Comp 480/580: Assignment #3

Rice University — Due Date: Thursday, 03/28/2019

1 Build your own Approximate Search Engine with MinHash

In class, we learned MinHash (or Minwise Hashing), where we used different hash-functions and converts each set S into m values of $h_{min}(S)$ for these m functions $h_1, h_2, ..., h_m$. Recall that the probability that $h_{min}(A) = h_{min}(B)$ is true is equal to the Jaccard similarity $J(A, B)$ of set A and B.

1.1 Warmup Tasks

- 1. Implement simple MinHash signatures generator. Given an input string and m, return m hash-codes (you can re-use the same hash function you implemented when hashing the 3-grams of the string).
- 2. Implement MinHash hashtable class which has insert and look-up function. The constructor takes in four parameters, K, number of hash functions and L, number of hash tables, B the size of hash tables (or the number of the buckets in hash tables). The function "insert" will take in hashcodes and an id. The role of the function is to insert this id into a different bucket in each hash table according to its hashcodes (as explained in class). Then "lookup" will retrieve all the items in the hash tables according to the supplied hashcodes (as explained in class).

1.1.1 Deliverables

MinHash.py

```
def MinHash(A, k):
...

class HashTable():
    def __init__(self, K, L, B, R):
        ...

    def insert(self, hashcodes, id):
        ...

    def lookup(self, hashcodes):
        ...
```

1.2 The Main Component

- Download Aol dataset which includes anonymized user ids and click data. (url)
- **Data Preprocessing**: get the unique urls from the data file.

```
import csv
import pandas as pd

data = pd.read_csv('user-ct-test-collection-01.txt', sep="\t")
urllist = data.ClickURL.dropna().unique()
```

- Insert all URLs to your MinHash Hashtables. Use $K = 2, L = 50, B = 64, R = 2^{20}$ You can treat each URL as a string and use 3-gram in your MinHash function.
- Sample 200 URLs from urllist as the query set. For each URL, compute the MinHash codes and retrieve all the items in the hash tables. Evaluate the quality of retrieved items by reporting the average Jaccard similarity of the query URL and retrieved URLs.
    1. Report the mean Jaccard similarity of the URL retrieved.
    2. Now for every retrieved set (candidate set), filter it to find the top-10 URLs only. Report the mean Jaccard similarity of the top-10 URLs retrieved after filtering.
    3. Report the query time.
- Write a function to compute the pairwise Jaccard similarity of those 200 URLs in the query set with all the other elements using a brute-force way. Use the computational time for this task to estimate the expected total time of computing the pairwise Jaccard similarity of all URLs.
    1. Report the total time and time per query, compare it with the one from the previous step.
- Tuning K,L (K = $1, 2, 3, 4, 5, 6$) and $L = 20, 50, 100, 200$ observe how the average jaccard similarities of the query url and retrieved urls change accordingly.
    1. Report the mean Jaccard similarity of the URL retrieved and also the time per query for each of the combination.