**Source Code**

**The Same Code Works for Any Number of Users**

**HOMOMORPHIC.JAVA**

package com;

import java.math.BigInteger;

import java.util.Random;

public class Homomorphic {

static BigInteger p, q, lambda;

static BigInteger n;

static BigInteger nsquare;

static BigInteger g;

public static void KeyGeneration() {

p = new BigInteger("93955840068124884581126558102244121462213454272523424920404404150482245212871");

q = new BigInteger("68377774710584922465934285797917633639686254055898437333934414867248135853033");

n = new BigInteger("5209935468252426244906168976236492798472480324896944574739117200686986142896826797709880496714150297318988022948809468388693792207983833619963001111963867");

nsquare = new BigInteger("27143427583354627916845127171708935172399251412421917711184570478004519747428903887003817340194794421083792795384699640411328032870422726995099527579849510837767346279084881658584935665466683182030013040542601142703761446581044190740521565737147778553839441192124818060868025680465180077332990305641513593689");

lambda = new BigInteger("2604967734126213122453084488118246399236240162448472287369558600343493071448339684331275876461635492436521203654054740156535006447279020437357737405447280");

g = new BigInteger("2");

}

public static BigInteger Encryption(BigInteger m) {

Random rand = new Random(2);

BigInteger r = new BigInteger(512, rand);

return g.modPow(m, nsquare).multiply(r.modPow(n, nsquare)).mod(nsquare);

}

public static BigInteger Decryption(BigInteger c) {

BigInteger u = g.modPow(lambda, nsquare).subtract(BigInteger.ONE).divide(n).modInverse(n);

return c.modPow(lambda, nsquare).subtract(BigInteger.ONE).divide(n).multiply(u).mod(n);

}

}

**K-CLUSTER.JAVA**

package com;

import java.io.File;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.TreeSet;

import java.util.HashSet;

import java.util.Arrays;

import java.util.TreeMap;

import java.util.Iterator;

import java.util.Set;

import java.util.Map;

public class KCluster{

static ArrayList<File> files = new ArrayList<File>();

    static ArrayList<double[]> vector = new ArrayList<double[]>();

public static double distance(double[] vector1, double[] vector2){

double dot = 0, magnitude1 = 0, magnitude2=0;

for(int i=0;i<vector1.length;i++){

dot+=vector1[i]\*vector2[i];

magnitude1+=Math.pow(vector1[i],2);

    magnitude2+=Math.pow(vector2[i],2);

}

magnitude1 = Math.sqrt(magnitude1);

    magnitude2 = Math.sqrt(magnitude2);

    double d = dot / (magnitude1 \* magnitude2);

    return d == Double.NaN ? 0 : d;

}

public static void kcluster(){

HashMap<double[],TreeSet<Integer>> clusters = new HashMap<double[],TreeSet<Integer>>();

HashMap<double[],TreeSet<Integer>> step = new HashMap<double[],TreeSet<Integer>>();

HashSet<Integer> random = new HashSet<Integer>();

TreeMap<Double,HashMap<double[],TreeSet<Integer>>> clusters\_sim = new TreeMap<Double,HashMap<double[],TreeSet<Integer>>>();

int k = 2;

int maxiter = 20;

for(int i=0;i<20;i++){

clusters.clear();

    step.clear();

    random.clear();

while(random.size() < k){

random.add((int)(Math.random()\*vector.size()));

}

for(int r : random){

double[] temparray = new double[vector.get(r).length];

System.arraycopy(vector.get(r),0,temparray,0,temparray.length);

step.put(temparray,new TreeSet<Integer>());

}

boolean flag = true;

int iter = 0;

while(flag){

clusters = new HashMap<double[],TreeSet<Integer>>(step);

for(int p=0;p<vector.size();p++){

double[] centroid = null;

double similarity = 0;

for(double[] cent : clusters.keySet()){

double csimilarity = distance(vector.get(p),cent);

if(csimilarity > similarity){

similarity = csimilarity;

centroid = cent;

}

}

if(clusters.get(centroid) != null)

clusters.get(centroid).add(p);

}

    step.clear();

    for(double[] centroid : clusters.keySet()){

double[] change\_centroid = new double[centroid.length];

for(int d : clusters.get(centroid)){

double[] doc = vector.get(d);

for(int p=0;p<change\_centroid.length;p++)

change\_centroid[p]+=doc[p];

