CS246: Mining Massive Datasets

Assignment number: 1

Fill in and include this cover sheet with each of your assignments. Assignments and code are due at 5:00 PM on Scoryst and SNAP respectively. Failure to include the coversheet with you assignment will be penalized by 2 points.
Each student will have a total of *two* free late periods. *One late period expires at the start of each class.* (Assignments are due on Thursdays, which means the first late period expires on the following Tuesday at 5:00 PM.) Once these late periods are exhausted, any assignments turned in late will be penalized 50% per late period. However, no assignment will be accepted more than *one* late period after its due date. (If an assignment is due on Thursday 5 PM then we will not accept it after the following Tuesday 5 PM.)

Your name: Priyank mathur
Email: priyankm@stanford.edu **SUNetID:** priyankm

Collaborators: Dibyajyoti Ghosh

I acknowledge and accept the Honor Code.

(Signed) Priyank Mathur

Q1.

1) Source Code available towards the end of this file.

2) I solved this problem using a single map reduce job. The way I tackled this is for each line in the file, I outputted 2 sets of key value pairs from the mapper. First a list of direct relations to that user and the second was a set of distance 2 friends created from generating all pairs from the friends list of that line. For eg:

For line : 1 2,3
I generate the following pairs

```
1    1-2
1    1-3
2    2-3
```

Where the value is of the form distance-user2

In the reducer, for each user I kept track of others users distance to it and counted the number of people at distance of 2 from it. I sorted this counted list and outputted users who had most number of mutual friends. [1]

Sample output –

```
0    38737,18591,27383,34211,337,352,1532,12143,12561,17880
1    35621,44891,14150,15356,35630,13801,13889,14078,25228,13805
2    41087,1,5,95,112,1085,1404,2411,3233,4875
3    27679,1,10,16,29,30,38,82,83,85
```

3)

```
924         439,2409,6995,11860,15416,43748,45881
8941        8943,8944,8940
8942        8939,8940,8943,8944
9019        9022,317,9023
9020        9021,9016,9017,9022,317,9023
9021        9020,9016,9017,9022,317,9023
9022        9019,9020,9021,317,9016,9017,9023
9990        13134,13478,13877,34299,34485,34642,37941
9992        9987,9989,35667,9991
9993        9991,13134,13478,13877,34299,34485,34642,37941
```

[1] Cems.uwe.ac.uk, (2014). Paul Matthews Blog @ UWE | Teaching MapReduce. [online] Available at: http://www.cems.uwe.ac.uk/~pmatthew/blog/2014/03/04/teaching-mapreduce [Accessed 21 Jan. 2015].

Q2) a) $\text{Conf}(A \to B) = Pr(B/A)$

$$= \frac{Pr(B \wedge A)}{P(A)}$$

which ignores $P(B)$. In case $P(B)$ is very high or 1, ie, B occurs in every bucket, this rule does not have any relevance even though it has high confidence.

$$\text{Lift}(A \to B) = \frac{\text{Conf}(A \to B)}{S(B)} = \frac{Pr(A \wedge B)}{Pr(A)\, Pr(B)}$$

Lift does not suffer from this it takes $P(B)$ into consideration and ~~would~~ would hence be lower for very high $P(B)$.

$$\text{Conv}(A \to B) = \frac{1 - S(B)}{1 - \text{Conf}(A \to B)} = \frac{1 - Pr(B)}{1 - \text{conf}(A \to B)}$$

This is also safe from this drawback, since it has a $P(B)$ factor in the numerator. What this causes is that the metric goes up as confidence increases but goes down in case $P(B)$ itself is too high.

Q2) b) :- $\text{conf}(A \to B) = \dfrac{\Pr(A \wedge B)}{P(A)}$.

$$\text{conf}(B \to A) = \frac{\Pr(A \wedge B)}{P(B)}$$

~~The~~ [illegible] [illegible] [illegible] [illegible]

conf (A → B) will be equal to conf (B → A) only in case P(A) = P(B). Since this may not be always true, confidence is not symmetric.

———— x ————

$$\text{Lift}(A \to B) = \frac{\Pr(A \wedge B)}{\Pr(A)\Pr(B)}$$

$$\text{Lift}(B \to A) = \frac{\Pr(B \wedge A)}{\Pr(B)\ \Pr(A)}$$

As we can see, Lift (A → B) will always be equal to lift (B → A) hence it is symmetrical.

———— x ————

$$\text{conv}(A \to B) = \frac{1 - \Pr(B)}{1 - \text{conf}(A \to B)} = \frac{1 - \Pr(B)}{1 - \frac{\Pr(A \wedge B)}{\Pr(A)}}$$

$$= \frac{\Pr(A) - \Pr(B)\Pr(A)}{\Pr(A) - \Pr(A \wedge B)}$$

contd.

Similarly,

$$\text{Conv}(B \to A) = \frac{Pr(B) - Pr(B)\,Pr(A)}{Pr(B) - Pr(A \wedge B)}$$

Hence, Conv $(A \to B)$ will be equal to Conv $(B \to A)$ only in case $P(A) = P(B)$. Since that may not always be true, Conviction in general is not symmetric.

Q2) c). $\text{conf}(A \to B) = \frac{Pr(A \wedge B)}{Pr(A)}$

In case $A \to B$ is a perfect implication, B is always in a basket that contains A.
i.e. $P(A \wedge B) = P(A)$.

$\Rightarrow \text{Conf}(A \to B) = \frac{Pr(A \wedge B)}{Pr(A)} = \frac{Pr(A)}{Pr(A)} = 1$

which is maximal since a probability can not be > 1.

