CS 784 - Project Phase 2: *Brand Name Extraction from Product Name*

Abhinav Mehra, Danish Khan, Mehreen Ali

Introduction

In this stage, we automated brand name extraction from the “Product Name” attribute value in the JSON representation of a product. We followed a dictionary-based approach. We were provided with a dictionary of ~8K brand name values that was extracted by processing a large database of electronic products. We followed following steps for our automation method development and result analysis:

I. From a given data set of labeled product pairs (approximately ten thousand such pairs), we randomly sampled 350 products. For each product, we extracted the brand name from the product name attribute manually. This information is available in “golden_data.txt” file.
II. This sample is further split randomly into a development set and a testing set. We have 220 product names in our training/development set and 129 products in testing set. Training set is stored in “setI.txt” file and test set in “setJ.txt”.
III. Develop extractor function and use training set to tune our function. Then, use the developed function on test set and calculate the precision and recall. Section 3 talks in detail about the extractor function.

Precision and Recall

We used following method to calculate precision and recall for our brand extractor function on test data.

Precision = (#True_Positives)/(#True_Positives + #False_Positives)

Recall = (#True_Positives)/(#True_Positives + #False_Negatives)

True_Positive: Count of products where brand extracted by function is same as brand assigned manually on sample data set.

True_Negative: Count of products where brand extracted by function is wrong from sample data.

Fase_Positive: Count of products where brand extractor function predicted a brand but actual sample data didn’t have a brand name.

False_Negative: Count of products where brand extractor function didn’t predict a brand but actual sample data had an associated brand.

Same data with assigned brand names is stored in “golden_data.txt” file. Result of predicted brand names by brand extractor function is stored in “result.txt” file.

Using above method, we got following results for our brand extractor on the test data:

Precision	93.2%
Recall	95.4%

Brand Name Extractor Description
We used a dictionary based approach where we were given a long list of brand names. We developed our extractor function while working with the training data. We kept extending our given brand list in each iteration and then later on we used this extended brand list on test data. Our extractor function used following steps to extract the brand name from its given product name:

1. Generate combinations of words in the given product name string. During the training phase, we made this little smarter by removing non-relevant combinations including braces, special characters, etc. For example, product name “Maxell 625156 CD-R Music Discs” generates following combinations each of which is looked into the brand dictionary: {“Maxell”, “CD-R”, “Music”, “Discs”, “Maxell CD-R”, “Maxell Music”, etc.}
2. We took a list of permissible suffixes which if absent or present in the given search string from the brand name, should still be assigned that brand name. For example, “Apple Inc.” and “Apple” both should be assigned the brand name as “Apple”. We generated a list of permissible suffixes while working on training data. Later we extended our brand list with brands with these suffixes as well.
3. We also took into consideration space and punctuations related differences in our extractor function. Function removes all tab and multiple spaces in the product name before splitting into words.
4. For each combination, we looked up for a potential match in the given brand name dictionary. For comparing brand names, we used following condition:
    a. If two strings (one from the generated word combination from product name, second from the dictionary) matches completely, then assign the brand name to current product.

Result Analysis

We started extracting brand names from product name on the training data by directly comparing with given brand dictionary. Doing so, initially gave poor precision and recall (precision = 0.86, recall = 0.74). While looking at the errors, we observed a lot of mismatch were related to common prefix, suffix related mis-matches. This resulted in false negative results giving lower recall on our test data. Extending our brand dictionary with common such mismatch helped increase both precision and recall in training data. We did not implement edit distance with a threshold distance for acceptance in our current implementation, but we expect to see improvement by applying this approach.

Another interesting observation was product names having multiple brand names. 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.

Appendix

Golden Data File
http://pages.cs.wisc.edu/~dkhan/golden_data.txt

Test/Development Set File
http://pages.cs.wisc.edu/~dkhan/setI.txt

Training/Data File
http://pages.cs.wisc.edu/~dkhan/setJ.txt

Code for Extractor Function:
https://github.com/DanishKhan14/CW-UWMadison/blob/master/CS784/brandExtractor.py

Result for test data:
http://pages.cs.wisc.edu/~dkhan/result.txt

README
http://pages.cs.wisc.edu/~dkhan/README