

# A Report on: Quantum Inspired Neural Networks for Signature Verification

May 1, 2018

*Design Project (EEE F376)*

*Under the guidance of*

Mr. Ashish Patel  
Electrical and Electronics Department, BITS Pilani



**Submitted by:**

Arjun Gupta

2014B4A30754P

Sonali Rajesh Medani

2014B2A30563P

Ameya Dhamanaskar

2014A3PS0187P

## Acknowledgment

We would like to express our gratitude to our project in-charge Mr. Ashish Patel. We were quite eager to learn about advance concepts of Neural Networks such as Quantum Neural Networks and we are extremely grateful for the opportunity provided to us to work under his guidance on the topic. We would also like to thank Dr. Sanjay Singh and Mr. Sumeet Saurav from CEERI Pilani. Their constant guidance, insights and regular discussions have been extremely valuable. They have provided us with a wonderful research experience, which has shaped our approach to tackle research problems especially on QNN. We will be eternally grateful for the guidance provided.

# Contents

Acknowledgment

Abstract

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Image Preprocessing</b>	<b>2</b>
<b>3</b>	<b>Feature Extraction</b>	<b>4</b>
<b>4</b>	<b>Basic Concepts of QNN</b>	<b>5</b>
4.1	Quantum Neural Network . . . . .	5
4.2	The Qubit Representation . . . . .	5
4.3	Conversion from Quantum Bits to Real Value [1] . . . . .	6
4.4	Qubit Updation . . . . .	6
4.5	Boundary Parameters . . . . .	7
<b>5</b>	<b>Algorithm</b>	<b>7</b>
<b>6</b>	<b>Experimental Results</b>	<b>9</b>
	<b>References</b>	<b>11</b>
	<b>Appendix</b>	<b>12</b>

## Abstract

In this report, Enhanced Quantum based Neural Network Learning paper is implemented for Signature Verification. The proposed algorithm forms a neural network architecture constructively by adding the hidden layer neurons. The connection weight and threshold of the neurons are decided using the quantum computing concept. The quantum computing concept gives large subspace for selection of appropriate connection weights in evolutionary ways. Also, the threshold value is decided using the quantum computing concept. To uniquely identify the signatures, a total of 45 features are extracted from each signature of dataset BHSig260. The performance of the proposed algorithm is evaluated by rigorous training and testing with these signatures, and the results confirm its accuracy and effectiveness.

# 1 Introduction

Human brain has the great ability to unravel and classify the complex patterns of the real world. Inspiring from the brain anatomy, artificial neural network was introduced in 1943. Human brain needs training specific to the kind of task involved. Analogously, artificial neural network also needs training algorithm. Many models have been proposed like back propagation, perceptron and recurrent network which represent working of the human brain. These models have been successfully applied in several fields like economics, defense, stock market, engineering, medical, computer network and many more. However, performance of neural network in these mentioned areas depends on many parameters like, quality of input data set, number of hidden layer neurons, threshold of neurons, connection weights, etc. To enhance the approximation and generalization ability of classical artificial neural network (ANN), the principles of quantum computation are employed. However, as yet, there is little understanding of the essential components of artificial neural networks based on quantum theoretical concepts and techniques. The basal model and theory of quantum neural networks are in research. At present, there is not a set of perfect theory to direct the construction of model. Quantum computing concept was, firstly, introduced in classical computing. The significant work has been done by Han and Kim to solve the knapsack problem using the quantum computing concept with and without termination criteria. Here, qubit  $q$  is defined as a smallest unit of information which have better characteristic of the population diversity than other representations. Since qubits are linear superposition of states of probabilistic thus, with the help of Gaussian random generation it gives diversity to select the optimal value of parameters from large subspace.

A neural network algorithm has been proposed in which optimization of the learning parameters was carried out using quantum computing concept. Earlier an algorithm was proposed based on binary neural network learning algorithm in which the neural network architecture is formed constructively. In this algorithm, the connection weights are decided using the quantum computing concept. Further improvement is proposed in this paper by deciding the connection weights and threshold using the quantum computing concept. The neural network formed in this way is trained and tested on a signature dataset.

Application: Handwritten signatures are widely used to authenticate financial transactions and documents. Signature Verification is used to authenticate signatures, by capturing their unique features, to avoid forgery. Various

forms of biometric security systems exists such as fingerprint, iris, speech, heart sound and keystroke based recognition, all of which depend on the physical attributes of the users. But still, signature verification is one of the most popular attribute accepted by the public, as it: (1) is more comfortable, (2) is more economical and (3) requires less storage space. It is an automated method of verifying a signature with the actual authorized signature by capturing some of its unique features like, the shape of signature (i.e., static or off-line signature verification) or the parameters that can capture the unique features of how the authenticator signs his/her name in real-time. To make this task more efficient, an enhanced quantum-based neural network learning algorithm for signature verification is proposed.

## 2 Image Preprocessing

We don't want our algorithm to be affected by the selected dataset and thus we pre process the image so that no matter how we take an image of a signature, the proposed algorithm classify it correctly. Following are the pre processing steps:

1. **Converting to Black and White:** We convert the image to black and white so that the pen colour does not effect out feature extraction process. By doing so, the colour of the image is made uniform.
2. **Noise Removal:** The noise removal process is performed after converting all images in uniform i.e. black and white color. The signature images have noise due to two main sources: first, the background paper on which the signature is taken which may not be uniform of the same color. Secondly, the noise arises while scanning the paper having signatures. This noise will hinder the training and testing of signatures and hence must be removed. Median Filtering is used as a remedy here.

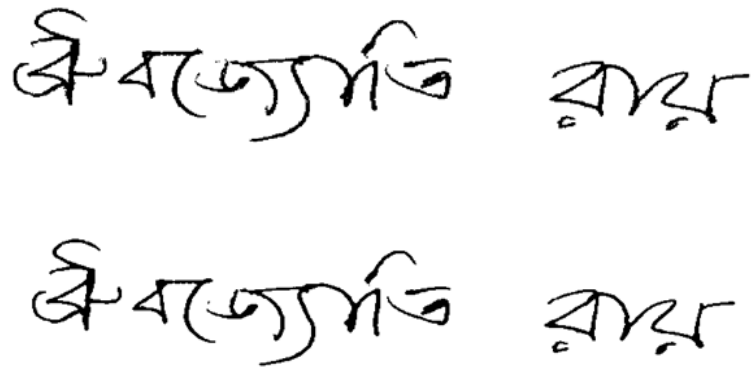


Figure 1: Top image is the original signal signature and bottom is after the median filtering

3. **Image Resizing:** Every image size is reduced to 128 X 128 pixels so that all the images have the uniform size.
4. **Image Thinning:** A signature impression may be made with pens of varying tips. However, the difference in tip size shouldn't be a factor to distinguish signatures. The thickness of every stroke in a signature is reduced to a width of a single pixel. The steps discussed above help to standardize a given signature image [2].

