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

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.

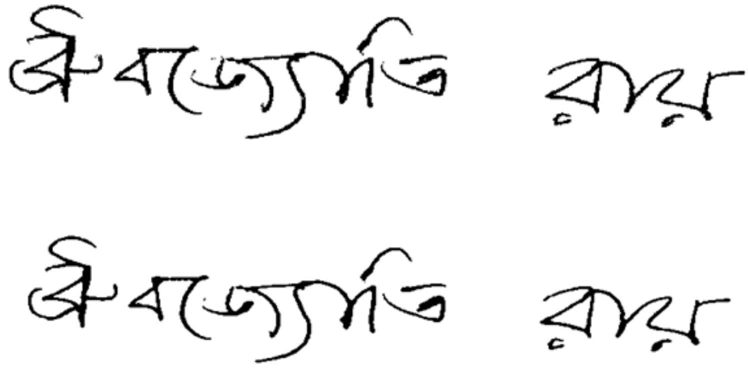


Figure 1: Top image is the original signal signature and bottom is after the median filtering

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].

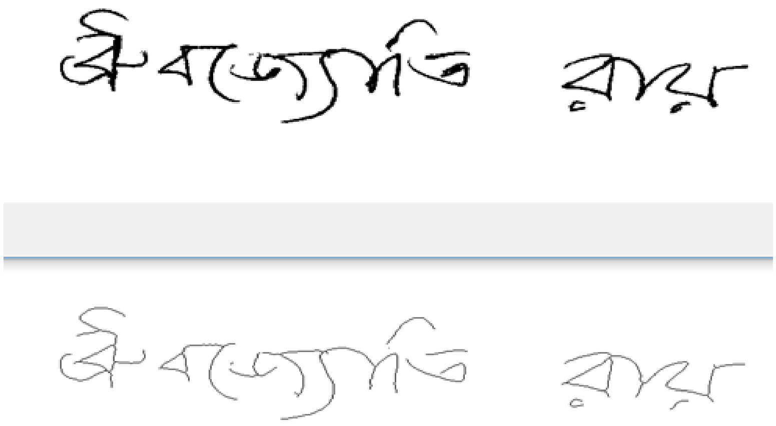


Figure 2: Top image is the original signal signature and bottom is after thinning

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.

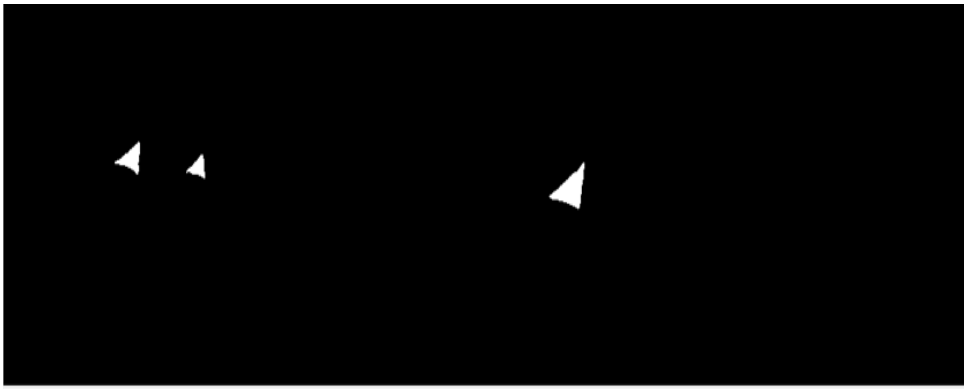


Figure 3: White Regions shows the loops

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 $P_1 = (X_1^1, X_1^2, ..., X_1^{c1})$ denote the input samples, where c_1 is the number of input samples and $X_l^i = (x_i^1, x_i^2, ..., x_i^e)$ 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:

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

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:

$$\left\langle \begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_q \\ \beta_1 & \beta_2 & \cdots & \beta_q \end{array} \right\rangle$$

where $0 \leq \alpha_i, \beta_i \leq 1$, $(\alpha_i)^2 + (\beta_i)^2 = 1$, and i = 1,2,....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 $(\alpha_i)^2 + (\beta_i)^2 = 1$,

$$\langle \alpha_1 | \alpha_2 | .... | \alpha_q \rangle \tag{2}$$

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 $r = [r_1, r_2, ..., r_q]$, where $0 \leq r_i \leq 1$, i =

1,2,....,q; the corresponding bit in b takes “1” if $r_i \le (\alpha_i)^2$, 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 λ_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 μ and variance σ, which can be represented as N(μ, σ).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 bin2dec(S_j) formula to select value from Gaussian random generator [5]. The value of matrix S_j is generated as follows:

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

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 λ_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 $-\inf \, and \inf$ respectvely. 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:

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

5 Algorithm

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

Step1: Take Input sample as $(X_1^1, X_1^2, X_1^3, ..., X_1^{c_1})$ and $(Y_1^1, Y_1^2, Y_1^3, ..., Y_1^{c_1})$ corresponding to each person. Take first neuron with the weights $W_g^{quant} = (Q_{w1}, Q_{w2}, ..., Q_{we})$ 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_$net_B = -\inf$
max_$net_B = -\inf$
Implement Conversion Process of W_g^{quant}
for i = 1 to c_1
$net_A(i) = \sum W_g^{real} x Y_1^i$
max_net_B = max(max_net_B, net_B(i))
min_net_B = min(min_net_B, net_B(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 = to c_1

net_A(i) = $\sum W_g$ x 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) = $\sum W_g$ x Y_1^i
if (net_A(i) $> Th_t^{real}$)
increase count2 by 1
endif
endfor
F_t = count1 + 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 Efficency	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

```
function loopcount = loops(image)
%image = imread('C:\Users\Arjun Gupta\Downloads\4-2\QNN\BHSig260\Bengali\001\B-S-1-G-01.tif');
%imshow(image)
image = medfilt2(image);
%figure, imshow(image)
image1 = image;
image1 = im2bw(image1, 0.5);
image1 = imcomplement(image1);
image1 = imfill(image1,'holes');
image2 = image;
image2 = im2bw(image2, 0.5);
image2 = imcomplement(image2);
imshow(image1-image2)
loopcount = max(max(bwlabel(image2-image1)));
```

Signature Preprocessing

```
clear all
clc
close all
srcFiles = dir('C:\Users\Arjun Gupta\Downloads\4-2\QNN\test1\*.tif');  % the folder in which ur images exists
dense_col = zeros(length(srcFiles),5);
dense_row = zeros(length(srcFiles),5);
features = zeros(length(srcFiles),13);
output_count = 1;
var_count = 1;

for si = 1 : length(srcFiles)
    filename = strcat('C:\Users\Arjun Gupta\Downloads\4-2\QNN\test1\',srcFiles(si).name);
    I = imread(filename);
%I=imread('signature.png');
figure,imshow(I);
%pause
loopcount = loops(I);
imshow(I)
I = medfilt2(I);    %removing noise

I2=imresize(I,[128 ,128]);
%figure, imshow(I2)
%figure,imshow(I2);
%pause
%I3=rgb2gray(I2);
I3=im2double(I2);
I3=im2bw(I3);
%figure, imshow(I)
```

mainnewton.m | main.m | data30.m | signature_preprocess.m | loops.m

```
%figure, imshow(I)

%converting image to black and white
I3 = bwmorph(~I3, 'thin', inf);                 %thining the image
I3=~I3;
figure,imshow(I3);


[rows cols] = size(I3);
top = 0;
bottom = 0;
left = 0;
right = 0;


k=1;
for i=1:128
    for j=1:128
        if(I3(i,j)==0)
            u(k)=i;
            v(k)=j;
            k=k+1;
            %I3(i,j)=1;
        end
    end
end
C=[u;v];%the curve of the signature
N=k-1;%the number of pixels in the signature
```

