Natural Language Processing

Lecture 7 : Part-of-Speech Tagging

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture partially based on material originally developed by :
Marco Kuhlman, Linköping University
Mark-Jan Nederhof, University of St. Andrews

Part-of-speech tagging

©The New York Times

Part-of-speech

©seo-herd

Part-of-speech

Part-of-speech (POS) tags are lexical categories such as noun, verb, adjective, adverb, pronoun, preposition, article, etc.

Also known as word classes or morphological classes. Recall these classes are defined either distributionally or else functionally (see essentials of linguistics lecture).

We call **tagset** the set of all POS tags used by some model.

Different languages, different grammatical theories, and different applications may require different tagsets.

Part-of-speech

POS tags fall into two broad categories: closed class and open class.

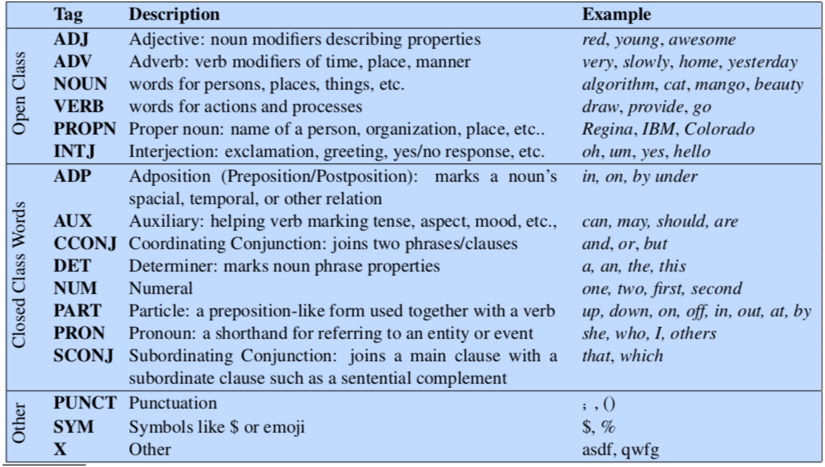
Closed class includes prepositions, pronouns, articles, etc. New words in this class are rarely coined.

Open class consists of four major groups: nouns (including proper nouns), verbs, adjectives, and adverbs. New words appear almost always in this class.

Interjections are also a (minor) group in open class.

Part-of-speech

The Universal Dependencies (UD) tagset contains 17 tags:

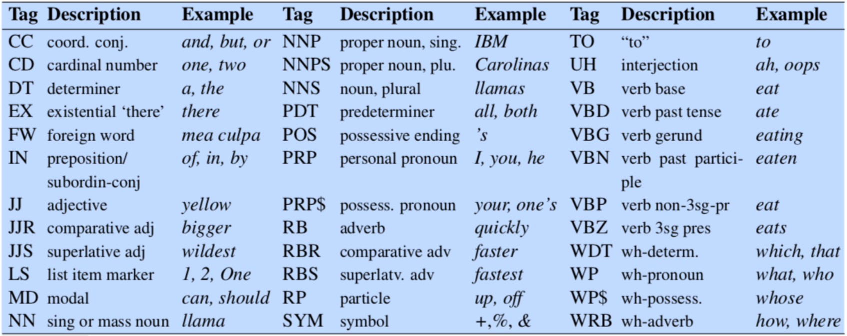


	Tag	**Description**	**Example**
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions 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, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	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	*that, which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

UD dataset has POS tagged corpora for 100+ languages, at time of writing.

Part-of-speech

The English-specific Penn Treebank (PTB) tagset is also very popular; it contains 45 tags.



Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"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, one's*	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*

Part-of-speech tagging

Part-of-speech tagging

Words are **ambiguous**: depending on the context in which they appear, they may have different tags.

Example : Word ‘book’ can be tagged as VERB or as NOUN

- **book/VERB** that flight
- hand me that **book/NOUN**

Only about 15% of the word types in the Brown corpus are ambiguous. But 67% of the word tokens are ambiguous.

Part-of-speech tagging

The task of **part-of-speech tagging** involves the assignment of the **proper** (unique) POS tag to each word in an input sentence.

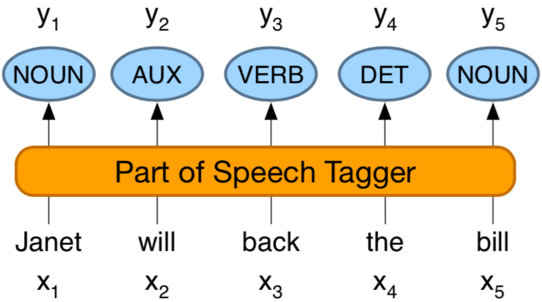
POS tagging must be done in the context of an entire sentence, on the basis of the **grammatical relationship** of a word with its neighboring words.

This is an instance of a more general task called sequence labelling, which we will discuss later.

Part-of-speech tagging

The input is a sequence $x_1, x_2, \ldots, x_n$ of (tokenized) words, and the output is a sequence $y_1, y_2, \ldots, y_n$ of tags, with y_i the tag assigned to x_i.

We assume the tagset is fixed, not part of the input.



Part-of-speech tagging

In POS tagging we need to output a whole sequence of tags $y_1, y_2, \ldots, y_n$ for the input string, not just a category.

POS tagging is therefore a **structured prediction** task, not a classification task.

Structured prediction already mentioned in the introduction lecture.

The number of output structures can be **exponentially** large in the length of the input, which makes structured prediction more challenging than classification.

Evaluation

Evaluation

The **accuracy** of a part-of-speech tagger is the percentage of test set tags that match human gold labels.

Human ceiling: how often do human annotators agree on the same tag? For PTB this is around 97%.

Accuracies over 97% have been reported across several languages, using the UD tagset.

This also holds for the algorithms we will present in this lecture.

Evaluation

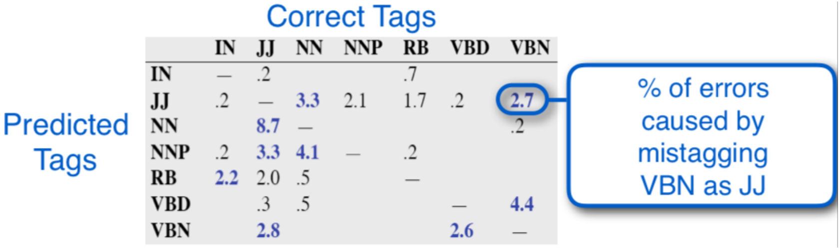
Most Frequent Class baseline: assigning each token to the class it occurred in most often in the training set. This baseline has an accuracy of about 92%.

The most frequent tag NOUN is assigned to unknown words.

Always compare your classifier against a baseline at least as good as the most frequent class baseline.

Practical issues

Qualitative evaluation: generate a **confusion matrix** for development set. This is a record of how often a word with gold tag y_i was mistagged as y_j, $j \neq i$.