Hence conf is desirable with max. of 1.

$\text{Lift}(A \to B) = \frac{\text{Conf}(A \to B)}{Pr(B)} = \frac{Pr(A \wedge B)}{Pr(A)\,Pr(B)}$

Similar to above, in case of perfect implication, $\text{conf}(A \to B) = 1$.

$\Rightarrow \text{Lift}(A \to B) = \frac{1}{Pr(B)}$.

For a given dataset, $Pr(B)$ can be considered constant. Thus the numerator is maximum when the rule is a perfect implication.

Therefore, lift is also desirable with maximum value of $\frac{1}{Pr(B)}$.

$$\text{conv}(A \rightarrow B) = \frac{1 - Pr(B)}{1 - \text{conf}(A \rightarrow B)}$$

As we saw, in case of perfect implication

$\text{conf}(A \rightarrow B) = 1$

$\Rightarrow$ as $\text{conf}(A \rightarrow B) \rightarrow 1$

$\text{conv}(A \rightarrow B) \rightarrow \infty$.

As conf approaches 1, conv. approaches ∞.

Hence, with increase in conf. the conv is higher.

Conviction is also desiralse with a max val of ∞.

☆ A special case here is when $P(B) = 1$. $\Rightarrow \text{conf}(A \rightarrow B) = 1$.

in which case it becomes undefined.

Q2.

d)

```
Rule, Confidence
'DAI93865 -> FRO40251', 1.0
'GRO85051 -> FRO40251', 0.999176276771005
'GRO38636 -> FRO40251', 0.9906542056074766
'ELE12951 -> FRO40251', 0.9905660377358491
'DAI88079 -> FRO40251', 0.9867256637168141
```

e)

```
Rule, Confidence
'DAI23334,ELE92920 -> DAI62779', 1.0
'DAI31081,GRO85051 -> FRO40251', 1.0
'DAI55911,GRO85051 -> FRO40251', 1.0
'DAI62779,DAI88079 -> FRO40251', 1.0
'DAI75645,GRO85051 -> FRO40251', 1.0
```

Source Code available towards the end of this file.

(Q3) a) To prove:

$$d(x,y) + d(y,z) \geq d(x,z) \quad \forall (x,y,z)$$

where,

$$d(x,y) = 1 - sim(x,y)$$
$$= 1 - Pr[h(x) = h(y)]$$
$$= Pr[h(x) \neq h(y)] \text{ —— } (A)$$

We can make the following Observations-

(1) The event $h(x) \neq h(y)$ $\forall (x,y)$ is a binary event and can take values true or false.

(2) $h(x) \neq h(y)$ implies that ~~both~~ ~~either~~ ~~[illegible]~~ one of the following must hold

- $h(x) \neq h(z)$
- $h(y) \neq h(z)$ [1]

Because ~~if~~ both do not hold, by trasitivity property of equality, $h(x) = h(z) = h(y)$ which is against our assumption above.

[1] https://class.coursera.org/mmds-001/lecture/43

Hence,

We can say that for event $h(x) \neq h(y)$ to occur, one of $h(x) \neq h(z)$ or $h(y) \neq h(z)$ must occur.

Therefore,

$$Pr[h(x) \neq h(y)] \leq Pr[h(x) \neq h(z)] + Pr[h(y) \neq h(z)]$$

from equation Ⓐ

$$\boxed{d(x,y) \leq d(x,z) + d(y,z)}$$

Q3) b) $\quad Sim_{over}(A,B) = \dfrac{|A \cap B|}{\min(|A|,|B|)}$

Assume $A = \{1,2,3\}$

$B = \{1\}$

$C = \{2\}$

$Sim_o(A,B) = 1/1 = 1$

$Sim_o(A,C) = 1/1 = 1$

$Sim_o(B,C) = 0/1 = 0$

$\Rightarrow \quad d(A,B) = 1-1 = 0$

$d(A,C) = 1-1 = 0$

$d(B,C) = 1-0 = 1$

These distances do not obey the triangle inequality as

$d(A,B) + d(A,C) \ngeq d(B,C)$

Hence, there is no LSH scheme for overlap similarity.

Q3.) 'c) $$\text{Sim}_{Dice}(A, B) = \frac{|A \wedge B|}{\frac{1}{2}(|A| + |B|)}$$

$$= \frac{2\,|A \wedge B|}{|A| + |B|}$$

Assume $A = \{1, 2\}$

$B = \{1\}$

$C = \{2\}$

$$\text{Sim}_D(A, B) = \frac{2 \times 1}{3} = 2/3$$

$$\text{Sim}_D(A, C) = \frac{2 \times 1}{3} = 2/3$$

$$\text{Sim}_D(B, C) = 0/3 = 0$$

$\Rightarrow$ $d(A, B) = 1 - 2/3 = 1/3$

$d(A, C) = 1 - 2/3 = 1/3$

$d(B, C) = 1 - 0 = 1$

These distances do not obey the triangle inequality as

$$d(A,B) + d(A,C) \not\geq d(B,C)$$

Hence this similarity does not have a LSH scheme.