mainnewton.m | main.m | data30.m | signature_preprocess.m | loops.m

```
N=k-1;%the number of pixels in the signature
oub=sum(C(1,:))/N ;  %the original x co-ordinate center of mass of the image
ovb=sum(C(2,:))/N ;  %the original y co-ordinate center of mass of the image


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*******ROTATE******%%%%%%%%%%%%%%%%%%%%%%%%
%moving the signature to the origin
for i=1:N
    u(i)=u(i)-oub+1;
    v(i)=v(i)-ovb+1;
end
% the new curve of the signature
C=[u;v];

ub=sum(C(1,:))/N;
vb=sum(C(2,:))/N;
ubSq=sum((C(1,:)-ub).^2)/N;
vbSq=sum((C(2,:)-vb).^2)/N;

for i=1:N
    uv(i)=u(i)*v(i);
end

uvb=sum(uv)/N;
M=[ubSq uvb;uvb vbSq];
%calculating minimum igen value of the matrix
minIgen=min(abs(eig(M)));
%the eigen vector
```

mainnewton.m main.m data30.m signature_preprocess.m loops.m

```matlab
%the eigen vector
MI=[ubSq-minIgen uvb;uvb vbSq-minIgen];
theta(si)=(atan((-MI(1))/MI(2))*180)/pi;


thetaRad=(theta*pi)/180;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%cropping the picture to exact size of the image
for i=1:rows
    for j=1:cols
        if (I3(i,j)==0)
            top = i;
            break;
        end
    end
    if(top~=0)
        break;
    end
end

for i=rows:-1:1
    for j1=1:cols
        if (I3(i,j1)==0)
            bottom = i;
            break;
        end
    end
```

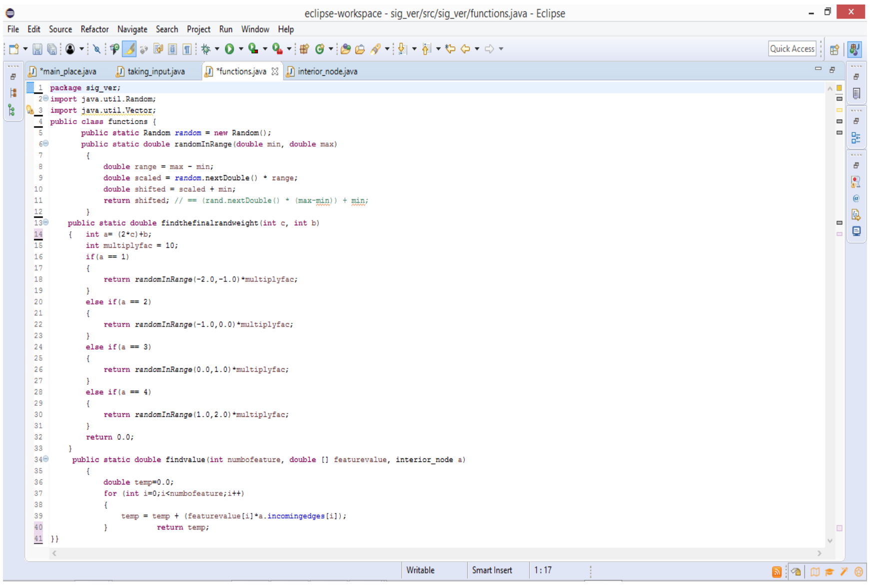
mainnewton.m main.m data30.m signature_preprocess.m* loops.m

```matlab
    end
    if(bottom~=0)
        break;
    end
end

for j=1:cols
    for i=1:rows
        if (I3(i,j)==0)
            left = j;
            break;
        end
    end
    if(left~=0)
        break;
    end
end

for j=cols:-1:1
    for i=1:rows
        if (I3(i,j)==0)
            right = j;
            break;
        end
    end
    if(right~=0)
        break;
    end
end
```

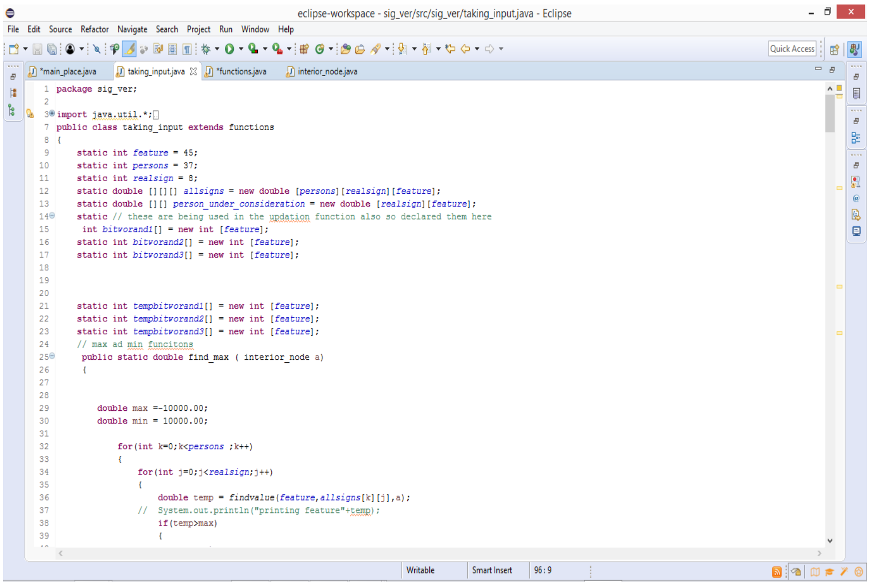
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%finiding 5 most dense rows and cols
countz = zeros(1,cols) ;
colz = 1;
for i=1:cols
    for j=1:rows
        if(I3(j,i)==0)
            countz(1,colz) = countz(1,colz)+1;
        end
    end
    colz=colz+1;
end

countz = countz./rows;

for i = 1:20
     [max_num,Y]=max(countz(:)) ;
    %[X Y]=ind2sub(size(countz),max_num);
    dense_col(si,i) = Y;
    countz(1,Y) = 0;
end

countz1 = zeros(1,rows) ;
rowz = 1;
for i=1:rows
    for j=1:cols
        if(I3(i,j)==0)
            countz1(1,rowz) = countz1(1,rowz)+1;
        end
```

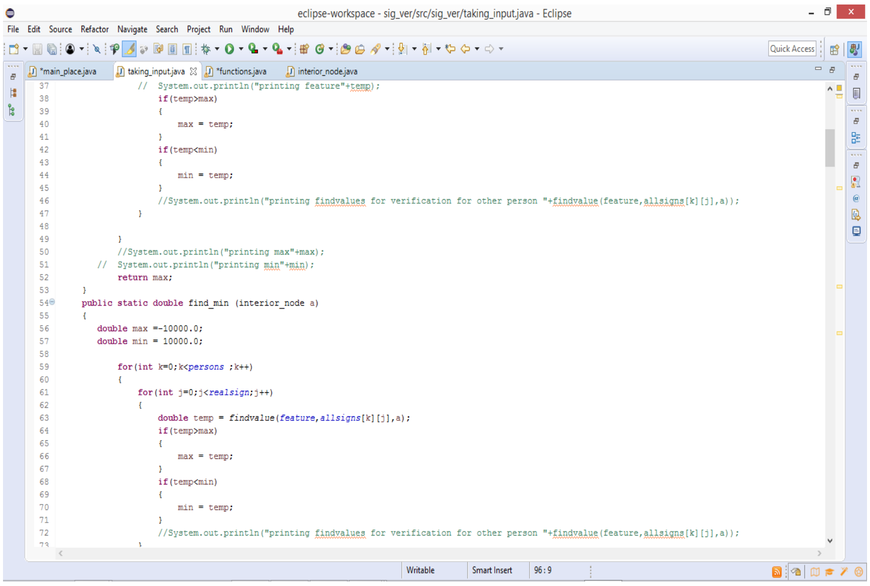
```
    end
    rowz=rowz+1;
end

countz1 = countz1./cols;

for i = 1:21
     [max_num,Y]=max(countz1(:)) ;
    %[X Y]=ind2sub(size(countz),max_num);
    dense_row(si,i) = Y;
    countz1(1,Y) = 0;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%storing all obtained features in a matrix
features(si,1) = theta(si);
features(si,2:3) = [oub,ovb];
features(si,4) = loopcount;
%features(si,5) = count;
features(si,5:45) = [dense_col(si,:), dense_row(si,:)];

