|  |
| --- |
|  |

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| SN | Topic | Page no. |
| 1. | Member-1(Sumaiya Ahmed Sushmi, ID-20201058010) |  |
|  | 1. Adaptive Decision Boundary |  |
|  | 1. Single Linkage Algorithm |  |
|  | 1. DDA Line drawing |  |
|  | 1. Bresenham Line Drawing |  |
|  | 1. Bresenham Circle Drawing |  |
|  | 1. Mid-Point Circle Drawing |  |
| 2. | Member-2(Isfat Ara Hasan Ema, ID-20201112010) |  |
|  | 1. Adaptive Decision Boundary |  |
|  | 1. Single Linkage Algorithm |  |
|  | 1. DDA Line drawing |  |
|  | 1. Bresenham Line Drawing |  |
|  | 1. Bresenham Circle Drawing |  |
|  | 1. Mid-Point Circle Drawing |  |
| 3 | Member-3(Fatema Zahan Shayla, ID-20201059010) |  |
|  | 1. Adaptive Decision Boundary |  |
|  | 1. Single Linkage Algorithm |  |
|  | 1. DDA Line drawing |  |
|  | 1. Bresenham Line Drawing |  |
|  | 1. Bresenham Circle Drawing |  |
|  | 1. Mid-Point Circle Drawing |  |

*NORTH WESTERN UNIVERSITY, KHULNA*

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab Report***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Sumaiya Ahmed Susmi***  ***Id: 20201058010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

1. **Algorithm Name: Adaptive Decision Boundary**

Code:

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

class Perceptron {

private:

vector<double> weights;

double learningRate;

public:

Perceptron(int inputSize, double lr) : learningRate(lr) {

srand(static\_cast<unsigned int>(time(0)));

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

weights.push\_back(static\_cast<double>(rand()) / RAND\_MAX);

}

}

int predict(const vector<double>& inputs) const {

double sum = 0.0;

for (size\_t i = 0; i < inputs.size(); ++i) {

sum += weights[i] \* inputs[i];

}

return (sum >= 0.0) ? 1 : -1;

}

void train(const vector<vector<double>>& trainingData, const vector<int>& labels, int maxEpochs) {

for (int epoch = 0; epoch < maxEpochs; ++epoch) {

for (size\_t i = 0; i < trainingData.size(); ++i) {

int prediction = predict(trainingData[i]);

int error = labels[i] - prediction;

for (size\_t j = 0; j < weights.size(); ++j) {

weights[j] += learningRate \* error \* trainingData[i][j];

}

}

}

}

const vector<double>& getWeights() const {

return weights;

}

};

int main() {

vector<vector<double>> trainingData = {{2, 3}, {4, 5}, {1, 1}, {5, 2}};

vector<int> labels = {1, 1, -1, -1};

Perceptron perceptron(2, 0.1);

perceptron.train(trainingData, labels, 1000);

const vector<double>& weights = perceptron.getWeights();

cout << "Learned Weights: ";

for (size\_t i = 0; i < weights.size(); ++i) {

cout << weights[i] << " ";

}

cout << endl;

vector<vector<double>> testData = {{3, 4}, {1, 2}};

for (size\_t i = 0; i < testData.size(); ++i) {

int prediction = perceptron.predict(testData[i]);

cout << "Prediction for [" << testData[i][0] << ", " << testData[i][1] << "]: " << prediction << endl;

}

return 0;

}

1. **Algorithm Name: Single Linkage Algorithm**

Code:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

class CustomHierarchicalClustering {

private:

vector<vector<double>> inputData;

vector<vector<double>> distanceMatrix;

public:

CustomHierarchicalClustering(const vector<vector<double>>& inputPoints) : inputData(inputPoints) {

initializeDistanceMatrix();

}

void initializeDistanceMatrix() {

size\_t numPoints = inputData.size();

distanceMatrix.resize(numPoints, vector<double>(numPoints, 0.0));

for (size\_t i = 0; i < numPoints; ++i) {

for (size\_t j = i + 1; j < numPoints; ++j) {

double distance = calculateEuclideanDistance(inputData[i], inputData[j]);

distanceMatrix[i][j] = distance;

distanceMatrix[j][i] = distance;