Q4.) a) $W_j = \{x \in A : g_j(x) = g_j(z)\} \quad (1 \leq i \leq L)$

That is W_j is the set of elements in the same bucket as z hashed by g_j

Also, $T = \{x \in A : d(x, z) > c\lambda\}$.

$$\Rightarrow \Pr[g(x) = g(z)]$$

$$= \left[\Pr[h(x) = h(z)]\right]^k$$

$\because$ H is $(\lambda, c\lambda, p_1, p_2)$-sensitive

$$[\Pr[h(x) = h(z)]]^k \leq p_2^k$$

$$\leq p_2^{\lceil \log_{1/p_2} n \rceil}$$

$$\leq n^{\log_{1/p_2} p_2}$$

$$\leq n^{-\log_{1/p_2} 1/p_2}$$

$$\leq n^{-1}$$

$$\Rightarrow \Pr(g(x) = g(z)) \leq 1/n. \quad — \textcircled{A}$$

To prove

$$\Pr\left[\sum_{1}^{L} |T \cap W_j| \geq 3L\right] \leq 1/3$$

<u>Using markov inequality on LHS.</u>

$$Pr[\sum_1^L |T \cap w_j| \geq 3L] \leq \frac{E[\sum_1^L |T \cap w_j|]}{3L} \quad — \textcircled{B}$$

LHS represents the probability that all the 3L points that we gather from L buckets are greater than $c\lambda$ from the query point, ie, an error condition.

From $\textcircled{A}$, we know than for pts (x,z) such that $d(x,z) \geq c\lambda$

$$Pr(g(x)=g(z)) \leq 1/n$$

suppose t elements from T fall into w_j for any j

$\Rightarrow Pr[\text{having } t \text{ elements from } T \text{ in } w_j] \leq 1/n^t$ — $\textcircled{C}$

From equation $\textcircled{B}$,

$$\frac{E[\sum_1^L |T \cap w_j|]}{3L}$$

$$= \frac{E[|T \cap w_1|] + E[|T \cap w_2|] + \cdots + E[|T \cap w_L|]}{3L}$$

From $\textcircled{C}$

$$\leq \frac{1+1 \cdots 1}{3L} \leq \frac{1k}{3k} \leq 1/3$$

Q4) b). $x^* \in A$: $d(x^*, z) \leq \lambda$

To prove –

$$Pr[\forall 1 \leq j \leq L, g(x^*) \neq g(z)] \leq 1/e.$$

$$Pr[\forall 1 \leq j \leq L, g(x^*) \neq g(z)]$$

$$= [Pr[g(x^*) \neq g(z)]]^L$$

(x^* & z do not hash to any of the buckets)

$$= [1 - Pr[g(x^*) = g(z)]]^L$$

∵ $g \in G$ is an and construct for $h \in H^k$, we get

$$= [1 - [Pr[h(x^*) = h(z)]]^k]^L \text{ —— } (A)$$

Also, H is a family with $(\lambda, c\lambda, p_1, p_2)$ sensitive

$$Pr[h(x^*) = h(z)] \geq p_1$$

$$\Rightarrow 1 - Pr[h(x^*) = h(z)] \leq 1 - p_1$$

(A) becomes –

$$[1 - [Pr[h(x^*) = h(z)]]^k]^L \leq$$

$$[1 - p_1^k]^L \text{ —— } (B)$$

We know.

$$k = \log_{1/p_2} n$$

Hence,

$$p_1^k = p_1^{[\log_{1/p_2} n]}$$

$$= n^{[\log_{1/p_2} p_1]} \quad \text{(base shift)}$$

$$= n^{\left[-\frac{\log_e 1/p_1}{\log_e 1/p_2}\right]} \quad \text{(base change)}$$

$$= n^{-e} \quad \text{where } e = \frac{\log 1/p_1}{\log 1/p_2}.$$

Equation Ⓑ becomes

$$\Rightarrow [1 - n^{-e}]^L$$

Since, $\forall x \in R$, $(1 - x/n)^n \leq e^{-x}$

$$\Rightarrow (1-x)^1 \leq e^{-x}$$

$$\Rightarrow [1 - n^{-e}] \leq e^{-n^{-e}}$$

$$\Rightarrow [1 - n^{-e}]^L \leq [e^{-n^{-e}}]^L$$

$$\leq e^{-L/n^e}$$

$$\because L = n^e$$

$$\Rightarrow [1 - n^{-e}]^L \leq e^{-1} \leq 1/e$$

$$\Rightarrow \boxed{P_r[\forall 1 \leq j \leq L, g(x^*) \neq g(z)] \leq 1/e}$$

Q4) c) ~~fco~~ To prove :

Point chosen is $c\lambda$- ANN

That is, the pt chosen (x) is such that

$d(x,z) \leq c\lambda$, where z is the query.

We know that x is a point chosen uniformly from L buckets and is among the total of $3L$. In case the total of L buckets $> 3L$, then from part 4@ we know the probability

Pr [Choosing $3L$ from L buckets where all ~~eee~~ points are greater than $c\lambda$ dist] $\leq 1/3$ — (A).

Also, from 4(b) we know that for a pt.

$x^* \in A : d(x^*, z) \leq \lambda$

$pr[g_j(x^*) \neq g_j(z), 1 \leq j \leq L] \leq 1/e$.

Suppose there are q pts. that are within λ distance from z.