I4 = I3(top:bottom, left:right);
figure,imshow(I4);
```

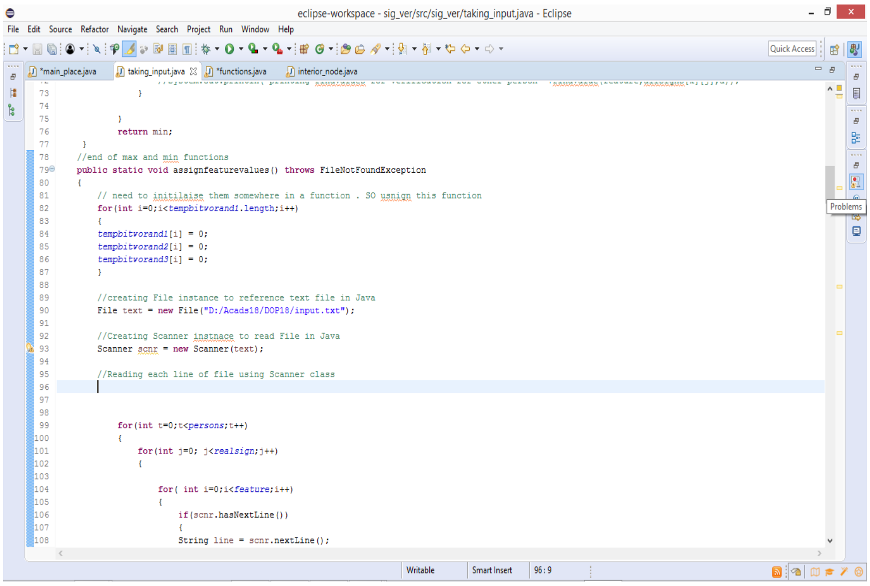
JAVA Code

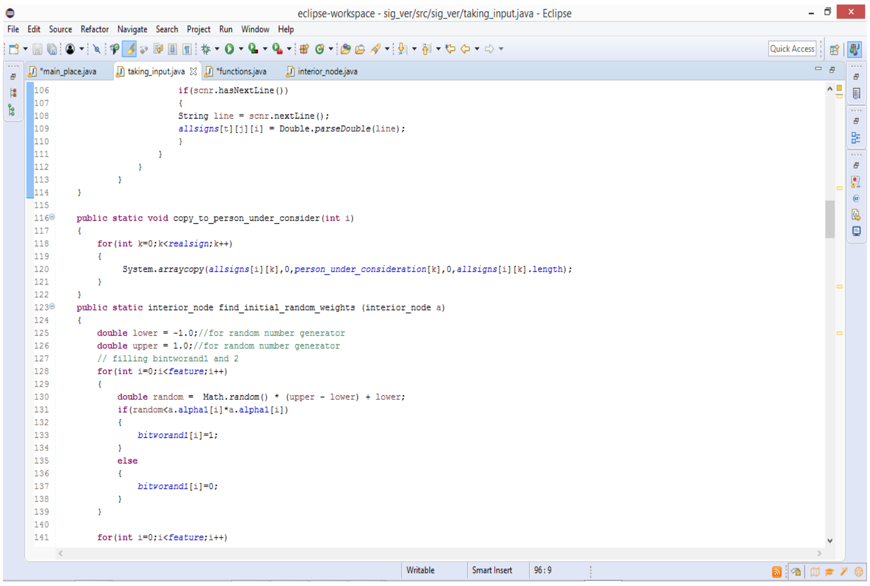


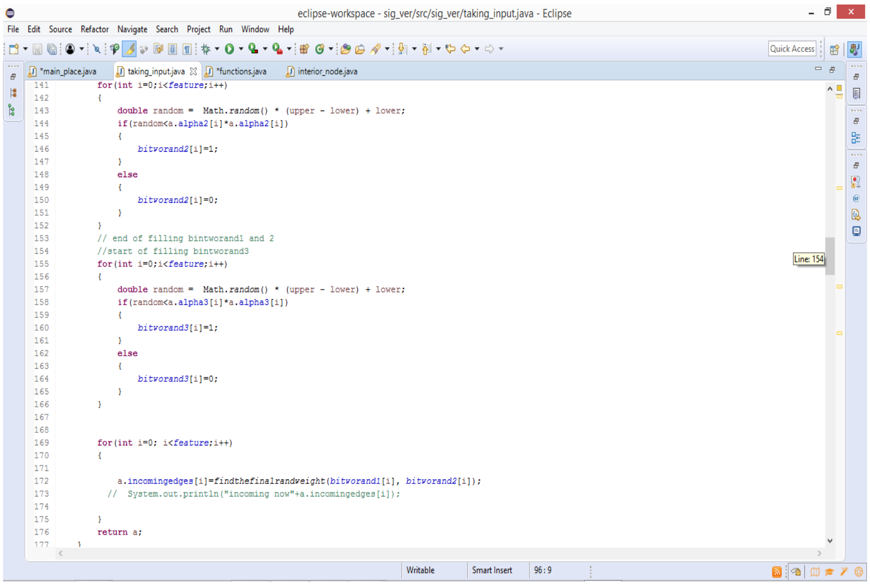

```java
package sig_ver;

import java.util.Vector;

