Contents

1 tools/Rectangle.java

```
import java.io.Serializable;

public class Rectangle implements Serializable{
    private Point coordinates;
    private double width, length;

    public Rectangle(Point coordinates, double width, double length){
        this.coordinates = coordinates;
        this.width = width;
        this.length = length;
    }

    public Point getCoordinates() {
        return coordinates;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public void setCoordinates(Point coordinates) {
        this.coordinates = coordinates;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public Point getCenter(){
        return new Point((width+coordinates.getX())/2,(length
            +coordinates.getY())/2);
    }

}
```

2 tools/Point.java

```
import java.io.Serializable;


public class Point implements Serializable {
    private double x, y;

    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }
}
```

3 tools/RectangleDB.java

```
import java.io.Serializable;


public class RectangleDB implements Serializable{
    private Point lt,rt,lb,rb;

    public RectangleDB(Point lt, Point rt, Point lb, Point rb) {
        this.lt = lt;
        this.rt = rt;
        this.lb = lb;
        this.rb = rb;
    }

    public Point getLt() {
        return lt;
    }

    public void setLt(Point lt) {
        this.lt = lt;
    }

    public Point getRt() {
        return rt;
    }

    public void setRt(Point rt) {
        this.rt = rt;
    }

    public Point getLb() {
        return lb;
    }

    public void setLb(Point lb) {
        this.lb = lb;
    }

    public Point getRb() {
        return rb;
    }

    public void setRb(Point rb) {
        this.rb = rb;
    }

    public boolean isNeighbour (RectangleDB r){
```

```
        if(this.getRt() == r.getLt() && this.getRb() == r.getLb() )
            return true;
        if(this.getLt() == r.getRt() && this.getLb() == r.getRb())
            return true;
        if(this.getLt() == r.getLb() && this.getRt() == r.getRb())
            return true;
        if(this.getLb() == r.getLt() && this.getRb() == this.getRt())
            return true;

        return true;


    }
}
```

4 models/indoormapping/Room.java

```
import tools.Point;
import tools.RectangleDB;
import tools.Rectangle;

import java.io.Serializable;

public class Room implements Serializable{
    private String roomName;
    private String id;
    private String building_id;
    private Rectangle roomRectangle;
    private RectangleDB rectangleDB;
    private String roomDescription;
    private double width, length;
    private double est_time;
    private int excitement;


    public Room(String id, String building_id, String roomName,
         RectangleDB rectangleDB, double width, double length, double
         est_time, int excitement){
        this.roomName = roomName;
        this.id = id;
        this.width = width;
        this.length = length;
        this.rectangleDB = rectangleDB;
        this.building_id = building_id;
        this.roomRectangle = new Rectangle(new Point(0,0),0,0); //for
            testing!!!
        this.est_time = est_time;
        this.excitement = excitement;
    }

    public String getBuilding_id() {
        return building_id;
    }

    public Room(String building_id, String roomName, RectangleDB
         rectangleDB, double width, double length){
        this.building_id = building_id;
        this.roomName = roomName;
        this.width = width;
        this.length = length;
        this.roomDescription = "";
        this.rectangleDB = rectangleDB;
```

```
        this.roomRectangle = new Rectangle(new Point(0,0),0,0); //for
            testing!!!
    }

    public Room(Rectangle r){
        this.roomRectangle = r;
        roomName = "";
        roomDescription = "";
    }

    public String getRoomName() {
        return roomName;
    }

    public void setRoomName(String roomName) {
        this.roomName = roomName;
    }

    public Rectangle getRoomRectangle() {
        return roomRectangle;
    }

    public void setRoomRectangle(Rectangle roomRectangle) {
        this.roomRectangle = roomRectangle;
    }

    public String getRoomDescription() {
        return roomDescription;
    }

    public void setRoomDescription(String roomDescription) {
        this.roomDescription = roomDescription;
    }

    public boolean isNeighbour(Room room){

        if(this == room)
            return false;

        if(this.getRoomRectangle().getCoordinates().getX() +
            this.getRoomRectangle().getWidth()
               == room.getRoomRectangle().getCoordinates().getX()
               || this.getRoomRectangle().getCoordinates().getY() +
                   this.getRoomRectangle().getLength()
               == room.getRoomRectangle().getCoordinates().getY()
               || room.getRoomRectangle().getCoordinates().getX() +
                   room.getRoomRectangle().getWidth()
               == this.getRoomRectangle().getCoordinates().getX()
               || room.getRoomRectangle().getCoordinates().getY() +
                   room.getRoomRectangle().getLength()
```

```
                == this.getRoomRectangle().getCoordinates().getY())
            return true;

        return false;

    }

    public String getId() {
        return id;
    }

    public RectangleDB getRectangleDB() {
        return rectangleDB;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public void setEst_time(double est_time) {
        this.est_time = est_time;
    }

    public int getExcitement(){return excitement; }

    public double getEst_time() {

        return est_time;
    }

}
```

