ChatScript Pos-Parser Manual

Revision 11/19/2016 cs7.7

How to support non-English languages at end of manual.

Perspective on State of the Art Pos-tagging and Parsing

Back in school we learned the various parts of speech (noun, verb, adjective, adverb, preposition . . .) and were taught basic rules of grammar. Why? To help us understand meaning. But fancy grammar is not needed for basic meaning. *Me hungry* is perfectly understandable. And context also dictates meaning: *What do you like to do?* And *What food do spiders eat?* Can both be answered with *fly* with entirely different meanings.

And most chatbots, even in ChatScript, don't use pos-tagging or grammar.

They don't understand general meaning but they can hunt for specific meanings by looking for keywords in particular combinations or orders. But ultimately, you can handle more meaning when you can parse a sentence.

Pos-tagging (part-of-speech tagging) is a strange beast. Pos-taggers define a range of labels for their convenience which can be more or less than what you'd expect.

I learned something like 8 parts of speech as a child. Pos-taggers vary from 38 possible tags to over a hundred. *Existential there* is a common pos-tag, as in *there is a table*. On the more side, pos-taggers typically separate verbs into their various tenses. On the less side, the standard Penn Treebank system lumps prepositions and subordinate conjunctions into the single label IN.

Pos-tagging by itself is often useless. It exists to support parsing. Nowadays most people treat postaggers and parsers as a done deal. A long time ago a guy named Brill was discovered that you can get 90% of words correctly tagged merely by tagging it with its most common tag. Nowadays statistical pos taggers like the Stanford parser are the gold standard and on Wall Street Journal type articles it is said to be about 97.3% accurate in tagging.

Training probability pos-taggers means it generalizes what it has seen to build a relatively simple model of how probable a tag is given surrounding words. You can then test it on the same original data (something you don't normally do with machine learning models) and see how they do. TreeTagger trained on a large sample of Wall Street Journal sentences, achieves 97.5% accuracy on the replay. Trained on the switchboard corpus of telephone conversations yielded 89.7% against those. Pos-tagging degrades rapidly on data it was not trained

for. One review said that on bug reports the Stanford Pos-Tagger was between 83.6% and 90.5% accurate using Wall Street Journal trained models.

Another study showed that trained on WSJ data (lots of clean data) and applied to the same kind of data, atis: Recordings of Air Travel Information System dialogue, i.e. natural speech. 2. brown: The original Brown Corpus, containing various topics. 3. swbd: Switchboard Corpus of telephone conversations. 4. wsj/00-05: Wall Street Journal articles (collection 0 to 5). 5. wsj/00-22,24: Wall Street Journal articles (collection 0 to 22, and 24). 6. wsj/23: Wall Street Journal articles (collection 23). wsj/23: Percent Correct: 97.454%

Statistical parsers look at tuples (typically 3) of words to predict what part-of-speech a word will be. And because they do so well, pos-tagging is now separated into a separate phase from parsing. These "good" results are for kinds of material the tagger is trained for and degrades on other works until trained there. They are rarely trained on other kinds of work because the training data isn't there.

But there's a fundamental flaw with the that each word can be given a correct pos tag. Idioms are not addressed. For example: the idiom *until recently* is an adverb idiom. Yet under normal pos-taggers it becomes a preposition and an adverb. So where is the object of the preposition? But hey, at least it's one of the words is right.

More than is a particularly interesting combo. *More than one ate meat.* They label *more* an adjective and than a preposition/conjunction while they correctly label the subject as one and the noun phrase contains a quantifier phrase. What about *they sold more than 150 units.* The word than is either a preposition or a subordinate conjunction under some pos-taggers. But more than is also an idiom whose composite pos would be as a quantifier.

Quantifiers join articles and determiners as something that comes before a noun (*all children*) and denote quantity. Then mix that with being allowed to omit words. I love her more than you really means_ I love her more than [I love] you_ and *than* is a conjunction.

Or *by the time we left, we were tired.* You take a preposition determiner noun (*by the time*) but as an idiom it's a subordinate conjunction. CMU's Link Parser recognizes and marks idioms and their behavior correctly. And so does ChatScript. Stanford will collapse some multi-word prepositions into a single one during parsing, but not during pos-tagging. So you end up with strange markings. Another example is *inch by inch, they crawled.* The pos and tree for that in Stanford is rubbish, starting with noun, preposition, noun.