public class interior_node {

      int featur = 45;
      double [] incomingedges = new double[featur];
         double value;
      double outgoingedge;
      int class_classifying;
      int number_within_class;
      double alpha;
      double alphagreater;
      double alpha1[] = new double[featur];
      double alpha2[] = new double[featur];
      double alpha3[] = new double[featur];
}

```

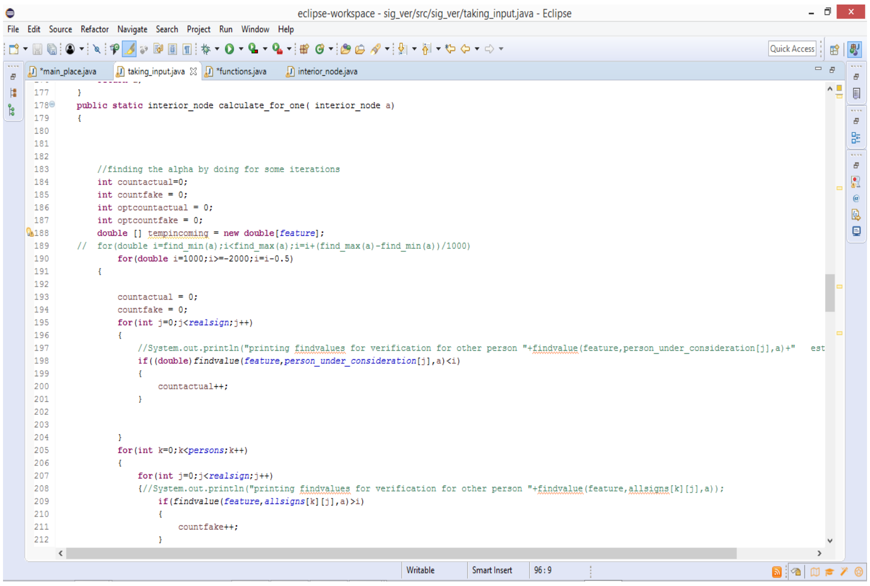

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
package sig_ver;
import java.util.*;
public class taking_input extends functions
{
static int feature = 45;
static int persons = 37;
static int realsign = 8;
static double [][][] allsigns = new double [persons][realsign][feature];
static double [][] person_under_consideration = new double [realsign][feature];
static // these are being used in the updation function also so declared them here
int bitvorand1[] = new int [feature];
static int bitvorand2[] = new int [feature];
static int bitvorand3[] = new int [feature];
static int tempbitvorand1[] = new int [feature];
static int tempbitvorand2[] = new int [feature];
static int tempbitvorand3[] = new int [feature];
// max ad min funcitons
public static double find_max (interior_node a)
{
double max =-10000.00;
double min = 10000.00;
for(int k=0;k<persons ;k++)
{
for(int j=0;j<realsign;j++)
{
double temp = findvalue(feature,allsigns[k][j],a);
// System.out.println("printing feature"+temp);
if(temp>max)
{
Writable
Smart Insert
96 : 9

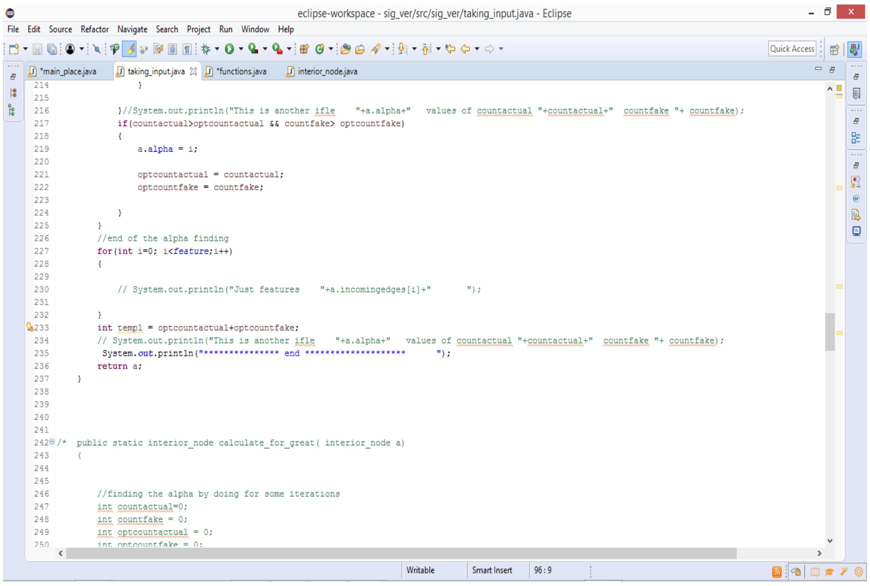

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
// System.out.println("printing feature"+temp);
if(temp>max)
{
max = temp;
}
if(temp<min)
{
min = temp;
}
//System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
}
}
//System.out.println("printing max"+max);
// System.out.println("printing min"+min);
return max;
}
public static double find_min (interior_node a)
{
double max =-10000.0;
double min = 10000.0;
for(int k=0;k<persons ;k++)
{
for(int j=0;j<realsign;j++)
{
double temp = findvalue(feature,allsigns[k][j],a);
if(temp>max)
{
max = temp;
}
if(temp<min)
{
min = temp;
}
//System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
Writable
Smart Insert
96 : 9

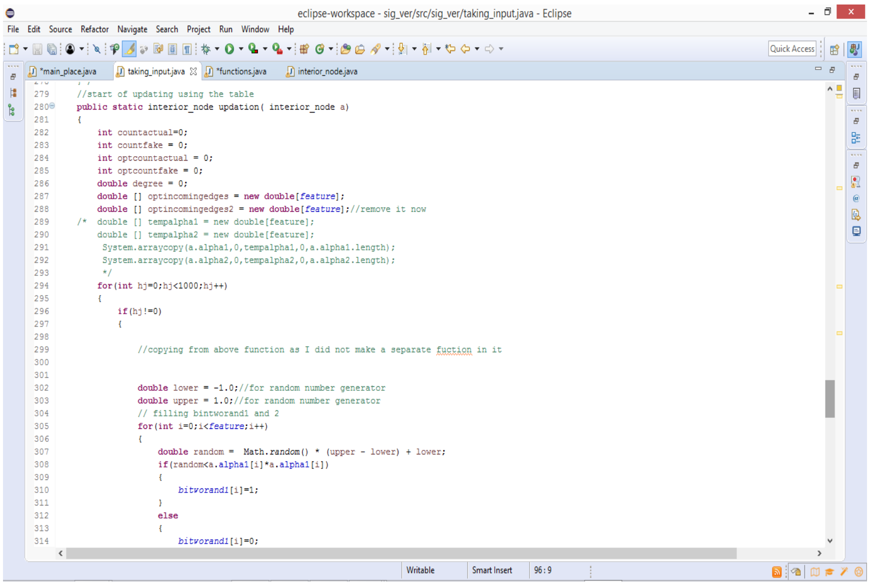

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
}
}
return min;
}
//end of max and min functions
public static void assignfeaturevalues() throws FileNotFoundException
{
// need to initilaise them somewhere in a function . SO usnign this function
for(int i=0;i<tempbitworand1.length;i++)
{
tempbitworand1[i] = 0;
tempbitworand2[i] = 0;
tempbitworand3[i] = 0;
}
//creating File instance to reference text file in Java
File text = new File("D:/Acads18/DOP18/input.txt");
//Creating Scanner instnace to read File in Java
Scanner scnr = new Scanner(text);
//Reading each line of file using Scanner class
for(int t=0;t<persons;t++)
{
for(int j=0; j<realsign;j++)
{
for(int i=0;i<feature;i++)
{
if(scnr.hasNextLine())
{
