CSME 206A – Natural Language & Speech Processing **Spring Semester**

Lecture 13: Parts of Speech Tagset and Statistical-based Tagging

Lecturer: K.R. Chowdhary *: Professor of CS*

13.1 Part of Speech Tagset

There are standard eight parts of speech (POS) in English: *noun, verb, pronoun, preposition, adverb, conjunction, participle*, and *article*. The POS are also called *word classes, morphological classes*, or *lexical tags*. They are important as they give significant amount of information about word and its neighbors. It is true for nouns and verbs. Also, when we have identified, e.g., possessive pronouns *my, your, his, her, its* and personal pronouns *I, he, you, me*, we are able to identify the vicinity words.

The POS are also used for *Information retrieval*, as knowing POS can help us as which morphological affixes it can have. They can also help in selecting important words, like, nouns, from the text.

Some examples of POS are as follows:

- *Prepositions:* on, under, over, near, by, at, from, to, with
- *Pronouns:* she, who, I, others
- *Wh-pronouns:* what, who, whom, why, where
- *Conjunctions:* and, but, or, as, if, whom
- *Auxiliary verbs:* can, may, should, is, are
- *Participle:* up, down, on, off, in, out, at, by

Words can be analysed into parts-of-speech, which are major lexical syntactic categories, like, as N (Noun), V (Verb), A (Adjective), P (Preposition), or more minor categories, such as $Comp$ (Complementizer), Det (Determiner), Deg (Degree intensifier), and so on. Some examples are as follows:

```
N: car, cars; woman, women...
V: thinks, thinking; sold, selling...
Adj:  old, older, oldest; pedantic...
Prep:  in, on, with(out), although...
Comp:  that, if...
Det:  the, a, those, that, some...
Deg:  so, very...
```

The N, V, A are the categories of the contentful or open-class vocabulary. Membership of these categories is large (as a glance at any dictionary will tell you) and open-ended (people invent new words (neologisms)

like, fax, biro) and often open-class words belong to more than one category (e.g. storm can be a noun or verb, and morphologically-related stormy is an adjective); that is, they are ambiguous in terms of lexical syntactic category. (Some words are ambiguous at the level of lexical semantics though not in terms of lexical syntactic category e.g. match, N: game vs. lighter). Adverbs also form a large open-ended class, but they are highly related to adjectives and often formed by adding the suffix +ly to adjectives (badly, stormily, etc) so we would not give them a separate category but treat them as A[+Adv].

The other categories are those of functional or closed-class words, which typically play a more 'grammatical' role with more abstract meaning. Membership of these categories is smaller and changes infrequently. For example, prepositions convey some meaning but often this meaning would be indicated by case endings or inflection on words in other languages and sometimes there are English paraphrases which dispense with the preposition: "Kim gave a cat to Sandy" / "Kim gave Sandy a cat." Degree intensifiers in adjectival or adverbial phrases very beautiful(ly) convey a meaning closely related to the comparative suffix more beautiful / taller. Determiners, such as the (in)definite articles (the, a), demonstrative pronouns (e.g. this, that) or quantifiers (e.g. some, all) help determine the reference of a noun (phrase) – quite frequently articles are absent or indicated morphologically in other languages (hence the common non-native speaker error of the form: "please, where is train station?").

The complete set of lexical syntactic categories (for English) depends on the syntactic theory, but the smallest sets contain around 20 categories (almost cor- responding to traditional Greek/Latin-derived parts-of-speech) and the largest thousands.

