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

Lecture 12: Tokenization and Parts of Speech Tagging

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

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

12.1 Introduction

Part-of-speech (POS) tagging is the foundation of many natural language processing applications. Rule-based POS tagging is a well-known solution, which assigns tags to the words using a set of pre-defined rules. Many researchers favor statistical-based approaches over rule-based methods for better empirical accuracy. However, until now, the computational cost of rule-based POS tagging has made it difficult to study whether more complex rules or larger rule-sets could lead to accuracy competitive with statistical approaches.

In one approach, two hardware accelerators have been used, the Automata Processor (AP) and Field Programmable Gate Arrays (FPGA), to accelerate rule-based POS tagging by converting rules to regular expressions and exploiting the highly-parallel regular-expression-matching ability of these accelerators. The relationship between rule set size and accuracy has been studied, and it is observed that adding more rules only poses minimal overhead on the AP and FPGA. This allows a substantial increase in the number and complexity of rules, leading to accuracy improvement.

What tagging models are most appropriate as front ends for probabilistic context-free grammar parsers? In particular, we ask if using a “multiple tagger”, a tagger that returns more than one tag, improves parsing performance. The conclusion obtained is somewhat surprising: single-tag Markov-model taggers are quite adequate for the task. First of all, parsing accuracy, as measured by the correct assignment of parts of speech to words, does not increase significantly when parsers select the tags themselves. In addition, the work required to parse a sentence goes up with increasing tag ambiguity, though not as much as one might expect. Thus, for the moment, single taggers are the best taggers.

Recent years have seen a spate of research on various techniques for “tagging”- assigning a part of speech (or “tag”) to each word in a text.

As we are living in the era of big-data and mobile computing, effective and efficient natural language processing (NLP) applications become increasingly important, and they greatly affect the quality of human-computer interaction (HCI). The most efficient and high-quality NLP applications use extensive, time-consuming statistical or neural-network models, which make them infeasible for real-time applications.

A part-of-speech tagger assigns part-of-speech tags (e.g., noun, verb) to words in a sentence. POS tagging is a building block for a wide range of NLP tasks. For example, in parsing, words’ parts of speech determine proper word combinations; in named-entity resolution, it identifies the entities and the relationships between them; and in detecting sentiment contrasts, some words could have differing sentiments in different parts of speech. Moreover, in software engineering, POS tagging helps in recognizing essential words from software artifacts such as bug reports. Generally in NLP, and specifically in POS tagging, statistical and neural network (NN)-based approaches have been favored over rule-based approaches, because they have shown higher accuracy and the training is straightforward to automate, while early rule- based tagging required

Table 12.1: A sentence tagged with multiple tags

The	can	will	rust
article	modal-verb	**modal-verb**	noun
	noun	noun	**verb**
	verb	verb	

manual rule generation and execution time increased linearly with the number of rules. This limits the number of rules, thus limiting accuracy. However, rules can now be learned automatically and incorporate textual information (i.e., surrounding tags and words).

One justification for tagging research is that a tagger can serve as a front end to a parser: the tagger assigns the tags to the incoming words and thus the parser can work at the tag level, where parsers do best. This raises questions of how well different types of taggers work as front ends to parsers. Despite the abundance of work on taggers, these questions have yet to be addressed; it is still uncommon actually to read of a tagger used with a parser, and when one is so used there is no analysis of suitability.

This question becomes more important because of two strands within tagging research. While most taggers return a single best tag for each word (we call these "single taggers"). Some work has been done on taggers that return a list of possible tags in those cases where a second (or even third) best choice might be close to the best according to the tagger's metric (we call these "multiple taggers"). One obvious reason to do this would be to let the parser make the final decision.

12.2 Tokenization

The linguistic annotation of naturally occurring text can be seen as a progression of transformations of the original text, with each step abstracting away surface differences. For natural language processing, the tokenization is usually the first step of this type of transformation. To a computer, the text is just a long string of characters; the tokenization means dividing up the input text, into subunits, called *tokens*. These tokens are then fed into subsequent natural language processing tasks such as *morphological analysis*, *part-of-speech tagging*, and *parsing*. Since these subsequent treatments are usually designed to work on individual sentences, a subsidiary task of tokenization is often to identify sentence boundaries as well as token boundaries. Though rarely discussed, and quickly dismissed tokenization in an automated text processing system posses a number of questions, while only few of them have completely perfect answers.

