Research

James Won

December 2020

1 Introduction

In this report we will look into the pattern recognition of definitions in math textbooks. Specifically we are looking at the definitions that are not explicitly stated but are said inside a sentence. The tools used were SpaCy, Python, and xml.etree.ElementTree. The dataset used was from the NIST Digital Library of Math Functions. This dataset consisted of xml files that held short sentences of math definitions. At a high level, xml.etree.ElementTree was used to parse these xml files which sent the sentences to SpaCy to apply pattern recognition to the sentence and find the subject and the definition of the sentence.

2 Parsing

The dataset consisted of 36 chapters, each chapter had multiple subsections. Each subsection was it's own xml file. The files consisted mainly of sentence tags and math tags. Math tags are used to wrap mathematical functions. Initially parsing was done through the sentence tag, but this did not account for the math tags inside the sentence tag. Thus in the sentence "for loop", I nested a "for loop" for the math tag as well. This allowed me to correctly retrieve the full sentence. In the nested for loop I also take the math functions and add them to an array. This is used later to compare tokens of the sentence to find the definition and subject.

3 Pattern Detection

The simplest and most obvious definition to be found in a sentence is characterized with is defined by . Using SpaCy pattern and matcher functions, I created a simple method to find sentences that contain the substring is defined by . Once the matcher finds the pattern it returns the index of where the pattern begins and ends. Originally to get the subject and definition I just grabbed the next token and the previous token but this does not work for math functions for two reasons. Firstly, the math function can have a space in the function which will split the into multiple tokens. Secondly, SpaCy tokenizes differently than

the built in python split() method. For example the string 'f(t)' in the normal split() function will be one token but in SpaCy it will be split into 'f(t' and ')'. This is where the array of math functions comes in. I compare the previous and next tokens to the items in the array. If it is a substring of an array item, I concatenate the next token until it equals the full math string.

Initially using this pattern I was able to get 15/16 sentences in the file 1.tex.xml. The issue with this first pattern is that the subject would not always be the token right before the "is defined by" phrase. For example, given the sentence, "The Stieltjes transform of a real-valued function f(t) is defined by $\mathcal{S}(f)(s) = \mathcal{S}f\ (s) = \int_0^\infty \frac{f(t)}{s+t}\mathrm{d}t$.". The normal pattern matcher would find 'f(t)' as the subject but the actual subject for this sentence is "Stieltjes transform of a real-valued function f(t)". 14 of the 18 sentence in the 1.tex.xml file were of this pattern. The new complex algorithm accounts for this and finds 18 sentences. It correctly gets the subject of 17/18 sentences. The one sentence that is missed comes from sentence 47 which is nuanced and difficult to detect a pattern.

The second definition pattern I looked at was, if ... then... . This is another clear example of a definition because there is a conditional statement after the "if" and the definition that is implied after the "then". At first I used regex to find the pattern and this gave me 20/25 definitions in 1.14.xml. I then switched the pattern recognition to ["LEMMA": "if", "OP": '*', "LEMMA": "then"] which produced much better results. Using this pattern, I was able to gain 25/25 sentences that contain the if... then... statement. To grab the subject and the definition I simply take the tokens between the "if" and "then" for the subject, this is because everything between those two words are a part of the conditional "if" statement. The definition is the remaining part of the sentence after "then". This algorithm was able to pick up correctly the 25 subjects and definitions. It would be interesting to see this on a higher scale with several hundred sentences.

Other definitions I tested were, then... where... , suppose... then... , let... be... . The first pattern, then... where... can be found 22 times in the 1.tex.xml file. The definitions are correctly found using the same type of algorithm as if... then... , but it can be seen that the then...where... definition is often the second part of the definition of the full sentence. It usually follows if... then... where... . "Where" is used as a describing subsection of the definition in the "then" clause. 15/22 sentences were of this nature. Thus it is better to use this phrase in conjunction with the if... then... statement to get the best results. The word 'is' can be used often for definitions but by itself it is not a good representation of a pattern. This is why I use 'Suppose' to filter sentences. Similar to if... then... I take the tokens in between the two words as the subject and the rest of the sentence after 'then' as the definition. This works for the most part for example: Suppose f(t) is continuously differentiable on $(-\infty, \infty)$ and vanishes outside a bounded interval. Subject = 'f(t)' and definition = 'continuously differentiable on $(-\infty, \infty)$ and vanishes outside a bounded interval'. Here we can see that the entire definition is found. But there are sentences like: Suppose f(t) is a real- or complex-valued function and

s is a real or complex parameter. Here we would want two different definitions, one for 'f(t)' and one for 's' but the pattern will parse it like: subject = 'f(t)' and definition = 'a real- or complex-valued function and s is a real or complex parameter'. Clearly the split should happen at the 'and' but this cannot be used in the pattern because the previous sentence shows that 'and' can still be used for the same definition. In 1.tex.xml, I found 28/29 sentences with 5 of the 28 sentences incorrectly guessing the definition. Here the research can benefit from part of speech parsing which will be talked about in more detail later in the report. Lastly, I looked at let... be... . The word "let" can be seen as a defining word but is too generic by itself. After looking at the sentences I noticed that many sentences had the pattern let...be... or let... where... . Using this pattern recognition, I was able to find 23/23 sentences in 1.tex.xml. And this pattern incorrectly defined 4 out of the 23 sentences.

4 Conclusion

Future work in this research project can benefit from more definition phrases and using SpaCy's part of speech recognition. This semester, I looked at defintion patterns is defined by , if... then... , let... be... , suppose... then... , and then... where... . Expanding this list to contain more phrases will provide more definitions. Further research can also benefit from part of speech tagging. This will make the biggest impact in complex sentences. By changing the math functions into something SpaCy could understand would allow us to use the built in function to find the subject of the sentence. Changing the math function to the word "MATH" may work as "MATH" is a noun and can be the subject of a sentence. Then from here we can distinguish which part of the sentence is correlated to the subject and get the definition for the correct subject. The research conducted this semester is a strong start for future additions that can branch from the pattern recognition to more complex NLP part of speech tagging. It would also be interesting to see a more robust parser that could include paragraphs, due to the nature that definitions may not live in just one sentence and could be spread across an entire paragraph. Conducting more tests on different parts of the dataset can also provide a clearer picture of accuracy of the pattern recognition.