巨量資料 - 作業四

108062566 資工碩一 陳法佑

0. LSH 由 Shingles, MinHashing, Locality_sensitive_hashing 組合而成

```
def LSH(sc, docs_dir_path, words_per_shingle, minhash_func_nums, lsh_row_nums, output_top_n):
    doc_shingles_dict = shingling(sc, docs_dir_path, words_per_shingle)
    min_hashes_values = min_hashing(sc, doc_shingles_dict, minhash_func_nums)
    buckets_candidates = locality_sensitive_hashing(sc, min_hashes_values, lsh_row_nums)
    compute_all_candidates_pairs_similarity(sc, min_hashes_values, buckets_candidates, output_top_n)
```

結果

(Doc_idx1, Doc_idx2): Jaccard Similarity

(12, 20) : 1.0

(52, 84) : 1.0

(30, 35) : 0.75

(47, 49) : 0.73

(48, 49) : 0.55

(88, 49) : 0.44

(38, 23) : 0.43

(48, 47) : 0.42

(14, 40) : 0.36

(88, 47) : 0.3

1. Shingling

```
def doc_to_shingle_map(line, words_per_shingle):
    words = line.split(' ')
    shingles = []
    for i in range(len(words)-words_per_shingle+1):
        shingles.append(tuple((words[i: i+words_per_shingle])))
    return shingles


def shingling(sc, docs_dir_path, words_per_shingle):
    print("----------------------------shingling----------------------------")
    doc_shingles_dict = {}
    docs_names = os.listdir(docs_dir_path)
    docs_names.sort(key=lambda x: int(x.split('.')[0]))
    for doc_idx, doc_name in list(enumerate(docs_names, start=1)):
        doc_path = os.path.join(docs_dir_path, doc_name)
        doc_shingles_dict[doc_idx] = set(sc.textFile(doc_path).flatMap(
            lambda x: doc_to_shingle_map(x, words_per_shingle)
        ).collect())
    return doc_shingles_dict
```

將 documents 按照名稱排序後，用 map 對各個 document 搜集他們的 shingles，最後存入 dict。

2. MinHashing

```python
def generate_hash_func(N, p):
    a = random.randint(0, 10000)
    b = random.randint(0, 10000)
    return lambda x: ((a * x + b) % p) % N


def shingles_minHashValue_map(shingles, hash_func, shingle_row_dict):
    min_hash_value = float("inf")
    for shingle in shingles:
        row_idx = shingle_row_dict[shingle]
        hash_value = hash_func(row_idx)
        if hash_value < min_hash_value:
            min_hash_value = hash_value
    return min_hash_value


def min_hashing(sc, doc_shingles_dict, hash_func_nums):
    print("------------------------------min_hashing------------------------------")
    total_shingles = set([item for sublist in doc_shingles_dict.values() for item in sublist])
    N = len(total_shingles)  # 26320
    p = 26339  # p > N and p is a prime number
    shingle_row_dict = dict(zip(total_shingles, list(range(1, N+1))))
    hash_funcs = [generate_hash_func(N, p) for _ in range(hash_func_nums)]
    docs_shingles_rdd = sc.parallelize(list(doc_shingles_dict.values()))
    min_hashes_values = []
    for hash_func in hash_funcs:
        min_hash_values = docs_shingles_rdd.map(
            lambda doc_shingles, hash_func=hash_func: shingles_minHashValue_map(doc_shingles, hash_func,
                                                                                shingle_row_dict)
        )
        min_hashes_values.append(min_hash_values)
    return min_hashes_values
```

將所有 documents 的 shingles 搜集在一起並存成 set 的形式，然後將每個 shingles 賦予一對一 1 到 N 的整數存成一個 「shingles_row_dict」，此處假設總共有 N 個 shingles。接著生成 100 個隨機的 Hashing Functions，再以 for loop 循環，每一次的 iteration 計算從「第一階段 Shingling」output 的 dict 中所有 documents 的 minHash 值，計算時會用到 「hash_function」、「shingles_row_dict」及「一個 document 的所有 shingles」。而每一次的 iteration output 的維度是 (1, 101)，而總共 iterates 100 次，所以最終 output 的維度是 (100, 101)。