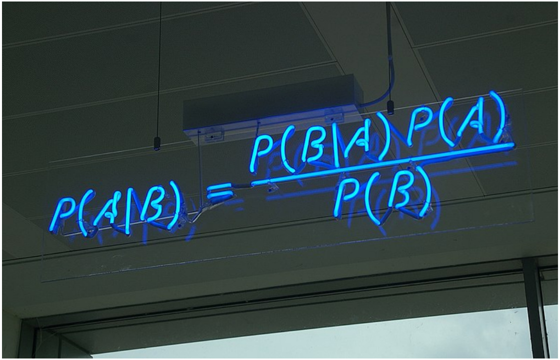
Correct Tags

Predicted Tags

	IN	JJ	NN	NNP	RB	VBD	VBN
IN	—	.2			.7		
JJ	.2	—	3.3	2.1	1.7	.2	2.7
NN		8.7	—				.2
NNP	.2	3.3	4.1	—	.2		
RB	2.2	2.0	.5		—		
VBD		.3	.5			—	4.4
VBN		2.8				2.6	—

% of errors caused by mistagging VBN as JJ

Hidden Markov model



https://en.wikipedia.org/wiki/Bayes%27_theorem

Hidden Markov model

Hidden Markov models (HMMs) first applied in speech recognition, starting in the mid-1970s.

Later applied in several areas, including NLP and analysis of biological sequences.

HMM is a **generative model**: it models how a class could generate some input data. You might use the model to generate examples.

Contrast with discriminative models, discussed later, which only learn to distinguish classes, without learning much about them.

Hidden Markov model

Let $w_{1:n} = w_1, w_2, \ldots, w_n$ be an input sequence of words, and let $\mathcal{Y}(w_{1:n})$ be the set of all possible tag sequences $t_{1:n} = t_1, t_2, \ldots, t_n$ for $w_{1:n}$.

The goal of POS tagging is to choose the **most probable** tag sequence $\widehat{t}_{1:n} \in \mathcal{Y}(w_{1:n})$

$$\widehat{t}_{1:n} \quad = \quad \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} \; P(t_{1:n} \mid w_{1:n})$$

This is the typical formulation for a structured prediction problem. Note that $\mathcal{Y}(w_{1:n})$ is not the set of all strings of length n over the tagset.

Term $P(t_{1:n} \mid w_{1:n})$ is referred to as the **posterior** probability and can be **difficult** to model/compute.

Hidden Markov model

We break down the posterior probability using **Bayes rule**:

$$\begin{aligned}\hat{t}_{1:n} &= \underset{t_{1:n} \in \mathcal{Y}(w_{1:n})}{\operatorname{argmax}} \; P(t_{1:n} \mid w_{1:n}) \\ &= \underset{t_{1:n} \in \mathcal{Y}(w_{1:n})}{\operatorname{argmax}} \; \frac{P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n})}{P(w_{1:n})} \\ &= \underset{t_{1:n} \in \mathcal{Y}(w_{1:n})}{\operatorname{argmax}} \; P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n})\end{aligned}$$

where we have used the fact that $w_{1:n}$ is given, so $P(w_{1:n})$ is a constant.

The term $P(t_{1:n})$ is referred to as the **prior** or **marginal** probability. The term $P(w_{1:n} \mid t_{1:n})$ is the **likelihood** of the words given the tags.

HMM models $P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n})$, which equals the **joint probability** $P(t_{1:n}, w_{1:n})$.

Hidden Markov model

HMM POS taggers make two simplifying assumptions.

The **first** is that the probability of a word depends only on its own POS tag and is independent of neighboring words and tags:

$$P(w_{1:n} \mid t_{1:n}) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

The factor $P(w_i \mid t_i)$ is referred to as the **emission** probability.

The emission probability answers the following questions: If we were going to generate t_i, how likely is it that the associated word would be w_i?

Hidden Markov model

The **second** assumption is the Markov assumption that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence

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

This is connected with 2-gram models. The right-hand side requires start- and end-markers which are ignored here for simplicity and will be discussed later.

The factor $P(t_i \mid t_{i-1})$ is referred to as the **transition** probability.

Hidden Markov model

Putting everything together we have:

$$\begin{aligned}\widehat{t}_{1:n} &= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(w_{1:n} \mid t_{1:n}) \cdot P(t_{1:n}) \\ &\approx \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} \left(\prod_{i=1}^{n} P(w_i \mid t_i) \cdot P(t_i \mid t_{i-1}) \right)\end{aligned}$$

Probability estimation

Probability estimation

Assume a **tagged corpus**, where each word has been tagged with its gold label. We implement **supervised** learning using the relative frequency estimator.

See lecture on language model for definition of count $C(\cdot)$.

For the transition probability we obtain:

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

Similarly, for the emission probability we obtain:

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

Probability estimation

Example :

$$P(\mathrm{NN} \mid \mathrm{DT}) = \frac{C(\mathrm{DT\ NN})}{C(\mathrm{DT})} = \frac{56509}{116454} \approx 0.49$$

$$P(\mathrm{is} \mid \mathrm{VBZ}) = \frac{C(\mathrm{VBZ, is})}{C(\mathrm{VBZ})} = \frac{10073}{21627} \approx 0.47$$

Using the WSJ tagset: VBZ = verb 3rd singular, NN = singular noun, DT = determiner.

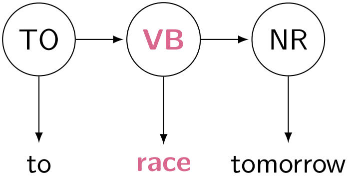
Probability estimation

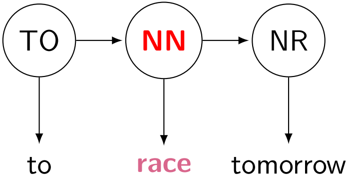
Example :

He/PPS is/VBZ expected/VBN to/TO **race/VB** tomorrow/NR

Why is VB more likely than NN for token **race** in sentence above?

There are at least two sequences of tags which we could consider:





Probability estimation

The two emission probabilities turn out not to differ too much:

$$\begin{aligned} P(\textbf{race} \mid \textbf{VB}) &= 0.00012 \\ P(\textbf{race} \mid \textbf{NN}) &= 0.00057 \end{aligned}$$

Nor is there much difference on whether NR follows NN or VB:

$$\begin{aligned} P(\text{NR} \mid \textbf{VB}) &= 0.0027 \\ P(\text{NR} \mid \textbf{NN}) &= 0.0012 \end{aligned}$$

However, the big difference is in whether NN or VB follows TO:

$$\begin{aligned} P(\textbf{VB} \mid \text{TO}) &= 0.83 \\ P(\textbf{NN} \mid \text{TO}) &= 0.00047 \end{aligned}$$

Probability estimation

Altogether we get:

$$\begin{aligned} P(\mathbf{VB} \mid \mathrm{TO})\ P(\mathbf{race} \mid \mathbf{VB})\ P(\mathrm{NR} \mid \mathbf{VB}) &= 0.00000027 \\ P(\mathbf{NN} \mid \mathrm{TO})\ P(\mathbf{race} \mid \mathbf{NN})\ P(\mathrm{NR} \mid \mathbf{NN}) &= 0.00000000032 \end{aligned}$$

therefore VB is the more likely tag for **race** in this sentence, assuming the preceding one is TO and the next one is NR.

HMMs as automata

©Pexels

HMMs as automata

We can formally define an HMM as a special type of **probabilistic finite state automaton** which generates sentences (instead of accepting sentences).

The states represent ‘hidden’ information, that is, the POS tags which are not observed.

Special start and final states are also used which are not POS tags.

The transition function is defined according to the transition probabilities.

Each state **generates** a word according to the emission probabilities. The generated output is an observable word sequence.