Often words are ambiguous between different lexical categories. What are the possibilities for broken, purchase, that and can? There are diagnostic rules for determining the category appropriate for a given word in context; e.g., if a word follows a determiner, it is a noun, as in, "the song was a hit." If a word precedes a noun, is not a determiner and modifies the noun's meaning, it is an adjective. For example, "the smiling boy laughed." Can you think of an exception to the last rule? These rules and categorical distinctions can be justified by doing distributional analysis both at the level of words in sentences. The process is more long-winded, though. The following template schemata are enough to get you to the rules above, which are abstractions based on identifying the classes, like noun, determiner, and adjective

1. – boy(s) can run
2. – older boy(s) can run
3. The – boy(s) can run
4. The older – can run

There are other ways to make these distinctions too. For example, nouns often refer to fairly permanent properties of individuals or objects, boy, car, etc., verbs often denote transitory events or actions, *smile, kiss*, etc. However, there are many exceptions: *storm, philosophy, weigh, believe*, etc. Linguists have striven to keep syntax and semantics separate and justify syntactic categories on distributional grounds, but there are many interactions between meaning and syntactic behavior.

13.1.1 Penn Treebank Tagset

Parts-of-speech tagging or tagging in short is process of assigning a parts-of-speech or other lexical class marker to each word in a given text. The tagging is also called *tokenization* in terms of computer based processing for natural language text. The parts-of-speech tagging (grammatical tagging) or disambiguation of word-category, is a process of marking-up word in a text (corpus) corresponding to a particular POS. This

is carried out based on its definition as well as its context[1]. Various POS in English language are: noun, verb, adjective, adverb, pronoun, preposition, conjunction, and interjection.

There are tag-sets used for parts of speech Tagging. The Table 13.1 shows the Penn *Treebank tagset*. The tagged version of the Penn Treebank corpus is produced in two stages, using a combination of automatic POS assignment and manual correction.

Table 13.1: Penn Treebank POS tagset

Tag	Description	
1	CC	Conjunction, coordinating
2	CD	Numeral, cardinal
3	DT	Determiner (e.g., a,an, the, this, that, those)
4	EX	existential there
5	FW	Foreign word
6	IN	Preposition (e.g., of, by, in)
7	JJ	Adjective (e.g., yellow, other)
8	JJR	Adjective, comparative (e.g., bigger)
9	JJS	Adjective, superlative (e.g., biggest)
10	LS	List item marker
11	MD	Modal auxiliary (can cannot could couldn't)
12	NN	Noun (Rajan)
13	NNP	Proper noun (e.g., IBM)
14	NNPS	Proper noun, plural (e.g., Indians)
15	NNS	Plural noun (e.g., students)
16	PDT	Pre-determiner
17	POS	Genetive marker
18	PRP	Pronouns, personal
19	PP$	Possessive pronoun
20	RB	Adverb (e.g., very)
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	"to" as preposition
24	SYM	Symbol
25	TO	to go 'to'
26	UH	Interjection
27	VB	Verb (e.g., eat)
28	VBD	Verb past tense (e.g., ate)
29	VBG	Verb, present participle, or gerund
30	VBP	Verb, preset tense, not 3rd person singular
31	VBN	Verb, Past participle (e.g., taken)
32	VBZ	Verb 3rd pers. singular (e.g., eats)
33	WDT	wh-detemining (e.g., which)
34	WP	wh-Pronoun (e.g., who, when)
35	WP$	Possessive wh-pronoun (e.g., whose)
36	WRB	wh-Adverb (e.g., where, when)

Following are the examples of tagged sentences:

[1]Context: Relationship with adjacent and related words in a sentence, or phrase, or a paragraph.

Sentence: "Book that flight."
Tag Sequence: VB DT NN.

Sentence: "Does that flight serve lunch?"
Tag sequence: VBZ DT NN VB NN ?

We note the ambiguous word "book" in the above example, which makes it difficult to resolve the meaning of the sentence. The POS also resolves the ambiguity using a Corpus (like *Brown corpus*, or *Penn Treebank tag-set*). The disambiguation is carried out based on frequency of use of those words as well based on the context in that sentence.

Using the tag-set and corpus (tagged collection of sentences) it is possible to tag the words in a sentence and resolve the POS. The following example is a longer sentence with tags:

The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS.

