Objective

The objective is to create the maximum possible matches from different partner inventories, given that the level of accuracy in the match is high enough.

Why? - To remove potential duplication and provide better customer experience on Agoda website and app

Approach

Since matches between different partner inventories might not be always exact, I'll use fuzzy matching algorithm to calculate matching score. One of the popular fuzzy matching algorithm is Levenshtein ratio.

Levenshtein distance (lev(a,b)) is defined as minimum number of single character edits(insertion, deletion, substitution) required to convert one string into another.

Levenshtein ratio: (|a| + |b| - lev(a,b)) / (|a| + |b|) where |a| = length of string a; |b| = length of string b.

If both string are identical lev(a,b) will 0, hence Levenshtein ratio will be 1.

Following are the basic steps in the implementation:

1. String Pre-processing and cleaning (Make lowercase, remove spaces, comma, period)
2. Create a *mapping_key* (which can be a concatenation of hotel name or hotel address or hotel city). This key is used to calculate matching score
3. Encode *mapping_key* into utf-8
4. For a given country, calculate matching score of a *p1.mapping_key* from partner1 with all *p2.mapping_key* in partner2. Extract *p2.mapping_key* with the maximum matching score. Do this for all countries and all *p1.mapping_key* from partner1.
5. Return the output containing best match for all *p1.mapping_key* in partner1
6. Evaluate the performance of the model on the example dataset using metric precision and recall
    a. Precision: of all the predicted mappings, how many were correct matches
    b. Recall: of all the mappings present in the data, how many were correctly matched
7. Identify threshold based on precision-recall trade-off to achieve >98% precision
8. Set this threshold and calculate the mapping from mappinghotelsdataset.xlsx datasets. Output the final mapping for submission

Result

Below are the comparison of two matching algorithm implemented on example data:

1. lev from Levenshtein module
2. fuzz from fuzzywuzzy module

Performance comparison of different model with different mapping key:

for mapping_key = [hotel name + hotel address]

Model	Threshold	Precision	Recall	Runtime*(example)	Runtime*(main data)
lev	0.53	98.5%	65.9%	6.1 sec	45 sec
fuzz	0.56	98.3%	69.1%	6.5 sec	162 sec

for mapping_key = [hotel name + hotel city]

Model	Threshold	Precision	Recall	Runtime*(example)	Runtime*(main data)
lev	**0.61**	**98.6%**	**72.9%**	**6.0 sec**	**29 sec**
fuzz	0.64	98.1%	73.5%	6.4 sec	145 sec

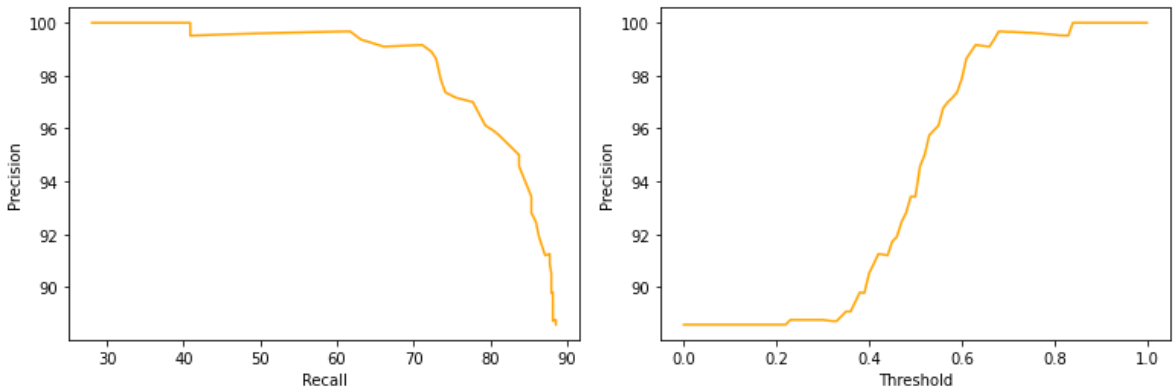
for mapping_key = [hotel name]

Model	Threshold	Precision	Recall	Runtime*(example)	Runtime*(main data)
lev	0.69	98.1%	73.7%	6.0 sec	25 sec
fuzz	0.73	98.1%	73.3%	6.5 sec	140 sec

*rough estimate based on run on local machine(macbook pro, 2.8 GHz Quad-Core Intel Core i7, 16 GB RAM)

Fuzz has better recall generally then lev but is considerably slower in performance. So, for better performance, the final output file(mapping.csv) was generated using lev and mapping key of hotel name + hotel city.

Precision-Recall trade-off graph for levenshtein implementation:



As seen in the above graph, Precision goes down as we increase recall. To achieve a precision of above 98%, the best recall we get is close to 73% and corresponding threshold of matching score is ~0.61

Also interesting to note that just using hotel name gives better precision-recall trade-off on example data. But this was not preferred as it might lead to increase in false positive (matching exact hotel name in two different city).

Recall on mappinghotelsdataset.xlsx = (Total rows matched)/ total rows = 8283/10000 ~83%

Possible Improvements

1. Further pre-processing of string might improve the performance and accuracy of the model
2. Use pincode to improve performance (by matching only within pincode for a country). Might miss matches if pincode info is not present in either of the dataset
3. Try more advance string matching approach like word2vec with similarity metrics like cosine