Pr[none of q pts map to same bucket as z] $\leq 1/e^q$ ——— (B)

~~From (A), (B)~~

~~Pr[point cho~~

from Ⓐ &Ⓑ equations

$\Pr[\text{point chosen has dist} \geq c\lambda] \leq \frac{1}{3} + \frac{1}{e^{a}}$

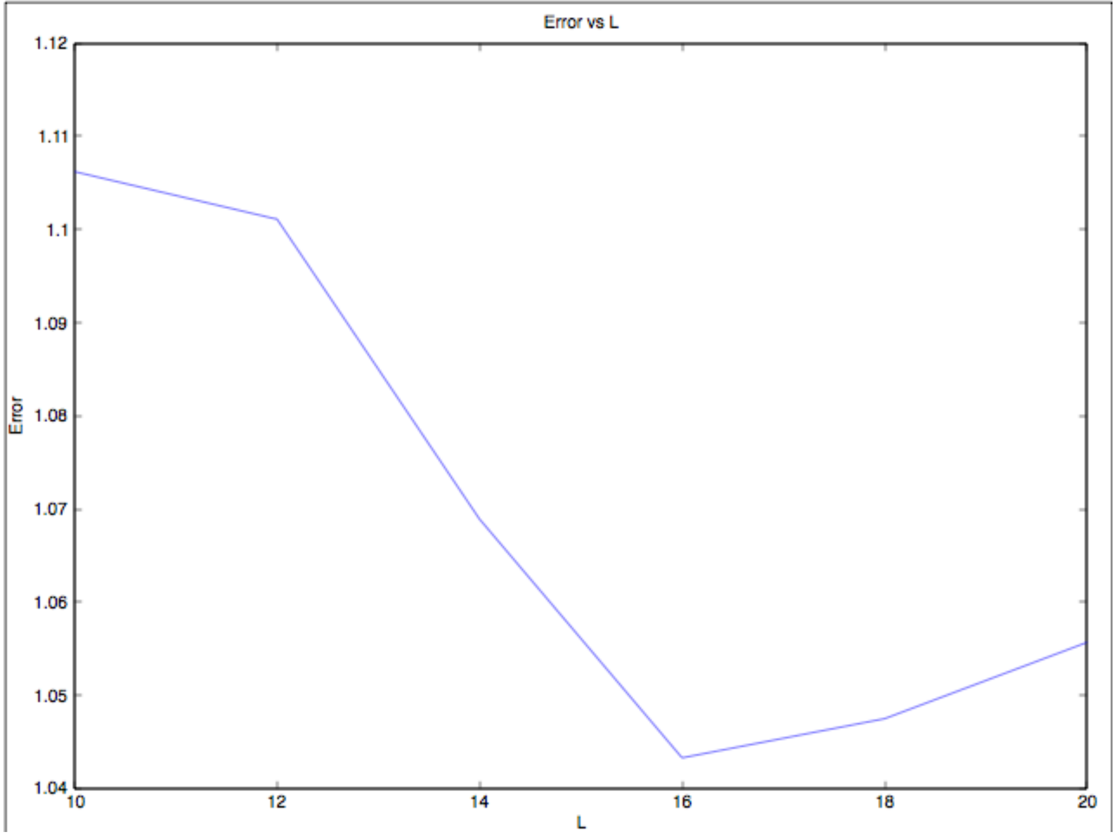
$\Pr[\text{point chosen is } (c,\lambda)\text{-ANN}] \geq 1 - \frac{1}{3} - \frac{1}{e^{a}}$

$\geq \frac{2}{3} - \frac{1}{e^{a}}$

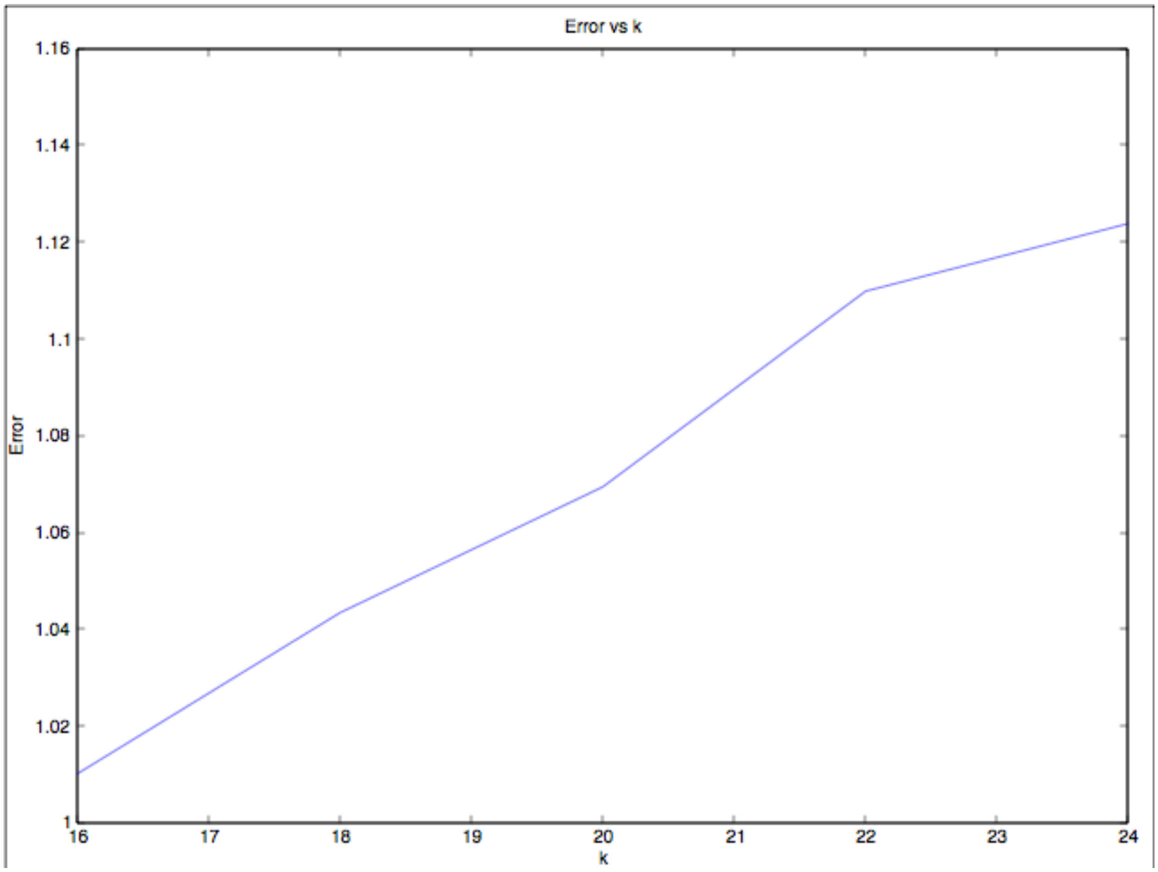
Q4.
d)
1)
Average search time for LSH - 0.019817
Average search time for Linear search - 0.25485