13.1.2 Libraries for Tokenization, Stemming and Tagging

These tools have built-in functions to perform number of commonly used tasks, which can be directly called, or using these script can be written to perform more complex jobs of NLP and speech processing. For example, for NLP, they can tokenize the given text, can do stemming and POS (parts of speech) tagging, can find out word frequencies in given documents, parsing of NL sentences, etc. These inputs can help to compute, for example, $tf \times idf$ (term frequency * inter-document frequency), which can be helpful in IR (Information Retrieval), IE (Information Extraction), text classification, etc. In the following part, we discuss some such tools, which are either open source or they can be obtained on request from respective research laboratories.

The NLTK (Natural Language Toolkit) is a collection of Python libraries and programs for *symbolic* and *statistical natural language processing*.

The POS tagging is carried out as part of computational linguistics, using some algorithms. These algorithms associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive categories: *rule-based* and *stochastic* based. For example, *Brill's tagger*, one of the first and most widely used English POS-taggers, makes use of *rule-based algorithms*.

Consider that tasks of stemming and parts-of-speech (POS) tagging are independent, and both operate on sequences of tokens. If the stemming task is done first, the information required for POS tagging is lost. If tagging task is performed first, the stemming process must be able to skip over the tags. If these two tasks are done independent to each other, it become difficult to align the resultant texts. Hence, as the combinations of tasks increase, it becomes extremely difficult to manage the data. To address this problem, NLTK version 1.4 onward comes with a new architecture where tokens are based on Python's native dictionary datatype, such that the tokens can have an arbitrary number of *named properties*. The *Tag* and *Stem* are the examples of these properties. The NLTK allows for even whole sentence and document to be represented as single token, with *Sub-tokens* attribute that hold sequences of smaller tokens.

A *parse-tree* can also be treated as a token, which have special property/attribute of *Children*. The benefit of this type of architecture in NLTK is that, it unifies many different data types, and allows distinct tasks to be run independently. Of course, this architecture comes with an overhead for programmers, because the program need to keep track of a growing number of property names.

Example 1 *Determining Part of speech of a sentence, uisng Python NLTK Library.*

```
$ python
python 3.7.6 (default, Jan 8 2020
>>> import nltk
>>> from nltk.corpus import brown
>>> from nltk import tokenize
>>> text="This is a simple sentence, that any one can
     write."
>>> tokens=nltk.word_tokenize(text)
>>> print(tokens)
['This', 'is', 'simple', 'sentence', ',', 'that', 'any',
  'one', 'can', write', '.']
>>> tagged=nltk.pos_tag(tokens)
>>> print(tagged)
[('This','DT'), ('is', 'VBZ'), ('a', 'DT'),
('simple', 'JJ'), ('sentence', 'NN'), (',', ','),
('that', 'IN'), ('any', 'DT'), ('one', 'CD'),
('can', 'MD'), ('write', 'VB'), ('.', '.')]
>>>
```

Example 2 *Parts-of-speech tagging.*

```
>>> import nltk
>>> text=nltk.word_tokenize("Part of speech tagging and POS
                                                    tagger")
>>> text
 ['Part', 'of', 'speech', 'tagging', 'and', 'POS', 'tagger']
>>> nltk.pos_tag(text)
 [('part', 'NN'), ('of', 'IN'), ('speech', 'NN'),
 ('tagging', 'NN'), ('and', 'CC'), ('POS', 'NNP'), ('tagger',
                                                     'NN')]
```

13.2 Statistical-based POS tagging

The appeal of stochastic techniques over traditional rule-based techniques comes from the ease with which the necessary statistics can be automatically acquired and the fact that very little handcrafted knowledge need be built into the system. In contrast, the rules in *rule-based taggers* are usually difficult to construct and are typically not very robust. Stochastic taggers have obtained a high degree of accuracy without performing any syntactic analysis on the input. These stochastic part of speech taggers make use of a *Markov model* which captures lexical and contextual information. The parameters of the model can be estimated from tagged or untagged text. Once the parameters of the model are estimated, a sentence can then be automatically tagged by assigning it the tag sequence which is assigned the highest probability by the model. Performance is often enhanced with the aid of various higher level pre- and post-processing procedures or by manually tuning the model.