}

}

}

double calculateEuclideanDistance(const vector<double>& point1, const vector<double>& point2) const {

double sum = 0.0;

for (size\_t i = 0; i < point1.size(); ++i) {

sum += pow(point1[i] - point2[i], 2);

}

return sqrt(sum);

}

pair<size\_t, size\_t> findClosestClusters() const {

size\_t numPoints = distanceMatrix.size();

pair<size\_t, size\_t> minDistanceClusters = {0, 1};

double minDistance = distanceMatrix[0][1];

for (size\_t i = 0; i < numPoints; ++i) {

for (size\_t j = i + 1; j < numPoints; ++j) {

if (distanceMatrix[i][j] < minDistance) {

minDistance = distanceMatrix[i][j];

minDistanceClusters = {i, j};

}

}

}

return minDistanceClusters; }

void updateDistanceMatrix(const pair<size\_t, size\_t>& clusters) {

size\_t numPoints = distanceMatrix.size();

for (size\_t i = 0; i < numPoints; ++i) {

if (i != clusters.first && i != clusters.second) {

distanceMatrix[i][clusters.first] = min(distanceMatrix[i][clusters.first], distanceMatrix[i][clusters.second]);

distanceMatrix[clusters.first][i] = distanceMatrix[i][clusters.first];

}

}

for (size\_t i = 0; i < numPoints; ++i) {

distanceMatrix[i].erase(distanceMatrix[i].begin() + clusters.second);

}

distanceMatrix.erase(distanceMatrix.begin() + clusters.second);

}

void performHierarchicalClustering() {

size\_t numPoints = distanceMatrix.size();

while (numPoints > 1) {

pair<size\_t, size\_t> clusters = findClosestClusters();

updateDistanceMatrix(clusters);

cout << "Merged clusters " << clusters.first << " and " << clusters.second << ", New Distance Matrix:" << endl;

printDistanceMatrix();

--numPoints;

}

}

void printDistanceMatrix() const {

for (const auto& row : distanceMatrix) {

for (double distance : row) {

cout << distance << " ";

}

cout << endl;

}

cout << endl;

}

};

int main() {

vector<vector<double>> inputPoints = {{1, 2}, {5, 8}, {1.5, 1.8}, {8, 8}, {1, 0.6}, {9, 11}};

CustomHierarchicalClustering hierarchicalClustering(inputPoints);

cout << "Initial Distance Matrix:" << endl;

hierarchicalClustering.printDistanceMatrix();

hierarchicalClustering.performHierarchicalClustering();

return 0;

}

1. **Algorithm Name: DDA Line generation Algorithm**

Code:

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <iostream.h>

int main(void)

{

int gdriver = DETECT, gmode, errorcode;

initgraph( & gdriver, & gmode, "C:\\tc\\bgi");

cout << "\n Enter X1,Y1,X2,Y2";

int x1, y1, x2, y2;

cin >> x1 >> y1 >> x2 >> y2;

int dx = x2 - x1;

int dy = y2 - y1;

int length;

if (dx >= dy)

length = dx;

else

length = dy;

dx = dx / length;

dy = dy / length;

int sx;

if (dx >= 0)

sx = 1;

else

sx = -1;

int sy;

if (dy >= 0)

sy = 1;

else

sy = -1;

float x = x1 + 0.5 \* (sx);

float y = y1 + 0.5 \* (sy);

int i = 0;

while (i <= length)

{

putpixel(int(x), int(y), 15);

x = x + dx;

y = y + dy;

i = i + 1;

}

getch();

closegraph();

}

1. **Algorithm Name: Bresenham's Line Algorithm**

Code:

#include<iostream.h>

#include<graphics.h>

void drawline(int x0, int y0, int x1, int y1)

{

int dx, dy, p, x, y;

dx=x1-x0;

dy=y1-y0;

x=x0;

y=y0;

p=2\*dy-dx;

while(x<x1)

{

if(p>=0)

{

putpixel(x,y,7);

y=y+1;

p=p+2\*dy-2\*dx;

}

else

{

putpixel(x,y,7);

p=p+2\*dy;

}

x=x+1;

}

}

int main()

