Advance Data Structures and Algorithms (SCS 2009)

Group Assignment 1: Locality Sensitive Hashing

Deadline : 18th November 2013

1. Locality Sensitive Hashing (LSH) implementation

In this problem set, you will implement the locality sensitive hashing algorithm to find pairs of sentences that are copies of one another. Locality sensitive hashing algorithm is described in the document **Mining of Massive Datasets by A. Rajaraman and J. Ullman1** and that complete document is uploaded with your assignment. However, since it contains some advanced concepts which may not become applicable to this assignment an edited and shorter version of the same document is also uploaded to the LMS. The idea is that you implement a LSH algorithm that finds pairs of very similar/duplicate sentences using *the data set that would also be uploaded to the LMS shortly.*

Nature of the data set

The data set would contain simple sentences in a text file and each line of the file contains a single sentence. First field in the file is the sentence id which is the followed by the words of the sentence. The sentence should be free from all punctuations and all words in the sentence are separated by a single space. Up until the data set is uploaded (may be 2000 sentences) you may test your implementation using a simple data set at your discretion.

The Task

The assignment asks you to implement Locality Sensitive Hashing based approach to quickly finding pairs of sentences that are at the word level edit distance at most 1. This means that given two sentences S1 and S2 they are at edit distance 1 if S1 can be transformed to S2 by: adding, removing or substituting a single word. You may read the documents stated above to get a better idea.

For example, consider the following sentences where each letter represents a word:

- S_1: A B C D
- S_2: A B X D
- S_3: A B C
- S_4: A B X C

Then pairs the following pairs of sentences are at word edit distance 1 or less: (S1, S2),
(S_1, S_3), (S_2, S_4), (S_3, S_4).

Step 1: Data preprocessing

In this step, you need to read the input file, parse it, and convert words of sentence to unique integer ids. Replacing word ids with their integer ids will later allow you to more efficiently work with the data. A simple strategy for this conversion is to simply read over the input data and build a hash table that maps words to unique ids.
Also, you might want to remove duplicate sentences from the file — check out the 2nd implementation tip of Step 4.

In the example below each word is replaced by its word id.
•Input1: I love big data mining

•Input2: You love big data base

•Output1: 1 2 3 4 5

•Output2: 6 2 3 4 7

Implementational tip: When testing/debugging your code use small examples and small
datasets before you run it on full data. This will considerably speed up your progress.
Generally it will also be useful for you to save intermediate outputs of each step so that you
do not need to parse/generate all the data from scratch (i.e., step 1).

Step 2: Shingling

Now we aim to take the sentences and generate shingles. You will now convert the word- integer lists from step 1 to list of 3-shingles. Come up with some way to hash each 3-shingle to a unique integer.

•Input1: 1 2 3 4 5

•Input2: 6 2 3 4 7

•Output1: 47 82 23

•Output2: 21 82 29

Notice how (2, 3, 4) is hashed to the same integer 82.

Implementational tip: Very similar to step 1, you will need to create the hash for the 3-shingles. A Clever choice of a hash function is very important. Actually, it may be beneficial to skip step 1 and directly hash shigles of word strings rather than shingles of word ids.

Step 3: Locality Sensitive Hashing

find duplicates

Now you will generate sets of candidate pairs using LSH. Generate min-hash signature matrix using permutation hash functions. Our recommendation is to use 3 bands of 8 rows each. You can further tune these values yourself to yield best results. Output of this step would be the set of candidate pairs of sentences, that have Jaccard Similarity larger than a certain threshold.

Implementational tip: In the assignment, the use of good hash function is very important. Use of MD5 or SHA hash function would yield good results but such hash functions are slow to use. Put some effort to devise a clever hash function.

Step 4: Output generation

Based on candidate pairs obtained by LSH now output the IDs of pairs of sentences that are at word edit distance 1 or less. Each line of the output file ***scs2009_sentences*.out** should contain two numbers separated by space: "<I> <J>\n", which means that sentence IDs I and J are at word edit distance of 1 or less. You need to save each pair only once (i.e., only save when I < J). You do not need to save pairs of identical sentences.

Implementational tip: You can quickly check whether two sequences I and J (assume I is longer than J, |I| _ |J|) are edit distance 1 apart. First if I and J differ in length for more than 1 element, then they are surely more than edit distance 1 apart. Second, we can then do the following: first we traverse over the common prefix of I and J and the check for two cases: addition/deletion of a element and substitution of a element: (addition) skip 1 element of I and then share the rest of the elements with J (since J is shorter this is the same as inserting an element in J); (2) skip 1 element in both I and J (this corresponds to substitution) and then check that I and J share the rest of the elements.

Impementational tip: You may need to remove duplicate sentences as a preprocessing.

Submission instructions

- Please submit your solution by uploading a zip file on or before the deadline (18th November 2013) and make sure that you name your file using the Group number.
- Upload all your code and the output file **scs2009_sentences.out** that contains IDs of pairs of sentences that are at word edit distance of 1 or less and also submit the same in a report and on the cover page of your report include the index numbers of all the group members .

- Each line of the output file should contain 2 space separated **numbers: <I> <J>\n,** which means that sentence ids I and J are at word edit distance of 1 or less.
- You need to save each pair only once (i.e., only when I < J). You can also include a README if there are notes that you would like us to look at.
- A good solution should run and produce good results.