Creating a statistical tagger first requires a tagged corpus – a text or set of texts in which every word has been assigned its correct tag by hand. The tagged corpus is then divided into two disjoint sets of sentences, a large set used for "training" – collecting the statistics needed by the tagger-and a smaller set for "testing" – determining how well the tagger can find the correct tag sequence.

A traditional single tagger is a kind of tagger that returns the tag sequence $t_{1,n}$ maximizing $P(t_{1,n}|w_{1,n})$, where $w_{1,n}$ is a sequence of n words and $t_{1,n}$ are the corresponding n tags. To put this into words, for a sentence of length n the tagger tries to find the tag sequence $t_{1,n}$ that has the highest probability given the words of the sentence $w_{1,n}$. In fact, it finds this sequence using the standard Markov-model Viterbi algorithm.

The Second approach is a tagger that computes $P(t_i|w_{1,n})$ for each tag t_i. This differs from the earlier tagger in that the first finds a tag sequence for the entire sentence "all at once", while the second looks at each position in the sentence and computes the probability for each possible tag for that word. This kind of tagger is better if one wants to find multiple tags for a given word. For example, for the sentence, "The can will rust", the tagger computes the probability that "will" is a noun, that it is a modal, etc. Thus one knows not just the most probable part of speech, but also the second most probable, etc. We also know how great the difference is between the first choice and the second, the second and third, etc. So while the first tagger returns what it considers the best overall tag sequence, the second tagger can identify alternative tags at a position with tag probabilities close to the best.

The third kind of the statistical tagger is an "all tagger" that simply returns all tags with non-zero probability for that word. E.g., for "will" it returns "modal-verb", "noun", etc.

All the taggers share the same probabilistic model, that is, the same way of computing the probabilities of tag sequences given the words of the sentence. The model is based upon the reasonably standard *bigram tagging model*:

$$argmax_{t_{1,n}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}). \tag{13.1}$$

Here $argmax_{t_{1,n}}$ says to find the tag sequence $t_{1,n}$ that maximizes the quantity that follows. Within the product, for each tag t_i we compute the product $P(w_i|t_i)P(t_i|t_{i-1})$. The first of these terms, $P((w_i|t_i)$, is often called "word model" in that it causes the tagger to prefer tags that are common for the word in question. The second term, $P(t_i|t_{i-1})$, is called the "tag-context model" as it tends to make the tagger to prefer tags that are likely to come after the tag for the previous word.

It is the responsibility of the training phase to collect these two kinds of probabilities. However, a common problem for statistical taggers is that the set of examples found in the training data is not exhaustive, so that in the test data the tagger encounters unforeseen situations. A typical case is when the tagger encounters a word it has not previously seen. In this case $P(w_i|t_i)$, the probability of the word given the tag, is zero for all possible tags and the tagger "blows up". The solution is to "smooth" the data collected in the training phase so that these situations have not zero probability, but rather some low probability, presumably based upon some kind of auxiliary evidence. One of the resource as Corpus is *Brown Corpus*, which can be used for training.

For those words not found in the Corpus, the POS can be found by other means, e.g., if the word ends with "-ed", it is likely to be a verb, if it is "-ly" is is adverb, etc. Note that, this has been based on the last two letters of the word to be tagged. Similarly, some more empirical solutions can be the base for words not found in the Corpus.

The *Unigram Tagger* is a statistical tagger that assigns the most likely tag to the word based on the *training corpus*. To identify the most likely tag for each word, a *unigram tagger* counts the frequency of tags for each word in the training corpus. The default tag noun is used for unseen words. The unigram POS tagger is simple and fast, and it is usually used as a baseline tagger for rule-based approaches.

