COMP 9313 Project -1 Report

Name: Nagendra Reddy

Zid: Z5249335

1)Implementation details of your `c2lsh()`. Explain how your major transform function works.

I used only Filter , Count and Keys rdd functions in my final submission. I tried different algorithms to optimize but could find pseudocode revolves around the offset value. Rather than incrementing offset with 1 , I tried with max(beat_n) value and found Binary search implementation to find Offset in less steps. For this , we need to initialize Offset =1 and multiply by 2 if our length of candidates set obtained is less than beta_n. So now offset increments in multiples of 2 like 2,4,8,16,32…. . So at one point of offset value our candidates > beta_n , so fix this offset value as Offset_Maximum and Offset_Minimum as before offset (Which is Offset/2) . Our new Offset will be mid value of offset_Maximum and Offset_Minimum.

So now we fixed our offset search space in between offset_Maximum and Offset_Minimum.

Offset=(offset_Maximum+ Offset_Minimum)/2

Since it is in while , run the algorithm with new offset . Now we will arrive at 4 different cases

1) If candidates<beta_n : then
   offset_minimum=offset
   offset=(offset_Maximum+ Offset_Minimum)/2
2) If candidates > beta_n: then
   Offset_maximum=offset
   offset=(offset_Maximum+ Offset_Minimum)/2

we need to have stop conditions for while loop .

3) Basic stop condition when candidates is equal to beta_n return the candidate list
4) Sometimes we might not get exact Candidates , so we need to find the offset which gives atleast beta_n

Rdd Steps:

1) I am filtering datahashes rdd using Count function (Defined) . Count function returns number of collisions between 2 lists and sends the number . Filter checks whether the number is greater than or equal to Alpha_m . If "YES" it will be in rdd or else omitted .
2) Now counting total number of Candidates in RDD using rdd.count() function if rdd.count()<beta_n then offset = 2* offset_previous.
3) If we got enough Neighbours return rdd.keys()

2) Show the evaluation result of your implementation using your own test cases**.**

```
def generate2( dimension, count, seed, start=0, end=100):
    data = [
        [n] * dimension
```

```
    for n in range(start, end)
    for i in range(count)
  ]
  query = [ seed ] * dimension
  return data, query
data8, query8 = generate2( 13, 9, 100, 0, 120)
```

```
running time: 51.39911675453186
Number of candidate:  63
set of candidate:  {896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 91
7, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 873, 874, 875, 876, 877, 878, 879,
880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895}
```


```
running time: 51.39911675453186
Number of candidate:  63
set of candidate:  {896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906,
907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921,
922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 873,
874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888,
889, 890, 891, 892, 893, 894, 895}
```


3)What did you do to improve the efficiency of your implementation?

Rather than Increasing Offset by 1 . I had done by multiples of 2 and then finding Offset_Minimum and Offset_Maximum . After setting Min and Max , I used Binary Search Algorithm to find the best Offset which gives Atleast beat_N candidates.