String line = scnr.nextLine();
Problems
Writable
Smart Insert
96 : 9

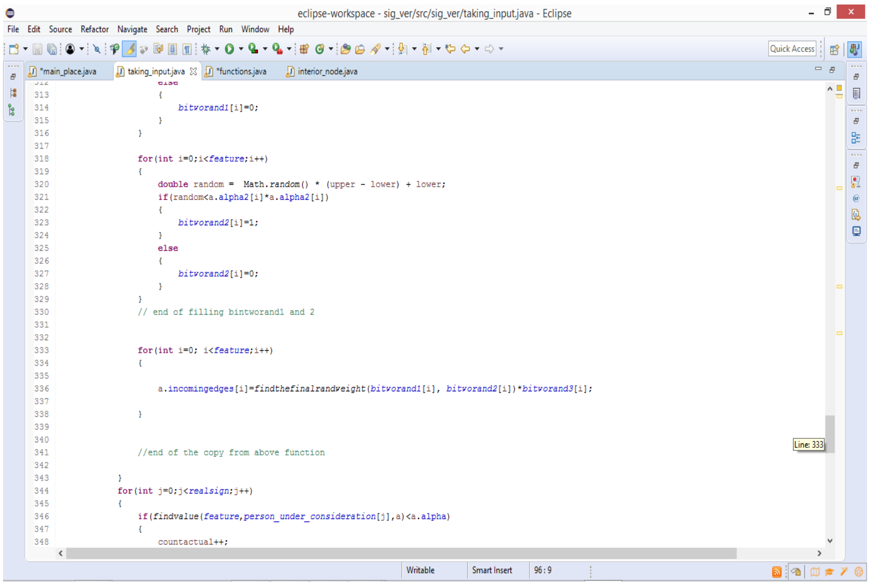

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
if(scnr.hasNextLine())
{
String line = scnr.nextLine();
allsigns[t][j][i] = Double.parseDouble(line);
}
}
}
}
}
public static void copy_to_person_under_consider(int i)
{
for(int k=0;k<realsign;k++)
{
System.arraycopy(allsigns[i][k],0,person_under_consideration[k],0,allsigns[i][k].length);
}
}
public static interior_node find_initial_random_weights (interior_node a)
{
double lower = -1.0;//for random number generator
double upper = 1.0;//for random number generator
// filling bintworand1 and 2
for(int i=0;i<feature;i++)
{
double random = Math.random() * (upper - lower) + lower;
if(random<a.alpha1[i]*a.alpha1[i])
{
bitworand1[i]=1;
}
else
{
bitworand1[i]=0;
}
}
for(int i=0;i<feature;i++)
Writable
Smart Insert
96 : 9


eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
for(int i=0;i<feature;i++)
{
double random = Math.random() * (upper - lower) + lower;
if(random<a.alpha2[i]*a.alpha2[i])
{
bitworand2[i]=1;
}
else
{
bitworand2[i]=0;
}
}
// end of filling bintworand1 and 2
//start of filling bintworand3
for(int i=0;i<feature;i++)
{
double random = Math.random() * (upper - lower) + lower;
if(random<a.alpha3[i]*a.alpha3[i])
{
bitworand3[i]=1;
}
else
{
bitworand3[i]=0;
}
}
for(int i=0; i<feature;i++)
{
a.incomingedges[i]=findthefinalrandweight(bitworand1[i], bitworand2[i]);
// System.out.println("incoming now"+a.incomingedges[i]);
}
return a;
}
Line: 154
Writable
Smart Insert
96 : 9

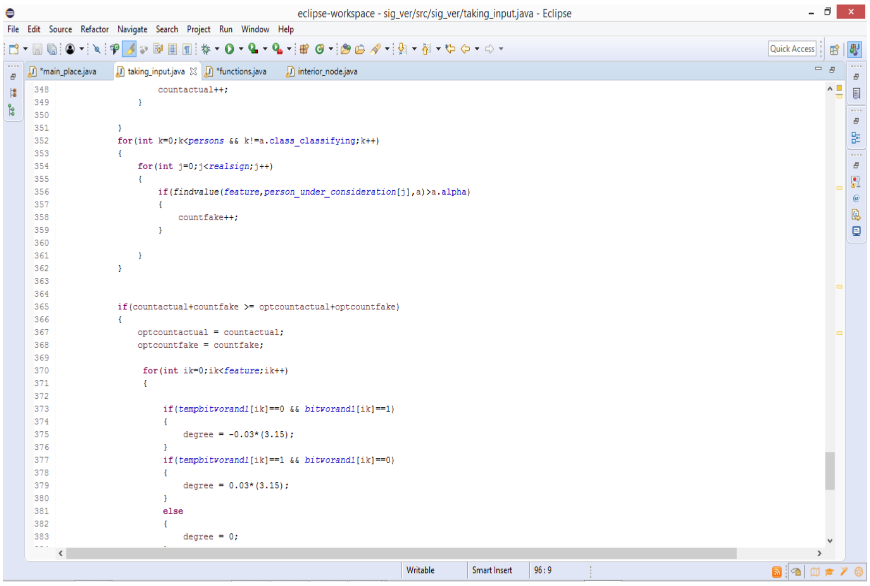

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
}
public static interior_node calculate_for_one(interior_node a)
{
//finding the alpha by doing for some iterations
int countactual=0;
int countfake = 0;
int optcountactual = 0;
int optcountfake = 0;
double [] tempincoming = new double[feature];
// for(double i=find_min(a);i<find_max(a);i=i+(find_max(a)-find_min(a))/1000)
for(double i=1000;i>=-2000;i=i-0.5)
{
countactual = 0;
countfake = 0;
for(int j=0;j<realsign;j++)
{
//System.out.println("printing findvalues for verification for other person "+findvalue(feature,person_under_consideration[j],a)+" est
if((double)findvalue(feature,person_under_consideration[j],a)<i)
{
countactual++;
}
}
for(int k=0;k<persons;k++)
{
for(int j=0;j<realsign;j++)
{//System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
if(findvalue(feature,allsigns[k][j],a)>i)
{
countfake++;
}
Writable
Smart Insert
96 : 9


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
}
}//System.out.println("This is another ifle   "+a.alpha+"   values of countactual "+countactual+" countfake "+ countfake);
if(countactual>optcountactual && countfake> optcountfake)
{
a.alpha = i;
optcountactual = countactual;
optcountfake = countfake;
}
}
//end of the alpha finding
for(int i=0; i<feature;i++)
{
// System.out.println("Just features   "+a.incomingedges[i]+"   ");
}
int temp1 = optcountactual+optcountfake;