HMMs as automata

HMM definition:

- finite set of **output symbols** V
- finite set of **states** Q, with initial state q_0 and final state q_f
- **transition probabilities** $a_{q,q'}$ for each pair q, q',
  $q \in Q \setminus \{q_f\}$, $q' \in Q \setminus \{q_0\}$
- **emission probabilities** $b_q(u)$ for each pair q, u,
  $q \in Q \setminus \{q_0, q_f\}$, $u \in V$

Textbook uses distribution π_q as initial state, that is, $a_{q_0,q} = \pi_q$.

HMMs as automata

Transition and emission probabilities are **subject to**:

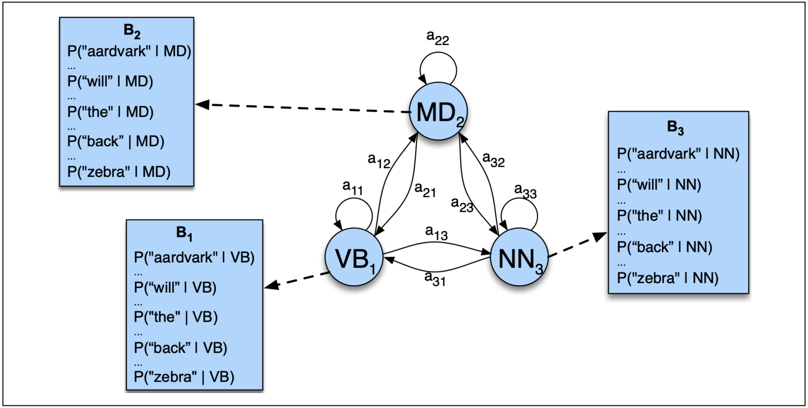
- $\sum_{q'} a_{q,q'} = 1$ for all $q \in Q \setminus \{q_f\}$
- $\sum_{u} b_q(u) = 1$ for all $q \in Q \setminus \{q_0, q_f\}$

Probabilities $a_{q_0,q}$ define the so-called **initial** probability distribution, sometimes also denoted as $\pi(q)$.

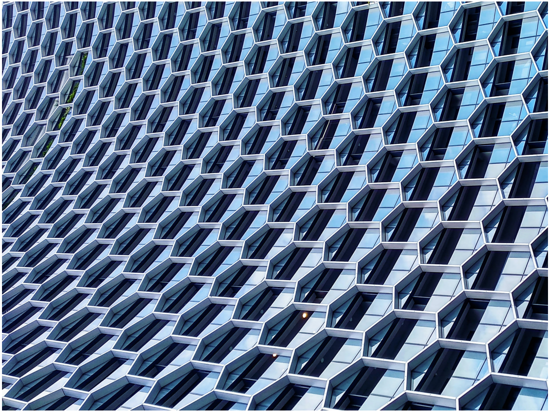
Probabilities a_{q,q_f} are the **stop** probabilities, not used in the textbook.

HMMs as automata

Example : Small excerpt, q_0 ad q_f not shown:



Viterbi algorithm



Ankit Dembla from Unsplash

Decoding

Decoding problem for HMMs: Given a sequence of observations $w_{1:n}$, find the most probable sequence of states/tags $\widehat{t}_{1:n}$

$$\begin{aligned}\widehat{t}_{1:n} &= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(t_{1:n} \mid w_{1:n}) \\ &= \operatorname*{argmax}_{t_{1:n} \in \mathcal{Y}(w_{1:n})} \left(\prod_{i=1}^{n} P(w_i \mid t_i) \cdot P(t_i \mid t_{i-1}) \right)\end{aligned}$$

Also called **inference** problem.

In the context of systems with hidden variables, the decoding problem asks to retrieve some hidden structure underlying the observed (input) structure.

Decoding

In order to compute $\widehat{t}_{1:n}$ we can use the following **naive algorithm**

- enumerate all sequences (paths) of POS tags $t_{1:n}$ consistent with observed sentence $w_{1:n}$
- perform a max search over all the joint probabilities $P(t_{1:n}, w_{1:n})$

This algorithm requires **exponential time**, since there can be exponentially many $t_{1:n}$ sequences in $\mathcal{Y}(w_{1:n})$!

This is a very common scenario for structured prediction problems.

Viterbi algorithm

The classical decoding algorithm for HMMs is the **Viterbi algorithm**, an instance of dynamic programming.

Related to algorithms for computing minimum edit distance.

The Viterbi algorithm computes the optimal sequence $\widehat{t}_{1:n}$ and the associated joint probability $P(\widehat{t}_{1:n}, w_{1:n})$ in **polynomial time**, exploiting dynamic programming.

Viterbi algorithm

Let $w_{1:n} = w_1, w_2, \ldots, w_n$ be the input sequence.

In what follows

- q denotes a state/tag of the HMM
- i denotes an input position, $0 \leqslant i \leqslant n+1$

Input positions 0 and $n+1$ represent start and end markers, respectively.

We use a two-dimensional table $vt[q, i]$ denoting the probability of the **best path** to get to state q after scanning $w_{1:i}$.

This table is also called the Viterbi lattice.

We use a two-dimensional table $bkpt[q, i]$ for retrieving the best path.

Viterbi algorithm

Initialisation step: for all q

- $vt[q,1] = a_{q_0,q} \cdot b_q(w_1)$
- $bkpt[q,1] = q_0$

Recursive step: for all $i = 2, \ldots, n$ and for all q

- $vt[q,i] = \max_{q'}\; vt[q', i-1] \cdot a_{q',q} \cdot b_q(w_i)$
- $bkpt[q,i] = \operatorname{argmax}_{q'}\; vt[q', i-1] \cdot a_{q',q} \cdot b_q(w_i)$

Termination step:

- $vt[q_f, n+1] = \max_{q'}\; vt[q', n] \cdot a_{q',q_f}$
- $bkpt[q_f, n+1] = \operatorname{argmax}_{q'}\; vt[q', n] \cdot a_{q',q_f}$

Viterbi algorithm

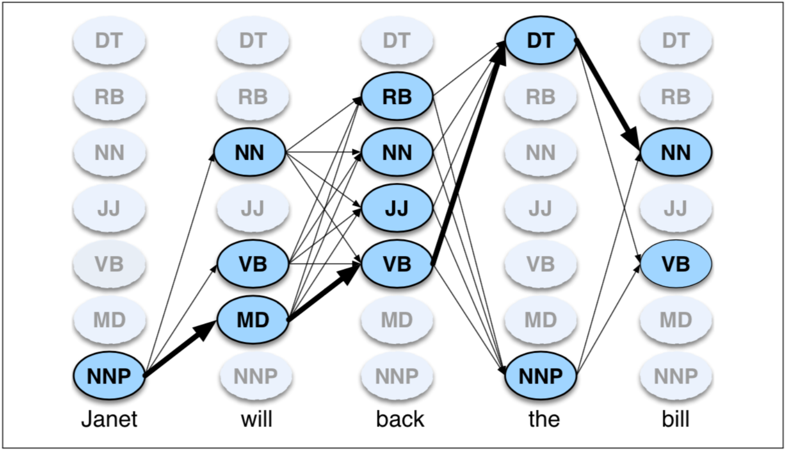
After execution of the algorithm we have

$$vt[q_f, n+1] \quad = \quad P(\widehat{t}_{1:n}, w_{1:n})$$

where $\widehat{t}_{1:n} = q_1, \ldots, q_n$ is the **most likely sequence** of tags for $w_{1:n}$.

