Project 2 Report - Task 2

Hoang Dang 1080344

Task 1a. The algorithm takes the data into *DataFrame*'s and cleans the data by removing all characters that are not numerical or alphabetical and enforces all characters to lowercase. It was manually determined that *abt'*s name field always starts with the brand and ends with the serial number of the product. The *buy* data is not always nicely formatted, however, most of this data is in a similar format to *abt*. Therefore, the program assumes that the first and last words in a product name are brands and serial numbers respectively and checks all the data for cases that perfectly match both in brand and serial number. In the case of *buy*, the manufacturer is also checked if brand is not a match. After that, all matched products are removed from the tables, and a second check is carried out to find as many edge cases as possible. These checks include: shortening the length of the current serial number by 1 character to allow for small errors in serial numbers; finding all other substrings inside the data that may potentially be serial numbers and cross checking; and using *fuzzywuzzy*'s *token_set_ratio()* to check the names and descriptions for at least 90% matching in wording. The condition for edge cases is that the brands must first match, and then any of the above conditions satisfied, as the probability of such intersection being a false positive is low.

Evaluation. Overall, the algorithm performs well with > 0.9 Precision and Recall, but still struggles to cover all the edge cases where data is not well-formatted. To improve performance, future implementations can sacrifice storage for better language processing. In particular, if two products are matched, the brand and serial number may be kept in dictionaries to help with brand-identification in edge cases and help remove duplication of products with the same serial number. The algorithm can also implement blocking where products that don't match on brand and serial number can be put into blocks, then compared on the basis of noun and/or adjective matching using the *nltk* library. If the products match at least 90% for noun and/or adjective matching, as well as some matching of unrecognised substrings (ie. 1080p, 48GB) – as these usually help distinguish products – there is a good probability of this matching being a true positive.

Task 1b. This script cleans data similarly to part 1a, leaving only alphabetical and numerical characters and enforcing all data into lowercase. The blocking method is primarily based on the insight that brand information is usually the first substring in the product name. Hence, the program assumes that the first words in the name field of all products are brands and puts all products of the same brand into one block. Because the *abt* data generally has better formatting, the algorithm starts by taking all the brands from *abt* and assigning blocks; then it iterates through *buy* and checks to see if the brand of the *buy* product is already part of the dictionary of blocks. In the case of *buy*, if the brand doesn't match any existing blocks, the manufacturer is also checked for matches. If no existing blocks can be found, new blocks are also assigned for *buy*.

Evaluation. This algorithm works well for its purpose, achieving almost perfect Pair Completeness and excellent Reduction Ratio. However, it still generates a substantial number of False Positive counts given the simplistic nature of the design. To further reduce this False Positive count and reduce potential build up in computational time in later processing, a more precise model should be implemented. To achieve better precision, current blocks can be further broken down into smaller categories by noun or adjective. Again, the use of the *ntlk* library may come in handy here. For example, a product with the brand 'Apple' may have blocks 'ApplePhone', 'AppleWatch', 'AppleCase', etc… The main issue with this approach is picking which substring to add to the brand for blocking, since picking the wrong substring would actually harm both Precision and Pair-Completeness. One simple strategy is to sort all the nouns of the product in ascending order and pick the first one in the list.