EC999: Part of Speech Tagging

Thiemo Fetzer

University of Chicago & University of Warwick

November 19, 2016

Part of Speech Tagging

Classical *Part's of speech* are: nouns, verbs, pronouns, preopositions, adverbs, conjunctions, participles and articles.

Part of spech (POS) tagging is an essential step in language processing that is very useful for a range of auxiliary tasks.

- dimensionality reduction (removing words)
- word sense disambiguation
- Named Entity Recognition
- information extraction

$\Rightarrow$ we will introduce the formalization of common tools for POS tagging and introduce use pipelines in *R*.

Part of Speech Tagging

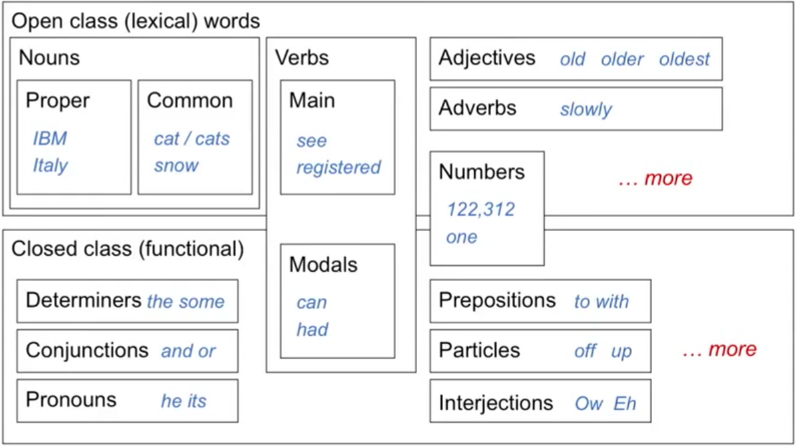
Classical *Part's of speech* are: nouns, verbs, pronouns, preopositions, adverbs, conjunctions, participles and articles.

Part of spech (POS) tagging is an essential step in language processing that is very useful for a range of auxiliary tasks.

- dimensionality reduction (removing words)
- word sense disambiguation
- Named Entity Recognition
- information extraction

$\Rightarrow$ we will introduce the formalization of common tools for POS tagging and introduce use pipelines in *R*.

Part of Speech Tagging



Open Versus Closed Class

Typically two types of high level groups are defined

- **Closed class**: considered as closed as the set of closed class words hardly changes over time.
  - determiners: a, an, the
  - pronouns: I, he, she, they
  - prepositions: over, under, near
- **Open class**: New entries into classes of all types, think of proper nouns becoming verbs - such as "Google" and "to google".

Word Class Ambiguity makes this a challenging task

Part of speech tagging is challenging as words can be members of multiple classes, depending on the *context* of use.

- `get/VB off/IN my/PRP$ back/NN`
- `win/VB the/DT voters/NNS back/RB`
- `I/PRP promise/VBP to/TO back/VB the/DT bill/NN ./.`

Part of speech tagging task is a relatively *easy* task. For every word that has ambiguity, there is a constrained set of ambiguous tags to chose from. Most POS implementation work off the information contained by the word itself and on information contained by small *windows* around the word.

The Penn Treebank Part-of-Speech Tagset

There are many lists of parts-of-speech, most modern language processing on English uses the 45-tag Penn Treebank tagset.

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	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, sing.	*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*			

There are other tagsets T with anything between 8 to 1,200, but this is the most commonly used

A naive POS tagger

The *baseline* POS tagger uses a simple tag-allocation rule: assign a tag $t^* \in T$ to a word w_j if

$$t^* = argmaxP(t_i|w_j)$$

This tag allocation rule assigns the tag t to a word w_j that has the highest likelihood for that word. These conditional likelihoods can be estimated from some tagged *training* data.

It achieves surprising accuracy of around 90%.

Reason for surprising high accuracy is due to fact that a lot of *stopwords* that make up the bulk of the quantity of tokens of text have mostly unambigous tags.

