJ,back] = max{v[NN,will]*P(JJ|NN)*P(back|JJ),
v[VB,will]*P(JJ|VB)*P(back|JJ),
v[MD,will]*P(JJ|MD)*P(back|JJ)}

v[VB,back] = max{v[NN,will]*P(VB|NN)*P(back|VB),
v[VB,will]*P(VB|VB)*P(back|VB),
v[MD,will]*P(VB|MD)*P(back|VB)}

The Viterbi Algorithm: Example

DT	0	0	0
RB	0	0	***
NN	0	***	***
JJ	0	0	***
VB	0	***	***
MD	0	***	0
NNP	$8.85x10-6$	0	0
	Janet	**will**	**back**

- Assemble the best tag sequence?
  - use backpointers
  - trace backwards from the max score at the last time step

Outline

Word Classes

Part-of-Speech Tagging

Named Entities and Named Entity Tagging

HMM Part-of-Speech Tagging

Conditional Random Fields (CRFs)

POS tagging in Morphologically Rich Languages

Summary

Challenges for HMMs

- Unknown words: proper names, acronyms, novel nouns and verbs
- Add features to hep handle unknown words
  - capitalization → likely proper nouns
  - morphology → suffix to indicate word class
  - info on previous or following word → *the* is unlikely to precede a verb
- Difficult to include features into HMMs
  all computation is based on the two probabilities P(tag|tag) and P(word|tag).
  How to encode extra knowledge into these probabilities?
  Complicated conditioning → more and more difficult
- Linear chain CRFs

CRF: Introduction and Definition

- Given a sequence of input words $X = x_1 \ldots x_n$
  compute a sequence of output tags $Y = y_1 \ldots y_n$

- HMM: compute the best tag sequence that maximizes $P(Y|X)$ relying on Bayes' rule and the likelihood $P(X|Y)$

$$\begin{aligned}
\hat{Y} &= \underset{Y}{\operatorname{argmax}}\, p(Y|X) \\
&= \underset{Y}{\operatorname{argmax}}\, p(X|Y)p(Y) \\
&= \underset{Y}{\operatorname{argmax}} \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1})
\end{aligned}$$

- CRF: compute the posterior $p(Y|X)$ directly, training the CRF to discriminate among the possible tag sequences

$$\hat{Y} = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}}\, P(Y|X)$$

CRFs: Introduction and Definition

- CRF: assigns a probability to an entire output (tag) sequence Y, out of all possible sequences $\mathcal{Y}$, given the entire input (word) sequence X.
- CRFs do not compute a probability for each tag at each time step. Instead: log-linear functions over a set of relevant features. Local features are aggregated and normalized to produce a global probability for the whole sequence
- Feature function F: maps input sequence X and output sequence Y to a feature vector

CRFs: Definition

- K features, with a weight w_k for each feature F_k:

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)}$$

- Pull out denominator into a function $Z(X)$

$$p(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)$$

- K functions $F_k(X,Y)$ are called global features:
  each one is a property of the input sequence X and the output sequence Y
- Compute by decomposing into a sum of local features for each position i in Y

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

CRFs: Introduction and Definition

- Each local features f_k in a linear-chain CRF can make use of
  - the current output token y_i
  - the previous output token y_{i-1}
  - the entire input string X (or any subpart of it)
  - the current position i
- Constraint to current and previous output tokens y_i and y_{i-1}: characterizes a linear chain CRF
  ($\rightarrow$ this limitation allows to apply a version of Viterbi algorithm)
- General CRF: make use of any output token
  ($\rightarrow$ more complex inference)

Features in a CRF POS tagger

- Each local feature f_k at position i can depend on any information from (y_{i-1}, y_i, X, i)

- Assume that all features take on the value 1 or 0

$$\mathbb{1}\{x_i = \textit{the},\ y_i = \text{DET}\}$$
$$\mathbb{1}\{y_i = \text{PROPN},\ x_{i+1} = \textit{Street},\ y_{i-1} = \text{NUM}\}$$
$$\mathbb{1}\{y_i = \text{VERB},\ y_{i-1} = \text{AUX}\}$$

- Feature templates populate the set of features from every instance in the data set

$$\langle y_i, x_i \rangle, \langle y_i, y_{i-1} \rangle, \langle y_i, x_{i-1}, x_{i+2} \rangle$$

f_{3743}: y_i = VB and x_i = back
f_{156}: y_i = VB and y_{i-1} = MD
f_{99732}: y_i = VB and x_{i-1} = will and x_{i+2} = bill

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

Features in a CRF POS tagger

- Word shape features to handle unknown words
- Represent abstract letter pattern of the word
  - lower-case letters → x
  - upper-case letters → X
  - numbers → d
  - retain punctuation
- Prefix and suffix features
- Example: *well-dressed*

$$\begin{aligned}
&\text{prefix}(x_i) = \texttt{w}\\
&\text{prefix}(x_i) = \texttt{we}\\
&\text{suffix}(x_i) = \texttt{ed}\\
&\text{suffix}(x_i) = \texttt{d}\\
&\text{word-shape}(x_i) = \texttt{xxxx-xxxxxxx}
\end{aligned}$$

Outline

Word Classes

Part-of-Speech Tagging

Named Entities and Named Entity Tagging

HMM Part-of-Speech Tagging

Conditional Random Fields (CRFs)

POS tagging in Morphologically Rich Languages

Summary

POS tagging in Morphologically Rich Languages

- Morphologically rich languages
  - large vocabulary → data sparsity
  - more information contained
- POS tagger for morphologically rich languages need to label more information (e.g. case or gender)
- Tagsets for morphologically rich languages: sequences of morphological tags

1.	Yerdeki **izin** temizlenmesi gerek. **The trace** on the floor should be cleaned.	iz + Noun+A3sg+Pnon+Gen
2.	Üzerinde parmak **izin** kalmiş. **Your** finger **print** is left on (it).	iz + Noun+A3sg+P2sg+Nom
3.	Içeri girmek için **izin** alman gerekiyor. You need **permission** to enter.	izin + Noun+A3sg+Pnon+Nom

- Results in large tag set → combine with morphological analysis

Outline

Summary

- Languages generally have a small set of **closed class words** that are highly frequent, ambiguous, and act as function words, and **open-class words** like nouns, verbs and adjectives

- Part-of-speech tagging: the process of assigning a part-of-speech label to each of a sequence of words

- The probabilities in HMM taggers: estimated by maximum likelihood estimation on tag-labeled training corpora.

- The Viterbi algorithm is used for decoding, finding the most likely tag sequence

- CRF taggers: a log-linear model that can choose the best tag sequence given an observation sequence, based on a set of features