5 models/indoormapping/Floor.java

```
import java.io.Serializable;
import java.util.ArrayList;


public class Floor implements Serializable{
    Room[] rooms;

    public Floor(Room[] rooms)
    {
        this.rooms = rooms;
    }

    public Room[] getRooms() {
        return rooms;
    }

    public Room[] getNeighbours(Room room) {
        ArrayList<Room> neighbours = new ArrayList<>();
        for(Room r:rooms){
            if(r.isNeighbour(room))
                neighbours.add(r);
        }
        return neighbours.toArray(new Room[neighbours.size()]);
    }

    public void setRooms(Room[] rooms) {
        this.rooms = rooms;
    }

}
```

6 models/indoormapping/Building.java

```
import tools.RectangleDB;
import java.io.Serializable;


public class Building implements Serializable{
    private RectangleDB rectangle;
    private String name;
    private double width;
    private double length;
    private String id;
    private Room[] rooms;



    public Building(String id, RectangleDB rectangle, String name,
         double width, double length, Room[] rooms) {
        this.rectangle = rectangle;
        this.id = id;
        this.name = name;
        this.width = width;
        this.length = length;
        this.rooms = rooms;
    }

    public Building(RectangleDB rectangle, String name, double width,
         double length) {
        this.rectangle = rectangle;
        this.id = "";
        this.name = name;
        this.width = width;
        this.length = length;
    }

    public void setRectangle(RectangleDB rectangle) {
        this.rectangle = rectangle;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setWidth(double width) {
        this.width = width;
    }
```

```
    public void setLength(double length) {
        this.length = length;
    }

    public RectangleDB getRectangle() {
        return rectangle;
    }

    public String getName() {
        return name;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public Room[] getRooms() {
        return rooms;
    }

    public void setRooms(Room[] rooms) {
        this.rooms = rooms;
    }
}
```

