Knowledge-based Semantic Role Labeling

Release 0.1

September 11, 2014

Contents

This repository contains:

- A reproduction of (Swier & Stevenson, 2005) results for recent versions of resources and NLP tools (MST Parser, latest VerbNet, FrameNet and VN-FN mappings).
- Some improvements of our own
- Experiments on domain-specific knowledge-based semantic role labeling

Documentation:

CHAPTER 1

Resources parsing

1.1 VerbNet parsing

The parsing of VerbNet is done by the `verbnetreader` module. It reads every XML file in data/verbnet3-2, each of which represents a VerbNet root class and all its subclasses.

For each frame found in every class and subclasses of a file, a `VerbnetOfficialFrame` object is instanciated and associated with every member verb of the class and its subclasses.

The difficult part is to build the structure and the role list of the `VerbnetOfficialFrame`, that is, to convert the `primary` attribute of the `DESCRIPTION` element to an acceptable list of syntactic elements (that is, compatible with FrameNet annotations), and to retrieve the roles of the elements that have one.

The following operations are applied to the elements of the `primary` attribute:

- everything that is after a "." or a "-" is removed (for instance, NP.theme becomes NP, S-Quote becomes S)
- adverbs are removed from the structure (not handled yet)
- "v" is replaced by "V" (no distinction between transitive and non-transitive verbs needed)
- "a/b" is interpreted as [a or b].
- every preposition is made explicit. That means that "S_INF" is replaced with "to S" and more importantly, "PP" is replaced by "prep NP".
- every "PP S_ING" is converted to "prep S_ING" (and not "prep NP S_ING")

In order to know which preposition can be used in a PP, which is not always explicit in the primary structure, we look into the `SYNTAX` element of the frame. The information can take three different forms :

- a `LEX` element, which `value` attribute is the only acceptable preposition
- a `PREP` element with a `value` attribute, which is the only acceptable preposition
- a `PREP` element whitout `value` attribute, but which contains selective restrictions data. In this case, the `type` attribute of the SELRESTRS/SELREST element is the class of acceptable prepositions

Sometimes, keywords (prepositions or what/who/it/...) whom presence cannot be deduced from the primary structure appear in the `SYNTAX` element. That is why for every element of the primary structure, we have to find the position of the matching element of the `SYNTAX` and add any keywords that occured between this position and the previous one to our final structure if it was not present in the primary structure. Note that case it is also possible to find keywords in the primary structure and not in the `SYNTAX`, so a parsing based on the `SYNTAX` is not necessarily a better idea.

The resulting structure is the list of everything that will have an importance in the frame matching, that is NP, V, S and also every required preposition (which, as, ...). Those emplacements are represented by strings unless there are several

possibilities (for instance, when a class of preposition can introduced a NP). In that case, the element is represented by the set of every acceptable string.

We use two sources of informations to build the role list:

- the `value` attributes of the elements of `SYNTAX` where it is present
- everything that is after a "." in the primary structure. For instance, we extract "theme" in "NP.theme". We make sure that the extracted role is one of the possible role listed in the `THEMROLES` element (after some case modifications), to avoid extracting every "attribute" of "PP.attribute".

Conflicts can occur between the sources of information. For instance, in `calibrate_cos-45.6`, reading the structure of the frame `NP.attribute V NP.extent` leads to the conclusion that there are two slots with the role `Attribute` and `Extent`. In the `SYNTAX` element, another NP with a `Patient` role is present between the first NP and the verb. For those cases, as the final structure comes mainly from the primary structure, we trust the roles found in the primary element, and the second element keeps an `Extent` role.

1.2 FrameNet parsing

1.2.1 Fulltext corpus data

The parsing of FrameNet mainly consists in instanciating `FrameInstance` objects from FrameNet's XML corpus. After all steps described below, the frame occurrence is translated into a `VerbnetFrameOccurrence`, an object close enough to `VerbnetOfficialFrame` to allow for frame matching. using the content of the XML in data/fndata-1.5/fulltext/.

