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

Out approach uses the Indiana "Cooperative Remote Search Task" (CReST) Corpus[2] that was compiled from 2007 to 2010 by Kathleen Eberhard et al. at University of Notre Dame from 2007 to 2010. Eberhard et. al. collected and codified a pair of teammates performing a search task. The corpus was compiled explicitly as a data source for commonly problematic aspects of natural human dialogue in computational linguistics. The teammates are linked only by audio, and they were each privy to only certain relevant information for the task. This ensured that the scenario would elicit certain forms of problematic dialogue that are not necessarily present when participants have a larger shared knowledge base.

This corpus is broken down into several levels: utterances, token, parts of speech, and disfluencies (as well as a few others that we are ignoring for our purposes). the parts of speech and disfluencies are marked at a token level. To form a training and testing set, we processed the corpus into a data structure that combines these different tagging purposes into a single data structure so that we can tie together the utterance to its tokens, parts of speech, and

[1] Repair: the interface between interaction and cognition. Albert and de Ruiter 2018
[2] The Indiana "Cooperative Remote Search Task" (CReST) Corpus. Eberhard et al. 2010

disfluencies. The tokens, parts of speech, and disfluencies are lists of the same size and ordering, so that they can be easily associated computationally.

Once we were able to parse the data in this way, we isolated utterances that were tagged as disfluent and at least two tokens long. The assumptions here are: 1. Disfluent utterances signal that a repair is happening. If an utterance were fluent, then the speaker has already said what he intends, and may change the meaning later, but the utterance itself does not need repair. 2. If an utterance is only a single word, there is no semantics to repair. A simple "uh" utterance, for example, is disfluent, but cannot be repaired into a fully realized text. There is still information present, but it is meta-textual. For example, it could signal confusion or distraction, but tracking this sort of semantic information is beyond the scope of text repair.

The corpus yielded about one hundred repair candidate utterances. Reading through these, we narrowed this to 55 repairable utterances. Candidates were thrown out for several reasons. Some where textually fluent, but were marked with disfluencies by the corpus given the audio log. Some were more than one token, but only a single word (e.g. huh?). The remaining fifty-five should form a good, natural corpus with which to operate. Our sample size is a bit small, but should be large enough to be relevant on the scale normally used within computational linguistics. For compairson, van Deemter[3] outlines three experiments where there are 25, 14, and 34 subjects are used for the human control set. Our data set is of this same order, and so fits the standard set by other research.

In addition to our work preparing the repairable dialog, we have also built the scaffolding necessary to leverage our algorithm on novel data. The Sentencizer, Tokenizer, and POSTagger modules work together to create a data set that useable by our repair module. The disfluency data provided by the corpus was used to generate our training and test sets, but will not be necessary in order to leverage our repair mechanisms. Having disfluencies or repair tagged for a certain utterance is outside the scope of our work. If our algorithm is given text that does not need repair, it may become damaged as the features of the repair mechanism may flag false positives.

Of note in the POSTagger is that we use a different corpus to the CReST corpus. The reasons for this are twofold: 1. The data provided by the CReST corpus are limited to the words used in the specific task present in that experimentally generated data, and may not generalize easily to other domains. 2. The sheer number of unique tokens present in the CReST corpus is significantly smaller than that provided by our POS corpus. The wider data set allows for more robust, reliable tagging, which will be leveraged in the repair module.

[3]Finetuning NLG through experiments with human subjects: the case of vague descriptions. Kees van Deemter. 2004

3 Remaining Work

The CReST corpus has been scraped for the training and testing data, but we still need to parse these into a useable format. First, we need to code each utterance into a syntactically repaired version. E.g. "I like walked down the hall" maps to "I walked down the hall". Once this is done for the entire set, we will randomly choose a subset for our training set. 30-40 utterances should be sufficient for training, with the remaining 15-25 serving as the testing set at the end of the study.

Once we have training data, we can begin work on the repair features. We anticipate there will be several features such as "collapse repeated words" and "erase simple disfluencies such as 'um"'. The training set will help guide us as we add features to the algorithm. Once we are satisfied with the score of the testing set, we can then, only once, test our algorithm on the testing set. Given that our data is especially precious, we cannot allow our training and testing sets to intermingle or we will reduce the generalizability of our project. In fact, since we are coding the repaired sentences by hand, we may be reducing generalizability already. However, given our limited resources, this is a compromise that we will make.

In the future, it would be ideal to obtain a larger dataset that is then hand-repaired by several, independent humans, in a way blind to the feature creators. Then, when split into training and test data, we could ensure that the problematic nature of our current system does not enter into the decisions made in creating the repair features.

4 Evaluation

When judging referring expressions in computational linguistics, one common method is the Dice metric defined as follows:

$$s(A, B) = \frac{2 * \|A \cap B\|}{\|A\| + \|B\|}$$

That is, for two expressions, we measure their similarity by counting the tokens they share and dividing by the total number of tokens. A perfect score of 1 would mean that the intersecting tokens of the two sets are equal in size to the addition of the two sets (i.e. they are the same set). There are some concerns with the Dice metric[4] For example, a single missing token is penalized more heavily in the linear judgement than is adding an extra token. Secondly, the Dice metric is symmetric, so $s(A, B) = s(B, A)$, which has similar problems. Further, the semantic content of the repaired sentence (produced freely by a human) could rephrase in such a way that the tokens have almost no overlap, but the semantics are extremely similar.

[4]Beyond DICE: measuring the quality of a referring expression. Kees van Deemter. 2010.

However, given these caveats, Dice is still a reliable method for a first pass on the effectiveness of our algorithm. It has the advantages of being easily implemented and objective. Were we to judge on semantics, we would need an entire NLU pipeline to format both utterances into (e.g.) a logical format, and then a method to compare the similarity of the logics. Or, we could explore the similarities subjectively, but any subjective judgement is going to be obscured by the biases of any judge. (E.g. the creators have incentive to judge leniently, but classmates may have incentive to judge harshly.)

5 Materials

5.1 Data List

We anticipate needing at least a few dozen 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 mechanized train and test. Details can be found in Eberhard 2010[5].

5.2 Code Libraries

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

5.3 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.

6 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 each of the tokens of a sentence with a part of speech found most appropriate

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

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 a naive implementation 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. Currently, the tagger works at about 95% accuracy according to the testing data. It currently operates on the corpus from assignment 6, so results may be somewhat smaller when operating on the distinct CReST corpus.