7 com/company/Planner.java

```
import models.indoormapping.Room;
import tools.Point;
import org.json.JSONArray;
import org.json.JSONObject;
import tools.RectangleDB;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;

public class Planner {
    private static final String[] pddlIntro = {
        "(define (problem simplemuseum)",
        "(:domain museum)",
        "(:objects",
        "    visitor - person"
    };

    public static String route(JSONArray jsonArray, int deadline, String
         building_id){
        ArrayList<Room> rooms = getRoomsFromJSON(jsonArray);
        makePDDL(rooms,deadline);
        building_id = building_id.replaceAll("[\\s&;]+","");
        ArrayList<String> toExecute = new ArrayList<>();
        toExecute.add("#!/bin/bash\n" +
                "ulimit -t 1\n");
        toExecute.add( "./planner " + "--optimise " + "domain.pddl
            "+building_id+"_temp.pddl");
        Path execPath = Paths.get("five-seconds");
        try {
            Files.write(execPath,toExecute, Charset.forName("UTF-8"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        String command = "./five-seconds";
        Process process = null;
        try {
            process = Runtime.getRuntime().exec(command);
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
        BufferedReader reader = new BufferedReader(new InputStreamReader(
                process.getInputStream()));
        ArrayList<String> output = new ArrayList<>();
        String s;
        try {
            while ((s = reader.readLine()) != null) {
                System.out.println("Script output: " + s);
                output.add(s);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return makeFinalResult(parseOutput(output),rooms);

    }
    public static ArrayList<Room> getRoomsFromJSON(JSONArray jsonArray){

        ArrayList<Room> toReturn = new ArrayList<>();
        for(Object o:jsonArray){
            JSONObject room = (JSONObject) o;
            Point lt = new
                Point(room.getJSONObject("rectangle").getJSONObject("lt").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("lt").getDouble("y"));
            Point rt = new
                Point(room.getJSONObject("rectangle").getJSONObject("rt").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("rt").getDouble("y"));
            Point lb = new
                Point(room.getJSONObject("rectangle").getJSONObject("lb").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("lb").getDouble("y"));
            Point rb = new
                Point(room.getJSONObject("rectangle").getJSONObject("rb").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("rb").getDouble("y"));
            RectangleDB r = new RectangleDB(lt,rt,lb,rb);
            Room toAdd = new
                Room(room.getString("_id"),room.getString("building_id"),room.getString("name"),r,
                   room.getDouble("width"),room.getDouble("height"),room.getDouble("est_time"),room.get
            toReturn.add(toAdd);
        }
        return toReturn;
    }

    private static void makePDDL(ArrayList<Room> rooms, int deadline){

        Path tempPath =
            Paths.get(rooms.get(0).getBuilding_id()+"_temp.pddl");
        ArrayList<String> toWrite = new ArrayList<>();
        toWrite.addAll(Arrays.asList(pddlIntro));
        toWrite.add("   ");
```

```
        for(Room r:rooms){
            toWrite.set(toWrite.size()-1,toWrite.get(toWrite.size()-1)+"e"+rooms.indexOf(r)+"
                ");
        }
        toWrite.set(toWrite.size()-1,toWrite.get(toWrite.size()-1)+"-
            exhibit");
        toWrite.add(")");
        toWrite.add("(:init");
        for(Room r:rooms){
            toWrite.add("(want-to-see "+"e"+rooms.indexOf(r)+")");
        }

        toWrite.addAll(getBuildingLayout(rooms));
        for(Room r:rooms){
            toWrite.add("(= (time-to-see "+"e"+rooms.indexOf(r)+")
                "+(int) r.getEst_time()+")");
        }
        toWrite.add("(at visitor "+"e0"+")");
        toWrite.add("(= (seen) 0)");
        for(Room r:rooms){
            toWrite.add("(= (excitement "+"e"+rooms.indexOf(r)+") "+
                r.getExcitement()+(")"));
        }
        toWrite.add("(open)");
        toWrite.add("(at "+deadline+" (not (open)))");
        toWrite.add(")");
        toWrite.add("(:goal (and");

        for(Room r:rooms){
            toWrite.add(";(visited "+"e"+rooms.indexOf(r)+")");
        }
        toWrite.add(")");
        toWrite.add(")");
        toWrite.add("(:metric maximize (seen))");
        toWrite.add(")");
        try {
            Files.write(tempPath,toWrite, Charset.forName("UTF-8"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static ArrayList<String> getBuildingLayout(ArrayList<Room>
        rooms){
        ArrayList<String> toReturn = new ArrayList<>();
        for(int i = 0; i<rooms.size()-1; ++i){
            for(int j=i+1; j<rooms.size();++j){
                if(rooms.get(i).getRectangleDB().isNeighbour(rooms.get(j).getRectangleDB())){
                    toReturn.add("(path " + "e"+i + " "+"e"+ j + ") (path
                        " +"e"+j+" "+"e"+i+")");
```

```
                toReturn.add("(= (time-to-walk "+"e"+i+" "+"e"+j+")
                    0)");
                toReturn.add("(= (time-to-walk "+"e"+j+" "+"e"+i+")
                    0)");
            }
        }
    }
    return toReturn;
}

private static ArrayList<String> parseOutput(ArrayList<String>
    output){
    ArrayList<String> toReturn = new ArrayList<>();
    ArrayList<String> currentResult = new ArrayList<>();

    for(String s:output){
        if(s.equals(";;;; Solution Found")) {
            toReturn = currentResult;
            currentResult.clear();
        }
        if(!((s.startsWith(";")|| s.isEmpty()))) {
            if (s.contains("(") && s.contains(")")) {
                currentResult.add( s.substring(s.indexOf("(")+1,
                    s.indexOf(")")));
            }
        }
        if(s.startsWith("..")){
            toReturn = currentResult;
        }
    }

   return toReturn;
}

private static String makeFinalResult(ArrayList<String> output,
    ArrayList<Room> rooms){
    String toReturn = "";
    for(String s:output){
        String[] tempSplit = s.split(" ");
        if(tempSplit[0].equals("view")){
            int roomIndex
                =Integer.parseInt(tempSplit[2].substring(tempSplit[2].indexOf("e")+1));
            toReturn += "view:"+rooms.get(roomIndex).getId()+";";

        } else if(tempSplit[0].equals("walk")){
            int roomIndexOr =
                Integer.parseInt(tempSplit[2].substring(tempSplit[2].indexOf("e")+1));
            int roomIndexDest =
                Integer.parseInt(tempSplit[3].substring(tempSplit[3].indexOf("e")+1));
```

```
                toReturn +=
                    "walk:"+rooms.get(roomIndexOr).getId()+","+rooms.get(roomIndexDest).getId()+";";
            }
        }

        return toReturn.substring(0,toReturn.length()-1);
    }
}
```