The FrameNet fulltext corpus in stored in XML files which contain many `sentence` elements, each of which contains one or more frames. The text of the sentence can be found in the `text` attribute, and each frame-specific data is embed in an `annotationSet` element.

Neither the predicate nor any argument is expressed as text: what we recover are the offset of the first and last characters of them (with the convention that the first character's offset is 0).

- the frame's name can be recovered in the `frameName` attribute of the `annotationSet` element
- the predicate's lemma can be recovered in the `luName` attribute of the `annotationSet` element
- the predicate's start and end can be recovered in the `layer` element which name is `Target`
- the arguments' starts, ends and correct roles can be recovered in the `layer` element which name is `FE` (Frame Elements)
- the arguments' phrase types can be recovered in the `layer` element which name is `PT`. The only reliable sign that a `label` element in the PT `layer` refers to a given argument is that it has the same `start` and `end` attribute (ie, the order can differ).

Some arguments are marked as non-instanciated. In this case, the `label` element has an `itype` attribute, which value indicates the type of null instanciation (this type is currently not taken into account). These arguments are loaded into the frame structure but never used at any further point in the program.

There are many other things that we have to be careful about:

- not every annotated frame is a verbal frame. All non-verbal frames are discarded. We used to identify verbal frames by checking that the `luName` attribute of the `annotationSet` element ends with ".v", but we realize that their are many cases were this yields strange results. We now use the part of speech annotations available in the PENN `layer`, which can be equal to "MD", "VV", "VVD", "VVG", "VVN", "VVP" or "VVZ" for verbs.

- we need to attribute a sentence ID to every extracted frame, so that we can later recover syntactic data about it in the parsed corpus. Some sentences appears twice in the fulltext corpus, and there will be only one matching sentence in the parsed corpus. So the two identical sentence have to be given the same ID.
- some annotation sets are empty or incomplete. For these, the `status` attribute of the `annotationSet` is set to `UNANN`.
- 14 argument lack a matching phrase type label. In this case, we are forced to discard the frame.
- there are sometimes multiple layers of annotations with overlapping arguments. This is not a problem as long are every layer is fully annotated, which is not always the case. In 120 cases, a secondary layer lacks phrase type data.
- 35 frames do not contain any instanciated arguments. They are discarded during the frame-matching process.
- In 4 frames, the predicate itself is an argument. In this case, it is marked as non-instanciated by the FrameNet reader module.
- there is one frame whose name is "Test35" that needs to be discarded
- there is one frame for which the predicate data are missing

Also, the 705 frames which predicate cannot be found in any VerbNet class are discarded.

1.2.2 LU corpus

The `framenetreader` module is also able to parse the Lexical Units corpus. The task is nearly the same, except that the sentences are embed into `subCorpus` elements, and that the predicate's lemma and the frame's name depend on the file and are therefore not specified in the frame data. They can be retrieved as the `name` and `frame` attribute of the file's root element.

1.2.3 Core arguments

The fulltext corpus lacks a way to distinguish core from non-core arguments. Fortunately, the frame name is given for every frame, so what we have to do is looking for this frame in the FrameNet frame index.

The list of core arguments for a frame is the set of every `name` attribute of `FE` elements which `coreType` attribute is "Core" or "Core-Unexpressed" in the frame XML file.

For efficiency reasons, the list of every frames' core arguments is computed at the beginning of the script by the framenetcoreargs module.

There are no cases of mismatch of frame names or role names between the fulltext corpus and the FrameNet frame index, except the discarded "Test35" frame.

CHAPTER 2

Frames and arguments extractions

Our aim is to be able to create a SRL program that takes raw text as input. This means that we need to find a way to extract the verbal frames and arguments from a text for which we only have the syntactic annotations of the MST parser.