}

for(int p=0;p<change\_centroid.length;p++){

change\_centroid[p]/=clusters.get(centroid).size();

}

    step.put(change\_centroid,new TreeSet<Integer>());

}

//check break conditions

    String oldcentroid = "", newcentroid="";

for(double[] d : clusters.keySet())

oldcentroid+=Arrays.toString(d);

for(double[] d: step.keySet())

newcentroid+=Arrays.toString(d);

    if(oldcentroid.equals(newcentroid))

flag = false;

if(++iter >= maxiter)

flag = false;

    }

double sumsim = 0;

    for(double[] cent : clusters.keySet()){

TreeSet<Integer> cls = clusters.get(cent);

for(int value : cls){

    sumsim+=distance(cent,vector.get(value));

    }

    }

clusters\_sim.put(sumsim,new HashMap<double[],TreeSet<Integer>>(clusters));

}

for(double[] cent : clusters\_sim.get(clusters\_sim.lastKey()).keySet()){

StringBuilder sb = new StringBuilder();

for(int pts : clusters\_sim.get(clusters\_sim.lastKey()).get(cent)){

if(!exists(files.get(pts).getName()))

sb.append(files.get(pts).getName()+",");

}

if(sb.length() > 0){

sb.deleteCharAt(sb.length()-1);

MergeCluster mc = new MergeCluster();

mc.setCluster(sb.toString());

mc.setId(Recluster.map.size()+1);

Recluster.map.put(cent,mc);

}

}

}

public static boolean exists(String name){

boolean flag = false;

for(Map.Entry<double[],MergeCluster> me : Recluster.map.entrySet()){

MergeCluster mc = me.getValue();

String arr[] = mc.getCluster().split(",");

for(int i=0;i<arr.length;i++){

if(arr[i].trim().equals(name.trim())){

flag = true;

break;

}

}

}

return flag;

}

}

**MERGECLUSTER.JS**

package com;

public class MergeCluster{

String cluster;

int id;

public void setId(int id){

this.id = id;

}

public int getId(){

return id;

}

public void setCluster(String cluster){

this.cluster = cluster;

}

public String getCluster(){

return cluster;

}

**PROCESS THREAD.JAVA**

package com;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

import java.util.ArrayList;

import java.math.BigInteger;

import java.util.Map;

import java.math.BigDecimal;

import java.math.RoundingMode;

public class ProcessThread extends Thread{

Socket socket;

ObjectOutputStream out;

ObjectInputStream in;

BigDecimal BigDecimalTWO = new BigDecimal(2);

public ProcessThread(Socket soc){

socket=soc;

try{

out = new ObjectOutputStream(socket.getOutputStream());

in = new ObjectInputStream(socket.getInputStream());

}catch(Exception e){

e.printStackTrace();

}

}

public BigDecimal sqrt(BigDecimal number1, BigDecimal guess1, RoundingMode rounding1){

BigDecimal result1=BigDecimal.ZERO;

BigDecimal flipA1=result1;

BigDecimal flipB1=result1;

boolean first1=true;

while(result1.compareTo(guess1) !=0){

if(!first1)

guess1 = result1;

else

first1=false;

result1 = number1.divide(guess1, rounding1).add(guess1).divide(BigDecimalTWO, rounding1);

if(result1.equals(flipB1))

return flipA1;

flipB1 = flipA1;

flipA1 = result1;

}

return result1;

}

public double distance(BigDecimal[] vector1, BigDecimal[] vector2){

BigDecimal dot = new BigDecimal("0");

BigDecimal magnitude1 = new BigDecimal("0");

BigDecimal magnitude2 = new BigDecimal("0");

int loop = vector1.length;

for(int i=0;i<loop;i++){

BigDecimal sum = vector1[i].multiply(vector2[i]);

dot = dot.add(sum);

BigDecimal p1 = vector1[i].pow(2);

BigDecimal p2 = vector2[i].pow(2);

magnitude1 = magnitude1.add(p1);

magnitude2 = magnitude2.add(p2);

}

magnitude1 = sqrt(magnitude1,BigDecimal.ONE,RoundingMode.HALF\_UP);

magnitude2 = sqrt(magnitude2,BigDecimal.ONE,RoundingMode.HALF\_UP);

magnitude2 = magnitude1.multiply(magnitude2);

dot = dot.divide(magnitude2,10,RoundingMode.HALF\_UP);

double d = dot.doubleValue();

return d == Double.NaN ? 0 : d;