8 com/company/Server.java

```
import org.apache.commons.lang3.tuple.Pair;
import org.json.JSONObject;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.bayes.NaiveBayes;

import java.net.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import java.io.*;

public class Server
{
    String currentBID;
    ArrayList<Pair<String,BayesNet>> classifiersBN;
    ArrayList<Pair<String,NaiveBayes>> classifiersNB;
    public Server()
    {
        classifiersBN = new ArrayList<>();
        classifiersNB = new ArrayList<>();

        try {
            ServerSocket sSocket = new ServerSocket(5000);
            System.out.println("Server started!");
            currentBID = "";

            while(true) {
                Socket socket = sSocket.accept();
                ClientThread cT = new ClientThread(socket);
                new Thread(cT).start();
            }
        } catch(IOException exception) {
            System.out.println("Error: " + exception);
        }
    }

    class ClientThread implements Runnable
    {
        Socket socket;
        public ClientThread(Socket socket)
        {
            this.socket = socket;
        }

        public void run()
        {
```

```
try {
    PrintWriter pw = new
        PrintWriter(socket.getOutputStream(), true);
    BufferedReader br = new BufferedReader(new
        InputStreamReader(socket.getInputStream()));
    String rec = br.readLine();
    while (rec == null) {
        rec = br.readLine();
    }
    System.out.println(rec);
    JSONObject recJSON = new JSONObject(rec);
    if(recJSON.getString("command").equals("learn")){
        BayesNet bn =
            Learner.learnFromJSON_BN(recJSON.getString("building_id"),recJSON.getJSONArray("
        NaiveBayes nb =
            Learner.learnFromJSON_NB(recJSON.getString("building_id"),recJSON.getJSONArray("
        int i =
            buildingClassifierBNInitialised(recJSON.getString("building_id"));
        int j =
            buildingClassifierNBInitialised(recJSON.getString("building_id"));
            //not really needed.
        if(i>-1){
            classifiersBN.set(i,Pair.of(recJSON.getString("building_id"),bn));
        } else {
            classifiersBN.add(Pair.of(recJSON.getString("building_id"),bn));
        }

        if(i>-1){
            classifiersNB.set(i,Pair.of(recJSON.getString("building_id"),nb));
        } else {
            classifiersNB.add(Pair.of(recJSON.getString("building_id"),nb));
        }
        pw.write("Done!");
    } else
        if(recJSON.getString("command").equals("classify")){
        int i =
            buildingClassifierBNInitialised(recJSON.getString("building_id"));
        int j =
            buildingClassifierNBInitialised(recJSON.getString("building_id"));
            //same

        if(i>-1){
            String res
                ="BN:"+Learner.classify_BN(recJSON.getString("building_id"),
                    recJSON.getJSONArray("learning_set"),classifiersBN.get(i).getRight());
            res +=
                ",NB:"+Learner.classify_NB(recJSON.getString("building_id"),
                    recJSON.getJSONArray("learning_set"),classifiersNB.get(j).getRight());
            pw.write(res);
            System.out.println(res);
```

```
                } else
                    if(Files.exists(Paths.get(recJSON.getString("building_id")+".arff"))){
                    BayesNet bn =
                        Learner.getClassifierBN(recJSON.getString("building_id"));
                    NaiveBayes nb =
                        Learner.getClassifierNB(recJSON.getString("building_id"));
                    classifiersBN.add(Pair.of(recJSON.getString("building_id"),bn));
                    classifiersNB.add(Pair.of(recJSON.getString("building_id"),nb));

                    i =
                        buildingClassifierBNInitialised(recJSON.getString("building_id"));
                    j =
                        buildingClassifierNBInitialised(recJSON.getString("building_id"));
                        //same
                    String res
                        ="BN:"+Learner.classify_BN(recJSON.getString("building_id"),
                           recJSON.getJSONArray("learning_set"),classifiersBN.get(i).getRight());
                    res +=
                        ",NB:"+Learner.classify_NB(recJSON.getString("building_id"),
                           recJSON.getJSONArray("learning_set"),classifiersNB.get(j).getRight());
                    pw.write(res);
                } else {
                    pw.write("Weird classify request!");
                }
            }  else if(recJSON.getString("command").equals("route")){
                pw.write(Planner.route(recJSON.getJSONArray("request_set"),recJSON.getInt("deadline"
                       recJSON.getString("building_id")));
            }
            pw.flush();
        } catch(IOException exception) {
            System.out.println("Error: " + exception);
        }
    }
}

public int buildingClassifierBNInitialised(String building_id){
    for(Pair<String,BayesNet> p:classifiersBN){
        if(p.getLeft().equals(building_id))
            return classifiersBN.indexOf(p);
    }
    return -1;
}

public int buildingClassifierNBInitialised(String building_id){
    for(Pair<String,NaiveBayes> p:classifiersNB){
        if(p.getLeft().equals(building_id))
            return classifiersNB.indexOf(p);
    }
    return -1;
}
```

```
}
```

9 com/company/Main.java

```
public class Main {

    public static void main(String[] args) {
        new Server();
    }
}
```