3. Locality-Sensitive-Hashing

```python
def locality_sensitive_hashing(sc, min_hashes_values, row_nums):
    print("----------------------------locality_sensitive_hashing----------------------------")
    band_nums = len(min_hashes_values) // row_nums
    doc_idxs_rdd = sc.parallelize(list(range(1, min_hashes_values[0].count()+1)))
    all_docIdxs_with_minHashVals = []
    for min_hash_values in min_hashes_values:
        all_docIdxs_with_minHashVals.append(doc_idxs_rdd.zip(min_hash_values))
    all_buckets_candidates = None
    for i in range(band_nums):
        band_docIdxs_with_minHashVals = None
        for j in range(row_nums):
            row_idx = i * row_nums + j
            temp = all_docIdxs_with_minHashVals[row_idx]
            if j == 0:
                band_docIdxs_with_minHashVals = temp
            else:
                band_docIdxs_with_minHashVals = band_docIdxs_with_minHashVals.join(temp)
        band_minHashVals_with_docIdxs = band_docIdxs_with_minHashVals.map(lambda x: (x[1], [x[0]]))
        baskets_with_docIdxs = band_minHashVals_with_docIdxs.reduceByKey(lambda x, y: x + y)
        candidates = baskets_with_docIdxs.filter(lambda x: len(x[1]) > 1).map(lambda x: tuple(x[1]))
        if not all_buckets_candidates:
            all_buckets_candidates = candidates
        else:
            all_buckets_candidates = all_buckets_candidates.union(candidates)
    all_buckets_candidates = list(set(all_buckets_candidates.collect()))
    return all_buckets_candidates
```

因為「第二階段 MinHashing」output 的「min_hashes_values」，他的 row 是「各個 hash function」、col 是「各個 document」，但又因為「Locality-Sensitive-Hashing」需要用到同個 document 的兩兩 signatures，所以需要先做一個轉換。轉換方式是將「min_hashes_values」中每一個 item 都附上其 column index 也就是 document index 的值，接著以 for loop iterates 「band_nums」次，每一次將 「min_hashes_values」的 2 個 rows（題目要求） join 起來組合成「band」，此時資料形式為: [(doc_idx, (row_i_min_hash_value, row_j_min_hash_value))…..]，接著將 key, value 互換順序，表示成 [((row_i_min_hash_value, row_j_min_hash_value), doc_idx)…..]，此時的 key 等同於一個「basket」，如果兩個 document 的 key 一樣，表示落入同一個 basket，因此實作中使用 「reduceByKey」實作此想法，最後再把 reduce 後有兩個以上落入到同一個 basket 的 document indexes 抓出來作為 candidates。

4. compute_similarity

```python
def compute_all_candidates_pairs_similarity(sc, min_hashes_values, buckets_candidates, top_n):
    print("----------compute_similarity----------")
    min_hashes_values = [min_hash_values.collect() for min_hash_values in min_hashes_values]
    pairs_similarity_dict = {}
    for bucket_candidates in buckets_candidates:
        for left_idx in range(len(bucket_candidates)-1):
            for right_idx in range(left_idx+1, len(bucket_candidates)):
                left_doc_idx = bucket_candidates[left_idx]
                right_doc_idx = bucket_candidates[right_idx]
                same_nums = 0
                total_nums = len(min_hashes_values)
                for min_hash_values in min_hashes_values:
                    if min_hash_values[left_doc_idx-1] == min_hash_values[right_doc_idx-1]:
                        same_nums += 1
                pairs_similarity_dict[(left_doc_idx, right_doc_idx)] = float(same_nums / total_nums)
    sorted_similarity_pairs_dict = {
        k: v for k, v in sorted(pairs_similarity_dict.items(), key=lambda item: item[1], reverse=True)
    }

    top_n_similarity_pairs = list(sorted_similarity_pairs_dict.items())[:top_n]
    print(top_n_similarity_pairs)
    return top_n_similarity_pairs
```

最後就是將落入同一個 baskets 的 candidates 兩兩計算 Jaccard Similarity，再將相似度從高到低的印出。