The sequence of tags $\widehat{t}_{1:n}$ can be **reconstructed** starting with $bkpt[q_f, n+1]$ and following the backpointers.

Example



The above graph is called trellis, we will come back to this structure later.

Forward algorithm

Mitchell Luo from Unsplash

Forward algorithm

With the goal of developing an **unsupervised** algorithm for the estimation of HMM, we now develop some auxiliary algorithms.

Consider the probability of the sequence $w_{1:n}$, defined as

$$\begin{aligned} P(w_{1:n}) &= \sum_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(t_{1:n}, w_{1:n}) \\ &= \sum_{t_{1:n} \in \mathcal{Y}(w_{1:n})} \left(\prod_{i=1}^{n} P(w_i \mid t_i) \cdot P(t_i \mid t_{i-1}) \right) \end{aligned}$$

This is also called the likelihood of the sequence $w_{1:n}$.

Forward algorithm

The **forward algorithm** computes $P(w_{1:n})$ in **polynomial time**, exploiting dynamic programming.

The forward algorithm is very similar to Viterbi's algorithm, but using summation instead of maximisation.

The two algorithms use two different **semirings**.

No use of backpointers, since we do not need to retrieve an optimal sequence.

Forward algorithm

Let $w_{1:n} = w_1, w_2, \ldots, w_n$ be the input sequence.

We use a two-dimensional table $\alpha[q, i]$ denoting the sum of probabilities of **all paths** that reach state q after scanning $w_{1:i}$.

Formally, this is the **joint probability** of $w_{1:i}$ and state q at i, and can be written as

$$\alpha[q, i] \quad = \quad \sum_{\substack{t_{1:i} \in \mathcal{Y}(w_{1:i}) \\ \text{s.t. } t_i = q}} P(t_{1:i}, w_{1:i})$$

Each of the above quantities is called **forward probability**.

Forward algorithm

Initialisation step: for all q

- $\alpha[q, 1] = a_{q_0,q} \cdot b_q(w_1)$

Recursive step: for all $i = 2, \ldots, n$ and for all q

- $\alpha[q, i] = \sum_{q'} \alpha[q', i-1] \cdot a_{q',q} \cdot b_q(w_i)$

Termination step:

- $\alpha[q_f, n+1] = \sum_{q'} \alpha[q', n] \cdot a_{q',q_f}$

After execution of the algorithm we have

$$\alpha[q_f, n+1] = P(w_{1:n})$$

Trellis

An intuitive interpretation of the forward algorithm is in terms of expansion of the HMM into a **trellis**, defined as follows.

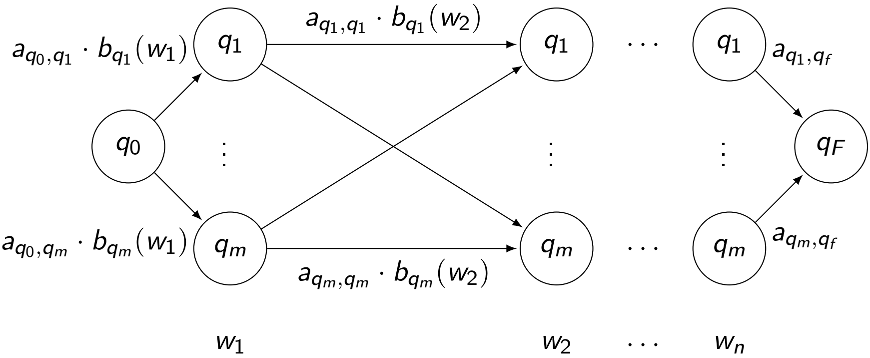
We have already used a trellis in a previous example for Viterbi algorithm.

Given input $w_{1:n} = w_1, w_2, \ldots, w_n$

- introduce special nodes for q_0 and q_f
- for each token w_i and for each state $q \neq q_0, q_f$, introduce a node with label q
- for each pair of nodes q, q' associated with tokens w_{i-1}, w_i, respectively, introduce an arc with weight provided by the product $a_{q,q'} \cdot b_{q'}(w_i)$
- introduce special arcs for node q_f

Each path through the trellis corresponds to a sequence $t_{1:n}$ of states consistent with input $w_{1:n}$.

Trellis

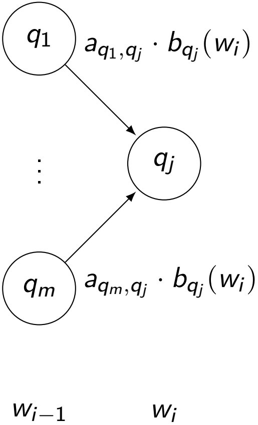

$a_{q_0,q_1} \cdot b_{q_1}(w_1)$
q_1
$a_{q_1,q_1} \cdot b_{q_1}(w_2)$
q_1
$\cdots$
q_1
a_{q_1,q_f}
q_0
$\vdots$
$\vdots$
$\vdots$
q_F
$a_{q_0,q_m} \cdot b_{q_m}(w_1)$
q_m
q_m
$\cdots$
q_m
a_{q_m,q_f}
$a_{q_m,q_m} \cdot b_{q_m}(w_2)$
w_1
w_2
$\cdots$
w_n

Trellis

In the forward algorithm we compute one value for each node

- the value at initial node q_0 is 1
- the value at each intermediate node is $\alpha[q, i]$, computed by summing a sequence of values, one for each incoming edge
- for each incoming edge, this value is obtained by multiplying edge value and value of previous node
- value at final node q_f is the desired probability $P(w_{1:n})$

Summation in trellis



$$\begin{aligned}\alpha[q_j, i] \ = \\ &\alpha[q_1, i-1] \cdot a_{q_1,q_j} \cdot b_{q_j}(w_i) + \\ &\cdots \ + \\ &\alpha[q_m, i-1] \cdot a_{q_m,q_j} \cdot b_{q_j}(w_i)\end{aligned}$$

Backward algorithm

Silvio Kundt from Unsplash

Backward algorithm

The **backward algorithm** uses a two-dimensional table $\beta[q, i]$ denoting the sum of probabilities of **all paths** that start at state q, scan sequence $w_{(i+1):n}$, and reach state q_f.

Formally this is the **joint probability** of $w_{(i+1):n}$ and state q at i, and can be expressed as

$$\beta[q, i] \quad = \quad \sum_{\substack{t_{(i+1):n} \\ \in \mathcal{Y}(w_{(i+1):n}) \\ \text{s.t. } t_i = q}} P(t_{(i+1):n}, w_{(i+1):n})$$

Each of the above quantities is called **backward probability**.

