Allen kombasseril
z5232188

Report

Below is the code for my function

```
def c2lsh(data_hashes, query_hashes, alpha_m, beta_n):

  y=data_hashes.map(lambda x: (x[0],[abs(a_i - b_i) for a_i, b_i in zip(x[1], query_hashes)]))


  offset=0



  while True:

    h=y.filter(lambda x : len([ y for y in x[1] if y<=offset ]) >=alpha_m )

    if(h.count()>=beta_n):
      f=h.keys()
      break

    else:
      offset=offset+1

  return f
```

Q1.

`y=data_hashes.map(lambda x: (x[0],[abs(a_i - b_i) for a_i, b_i in zip(x[1], query_hashes)]))`

This map transformation is used to subtract each list in Data_hashes against query_hash , hence corresponding elements are subtracted and absolute values are saved .

Result will be of format:

[(candidate_key_a,[a1,a2,a3,a4,a4,a5....]),(candidate_key_b,[b1,b2,b3,b4..,]),(candidate_key_,[c,c1,c2,c3,....])]

Now in a while loop, we use

`h=y.filter(lambda x : len([ y for y in x[1] if y<=offset ]) >=alpha_m )`

This filter transformation is used to filter candidates which do not satisfy a criteria,
The criteria is that the number of elements in the list which are greater than of equal to offset should be greater than the alpha_m.

This is run in a while loop , so as the offset increments by 1 each time , more number of candidate keys would be eligible until the number of eligible candidates is greater than beta_m and we return the filtered result.

---

h.count()
This function simply returns the number of elements in the RDD

---

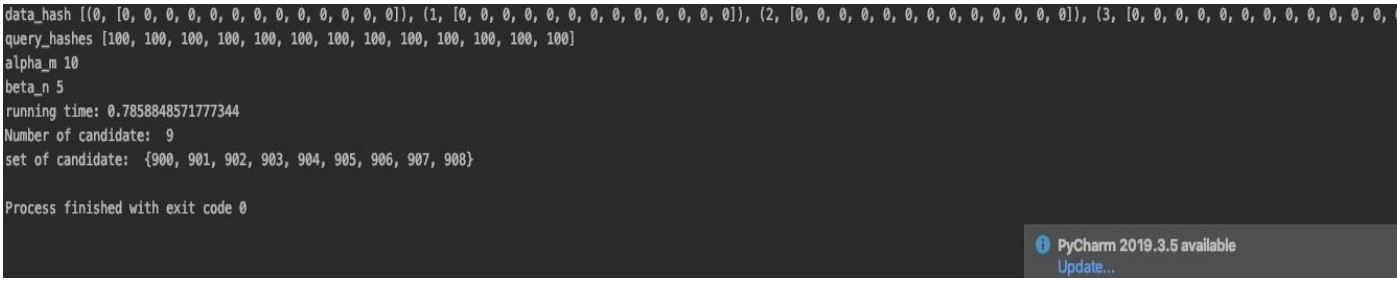
f=h.keys()
This function just returns the keys of the rdd , which are the candidate ids .

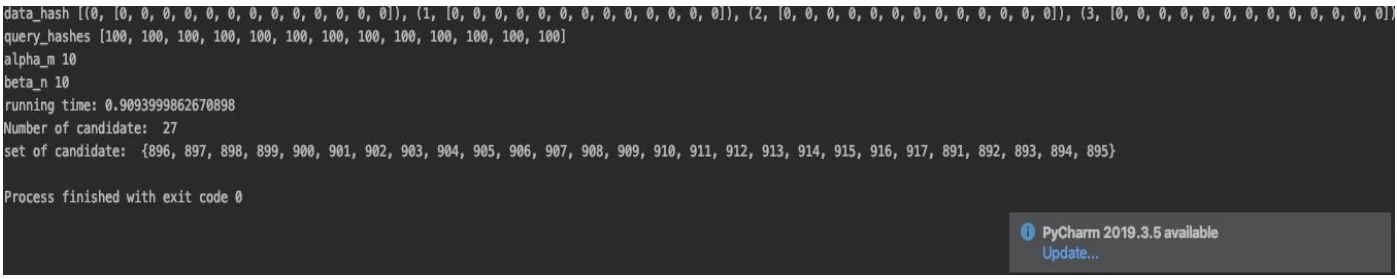
Q2.
Testing:

I first tested my data on the given example, and then I used a generate function to generate random test cases.

I honestly just tested my algorithms with the test cases posted on the forum .
I took the help of the generate function a student posted on the forum to test my test cases, and it was submitting the same results .

The generate function just generate a list of data_hashes randomly from -1000 to 1000 and the number of data_hashes can also be passed as a variable, the query hash was also generated randomly in a similar way.





Q3.

How i improved my algorithm:

By using pyspark we are improving our algorithm by operating on RDDs parallely in each partition ,

We use the power of parallel computation, to compute certain elements in the RDD at once which otherwise require a for loop to iterate through every element and perform an action on it.
Here i have used filter function and map function once , both of them are narrow transformations which do not require re-partitioning , i was careful not to use any wide transformations as they are time consuming because of repartitioning.