{

int gdriver=DETECT, gmode, error, x0, y0, x1, y1;

initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

cout<<"Enter co-ordinates of first point: ";

cin>>x0>>y0;

cout<<"Enter co-ordinates of second point: ";

cin>>x1>>y1;

drawline(x0, y0, x1, y1);

return 0;

}

1. **Algorithm Name: Bresenham’s circle drawing algorithm**

Code:

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

void drawCircle(int x, int y, int xc, int yc);

void main()

{

int gd = DETECT, gm;

int r, xc, yc, pk, x, y;

initgraph(&gd, &gm, "C:TCBGI");

cout<<"Enter the center co-ordinates\n";

cin>>xc>>yc;

cout<<"Enter the radius of circle\n";

cin>>r;

pk = 3 - 2\*r;

x=0; y = r;

drawCircle(x,y,xc,yc);

while(x < y)

{

if(pk <= 0)

{

pk = pk + (4\*x) + 6;

drawCircle(++x,y,xc,yc);

}

else

{

pk = pk + (4\*(x-y)) + 10;

drawCircle(++x,--y,xc,yc);

}

}

getch();

closegraph();

}

void drawCircle(int x, int y, int xc, int yc)

{

putpixel(x+xc,y+yc,GREEN);

putpixel(-x+xc,y+yc,GREEN);

putpixel(x+xc, -y+yc,GREEN);

putpixel(-x+xc, -y+yc, GREEN);

putpixel(y+xc, x+yc, GREEN);

putpixel(y+xc, -x+yc, GREEN);

putpixel(-y+xc, x+yc, GREEN);

putpixel(-y+xc, -x+yc, GREEN);

}

1. **Algorithm Name: Mid-Point Circle Drawing Algorithm**

Code:

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

void circlemidpoint(int,int,int);

void drawcircle(int,int,int,int);

int main()

{

int xc,yc,r;

int gd=DETECT,gm;

initgraph(&gd,&gm,””);

cout<<“Enter center coordinate of circle: ”;

cin>>xc>>yc;

cout<<“Enter radius of circle:”;

cin>>r;

circlemidpoint(xc,yc,r);

getch();

closegraph();

return 0;

}

void circlemidpoint(int xc,int yc,int r)

{

int x=0,y=r;

int p=5/4-r;

while(x<y)

{

drawcircle(xc,yc,x,y);

x++;

if(p<0)

{

p=p+2\*x+1;

}

else

{

y–;

p=p+2\*(x-y)+1;

}

drawcircle(xc,yc,x,y);

delay(100);

}

}

void drawcircle(int xc,int yc,int x,int y)

{

putpixel(xc+x, yc+y, GREEN);

putpixel(xc-x, yc+y, RED);

putpixel(xc+x, yc-y, YELLOW);

putpixel(xc-x, yc-y, BLUE);

putpixel(xc+y, yc+x, WHITE);

putpixel(xc-y, yc+x, RED);

putpixel(xc+y, yc-x, GREEN);

putpixel(xc-y, yc-x, RED);

}

*NORTH WESTERN UNIVERSITY, KHULNA*

******

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab Report***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Isfat Ara Hasan Ema***  ***Id: 20201112010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date : Teacher’s Signature***

1. **Algorithm Name: Adaptive Decision Boundary**

Code:

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

class AdaptiveDecisionBoundary {

private:

vector<double> weights;

double learningRate;

public:

AdaptiveDecisionBoundary(int inputSize, double learningRate) : learningRate(learningRate) {

srand(static\_cast<unsigned int>(time(0)));

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

weights.push\_back(static\_cast<double>(rand()) / RAND\_MAX);

}

}

int predict(const vector<double>& inputs) const {

double sum = 0.0;

for (size\_t i = 0; i < inputs.size(); ++i) {

sum += weights[i] \* inputs[i];

}

return (sum >= 0.0) ? 1 : -1;

}

void train(const vector<vector<double>>& trainingData, const vector<int>& labels, int maxEpochs) {

for (int epoch = 0; epoch < maxEpochs; ++epoch) {

for (size\_t i = 0; i < trainingData.size(); ++i) {

int prediction = predict(trainingData[i]);

int error = labels[i] - prediction;

for (size\_t j = 0; j < weights.size(); ++j) {

weights[j] += learningRate \* error \* trainingData[i][j];