Backward algorithm

Initialisation step: for all q:

- $\beta[q, n] = a_{q,q_f}$

Recursive step: for all $i = n - 1, \ldots, 1$ and for all q:

- $\beta[q, i] = \sum_{q'} \beta[q', i + 1] \cdot a_{q,q'} \cdot b_{q'}(w_{i+1})$

Termination step:

- $\beta[q_0, 0] = \sum_{q'} \beta[q', 1] \cdot a_{q_0,q'} \cdot b_{q'}(w_1)$

After execution of the algorithm we have

$$\beta[q_0, 0] = \alpha[q_f, n + 1] = P(w_{1:n})$$

Duality

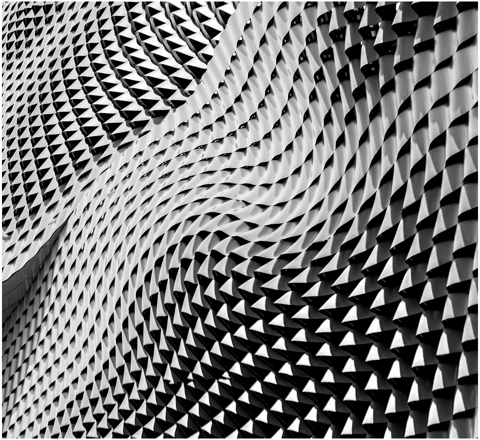
We observe that the backward probability is the **dual** of the forward probability. More precisely, we have

$$\alpha[q,i] \cdot \beta[q,i] \quad = \quad \sum_{\substack{t_{1:n} \in \mathcal{Y}(w_{1:n}) \\ \text{s.t. } t_i = q}} \quad P(t_{1:n}, w_{1:n})$$

In words, this is the probability of all paths in the HMM trellis for $w_{1:n}$ that go through state q at position i.

A relation similar to the above plays an important role in our unsupervised learning algorithm.

Forward-backward algorithm



Ricardo Gomez Angel from Unsplash

Forward-backward algorithm

We have already seen that, given a corpus annotated with POS tags, we can do supervised training of HMM using the relative frequency estimator.

Suppose we are given an **unannotated corpus** and a tagset. Now we **cannot count** transitions and emissions directly, because we don't know which path through the HMM is the right one.

Can we train HMM with tags as internal states? This is possible using the forward-backward algorithm.

Also called Baum-Welch algorithm.

Forward-backward algorithm

To keep discussion simple, assume we have a single unannotated sentence $w_{1:n}$ to train the model.

Let vector θ be an **assignment** for all the parameters $a_{q,q'}$ and $b_q(w_i)$ of the HMM.

We write $P_\theta(t_{1:n}, w_{1:n})$ to denote the joint distribution for $t_{1:n}$ and $w_{1:n}$, $t_{1:n} \in \mathcal{Y}(w_{1:n})$, based on θ.

Forward-backward algorithm

All of the **expectations** defined below are computed under distribution $P_\theta(t_{1:n} \mid w_{1:n})$, over all paths in $\mathcal{Y}(w_{1:n})$.

$c(q, q')$ is the expectation of the transition (q, q').

$c(q, u)$ is the expectation of the emission of symbol $u \in V$ at state q.

$c(q)$ is the expectation of each state q.

Forward-backward algorithm

The **forward-backward algorithm** is an iterative algorithm for the unsupervised learning of parameter vector θ.

The algorithm starts with some initial assignment θ, and updates θ by **iterating** the following steps

- E-step (expectation) computes feature expectations $c(q, q')$, $c(q, u)$ and $c(q)$, on the basis of P_θ
- M-step (maximization) estimates a new assignment $\hat{\theta}$, in a way that maximizes the log-likelihood of the training data

The algorithm stops when the parameters do not change much anymore.

E-step

We show how to compute expectations $c(q, q')$, $c(q, u)$ and $c(q)$.

The sum of probabilities of all paths $t_{1:n} \in \mathcal{Y}(w_{1:n})$ that go through transition (q, q') at input position i is

$$\alpha[q, i] \cdot a_{q,q'} \cdot b_{q'}(w_{i+1}) \cdot \beta[q', i+1]$$

Dividing by the sum of probabilities of all paths, we obtain the probability of (q, q') at position i given $w_{1:n}$

$$c(q, q', i) = \frac{\alpha[q, i] \cdot a_{q,q'} \cdot b_{q'}(w_{i+1}) \cdot \beta[q', i+1]}{\sum_{t_{1:n} \in \mathcal{Y}(w_{1:n})} P(t_{1:n}, w_{1:n}) = P(w_{1:n})}$$

Summing up for all positions i in $w_{1:n}$ we get

$$c(q, q') = \sum_{i} c(q, q', i)$$

E-step

The sum of probabilities of all paths $t_{1:n} \in \mathcal{Y}(w_{1:n})$ that go through transition q at input position i while emitting w_i is

$$\alpha[q, i] \cdot \beta[q, i]$$

Dividing by the sum of probabilities of all paths, we obtain the probability of emitting w_i at state q, given $w_{1:n}$

$$c(q, w_i, i) = \frac{\alpha[q, i] \cdot \beta[q, i]}{P(w_{1:n})}$$

Summing up for all positions i with emission $u \in V$ we get

$$c(q, u) = \sum_{i : w_i = u} c(q, u, i)$$

E-step

We have that

- an occurrence of a state must be followed by a transition
- an occurrence of a state must be associated with an emission

Then it is not difficult to see that

$$c(q) = \sum_{q'} c(q, q') = \sum_{u} c(q, u)$$

M-step

We estimate a new parameter assignment $\hat{\theta}$ based on expectations $c(q, q')$, $c(q, u)$ and $c(q)$.

$$\hat{a}_{q,q'} = \frac{c(q, q')}{c(q)}$$

$$\hat{b}_q(u) = \frac{c(q, u)}{c(q)}$$

Very similar to the relative frequency estimator, but we now use feature expectations in place of counts.

We now have a **refined** probability distribution $P_{\hat{\theta}}(t_{1:n}, w_{1:n})$.

Forward-backward algorithm

The forward-backward algorithm generally converges to some **local optimum**, with a (relative) maximum for the likelihood of training data.

Starting with different initial guesses for parameter vector θ may lead to different solutions (different local optima).

No effective algorithm is known to compute **global** optimum, maximising likelihood of unannotated training material.

The forward-backward algorithm is an instance of a more general class of algorithms called **EM** (expectation-maximisation).

Research papers

Iñaki del Olmo from Unsplash

Research papers

Title: An Interactive Spreadsheet for Teaching the Forward-Backward Algorithm
Authors: Jason Eisner
Conference: Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics
Content: This paper introduces the forward-backward algorithm.

`https://aclanthology.org/W02-0102`

Research papers

Title: Inside-Outside and Forward-Backward Algorithms Are Just Backprop (tutorial paper)
Author: Jason Eisner
Conference: Workshop on Structured Prediction for NLP
Content: Computing the expected counts of features requires an algorithm such as inside-outside or forward-backward. Conveniently, each such algorithm can be obtained by automatically differentiating an algorithm that computes the log-probability of the the sentence. This mechanical procedure produces correct and efficient code.

`https://www.aclweb.org/anthology/W16-5901.pdf`

HMM Limitations

In general, it is hard for generative models like HMMs to add rich features directly in a clean way.

We might need special morphological features in HMM for unknown words.

Independence assumptions for the model features are quite strong.

When multiplying model features we assume independence, if this does not hold sequence probabilities do not sum up to one.

Inconsistency between local training and global testing: probabilities of each label are trained locally, but output is the highest probability sequence, which is searched globally.

This mismatch is detrimental to performance.

Conditional random fields

Federico Respini from Unsplash

Conditional random fields

Conditional random fields (CRF) are discriminative sequence models based on log-linear models. Discriminative classifiers learn what features from the input are most useful to discriminate between the different possible classes.

The model can only distinguish the classes, perhaps without learning much about them: it is unable to generate observations/examples.

