Computational First Position Repair Processing

Charles Threlkeld and Alexandru Ungureanu

3 April 2018

1 Problem Statement

In natual dialog, people often correct themselves mid-statement. For example, if I say "I'm going to the library, no Halligan," a person will understand that I am going to Halligan. However, machines will often fail to process this utterance correctly. "I'm going to the library," is a full sentence, and the computer may recognize it as such, disregarding the repair. Alternatively, the machine may simply fail to parse the utterance and ask for a repetition. For a more exhaustive survey of the current research on linguistic repair, see Albert and de Ruiter[1]. Our algorithm aims to clarify the semantic meaning of substitution self-repair. It will dispose of any super-semantic meaning of the processed disfluencies.

2 Approach

Our working hypothesis is that substitution phrases take the same part-of-speech structure as the phrases they are intended to substitute. So, in the example above, "library" is substituted with "Halligan," both of which are nouns. In this case, a naive implementation of our algorithm fails on strict syntactic substitution, but will succeed on a semantic level. That is, "I'm going to the Halligan" is not grammatically correct, but when semantically represented in a robotic architecture (or other semantic processing), the semantics are sufficiently correct.

So, our approach is to use a part-of-speech tagger on a sentence (or utterance) level. Then, we will flag words or phrases that signal self-repair ("I mean", "actually", "uh"). Some of these words may also be used to hold the turn in dialog, rather than for repair, so our algorithm assumes that a repair is being made. If none of the signal words are found, it will try to find similar part-of-speech structure for two parts of the sentence and discard the middle part of the utterance.

[1] Repair: the interface between interaction and cognition. Albert and de Ruiter 2018

3 Evaluation

In order to evaluate our algorithm, we will verify how similar the test output is to "human" syntax. Specifically, we will read the output and judge on the semantic quality of the repaired sentences after repairing the "input" sentences ourselves, then statistically analyze the performance of our algorithm and check to avoid overfitting. Ideally, we would want to automatize the evaluation phase, but that would require access to some third-party semantics quality assurance software.

4 Materials

4.1 Data List

We anticipate needing at least a few hundred sentences of dialog that has been recognized as exhibiting first-person repair. The larger our corpus, the more general we can make our algorithm, but since this type of repair happens almost exclusively in spoken dialog, and not prepared speech or written text, we will have to find a corpus of transcribed natural dialog.

We have found the Cooperative Remove Search Task (CReST) corpus that was put together by Kathleen Eberhard et al. in 2010. It codes for disfluencies and should serve our needs. It has already been tagged for part-of-speech, so we can use it for both training and testing of our algorithm. Though the current implementation is deterministic and the scope does not include identification of repair, if more work can be done, then it may be possible to do a more standard train and test. Details can be found in Eberhard 2010[2]

4.2 Code Libraries

We anticipate to use breeze linear algebra and optimization libraries and possibly some other statistics scala libraries.

4.3 Additional Material

We have located several papers by psycholinguists studying the mechanisms of dialogue repair as well as the current state of robotic implementation of these mechanisms.

5 Results

At this point, we designed and identified the different components of our project: the Sentenceizer (splits an input file into the sentences to be repaired), the Tokenizer (splits a sentence into tokens), the POSTagger (tags

[2]Eberhard et al. The Indiana "Cooperative Remote Search Task" (CReST) Corpus 20017

each of the tokens of a sentence with a part of speech found most appropriate by our POS tagging algorithm), and the Repair driver (based on the POS taggings of the tokens and the flagged repair phrases, repairs the phrase and outputs the resulting sentence).

In addition to setting up the scala project and the source file for each component, we implemented a naive training for the POS Tagging component of our project and created the skeleton of the POSTagger class, which will take a list of tokens, and, based on the training phase learning, assign the "appropriate" part of speech tag to each token. Also, we wrote a simplifier that maps Penn tags to the more general tags Adjective, Adverb, Conjunction, Determiner, Noun, Number, Preposition, Verb, Beginning.