}

@Override

public void run(){

try{

Object input[]=(Object[])in.readObject();

String type=(String)input[0];

if(type.equals("centers")){

Homomorphic.KeyGeneration();

System.out.println("enter");

String[] receive\_centers = (String[])input[1];

String file = (String)input[2];

ArrayList<String[]> centers = new ArrayList<String[]>();

BigDecimal d1[] = new BigDecimal[receive\_centers.length];

for(int i=0;i<receive\_centers.length;i++){

java.math.BigInteger bd = new java.math.BigInteger(receive\_centers[i]);

d1[i] = new BigDecimal(bd);

}

StringBuilder sb = new StringBuilder();

ViewShare vs = new ViewShare();

for(Map.Entry<double[],MergeCluster> me : Recluster.map.entrySet()){

double cent[] = me.getKey();

String enc[] = new String[cent.length];

for(int i=0;i<cent.length;i++){

String value = Double.toString(cent[i]);

BigInteger encrypt = new BigInteger(value.getBytes());

encrypt = Homomorphic.Encryption(encrypt);

enc[i] = encrypt.toString();

}

BigDecimal d2[] = new BigDecimal[enc.length];

for(int i=0;i<enc.length;i++){

java.math.BigInteger bd = new java.math.BigInteger(enc[i]);

d2[i] = new BigDecimal(bd);

}

BigDecimal d3[] = new BigDecimal[enc.length];

for(int i=0;i<d1.length;i++){

d3[i] = d1[i];

}

for(int i=d1.length;i<d2.length;i++){

d3[i] = new BigDecimal("0.0");

}

double distance = distance(d2,d3);

if(distance > 0.05){

System.out.println(file+" "+distance+" "+ me.getValue().getCluster());

sb.append(file+" "+distance+" "+ me.getValue().getCluster()+"\n");

Object row[] = {file,distance,me.getValue().getCluster()};

vs.dtm.addRow(row);

}

}

System.out.println("Total Centroids : "+Recluster.map.size());

System.out.println("Total matches : "+vs.dtm.getRowCount());

BigInteger m1 = new BigInteger(Recluster.map.size()+"");

BigInteger m2 = new BigInteger(vs.dtm.getRowCount()+"");

BigInteger enc1 = Homomorphic.Encryption(m1);

BigInteger enc2 = Homomorphic.Encryption(m2);

BigInteger total = enc1.multiply(enc2).mod(Homomorphic.nsquare);

Object res[] = {sb.toString(),total.toString(),Recluster.map.size()+""};

out.writeObject(res);

out.flush();

vs.setVisible(true);

vs.setSize(600,400);

}

}catch(Exception e){

e.printStackTrace();

}

}

}

**SERVERTHREAD.JAVA**

package com;

public class ServerThread extends Thread

{

Main server;

public ServerThread(Main server){

this.server=server;

start();

}

public void run(){

server.start();

}

}

**READ DATASET.JAVA**

package com;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.File;

import java.io.FileWriter;

import java.util.ArrayList;

import java.util.HashSet;

import java.text.DecimalFormat;

public class ReadDataset{

static ArrayList<String[]> file\_array = new ArrayList<String[]>();

static ArrayList<File> files = new ArrayList<File>();

static ArrayList<String> unique\_terms;

static ArrayList<double[]> vector = new ArrayList<double[]>();

static DecimalFormat format = new DecimalFormat("#.###");

public static void clear(){

file\_array.clear();

files.clear();

if(unique\_terms != null){

unique\_terms.clear();

}

vector.clear();

}

public static void readFile(File folder)throws Exception{

clear();

File list[] = folder.listFiles();

for(int i=0;i<list.length;i++){

files.add(list[i]);

}

HashSet<String> hs = new HashSet<String>();

for(File file : files){

BufferedReader br = new BufferedReader(new FileReader(file));

String line = null;

StringBuilder buffer = new StringBuilder();

while((line = br.readLine())!=null){

buffer.append(line);

}

br.close();

String[] tokens = buffer.toString().replaceAll("[\\W&&[^\\s]]","").split("\\W+");

for(String terms : tokens){

hs.add(terms);

}

file\_array.add(tokens);

}

unique\_terms = new ArrayList<String>(hs);