We see significant improvement in run time of LSH, which is approximately 10 times faster than linear search.

2)



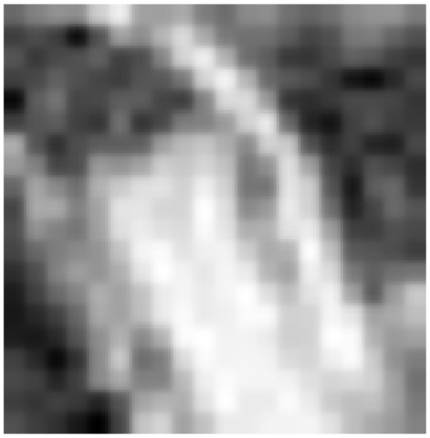
We notice from the above graph that increasing L (number of tables) generally should decrease the error of the approximate search of LSH. This may be because we are increasing the number of terms in the OR construct (buckets) and hence increasing the probability of similar things hashing together.



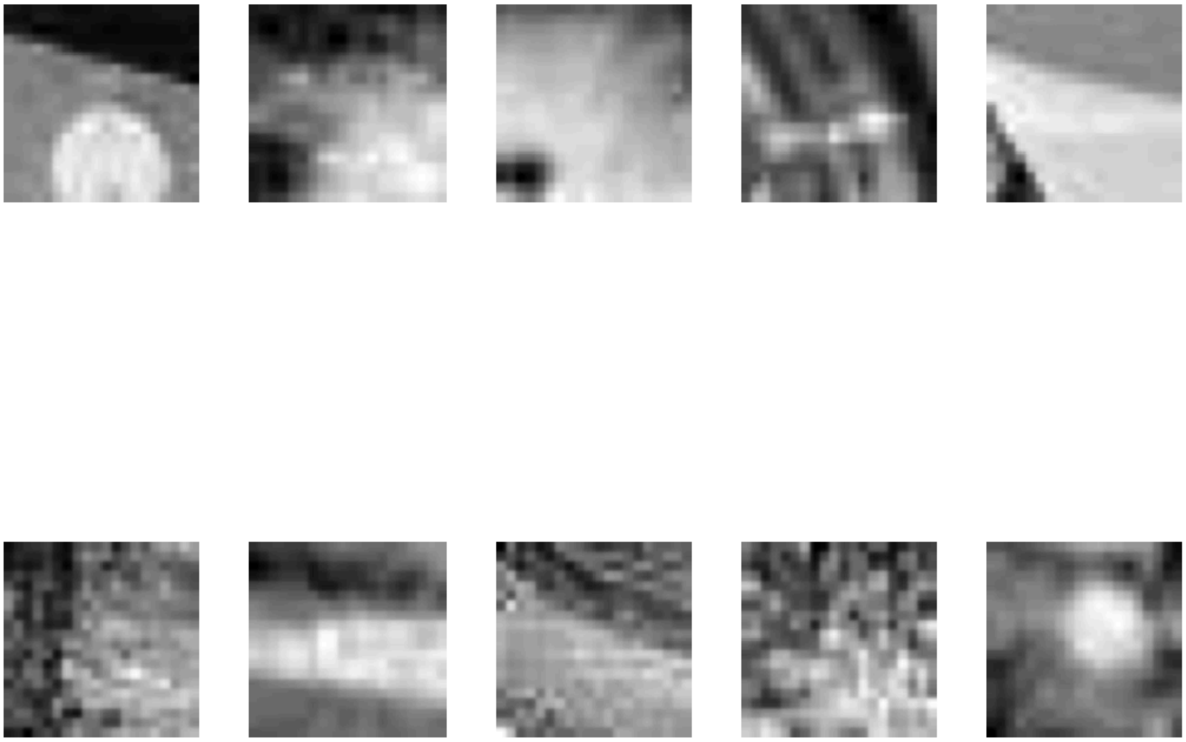
We notice from the above graph that increasing k (key length) increases the error of the search by LSH. This may be because we are increasing the number of terms in the AND construct (rows) and hence decreasing the probability of similar things hashing together.