We describe the **linear chain CRF**, the version of CRF that is most commonly used for language processing, and the one whose conditioning closely matches HMM.

Conditional random fields

Let $x_{1:n} = x_1, x_2, \ldots, x_n$ be the input word sequence, and let $\mathcal{Y}(x_{1:n})$ be the set of all possible tag sequences $y_{1:n} = y_1, y_2, \ldots, y_n$ for $x_{1:n}$.

CRFs solve the problem:

$$\widehat{y}_{1:n} = \operatorname*{argmax}_{y_{1:n} \in \mathcal{Y}(x_{1:n})} P(y_{1:n} \mid x_{1:n})$$

In contrast with HMMs, we directly model the posterior probability $P(y_{1:n} \mid x_{1:n})$.

Conditional random fields

In contrast to HMM, the CRF does not compute a probability for each tag at each time step.

Instead, at each time step the CRF computes **log-linear functions** over a set of relevant local features, and these features are aggregated and normalised to produce a global probability.

In this way we do not need the independence assumption of HMM.

We can think of CRF as a multinomial logistic regression model, but applied to a full sequence pair $(x_{1:n}, y_{1:n})$ rather than a pair (x, y) of individual tokens.

See Jurafsky & Martin §5.3 for multinomial logistic regression.

Conditional random fields

Let us assume we have K **global feature** functions $F_k(x_{1:n}, y_{1:n})$, and weights w_k for each feature.

We define

$$P(y_{1:n} \mid x_{1:n}) \quad = \quad \frac{\exp\left(\sum_{k=1}^{K} \ w_k F_k(x_{1:n}, y_{1:n})\right)}{\sum_{y'_{1:n} \in \mathcal{Y}(x_{1:n})} \ \exp\left(\sum_{k=1}^{K} \ w_k F_k(x_{1:n}, y'_{1:n})\right)}$$

The denominator is the so-called **partition function** $Z(x_{1:n})$, a normalization term that only depends on the input sequence $x_{1:n}$

$$Z(x_{1:n}) \quad = \quad \sum_{y'_{1:n} \in \mathcal{Y}(x_{1:n})} \exp\left(\sum_{k=1}^{K} \ w_k F_k(x_{1:n}, y'_{1:n})\right)$$

Conditional random fields

We compute the global features by decomposing into a sum of **local features**, for each position i in $y_{1:n}$

$$F_k(x_{1:n}, y_{1:n}) \quad = \quad \sum_{i=1}^{n} f_k(y_{i-1}, y_i, x_{1:n}, i)$$

Practical assumption: Each local feature depends on the **current** and **previous** output tokens, y_i and y_{i-1} respectively.

The specific constrain above characterises **linear chain CRF**. This limitation makes it possible to use the Viterbi algorithm.

In contrast, a general CRF allows a feature to make use of any output token, representing long-distance dependencies and requiring more complex inference/decoding algorithms.

Local features

Kilimanjaro STUDIOz on Unsplash

Local features

For a **predicate** x, we write $\mathbb{I}\{x\}$ to denote 1 if x is true and 0 otherwise.

Example of features in linear-chain CRF:

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

What features to use is a decision by the system designer.

This task is also called feature engineering or feature selection.

Local features

To avoid feature handwriting, specific features are automatically instantiated from **feature templates**.

Here are some templates using information from $y_{i-1}, y_i, x_{1:n}, i$:

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

Example : Using dataset

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

when x_i is the word **back** the following features would be generated (indices generated at random)

- $f_{3743} : \mathbb{I}\{x_i = \text{back}, y_i = \text{VB}\}$
- $f_{156} : \mathbb{I}\{y_i = \text{VB}, y_{i-1} = \text{MD}\}$
- $f_{99732} : \mathbb{I}\{y_i = \text{VB}, x_{i-1} = \text{will}, x_{i+2} = \text{bill}\}$

Local features

It is important to use special features for unknown words.

Word shape features are used to represent letter patterns.

Example : word 'DC10-30' can be captured by feature $f : \mathbb{I}\{x_i = \text{X+d+-d+}\}$, where X denotes capital letters, d denotes digits, and $+$ is the standard regular expression operator.

Prefix and **suffix** features are used to represent word morphological patterns.

Example : word 'well-dressed' can be captured by feature $f : \mathbb{I}\{\text{prefix}(x_i) = \text{well-}\}$.

The result of the above feature templates can be a very large set of features. Generally, features are thrown out if they have count smaller than some **cutoff** in the training set.

Inference

SpaceX on Unsplash

Inference

The inference problem for linear-chain CRF is espressed as:

$$\begin{aligned}
\widehat{y}_{1:n} &= \operatorname*{argmax}_{y_{1:n} \in \mathcal{Y}(x_{1:n})} P(y_{1:n} \mid x_{1:n}) \\
&= \operatorname*{argmax}_{y_{1:n} \in \mathcal{Y}(x_{1:n})} \frac{1}{Z(x_{1:n})} \cdot \exp\left(\sum_{k=1}^{K} w_k F_k(x_{1:n}, y_{1:n})\right) \\
&= \operatorname*{argmax}_{y_{1:n} \in \mathcal{Y}(x_{1:n})} \sum_{k=1}^{K} w_k F_k(x_{1:n}, y_{1:n}) \\
&= \operatorname*{argmax}_{y_{1:n} \in \mathcal{Y}(x_{1:n})} \sum_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(y_{i-1}, y_i, x_{1:n}, i)
\end{aligned}$$

Inference

We can still use the Viterbi algorithm, because the linear-chain CRF depends at each time-step on only one previous output token y_{i-1}.

Recall that for HMMs the recursive step states, for each position i and state q:

$$vt[q, i] = \max_{q'} \; vt[q', i-1] \cdot a_{q',q} \cdot b_q(w_i)$$

For CRF we need to replace the prior and the likelihood probabilities with the CRF features (t, t' are tags):

$$vt[t, i] = \max_{t'} \; vt[t', i-1] + \sum_{k=1}^{K} \; w_k f_k(t', t, x_{1:n}, i)$$

Training

Adrien Converse from Unsplash

Training

The parameters w_i in CRF can be learned in a **supervised** way as in the method of logistic regression.

See Jurafsky & Martin §5.5 and §5.6.

We minimise the **negative log-likelihood** as our objective function.

It is possible to show that this is equivalent to minimising the expectation of the cross-entropy of all the conditional probability distributions.

As in the case of multinomial logistic regression, **L1** or **L2 regularization** is important.

To optimise the objective function, we use **stochastic gradient descent**. The local nature of linear-chain CRFs can be exploited to efficiently compute the necessary derivatives.

Training

Let $D = \{(y^{(h)}_{1:n_h}, x^{(h)}_{1:n_h}) \mid 1 \leqslant h \leqslant N\}$ be a training set, where each $x^{(h)}_{1:n_h}$ is a sentence and each $y^{(h)}_{1:n_h}$ is the associated sequence label. Let also $\mathbf{w}$ be the parameter vector.