}

public static void buildVector(){

for(String[] tokens : file\_array){

double[] tf\_idf = new double[unique\_terms.size()];

for(int i=0;i<unique\_terms.size();i++){

double value = getTermFrequency(tokens,unique\_terms.get(i)) \* getInverseDocument(file\_array,unique\_terms.get(i));

tf\_idf[i] = Double.parseDouble(format.format(value));

}

vector.add(tf\_idf);

}

}

public static double getTermFrequency(String[] doc, String word){

double counts = 0;

for(String terms : doc){

if(terms.equalsIgnoreCase(word))

counts++;

}

return counts/doc.length;

}

public static double getInverseDocument(ArrayList<String[]> docs, String word){

double counts = 0;

for(String[] doc : docs){

for(String terms : doc){

if(terms.equalsIgnoreCase(word)){

counts++;

break;

}

}

}

return Math.log(docs.size()/counts);

}

public static void saveVector(){

try{

StringBuilder sb = new StringBuilder();

for(int i=0;i<vector.size();i++){

double data[] = vector.get(i);

sb.append(files.get(i).getName()+",");

for(int j=0;j<data.length;j++){

sb.append(data[j]+",");

}

sb.deleteCharAt(sb.length()-1);

sb.append(System.getProperty("line.separator"));

}

FileWriter fw = new FileWriter("vector.txt");

fw.write(sb.toString());

fw.close();

}catch(Exception e){

e.printStackTrace();

}

}

}

**VIEWCLUSTERS.JAVA**

package com;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextArea;

import javax.swing.JScrollPane;

import java.awt.Font;

import java.awt.BorderLayout;

public class ViewClusters extends JFrame{

JPanel p1;

JTextArea area;

JScrollPane jsp;

Font f1;

public ViewClusters(){

super("View Clusters Data");

p1 = new JPanel();

p1.setLayout(new BorderLayout());

f1 = new Font("Times New Roman",Font.BOLD,16);

area = new JTextArea();

area.setEditable(false);

area.setLineWrap(true);

area.setFont(f1);

jsp = new JScrollPane(area);

p1.add(jsp,BorderLayout.CENTER);

getContentPane().add(p1,BorderLayout.CENTER);

}

}

**VIEWSHARE.JAVA**

package com;

import javax.swing.JFrame;

import javax.swing.JTable;

import javax.swing.JScrollPane;

import javax.swing.table.DefaultTableModel;

import java.util.ArrayList;

import java.awt.Font;

public class ViewShare extends JFrame{

JTable tab;

DefaultTableModel dtm;

JScrollPane jsp;

public ViewShare(){

super("View Share");

dtm = new DefaultTableModel(){

public boolean isCellEditable(int row\_no,int column\_no){

return false;

}

};

tab = new JTable(dtm);

tab.getTableHeader().setFont(new Font("Courier New",Font.BOLD,14));

tab.setFont(new Font("Courier New",Font.BOLD,13));

tab.setRowHeight(30);

jsp = new JScrollPane(tab);

dtm.addColumn("Bob File");

dtm.addColumn("Distance");

dtm.addColumn("Match Files");

getContentPane().add(jsp);

}

}

**RECLUSTER.JAVA**

package com;

import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeSet;

import java.util.LinkedHashMap;

import java.io.File;

public class Recluster{

static ArrayList<String> dataset = new ArrayList<String>();

static LinkedHashMap<double[],MergeCluster> map = new LinkedHashMap<double[],MergeCluster>();

static LinkedHashMap<double[],MergeCluster> mergemap = new LinkedHashMap<double[],MergeCluster>();

static int reader = 0;

static int merge = 0;

public static void readVector(){

try{

reader = 0;

merge = 0;

map.clear();

dataset.clear();

mergemap.clear();

BufferedReader br = new BufferedReader(new FileReader("vector.txt"));

String line = null;

while((line = br.readLine())!=null){

line = line.trim();

if(line.length() > 0)

dataset.add(line);

}

br.close();

}catch(Exception e){

e.printStackTrace();

}

}

public static void recluster(){

int divide = dataset.size()/2;

int first = divide;

int start = 0;

recursive(start,first);

int second = first + divide;

if(second != dataset.size())

second = divide + 1;

else

second = divide;

recursive(start,second);

}

public static void recursive(int start,int index){

int divide = index/2;

int first = divide;

KCluster.files.clear();

KCluster.vector.clear();

for(int i=start;i<first;i++){

String arr[] = dataset.get(reader).split(",");

KCluster.files.add(new File(arr[0]));