Accuracy of Naive POS

For example for the word `well`:

- `Get/VB well/RB soon/RB !/.`
- `This/DT oil/NN well/NN is/VBZ profitable/JJ ./.`

For the word `well`, a training corpus suggests

Part-of-Speech	Total	(over Absolute Total)	Probability
adv	237,644,762	337,697,034	81.03%
adj	38,018,925	"	11.26%
x	20,818,507	"	6.16%
noun	4,839,300	"	1.43%
verb	296,019	"	0.09%
pron	42,918	"	0.01%
.	17,877	"	0.01%
det	12,313	"	0
num	3,822	"	0
prt	2,270	"	0
adp	31	"	0
Totals	**337,697,034**		**100%**

This suggests that the naive model would suggest that the most likely class for the word is `RB` - adverb form.

A (shallow) deep dive: Hidden Markov Models

One direction to improve on baseline POS tagger is to use information contained in structure around a word. So suppose you have a word sequence $w_1, ..., w_n$ (like a sentence), then the optimization problem that you want to solve is to assign a sequence of tags $t_1, ..., t_n$ to these words, that maximizes the probability

$$(t_1, ..., t_n)* = argmaxP((t_1, ..., t_n)|(w_1, ..., w_n))$$

The underlying (hidden) true states of the world is the correct tag sequence $t_1, ..., t_n$.
It is impractical (impossible) to estimate $P((t_1, ..., t_n)|(w_1, ..., w_n))$ directly from training data due to the sparsity. So we employ a simplifying assumption.

A (shallow) deep dive: Hidden Markov Models

We can apply *Bayes Rule*, so the optimization problem becomes

$$(t_1, ..., t_n)^* = argmax \frac{P((w_1, ..., w_n)|(t_1, ..., t_n))P(t_1, ..., t_n)}{P((w_1, ..., w_n))}$$

This optimization problem is equivalent to solving [why?]

$$(t_1, ..., t_n)^* = argmax P((w_1, ..., w_n)|(t_1, ..., t_n))P(t_1, ..., t_n)$$

A (shallow) deep dive: Hidden Markov Models

For Hidden Markov POS models, we make two additional assumptions

$$P((w_1, ..., w_n)) = \prod_{i=1}^{n} P(w_i|t_i)$$

and

$$P((t_1, ..., t_n)) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

This allows us to rewrite the optimization problem as

$$(t_1, ..., t_n)^* = argmax \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

An example: Hidden Markov Models

Consider the sentence

`Janet will back the bill`

which is correctly tagged as

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

We obtain the following information from a tagged training corpus.

An example: Hidden Markov Models

	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	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Displaying $P(w_i|t_i)$ and $P(t_i|t_{i-1})$.

An example: Hidden Markov Models

	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

Displaying $P(w_i|t_i)$ and $P(t_i|t_{i-1})$.

An example: Finding optimal path

	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	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

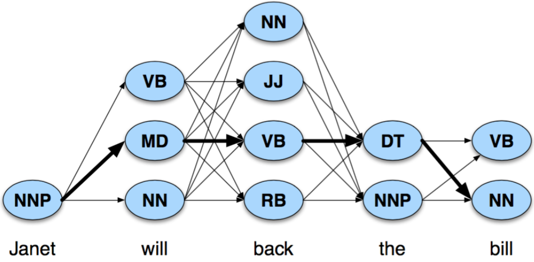
- In the corpus, the word `Janet` only appears with tag `NNP`.
- the word `will` has three possible tags `MD`, `VB`, `NN`.
- The probability that a random word of type modal (MD) is the word `will` is $0.31 = P(\texttt{will}|\texttt{MD})$

An example: Finding optimal path

	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

- Transition matrix presents estimated $P(t_i|t_{i-1})$.
- the row sums should add to 1 - they dont since not the whole tagset is displayed.
- $P(\texttt{VB}||\texttt{MD}) = 0.79$, probability that `MD` is followed by tag `VB`.

An example: Finding optimal path



- The optimization problem can be modelled as an optimization problem on a *directed path*.
- We want to find the path that has highest likelihood.
- Brute forcing - the computation of all possible values for $\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$ is computationaly extremely inefficient, and becomes infeasible very fast.
- *Viterbi algorithm* is a dynamic programming algorithm that solves this program efficiently.