Back to the idea that Stanford is good doing 97.3% accurate in pos-tagging. 97.3% accuracy in postagging means that out of every 100 words, almost 3 are wrongly tagged. The average sentence length of populist publications like Reader's Digest in 1985 was 20-22 words. Heavy duty publications like the Wall Street Journal or the New York Times averaged around 26-27 words. So presumably almost one in every four WSJ sentences is wrongly tagged. Even if

you pos-tag a sentence perfectly you can wrongly parse it. When pos-tagging is wrong, what chance do you have? The stated accuracy of parsing of the Stanford parser is 56%. Almost half of what it reads is parsed incorrectly.

Google recently released SyntaxNet, an even better parser. Yet if you give it `wind the clock`, it thinks `wind` is a noun. It doesn't even have a sentence there.

ChatScript faces uncommon problems from most Natural Language tools.

The best taggers/parsers are trained on the Wall Street Journal, where everything is properly cased. If you pass *i like you* to the Stanford parser, it will tell you that *i* is a foreign word, *like* is a preposition, and *you* is a pronoun.

Chatters often never use upper case and speech recognitions devices don't output it either. So ChatScript works to handle all cases of things. And, following established tradition, I define my own tags for my own convenience. In addition to the usual tags differentiating various kinds of nouns and verb, I also differentiate words based on how it is used. Nouns can be used as adjectives, so bank teller is an adjective_noun noun_singular pair. Verbs can be used as nouns or adjectives, so we have the gerund and the noun infinitive and the adjective participle.

Postagging is harder than it looks. Consider these uses of the word *up*.

what jumps up? # adverb up, pronoun what *what jumps up the wall* # preposition up, pronoun what *what jumps up the ante?* # verb up, determiner what *what do people jump up?* # preposition up, pronoun "what" as object

Hence ChatScript has a built-in pos-tagger/parser using rules of grammar along with statistics. This is the tack taken by *Constraint Grammar* parsers, which can achieve high degrees of pos-tagging accuracy (99.x%) and run rapidly. Their primary flaw is you have to hand code all the rules of grammar (say around 1500 of them) and need a lot of grammatical expertise. And there are no open-source versions of them.

Complaints about statistical pos-parsers

So one complaint I have is with accuracy. Another complaint is speed and memory requirements. The simpler of the Stanford parsers is the PCFG one. On an Intel i5 2.53 Mhz machine, a 27 word sentence takes 1.5 seconds and 100 Mb. ChatScript takes under 10 milliseconds and 50Mb with a full dictionary. You can run with a large dictionary in 16Mb for a cellphone.

And a third complaint is that Stanford's will always return a parse, even if the parse is not reasonable. The parse of *is it on me*? given that it treats on as an adverb instead of a preposition, is garbage and is similar to *is it me*? with an extra adverb tacked on. For other sentences it can't say *I find no verb so either this isn't a sentence or something is wrong.* It has no way of knowing that maybe it failed when sometimes one clearly could if one knew grammar.

My fourth complaint, the output is a tree structure, which then needs to be reinterpreted by some other tool.

And my fifth complaint is that Stanford's errors are uniformly distributed. It doesn't matter if the sentence is a short one, a first grade sentence, whatever. It will make errors in simple sentences as well as complex ones. Particularly for chat, simple sentences are more likely, so I would prefer a pos-tagger that was more accurate on easy stuff.

And my final complaint is that you can't readily fix any Stanford's mistakes. It has to be trained on a corpus and creating one that will fix issues is hard and not ever done.

Interesting Pos-parsing examples:

The Stanford Parser (and I pick on it merely as representative of all statistical parsers of which it is considered one of the best) is 97.3% accurate in uniform ways. That is, it is flawed on all kinds of input from simple sentences to complex ones. Simple sentences can defeat it. It can handle *It is on me* but not *Is it on me*? which becomes: *is* (verb) *it* (pronoun) *on* (adverb) *me* (pronoun) making on an adverb instead of a preposition.

And it has weird variances: *Get it off* of me labels *off* as a particle (correctly) but _get the spider off of me labels *off* as an adverb (differently and wrongly). Or if you say *I saw Dad* do it it correctly labels *Dad* as a proper noun. Same for *Father* and *Mother*. But *Mom* it labels as an ordinary singular noun, even while keeping it capitalized in its output. Or consider *my dog is a blue being* which it correctly labels *blue* as an adjective and *being* as a noun. But *my dog is a big blue being* it labels *blue* as a noun and *being* as a verb phrase present participle.

It is particularly inept at distinguishing pronouns from determiners (always considering them determiners), so it will parse this sentence *I'll take all I can get* by labelling *all* as a determiner. Similarly with *All she had to do was ask*, it treats *all* as a determiner.