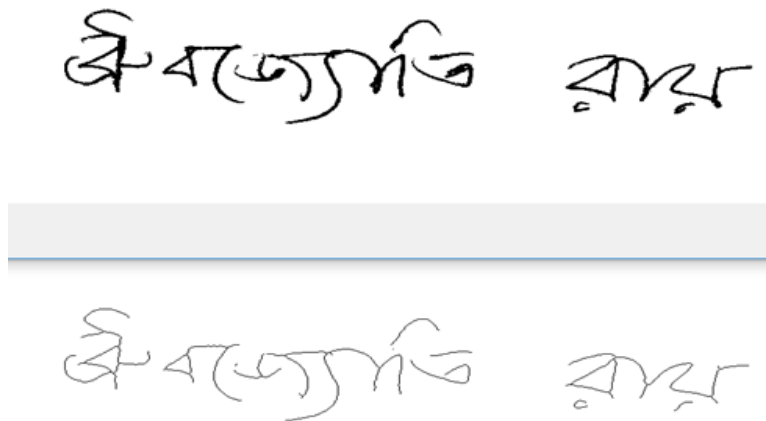


Figure 2: Top image is the original signal signature and bottom is after thinning

5. **Image Cropping:** Sometimes due to variable distance of camera from

the signature the paper content image is more. Thus to reduce that effect we crop image to exact size of signature.

### 3 Feature Extraction

The unique features are extracted from the pre processed x-y coordinate that are further given as input to QNN model for signature recognition.

1. **Angle of Signature:** It may happen that the same authenticator may use different elevation angles for every instant w.r.t origin in the x-y grid. Therefore, the angle of rotation must be standardized [3].
2. **Centre of Mass Coordinates:** Every person signs bit differently every time but the centre of mass remains approximately same as the width and the height vary relatively.
3. **No. of Loops:** Count the no. of loops in a signature as they are peculiar to particular signatures.

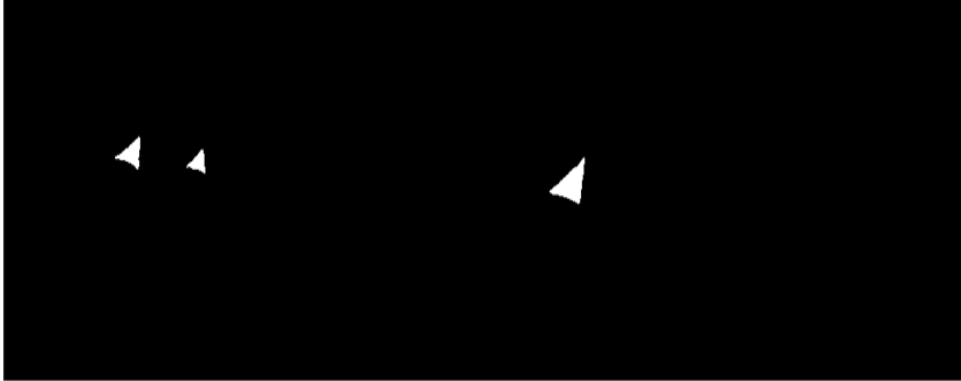


Figure 3: White Regions shows the loops

4. **Dense Rows and Columns:** This gives 30 feature values. Calculate the density of rows and columns and choose the highest 10 values for both rows and columns. Furthermore distances of each most dense row and column is calculated from the origin.
5. **Density of 5 most dense patches:** We took 9 X 9 square patches and calculated their densities and then chose the highest 11 densities.



## 4 Basic Concepts of QNN

In this section some basic necessary concepts are discussed in brief, which helps to illustrate the proposed algorithm. Preliminaries related to quantum computing is explained:

### 4.1 Quantum Neural Network

This method forms a neural network architecture, which consists of four layers: an input layer, two hidden layers, and the output layer. The number of input nodes is equal to the number of attributes of the signature dataset. Let  $P_1 = (X_1^1, X_1^2, \dots, X_1^{c_1})$  denote the input samples, where  $c_1$  is the number of input samples and  $X_i^j = (x_i^1, x_i^2, \dots, x_i^e)$  where  $e$  denotes the number of attributes in one instance of the input sample. The number of input layer nodes is equal to  $e$  [4]. The number of neurons in the hidden layer is decided constructively. For  $i^{th}$  hidden layer neuron, the connection weights are denoted as follows:

$$W_i^{real} = (w_{i1}, w_{i2}, \dots, w_{ie}) \quad (1)$$

In the proposed algorithm, these connection weights are decided using the quantum computing concept.

### 4.2 The Qubit Representation

Quantum bits, which differ from traditional bits, use probability to represent binary information. A characteristic of quantum bit representation is the ability to represent a linear superposition of “1” and “0” states probabilistically. A quantum bit individual containing a string of  $q$  quantum bits can be defined as:

$$\left\langle \begin{matrix} \alpha_1 & \alpha_2 & \dots & \alpha_q \\ \beta_1 & \beta_2 & \dots & \beta_q \end{matrix} \right\rangle$$

where  $0 \leq \alpha_i, \beta_i \leq 1$ ,  $(\alpha_i)^2 + (\beta_i)^2 = 1$ , and  $i = 1, 2, \dots, q$ ,  $(\alpha_i)^2$  is the probability that the  $i^{th}$  quantum bit will be found in state “1” and  $(\beta_i)^2$  is the probability that the  $i^{th}$  quantum bit will be found in state “0.” Since  $(\alpha_i)^2 + (\beta_i)^2 = 1$ ,

$$\langle \alpha_1 | \alpha_2 | \dots | \alpha_q \rangle \quad (2)$$

The observation is a process that produces a binary string  $b$  from (1), which operates as follows. For a quantum bit individual with  $q$  quantum bits, generate a  $q$  random number vector  $r = [r_1, r_2, \dots, r_q]$ , where  $0 \leq r_i \leq 1$ ,  $i =$

1,2,...,q; the corresponding bit in b takes "1" if  $r_i \leq (\alpha_i)^2$ , or "0" otherwise [5].

