Brand Name Extraction

Project Stage 2

Ainura Ainabekova, Nick Schober, Sunny Shah

3/10/2016
UW-Madison CS 784

INTRODUCTION

For this project, our group initially worked on separate extractors simultaneously in an effort to develop ideas that could be shared to build an even stronger extractor. Overall, we got solid results with a dictionary-based approach without utilizing ML or other methods. We ended up with two extractors with slightly different algorithmic implementation, but with similar results. In this report, we'll cover the results of the extractor which performed slightly better, a description of its implementation, and an analysis of how it could be improved.

ACCURACY - PRECISION AND RECALL

Here are our results on the testing data (Set J):

Precision: 86.4%

Recall: 86.4%

For reference, our results on the development data (Set I):

Precision: 93.7%

Recall: 97.3%

EXTRACTOR METHODOLOGY

Before developing the extractors, we expanded upon the given dictionary, which was shared by both. During "tuning" of the extractors, we added some of the brand names which were included in our development data (Set I) but not in the original dictionary. We did *not* add brand names which only existed in Set J but not Set I or the original dictionary, as we felt this would invalidate our final results.

Then, we ran the whole set of brand names through another program to generate variations. For example, for any brand with the word "and", we also generated those same brands with "and" replaced with "&" and "+". We also generated synonyms for each brand, adding words such as "Inc.", "Corp.", ".com", and so on. Overall, in this way we took a wide net approach.

Our final extractor's approach is to store our enhanced brand name dictionary in an unordered set for easy lookup. While building the dictionary, the program tracks the largest-sized string as this is used to aid matching candidate brand names to brands in the dictionary (without evaluating every possible substring in the product name).

Then, as the extractor executes, as it reads in each product name from the test set, we slide a "window" through the string, starting at the front where the brand name tends to be more likely to reside. This window starts at the size(in # words) of the largest brand name identified during the dictionary load.

The program then slides this window left to right, evaluating each contiguous substring of that size in the product name. For each substring brand candidate, we do an O(1) lookup in the loaded dictionary to determine whether the candidate is indeed a known brand. If so, we stop looking for further results, and move onto the next product name in the file.

On the other hand, if the window reaches the end of the product name without finding a brand name of that size, it starts back at the beginning with a window one word size smaller, continuing in this way until we slide through looking for single-word brand names. If no brand name is found, we consider it null and move on to the next product.

The primary reason we handle it in this way is to ensure that we always identify the most specific brand name possible. By starting with the largest-possible size brand name in our set and iterating downwards, in cases where the brand might be something like

“Apple Sauce Inc.”, we identify the correct brand instead of finding “Apple” and quitting early.

The extractor can be run in a test mode on our golden data (Set I) to also automatically determine whether the correct brands were extracted, and report precision and accuracy.

As a final note, there is certainly room to improve the efficiency of this extractor method. On the size of our test data set, it worked fine, but may need optimization for real-world datasets. Since efficiency of our algorithm was not noted as a criterion for grading, we valued accuracy over speed in the methodology.

RESULTS ANALYSIS - HOW TO IMPROVE?

Although our results are fairly good, they are not perfect, and in a real-world production system we would want to strive to improve our precision and recall. Here are some reasons why our extractors did not attain perfect accuracy, and potential solutions.

During development testing on Set I, the highest number of errors were false positive cases where common words such as “Advance” and “Comprehensive”, which existed in the brand dictionary, were extracted as the brand names from product names that were in fact a generic product and the brand should have been null. These errors would be challenging to develop an elegant solution for while using a dictionary-based approach to brand extraction.

One way to address this would be to consider eliminating these names from the dictionary if they are causing more harm than good. When scaling the extractor up to apply to a much larger bed of data, we’d need to figure out how often the brand names “Advance” or “Comprehensive” existed as true hits. Depending on whether or not they existed more often than they caused false positives, we could potentially remove those names from the dictionary to improve accuracy.

A different potential solution would be to add extra handling for the specific brand names which are also common words. For example, the brand Century was returned for the product with ID 446015#Walmart.com. In fact, this product should have had a null brand name, and it was simply that the words “21st Century” appeared in the product name. For this case, we could refine the approach and at the time Century is extracted as the brand name, we could do further analysis on whether or not to return Century as the brand name. For example, we could rule it out in cases where it is preceded by the word

“21st” or another similar number. However, adding special case handling like this would greatly complicate the code, and what would we do in the event that we wind up with another brand named “21st Century”? It seems an ML-based solution may be better equipped to handle the false positives that our dictionary-based approach struggled with.

The second most frequent failure in our testing on Set I involved cases where multiple brand names existed in a single product name. One example of this was an accessory for an Apple product which was made by a company named Fosmon. Fosmon was in our dictionary, but it happened that Apple appeared first in the product name and as such, it was returned as the brand name.

To improve accuracy for these cases, we could build a second dictionary of “very common” brand names. Then, during the extraction process, we could check whether the first brand we extract exists in that second dictionary. If it does, we could do a quick “double-check” of the product name with the very common brand name removed. If we find a second brand name, we could return that instead. This would likely resolve more errors than it would cause, but there are surely cases where it would introduce a new failure(example: HP computer with Intel processor).

We saw a drop in both precision and recall on our testing of the Set J test data, largely due to false negatives where the correct brand name was not found in the dictionary. This can primarily be explained by the fact that we intentionally excluded brand names which *only* existed in Set J (and not Set I or the original given dictionary) when building our brand name dictionary, as we felt this would border too closely on “cheating” / looking at the test data. Our precision and recall were still fairly good, however, and would attain similar results as what we saw on Set I if we revised the dictionary by adding the brand names which led to false negative cases.

Overall, what we saw here is that the dictionary-based approach has very high recall with a sufficiently robust dictionary, but is more prone to false positives (thus, its recall will likely be higher than its precision).

OTHER NOTES

What is a brand?

Overall, our approach to what defines a “brand” for the purposes of this project is to get as specific as possible, and include manufacturer+product brand (example: “Apple iPhone”) whenever possible. In cases where we can find both the manufacturer and the

brand name (example: “Nintendo WiiU”), we will return the full concatenation as the brand, but in cases where only the manufacturer can be found (example: “Nintendo”), we return that as the brand, as it meets many people’s subjective definition of a brand and is better than returning nothing.