// System.out.println("This is another ifle   "+a.alpha+"   values of countactual "+countactual+" countfake "+ countfake);
System.out.println("*************** end ********************   ");
return a;
}
/* public static interior_node calculate_for_great(interior_node a)
{
//finding the alpha by doing for some iterations
int countactual=0;
int countfake = 0;
int optcountactual = 0;
int optcountfake = 0;
Writable
Smart Insert
96 : 9

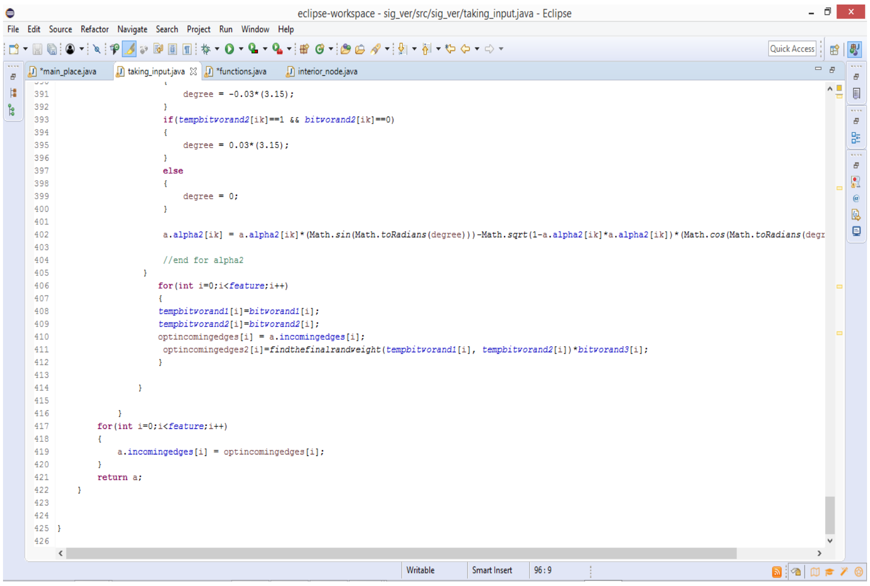

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
*main_place.java
taking_input.java
*functions.java
interior_node.java
//start of updating using the table
public static interior_node updation(interior_node a)
{
int countactual=0;
int countfake = 0;
int optcountactual = 0;
int optcountfake = 0;
double degree = 0;
double [] optincomingedges = new double[feature];
double [] optincomingedges2 = new double[feature];//remove it now
/* double [] tempalpha1 = new double[feature];
double [] tempalpha2 = new double[feature];
System.arraycopy(a.alpha1,0,tempalpha1,0,a.alpha1.length);
System.arraycopy(a.alpha2,0,tempalpha2,0,a.alpha2.length);
*/
for(int hj=0;hj<1000;hj++)
{
if(hj!=0)
{
//copying from above function as I did not make a separate fuction in it
double lower = -1.0;//for random number generator
double upper = 1.0;//for random number generator
// filling bintworand1 and 2
for(int i=0;i<feature;i++)
{
double random = Math.random() * (upper - lower) + lower;
if(random<a.alpha1[i]*a.alpha1[i])
{
bitworand1[i]=1;
}
else
{
bitworand1[i]=0;
Writable
Smart Insert
96 : 9

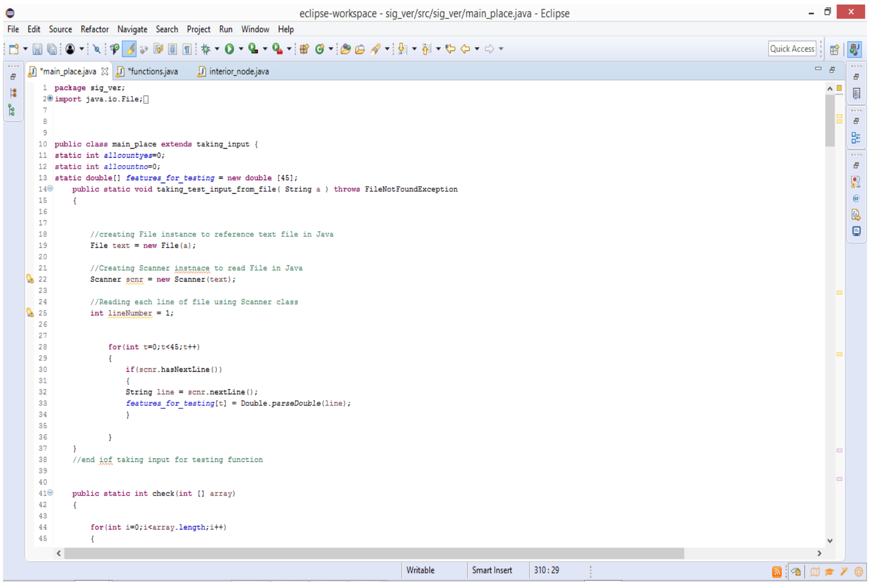


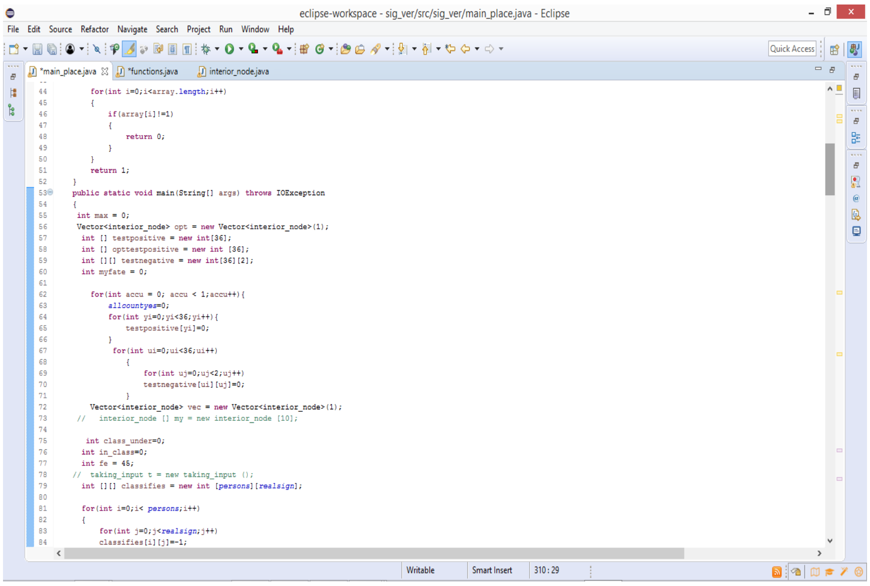
eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
{
bitworand1[i]=0;
}
}
for(int i=0;i<feature;i++)
{
double random = Math.random() * (upper - lower) + lower;
if(random<a.alpha2[i]*a.alpha2[i])
{
bitworand2[i]=1;
}
else
{
bitworand2[i]=0;
}
}
// end of filling bintworand1 and 2
for(int i=0; i<feature;i++)
{
a.incomingedges[i]=findthefinalrandweight(bitworand1[i], bitworand2[i])*bitworand3[i];
}
//end of the copy from above function
}
for(int j=0;j<realsign;j++)
{
if(findvalue(feature,person_under_consideration[j],a)<a.alpha)
{
countactual++;
Line: 333
Writable
Smart Insert
96 : 9

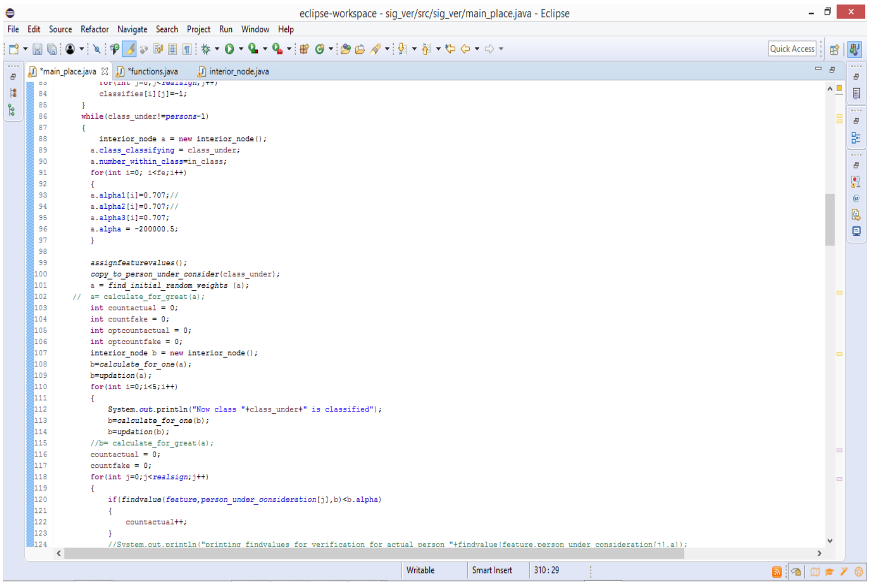


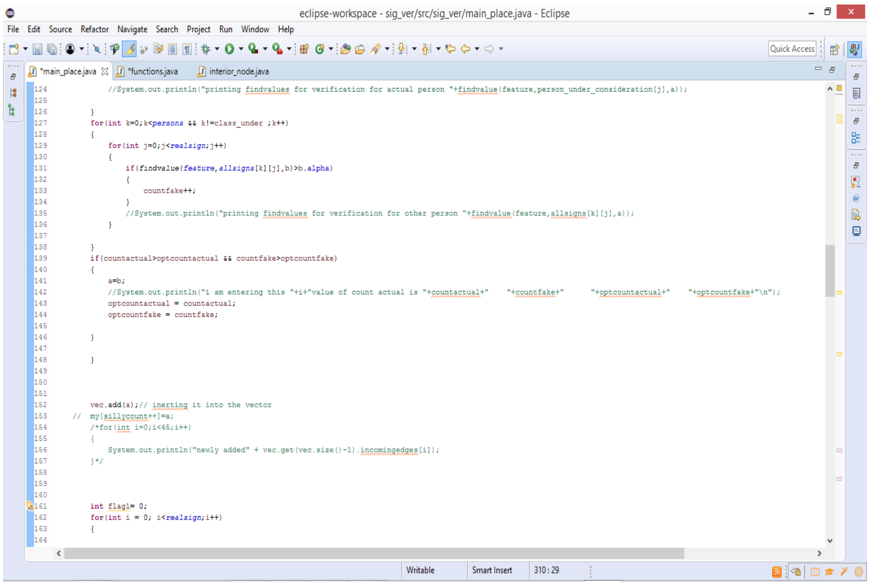
eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
countactual++;
}
}
for(int k=0;k<persons && k!=a.class_classifying;k++)
{
for(int j=0;j<realsign;j++)
{
if(findvalue(feature,person_under_consideration[j],a)>a.alpha)
{
countfake++;
}
}
}
if(countactual+countfake >= optcountactual+optcountfake)
{
optcountactual = countactual;
optcountfake = countfake;
for(int ik=0;ik<feature;ik++)
{
if(tempbitworand1[ik]==0 && bitworand1[ik]==1)
{
degree = -0.03*(3.15);
}
if(tempbitworand1[ik]==1 && bitworand1[ik]==0)
{
degree = 0.03*(3.15);
}
else
{
degree = 0;
Writable
Smart Insert
96 : 9

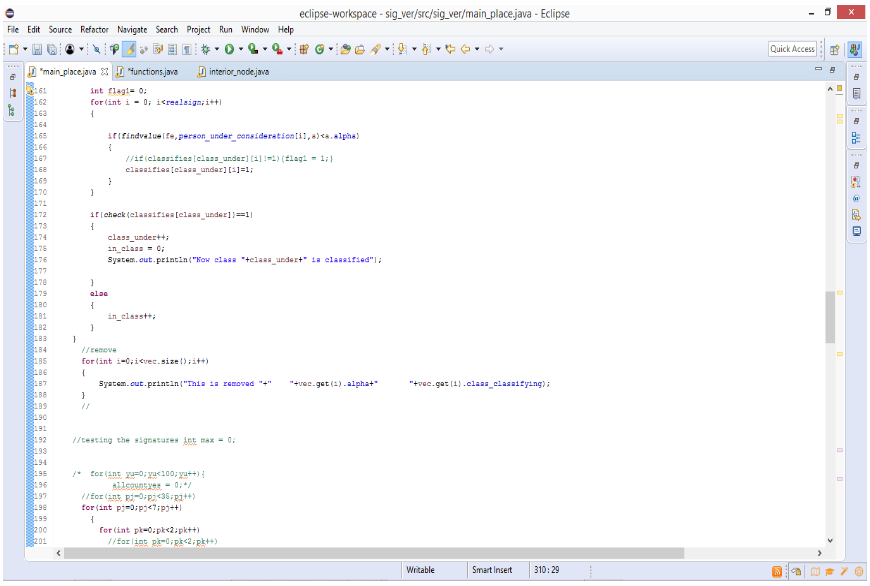

