Homework 4 : LSH

108062608 黃柏翰

Map Reduce Algorithm and Code

為了後續的多個 Hash function,檔案中的 prime.txt 是為了讀取 hash function 中所需的質數部分

此次作業的程式碼分成 4 個部分:

前處理 -> shingle -> Min Hash -> Reduce -> 找出 top 10 similar

前處理:做 shingle 的前處理，以及 hash function 的準備

```
##############################
#     Preprocess / Build Hash   #
##############################
def shingle(path):
    with open (path, 'r') as f:
        all_data = f.readlines()
    data = ''
    for i in all_data:
        data += i.replace('\n', '').replace(',', '').replace('"', '').replace('.', '').lower()
    data = data.split()
    return data


def update_Uset(data):
    global U_set, counter, U_list
    temp = []
    for i in range(len(data)-2):
        key = data[i]+data[i+1]+data[i+2]
        temp.append(key)
        if key in U_set:
            continue
        U_set.update({key : counter})
        counter += 1
    U_list.append((file_counter, temp))


def build_Uset(path):
    global file_counter
    for file in os.listdir(path):
        if file.endswith(".txt"):
            f_path = path + file
            data = shingle(f_path)
            update_Uset(data)
            file_counter += 1


def preprocess(path):
    global N
    build_Uset(path)
    N = len(U_set)
    # aft_shingle represents the combination of each article
    shingle_res = sc.parallelize(U_list)
    return shingle_res
```

```
def preprocess(path):
    global N
    build_Uset(path)
    N = len(U_set)
    # aft_shingle represents the combination of each article
    shingle_res = sc.parallelize(U_list)
    return shingle_res


# build prime list for hashing
# where prime txt is made of prime find on internet, where > N
# Sample once in a min hashing

def prepare_prime():
    global prime
    path = 'prime.txt'
    with open(path, 'r') as f:
        data = f.readlines()

    d = ''
    for i in data:
        d += i.replace('\n', '')
    prime = list(map(int, d.split()))


def find_prime(number):
    hash_list = []
    for _ in range(number):
        p = prime[random.randrange(len(prime))]
        a = random.randrange(1,200)
        b = random.randrange(1000)
        hash_list.append((p,a,b))
    return hash_list
```

Shingle

先將文章取出三個字為一組的 element, 再將其放到 dictionary, 最後利用查找的方式 assign 存在的 index 值為一

程式碼

```
################################
#            shingle           #
################################
# From preprocessing we get the shingle_res of list, and we hash them into list in rdd
# by get shingle with check function, also create (key, value) pair for it
def check_set(input_list):
    return_list = [0]*N
    for i in input_list[1]:
        return_list [U_set[i]]  = 1
    return [input_list[0], return_list]


#  The result can be build by map two list or maping and creating at the same time
def get_shingle(shingle_res):
    shingle_res = shingle_res.map(check_set)
    return shingle_res
```

Min Hash

將 shingle result, 利用 hashing 找出對應新的順序，並和其 shingle 結果相乘找 min 即為 signature

程式碼

```
#############################
#          Min Hash         #
#############################
# we find all the signature at once
# build min hashing
# filter 0 and find min in each long signature

# Find the minimum value
def filtering(input_list):
    sig = []
    for i in input_list:
        sig.append(min(filter(lambda  x: x>0, i)))
    return sig

# Make the list into (index,0, index, index)
def exe_Min_hash(input_list):
    signature = []
    for p,a,b in hash_list:
        comp = []
        for i in range(len(input_list)):
            index = ((a*(i)+b)%p)%N
            comp.append(index)
        comp_min = [a*b for a,b in zip(comp, input_list)]
        signature.append(comp_min)
    return signature


def Min_hash(shingle_res):
    global hash_list
    hash_list = find_prime(Number)
    MinHash_res = shingle_res.mapValues(exe_Min_hash)
    MinHash_res = MinHash_res.mapValues(filtering)
    return MinHash_res

def update_MinHash(MinHash_res):
    global find_minhash
    for i in MinHash_res:
        find_minhash.update({i[0] : i[1]})
```

LSH

為利用 MinHash 做好的 signature 做 LSH, 將 candidate pair 的 signature 以 jaccard similarity 的方式量測相似度

程式碼

```
################################
#    Locality Sensitive Hash   #
################################
# if hash into same bucket, add the index into bucket
def hash_bucket(input_list):
    global LSH_band
    L = []
    for i in range(0, Number, r):
        p1, a, b = LSH_band[int(i/2)]
        p2, a, b = LSH_band[int(i/2)]
    # Make it into ( (row, N), key)
    # N for the bucket we hash into
        N = ((input_list[1][i]*p1)*(input_list[1][i+1]*p2))%K
        L.append( ((i, N), input_list[0]) )
    return L

def LSH(MinHash_res):
# for updating the LSH band, each band contains 2 row of each document's signature
    global LSH_band
    LSH_band = find_prime(b)
    # Make it into ( (row, N), key) with flatMap to split them
    LSH_res = MinHash_res.map(hash_bucket).flatMap(lambda x : x).map(lambda x : (x[0], [x[1]]) )

    # if the two such pair's (row, N) signature are the same
    # Add them into same bucket, Or just regard it.
    LSH_res = LSH_res.reduceByKey(lambda x, y : x + y)
    # Make it into ( (row, N), key_list), where key_list is the condidate key pair
    LSH_res = LSH_res.filter(lambda x : len(x[1])>1)
    return LSH_res
```

找出 top 10 similar:

```
##############################
#       Find Similar Item      #
##############################
def Jaccard_Similarity(a, b):
    I = 0
    for i in range(len(a)):
        if a[i] == b[i]:
            I += 1
    return(I/100)


def find_similar_item(input_list):
    Similar_list = []
    for i in input_list:
        # find all comb of the condidate key pair
        # compare it with its shingle result (Jaccard Similarity)
        for combination in combinations(i[1], 2):
            Similar_list.append((combination[0]+1, combination[1]+1, Jaccard_Similarity(find_minhash[combination[0]], find_minhash[combination[1]])))

    # list(set(Similar_list) for eliminate the repeated item
    Ans = sorted(list(set(Similar_list)), key = lambda x : x[2], reverse = True)
    return Ans


def write_file(ans):
    with open('output.txt', 'w') as f:
        # cloumns means article number not index
        for i in range(10):
            L = ans[i]
            msg = '(' + str(L[0]) + ',' + str(L[1]) + '):'+ str(L[2])
            f.writelines(msg+"\n")
```

最後寫入 output.txt

Column 表示

Article number, Article number, similarity

(12,20):1.0
(47,49):0.74
(30,35):0.72
(23,38):0.57
(48,49):0.49
(47,48):0.34
(14,40):0.31
(23,48):0.14
(23,47):0.08
(9,22):0.08