### 4.3 Conversion from Quantum Bits to Real Value [1]

The algorithm which is proposed here works on classical computers, therefore conversion from quantum bits to real value is required . The weight matrix in terms of quantum bits  $W'_j$  is converted into a real value weight matrix  $W_j^{real}$ . Similarly the threshold value in terms of quantum bits  $\lambda'_j$  is converted into real value  $\lambda_j^{real}$ . This conversion process starts by taking random number matrices R, where  $R_j = [r_{j1}, r_{j2}, ..., r_{jk}]$ . Then, further mapping is done by using binary matrix  $S_j$  where  $S_j = [s_{j1}, s_{j2}, ..., s_{jk}]$  and Gaussian random number generator with mean value  $\mu$  and variance  $\sigma$ , which can be represented as  $N(\mu, \sigma)$ . The mapping between binary value to Gaussian number generator is done with the help of formula binary to decimal conversion. The value of matrix  $S_j$  is passed into  $\text{bin2dec}(S_j)$  formula to select value from Gaussian random generator [5]. The value of matrix  $S_j$  is generated as follows:

$$if(r_{ji} \leq (\alpha_{ji})^2) \text{ then } s_{ji} = 1 \text{ else } s_{ji} = 0.$$

### 4.4 Qubit Updation

Evolutionary algorithms are applied to optimize the solution of varying parameters and its find out in several iteration by observing their fitness [6]. Therefore, to evolve tehe new value of weight matrix  $W_j^{real}$  Fitness function for weight  $W_j^{real}$  and real threshold  $\lambda_j^{real}$ , the quantum weights  $W'_j$  and quantum threshold  $\lambda'_j$  are updated using quantum update function which utilizes the fitness value, let us denote fitness by F and  $F^*$ . To update qubit, the required quantum gate is as follows:

$s_{ji}^g$	$s_{ji}^*$	$F_g < F^*$	$\Delta\theta$
0	0	false	0
0	0	true	0
0	1	false	$-0.03 * \Pi$
0	1	true	0
1	0	false	$0.03 * \Pi$
1	0	true	0
1	1	false	0
1	1	true	0

$$U(\Delta\theta) = \begin{vmatrix} \cos \Delta\theta & -\sin \Delta\theta \\ \sin \Delta\theta & \cos \Delta\theta \end{vmatrix}$$

where  $\Delta\theta$  is called rotation angle.

## 4.5 Boundary Parameters

In the proposed algorithm, the threshold  $Th_i^{real}$  of the neuron is evolved using the quantum computing concept and boundary parameters. Here, to select threshold,  $min\_net$  and  $max\_net$  parameters are introduced. These parameters are initialized as  $-\inf$  and  $\inf$  respectively. The following formulations show the calculation of boundary parameter.

$$(TBP)_t^{max} = \max(max\_net_A, max\_net_B);$$

$$(TBP)_t^{min} = \min(min\_net_A, min\_net_B);$$

Thus, threshold in terms of these boundary parameter is formulated as follows:

$$Th_t^{real} = \begin{cases} (TBP)_t^{min} & \text{if } Th_t^{real} < (TBP)_t^{min} \\ (TBP)_t & \text{if } (TBP)_t^{min} \leq Th_t^{real} \leq (TBP)_t^{max} \\ (TBP)_t^{max} & \text{if } Th_t^{real} > (TBP)_t^{max} \end{cases}$$

## 5 Algorithm

The overall process is explained in the form of an algorithm:-

**Step1:** Take Input sample as  $(X_1^1, X_1^2, X_1^3, \dots, X_1^{c_1})$  and  $(Y_1^1, Y_1^2, Y_1^3, \dots, Y_1^{c_1})$  corresponding to each person. Take first neuron with the weights  $W_g^{quant} = (Q_{w1}, Q_{w2}, \dots, Q_{we})$  where  $e$  is the number of attributes and  $g$  denotes the iteration number.

**Step2:** for  $g = 1$  to  $m$

Initialization of other parameters

$$F^* = 0, S^* = 0$$

$$max\_net_A = -\inf$$

$$min\_net_A = \inf$$

```

max_netB = - inf
max_netB = - inf
Implement Conversion Process of  $W_g^{quant}$ 
for i = 1 to  $c_1$ 
  netA(i) =  $\sum W_g^{real} xY_1^i$ 
  max_netB = max(max_netB, netB(i))
  min_netB = min(min_netB, netB(i))
endfor
Call Quantum Threshold Function ( $W_g^{real}$ )
if( $F_g \geq (c_1 + c_2)$ )
  Finish learning process
  Assigned new dataset to class A as  $P_{l+1}$  and class B as (P- $P_{l+1}$ )
  Repeat step1 and step2 for learning of class  $P_{l+1}$  and (P- $P_{l+1}$ )

else
  Evaluate  $F_g, F^*, s_i^g, s_i^*$  and update quantum bits of weights by using
  qubits updation weight updation.
   $F^* = \max(F^*, F_g)$ 
endif
if ((g==m) and ( $F^* \leq (c_1 + c_2)$ ))
  Add new neuron for unlearnt sample
  (( $c_1 + c_2$ ) -  $F^*$ ) and finalize its weight by using
  Step1 and Step2. For second neuron number of samples will
  be (( $c_1 + c_2$ ) -  $F^*$ ) not  $c_1 + c_2$ .
endif
g=g+1
endfor
Repeat the process for each person from step1 to step2

```

## Quantum Threshold Function

**Step1:** Initialization of different parameters

```

for t=1 to z
z is a variable whose value is decided by the user by deciding the number of
iterations for threshold updation  $Th_t$ 
 $Th_t = (q_i^{Th})$ 
count1=0
count2=0
 $F_{Th}^* = 0$  Implement Conversion Process of  $Th_t$  to generate
real value.
for i = 1 to  $c_1$ 

```

```

 $net_A(i) = \sum W_g \times X_1^i$ 
if ( $net_A(i) \leq Th_t^{real}$ )
  increase count1 by 1
endif
endfor
for i = 1 to  $c_2$ 
 $net_A(i) = \sum W_g \times Y_1^i$ 
if ( $net_A(i) > Th_t^{real}$ )
  increase count2 by 1
endif
endfor
 $F_t = \text{count1} + \text{count2}$ 
 $F_{Th}^* = \max(F_{Th}^*, F_t)$ 
  update quantum bits for  $Th_{t+1}$  using
  qubits updation and weight updation
  Generate real value of  $Th_{t+1}$ 
  t=t+1
 $F_g = F_{Th}^*$ 
endfor
return  $F_g$ 

```

## 6 Experimental Results

The proposed algorithm has been implemented in two parts.

1. The first part includes processing of the signature image and extraction of features which are implemented in MATLAB.
2. The second part consists of training of neural network and then testing, which has been implemented in Java.

(Implementations are done on Intel i-5 4th generation processor).

The algorithm is tested on a standard online database of signatures: BH-Sig260

To train the proposed algorithm we have taken 8 signatures each of 37 people, with 45 extracted attributes per signature. Furthermore, to test the algorithm, 2 signatures (one forged and one real) each of 37 people were tested. The quantum-based neural network forms network structure constructively, which reduces unnecessary training of the system. The connection weights

are decided using the quantum computing concept. To find the proper separation between input classes, a quantum threshold with boundary parameter is also proposed. The threshold boundary parameter helps to find the optimal value of threshold with the help of min, max function. This enhanced quantum-based neural network learning algorithm proposed by Prof. Om Prakash [4], is employed successfully to classify offline signatures.

