INF558 HW7 Report

Task 1 & Task 2

String similarity is calculated in the function SimilarityScore, component includes name, address, phone, cuisine.

- Name: Jaccard similarity. Name is separated by space and find the similarity between two sets. String in name often overlapped in some words, for example, {'katsu'} { 'katsu', 'restaurant'}, so the method can help get better similarity.
- Address: Levenshtein is applied because sometimes there are abbreviation in address, for example, st. – street, and the method can get higher similarity for the case.
- Phone: The value is highly accurate and identical for the matched records, therefore, the matched record got 1, otherwise, 0.
- Cuisine: Levenshtein is applied for the attribute because it can get higher similarity for the case like “French – French(New)”, but it’s still difficult to match due to inconsistency values between two files and thus zero weight is assigned to this attribute.

```
def SimilarityScore(record1, record2):
    names = rltk.jaccard_index_similarity(record1.name, record2.name)
    address = rltk.levenshtein_similarity(record1.address, record2.address)
    cuisine = rltk.levenshtein_similarity(record1.cuisine, record2.cuisine)
#     phone = rltk.levenshtein_similarity(record1.phone, record2.phone)

    if record1.phone != record2.phone:
        phone = 0.
    else: phone = 1.
    #0.7  0.2 0.1 > 0.8 104
    #0.4 0.4 0.2 >0.59 106
    #0.4 0.4 0.2 >0.53 113
    return 0.4*phone + 0.4*names + 0.2*address
```

Following is the weight I’ve tried and their result.

1. 0.7*phone + 0.2*names + 0.1*address threshold: 0.8 matched pair: 104
2. 0.4*phone + 0.4*names + 0.2*address threshold: 0.59 matched pair: 106
3. 0.4*phone + 0.4*names + 0.2*address threshold: 0.53 matched pair: 113

Since phone is less likely to have error or typo among all attributes, so the weight is greater than others in the first trial. But from the result of matched record it can tell that name is highly accurate so increase the weight for name and finally, inspect the similarity for wrong matched records to adjust the threshold.

Task 3

For scaling up, I used hashing to speed up the process. The hash function will add up all the digits in the phone and get the remainder divide by 11. After applying the

hash function, there will be a total of 16413 comparisons. Following is the record distribution in each chunk.

```
Zagats: {8: 31, 1: 35, 5: 31, 10: 26, 7: 45, 0: 24, 4: 38, 2: 22, 9: 35, 3: 22, 6: 22}

Fodors: {0: 55, 1: 44, 5: 52, 8: 39, 4: 50, 9: 65, 2: 43, 10: 46, 3: 46, 7: 59, 6: 34}
```

Before applying hash function, there are 114 matches, and got 113 matches afterwards. Following figure is the missing record. I think the reason of missing 1 record is because they are thrown into different chunks in the first place, therefore, they are not able to match even they pass the threshold.

Palace Court 3570 Las Vegas Blvd. S Las Vegas	"702/731-7547"	"Continental"
Palace Court 3570 Las Vegas Blvd. S. Las Vegas	"702-731-7110"	"French (New)"