Jeanelle Abanto
1133815
JABANTO

In doing the product comparison, the amazon titles and google names were used. Preprocessing done includes removing symbols, transforming texts to lowercase, removing stopwords, lemmatizing the words, removing duplicate words, and sorting words lexicographically to improve string match results as much as possible. For each preprocessed amazon title, the similarity function *JaroWinkler()* was used to get the normalized similarity scores with each google name. The function parameter *qval* was set to 3 in order to get an optimal result. Setting *qval* to 2 yielded more false positives, whereas setting *qval* to 4 yielded less true positives. The minimum threshold for the similarity function was set to 0.52 to get a balanced result of recall and precision. The google name with the highest similarity score with the amazon item is then considered a match. A threshold below 0.52 yields a precision below 90% and recall above 90%. On the other hand, a threshold above 0.52 yields a recall below 90%, with precision above 90%. For the purposes of achieving recall and precision of 90%, the threshold 0.52 yields just the right balance with recall at 90%, and precision at 91%. Overall, the product comparison using *JaroWinkler()* yielded satisfactory recall and precision results. There exist different similarity functions, e.g. token-based similarity functions, and others. With token-based similarity functions, the description may be used to compare the similarity of two items. Checking for the similarity in description may help improve identifying similarities between two items by adding information to just the similarity or dissimilarity in item name.

A similar preprocessing was applied on amazon titles and google names for blocking, with the addition of removing numbers. After preprocessing, the texts were split into sorted tokens. The blocking principle implemented is similar to the idea of bigrams, i.e., the blocking keys were generated by taking a combination of two tokens in order to place records with possible word insertions into the same block, e.g., 'amazon block key' and 'amazon key' can be put in the same block by taking a combination of 2 tokens. The initial keys are generated from the amazon titles and are put into a list thus, if the generated google block key already exists in the list, it means the item will be put in the same block as an amazon item. If the generated google block key does not exist in the list, it means the key does not have a match with amazon thus, the google item will be put into a 'g_nomatch' block. Generated amazon keys which have matches with google are placed on a list of matched keys. Generated amazon keys that do not exist on the list of matched keys are changed to 'a_nomatch' key indicating that the block does not have a match in google. Duplicate items are then removed afterwards, and finally, all items in '*_nomatch' blocks are removed if the item exists on other blocks with a potential match. Thus, items left on '*_nomatch' blocks are presumed to have no match. Although the blocking program runs a bit slow due to the generation of combinations of 2 words with complexity $O(n\ C(m,2))$ where m is the length of the title and n is the number of records, the blocking method was able to achieve PC and RR measurements of 90% and 99% respectively. To achieve faster running time, the item names can be limited to a max number of words or, an entirely new blocking method without the use of combinations can be implemented to improve time complexity, e.g., using tokens as keys. However, use of tokens alone may result in a drop in RR as it will put more records into a single block.