10 com/company/Learner.java

```
import org.json.JSONArray;
import org.json.JSONObject;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instances;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.Set;


public class Learner {

    public static NaiveBayes learnFromJSON_NB(String building_id,
         JSONArray JSONData){
        makeLearnerARFFfromJSON(building_id,JSONData);
        NaiveBayes nb = new NaiveBayes();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            nb.buildClassifier(instances);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return nb;
    }

    public static BayesNet learnFromJSON_BN(String building_id,
         JSONArray JSONData){
        makeLearnerARFFfromJSON(building_id,JSONData);
        BayesNet bn = new BayesNet();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            bn.buildClassifier(instances);
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }

        return bn;
    }

    public static BayesNet getClassifierBN(String building_id){
        BayesNet bn = new BayesNet();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            bn.buildClassifier(instances);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return bn;
    }

    public static NaiveBayes getClassifierNB(String building_id){
        NaiveBayes nb = new NaiveBayes();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            nb.buildClassifier(instances);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return nb;
    }

    public static String classify_NB(String building_id, JSONArray
         JSONdata, NaiveBayes nb){
        makeClassifierARFF(building_id,JSONdata);
        Instances unlabeled;
        Instances labeled = null;
        double clsLabel = 0;
        try {
            unlabeled = new Instances(new BufferedReader(new
                FileReader(building_id+"_temp.arff")));
            unlabeled.setClassIndex(unlabeled.numAttributes()-1);
            labeled = new Instances(unlabeled);
            clsLabel = nb.classifyInstance(unlabeled.firstInstance());
            labeled.firstInstance().setClassValue(clsLabel);
            Files.deleteIfExists(Paths.get(building_id+"_temp.arff"));
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
        if (labeled != null) {
            return
                labeled.instance(0).classAttribute().value((int)clsLabel);
        }
        return "na";

    }

    public static String classify_BN(String building_id, JSONArray
         JSONdata, BayesNet bn){
        makeClassifierARFF(building_id,JSONdata);
        Instances unlabeled;
        Instances labeled = null;
        double clsLabel = 0;
        try {
            unlabeled = new Instances(new BufferedReader(new
                FileReader(building_id+"_temp.arff")));
            unlabeled.setClassIndex(unlabeled.numAttributes()-1);
            labeled = new Instances(unlabeled);
            clsLabel = bn.classifyInstance(unlabeled.firstInstance());
            labeled.firstInstance().setClassValue(clsLabel);
            Files.deleteIfExists(Paths.get(building_id+"_temp.arff"));
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (labeled != null) {
            return
                labeled.instance(0).classAttribute().value((int)clsLabel);
        }
        return "na";
    }

    public static void makeLearnerARFFfromJSON(String building_id,
         JSONArray JSONData){
        Path arffPath = Paths.get(building_id+".arff");
        Path roomsPath = Paths.get(building_id+"_rooms.data");
        Path rpPath = Paths.get(building_id+"_RPs.data");

        ArrayList<String> arffData = new ArrayList<>();
        ArrayList<String> rpids = getRPs(JSONData);
        ArrayList<String> rooms = getRooms(JSONData);

        arffData.add("@relation room"); arffData.add("");
        for(String s:rpids){
            arffData.add("@attribute "+s+" NUMERIC");
        }
        arffData.add("@attribute class {");
        for(int i=0;i<rooms.size()-1;++i){
            arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(i)+",");
```

```
        }
        arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(rooms.size()-1)+"}");
        arffData.add("@data");
        arffData.addAll(getOrderedReadings(JSONData,rpids));
        try {
            Files.write(arffPath,arffData, Charset.forName("UTF-8"));
            Files.write(roomsPath,rooms, Charset.forName("UTF-8"));
            Files.write(rpPath,rpids, Charset.forName("UTF-8"));

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public static void makeClassifierARFF(String building_id,JSONArray
         JSONData){

        Path tempPath = Paths.get(building_id+"_temp.arff");
        Path roomsPath = Paths.get(building_id+"_rooms.data");
        Path rpPath = Paths.get(building_id+"_RPs.data");

        ArrayList<String> arffData = new ArrayList<>();
        ArrayList<String> rpids = new ArrayList<>();
        ArrayList<String> rooms = new ArrayList<>();

        try {
            BufferedReader br =
                Files.newBufferedReader(roomsPath,Charset.forName("UTF-8"));
            String line;
            while((line = br.readLine()) != null){
                rooms.add(line);
            }
            br = Files.newBufferedReader(rpPath,Charset.forName("UTF-8"));
            while((line = br.readLine()) != null){
                rpids.add(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        arffData.add("@relation room"); arffData.add("");
        for(String s:rpids){
            arffData.add("@attribute "+s+" NUMERIC");
        }
        arffData.add("@attribute class {");
        for(int i=0;i<rooms.size()-1;++i){
            arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(i)+",");
        }
        arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(rooms.size()-1)+"}");
        arffData.add("@data");
```

```
        arffData.addAll(getOrderedReadings(JSONData,rpids));
        try {
            Files.write(tempPath,arffData, Charset.forName("UTF-8"));
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public static ArrayList<String> getRPs(JSONArray JSONdata){
        ArrayList<String> rpids = new ArrayList<>();
        for(Object reading: JSONdata){
            for(Object pair:
                 ((JSONObject)reading).getJSONArray("rpv_pair")){
                rpids.add(((JSONObject) pair).getString("RPID"));
            }
        }
        Set<String> noDuplicates = new LinkedHashSet<>(rpids);
        rpids.clear();
        rpids.addAll(noDuplicates);
        return rpids;
    }

    public static ArrayList<String> getRooms(JSONArray JSONdata){
        ArrayList<String> rooms = new ArrayList<>();
        for(Object reading: JSONdata){
            rooms.add(((JSONObject) reading).getString("room_id"));
        }
        Set<String> noDuplicates = new LinkedHashSet<>(rooms);
        rooms.clear();
        rooms.addAll(noDuplicates);
        return rooms;
    }

    public static ArrayList<String> getOrderedReadings(JSONArray
         jsonData, ArrayList<String> rpids){
        ArrayList<String> toReturn= new ArrayList<>();
        for(Object o:jsonData){
            String curLine = "";
            for(String rp:rpids){
                curLine += Double.toString(getValueByRPID(((JSONObject)
                     o),rp))+",";

            }
            curLine += ((JSONObject) o).getString("room_id");
            toReturn.add(curLine);
        }

        return toReturn;
    }
```

```
    public static double getValueByRPID(JSONObject reading, String RPID){
        JSONArray pairs = reading.getJSONArray("rpv_pair");
        for(Object pair:pairs){
            if(RPID.equals(((JSONObject) pair).getString("RPID")))
                return ((JSONObject) pair).getDouble("value");
        }

        return 0;
    }
}
```