In true sense, the tokenization is not the first step in the abstraction process. If we look at an HTML (hypertext markup language) text, original document has typesetting distinctions (e.g. font size, font style, page layout, pictures and graphics), but these are filtered out and only the text is sent as output, and from that tokens are generated.

It is not that the typeset features are not important, which can be exploited by the machine, but these variety of features make it difficult for any general processing system to take them into account. Thus, input to a tokenizer is a stream of characters which consists of graphic tokens separated by layout (after the previous step probably only space and newline characters) and possibly enhanced by markup symbols. Unfortunately, the graphic tokens, usually defined as anything between two layout symbols, need not coincide with the linguistic tokens.

The tokenization process depends strongly on the type of text which is being processed, so that an analysis of the tokenization problems in the specific type of text must be done (with ways of checking the separator/non separator status of characters).

The tokenization is associated with lower or upper level processes. Even if there exists a tendency to gather both tasks under the vague label: "pre-processing", tokenization differs nevertheless from preliminary "cleaning procedures" such as:

- removing useless tags remaining from type-setting of the text, like, HTML, MS-Word, Tex, newspaper archives, etc;
- taking away "non-textual" items such as horizontal line and page-break tags in documents, or in electronic mail smileys – :-) or :-(– and quotation representation;
- eliminating parts which do not belong to natural languages: mathematical or chemical formulae, and programs codes.

However, the borderline is thin between the cleaning task vs tokenization, as some times it becomes difficult to resolve whether it is part of tokenization or of cleaning (filtering). In addition, there is additional preprocessing needed that is associated with segmenting a word into word units, acronym (NY, POS) and abbreviation (vb) recognition, hyphenation checking, number standardization, etc. Some tokenizers include even the delimitation of textual units such as sentences, paragraphs, notes, and so on. Because of this lack of consensus on a definition, in the sequel, we should prefer to use a broader definition of tokenization, that includes various treatments which have mentioned above.

There are Tools available, e.g., NLTK (Natural Language Tool Kit), in Python language, that can perform tokenization, where a token is sentence or word. The following example demonstrates this.

Example 1 *Tokenization of natural language text.*

The following commands in Python, with NLP tool NLTK installed, demonstrates the tokenization of a given text into sentence tokens, and word tokens. Since there is only one sentence, the sentence token is one only.

```
$ python
Python 3.7.6 (default, Jan 8 2020, 19:59:22)
[GCC 7.3.0]:: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more
                                                    information.
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> text="Natural Language and Speech Processing."
>>> print(sent_tokenize(text))
    ['Natural Language and Speech Processing.']

>>> print(word_tokenize(text))
    ['Natural',  'Language',  'and', 'Speech', 'Processing', '.']
>>>
```

So far, tokenization did not constitute an issue as such: in fact pre-processing was not considered as part of NLP. There are two factors for renewed interest in tokenization techniques. First, as far as NLP is concerned, the scale has changed. On one hand, very large corpora are being gathered, either for linguistic purpose (lexicography and grammar writing) or for NLP method tuning. For example, British National Corpus, which comprises 100 millions of tagged words. The corresponding documents were to be processed in order for the tagging to be possible. On the other hand, search engines regularly scan thousands of pages on the WEB in order to build and update efficient indexes. Their *precision* and *recall* results depend on their capacities for normalizing “on the fly”. Knowledge acquisition from freely occurring texts imply selective dissemination of information as well as processing very large streams of characters (for instance, the issue the issue of the daily paper *Le Monde* represent about 20 millions of words per year). This requires the tokenization to be accurate and robust.

For the purpose of evaluation of the technique used for tokenization, the data set is divided into a *training set* and a *test set*. An evaluation methodology is associated, providing targets and measures. It works fine for domains within which there can be a reasonable agreement on what constitutes an acceptable answer for a given input. It is the case for machine-translation, as human translation provides a touch-stone, and for document retrieval, as there is a whole tradition of indexing and ranking. It is already more difficult to rely on this approach for *morpho-syntactic* tagging, as it is hard to reach an agreement on a standard. It is even worse for parsing. It should not be surprising therefore that tokenization does not seem to be amenable to this type of evaluation. Even if a working definition could be reached, the task of segmenting manually some data in order to build test data does not constitute a realistic aim.