The **objective function** is:

$$\begin{aligned}
\mathcal{L}(D, \mathbf{w}) &= \frac{\lambda}{2}\|\mathbf{w}\|^2 - \sum_{h=1}^{N} \log P(y^{(h)}_{1:n_h} \mid x^{(h)}_{1:n_h}) \\
&= \frac{\lambda}{2}\|\mathbf{w}\|^2 - \sum_{h=1}^{N} \log \frac{1}{Z(x^{(h)}_{1:n_h})} \cdot \exp\left(\sum_{k=1}^{K} w_k F_k(y^{(h)}_{1:n_h}, x^{(h)}_{1:n_h})\right) \\
&= \frac{\lambda}{2}\|\mathbf{w}\|^2 - \sum_{h=1}^{N} \left(\sum_{k=1}^{K} w_k F_k(y^{(h)}_{1:n_h}, x^{(h)}_{1:n_h})\right) + \sum_{h=1}^{N} \log Z(x^{(h)}_{1:n_h})
\end{aligned}$$

Training

In order to compute the gradient of $\mathcal{L}(D, \mathbf{w})$ we have to be able to efficiently compute **feature expectations**:

$$\sum_{y_{1:n} \in \mathcal{Y}(x_{1:n})} P(y_{1:n} \mid x_{1:n}) \cdot F_k(x_{1:n}, y_{1:n})$$

In **practice**, feature expectations are computed “under the hood” by modern software libraries, as Pytorch, using automatic differentiation of the objective function.

Alternatively, feature expectations can be efficiently computed using the **forward-backward algorithm**, which we have already seen for unsupervised learning with HMMs.

Research papers

Iñaki del Olmo from Unsplash

Research papers

Title: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data
Authors: John D. Lafferty, Andrew McCallum, Fernando C. N. Pereira
Conference: ICML 2001
Content: This paper introduces a framework for building probabilistic models to segment and label sequence data. Conditional random fields offer several advantages over hidden Markov models, including the ability to relax strong independence assumptions made in those models. Conditional random fields also avoid a fundamental limitation of hidden Markov models, which can be biased towards states with few successor states.

`http://www.aladdin.cs.cmu.edu/papers/pdfs/y2001/crf.pdf`

Title: The Label Bias Problem
Author: Awni Hannun
Blog: Writing About Machine Learning
Content: Many sequence classification models suffer from the label bias problem. Understanding the label bias problem and when a certain model suffers from it is essential to understand the design of models like conditional random fields.

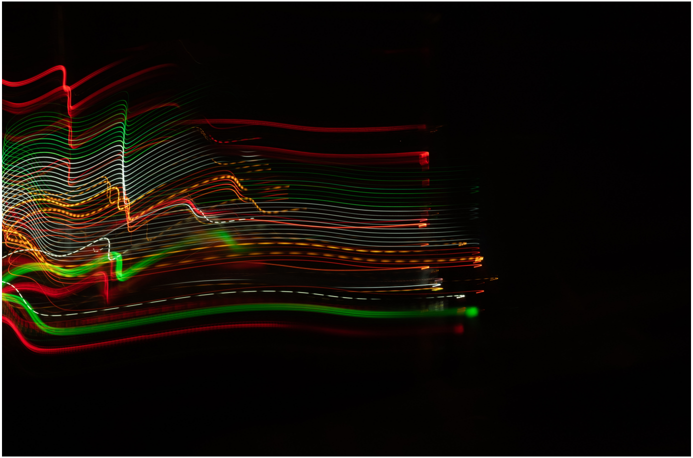
`https://awni.github.io/label-bias/`

Limitations

One limitation with HMM and CRF architectures is that the models are exclusively run **left-to-right**.

Bidirectional models are quite standard for deep learning, as we will see with the BiLSTM models to be introduced later.

Neural POS tagger



Omid Armin from Unsplash

Neural POS tagger

In neural network approaches to POS tagging, we construct distributed feature representations (dense vectors) for each tagging decision, based on the word and its context.

We present two main approaches.

Local search: Neural networks can perform POS tagging as a per-token classification decision.

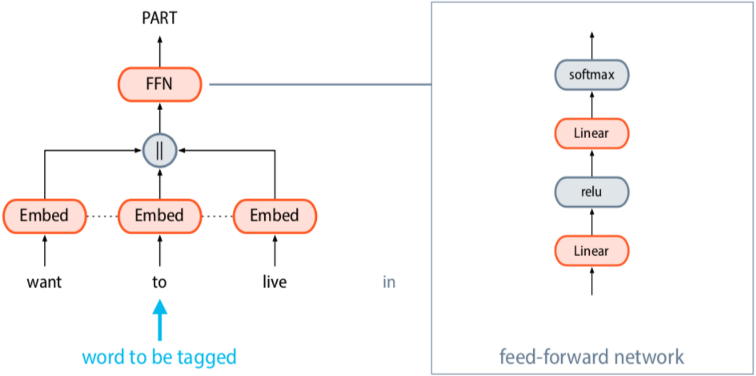
Global search: Alternatively, feature representations can be combined with the Viterbi algorithm to tag the entire sequence globally (joint tagging).

Local search

©Garrett Wade

Fixed-window neural model

Fixed-window models use a feed-forward neural network to implement local search.



Second layer maps the feature space into the tagset space.

Fixed-window neural model

The fixed-window model is very efficient, since it limits the context from which information can be extracted.

Same limitation of Markov models.

Sliding windows makes it difficult for network to learn systematic patterns arising from phenomena like constituency, since patterns are shifted to different positions.

Recurrent neural model

Let $x_1, x_2, \ldots, x_n$ be the input word sequence and $y_1, y_2, \ldots, y_n$ be the associated output POS tags.

We assume word embeddings $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ for $x_1, x_2, \ldots, x_n$.

Recurrent neural networks can be generally described as implementing the following recursive relation, for $t = 1, \ldots, n$:

$$\begin{aligned} \mathbf{h}_t &= f(g(\mathbf{e}_t, \mathbf{h}_{t-1})) \\ &= f(\mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{W}^e \mathbf{e}_t + \mathbf{b}) \end{aligned}$$

with f some nonlinear component.

$\mathbf{W}^h$, $\mathbf{W}^e$, $\mathbf{b}$ are learnable parameters. Gated RNNs such as LSTM networks implement more complex recurrence relations.

Recurrent neural model

We score each POS tag y by means of a **linear scalar** function of hidden state vector $\mathbf{h}_t$, and then retrive the highest score tag for x_t:

$$\begin{aligned} \psi(y, \mathbf{h}_t) &= \boldsymbol{\beta}_y \cdot \mathbf{h}_t \\ \widehat{y}_t &= \operatorname*{argmax}_y \; \psi(y, \mathbf{h}_t) \end{aligned}$$

The score $\psi(y, \mathbf{h}_t)$ can also be converted into a probability using the softmax operation:

$$P(y \mid x_{1:t}) = \frac{\exp \psi(y, \mathbf{h}_t)}{\sum_{y'} \exp \psi(y', \mathbf{h}_t)}$$

Cross-entropy or hinge loss can be used as objective function.

Recurrent bidirectional model

Hidden state vector $\mathbf{h}_t$ encodes left context up to position t but it **ignores** subsequent tokens, which might be relevant to y_t as well.

Bidirectional RNN are used to address this problem:

$$\begin{aligned}
\overrightarrow{\mathbf{h}}_t &= g(\mathbf{e}_t, \overrightarrow{\mathbf{h}}_{t-1}) \\
\overleftarrow{\mathbf{h}}_t &= g'(\mathbf{e}_t, \overleftarrow{\mathbf{h}}_{t+1}) \\
\mathbf{h}_t &= [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t]
\end{aligned}$$

The scoring function $\psi(y, \mathbf{h}_t)$ is then applied to the concatenation of the two vectors.