sectiontools/Rectangle.java

```
import java.io.Serializable;

public class Rectangle implements Serializable{
    private Point coordinates;
    private double width, length;

    public Rectangle(Point coordinates, double width, double length){
        this.coordinates = coordinates;
        this.width = width;
        this.length = length;
    }

    public Point getCoordinates() {
        return coordinates;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public void setCoordinates(Point coordinates) {
        this.coordinates = coordinates;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public Point getCenter(){
        return new Point((width+coordinates.getX())/2,(length
            +coordinates.getY())/2);
    }

}
```

sectiontools/Point.java

```
import java.io.Serializable;


public class Point implements Serializable {
    private double x, y;

    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }
}
```

sectiontools/RectangleDB.java

```
import java.io.Serializable;


public class RectangleDB implements Serializable{
    private Point lt,rt,lb,rb;

    public RectangleDB(Point lt, Point rt, Point lb, Point rb) {
        this.lt = lt;
        this.rt = rt;
        this.lb = lb;
        this.rb = rb;
    }

    public Point getLt() {
        return lt;
    }

    public void setLt(Point lt) {
        this.lt = lt;
    }

    public Point getRt() {
        return rt;
    }

    public void setRt(Point rt) {
        this.rt = rt;
    }

    public Point getLb() {
        return lb;
    }

    public void setLb(Point lb) {
        this.lb = lb;
    }

    public Point getRb() {
        return rb;
    }

    public void setRb(Point rb) {
        this.rb = rb;
    }

    public boolean isNeighbour (RectangleDB r){
        if(this.getRt() == r.getLt() && this.getRb() == r.getLb() )
```

```
            return true;
        if(this.getLt() == r.getRt() && this.getLb() == r.getRb())
            return true;
        if(this.getLt() == r.getLb() && this.getRt() == r.getRb())
            return true;
        if(this.getLb() == r.getLt() && this.getRb() == this.getRt())
            return true;

        return true;


    }
}
```