}

}

}

}

const vector<double>& getWeights() const {

return weights;

}

};

int main() {

vector<vector<double>> trainingFeatures = {{2, 3}, {4, 5}, {1, 1}, {5, 2}};

vector<int> trainingLabels = {1, 1, -1, -1};

AdaptiveDecisionBoundary decisionBoundary(2, 0.1);

decisionBoundary.train(trainingFeatures, trainingLabels, 1000);

const vector<double>& learnedWeights = decisionBoundary.getWeights();

cout << "Learned Weights: ";

for (size\_t i = 0; i < learnedWeights.size(); ++i) {

cout << learnedWeights[i] << " ";

}

cout << endl;

vector<vector<double>> testFeatures = {{3, 4}, {1, 2}};

for (size\_t i = 0; i < testFeatures.size(); ++i) {

int prediction = decisionBoundary.predict(testFeatures[i]);

cout << "Prediction for [" << testFeatures[i][0] << ", " << testFeatures[i][1] << "]: " << prediction << endl;

}

return 0;

}

1. **Algorithm Name: Single Linkage Algorithm**

Code:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

class HierarchicalClustering {

private:

vector<vector<double>> data;

vector<vector<double>> distanceMatrix;

public:

HierarchicalClustering(const vector<vector<double>>& inputData) : data(inputData) {

initializeDistanceMatrix();

}

void initializeDistanceMatrix() {

size\_t n = data.size();

distanceMatrix.resize(n, vector<double>(n, 0.0));

for (size\_t i = 0; i < n; ++i) {

for (size\_t j = i + 1; j < n; ++j) {

double distance = calculateEuclideanDistance(data[i], data[j]);

distanceMatrix[i][j] = distance;

distanceMatrix[j][i] = distance;

}

}

}

double calculateEuclideanDistance(const vector<double>& point1, const vector<double>& point2) const {

double sum = 0.0;

for (size\_t i = 0; i < point1.size(); ++i) {

sum += pow(point1[i] - point2[i], 2);

}

return sqrt(sum);

}

pair<size\_t, size\_t> findClosestClusters() const {

size\_t n = distanceMatrix.size();

pair<size\_t, size\_t> minDistanceClusters = {0, 1};

double minDistance = distanceMatrix[0][1];

for (size\_t i = 0; i < n; ++i) {

for (size\_t j = i + 1; j < n; ++j) {

if (distanceMatrix[i][j] < minDistance) {

minDistance = distanceMatrix[i][j];

minDistanceClusters = {i, j};

}

}

}

return minDistanceClusters;

}

void updateDistanceMatrix(const pair<size\_t, size\_t>& clusters) {

size\_t n = distanceMatrix.size();

for (size\_t i = 0; i < n; ++i) {

if (i != clusters.first && i != clusters.second) {

distanceMatrix[i][clusters.first] = min(distanceMatrix[i][clusters.first], distanceMatrix[i][clusters.second]);

distanceMatrix[clusters.first][i] = distanceMatrix[i][clusters.first];

}

}

for (size\_t i = 0; i < n; ++i) {

distanceMatrix[i].erase(distanceMatrix[i].begin() + clusters.second);

}

distanceMatrix.erase(distanceMatrix.begin() + clusters.second);

}

void performHierarchicalClustering() {

size\_t n = distanceMatrix.size();

while (n > 1) {

pair<size\_t, size\_t> clusters = findClosestClusters();

updateDistanceMatrix(clusters);

cout << "Merged clusters " << clusters.first << " and " << clusters.second << ", New Distance Matrix:" << endl;

printDistanceMatrix();

--n;

}} }

void printDistanceMatrix() const {

for (const auto& row : distanceMatrix) {

for (double distance : row) {

cout << distance << " ";

}

cout << endl; }

cout << endl;

}};

int main() {

vector<vector<double>> inputData = {{1, 2}, {5, 8}, {1.5, 1.8}, {8, 8}, {1, 0.6}, {9, 11}};

HierarchicalClustering hierarchicalClustering(inputData);

cout << "Initial Distance Matrix:" << endl;

hierarchicalClustering.printDistanceMatrix();

hierarchicalClustering.performHierarchicalClustering();

return 0;

}

1. **Algorithm Name: DDA Line generation Algorithm**

Code:

#include <iostream>

#include <graphics.h>

using namespace std;

void drawLineDDA(int x1, int y1, int x2, int y2) {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int dx = x2 - x1;

int dy = y2 - y1;

int steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);

float xIncrement = dx / (float)steps;

float yIncrement = dy / (float)steps;

float x = x1;

float y = y1;

putpixel(round(x), round(y), WHITE);

for (int i = 1; i <= steps; i++) {

x += xIncrement;

y += yIncrement;

putpixel(round(x), round(y), WHITE);

}

delay(5000);

closegraph();

int main() {

int x1, y1, x2, y2;

cout << "Enter the coordinates of the first point (x1 y1): ";

cin >> x1 >> y1;

cout << "Enter the coordinates of the second point (x2 y2): ";

cin >> x2 >> y2;

drawLineDDA(x1, y1, x2, y2);

return 0;

}

1. **Algorithm Name: Bresenham's Line Algorithm**

Code:

#include <iostream>

#include <graphics.h>

using namespace std;

void drawLineBresenham(int x1, int y1, int x2, int y2) {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int p = 2 \* dy - dx;

int xIncrement = (x1 < x2) ? 1 : -1;

int yIncrement = (y1 < y2) ? 1 : -1;

int x = x1;

int y = y1;

putpixel(x, y, WHITE);

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

x += xIncrement;

if (p < 0) {

p += 2 \* dy;

} else {

y += yIncrement;

p += 2 \* (dy - dx);

}

putpixel(x, y, WHITE);

}

delay(5000);

closegraph();

}

int main() {

int x1, y1, x2, y2;

cout << "Enter the coordinates of the first point (x1 y1): ";

cin >> x1 >> y1;

cout << "Enter the coordinates of the second point (x2 y2): ";

cin >> x2 >> y2;

drawLineBresenham(x1, y1, x2, y2);

return 0;

}

1. **Algorithm Name: Bresenham’s circle drawing algorithm**

Code:

#include <iostream>

#include <graphics.h>

using namespace std;

void drawCircleBresenham(int xc, int yc, int r) {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int x = 0, y = r;

int p = 3 - 2 \* r;

putpixel(xc + x, yc - y, WHITE);

if (r > 0) {

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc - y, WHITE);

putpixel(xc - x, yc + y, WHITE);

putpixel(xc + x, yc - y, WHITE);

}

while (x <= y) {

x++;

if (p > 0) {

y--;

p = p + 4 \* (x - y) + 10;

} else {

p = p + 4 \* x + 6;

}

putpixel(xc + x, yc - y, WHITE);

putpixel(xc - x, yc - y, WHITE);

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc + y, WHITE);

if (x != y) {

putpixel(xc + y, yc - x, WHITE);

putpixel(xc - y, yc - x, WHITE);

putpixel(xc + y, yc + x, WHITE);

putpixel(xc - y, yc + x, WHITE);

}

}

delay(5000);

closegraph();

}

int main() {

int xc, yc, r;

cout << "Enter the center coordinates of the circle (xc yc): ";

cin >> xc >> yc;

cout << "Enter the radius of the circle: ";

cin >> r;

drawCircleBresenham(xc, yc, r);

return 0;

}

1. **Algorithm Name: Mid-Point Circle Drawing Algorithm**

Code:

#include <iostream>

#include <graphics.h>

using namespace std;

void drawCircleMidpoint(int xc, int yc, int r) {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int x = r, y = 0;

int p = 1 - r;

putpixel(xc + x, yc - y, WHITE);

if (r > 0) {

putpixel(xc - x, yc - y, WHITE);

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc + y, WHITE);

}

while (x > y) {

y++;

if (p <= 0)

p = p + 2 \* y + 1;

else {

x--;

p = p + 2 \* y - 2 \* x + 1;

}

if (x < y)

break;

putpixel(xc + x, yc - y, WHITE);

putpixel(xc - x, yc - y, WHITE);

putpixel(xc + x, yc + y, WHITE);

putpixel(xc - x, yc + y, WHITE);

if (x != y) {

putpixel(xc + y, yc - x, WHITE);

putpixel(xc - y, yc - x, WHITE);

putpixel(xc + y, yc + x, WHITE);

putpixel(xc - y, yc + x, WHITE);