To judge the performance of the proposed, it is compared with ANN algorithm with respect to ANN for signature verification,

	ANN	QNN
Maximum Efficiency	68%	71%
Computation Time	2 min	5 min

It is clear from the results that there is definite improvement in Classification with QNN and is mainly due to the two reasons. Firstly, the unique features that have been selected to characterize signature. Secondly, the classification of signature dataset uses the quantum computing concept provides exploration due to which the large search space is achieved to get optimal value. It is also observed that more the number of iterations for weight and threshold updation, more is the accuracy.

## References

- [1] Om Patel, Aruna Tiwari, Vikram Patel, and Ojas Gupta. Quantum based neural network classifier and its application for firewall to detect malicious web request. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 67–74. IEEE, 2015.
- [2] Louisa Lam, Seong-Whan Lee, and Ching Y Suen. Thinning methodologies-a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 14(9):869–885, 1992.
- [3] KV Lakshmi and Seema Nayak. Off-line signature verification using neural networks. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 1065–1069. IEEE, 2013.
- [4] Om Prakash Patel, Aruna Tiwari, Rishabh Chaudhary, Sai Vidyaranya Nuthalapati, Neha Bharill, Mukesh Prasad, Farookh Khadeer Hussain, and Omar Khadeer Hussain. Enhanced quantum-based neural network learning and its application to signature verification. *Soft Computing*, pages 1–14, 2017.
- [5] Tzyy-Chyang Lu, Gwo-Ruey Yu, and Jyh-Ching Juang. Quantum-based algorithm for optimizing artificial neural networks. *IEEE transactions on neural networks and learning systems*, 24(8):1266–1278, 2013.
- [6] Mohamed Hammami, Youssef Chahir, and Liming Chen. Webguard: A web filtering engine combining textual, structural, and visual content-based analysis. *IEEE Transactions on Knowledge and Data Engineering*, 18(2):272–284, 2006.

# Appendix

## Counting loops

```
mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
1 function loopcount = loops(image)
2 %image = imread('C:\Users\Arjun Gupta\Downloads\4-2\QNN\BHSig260\Bengali\001\B-S-1-G-01.tif');
3 %imshow(image)
4 image = medfilt2(image);
5 %figure, imshow(image)
6 image1 = image;
7 image1 = im2bw(image1, 0.5);
8 image1 = imcomplement(image1);
9 image1 = imfill(image1, 'holes');
10 image2 = image;
11 image2 = im2bw(image2, 0.5);
12 image2 = imcomplement(image2);
13 imshow(image1-image2)
14 loopcount = max(max(bwlabel(image2-image1)));
```

## Signature Preprocessing

```
mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
1 clear all
2 clc
3 close all
4 srcFiles = dir('C:\Users\Arjun Gupta\Downloads\4-2\QNN\test1\*.tif'); % the folder in which ur images exists
5 dense_col = zeros(length(srcFiles),5);
6 dense_row = zeros(length(srcFiles),5);
7 features = zeros(length(srcFiles),13);
8 output_count = 1;
9 var_count = 1;
10
11 for si = 1 : length(srcFiles)
12     filename = strcat('C:\Users\Arjun Gupta\Downloads\4-2\QNN\test1\',srcFiles(si).name);
13     I = imread(filename);
14     %I=imread('signature.png');
15     figure,imshow(I);
16     %pause
17     loopcount = loops(I);
18     imshow(I)
19     I = medfilt2(I); %removing noise
20
21     I2=imresize(I,[128 ,128]);
22     %figure, imshow(I2)
23     %figure,imshow(I2);
24     %pause
25     %I3=rgb2gray(I2);
26     I3=im2double(I2);
27     I3=im2bw(I3);
28     %figure, imshow(I)
```



```

mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
28 %figure, imshow(I)
29
30 %converting image to black and white
31 I3 = bwmmorph(~I3, 'thin', inf); %thining the image
32 I3=~I3;
33 figure,imshow(I3);
34
35
36 [rows cols] = size(I3);
37 top = 0;
38 bottom = 0;
39 left = 0;
40 right = 0;
41
42
43 k=1;
44 for i=1:128
45     for j=1:128
46         if(I3(i,j)==0)
47             u(k)=i;
48             v(k)=j;
49             k=k+1;
50             I3(i,j)=1;
51         end
52     end
53 end
54 C=[u:v];%the curve of the signature
55 N=k-1;%the number of pixels in the signature

```

```

mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
55 N=k-1;%the number of pixels in the signature
56 oub=sum(C(1,:))/N; %the original x co-ordinate center of mass of the image
57 ovb=sum(C(2,:))/N; %the original y co-ordinate center of mass of the image
58
59
60 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*****ROTATE*****%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
61 %moving the signature to the origin
62 for i=1:N
63     u(i)=u(i)-oub+1;
64     v(i)=v(i)-ovb+1;
65 end
66 % the new curve of the signature
67 C=[u:v];
68
69 ub=sum(C(1,:))/N;
70 vb=sum(C(2,:))/N;
71 ubSq=sum((C(1,:)-ub).^2)/N;
72 vbSq=sum((C(2,:)-vb).^2)/N;
73
74 for i=1:N
75     uv(i)=u(i)*v(i);
76 end
77
78 uvb=sum(uv)/N;
79 M=[ubSq uvb;uvb vbSq];
80 %calculating minimum igen value of the matrix
81 minIgen=min(abs(eig(M)));
82 %the eigen vector

```

```

mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
82 %the eigen vector
83 MI=[ubSq-minIgen uvb;uvb vbSq-minIgen];
84 theta(si)=(atan((-MI(1))/MI(2))*180)/pi;
85
86
87 thetaRad=(theta*pi)/180;
88
89 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
90
91 %cropping the picture to exact size of the image
92 for i=1:rows
93     for j=1:cols
94         if (I3(i,j)==0)
95             top = i;
96             break;
97         end
98     end
99     if(top~=0)
100         break;
101     end
102 end
103
104 for i=rows:-1:1
105     for j1=1:cols
106         if (I3(i,j1)==0)
107             bottom = i;
108             break;
109         end
110     end
111 end

```

```

mainnewton.m x main.m x data30.m x signature_preprocess.m* x loops.m x +
110
111     if(bottom~=0)
112         break;
113     end
114 end
115
116 for j=1:cols
117     for i=1:rows
118         if (I3(i,j)==0)
119             left = j;
120             break;
121         end
122     end
123     if(left~=0)
124         break;
125     end
126 end
127
128 for j=cols:-1:1
129     for i=1:rows
130         if (I3(i,j)==0)
131             right = j;
132             break;
133         end
134     end
135     if(right~=0)
136         break;
137     end
138 end