The main sentence should parse to all was ask with an implied that she had to do clause modifying all. And, strangely, the Stanford pos tagger likes to label *ago* as IN (prepostion or subordinate conjunction), which it never is, instead of adverb like in *they lived twenty years ago*. I can't even imagine how that error exists in the system. Even the simple *Many are poor* has *many* considered a determiner – but there is no noun. And if you say *many of them are poor* it changes *many* from a determiner to an adjective. Still no noun anywhere in sight.

It also doesn't handle simple stuff like *what swims are safe*, considering *swim* to be a verb. And it generally always thinks *can* is a modal verb, like in *what can goes pop?*.

There are also other interesting issues with quantifier phrases. *A lot of children eat.* Do you want the subject to be *lot* or *children.* If *a lot* of is a quantifier idiom, then the subject is *children.* If you are doing raw pos-tagging, then the subject is *lot.* Either will work, but I'd rather have children be the subject. Same for *Some of the honey spilled.* Normal parses have *some* is the subject. But that's underinformative. CS treats *Some of the* as an idiom determiner, so the subject becomes *honey.*

Consider this sentence: *he painted the car purple.* The Standford tagger says: *he*/PRP *painted*/VBD *the*/DT *car*/NN *purple*/NN which means it claims _purpl_e is a noun. I can live with that, though one might prefer *purple* be considered an adjective. The parse, however, becomes problematic. Should it be considered *subect verb indirectobject object* or *subject verb objectcomplement.* Or is it *subject verb directobject* (*the object being car purple*)? Its parse looks just like a parse for *he painted the bank teller.*

Proper name merging

This gets even messier given that ChatScript will attempt named entity extraction – it will decide some sequences of words represent a single upper-case compound name.

By default ChatScript attempts to detect named entities given as multiple words, and merge them into a single token with underscores. *united states of america* becomes *United_States_of_America.* This is not without its hazards. The WORLDDATA ships with a large number of names of works like movies, tv shows, books, etc. These can collide. *Don't lie to me* can see *lie to me* as a possible TV show name. The merging code will ALWAYS merge quoted strings and capitalized sequences, so *Lie to Me* will get merged. It will also merge any sequence whose result is marked with the property ALWAYS_PROPER_NAME_MERGE. Currently world data adds this on to all location names (countries, cities, states) so they are always a single token no matter what case is used. The merging code will not automatically merge the names of TV shows. Nevertheless the sequence will be marked with its appropriate concepts because the system checks sequences of words even if they are not merged. So you would get *lie to me Lie to Me* all marked appropriately and it is up to the script to distinguish.

I personally run script early on that sees if `~NOUN_TITLE_OF_WORK` matches. If it does, it determines if it was used in a sentence involving relevant verbs like *I watched lie to me yesterday* or if the user entered it capitalized or quoted or if we were already in the TV topic. If none of those are true, the script erases the `~TV` topic mark using `^unmark`. This prevents the system from erroneously trying the TV topic. You can also just turn off proper name merging by changing the value of `$TOKEN` removing the `| #DO_PROPERNAME_MERGE` value usually applied, at the cost of having to deal with named sequences yourself in script.

ChatScript Parser

The ChatScript parser runs by default on all input, doing what it can to parse it. If it believes it failed to parse correctly then it will set the tokenflags appropriately. You can test for this like this:

```
u: ( %tokenflags&#FAULTY_PARSE)
```

You can suppress the parser by setting `$token` to NOT have the request to parse `| #DO_PARSE` If the parser is not run, then the pos-tags of all words will still be partially performed, reducing the set of word interpretations as much as it can without risking losing any correct pos-tag. But you will not get any parse information..

The fragments of a successful parse are retrievable using concepts. The main sentence, because it may not contiguous and is special, does not have a composite retrieval. Instead you can retrieve `~mainsubject`, `~mainverb`, `~mainindirect`, `~maindirect`. Things which are contiguous chunks are phrases, clauses, and verbals.

ChatScript has a complexity limit, and will not accept sentences near 255 words. Also, sentences containing more than 7 of some particular concept will not mark after the first 7. So a sentence with 8 nouns will not have a ~noun after the 7th. If the nouns were a mixture of singular and plural, then it will represent them, up to the 7 limit.

ChatScript recognizes these structures: the main sentence, prepositional phrases (called phrases), subordinate clauses (called clauses) and various verbal expressions (called verbals) like gerunds, adjective participles, and noun infinitives.

Parse results are all concepts and are matched just like any other. E.g.,

```
u: (_~mainsubject * _~mainverb * _~mainobject) A 3 piece sentence in normal order.
```

See the manual ChatScript System Variables and Engine-defined Concepts for a list.