3)

Query point (column 100) –

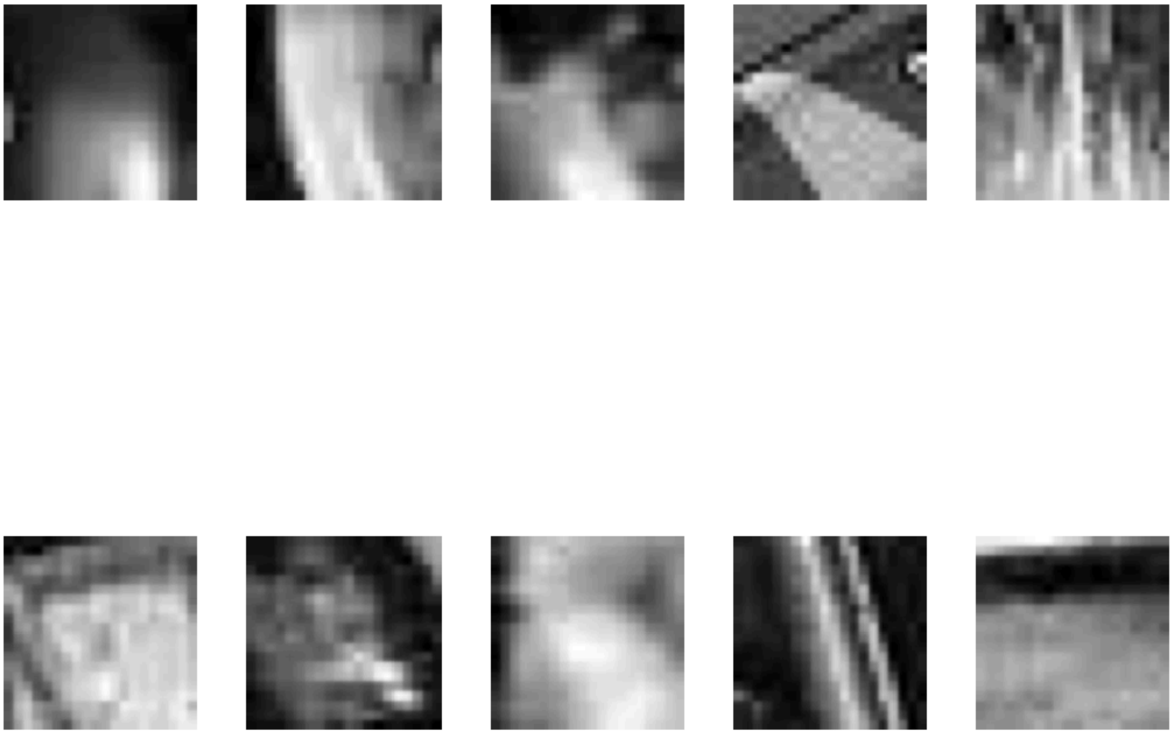


LSH –



The nearest neighbors found look to be similar to query image above but some options towards the end (towards bottom and right) appear to further diverge away from the query.

Linear –



The result of linear search is much better than LSH. The query image contains a bright-elongated spot in the middle and is darker towards the edges. Most of the neighbors found by linear search have similar appearance. In contrast, some of the neighbors produced by LSH (eg. number 3 and 5 in top row) are different from this pattern.

We can conclude from this experiment that even though LSH is significantly faster, it might not produce very accurate results.

Q1 Source code –

```
package edu.stanford.cs246.friendrecommender;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.PriorityQueue;
import java.util.Stack;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class FriendRecommender extends Configured implements Tool {
  public static void main(String[] args) throws Exception {
    System.out.println(Arrays.toString(args));
    int res = ToolRunner.run(new Configuration(), new FriendRecommender(), args);

    System.exit(res);
  }

  @Override
  public int run(String[] args) throws Exception {
    System.out.println(Arrays.toString(args));
    Job job = new Job(getConf(), "FriendRecommender");
    job.setJarByClass(FriendRecommender.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(Text.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);

    return 0;
  }

  public static class Map extends Mapper<LongWritable, Text, IntWritable, Text> {
          Text dist1Text = new Text();
          Text dist2Text = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException {


          // Get the information from the line
          String line = value.toString();
          String[] users = line.split("\\t");
          String friendString = users.length > 1 ? users[1] : "";
          String[] friends = friendString.split(",");

          /* Loop through friends list to generate pairs of following types -
           * For line : 1   2,3
           * we generate the following pairs
           * 1     1-2
           * 1     1-3
           * 2     2-3
           *
           * Where the value is of the form distance-user2
           */
          for (int i=0; !friendString.equals("") && i<friends.length; i++) {
                  String dist1Str = "1-" + friends[i];
                  dist1Text.set(dist1Str);
                  context.write(new IntWritable(Integer.parseInt(users[0])), dist1Text);

                  // Generate pairs with distance 2
                  for (int j=i+1; j<friends.length; j++) {
                          String dist2Str = "2-" + friends[j];
                          dist2Text.set(dist2Str);
                          context.write(new IntWritable(Integer.parseInt(friends[i])), dist2Text);

                          dist2Str = "2-" + friends[i];
                          dist2Text.set(dist2Str);
                          context.write(new IntWritable(Integer.parseInt(friends[j])), dist2Text);
                  }
          }
    }
  }

  public static class Reduce extends Reducer<IntWritable, Text, IntWritable, Text> {

          /*
           * For each user user1, keep track of how many common friends does he
           * have with another user user2. Then remove direct relations and
           * output the distance 2 users sorted by the number of mutual friends.
           */
```

```
    @Override
    public void reduce(IntWritable key, Iterable<Text> values, Context context)
          throws IOException, InterruptedException {
      HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

      for (Text val : values) {
        String association = val.toString();
        int length = Integer.parseInt(association.split("-")[0]);
        int friend = Integer.parseInt(association.split("-")[1]);

        // Keep track of other users and their distance
        if (length == 1)
         map.put(friend, 0);
        else if (length == 2) {
         if (map.containsKey(friend)) {
                 if (map.get(friend)!=0)
                         map.put(friend, map.get(friend)+1);
         }
         else {
                 map.put(friend, 1);
         }
        }
      }

      PriorityQueue<Entry<Integer, Integer>> queue = new PriorityQueue<Entry<Integer, Integer>>(10, new
EntryComparator());

      // Put them in a priority queue to get the 10 users
      // with most number of mutual friends
      for (Entry<Integer, Integer> e: map.entrySet()) {
          if (e.getValue()==0)
                  continue;

          if (queue.size() < 10)
                  queue.add(e);
          else {
                  int tempValue = ((Entry<Integer, Integer>) queue.peek()).getValue();
                  int tempKey = ((Entry<Integer, Integer>) queue.peek()).getKey();

                  if ((tempValue < e.getValue()) || (tempValue == e.getValue() && tempKey > e.getKey())){
                          queue.poll();
                          queue.add(e);
                  }
          }
      }

      String recommendations = "";
      Stack<Entry<Integer, Integer>> reverser = new Stack<Entry<Integer, Integer>>();

      // Reverse the order
      while (queue.size() > 0) {
          reverser.push(queue.poll());
      }

      if (reverser.size() > 0) {
          Entry<Integer, Integer> element = reverser.pop();
          recommendations += element.getKey();
        }

        while (reverser.size() > 0) {
          Entry<Integer, Integer> element = reverser.pop();
          recommendations += "," + element.getKey();
        }

        Text recommendationsText = new Text(recommendations);

        // Write out the output
        context.write(key, recommendationsText);
      }
    }
}

/*
 * Comparator to sort the entries in the priority queue
 * first by value and then by ascending numerical order
 * of the keys
 */
class EntryComparator implements Comparator<Entry<Integer, Integer>>
{
  public int compare(Entry<Integer, Integer> e1, Entry<Integer, Integer> e2)
  {
        if (e1.getValue() != e2.getValue())
                return e1.getValue().compareTo(e2.getValue());
        else
                return e2.getKey().compareTo(e1.getKey());
  }
}
```