2.1 Frame extraction

2.1.1 Verb extraction

As always, we are only interested in verbal frames which predicate is in VerbNet. Finding verbs is not very complex, since they are the words to which our parser attribute one of the following part-of-speech:

- *VB*: base form (as 'eat' in 'He should eat')
- *VBD*: past tense
- *VBG*: gerund
- *VBN*: past participle
- *VBP*: present tense
- *VBZ*: third personn of present
- *MD*: modal verb

Note : these tags are different from those of the fulltext annotations. The reason for this is unclear, and this only concerns verbs part-of-speech. As we are not currently using a part-of-speech tagger, our parser takes those POS annotation as input, after conversion of verbal POS into the more standard tags listed above, which are those which were used in the parser training data.

Parsing errors can already introduce some mistakes at this point, since our parser can fail at distinguishing some verbs from nouns.

We do not want to keep every of those extracted verbs. Auxiliary verbs (as "has" and "been"' in "He has been working") must be discarded. We can do this by discarding every occurence that belongs to a small list of verbs (like "have", "be" or "do"), but we might lose some intersting occurence of "have". Another solution to identify auxiliary verbs is to look for verbs which have at least one child bound to them by a *VC* (Verb Chain) relation and which are not modal verbs (because construction as "I can eat an apple" seem to result in two frames in the FrameNet fulltext annotations, so we should not discard "can" in this sentence).

We also need to determine what to do with isolated gerunds and past participles. They tend to be annotated in FrameNet, but we are not sure that they are compatible with the frame structures that exist in VerbNet.

2.1.2 Infinitive form

In order to find the VerbNet entries associated with a verb, we need to find its infinitive form. Our solution for that is to use WordNet, and more specifically the `morphy` method of `nltk.corpus.wordnet`. This is done in a separate python2 script, since *nltk* is not compatible with python3 yet.

Note: the relation between could and can is added manually since Morphy doesn't detect it. We would like to remove "can" since it's a modal not covered by VerbNet, but it also means to fire ("She canned her secretary") and to pocket, which are in VerbNet.

2.1.3 Error analysis

Some of the extracted frames are not annotated in the fulltext corpus. In most cases, this is because the annotations are not comprehensive. We analysed 50 of those frame after commit 18615 and found the following other reasons for the lack of annotation in the corpus:

```
Gérondif ou participe passé adjectival : 11
Non verbe auquel le corpus fulltext donne une POS de verbe : 8
Mauvais arbre syntaxique (=> auxiliaire non détecté comme tel) : 2
```

Because there is no simple way to know which of the extracted frames that do not appear in the corpus should be considered as errors, we chose not to penalize arguments extracted from those frames in the performances evaluation.

2.2 Arguments extraction

2.2.1 Standard method

Once we found a predicate, we can look for candidate argument nodes. This is done by exploring the subtree of the syntactic tree that starts at the predicate nodes. Our parser output is not well-fitted for applying the extraction method of Swier and Stevenson 2005 since it is dependency based and not constituency based. Our current method is to look for node in the subtree with a deprel that is either:

- SBJ or LGS for subject
- OBJ or OPRD for objects
- BNF, DTV or PRD

We also stop exploring the subtree at each node which contains a verb, because arguments that we might find under this node are much more likely to relate to this verb rather than the verb at the top node of the subtree.

This results in the extraction of very few PP: there are probably other dependency relations that should be extracted.

2.2.2 Heuristic method

Another method based on (Lang & Lapata, 2011) has been implemented. It results in the extraction of a greater proportion of the annotated arguments, but also extracts much more junk arguments.