```

```

mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
139 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
140 %finding 5 most dense rows and cols
141 countz = zeros(1,cols) ;
142 colz = 1;
143 for i=1:cols
144     for j=1:rows
145         if (I3(j,i)==0)
146             countz(1,colz) = countz(1,colz)+1;
147         end
148     end
149     colz=colz+1;
150 end
151 countz = countz./rows;
152
153
154 for i = 1:20
155     [max_num,Y]=max(countz(:)) ;
156     %[X Y]=ind2sub(size(countz),max_num);
157     dense_col(si,i) = Y;
158     countz(1,Y) = 0;
159 end
160
161 countz1 = zeros(1,rows) ;
162 rowz = 1;
163 for i=1:rows
164     for j=1:cols
165         if (I3(i,j)==0)
166             countz1(1,rowz) = countz1(1,rowz)+1;
167         end

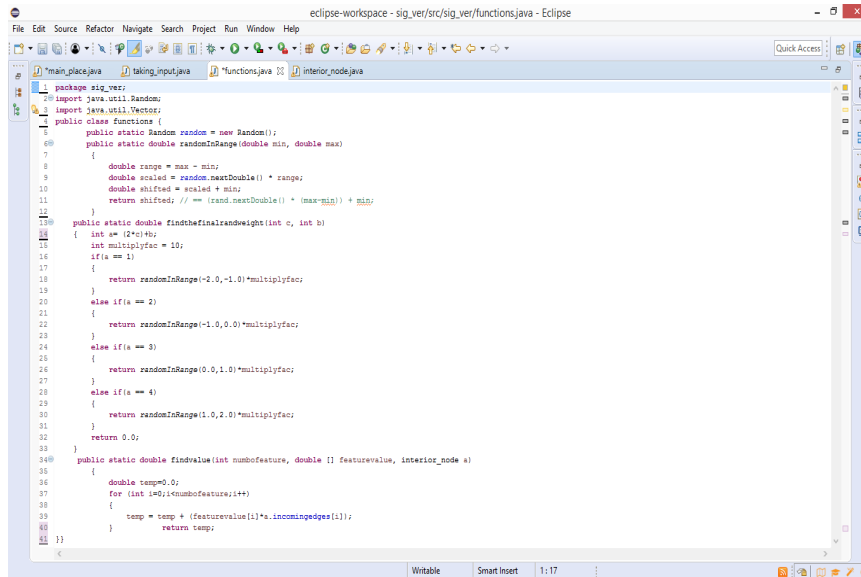
```

```

mainnewton.m x main.m x data30.m x signature_preprocess.m x loops.m x +
168     end
169     rowz=rowz+1;
170 end
171
172 countz1 = countz1./cols;
173
174 for i = 1:21
175     [max_num,Y]=max(countz1(:)) ;
176     %[X Y]=ind2sub(size(countz1),max_num);
177     dense_row(si,i) = Y;
178     countz1(1,Y) = 0;
179 end
180
181 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
182 %storing all obtained features in a matrix
183 features(si,1) = theta(si);
184 features(si,2:3) = [oub,ovb];
185 features(si,4) = loopcount;
186 %features(si,5) = count;
187 features(si,5:45) = [dense_col(si,:), dense_row(si,:)];
188
189 I4 = I3(top:bottom, left:right);
190 figure,imshow(I4);

```

# JAVA Code



```
eclipse-workspace - sig_ver/src/sig_ver/functions.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
package sig_ver;
import java.util.Random;
import java.util.Vector;
public class functions {
    public static Random random = new Random();
    public static double randomInRange(double min, double max)
    {
        double range = max - min;
        double scaled = random.nextDouble() * range;
        double shifted = scaled + min;
        return shifted; // == (rand.nextDouble() * (max-min)) + min;
    }
    public static double findTheFinalRandomWeight(int c, int b)
    {
        int a = (2*c)+b;
        int multiplyfac = 10;
        if(a == 1)
        {
            return randomInRange(-2.0,-1.0)*multiplyfac;
        }
        else if(a == 2)
        {
            return randomInRange(-1.0,0.0)*multiplyfac;
        }
        else if(a == 3)
        {
            return randomInRange(0.0,1.0)*multiplyfac;
        }
        else if(a == 4)
        {
            return randomInRange(1.0,2.0)*multiplyfac;
        }
        return 0.0;
    }
    public static double findValue(int numofeature, double [] featurevalue, interior_node a)
    {
        double temp=0.0;
        for (int i=0;i<numofeature;i++)
        {
            temp = temp + (featurevalue[i]*a.incomingedges[i]);
        }
        return temp;
    }
}
```

```
1 package sig_ver;
2
3 import java.util.Vector;
4
5 public class interior_node {
6
7     int featur = 45;
8     double [] incomingedges = new double[featur];
9     double value;
10    double outgoingedge;
11    int class_classifying;
12    int number_within_class;
13    double alpha;
14    double alphagreater;
15    double alpha1[] = new double[featur];
16    double alpha2[] = new double[featur];
17    double alpha3[] = new double[featur];
18 }
19
```

```

1 package sig_ver;
2
3 import java.util.*;
4
5 public class taking_input extends functions
6 {
7     static int ffeature = 45;
8     static int persons = 37;
9     static int realsign = 8;
10    static double [][] allsigns = new double [persons][realsign];
11    static double [][] person_under_consideration = new double [realsign][feature];
12    static int bitworand[] = new int [feature];
13    static int bitworand2[] = new int [feature];
14    static int bitworand3[] = new int [feature];
15    static int bitworand4[] = new int [feature];
16    static int bitworand5[] = new int [feature];
17    static int bitworand6[] = new int [feature];
18
19    static int tempbitworand[] = new int [feature];
20    static int tempbitworand2[] = new int [feature];
21    static int tempbitworand3[] = new int [feature];
22    static int tempbitworand4[] = new int [feature];
23    static int tempbitworand5[] = new int [feature];
24    static int tempbitworand6[] = new int [feature];
25    // max ad min functions
26    public static double find_max ( interior_node a)
27    {
28        double max = -10000.00;
29        double min = 10000.00;
30
31        for(int k=0;k<persons;k++)
32        {
33            for(int j=0;j<realsign;j++)
34            {
35                double temp = findvalue(feature,allsigns[k][j],a);
36                // System.out.println("printing feature"+temp);
37                if(temp>max)
38                {
39                    max = temp;
40                }
41            }
42        }
43        //System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
44        //System.out.println("printing max"+max);
45        //System.out.println("printing min"+min);
46        return max;
47    }
48
49    public static double find_min (interior_node a)
50    {
51        double max = -10000.00;
52        double min = 10000.00;
53
54        for(int k=0;k<persons;k++)
55        {
56            for(int j=0;j<realsign;j++)
57            {
58                double temp = findvalue(feature,allsigns[k][j],a);
59                if(temp>max)
60                {
61                    max = temp;
62                }
63                if(temp<min)
64                {
65                    min = temp;
66                }
67            }
68        }
69        //System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
70        //System.out.println("printing max"+max);
71        //System.out.println("printing min"+min);
72        return min;
73    }
74 }