Part of Speech Tagging in *R*

In *R* an easily accessible POS tagging tool that performs very well is accessible through the packages `OpenNLP`, which makes Apache's Open NLP platform accessible (`https://opennlp.apache.org/`).

It is a bit slow and the Apache NLP package is memory intensive (requests around)

Rather than working with a hidden markov model, its a maximum entropy classifier - which is just a fancy way of saying "logistic regression". We will introduce logistic regression for simple classification tasks.

Part of Speech Tagging in *R*

We will work with a developmental extension called the `tagger` package. Speed is an issue with NLP pipelines, OpenNLP extension takes arund 0.1 seconds per "sentence".

```
library(NLP)
library(openNLP)
# this is developmental, can be installed with the next two lines of code.
library(tagger)

## this installs 'pacman' which is a package to load developmental R extensions
if (!require("pacman")) install.packages("pacman")
pacman::p_load_gh(c("trinker/termco", "trinker/tagger"))

temp <- tag_pos("Janet will back the bill")

temp[[1]]
```

```
##     NNP      MD      VB      DT      NN
## "Janet"  "will"  "back"   "the"  "bill"
```

```
data.frame(tokens = temp[[1]], tags = names(temp[[1]]))
```

```
##     tokens tags
## NNP  Janet  NNP
## MD    will   MD
## VB    back   VB
## DT     the   DT
## NN    bill   NN
```

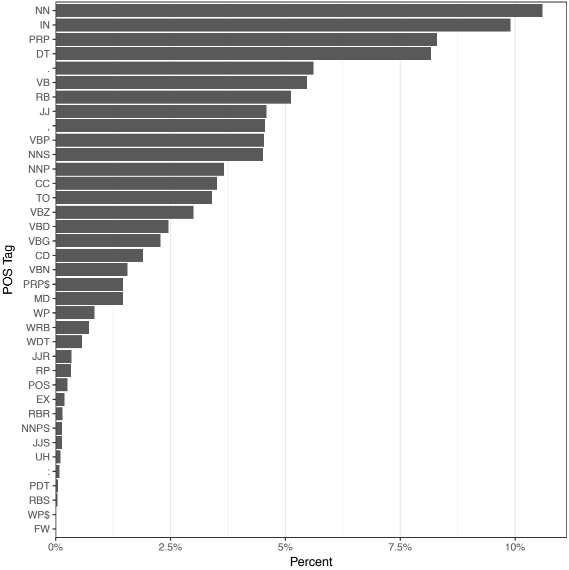
Part of Speech Tagging in *R*

```
data(presidential_debates_2012)

TAGGED <- tag_pos(presidential_debates_2012$dialogue)

head(TAGGED)
```

```
## [[1]]
##           PRP             MD             VB             IN             RB             IN
##          "We"          "'ll"         "talk"        "about" "specifically"        "about"
##            NN             NN             IN             DT             NN              .
##      "health"         "care"           "in"            "a"       "moment"            "."
##
## [[2]]
##         CC         WP        VBP        PRP         VB         DT         NN         NN
##      "But"     "what"       "do"      "you"  "support"      "the"  "voucher"   "system"
##          ,        NNP          .
##        "," "Governor"        "?"
##
## [[3]]
##         WP        PRP        VBP        VBZ         DT         NN         IN         JJ
##     "What"        "I"  "support"       "is"       "no"   "change"      "for"  "current"
##        NNS         CC         IN        NNS         TO        NNP          .
## "retirees"      "and"     "near" "retirees"       "to" "Medicare"        "."
##
## [[4]]
##          CC          DT          NN         VBZ         VBG          NN          CD
##       "And"       "the" "president"  "supports"    "taking"    "dollar"     "seven"
##          CD          CD          CD          IN          IN          DT          NN
##   "hundred"   "sixteen"   "billion"       "out"        "of"      "that"   "program"
##           .
##         "."
##
## [[5]]
##         CC         WP         IN         DT        NNS          .
##      "And"     "what"    "about"      "the" "vouchers"        "?"
##
```

Part of Speech Tagging in *R*

```r
plot(TAGGED)
```



Going forward

Part of Speech Tagging is an important and often neccesary task for NLP.

We will make use of POS tags in a range of applications, so a basic understanding is needed.