}

}

delay(5000);

closegraph();

}

int main() {

int xc, yc, r;

cout << "Enter the center coordinates of the circle (xc yc): ";

cin >> xc >> yc;

cout << "Enter the radius of the circle: ";

cin >> r;

drawCircleMidpoint(xc, yc, r);

return 0;

}

*NORTH WESTERN UNIVERSITY, KHULNA*

**

***Course Title: Computer Graphics and Pattern Recognition Sessional***

***Course Code: CSE-4302***

***Lab Report***

|  |  |
| --- | --- |
| ***Submitted by:***  ***Name: Fatema Zahan Shayla***  ***Id: 20201059010***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna*** | ***Submitted to:***  ***Name: M. Raihan***  ***Assistant Professor***  ***Department of Computer Science and Engineering***  ***North Western University, Khulna.*** |

***Submission Date :*** ***Teacher’s Signature***

1. **Algorithm Name: Adaptive Decision Boundary**

Code:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

struct FeatureVector {

double feature1, feature2;

FeatureVector(double f1, double f2) : feature1(f1), feature2(f2) {}

};

double euclideanDistance(const FeatureVector& vec1, const FeatureVector& vec2) {

return sqrt(pow(vec1.feature1 - vec2.feature1, 2) + pow(vec1.feature2 - vec2.feature2, 2));

}

class AdaptiveDecisionBoundary {

public:

AdaptiveDecisionBoundary(const vector<FeatureVector>& featureVectors) : featureVectors(featureVectors) {}

void trainModel() {

initializeClusters();

while (clusters.size() > 1) {

int minCluster1, minCluster2;

findClosestClusters(minCluster1, minCluster2);

mergeClusters(minCluster1, minCluster2);

} }

void testModel(const FeatureVector& testVector) {

int predictedCluster = predictCluster(testVector);

cout << "Predicted Cluster: " << predictedCluster << endl;

}

private:

vector<FeatureVector> featureVectors;

vector<vector<int>> clusters;

void initializeClusters() {

clusters.clear();

for (size\_t i = 0; i < featureVectors.size(); ++i) {

clusters.push\_back({static\_cast<int>(i)});

}}

double calculateDistance(int cluster1, int cluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (int index1 : clusters[cluster1]) {

for (int index2 : clusters[cluster2]) {

double distance = euclideanDistance(featureVectors[index1], featureVectors[index2]);

if (distance < minDistance) {

minDistance = distance;

} } }

return minDistance;

}

void findClosestClusters(int& minCluster1, int& minCluster2) {

double minDistance = numeric\_limits<double>::infinity();

for (size\_t i = 0; i < clusters.size(); ++i) {

for (size\_t j = i + 1; j < clusters.size(); ++j) {

double distance = calculateDistance(i, j);

if (distance < minDistance) {

minDistance = distance;

minCluster1 = i;

minCluster2 = j;

}}}}

void mergeClusters(int cluster1, int cluster2) {

clusters[cluster1].insert(clusters[cluster1].end(), clusters[cluster2].begin(), clusters[cluster2].end());

clusters.erase(clusters.begin() + cluster2);

}

int predictCluster(const FeatureVector& testVector) {

double minDistance = numeric\_limits<double>::infinity();

int predictedCluster = -1;

for (size\_t i = 0; i < clusters.size(); ++i) {

for (int index : clusters[i]) {

double distance = euclideanDistance(testVector, featureVectors[index]);

if (distance < minDistance) {

minDistance = distance;

predictedCluster = i;

} }}

return predictedCluster;

}};

int main() { vector<FeatureVector> featureData = {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {10, 12}, {11, 13}, {13, 14}};

AdaptiveDecisionBoundary decisionBoundaryModel(featureData);

decisionBoundaryModel.trainModel();

FeatureVector testFeatureVector = {5, 6};

decisionBoundaryModel.testModel(testFeatureVector);

return 0;

}

1. **Algorithm Name: Single Linkage Algorithm**

Code:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

double euclideanDistance(const vector<double>& point1, const vector<double>& point2) {

double sum = 0.0;

for (size\_t i = 0; i < point1.size(); ++i) {

sum += pow(point1[i] - point2[i], 2);