sectionMETA-INF/MANIFEST.MF

```
Main-Class: com.company.Main
```

sectionmodels/indoormapping/Room.java

```
import tools.Point;
import tools.RectangleDB;
import tools.Rectangle;

import java.io.Serializable;

public class Room implements Serializable{
    private String roomName;
    private String id;
    private String building_id;
    private Rectangle roomRectangle;
    private RectangleDB rectangleDB;
    private String roomDescription;
    private double width, length;
    private double est_time;
    private int excitement;


    public Room(String id, String building_id, String roomName,
         RectangleDB rectangleDB, double width, double length, double
         est_time, int excitement){
        this.roomName = roomName;
        this.id = id;
        this.width = width;
        this.length = length;
        this.rectangleDB = rectangleDB;
        this.building_id = building_id;
        this.roomRectangle = new Rectangle(new Point(0,0),0,0); //for
            testing!!!
        this.est_time = est_time;
        this.excitement = excitement;
    }

    public String getBuilding_id() {
        return building_id;
    }

    public Room(String building_id, String roomName, RectangleDB
         rectangleDB, double width, double length){
        this.building_id = building_id;
        this.roomName = roomName;
        this.width = width;
        this.length = length;
        this.roomDescription = "";
        this.rectangleDB = rectangleDB;
        this.roomRectangle = new Rectangle(new Point(0,0),0,0); //for
            testing!!!
```

```
}

public Room(Rectangle r){
    this.roomRectangle = r;
    roomName = "";
    roomDescription = "";
}

public String getRoomName() {
    return roomName;
}

public void setRoomName(String roomName) {
    this.roomName = roomName;
}

public Rectangle getRoomRectangle() {
    return roomRectangle;
}

public void setRoomRectangle(Rectangle roomRectangle) {
    this.roomRectangle = roomRectangle;
}

public String getRoomDescription() {
    return roomDescription;
}

public void setRoomDescription(String roomDescription) {
    this.roomDescription = roomDescription;
}

public boolean isNeighbour(Room room){

    if(this == room)
        return false;

    if(this.getRoomRectangle().getCoordinates().getX() +
         this.getRoomRectangle().getWidth()
            == room.getRoomRectangle().getCoordinates().getX()
            || this.getRoomRectangle().getCoordinates().getY() +
                 this.getRoomRectangle().getLength()
            == room.getRoomRectangle().getCoordinates().getY()
            || room.getRoomRectangle().getCoordinates().getX() +
                 room.getRoomRectangle().getWidth()
            == this.getRoomRectangle().getCoordinates().getX()
            || room.getRoomRectangle().getCoordinates().getY() +
                 room.getRoomRectangle().getLength()
            == this.getRoomRectangle().getCoordinates().getY())
        return true;
```

```
        return false;

    }

    public String getId() {
        return id;
    }

    public RectangleDB getRectangleDB() {
        return rectangleDB;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public void setEst_time(double est_time) {
        this.est_time = est_time;
    }

    public int getExcitement(){return excitement; }

    public double getEst_time() {

        return est_time;
    }

}
```

sectionmodels/indoormapping/Floor.java

```
import java.io.Serializable;
import java.util.ArrayList;


public class Floor implements Serializable{
    Room[] rooms;

    public Floor(Room[] rooms)
    {
        this.rooms = rooms;
    }

    public Room[] getRooms() {
        return rooms;
    }

    public Room[] getNeighbours(Room room) {
        ArrayList<Room> neighbours = new ArrayList<>();
        for(Room r:rooms){
            if(r.isNeighbour(room))
                neighbours.add(r);
        }
        return neighbours.toArray(new Room[neighbours.size()]);
    }

    public void setRooms(Room[] rooms) {
        this.rooms = rooms;
    }

}
```

11 models/indoormapping/Building.java

```
import tools.RectangleDB;
import java.io.Serializable;


public class Building implements Serializable{
    private RectangleDB rectangle;
    private String name;
    private double width;
    private double length;
    private String id;
    private Room[] rooms;



    public Building(String id, RectangleDB rectangle, String name,
         double width, double length, Room[] rooms) {
        this.rectangle = rectangle;
        this.id = id;
        this.name = name;
        this.width = width;
        this.length = length;
        this.rooms = rooms;
    }

    public Building(RectangleDB rectangle, String name, double width,
         double length) {
        this.rectangle = rectangle;
        this.id = "";
        this.name = name;
        this.width = width;
        this.length = length;
    }

    public void setRectangle(RectangleDB rectangle) {
        this.rectangle = rectangle;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setWidth(double width) {
        this.width = width;
    }
```

```
    public void setLength(double length) {
        this.length = length;
    }

    public RectangleDB getRectangle() {
        return rectangle;
    }

    public String getName() {
        return name;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public Room[] getRooms() {
        return rooms;
    }

    public void setRooms(Room[] rooms) {
        this.rooms = rooms;
    }
}
```