The model is still based on **local search**, each tagging decision is made independently and no global search is performed.

Global search

©Beth Durham

Neural structured prediction

Neural sequence labelling can be combined with global search by augmenting local scores as:

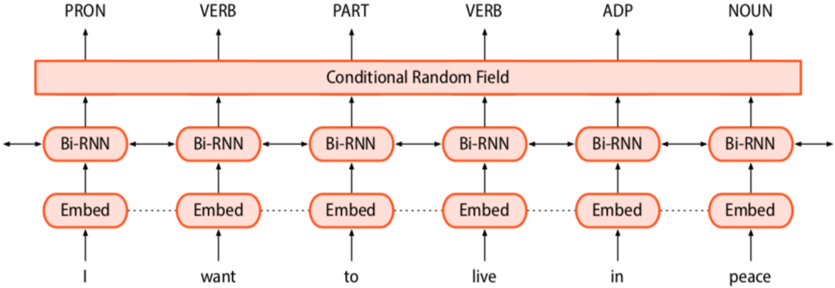
$$\psi(y_t, y_{t-1}, \mathbf{h}_t) \quad = \quad \boldsymbol{\beta}_{y_t} \cdot \mathbf{h}_t + \eta_{y_{t-1}, y_t}$$

where η_{y_{t-1}, y_t} is a scalar, **learnable** parameter for the POS tag transition (y_{t-1}, y_t).

As in CRF, Viterbi algorithm is used for **inference** (joint tagging) and the forward-backward algorithm is used for **training**.

Neural structured prediction

Global search neural model can be thought of as a **combination** of recurrent bidirectional model and CRF: the feature vector is extracted by the neural model and provided to the CRF algorithm.



When Bi-LSTM are used, the model is called LSTM-CRF.

Research papers

Iñaki del Olmo from Unsplash

Research papers

Title: Bidirectional LSTM-CRF Models for Sequence Tagging
Authors: Zhiheng Huang, Wei Xu, Kai Yu
Repository: arXiv:1508.01991 [cs.CL]
Content: This paper proposes a combination of Long Short-Term Memory based models with a Conditional Random Field layer. The model can produce state of the art accuracy on POS, chunking and NER data sets.

`https://arxiv.org/abs/1508.01991`

Morphologically rich languages

Morphologically rich languages (MRL) have much more information than English coded into word morphology, like **case** (nominative, accusative, genitive) or **gender** (masculine, feminine).

This information is really important for tasks like parsing and coreference resolution.

Tagsets for MRL are therefore sequences of morphological tags rather than a single primitive tag.

Instances of MRL are Arabic, Czech, Hungarian, Turkish, etc. Tagsets for these languages can be 4 to 10 times larger than English.

With such large tagsets, **specialized** POS taggers need to be developed where each word needs to be morphologically analyzed.

Sequence labelling

Matt Briney on Unsplash

Sequence labelling

POS tagging is an instance of a more general problem called **sequence labelling**, assigning to an input word sequence $x_1, x_2, \ldots, x_n$ an output sequence $y_1, y_2, \ldots, y_n$ over an arbitrary set of categories.

Again, this is a structured prediction problem, and categories must be assigned contextually.

We are going to briefly overview other NLP tasks that can be cast as sequence labelling problems.

Named entity recognition

Named entity recognition (NER) seeks to locate multi-word expressions referring to entities such as person names, organizations, locations, time expressions, quantities, monetary values, etc.

NER is a useful first step in lots of natural language understanding tasks.

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.

Besides tagging, in NER we also need to find the proper **span** of the target expression.

Named entity recognition

The standard approach for span-recognition is BIO tagging.

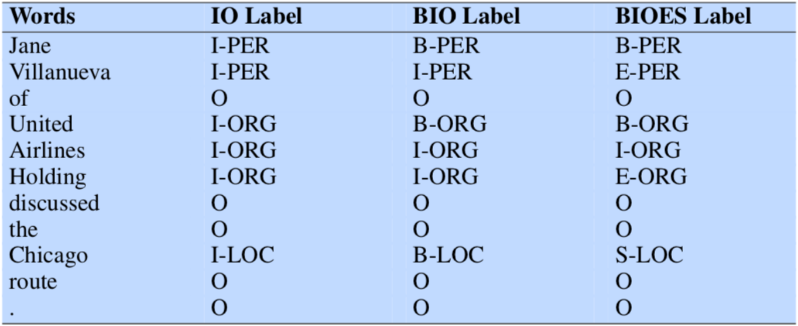
In **BIO tagging**

- tokens that begin a span are marked with label B
- tokens that occur inside a span in a position other than the leftmost one are marked with I
- tokens outside of any span of interest are marked with O

Spans never overlap.

Example

BIO tagging along with alternative tagging techniques for span.



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

Note the difference between the sequences BI and BB, indicating one 2-word expression vs. two 1-word expressions.

Datasets

CoNLL-2003

Named entity recognition dataset released as a part of CoNLL-2003 shared task: language-independent named entity recognition. Covering two languages: English and German.

WikiNER Dataset

Manually-labelled Wikipedia articles across nine languages: English, German, French, Polish, Italian, Spanish, Dutch, Portuguese and Russian.

Many more datasets in kaggle: `https://www.kaggle.com`.

Evaluation

Evaluation

Named entity recognizers are evaluated by precision, recall, and F1-score.

Precision is the percentage of named entities found by the learning system that are correct.

Recall is the percentage of named entities present in the corpus that are found by the system.

F1-score is the harmonic mean of the two.

Evaluation

More **specialized evaluation** can be obtained by considering individual tokens rather than entire entities.

Precision, recall and F1-score are computed for each individual BIO label. This accounts for

- assignment of wrong entity types
- wrong entity boundaries

In this **multi-class** setting, people also distinguish between different types of per-class F1-score

- macro averaged (unweighted mean)
- micro averaged (accuracy)
- weight averaged (considering each class support)

Aspect-based sentiment analysis

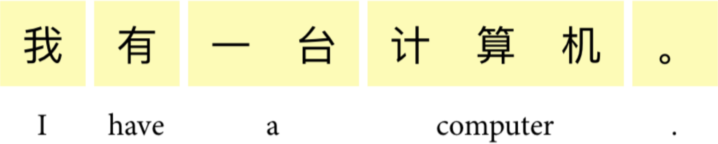
Aspect-based sentiment analysis aims to identify the aspects of the entities being reviewed and to determine the sentiment the reviewers express for each aspect.

More fine-grained task than sentiment analysis.

Word segmentation

Word segmentation is the task of dividing a text into its component words.

Languages which do not have a trivial word segmentation process include Chinese and Japanese, where words are not delimited.



我	有	一 台	计 算 机	。
I	have	a	computer	.

Code switching

Code switching is the phenomenon of switching between languages in speech and in text.

Quite common in online social media.

Code switching can be viewed as a sequence labelling problem, where the goal is to label tokens representing switch points.

Rule-based systems

While machine learning sequence models are the norm in academic research, many commercial approaches are often based on **hand-written** lists of rules, with some smaller amount of supervised machine learning.

This is especially true for NER.

One common approach is to

- make repeated rule-based passes over a text, starting with rules with very high precision but low recall
- use machine learning methods in subsequent stages, that take the output of the first pass into account