<u>Q2 Source Code</u> –

```
from collections import defaultdict
from itertools import chain, combinations

# Globals
data_file = 'browsing.txt'
support = 100
item_counter = 1

def getNewId():
    global item_counter
    retVal = item_counter
    item_counter += 1
    return retVal

# Containers
name_hash = defaultdict(getNewId)
id_hash = {}
C1_ctr = defaultdict(int)
C2_ctr = defaultdict(int)
C3_ctr = defaultdict(int)
R2_conf = defaultdict(int)
R3_conf = defaultdict(int)

def readFileGenerator(filename=data_file):
    '''
    Return lines of data
    '''
    file_obj = open(data_file)
    for line in file_obj:
        yield line

def count_singletons():
    '''
    Count single items
    '''
    for line in readFileGenerator():
        items = set(line.strip().split(" "))
        for item in items:
            key = name_hash[item] # maintain name to id mapping
            id_hash[key] = item # maintain id to name mapping
            C1_ctr[key] = C1_ctr[key] + 1

def isFrequentItem(item):
    '''
    Check if an item has support > min support
    '''
    key = name_hash[item] if item in name_hash else None
    return C1_ctr[key] > support if key else False

def isFrequentPair(pair):
    '''
    Check if a pair has support > min support
    '''
    key1 = name_hash[pair[0]] if pair[0] in name_hash else None
    key2 = name_hash[pair[1]] if pair[1] in name_hash else None
    if key1==None or key2==None:
        return False

    dict_key = tuple(sorted((key1, key2)))

    return C2_ctr[dict_key] > support if dict_key in C2_ctr else False

def prune_storage(storage):
    '''
    Remove counts of items where support < min support
    '''
    remove_keys = [key for key in storage.keys() if storage[key] < support]
    for k in remove_keys: del storage[k]

def prune_items(items, k=2):
    '''
    Returns a list of items to be used for creation of
    higher order item sets
    '''

    freq_items = {item:0 for item in items if isFrequentItem(item)}

    if k == 3:
        candidate_sets = combinations(freq_items, 2)
        freq_candidate_sets = [pair for pair in candidate_sets if isFrequentPair(pair)]

        '''
        Check each of the items appears in at least 2 freq pairs
        '''
        for it in freq_items:
            for c in freq_candidate_sets:
                if it in c:
                    freq_items[it] = freq_items[it] + 1

        freq_items = {item:cnt for item, cnt in freq_items.items() if cnt >= 2}

    return freq_items

def makePassK(k=2):
    '''
    Create and filter item sets of size k
    '''
    cnt = 0
    storage = C2_ctr if k==2 else C3_ctr
    storage.clear()

    for line in readFileGenerator():
        items = set(line.strip().split(" "))

        freq_items = prune_items(items, k)
        #print freq_items

        candidate_sets = combinations(freq_items, k)

        for c in candidate_sets:
            key1 = name_hash[c[0]]
            key2 = name_hash[c[1]]
            key3 = name_hash[c[2]] if k==3 else None
            dict_key = (key1, key2, key3) if k==3 else (key1, key2)
            dict_key = tuple(sorted(dict_key))
            storage[dict_key] = storage[dict_key] + 1

    prune_storage(storage)

def get3Rules(k):
    '''
    Generate rules from triples of form
    item1, item2 -> item3
    '''
    ks = set(k)
    for t in combinations(k, 2):
        C2_key = tuple(sorted(t))
        #print C2_key

        n1 = id_hash[t[0]]
        n2 = id_hash[t[1]]
        srt = sorted([n1, n2])
        lhs = srt[0] + "," + srt[1]
        ts = set(t)
        rhs = ks.difference(ts)
        rhs = rhs.pop()
        representation = lhs + " -> " + id_hash[rhs]

        num = C3_ctr[k] * 1.0 if k in C3_ctr else None
        denom = C2_ctr[C2_key] if C2_key in C2_ctr else None
        #print num , denom

        if num==None or denom==None:
            yield (representation, 0)
        else:
            conf = num / denom
            yield (representation, conf)

def get2Rules(k):
    '''
    Generate rules from pairs of form
    item1 -> item2
    '''
    ks = set(k)
    for t in combinations(k, 1):
        C1_key = t[0]
        #print C1_key

        lhs = str(C1_key)
        ts = set(t)
        rhs = ks.difference(ts)
        rhs = rhs.pop()
        representation = id_hash[C1_key] + " -> " + id_hash[rhs]

        num = C2_ctr[k] * 1.0 if k in C2_ctr else None
        denom = C1_ctr[C1_key] if C1_key in C1_ctr else None
        #print num , denom

        if num==None or denom==None:
            yield (representation, 0)
        else:
            conf = num / denom
            yield (representation, conf)

def getRules(k):
    '''
    Create and store rules of length k
    '''
    if k==3:
        for triplet in C3_ctr:
            #print triplet
            for rule in get3Rules(triplet):
                R3_conf[rule[0]] = rule[1]
    elif k==2:
        for pair in C2_ctr:
            for rule in get2Rules(pair):
                R2_conf[rule[0]] = rule[1]

# Execute commands
count_singletons()
makePassK(2)
makePassK(3)
getRules(2)
getRules(3)

# Display results
#sorted(R2_conf.items(), key=lambda x: (x[1],x[0]), reverse=True )[:20]
#sorted(R3_conf.items(), key=lambda x: (x[1],x[0]), reverse=True )[:20]
```