eclipse-workspace - sig_ver/src/sig_ver/taking_input.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
taking_input.java
*functions.java
interior_node.java
degree = -0.03*(3.15);
}
if(tempbitworand2[ik]==1 && bitworand2[ik]==0)
{
degree = 0.03*(3.15);
}
else
{
degree = 0;
}
a.alpha2[ik] = a.alpha2[ik]*(Math.sin(Math.toRadians(degree)))-Math.sqrt(1-a.alpha2[ik]*a.alpha2[ik])*(Math.cos(Math.toRadians(degr
//end for alpha2
}
for(int i=0;i<feature;i++)
{
tempbitworand1[i]=bitworand1[i];
tempbitworand2[i]=bitworand2[i];
optincomingedges[i] = a.incomingedges[i];
optincomingedges2[i]=findthefinalrandweight(tempbitworand1[i], tempbitworand2[i])*bitworand3[i];
}
}
}
for(int i=0;i<feature;i++)
{
a.incomingedges[i] = optincomingedges[i];
}
return a;
}
}
Writable
Smart Insert
96 : 9

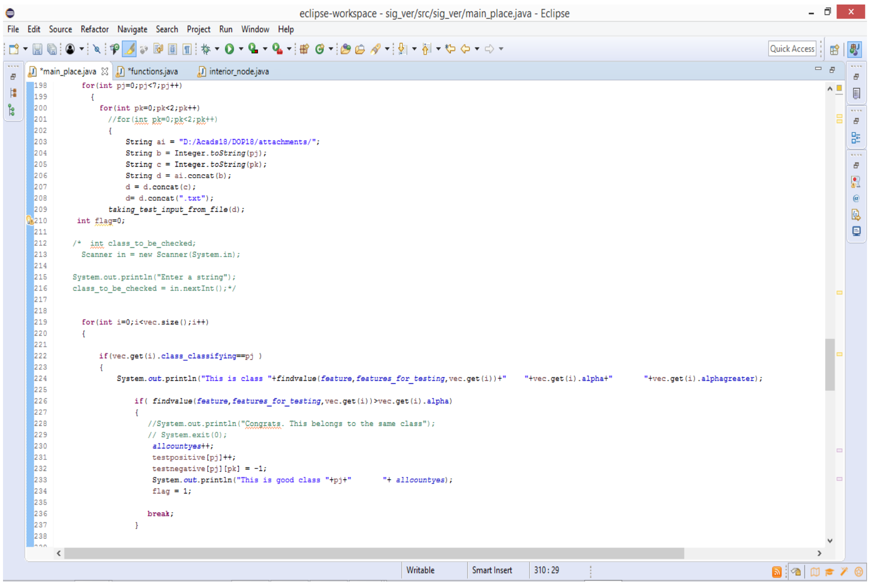

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
package sig_ver;
import java.io.File;
public class main_place extends taking_input {
static int allcountyes=0;
static int allcountno=0;
static double[] features_for_testing = new double [45];
public static void taking_test_input_from_file(String a) throws FileNotFoundException
{
//creating File instance to reference text file in Java
File text = new File(a);
//Creating Scanner instnace to read File in Java
Scanner scnr = new Scanner(text);
//Reading each line of file using Scanner class
int lineNumber = 1;
for(int t=0;t<45;t++)
{
if(scnr.hasNextLine())
{
String line = scnr.nextLine();
features_for_testing[t] = Double.parseDouble(line);
}
}
}
//end iof taking input for testing function
public static int check(int [] array)
{
for(int i=0;i<array.length;i++)
{
Writable
Smart Insert
310 : 29

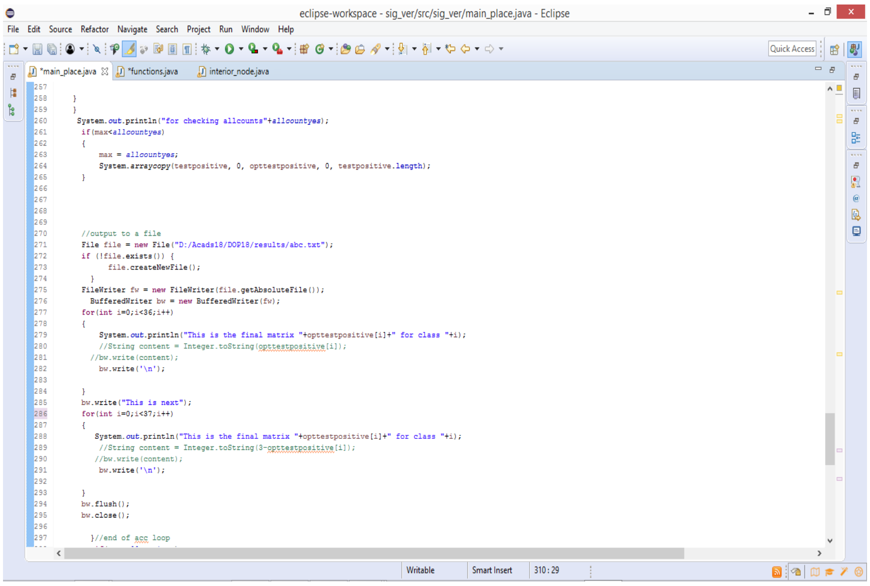

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
for(int i=0;i<array.length;i++)
{
if(array[i]!=1)
{
return 0;
}
}
return 1;
}
public static void main(String[] args) throws IOException
{
int max = 0;
Vector<interior_node> opt = new Vector<interior_node>(1);
int [] testpositive = new int[36];
int [] opttestpositive = new int [36];
int [][] testnegative = new int[36][2];
int myfate = 0;
for(int accu = 0; accu < 1;accu++){
allcountyes=0;
for(int yi=0;yi<36;yi++){
testpositive[yi]=0;
}
for(int ui=0;ui<36;ui++)
{
for(int uj=0;uj<2;uj++)
testnegative[ui][uj]=0;
}
Vector<interior_node> vec = new Vector<interior_node>(1);
// interior_node [] my = new interior_node [10];
int class_under=0;
int in_class=0;
int fe = 45;
// taking_input t = new taking_input ();
int [][] classifies = new int [persons][realsign];
for(int i=0;i< persons;i++)
{
for(int j=0;j<realsign;j++)
classifies[i][j]=-1;
Writable
Smart Insert
310 : 29

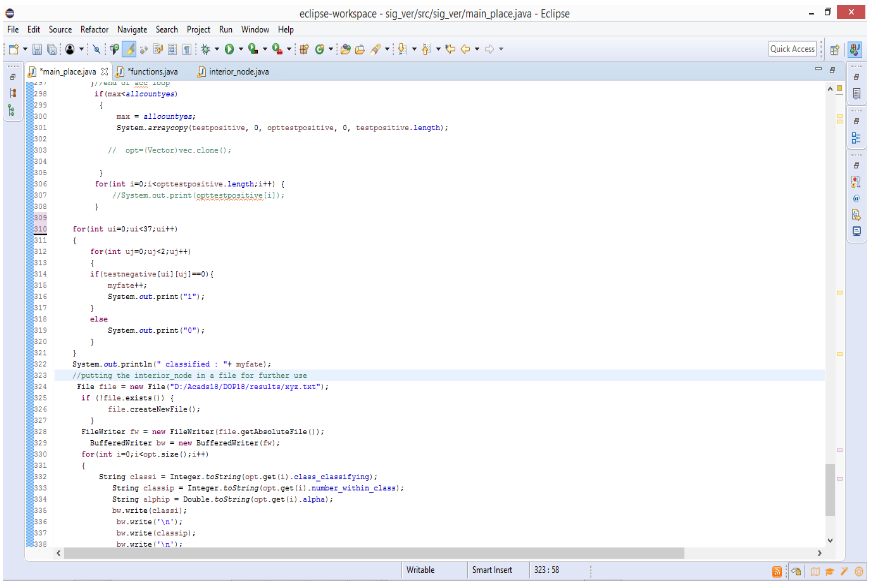

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
classifies[i][j]=-1;
}
while(class_under!=persons-1)
{
interior_node a = new interior_node();
a.class_classifying = class_under;
a.number_within_class=in_class;
for(int i=0; i<fe;i++)
{
a.alpha1[i]=0.707;//
a.alpha2[i]=0.707;//
a.alpha3[i]=0.707;
a.alpha = -200000.5;
}