In addition, you may find ^phrase(type matchvar) useful to retrieve all of a prepositional phrase or a noun phrase. Type is `noun`, `prepositional`, `verbal`, `adjective`. Optional 3rd argument `canonical` will return the canonical phrase rather than the original phrase. E.g.,for input:

```
u: (I ~verb _~directobject) $tmp = ^phrase(noun _0)
```

with input *I love red herring* `$tmp` is set to *red herring*.

Full Sentence Information

Sentence parse data comes in 3 kinds:

1. terminal punctuation - `s:` `?:` and `\!` pattern

2. global data:

```
%tense %voice %sentence %quotation %impliedyou %foreign
%command %parsed %question %length
%tense = { past, present, future}
%voice = {active, passive}
```

3. word data- concepts including part of speech and ROLE

Testing and development

There are 2 phases that can intermix. The first are rules that try to remove obviously illegal pos values. These rules are in `LIVEDATA/ENGLISH`. I have a standard test

```
:pennmatch raw
```

which takes a bit above 90K tokens in 3800 pennbank sentences and reports those that have incorrectly pruned (`REGRESS/PENNTAGS/penn.txt`). It's goal is to maximally prune while NEVER removing a valid pos-tag. It currently prunes badly 2% of the time and prunes 47K ambiguous words down to 15.7K.

The next phase is hard code parsing. It tries to find a sentence structure that works, using a garden path algorithm. Along the way it can assign pos, rerun rules, assign roles, and sometimes override prior pos decisions when it finds a conflict. It gets run as

```
:pennmatch
```

and currently is 94% right in pos-tagging.

```
:pos this is a sentence
```

displays what it is doing as it pos/parses.

Augmented English language support

TreeTagger has a pos-tagger for English (see Foreign language support about TreeTagger) and if you have a license for it, it works in conjunction with ChatScript's own tagger to yield a better result than either separately. In the bug reports study, TreeTagger performs the best in assigning tags to nouns and verbs, while the Stanford tagger performs the best in assigning tags to adjectives and adverbs. In general I care more about nouns and verbs.

Foreign Language support

While ChatScript supports UTF8 for input and scripting, a lot of ChatScript's power comes from the concise pattern matching code. That, in turn, stems from

the ability to define concepts, and to take advantage of ChatScript's dual input stream – original word and lemma (canonical form). So instead of having to write patterns naming all forms of a verb, you can just use the lemma form and mean all of them. Or the singular noun to catch plural as well. That ability depends on pos-tagging abilities.

CS allows you to use an external pos-tagger and/or parser and/or spelling corrector for other languages. It comes with a *German* simple bot to illustrate how to do this, using TreeTagger. Treetagger is one example you can use (many others exist). TreeTagger has windows, mac, linux versions. TreeTagger supports a wide range of languages for free but you need a license if you want to use it commercially. The local non-commercial copy is annoying in that it outputs a couple of processing status messages as it goes, but it would have no impact if CS is being used as a server, and at least allows you to experiment. I presume the commercial license would enable you to remove them.

http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/ (Author: Helmut Schmid, CIS, Ludwig-Maximilians-Universität, Germany).

The German bot assumes you have installed TreeTagger and the german data per instructions provided on his website. I have only tried the Windows installation.

To use a foreign tagger, you set the `$cs_externaltag` variable in your bot definition to a topic that will perform the work. And you need to disable ChatScript from performing the work. Make your bot definition `$cs_token` NOT use the following: `#DO_SPELLCHECK`, `#DO_PARSE`, `#DO_SUBSTITUTE_SYSTEM` (since that is english substitutions and punctuation processing).

In your tagging topic, you invoke your spellchecker, or tagger, or parser, either locally via `^popen` or remotely using `^tcpopen` or `^jsonopen`. And then process the return data and use one or more of these functions to alter ChatScript's internal data.

`^setcanon`(wordindex value) – changes the canonical value for this word `^settag`(wordindex value) – changes the pos tag for the word `^setoriginal`(wordindex value) – changes the original value for this word `^setrole`(wordindex value) – changes the parse role for this word

The first important one is ^setcanon, to allow the CS dual input processing to work in patterns. If you are doing spell checking, you may use `^setoriginal` to revise the original word. SetTag and SetRole are for the pos value and Role value (mainsubject, mainverb, etc). These values will come back in whatever notation your external system uses. All of this happens BEFORE Chatscript starts marking things and therefore if you use `:trace` prepare, you can see the results of the annotations in the concepts triggered.