13.2.1 Hidden Markov Model

The HMM (Hidden Markov Model) is based on augmenting the Markov chain (see section **??**, page no. **??**). The Markov chain model tells us about probability of sequences of random variables – the *states*, each of which can take on values from some set. These sets can be symbols, or words, or tags. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. Considering the application of Markov chain for POS tagging, there for what should matter is current word (or tag) to predict the next word (or tag), and we do not need to consider all the previous words or tags.

But, this Markov chain is useful when we need to compute probabilities for a sequence of observable evens. However, many times the events we are interested are hidden and are not observable directly, e.g., the POS tags of words in a sentence. In fact, we observe the words, and then we infer the tags from word sequence. These tags are hidden because they are not observable. The HMM allows us to compute the probabilities of hidden events (tags) based on the probabilities of the observed events, i.e., the words. The HMM is covered in detail in section **??** (page no. **??**), and here we will consider only its application is determination of POS tags of words.

A Hidden Markov Model (HMM) tagger assigns POS tags by searching for the most likely tag for each word in a sentence (similar to a *unigram* tagger). Unlike with the unigram tagger, an HMM tagger detects a tag sequence for a sentence as a whole, instead of assigning a tag for each word independently. First-order and second-order HMM taggers are usually called *Bigram* and *Trigram* taggers, respectively.

Given a sentence $w_1...w_n$, an HMM-based tagger chooses a tag sequence $t_1...t_n$ that maximizes the following joint probability:

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

A HMM comprises following components:

$A \equiv [a_{ij}]$, the matrix of transition probabilities, where i, j are states,

$B \equiv [b_{i,j}]$: probabilities of observable output, i, j are states.

$\Pi \equiv [\pi_i]$: the initial probabilities of events, i is state,

V: set of output labels, $|V| = m$, and

S: the set of states, $|S| = n$.

With following additional definitions:

$$\begin{aligned} a_{i,j} &\equiv P(s_t = j|s_{t-1} = i) \\ b_{i,j}(v) &\equiv P(V_t = v|s_{t-1} = i, s_y = j) \\ \pi_i &\equiv P(S_0 = i). \end{aligned}$$

Thus, a HMM is $\lambda = (S, V, A, B, \pi)$.

For POS tagging, the A matrix is tag transition probabilities $P(t_i|t_{i-1})$, which represent the probability of occurring of tag as t_i, given the previous tag t_{i-1}). For example, in the sentence "The can will rust",

(table **??**, page no. **??**), the probability of tag t_i = "model-verb" for the word w_i="will" is much higher given that the tag t_{i-1} = "noun" (for w_{i-1} = "can"). And, in the phrase "The can", given that "the" is an article, the probability of "can" as noun is very high compared to it being model-verb or verb.

We compute the maximum likelihood estimate of this transition probability by counting, out of the times we see the first tag in a labeled Corpus, how many times the tag has appeared in the labeled Corpus, and how many times the first tag is followed by second tag, i.e.

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}, \tag{13.2}$$

where C stands for count.

The B (output or emission) probabilities, $P(w_i|t_i)$, represent the probability, given a tag t_i, what is probability that the tag is associated with the word is w_i, i.e.,

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}. \tag{13.3}$$

It can be easily understood that above equation is not the posterior probability of Bayes rule.