}

return sqrt(sum);

}

void clustering(vector<vector<double>>& data) {

vector<vector<int>> clusters;

for (int i = 0; i < static\_cast<int>(data.size()); ++i) {

clusters.push\_back({i});

}

cout << "Initial Clusters:" << endl;

for (const auto& cluster : clusters) {

for (int index : cluster) {

cout << index << " ";

}

cout << endl;

}

int a;

cout << "For single or Complete linkage, type 1 or 2 respectively: ";

cin >> a;

while (clusters.size() > 1) {

double minDistance = numeric\_limits<double>::infinity();

pair<int, int> merge = {0, 1};

for (size\_t i = 0; i < clusters.size(); ++i) {

for (size\_t j = i + 1; j < clusters.size(); ++j) {

double distance;

if (a == 1) {

distance = euclideanDistance(data[clusters[i][0]], data[clusters[j][0]]);

if (distance < minDistance) {

minDistance = distance;

merge = {static\_cast<int>(i), static\_cast<int>(j)};

}

} else if (a == 2) {

distance = euclideanDistance(data[clusters[i][0]], data[clusters[j][0]]);

if (distance > minDistance) {

minDistance = distance;

merge = {static\_cast<int>(i), static\_cast<int>(j)};

}}}}

clusters[merge.first].insert(clusters[merge.first].end(), clusters[merge.second].begin(), clusters[merge.second].end());

clusters.erase(clusters.begin() + merge.second);

cout << "Clusters:" << endl;

for (const auto& cluster : clusters) {

for (int index : cluster) {

cout << index << " ";

}

cout << endl;

}

}

cout << "Final cluster:";

for (int index : clusters[0]) {

cout << " " << index;

}

cout << endl;

}

int main() {

vector<vector<double>> arr = {{1, 2}, {5, 8}, {1.5, 1.8}, {8, 8}, {1, 0.6}, {9, 11}};

clustering(arr);

return 0;

}

1. **Algorithm Name: DDA Line generation Algorithm**

Code:

#include <iostream>>

#include<conio.h>

#include<math.h>

using namespace std;

int RoundFunction(float number)

{

if (number - (int)number < 0.5)

{

return (int)number;

}

else

{

return (int)(number + 1);

}

}

void DDALineDrawing(int x0, int y0, int x1, int y1)

{

int dx = x1 - x0;

int dy = y1 - y0;

int maxCount;

if (abs(dx) > abs(dy))

{

maxCount = abs(dx);

}

else

{

maxCount = abs(dy);

}

float x\_increment = (float)dx / maxCount;

float y\_increment = (float)dy / maxCount;

float x = x0;

float y = y0;

cout<<"Output: "<<endl<<endl;

for (int i = 0; i < maxCount; i++)

{

cout << RoundFunction(x) << " " << RoundFunction(y) << "\n";

x += x\_increment;

y += y\_increment;

}

}

int main()

{

int x0,y0, x1, y1;

cout<<"Enter the value for X0: ";

cin>>x0;

cout<<"Enter the value for y0: ";

cin>>y0;

cout<<"Enter the value for x1: ";

cin>>x1;

cout<<"Enter the value for y1: ";

cin>>y1;

DDALineDrawing(x0, y0, x1, y1);

getch();

}

1. **Algorithm Name: Bresenham's Line Algorithm**

Code:

#include<iostream>

#include<conio.h>

using namespace std;

void BresenhamLineDrawing(int x1, int y1, int x2, int y2)

{

int newValue = 2 \* (y2 - y1);

int slop\_Err = newValue - (x2 - x1);

cout<<"Output: "<<endl<<endl;

for (int x = x1, y = y1; x <= x2; x++) {

cout << "(" << x << "," << y << ")\n";

slop\_Err += newValue;

if (slop\_Err >= 0) {

y++;

slop\_Err -= 2 \* (x2 - x1);

}}}

int main()

{ int x1, y1 , x2 , y2;

cout<<"Enter the value for X0: ";

cin>>x1;

cout<<"Enter the value for y0: ";

cin>>y1;

cout<<"Enter the value for x1: ";

cin>>x2;

cout<<"Enter the value for y1: ";

cin>>y2;