An analysis of incorrect extracted arguments and non-extracted correct arguments has been done on 50 frames and the reason for each mistakes was noted. This was done after commit 18615

```
Extraits incorrects :

    - argument non core : 10 (8 frames)
    - argument non core perturbant le frame matching : 3 (2 frames)
```

```
    - le prédicat n'est pas un verbe : 3 (2 frame)

    A cause des règles heuristiques (total : 12 arguments)
    - la règle 3 n'extrait pas un argument (souvent avec des gérondifs): 6 (6 frames)
    - le sujet est objet d'un autre verbe et donc pas extrait : 4 (4 frames)
    - erreur d'implémentation de la règle 6 (corrigée) : 2 (2 frames)

    A cause de l'arbre syntaxique (total : 8 arguments)
    - verbe rattaché au mauvais auxiliaire : 2 (2 frames)
    - subordonnée mal rattachée : 2 (2 frames)
    - construction "help (service somebody)" -> "help (service) (somebody)" : 1 (1 frame)
    - sujet pas marqué comme SBJ : 1 (1 frame)
    - argument adjoint rattaché à un autre verbe : 1 (1 frame)
    - structure trop complexe : 1 (1 frame)

Non extraits annotés :

    - le prédicat n'est pas un verbe : 2 (2 frame)
    - prédicat isolé de ses arguments par des virgules : 2 (1 frame)

    A cause des règles heuristiques (total : 10 arguments)
    - le sujet est objet d'un autre verbe et donc pas extrait : 4 (4 frames)
    - plusieurs verbes pour un seul objet : 3 (2 frames)
    - un auxiliaire (jeté par la règle 4) se trouvait être un argument : 2 (2 frames)
    - non extraction du sujet d'un gérondif qualifiant un nom : 1 (1 frame)

     A cause de l'arbre syntaxique (total : 12 arguments)
    - objet pas rattaché au verbe : 7 (6 frames)
    - verbe rattaché au mauvais auxiliaire : 2 (2 frames)
    - sujet rattaché à un autre verbe : 1 (1 frame)
    - subordonnée rattachée à un mauvais verbe : 1 (1 frame)
    - adjoint rattaché à un autre verbe : 1 (1 frame)
```

2.2.3 POS conversions

We want to use the part-of-speeches of our arguments for the building of the frame's VerbNet structure (like "NP V that S"). The conversion from the MST output part-of-speech tags to the more limited VerbNet tags (NP, S, ADV and keywords) is based on http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html. The exhaustive list of conversions is part of the code of argguesser. Some of them, like the conversion of the IN tag which can be either attributed to a preposition or a subordinating conjunction require complex operations which are hard-coded in the *_get_phrase_type* method of this module.

2.3 Evaluation

The frames and arguments extractions should be evaluated conjointly if we do not want to have to answer questions like "Do we penalize the extraction of a frame that is not in the annotated corpus but for which we did not find any argument?"

Another question is whether we should evaluate the argument extraction process before or after the frame-matching, if we consider that arguments for which the frame-matching finds no possible roles are discarded.

Currently, we only retrieve 1142 of the 10347 arguments of the fulltext corpus. One factor for this is that there are two missing file in framenet_parsed:

- ANC__110CYL070.xml

- NTI__BWTutorial_chapter1.xml

Moreover, we retrieve 1912 arguments that do not match any argument of the fulltext corpus. According partial credit for some arguments would probably be a good idea. We need to choose how: do we trust our headword extraction system enough to say that a partial match is an extracted argument that has the same headword as an annotated argument, or do we use a longuest-common-substring approach? The first approach carries a risk of artificially boosting the results, since the headword extraction is itself based on the MST output.

CHAPTER 3

Debug ressources

3.1 Sample of headwords extractions

To get a sample of 100 headword extractions into a pickle file, use:

```
./headwordextractor.py -s 100 sample_file
```

Each sample contains three lines which are:

- the real argument
- the closest matching node of the syntactic tree of this sentence
- the headword

3.2 Dump module

The dump module is a script which facilitate the visualisation of the impact of a code modification or of an option on the results.

The –dump option allows the user to dump the state of the algorithm after the frame-matching and after the probability model instead of displaying statistics, and the dumper module can display the differences between two dump files when called directly.

By default, only the differences that have an impact on the performance statistics are displayed (ie, slots that have a good state like one_role_good in one dump and a bad state like no_role in another). To display every difference, use the `--all` option of dumper.py.