Since the random variables are hidden in the HMM, we need to decode the same through observations. So, given the sets A and B, defined above, and sequence of observations be $O = o_1 o_2 ... o_T$, find the most probable sequence of states $Q = q_1 q_2 ... q_T$, where $q_i \in S$. For the POS tagging, the HMM will choose that tag sequence t_1^n which is most probable given the observable sequence of n words w_1^n, i.e., it maximizes the function:

$$\hat{t}_1^n = argmax_{t_1^n} P(t_1^n|w_1^n). \tag{13.4}$$

For the computation of above, we make use of Bayes theorem, as follows:

$$\hat{t}_1^n = argmax_{t_1^n} \frac{P(w_1^n|t_1^n) * P(t_1^n)}{P(w_1^n)}. \tag{13.5}$$

Since the denominator is common in all the computations, and we are only maximizing a function's output and not truly computing the probability, hence we can drop the denominator in the above equation, with effecting

$$\hat{t}_1^n = argmax_{t_1^n} P(w_1^n|t_1^n) * P(t_1^n). \tag{13.6}$$

The HMM taggers make two more simplifications: first is probability of a word depends only on its own tag, and is independent of neighboring words and their tags, expressed by:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i). \tag{13.7}$$

The second assumption is called *bigram* assumption; as per this the probability of a tag depends only on the previous tag, rather than the entire sequence of tags:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}). \tag{13.8}$$

Substituting the results from equations 13.7 and 13.8 into the original HMM equation 13.6, we get new result for the expression for HMM as,

$$\hat{t}_1^n = argmax_{t_1^n} P(t_1^n|w_1^n) \approx argmax_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i) * P(t_i|t_{i-1}) \tag{13.9}$$

where $P(w_i|t_i)$ is called B *emission* probability and $P(t_i|t_{i-1})$ is called A *transition* probability.

The TnT Tagger (also known as a *trigram* POS tagger) uses second-order Markov models and considers triples of consecutive words to simplify the probability computation. In TnT, the tag of a word is determined by the POS tags of the two previous words.

The *Maximum Entropy* (ME) Tagger incorporates more complex features into probabilistic models. Given a sentence $w_1...w_n$, an ME-based tagger models the conditional probability of a tag sequence $t_1...t_n$ as:

$$P(t_1...t_n|w_1...w_n) \approx \prod_{i=1}^{N} P(t_i|c_i), \tag{13.10}$$

where $c_1, ...c_n$ are contexts for each word $w_1...w_n$ in the sentence. An ME-based tagger models features as binary-valued functions representing constraints to compute $P(t_i|c_i)$. It will learn the weights of the features that can maximize the entropy of the probability model using the training corpus.

The POS tagging method uses the algorithm known as HMM (Hidden Markov Model). Based on this approach we pick up the most likely tag for the given word. For example, the HMM tagger choose the tag sequence that maximizes the following formula:

$$P(word \mid tag) * P(tag \mid previous\ n\ tags) \tag{13.11}$$

Obviously, this model is n-gram model.

In the following example, we make use of above derived formulas of HMM to compute the POS tag of a word in a sentence, given the tags of rest of the words.

Example 3 *HMM based POS tagger.*

Consider the following sentence:

"Secretariat/NNP is/VBZ expected/VBN to/TO race/?? tomorrow/NN."

Here POS for the word "race" is not given. We understand that "race" can be a verb or noun. Hence, we need to find out "to/TO race/??".

In terms of probability, we want to maximize the probability out of: $P(race \mid VB) * P(VB \mid TO)$ and $P(race \mid NN) * P(NN \mid TO)$.

We apply the equation (13.11) to compute the probabilities, using the probabilities given in the standard corpus.

The lexical likelihoods from Brown corpus are,

$P(race \mid NN) = 0.00041$.
$P(race \mid VB) = 0.00003$.

While tag sequence probabilities $P(t_i|t_{i-1})$ are given as,

$P(NN \mid TO) = 0.021$, and

$P(VB \mid TO) = 0.34$.

Multiplying the *lexical likelihoods* with the *tag sequence probabilities*, we obtain the result as follows:

$P(race \mid VB) * P(VB \mid TO) = 0.00001$
$P(race \mid NN) * P(NN \mid TO) = 0.000007$

Hence, the POS of word "race" is VB.□