12.3 Stemming

Example 2 *Stemming of given set of words.*

Following are the commands for stemming a set of word to reduce them to their stem words. This makes use of Porter Stemmer algorithm.

```
>>> from nltk.stem import PorterStemmer
>>> ps=PorterStemmer()
>>> words=["python", "pythoning", "pythonize", "pythonly"]
>>> for w in words: print(ps.stem(w))
...
python
python
python
pythonli
>>>
```

□

12.4 Taggers

Part-of-speech-tagging is the process of assigning parts of speech, such as noun, verb, etc., to each word. It has a wide range of applications in parsing, text-to-speech conversion, named entity resolution, machine translation, etc. POS tagging is generally categorized as a *rule-based*, *statistical-based*, or *neural network-based* model.

Most taggers are of statistical nature, which requires a tagged corpus – a text or a set of texts in which every word has been assigned its correct tag by hand. The tagged corpus is then divided two disjoint sets, a large set used for “training” – collecting the statistics needed by the tagger – and a small set for “testing” – determining how well the tagger can find the correct tag sequence.

Input to a tagging algorithm are: a string of words and a tag-set, and output is single best tag for each word (see Fig. 12.1).

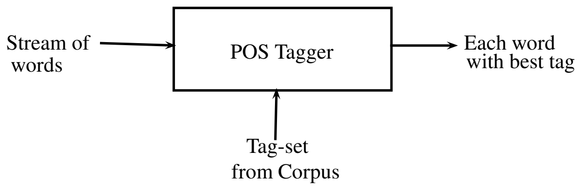

Figure 12.1: Process of Tagging

In rule-based methods, tags are assigned based on rules that embody repeatable patterns indicating a specific tag, and in statistical methods, tags are assigned based on a probability model.

The statistical taggers are basically three types:

Simple Tagger One approach for statistically based tagger is, the traditional *simple tagger*. This kind of tagger returns the tag sequence $t_{1,n}$ by maximizing $P(t_{1,n} \mid w_{1,n})$, where $w_{1,n}$ is a sequence of n words, and $t_{1,n}$ are corresponding n tags. In other 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 $w_{1,n}$. This sequence is found using *Markov-model Viterbi Algorithm*.

Multiple tags for a word Second approach of building a tagger is to compute $P(t_i \mid w_{1,n})$ for each tag. This differs from the earlier method in the sense that, the first method finds a tag sequence for the entire sentence, i.e., "all at once", where as the second method looks at each position in the sentence and computes the probability for each possible tag for that word. This tagger is better if one wants to find multiple tags for a given word. For example, given the sentence, "The can will rust", the tagger computes the probability that "will" is noun[1], that it is a modal. Thus, we not only know just most probable part of speech, but also the second most probable, etc. We shall also know how much is the difference between the first choice and the second, second and the third, and so on. So, while the first tagger returns what it considers the best over all tag sequence, the second tagger can identify alternative tags at a location with tag probabilities close to the best.

All Tagger The third approach can be one that returns all tags with non-zero probabilities for each word. For example, for "will", it may return "model-verb", "noun", etc.

All the above mentioned taggers share the same probabilistic model, that is, same way of computing the probabilities of tag sequences given the word sequences. The model is based on the reasonably standard *bigram tagging model*:

$$argmax_{t_{1,n}} \prod_{i=1}^{n} P(w_i \mid t_i) * P(t_i \mid t_{i-1}). \tag{12.1}$$

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 $P(w_i \mid t_i) * P(t_i \mid t_{i-1})$. The first of these, i.e., $P(w_i \mid t_i)$,

[1]For example, "will document" to transfer the rights of some property.

is again called *word model*, in that it causes the tagger to prefer tags that are common for the word w_i. The second term $P(t_i \mid t_{i-1})$, is called *tag-context model*, it tends to make the tagger prefer tags that are likely to come after the tag for the previous word.

It is responsibility of training phase to collect these two kinds of probabilities. However, the common problem in statistical tagger is that set of examples found in the training data is not exhaustive, due to this in the text data, the tagger encounters unforeseen situations. A typical case is, when the tagger encounters a word it has not seen earlier. In this case $P(w_i \mid t_i)$, is zero for all possible tags and the tagger "blows-up". A solution to the problem is to "smooth" the data collected in the training phase so that these situations do not carry forward zero probabilities, but may have some low probabilities, possibly, based on some kind of auxiliary evidence. An example of this is *Brown Corpus*.