For instance, to see the exact differences between the slot and the slot-class probability models:

```
./main.py --model=slot_class --dump test1
./main.py --model=slot --dump test2
./dumper.py test1 test2
```

Outputs:

```
Differences after frame matching:
Good -> bad:
Bad -> good:


Differences after probability model:
Good -> bad:
```

```
Sentence: The stimulus package will jumpstart the economy and make sure it
does n't slide into recession again .
Predicate: slide
Argument: into recession
Frame structure: ['NP', 'V', 'into', 'NP']
Frame name: Motion
Role list: {'Location'}
Correct role: Goal (FrameNet) -> {'Location'} (VerbNet)
Status: one_role_good

Sentence: The stimulus package will jumpstart the economy and make sure it
does n't slide into recession again .
Predicate: slide
Argument: into recession
Frame structure: ['NP', 'V', 'into', 'NP']
Frame name: Motion
Role list: {'Result'}
Correct role: Goal (FrameNet) -> {'Location'} (VerbNet)
Status: one_role_bad



Sentence: The stimulus package will jumpstart the economy and make sure it
does n't slide into recession again .
Predicate: slide
Argument: into recession
Frame structure: ['NP', 'V', 'into', 'NP']
Frame name: Motion
Role list: {'Location'}
Correct role: Goal (FrameNet) -> {'Location'} (VerbNet)
Status: one_role_good

Sentence: The stimulus package will jumpstart the economy and make sure it
does n't slide into recession again .
Predicate: slide
Argument: into recession
Frame structure: ['NP', 'V', 'into', 'NP']
Frame name: Motion
Role list: {'Result'}
Correct role: Goal (FrameNet) -> {'Location'} (VerbNet)
Status: one_role_bad

(...)

Bad -> good:


Sentence: But Jamaica is not simply turning blindly into a small version of
its bigger brother .
Predicate: turn
Argument: into a small version of its bigger brother
Frame structure: ['NP', 'V', 'into', 'NP']
Frame name: Undergo_change
Role list: {'Location'}
Correct role: Final_category (FrameNet) -> {'Result'} (VerbNet)
Status: one_role_bad
```

```
Sentence: But Jamaica is not simply turning blindly into a small version of
its bigger brother .
Predicate: turn
Argument: into a small version of its bigger brother
Frame structure: ['NP', 'V', 'into', 'NP']
Frame name: Undergo_change
Role list: {'Result'}
Correct role: Final_category (FrameNet) -> {'Result'} (VerbNet)
Status: one_role_good



Sentence: Right , yeah , I heard about that on the news , yeah .
Predicate: hear
Argument: about that
Frame structure: ['NP', 'V', 'about', 'NP']
Frame name: Perception_experience
Role list: {'Theme'}
Correct role: Phenomenon (FrameNet) -> {'Stimulus'} (VerbNet)
Status: one_role_bad

Sentence: Right , yeah , I heard about that on the news , yeah .
Predicate: hear
Argument: about that
Frame structure: ['NP', 'V', 'about', 'NP']
Frame name: Perception_experience
Role list: {'Stimulus'}
Correct role: Phenomenon (FrameNet) -> {'Stimulus'} (VerbNet)
Status: one_role_good
```

CHAPTER 4

Evaluation

Evaluation is done on FrameNet and is the main reason as to why we're using FrameNet annotated text. Basically, once the algorithm has assigned VerbNet roles to syntactic chunks, we check that they are correct by mapping the manual FrameNet annotation to VerbNet annotations.

4.1 Initial mappings

4.1.1 FrameNet verbs to possible Verbnet classes

A number of FrameNet lexical units verbs don't appear in VerbNet, here is the full list:

{'there be': 111, 'should': 66, 'do': 60, 'become': 58, 'may': 36, 'carry out': 21, 'ratify': 19, 'might': 19, 'lead_(to)': 19, 'deploy': 18, 'have to': 18, 'discuss': 15, 'arrest': 14, 'must': 14, 'cite': 13, 'set up': 8, 'belong': 8, 'have got': 8, 'reprocess': 7, 'decide': 7, 'born': 7, 'violate': 7, 'research': 7, 'utilise': 7, 'let': 6, 'refuse': 6, 'weaponize': 5, 'go on': 5, 'undergo': 5, 'lack': 4, 'make sure': 4, 'sentence': 4, 'end up': 4, 'forecast': 4, 'result_(in)': 4, 'invade': 4, 'process': 4, 'bring about': 4, 'vow': 4, 'renew': 4, 'defeat': 4, 'overthrow': 3, 'knock down': 3, 'talk_(to)': 3, 'drag on': 3, 'complement': 3, 'modify': 3, 'rape': 3, 'deserve': 3, 'come to a end': 2, 'comply': 2, 'tend': 2, 'thwart': 2, 'progress': 2, 'locate': 2, 'flight-test': 2, 'bring_together': 2, 'abide_((by))': 2, 'account': 2, 'defect': 2, 'restart': 2, 'contravene': 2, 'disable': 2, 'hold off': 2, 'resemble': 2, 'backstitch': 2, 'perpetrate': 2, 'bomb': 1, 'forget': 1, 'reinvigorate': 1, 'step up': 1, 'be_supposed_(to)': 1, 'cut short': 1, 'shield': 1, 'piece together': 1, 'get up': 1, 'wreak': 1, 'rebuff': 1, 'inhabit': 1, 'opt': 1, 'advocate': 1, 'birth': 1, 'quiet_down': 1, 'embark': 1, 'govern': 1, 'point (to)': 1, 'spit up': 1, 'contradict': 1, 'give_out': 1, 'propel': 1, 'sit down': 1, 'reinforce': 1, 'screw up': 1, 'make up': 1, 'plead': 1, 'best': 1, 'educate': 1, 'emplace': 1, 'diddle': 1, 'besiege': 1, 'postpone': 1, 'dry up': 1, 'redirect': 1, 'fire off': 1, 'obey': 1, 'carry on': 1, 'distinguish': 1, 'substantiate': 1, 'set fire to': 1, 'equal': 1, 'ought to': 1, 'evoke': 1, 'smoke': 1, 'go away': 1, 'break out': 1, 'put an end to': 1, 'mouth': 1, 'authorize': 1, 'set off': 1, 'give rise': 1, 'count (on)': 1, 'disobey': 1, 'memorise': 1, 'bolster': 1, 'put together': 1, 'come together': 1, 'subvert': 1, 'sit up': 1, 'master': 1, 'meet with': 1, 'set out': 1, 'flight test': 1, 'descend_(on)': 1, 'reign': 1, 'ford': 1, 'lunge': 1, 'go_((through))': 1, 'sum up': 1, 'overrun': 1, 'word': 1, 'look back': 1, 'back out': 1, 'turn out': 1, 'endanger': 1, 'take to the air': 1, 'jeopardise': 1, 'usher in': 1, 'come up': 1, 'attract': 1, 'proscribe': 1, 'jot down': 1, 'refute': 1, 'rout': 1, 'revisit': 1, 'ride_out': 1, 'answer': 1, 'lock up': 1, 'farm': 1, 'absorb': 1, 'await': 1, 'cater': 1, 'get ready': 1, 'come_(across)': 1, 'work together': 1, 'stymie': 1, 'remedy': 1, 'contact': 1, 'reshape': 1}

Observations:

- We're happy that auxiliaries such as may, do, and should are present, since we don't want to deal with them.
- This stems from one of VerbNet's most important design decision: only include frequent English verbs, which leads to a very useful lexicon for NLP. This explain why're we're missing out on a few long-tail verbs.
- Indeed, 50% of occurrences come from 10% of verbs

- 30% of verbs are particle verbs (that's not too much).
- What's the difference between point (to) and abide ((by))?

4.1.2 FrameNet-VerbNet mapping

This is done by using an existing VN-FN mapping provided by VerbNet. This mapping is a possible source of errors but looks quite robust.

4.2 Frame-matching

Two metrics are of interest here: how precise our matchings are, and what matching we missed.

4.3 Probability model

TODO: Include the bias-variance analysis and link to it from "Ideas".