System.out.print(arr[0]+" ");

double data[] = new double[arr.length-1];

for(int k=1;k<arr.length;k++){

data[k-1] = Double.parseDouble(arr[k]);

}

KCluster.vector.add(data);

reader = reader + 1;

}

System.out.println();

KCluster.kcluster();

if(merge == 0){

merge = 1;

}else{

mergeCluster();

}

start = first;

int second = first + divide;

if(second != index)

second = first + divide + 1;

else

second = first + divide;

for(int i=start;i<second;i++){

String arr[] = dataset.get(reader).split(",");

KCluster.files.add(new File(arr[0]));

System.out.print(arr[0]+" ");

double data[] = new double[arr.length-1];

for(int k=1;k<arr.length;k++){

data[k-1] = Double.parseDouble(arr[k]);

}

KCluster.vector.add(data);

reader = reader + 1;

}

System.out.println();

KCluster.kcluster();

mergeCluster();

}

public static void mergeCluster(){

ArrayList<double[]> vector = KCluster.vector;

for(int k=0;k<vector.size();k++){

double data[] = vector.get(k);

double temp[] = null;

double value = 0;

for(Map.Entry<double[],MergeCluster> me : map.entrySet()){

double cent[] = me.getKey();

double sim = KCluster.distance(cent,data);

if(value < sim){

value = sim;

temp = cent;

}

}

if(temp != null){

MergeCluster mc = map.get(temp);

//if(!exists(mc.getCluster().split(","),KCluster.files.get(k).getName())){

String str = "";

if(mc.getCluster().split(",").length == 1)

str = KCluster.files.get(k).getName();

else

str = mc.getCluster()+","+KCluster.files.get(k).getName();

mc.setCluster(str);

map.put(temp,mc);

mergemap.put(temp,mc);

//}

}else{

MergeCluster mc = new MergeCluster();

mc.setCluster(KCluster.files.get(k).getName());

map.put(data,mc);

mergemap.put(temp,mc);

}

}

}

public static boolean exists(String arr[],String name){

boolean flag = false;

for(int i=0;i<arr.length;i++){

if(arr.length > 1){

if(arr[i].trim().equals(name.trim())){

flag = true;

break;

}

}

}

return flag; }}

**SHAMIR.JAVA**

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.Random;

public final class Shamir

{

public static SecretShare[] split(final BigInteger secret, int needed, int available, BigInteger prime, Random random)

{

System.out.println("Prime Number: " + prime);

final BigInteger[] coeff = new BigInteger[needed];

coeff[0] = secret;

for (int i = 1; i < needed; i++)

{

BigInteger r;

while (true)

{

r = new BigInteger(prime.bitLength(), random);

if (r.compareTo(BigInteger.ZERO) > 0 && r.compareTo(prime) < 0)

{

break;

}

}

coeff[i] = r;

}

final SecretShare[] shares = new SecretShare[available];

for (int x = 1; x <= available; x++)

{

BigInteger accum = secret;

for (int exp = 1; exp < needed; exp++)

{

accum = accum.add(coeff[exp].multiply(BigInteger.valueOf(x).pow(exp).mod(prime))).mod(prime);

}

shares[x - 1] = new SecretShare(x, accum);

System.out.println("Share " + shares[x - 1]);

}

return shares;

}

public static BigInteger combine(final SecretShare[] shares, final BigInteger prime)

{

BigInteger accum = BigInteger.ZERO;

for(int formula = 0; formula < shares.length; formula++)

{

BigInteger numerator = BigInteger.ONE;

BigInteger denominator = BigInteger.ONE;

for(int count = 0; count < shares.length; count++)

{

if(formula == count)

continue; // If not the same value

int startposition = shares[formula].getNumber();

int nextposition = shares[count].getNumber();

numerator = numerator.multiply(BigInteger.valueOf(nextposition).negate()).mod(prime); // (numerator \* -nextposition) % prime;

denominator = denominator.multiply(BigInteger.valueOf(startposition - nextposition)).mod(prime); // (denominator \* (startposition - nextposition)) % prime;

}

BigInteger value = shares[formula].getShare();

BigInteger tmp = value.multiply(numerator) . multiply(modInverse(denominator, prime));

accum = prime.add(accum).add(tmp) . mod(prime); // (prime + accum + (value \* numerator \* modInverse(denominator))) % prime;

}

System.out.println("The secret is: " + accum + "\n");

return accum;