assignfeaturevalues();
copy_to_person_under_consider(class_under);
a = find_initial_random_weights (a);
// a= calculate_for_great(a);
int countactual = 0;
int countfake = 0;
int optcountactual = 0;
int optcountfake = 0;
interior_node b = new interior_node();
b=calculate_for_one(a);
b=updation(a);
for(int i=0;i<5;i++)
{
System.out.println("Now class "+class_under+" is classified");
b=calculate_for_one(b);
b=updation(b);
//b= calculate_for_great(a);
countactual = 0;
countfake = 0;
for(int j=0;j<realsign;j++)
{
if(findvalue(feature,person_under_consideration[j],b)<b.alpha)
{
countactual++;
}
//System.out.println("printing findvalues for verification for actual person "+findvalue(feature,person_under_consideration[j],a));
Writable
Smart Insert
310 : 29

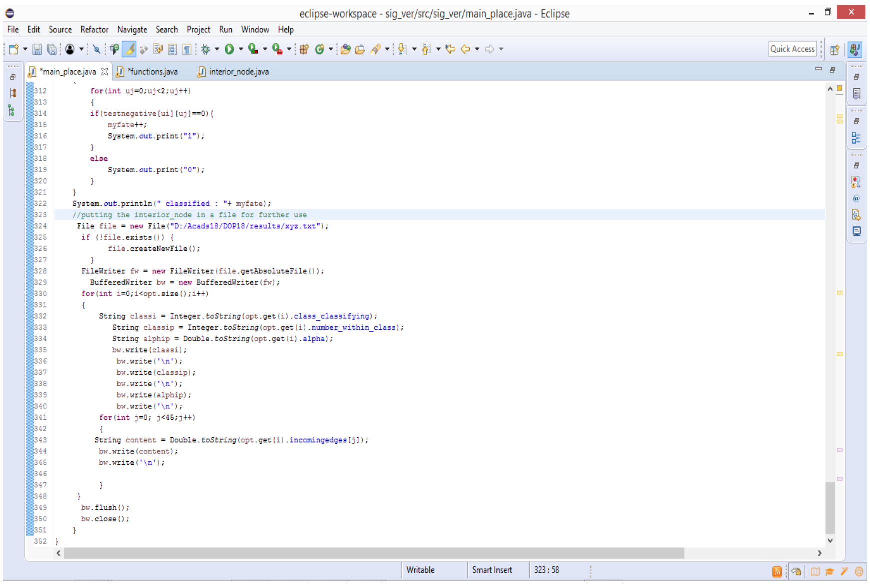

eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
//System.out.println("printing findvalues for verification for actual person "+findvalue(feature,person_under_consideration[j],a));
}
for(int k=0;k<persons && k!=class_under ;k++)
{
for(int j=0;j<realsign;j++)
{
if(findvalue(feature,allsigns[k][j],b)>b.alpha)
{
countfake++;
}
//System.out.println("printing findvalues for verification for other person "+findvalue(feature,allsigns[k][j],a));
}
}
if(countactual>optcountactual && countfake>optcountfake)
{
a=b;
//System.out.println("i am entering this "+i+"value of count actual is "+countactual+"   "+countfake+"   "+optcountactual+"   "+optcountfake+"\n");
optcountactual = countactual;
optcountfake = countfake;
}
}
vec.add(a);// inerting it into the vector
// my[sillycount++]=a;
/*for(int i=0;i<45;i++)
{
System.out.println("newly added" + vec.get(vec.size()-1).incomingedges[i]);
}*/
int flag1= 0;
for(int i = 0; i<realsign;i++)
{
Writable
Smart Insert
310 : 29


eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
int flag1= 0;
for(int i = 0; i<realsign;i++)
{
if(findvalue(fe,person_under_consideration[i],a)<a.alpha)
{
//if(classifies[class_under][i]!=1){flag1 = 1;}
classifies[class_under][i]=1;
}
}
if(check(classifies[class_under])==1)
{
class_under++;
in_class = 0;
System.out.println("Now class "+class_under+" is classified");
}
else
{
in_class++;
}
}
//remove
for(int i=0;i<vec.size();i++)
{
System.out.println("This is removed "+"   "+vec.get(i).alpha+"   "+vec.get(i).class_classifying);
}
//
//testing the signatures int max = 0;
/* for(int yu=0;yu<100;yu++){
allcountyes = 0;*/
//for(int pj=0;pj<35;pj++)
for(int pj=0;pj<7;pj++)
{
for(int pk=0;pk<2;pk++)
//for(int pk=0;pk<2;pk++)
Writable
Smart Insert
310 : 29


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
Writable
Smart Insert
310 : 29


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
Writable
Smart Insert
310 : 29


eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
Writable
Smart Insert
323 : 58


eclipse-workspace - sig_ver/src/sig_ver/main_place.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*main_place.java
*functions.java
interior_node.java
Writable
Smart Insert
323 : 58