|  |
| --- |
| **Paraphrasing using Active and Passive Sentence Conversion** |
|  |
| **Christian Gould Edward Jiang Yash Sinha** |
| Southern Methodist University Southern Methodist University Southern Methodist University |
| [cagould@smu.edu](mailto:cagould@smu.edu) [ejiang@smu.edu](mailto:ejiang@smu.edu) [ysinha@smu.edu](mailto:ysinha@smu.edu) |
|  |
|  |

Introduction

Our group decided to do the paraphrasing topic. With the paraphrasing topic, our goal was to research the capabilities of using NLP to rewrite sentences and keep their same meaning. This method allows NLP specialists to summarize sentences or paraphrase them in a more understandable way. In our research, we found a single topic to be very daunting, and we chose to focus on it instead of pursuing the overall topic. That single topic is the process of turning passive sentences to active, and vice-versa. It can be argued that this is still paraphrasing in a sense, so in the following paper, we will discuss how we perform active and passive voice conversion to paraphrase sentences.

Implementation

POS Parsing

Our program uses Python 3.10 with NLTK and spaCy libraries. We first parse a sentence by spaCy’s dependency parser and then form it to a NLTK tree. The dependency tree will label the relationship between words and also their part of speech (POS). With this information, our program can start by identifying whether a sentence is active or passive. The identification works as follows: the program will look at the root verb of the tree and check its POS. If it is an auxpass, then it is a passive sentence. If not, then the sentence is an active sentence. This approach gives us 100% accuracy in all active sentences test cases and about 90% accuracy in all passive sentences test cases we used. The reason for this 10% gap in recognition of passive sentences is because spaCy dependency parser does not recognize abbreviations as verbs. For example, in “Henry’s drunk”, the dependency parser cannot recognize the “‘s” as the root verb, which leads to it thinking that the sentence is active when it is actually passive.

Once we identify that it is an active sentence, we will convert it to a passive sentence. In order to have a passive counterpart, the active sentence must have a direct object. Without a direct object, an active sentence cannot be converted to passive. For example, “I quit” is an active sentence, but since it does not have a direct object, the program cannot convert it to a passive.

Our program identifies direct objects by looking at the “dobj” part of speech in the dependency parsers. A sentence can have multiple direct objects and our program is also capable of identifying all of them. Then, the program will look for any indirect objects. According to source 1, an indirect object must appear after the verb and before the first direct object. By using this rule, our program can pinpoint the phrase of the indirect object. In some cases, an object of preposition may appear instead of indirect objects. In this case, the program will look for the “pobj” part of speech to locate prepositional phrases. The main difference between prepositional phrases and indirect objects is that preposition phrases occur after the direct object while indirect objects occur before the direct object, even though they both receive the action from the direct object. Lastly, the subject is identified by any words before the root verb.

Active Sentences

Once the program finds all necessary POS, it then will use tree paths to get all modifiers before the direct objects and object of prepositions. The program performs a depth-first search on all nodes in the parse tree to look for direct object words or objects of preposition words. With this information, the program will start converting from active voice to passive voice. Active sentences generally fall into two categories:   
[subject] + [verb] + [indirect object] + [direct object] or [subject] + [verb] + [direct object] + [object of preposition], where the indirect object and object of preposition are optional. The program will then convert it to pass by the following formula: [direct object] + [be] + [verb] + [to] + [indirect object/object of preposition] + [by] + [subject]. With this formula, the program produces accurate results for converting active to passive sentences.

Passive Sentences

As for forming passive sentences, it works in generally the opposite manner. When we parse the sentence, the first thing we look for is an indirect object. In order to properly convert the sentence, we need an indirect object. For example, the sentence: “The test was taken” has no indirect object, and so cannot be converted to an active sentence.

The way in which we find the indirect object is the hardest part of our passive to active conversion. The first step is simple. We first search through the spacy dependency tree for any words that have agency. If there is a word with agency, we know that we have found our indirect object, and consequently have found our subject for the active sentence. If there is not an agent, then we assume that the sentence is not translatable to active since we need a subject in order to convert.

Outputs

In terms of accuracy, we performed tests on 50 passive sentences, and 39 of them were correctly converted. The sentences that were incorrectly converted were vastly more complicated, or simply could not be converted. One of these complex sentences was: “The very difficult test was taken by Curious Christian and Happy Yash last week.", which yielded: “Yash very takes the test”. This error occurred because an agent was not found properly. Very clearly to us as English speakers, there is a compound agent “Curious Christian and Happy Yash''. But, because our system is only rule-based, we have not gotten to the point where we are adding specific-enough rules to capture edge cases such as this. Because of this, whenever we attempt to use more complicated sentences, our algorithm breaks down, especially with multiple subjects and objects. For simpler sentences, it works very effectively.

Results

Dependency Trees

In the following case, advice is identified as a direct object. The program will look for this word in the tree and find all its children and connect them, which will result in “very good advice”. This is how the program deals with adjective modifiers, by using the tree path to find all related words. We then rearrange these words to form a passive sentence out of an active sentence.

Text

Description automatically generated

Figure 1: Active Sentence Dependency Tree

The output of this test case is: “Very good advice is gave to her generally.” In this case, we transform the voice successfully from active to passive with a minor grammar mistake.

Timeline

Description automatically generated with medium confidence

Figure 2: Passive Sentence Dependency Tree

And the resulting sentence becomes: “Christian takes the test”. A correct conversion! This is easy for our parser, because when it searches for the agent, it finds the word “by” to have the function “agent”, and then searches down the tree for the children of that word, which in this case is “Christian”.

1. Conclusion

In this program, we used a dependency parser to determine each word’s part of speech and use the nltk tree path to determine all modifiers that associate with the word. We then used grammar rules to identify parts of speech, and their purposes in those sentences, ultimately rearranging those words in order to change the sentence’s voice.

There are several limitations in the program. First, the program uses spaCy’s dependency parser and it will not recognize direct objects all the time. If spaCy fails, then the program will not produce correct outputs either. Additionally, the program does not have verb conjugation in it, aside from a couple hard-coded ones. Therefore, the output sentence might be grammatically incorrect vocabulary-wise, while still having the correct word-order. For example, the sentence “She generally gave herself very good advice” becomes “Very good advice is gave to herself by her generally. Finally, the program incorporates many hardcoded rules like the subject in active sentences is assumed to be in front of the root verb, which may not always be the case. The largest fallback of our implementation, however, is that it does not work well with complex sentences as it has difficulty handling multiple different parts of speeches and multiple objects. These are the limitations that could be mitigated with the implementation of a neural network, which would be the way to take this program to the next level, since we have learned that rules can only be so effective, especially with complex sentences. In the future, we plan on implementing a neural network to deal with these complex grammar rules to increase the paraphrasing program’s accuracy.