```

```

54    public static double find_max ( interior_node a)
55    {
56        double max = -10000.00;
57        double min = 10000.00;
58
59        for(int k=0;k<persons;k++)
60        {
61            for(int j=0;j<realsign;j++)
62            {
63                double temp = findvalue(feature,allsigns[k][j],a);
64                if(temp>max)
65                {
66                    max = temp;
67                }
68                if(temp<min)
69                {
70                    min = temp;
71                }
72            }
73        }
74        //System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
75        //System.out.println("printing max"+max);
76        //System.out.println("printing min"+min);
77        return max;
78    }
79
80    public static double find_min (interior_node a)
81    {
82        double max = -10000.00;
83        double min = 10000.00;
84
85        for(int k=0;k<persons;k++)
86        {
87            for(int j=0;j<realsign;j++)
88            {
89                double temp = findvalue(feature,allsigns[k][j],a);
90                if(temp>max)
91                {
92                    max = temp;
93                }
94                if(temp<min)
95                {
96                    min = temp;
97                }
98            }
99        }
100        //System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
101        //System.out.println("printing max"+max);
102        //System.out.println("printing min"+min);
103        return min;
104    }
105 }

```

```

73     }
74 }
75 }
76 return min;
77 }
78 //end of max and min functions
79 public static void assignfeaturevalues() throws FileNotFoundException
80 {
81     // need to initialize them somewhere in a function . SO assign this function
82     for(int i=0;i<tempbitvorand1.length;i++)
83     {
84         tempbitvorand1[i] = 0;
85         tempbitvorand1[i] = 0;
86         tempbitvorand1[i] = 0;
87     }
88
89     //creating File instance to reference text file in Java
90     File text = new File("D:/Acad18/DOF18/input.txt");
91
92     //Creating Scanner instance to read File in Java
93     Scanner scnr = new Scanner(text);
94
95     //Reading each line of file using Scanner class
96
97
98
99     for(int t=0;t<persons;t++)
100     {
101         for(int j=0;j<crealsigns;j++)
102         {
103             for(int i=0;i<feature;i++)
104             {
105                 if(scnr.hasNextLine())
106                 {
107                     String line = scnr.nextLine();
108

```

```

106         if(scnr.hasNextLine())
107         {
108             String line = scnr.nextLine();
109             allsigns[t][j][i] = Double.parseDouble(line);
110         }
111     }
112 }
113 }
114 }
115
116 public static void copy_to_person_under_consider(int i)
117 {
118     for(int k=0;k<crealsigns;k++)
119     {
120         System.arraycopy(allsigns[i][k],0,person_under_consideration[k],0,allsigns[i][k].length);
121     }
122 }
123
124 public static interior_node find_initial_random_weights (interior_node a)
125 {
126     double lower = -1.0;//for random number generator
127     double upper = 1.0;//for random number generator
128     // filling bintvorand1 and 2
129     for(int i=0;i<feature;i++)
130     {
131         double random = Math.random() * (upper - lower) + lower;
132         if(random<0.5)
133         {
134             bintvorand1[i]=1;
135         }
136         else
137         {
138             bintvorand1[i]=0;
139         }
140     }
141     for(int i=0;i<feature;i++)

```

```

141 for(int i=0;i<feature;i++)
142 {
143     double random = Math.random() * (upper - lower) + lower;
144     if(randomca.alpha2[i]*a.alpha2[i])
145     {
146         bitvorand2[i]=1;
147     }
148     else
149     {
150         bitvorand2[i]=0;
151     }
152 }
153 // end of filling bintvorand1 and 2
154 //start of filling bintvorand3
155 for(int i=0;i<feature;i++)
156 {
157     double random = Math.random() * (upper - lower) + lower;
158     if(randomca.alpha3[i]*a.alpha3[i])
159     {
160         bitvorand3[i]=1;
161     }
162     else
163     {
164         bitvorand3[i]=0;
165     }
166 }
167
168 for(int i=0; i<feature;i++)
169 {
170     a.incomingedges[i]=findthefinalrandweight(bitvorand1[i], bitvorand2[i]);
171     // System.out.println("incoming now"+a.incomingedges[i]);
172 }
173 return a;
174
175
176
177

```

```

177 }
178 public static interior_node calculate_for_one( interior_node a)
179 {
180 }
181
182 //finding the alpha by doing for some iterations
183 int countactual=0;
184 int countfake = 0;
185 int optcountactual = 0;
186 int optcountfake = 0;
187 double [] tempincoming = new double[feature];
188 // for(double i=find_min(a);i<find_max(a);i=i+(find_max(a)-find_min(a))/1000)
189 for(double i=1000;i<=2000;i+=0.5)
190 {
191     countactual = 0;
192     countfake = 0;
193     for(int j=0;j<realSign;j++)
194     {
195         //System.out.println("printing findvalues for verification for other person "+findvalue(feature,person_under_consideration[j],a)+" eat
196         if((double) findvalue(feature,person_under_consideration[j],a)<1)
197         {
198             countactual++;
199         }
200     }
201 }
202
203 for(int k=0;k<persons;k++)
204 {
205     for(int j=0;j<realSign;j++)
206     {
207         //System.out.println("printing findvalues for verification for other person "+findvalue(feature,allSigns[k][j],a);
208         if(findvalue(feature,allSigns[k][j],a)>1)
209         {
210             countfake++;
211         }
212     }
213 }
214
215
216
217

```

```

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java taking_input.java functions.java interior_node.java
214
215
216 //System.out.println("This is another ifle   "+a.alpha+" values of countactual "+countactual+" countfake "+ countfake);
217 if(countactual>optcountactual && countfake> optcountfake)
218 {
219     a.alpha = 1;
220     optcountactual = countactual;
221     optcountfake = countfake;
222 }
223
224
225 //end of the alpha finding
226 for(int i=0; i<feature;i++)
227 {
228     // System.out.println("Just Features   "+a.incomingedges[i]+"   ");
229 }
230
231
232
233 int temp1 = optcountactual+optcountfake;
234 // System.out.println("This is another ifle   "+a.alpha+" values of countactual "+countactual+" countfake "+ countfake);
235 System.out.println("***** end *****");
236 return a;
237 }
238
239
240
241
242 /* public static interior_node calculate_for_great( interior_node a)
243 {
244
245     //finding the alpha by doing for some iterations
246     int countactual=0;
247     int countfake = 0;
248     int optcountactual = 0;
249     int optcountfake = 0;
250 }