4.4 Bootstrapping

We still need to know how concerned we are with this, since it doesn't appear in 2005.

CHAPTER 5

Ideas

Improvements can happen in three main directions:

- find an actual possible class when there is not possible match,
- be more selective when choosing potential VerbNet classes,
- or implement a more informative probability model.

The most important improvement is the first one - it also means we have to frame our problem in terms of coverage, which the litterature didn't do yet.

5.1 Finding new matches

An error analysis is needed to understand where the errors come from: is it because the phrase types from VerbNet are too restrictive? Or is the reason clausal subjects? Clausal objects that are not handled correctly in VerbNet due to Levin legacy (Levin classes didn't handle clauses, VerbNet has put them at the end, but they're not well integrated) ?

The first result is that only 43% of FrameNet frames are not in the mapping, which is a VerbNet -> FrameNet mapping, not a FrameNet -> Verbnet one. We won't be able to perform matching on any of these frames.

The second one is that in our fulltext test corpus, 11% of frame occurrences don't have a predicate in VerbNet. Out of those occurrences, 99.3% have at least one associated Verbnet class that is present in our VerbNet->FrameNet mapping. We're only interested in evaluating those 5454 frame occurrences.

Frame matching (revision 18420) fails for 4069 arguments, finds one match for 6180 arguments, and finds more than one match for 2158 arguments. We're interested in those 4069 failing matchs: what's going on?

Concerning arguments that weren't matched, the main reason is non-core FrameNet roles. For the sentence, "In march, John bought a dress", our algorithm will try to match "in NP NP V NP", whereas we really want "NP V NP". The algorithm now ignores non-core FrameNet roles: only 2088 arguments are not matched. (It's possible but less likely that non-core arguments in FrameNet will be expressed in VerbNet, inducing an error.)

Other errors include:

- passive voice is never in VerbNet
- FrameNet annotates twice subjects introduced by "that", "who", and so on: one for the preposition, and the other one for the higher subject (splitted args for other reasons can cause problems too).
- missing VerbNet constructions (to believe in something), or missing VerbNet phrase types (NP vs. S)
- control/raising verbs can cause problems too.

Parsing VerbNet or FrameNet is sometimes an issue for complicated cases: we will be handling more and more of them over the time. “He said that S” comes to mind: currently, it’s seen as “NP V S”, whereas it’s seen from Verbnet as “NP V that S”.

5.2 Restricting potential VerbNet classes

5.2.1 Word Sense Disambiguation

As (Brown et al, 2011) have shown in their “VerbNet Class Assignment as a WSD task” paper, it’s possible to have 88.7% accuracy when disambiguating VerbNet classes for a specific verb (most frequent sense baseline of 73.8%). Would this be a good way to prune out unwanted classes?

One issue is that existing approaches use supervised learning.

5.2.2 Selectional restriction detection

VerbNet places selectional restrictions on thematic roles, but there’s no direct way to know if our FrameNet arguments respect those restrictions. Detecting it automatically would be a good way to prune out unwanted arguments.

5.3 Informative probability model

Three simple models are used in (Swier and Stevenson, 2005) - so simple that our analysis show that two of them suffer from high bias. After 10% of the corpus is seen, they’re no longer informative. The third one is more informative but doesn’t improve with more data (TODO include analysis from Guilhem).

One option is to continue in the direction of probability models: try to find more informative ones by including more linguistically-motivated data.

Another option is to use a probabilistic model which would allow us to modelize correctly the back-off procedure from (Swier and Stevenson, 2004).

The last option is to fully go the supervised route: use features and models from supervised litterature using our initial frame-matching data.

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*

CHAPTER 7

References

Robert Swier and Suzanne Stevenson. Exploiting a Verb Lexicon in Automatic Semantic Role Labelling. In Proceedings of HLT/EMNLP, 2005.

VerbNet Annotation Guidelines http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

VerbNet documentation http://verbs.colorado.edu/verb-index/Documentation.php

VerbNet presentation http://verbs.colorado.edu/~mpalmer/projects/verbnet.html