BresenhamLineDrawing(x1, y1, x2, y2);

getch();

}

1. **Algorithm Name: Bresenham’s circle drawing algorithm**

Code:

#include <stdio.h>

#include <dos.h>

#include <graphics.h>

#include<conio.h>

void CircleDrawing(int x\_Coordinate, int y\_Coordinate, int x, int y)

{

putpixel(x\_Coordinate+x, y\_Coordinate+y, RED);

putpixel(x\_Coordinate-x, y\_Coordinate+y, RED);

putpixel(x\_Coordinate+x, y\_Coordinate-y, RED);

putpixel(x\_Coordinate-x, y\_Coordinate-y, RED);

putpixel(x\_Coordinate+y, y\_Coordinate+x, RED);

putpixel(x\_Coordinate-y, y\_Coordinate+x, RED);

putpixel(x\_Coordinate+y, y\_Coordinate-x, RED);

putpixel(x\_Coordinate-y, y\_Coordinate-x, RED);

}

void BresenhamCircle(int x\_Coordinate, int y\_Coordinate, int Radius)

{

int x = 0, y = Radius;

int d = 3 - 2 \* Radius;

CircleDrawing(x\_Coordinate, y\_Coordinate, x, y);

while (y >= x)

{

x++;

if (d > 0)

{

y--;

d = d + 4 \* (x - y) + 10;

}

else

d = d + 4 \* x + 6;

CircleDrawing(x\_Coordinate, y\_Coordinate, x, y);

delay(50);

}

}

int main()

{

int x\_Coordinate, y\_Coordinate , Radius;

cout<<"Enter the value for x\_Coordinate: ";

cin>>x\_Coordinate;

cout<<"Enter the value for y\_Coordinate: ";

cin>>y\_Coordinate;

cout<<"Enter the value for Radius: ";

cin>>Radius;

int gd = DETECT, gm;

initgraph(&gd, &gm, "");

BresenhamCircle(x\_Coordinate, y\_Coordinate, Radius);

getch();

}

1. **Algorithm Name: Mid-Point Circle Drawing Algorithm**

Code:

#include<iostream>

#include<conio.h>

using namespace std;

void midPointCircleDrawing(int x\_Coordinate, int y\_Coordinate, int Radius)

{

int x = Radius, y = 0;

cout<<"Output: "<<endl;

cout << "(" << x + x\_Coordinate << ", " << y + x\_Coordinate << ") ";

if (Radius > 0)

{

cout << "(" << x + x\_Coordinate << ", " << -y + y\_Coordinate << ") ";

cout << "(" << y + x\_Coordinate << ", " << x + y\_Coordinate << ") ";

cout << "(" << -y + x\_Coordinate << ", " << x + y\_Coordinate << ")\n";

}

int Point = 1 - Radius;

while (x > y)

{

y++;

if (Point <= 0){

Point = Point + 2\*y + 1;

}

else

{

x--;

Point = Point + 2\*y - 2\*x + 1;

}

if (x < y)

break;

cout << "(" << x + x\_Coordinate << ", " << y + y\_Coordinate << ") ";

cout << "(" << -x + x\_Coordinate << ", " << y + y\_Coordinate << ") ";

cout << "(" << x + x\_Coordinate << ", " << -y + y\_Coordinate << ") ";

cout << "(" << -x + x\_Coordinate << ", " << -y + y\_Coordinate << ")\n";

if (x != y)

{

cout << "(" << y + x\_Coordinate << ", " << x + y\_Coordinate << ") ";

cout << "(" << -y + x\_Coordinate << ", " << x + y\_Coordinate << ") ";

cout << "(" << y + x\_Coordinate << ", " << -x + y\_Coordinate << ") ";

cout << "(" << -y + x\_Coordinate << ", " << -x + y\_Coordinate << ")\n";

}

}

}

int main()

{

int x\_Coordinate,y\_Coordinate,Radius;

cout<<"Enter the value for x\_Coordinate: ";

cin>>x\_Coordinate;

cout<<"Enter the value for y\_Coordinate: ";

cin>>y\_Coordinate;

cout<<"Enter the value for Radius: ";

cin>>Radius;

midPointCircleDrawing(x\_Coordinate, y\_Coordinate, Radius);

getch();

}