```

```

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java taking_input.java functions.java interior_node.java
279 //start of updating using the table
280 public static interior_node updation( interior_node a)
281 {
282     int countactual=0;
283     int countfake = 0;
284     int optcountactual = 0;
285     int optcountfake = 0;
286     double degree = 0;
287     double [] optincomingedges = new double[feature];
288     double [] optincomingedges2 = new double[feature]; //remove it now
289     double [] tempalpha1 = new double[feature];
290     double [] tempalpha2 = new double[feature];
291     System.arraycopy(a.alpha,0,tempalpha1,0,a.alpha.length);
292     System.arraycopy(a.alpha2,0,tempalpha2,0,a.alpha2.length);
293     //
294     for(int hj=0;hj<1000;hj++)
295     {
296         if(hj!=0)
297         {
298             //copying from above function as I did not make a separate function in it
299
300
301             double lower = -1.0; //for random number generator
302             double upper = 1.0; //for random number generator
303             // filling hintworand1 and 2
304             for(int i=0;i<feature;i++)
305             {
306                 double random = Math.random() * (upper - lower) + lower;
307                 if(random< a.alpha1[i]*a.alpha1[i])
308                 {
309                     bitworand1[i]=1;
310                 }
311                 else
312                 {
313                     bitworand1[i]=0;
314                 }
315             }
316         }
317     }

```



```

313     {
314         bitvorand1[i]=0;
315     }
316 }
317
318 for(int i=0; i<feature; i++)
319 {
320     double random = Math.random() * (upper - lower) + lower;
321     if(random<a.alpha2[i]*a.alpha2[i])
322     {
323         bitvorand1[i]=1;
324     }
325     else
326     {
327         bitvorand1[i]=0;
328     }
329 }
330 // end of filling bitvorand1 and 2
331
332 for(int i=0; i<feature; i++)
333 {
334     a.incomingEdges[i]=findtheFinalrandweight(bitvorand1[i], bitvorand2[i])*bitvorand3[i];
335 }
336
337 //end of the copy from above function
338
339 }
340
341 for(int j=0; j<realSign; j++)
342 {
343     if(findvalue(feature, person_under_consideration[j], a)<a.alpha)
344     {
345         countactual++;
346     }
347 }
348

```

```

348     countactual++;
349 }
350
351 }
352 for(int k=0; k<persons && k!=a.class_classifying; k++)
353 {
354     for(int j=0; j<realSign; j++)
355     {
356         if(findvalue(feature, person_under_consideration[j], a)>a.alpha)
357         {
358             countfake++;
359         }
360     }
361 }
362
363
364 if(countactual+countfake >= optcountactual+optcountfake)
365 {
366     optcountactual = countactual;
367     optcountfake = countfake;
368
369     for(int ik=0; ik<feature; ik++)
370     {
371         if(tempbitvorand1[ik]==0 && bitvorand1[ik]==1)
372         {
373             degree = -0.03*(3.15);
374         }
375         if(tempbitvorand1[ik]==1 && bitvorand1[ik]==0)
376         {
377             degree = 0.03*(3.15);
378         }
379         else
380         {
381             degree = 0;
382         }
383     }
384 }
385

```

```

391         degree = -0.03*(3.15);
392     }
393     if(tempbitvrand2[ik]==1 && bitvrand2[ik]==0)
394     {
395         degree = 0.03*(3.15);
396     }
397     else
398     {
399         degree = 0;
400     }
401
402     a.alpha2[ik] = a.alpha2[ik]*(Math.sin(Math.toRadians(degree)))-Math.sqrt(1-a.alpha2[ik]*a.alpha2[ik])*(Math.cos(Math.toRadians(degree)));
403
404     //end for alpha2
405 }
406 for(int i=0;i<feature;i++)
407 {
408     tempbitvrand1[i]=bitvrand1[i];
409     tempbitvrand2[i]=bitvrand2[i];
410     optincomingEdges[i] = a.incomingEdges[i];
411     optincomingEdges2[i]=findtheFinalrandweight(tempbitvrand1[i], tempbitvrand2[i])*bitvrand2[i];
412 }
413
414 }
415
416 }
417 for(int i=0;i<feature;i++)
418 {
419     a.incomingEdges[i] = optincomingEdges[i];
420 }
421 return a;
422 }
423
424 }
425 }
426

```

```

1 package sig_ver;
2 import java.io.File;
3
4
5
6
7
8
9
10 public class main_place extends taking_input {
11     static int allcountyes=0;
12     static int allcountno=0;
13     static double[] features_for_testing = new double (45);
14     public static void taking_test_input_from_file( String a ) throws FileNotFoundException
15     {
16
17
18         //creating File instance to reference text file in Java
19         File text = new File(a);
20
21         //Creating Scanner instance to read File in Java
22         Scanner socr = new Scanner(text);
23
24         //Reading each line of file using Scanner class
25         int lineNumber = 1;
26
27         for(int t=0;t<45;t++)
28         {
29             if(socr.hasNextLine())
30             {
31                 String line = socr.nextLine();
32                 features_for_testing[t] = Double.parseDouble(line);
33             }
34         }
35     }
36
37 }
38 //end of taking input for testing function
39
40
41 public static int check(int [] array)
42 {
43
44     for(int i=0;i<array.length;i++)
45     {
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java functions.java interior_node.java
44 for(int i=0;i<array.length;i++)
45 {
46     if(array[i]!=1)
47     {
48         return 0;
49     }
50 }
51 return 1;
52 }
53
54 public static void main(String[] args) throws IOException
55 {
56     int max = 0;
57     Vector<interior_node> opt = new Vector<interior_node>(1);
58     int [] testpositive = new int [36];
59     int [] opttestpositive = new int [36];
60     int [] testnegative = new int [36][2];
61     int myface = 0;
62
63     for(int accu = 0; accu < 1;accu++){
64         allcountysave=0;
65         for(int y=0;y<36;y++){
66             testpositive[y]=0;
67         }
68         for(int ui=0;ui<36;ui++){
69             for(int uj=0;uj<2;uj++){
70                 testnegative[ui][uj]=0;
71             }
72         }
73         Vector<interior_node> vec = new Vector<interior_node>(1);
74         // interior_node [] my = new interior_node [10];
75
76         int class_under=0;
77         int in_class=0;
78         int fa = 45;
79         // taking_input t = new taking_input ();
80         int [] [] classifiers = new int [persons][realign];
81         for(int i=0;i< persons;i++)
82         {
83             for(int j=0;j<realign;j++)
84                 classifiers[i][j]=1;
85         }
86     }
87 }
88
89 while(class_under!=persons-1)
90 {
91     interior_node a = new interior_node();
92     a.class_classifying = class_under;
93     a.number_within_class= in_class;
94     for(int i=0; i<fa;i++)
95     {
96         a.alpha1[i]=0.707;
97         a.alpha2[i]=0.707;
98         a.alpha3[i]=0.707;
99         a.alpha = -200000.0;
100     }
101
102     assignfeaturevalues();
103     copy_to_person_under_consider(class_under);
104     a = find_initial_random_weights (a);
105     // a= calculate_for_great (a);
106     int countactual = 0;
107     int countfake = 0;
108     int optcountactual = 0;
109     int optcountfake = 0;
110     interior_node b = new interior_node();
111     b=calculate_for_one(a);
112     b=update(a);
113     for(int i=0;i<fa;i++)
114     {
115         System.out.println("Now class "+class_under+" is classified");
116         b=calculate_for_great(b);
117         b=update(b);
118         //b= calculate_for_great(a);
119         countactual = 0;
120         countfake = 0;
121         for(int j=0;j<realign;j++)
122         {
123             if(findvalue(feature_person_under_consideration[j],b)<=alpha)
124             {
125                 countactual++;
126             }
127         }
128         //System.out.println("printing findvalues for verification for actual person "+findvalue(feature_person_under_consideration[1],a));
129     }
130 }
131
132 }
133
134 }