}

private static BigInteger[] gcdD(BigInteger a, BigInteger b)

{

if (b.compareTo(BigInteger.ZERO) == 0)

return new BigInteger[] {a, BigInteger.ONE, BigInteger.ZERO};

else

{

BigInteger n = a.divide(b);

BigInteger c = a.mod(b);

BigInteger[] r = gcdD(b, c);

return new BigInteger[] {r[0], r[2], r[1].subtract(r[2].multiply(n))};

}

}

private static BigInteger modInverse(BigInteger k, BigInteger prime)

{

k = k.mod(prime);

BigInteger r = (k.compareTo(BigInteger.ZERO) == -1) ? (gcdD(prime, k.negate())[2]).negate() : gcdD(prime,k)[2];

return prime.add(r).mod(prime);

}

public static void main(final String[] args)

{

final int CERTAINTY = 256;

final SecureRandom random = new SecureRandom();

final BigInteger secret = new BigInteger("123");

// prime number must be longer then secret number

final BigInteger prime = new BigInteger(secret.bitLength() + 1, CERTAINTY, random);

// 2 - at least 2 secret parts are needed to view secret

// 5 - there are 5 persons that get secret parts

final SecretShare[] shares = Shamir.split(secret, 2, 5, prime, random);

// we can use any combination of 2 or more parts of secret

SecretShare[] sharesToViewSecret = new SecretShare[] {shares[0],shares[1]}; // 0 & 1

BigInteger result = Shamir.combine(sharesToViewSecret, prime);

sharesToViewSecret = new SecretShare[] {shares[1],shares[4]}; // 1 & 4

result = Shamir.combine(sharesToViewSecret, prime);

sharesToViewSecret = new SecretShare[] {shares[0],shares[1],shares[3]}; // 0 & 1 & 3

result = Shamir.combine(sharesToViewSecret, prime);

}

}

**MAIN.JAVA**

package com;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JButton;

import javax.swing.JPanel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.UIManager;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.Color;

import java.awt.Font;

import javax.swing.JOptionPane;

import javax.swing.JFileChooser;

import java.io.File;

import java.awt.Cursor;

import java.awt.Cursor;

import java.util.Map;

import java.util.ArrayList;

import java.net.ServerSocket;

import java.net.Socket;

import java.math.BigInteger;

import java.io.ObjectOutputStream;

import java.io.ObjectInputStream;

import java.net.Socket;

public class Main extends JFrame{

JPanel p1;

JPanel p2;

JLabel title;

JButton b1,b2,b3,b4,b5;

Font f1;

JFileChooser chooser;

File file;

ServerSocket server;

ProcessThread thread;

public void start(){

try{

server = new ServerSocket(2222);

while(true){

Socket socket = server.accept();

socket.setKeepAlive(true);

thread=new ProcessThread(socket);

thread.start();

}

}catch(Exception e){

e.printStackTrace();

}

}

public Main(){

super("Bob");

p1 = new JPanel();

f1 = new Font("Courier New",Font.BOLD,14);

p1.setBackground(new Color(204, 110, 155));

title = new JLabel("<HTML><BODY><CENTER>A New Privacy-Preserving Distributed k-Clustering Algorithm</CENTER></BODY></HTML>".toUpperCase());

title.setForeground(Color.white);

title.setFont(new Font("Times New Roman",Font.BOLD,16));

p1.add(title);

chooser = new JFileChooser(new File("."));

chooser.setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

p2 = new JPanel();

p2.setLayout(null);

b1 = new JButton("Upload Dataset");

b1.setFont(f1);

b1.setBounds(320,50,400,50);

p2.add(b1);

b1.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

int option = chooser.showOpenDialog(Main.this);

if(option == chooser.APPROVE\_OPTION){

file = chooser.getSelectedFile();

Cursor hourglassCursor = new Cursor(Cursor.WAIT\_CURSOR);

setCursor(hourglassCursor);

try{

ReadDataset.readFile(file);

ReadDataset.buildVector();

}catch(Exception e){

e.printStackTrace();

}

Cursor normalCursor = new Cursor(Cursor.DEFAULT\_CURSOR);

setCursor(normalCursor);

JOptionPane.showMessageDialog(Main.this,"Dataset loaded");

}

}

});

b2 = new JButton("Generate Vector");

b2.setFont(f1);

b2.setBounds(320,130,400,50);

p2.add(b2);

b2.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

Cursor hourglassCursor = new Cursor(Cursor.WAIT\_CURSOR);

setCursor(hourglassCursor);