<u>Q4 Source Code</u> –

```
%% One time initialization
load patches;

% Perform linear search and return nearest neighbors
function nnlinind = linearsearch(query, data, num)

    d = sum(abs(bsxfun(@minus, query, data)));
    [ignore,ind]=sort(d);

    cand = ind(1:num+1);

    nnlinind = cand;

end


%% Calculate error ratio
function [err] = calcerror(linn, lshnn, data)
    err=0;

    for i=1:10,
        qcol = data(:, lshnn(i, 1));

        lshids = lshnn(i, 2:4);
        lshids = lshids(lshids ~= 0);
        lshdcols = data(:, lshids);

        linids = linn(i, 2:4);
        linids = linids(linids ~= 0);
        lindcols = data(:, linids);

        lshdist = sum(sum(abs(bsxfun(@minus, lshdcols, qcol))));
        lindist = sum(sum(abs(bsxfun(@minus, lindcols, qcol))));

        % disp(sprintf('%s%d%s%d%s%f', 'lshdist-', lshdist, '; lindist-', lindist, '; error-',
lshdist/lindist));
        err = err + (lshdist * 1.0 /lindist);
    end

    err /= 10;
end


%% driver function to calulate error
function [errors] = driver(data)
    errors = [];

    % Vary L and get nearest neighbors
    for i=10:2:20,
        T1=lsh('lsh',i,24,size(data,1),data,'range',255);
        linn = findnn(data, 'linearsearch', T1, 3);
        lshnn = findnn(data, 'lsh', T1, 4);
        err = calcerror(linn, lshnn, data);
        errors = [errors err];
    end

    % Vary k and get nearest neighbors
    for i=16:2:24,
        T1=lsh('lsh',10,i,size(data,1),data,'range',255);
        linn = findnn(data, 'linearsearch', T1, 3);
        lshnn = findnn(data, 'lsh', T1, 4);
        err = calcerror(linn, lshnn, data);
        errors = [errors err];
    end

end

% find nearest neighbors based on search type
function op = findnn(patches, searchtype, T1, num)
    neigbours = [];
    disp(searchtype);
    tic;

    for i=1:10,
        colno = i*100;
        query = patches(:, colno);

        if strcmp(searchtype, 'linearsearch'),
            nn=linearsearch(query, patches, num);
        elseif strcmp(searchtype, 'lsh'),
            [nn,numcand]=lshlookup(query, patches, T1, 'k', num, 'distfun', 'lpnorm', 'distargs',
{1});

            if numcand < num,
                pads = zeros(1, num-numcand);
                disp('-- Padding --');
                nn = [nn pads]
            end
        end

        disp(sprintf('%s%d%s%d', 'neigbours-', size(neigbours,2), '; numcand-', size(nn,2)));

        neigbours = [neigbours; nn];
    end

    op = neigbours;
    time = toc;

    disp(time/10);
end

%% plot nearest neighbors
function plotnn(data, query, nn, numcand)

    % plot the query point
    figure(1); clf;
    imagesc(reshape(query,20,20));
    colormap gray;
    axis image;
```

```
        set(gca,'YTickLabel', sprintf('',[]));
        set(gca,'XTickLabel', sprintf('',[]));

        % plot neigbours
        figure(2);clf;
        for k=1:numcand,
                subplot(2,5,k);
                imagesc(reshape(data(:,nn(k+1)),20,20));
                colormap gray;
                axis image;
                set(gca,'YTickLabel', sprintf('',[]));
                set(gca,'XTickLabel', sprintf('',[]));
        end

end

% Plot the 10 nearest neighbors for both types
query = patches(:, 100);
T1=lsh('lsh',10,24,size(patches,1),patches,'range',255);
[nn,numcand]=lshlookup(query, patches, T1, 'k', 11, 'distfun', 'lpnorm', 'distargs', {1});
plotnn(patches, query, nn, 10);

nn=linearsearch(query, patches, 10);
plotnn(patches, query, nn, 10);
```