MDA HW4 Report

105030013 張義群

1. Introduction

In this homework, we are asked to find similar documents by implementing Locality-sensitive Hashing in Map/Reduce concept. The algorithm consist of three steps: Shingling, Min-Hashing and Locality-Sensitive Hashing.

2. Implementation and Map/Reduce design

First parse all the input files and form a set of shingles, then turn them to RDDs for spark.

docs_rdd: (doc_id, [all k_grams tuples in doc_id])
shingles_rdd: (shing_id, (k-gram tuple))

1. Shingling

Use .cartesian() to get a cross product of docs_rdd and shingles_rdd, then use a mapper has_shingle() to get a (doc_id, (sh_id, bool: whether sh_id is in doc_id)) pair. After that we can use a filter to keep only the shingles that appears in each document, key-value pair looks like (doc_id, [sh_ids that is in doc_id]).

2. Min-Hashing

A hundred of randomly generated hash functions are used in this step to simulate the permutation step in Min-Hashing. Since we have collected the sh_ids (row) that is 1 in each doc_id (column), different from the approach shown in Ch.3 slides (p.43~p.44), we can directly hash all the rows and get the minimum value of all hashed rows, and that is our min-hash value. The mapper function for above operation is get_signature(). Besides from hashing the sh_ids, in this mapper function we also assign a band_id, which is useful when we perform LSH.

3. LSH

In LSH, candidate pairs are those that hash to the same bucket for ≧ 1 band. In this homework, the number of band is 50, which means that we have 2 rows in each band. To get candidate pairs, we group the pairs form last step by band_id, then use the mapper function hash_to_bucket() to hash the signature matrix values to k=10000 buckets. Mapper function hash_to_bucket() outputs ((band_id, bucket_id), [doc_id]) pairs, we the can use reduce function to group all pairs with the same key and keep only with more than one doc_id in the value list.

One example of the output pairs is ((39, 1259), ['030', '035']), means that the 39th band in document '030', '035' are hashed to bucket 1259.

4. Calculate Jaccard Similarity

After getting all the candidate pairs, we have to concatenate the characteristic vectors to each doc_id from step 1. to calculate the Jaccard Similarity between documents. The function .join() is helpful for doing this, after that, few mappers such as candidates_mapper() are used to rearrange the pairs. Noted that there might be more than two documents with same band_id hashed to the same bucket, so we need find all the combinations of candidates and calculate their similarity separately. The way to calculate the Jaccard Similarity in calculate_jaccard_sim() is add the vectors of the candidate pairs and find the ratio of (count of 2) / (count of 2 + count of 1), since 2 means both of them have one and this is the intersection. Lastly, get the distinct pairs using simple reducer function and sort them by similarity in decreasing order.