ReadDataset.saveVector();

Cursor normalCursor = new Cursor(Cursor.DEFAULT\_CURSOR);

setCursor(normalCursor);

JOptionPane.showMessageDialog(Main.this,"Vector generated");

}

});

b3 = new JButton("Run Recluster Algorithm");

b3.setFont(f1);

b3.setBounds(320,210,400,50);

p2.add(b3);

b3.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

Recluster.readVector();

Recluster.recluster();

ViewClusters vc = new ViewClusters();

int id = 1;

ArrayList<String> dup = new ArrayList<String>();

for(Map.Entry<double[],MergeCluster> me : Recluster.mergemap.entrySet()){

MergeCluster mc = me.getValue();

String arr[] = mc.getCluster().split(",");

StringBuilder sb = new StringBuilder();

for(int i=0;i<arr.length;i++){

if(!dup.contains(arr[i])){

dup.add(arr[i]);

sb.append(arr[i]+",");

}

}

if(sb.length() > 0){

sb.deleteCharAt(sb.length()-1);

mc.setCluster(sb.toString());

}

vc.area.append("Cluster id : "+id+"\n\n");

vc.area.append("Cluster data : "+mc.getCluster()+"\n\n");

vc.area.append("====================================\n");

id = id + 1;

}

vc.setSize(600,400);

vc.setVisible(true);

}

});

b4 = new JButton("Share with Alice");

b4.setFont(f1);

b4.setBounds(320,290,400,50);

p2.add(b4);

b4.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

Cursor hourglassCursor = new Cursor(Cursor.WAIT\_CURSOR);

setCursor(hourglassCursor);

send();

Cursor normalCursor = new Cursor(Cursor.DEFAULT\_CURSOR);

setCursor(normalCursor);

}

});

b5 = new JButton("Exit");

b5.setFont(f1);

b5.setBounds(320,370,400,50);

p2.add(b5);

b5.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

System.exit(0);

}

});

getContentPane().add(p1,BorderLayout.NORTH);

getContentPane().add(p2,BorderLayout.CENTER);

}

public static void main(String a[])throws Exception{

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

Main main = new Main();

main.setVisible(true);

main.setExtendedState(JFrame.MAXIMIZED\_BOTH);

new ServerThread(main);

}

public int getRandomCenter(){

java.util.Random r = new java.util.Random();

return r.nextInt(ReadDataset.vector.size());//1

}

public void send(){

try{

Homomorphic.KeyGeneration();

int random = getRandomCenter();

ArrayList<double[]> keys = new ArrayList<double[]>(Recluster.map.keySet());

double cent[] = keys.get(random);

String file = Recluster.map.get(cent).getCluster();

String enc[] = new String[cent.length];

for(int i=0;i<cent.length;i++){

String value = Double.toString(cent[i]);

BigInteger encrypt = new BigInteger(value.getBytes());

encrypt = Homomorphic.Encryption(encrypt);

enc[i] = encrypt.toString();

}

Socket socket=new Socket("localhost",1111);

ObjectOutputStream out=new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in=new ObjectInputStream(socket.getInputStream());

Object req[]={"centers",enc,file};

out.writeObject(req);

out.flush();

Object res[]=(Object[])in.readObject();

String msg = res[0].toString();

System.out.println(msg);

String match = (String)res[1];

String total = (String)res[2];

BigInteger encr = new BigInteger(match);

BigInteger dec = Homomorphic.Decryption(encr);

System.out.println(msg);

System.out.println("Total Centroids : "+total);

System.out.println("Total matches : "+(Integer.parseInt(total)-Integer.parseInt(dec.toString())));

}catch(Exception e){

e.printStackTrace();

}

}

}

**OTHER CODE WRITTEN FOR TESTING PURPOSE.**

**K-MEANS.PY**

import numpy as np

import sys

from sklearn.metrics import jaccard\_similarity\_score

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

MAX\_ITERATIONS = 10

k = sys.argv[2]

filename = sys.argv[1]

groundtruth = []

clusterResult = []

orderedpairs = {}

inputdata = {}

initial\_centroids = {}

clusterswithdistance = {}

centroid = {}

centroids = []

for i in range(3,len(sys.argv)):

centroids.append(int(sys.argv[i]))