```

```

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java functions.java interior_node.java
84 classifiers[i][j]=1;
85 }
86 while(class_under!=persons-1)
87 {
88     interior_node a = new interior_node();
89     a.class_classifying = class_under;
90     a.number_within_class= in_class;
91     for(int i=0; i<fa;i++)
92     {
93         a.alpha1[i]=0.707;
94         a.alpha2[i]=0.707;
95         a.alpha3[i]=0.707;
96         a.alpha = -200000.0;
97     }
98
99     assignfeaturevalues();
100     copy_to_person_under_consider(class_under);
101     a = find_initial_random_weights (a);
102     // a= calculate_for_great (a);
103     int countactual = 0;
104     int countfake = 0;
105     int optcountactual = 0;
106     int optcountfake = 0;
107     interior_node b = new interior_node();
108     b=calculate_for_one(a);
109     b=update(a);
110     for(int i=0;i<fa;i++)
111     {
112         System.out.println("Now class "+class_under+" is classified");
113         b=calculate_for_great(b);
114         b=update(b);
115         //b= calculate_for_great(a);
116         countactual = 0;
117         countfake = 0;
118         for(int j=0;j<realign;j++)
119         {
120             if(findvalue(feature_person_under_consideration[j],b)<=alpha)
121             {
122                 countactual++;
123             }
124         }
125         //System.out.println("printing findvalues for verification for actual person "+findvalue(feature_person_under_consideration[1],a));
126     }
127 }
128
129 }
130
131 }

```

```

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java functions.java interior_node.java
124 //System.out.println("printing findvalues for verification for actual person "+findvalue(feature, person_under_consideration[i], a));
125
126
127 for(int k=0; k<persons && k!=class_under; k++)
128 {
129     for(int j=0; j<real Signs; j++)
130     {
131         if(findvalue(feature, allsigns[k][j], b) > b.alpha)
132         {
133             countfake++;
134         }
135         //System.out.println("printing findvalues for verification for other person "+findvalue(feature, allsigns[k][j], a));
136     }
137 }
138
139 if(countactual > optcountactual && countfake > optcountfake)
140 {
141     s="b";
142     //System.out.println("i am entering this "+i+"value of count actual is "+countactual+" "+countfake+" "+optcountactual+" "+optcountfake+"n");
143     optcountactual = countactual;
144     optcountfake = countfake;
145 }
146 }
147 }
148 }
149 }
150
151
152 vec.add(a); // inserting it into the vector
153 myArrayList.remove(i);
154 //for(int i=0; i<45; i++)
155 {
156     System.out.println("newly added" + vec.get(vec.size()-1).incomingEdges[i]);
157 }
158 }
159 }
160
161 int flag= 0;
162 for(int i = 0; i<real Signs; i++)
163 {
164

```

```

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java functions.java interior_node.java
161 int flag= 0;
162 for(int i = 0; i<real Signs; i++)
163 {
164
165     if(findvalue(feature, person_under_consideration[i], a) < a.alpha)
166     {
167         //if(classifies(class_under[i])!=1){flag1 = 1;
168         classifies(class_under[i])=1;
169     }
170 }
171
172 if(check(classifies(class_under))!=1)
173 {
174     class_under++;
175     in_class = 0;
176     System.out.println("Now class "+class_under+" is classified");
177 }
178 }
179 else
180 {
181     in_class++;
182 }
183 }
184 //remove
185 for(int i=0; i<vec.size(); i++)
186 {
187     System.out.println("This is removed "+vec.get(i).alpha+" "+vec.get(i).class_classifying);
188 }
189 //
190
191 //testing the signatures int max = 0;
192
193
194
195 /* for(int y=0; y<100; y++)
196     allcountnes = 0;
197 //for(int g=0; g<8; g++)
198 for(int p=0; p<7; p++)
199 {
200     for(int ph=0; ph<2; ph++)
201     //for(int gh=0; gh<2; gh++)

```

```
eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
main_place.java functions.java interior_node.java
198 for(int p=0;p<7;p++)
199 {
200     for(int pk=0;pk<4;pk++)
201         //for(int ph=0;ph<2;ph++)
202         {
203             String at = "D:/Acad18/2018/attachments/";
204             String b = Integer.toString(p);
205             String c = Integer.toString(pk);
206             String d = at.concat(b);
207             d = d.concat(c);
208             d = d.concat(".txt");
209             taking_test_input_from_file(d);
210             int flag=0;
211             /* int class_to_be_checked;
212             Scanner in = new Scanner(System.in);
213             System.out.println("Enter a string");
214             class_to_be_checked = in.nextInt();*/
215             for(int i=0;i<vec.size();i++)
216             {
217                 if(vec.get(i).class_classifying==p)
218                 {
219                     System.out.println("This is class "+findValue(feature,feature_for_testing,vec.get(i))+ " "+vec.get(i).alpha+" "+vec.get(i).alphanegatives);
220                     if( findValue(feature,feature_for_testing,vec.get(i))>vec.get(i).alpha)
221                     {
222                         //System.out.println("Congrats. This belongs to the same class");
223                         // System.exit(0);
224                         allcountyes++;
225                         testpositive[p]++;
226                         testnegative[p][pk] = -1;
227                         System.out.println("This is good class "+p+" "+ allcountyes);
228                         flag = 1;
229                         break;
230                     }
231                 }
232             }
233         }
234     }
235 }
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
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
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755

```