# Parse the input data from .txt file and store into key value pair where key is gene id and value is the list of expression values

def parse(s):

count = 0

result = {}

with open(filename) as f:

for row in f:

count += 1

row = row.strip().split()

key = int(row[0])

result[key] = row[2:]

for x in result:

final = [float(i) for i in result[x]]

result[x] = final

return result

#

# def get\_initial\_centroids(inputdata, k):

# a = random.sample(inputdata, k)

# for i in xrange(0, k):

# if inputdata.\_\_contains\_\_(a[i]):

# key = i

# initial\_centroids[key] = inputdata.get(a[i])

# return initial\_centroids

def assign\_clusters\_by\_distance():

clusterswithdistance.clear()

for i in inputdata:

list1 = inputdata.get(i)

distancelist = []

distances = {}

for j in initial\_centroids:

list2 = initial\_centroids.get(j)

distance = calculate\_distance(list1, list2)

distancelist.append(distance)

distances[distance] = j

min\_dist = min(distancelist)

closestcluster = distances.get(min\_dist)

if clusterswithdistance.get(closestcluster):

existing = []

existing = clusterswithdistance.get(closestcluster)

existing.append(i)

clusterswithdistance[closestcluster] = existing

else:

firstone = [i]

clusterswithdistance[closestcluster] = firstone

return clusterswithdistance

def compute\_new\_centroids():

initial\_centroids.clear()

for cluster in clusterswithdistance.keys():

geneids = clusterswithdistance.get(cluster)

centroid = []

listofallgenes = []

for id in geneids:

if inputdata.\_\_contains\_\_(id):

listofallgenes.append(inputdata.get(id))

else:

print "Gene is not present"

num\_of\_column = len(listofallgenes[0])

for col in xrange(0, num\_of\_column):

sum\_of\_column = 0.0

for i in xrange(0, len(listofallgenes)):

sum\_of\_column += listofallgenes[i][col]

centroid.append(sum\_of\_column / len(listofallgenes))

initial\_centroids[cluster] = centroid

return initial\_centroids

def calculate\_distance(list1, list2):

sum = 0.0

for i in xrange(0, len(list1)):

sum += (list1[i] - list2[i]) \*\* 2

distance = np.sqrt(sum)

return distance

def get\_demo\_centroids():

for i in xrange(0 , len(centroids)):

centroid[i] = inputdata.get(centroids[i])

return centroid

def get\_groundtruth(s):

with open(filename) as f:

for row in f:

row = row.strip().split()

groundtruth.append(int(row[1]))

return groundtruth

def get\_jaccard\_score():

y\_true = get\_groundtruth(filename)

y\_pred = clusterResult

return jaccard\_similarity\_score(y\_true, y\_pred)

def check\_termination(prev, current):

for cluster1 in prev.keys():

previous\_centroids = prev.get(cluster1)

current\_centroids = current.get(cluster1)

if set(current\_centroids) != set(previous\_centroids):

return False

return True

inputdata = parse(filename)

initial\_centroids = get\_demo\_centroids()

clusterswithdistance = assign\_clusters\_by\_distance()

converge = False

iterations = 0

while iterations < MAX\_ITERATIONS:

iterations += 1

previous\_intial\_centroids = initial\_centroids

clusterswithdistance = assign\_clusters\_by\_distance()

new\_initial\_centroids = compute\_new\_centroids()

converge = check\_termination(previous\_intial\_centroids, new\_initial\_centroids)

for cluster in xrange(0, len(clusterswithdistance.keys())):

print "Cluster " + str(cluster) + " " + "Size is " + str(len(clusterswithdistance.get(cluster))) + " and GeneId's are: " + str(clusterswithdistance.get(cluster))

for cluster in clusterswithdistance.keys():

for id in clusterswithdistance.get(cluster):

orderedpairs[id] = cluster

for i in orderedpairs.keys():

clusterResult.append(orderedpairs.get(i))

print "JACCARD COEFFICENT IS: " + str(get\_jaccard\_score())

**CODE FOR VISUALIZATION:**

list = []

for key, value in inputdata.iteritems():

list.append((value))

list = np.array(list)

y = np.array(clusterResult)

target\_names = [0, 1, 2, 3, 4]

pca = PCA(n\_components=2)

X\_r = pca.fit(list).transform(list)

plt.figure()

plt.scatter(X\_r[:, 0], X\_r[:, 1], c=y, )

plt.legend()

plt.title('PCA')

plt.show()