12 com/company/Planner.java

```
import models.indoormapping.Room;
import tools.Point;
import org.json.JSONArray;
import org.json.JSONObject;
import tools.RectangleDB;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;

public class Planner {
    private static final String[] pddlIntro = {
        "(define (problem simplemuseum)",
        "(:domain museum)",
        "(:objects",
        "    visitor - person"
    };

    public static String route(JSONArray jsonArray, int deadline, String
         building_id){
        ArrayList<Room> rooms = getRoomsFromJSON(jsonArray);
        makePDDL(rooms,deadline);
        building_id = building_id.replaceAll("[\\s&;]+","");
        ArrayList<String> toExecute = new ArrayList<>();
        toExecute.add("#!/bin/bash\n" +
                "ulimit -t 1\n");
        toExecute.add( "./planner " + "--optimise " + "domain.pddl
            "+building_id+"_temp.pddl");
        Path execPath = Paths.get("five-seconds");
        try {
            Files.write(execPath,toExecute, Charset.forName("UTF-8"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        String command = "./five-seconds";
        Process process = null;
        try {
            process = Runtime.getRuntime().exec(command);
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
        BufferedReader reader = new BufferedReader(new InputStreamReader(
                process.getInputStream()));
        ArrayList<String> output = new ArrayList<>();
        String s;
        try {
            while ((s = reader.readLine()) != null) {
                System.out.println("Script output: " + s);
                output.add(s);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return makeFinalResult(parseOutput(output),rooms);

    }
    public static ArrayList<Room> getRoomsFromJSON(JSONArray jsonArray){

        ArrayList<Room> toReturn = new ArrayList<>();
        for(Object o:jsonArray){
            JSONObject room = (JSONObject) o;
            Point lt = new
                Point(room.getJSONObject("rectangle").getJSONObject("lt").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("lt").getDouble("y"));
            Point rt = new
                Point(room.getJSONObject("rectangle").getJSONObject("rt").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("rt").getDouble("y"));
            Point lb = new
                Point(room.getJSONObject("rectangle").getJSONObject("lb").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("lb").getDouble("y"));
            Point rb = new
                Point(room.getJSONObject("rectangle").getJSONObject("rb").getDouble("x"),
                   room.getJSONObject("rectangle").getJSONObject("rb").getDouble("y"));
            RectangleDB r = new RectangleDB(lt,rt,lb,rb);
            Room toAdd = new
                Room(room.getString("_id"),room.getString("building_id"),room.getString("name"),r,
                   room.getDouble("width"),room.getDouble("height"),room.getDouble("est_time"),room.get
            toReturn.add(toAdd);
        }
        return toReturn;
    }

    private static void makePDDL(ArrayList<Room> rooms, int deadline){

        Path tempPath =
            Paths.get(rooms.get(0).getBuilding_id()+"_temp.pddl");
        ArrayList<String> toWrite = new ArrayList<>();
        toWrite.addAll(Arrays.asList(pddlIntro));
        toWrite.add("   ");
```

```
        for(Room r:rooms){
            toWrite.set(toWrite.size()-1,toWrite.get(toWrite.size()-1)+"e"+rooms.indexOf(r)+"
                ");
        }
        toWrite.set(toWrite.size()-1,toWrite.get(toWrite.size()-1)+"-
            exhibit");
        toWrite.add(")");
        toWrite.add("(:init");
        for(Room r:rooms){
            toWrite.add("(want-to-see "+"e"+rooms.indexOf(r)+")");
        }

        toWrite.addAll(getBuildingLayout(rooms));
        for(Room r:rooms){
            toWrite.add("(= (time-to-see "+"e"+rooms.indexOf(r)+")
                "+(int) r.getEst_time()+")");
        }
        toWrite.add("(at visitor "+"e0"+")");
        toWrite.add("(= (seen) 0)");
        for(Room r:rooms){
            toWrite.add("(= (excitement "+"e"+rooms.indexOf(r)+") "+
                r.getExcitement()+(")"));
        }
        toWrite.add("(open)");
        toWrite.add("(at "+deadline+" (not (open)))");
        toWrite.add(")");
        toWrite.add("(:goal (and");

        for(Room r:rooms){
            toWrite.add(";(visited "+"e"+rooms.indexOf(r)+")");
        }
        toWrite.add(")");
        toWrite.add(")");
        toWrite.add("(:metric maximize (seen))");
        toWrite.add(")");
        try {
            Files.write(tempPath,toWrite, Charset.forName("UTF-8"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static ArrayList<String> getBuildingLayout(ArrayList<Room>
        rooms){
        ArrayList<String> toReturn = new ArrayList<>();
        for(int i = 0; i<rooms.size()-1; ++i){
            for(int j=i+1; j<rooms.size();++j){
                if(rooms.get(i).getRectangleDB().isNeighbour(rooms.get(j).getRectangleDB())){
                    toReturn.add("(path " + "e"+i + " "+"e"+ j + ") (path
                        " +"e"+j+" "+"e"+i+")");
```

```
                toReturn.add("(= (time-to-walk "+"e"+i+" "+"e"+j+")
                    0)");
                toReturn.add("(= (time-to-walk "+"e"+j+" "+"e"+i+")
                    0)");
            }
        }
    }
    return toReturn;
}

private static ArrayList<String> parseOutput(ArrayList<String>
    output){
    ArrayList<String> toReturn = new ArrayList<>();
    ArrayList<String> currentResult = new ArrayList<>();

    for(String s:output){
        if(s.equals(";;;; Solution Found")) {
            toReturn = currentResult;
            currentResult.clear();
        }
        if(!((s.startsWith(";")|| s.isEmpty()))) {
            if (s.contains("(") && s.contains(")")) {
                currentResult.add( s.substring(s.indexOf("(")+1,
                    s.indexOf(")")));
            }
        }
        if(s.startsWith("..")){
            toReturn = currentResult;
        }
    }

   return toReturn;
}

private static String makeFinalResult(ArrayList<String> output,
    ArrayList<Room> rooms){
    String toReturn = "";
    for(String s:output){
        String[] tempSplit = s.split(" ");
        if(tempSplit[0].equals("view")){
            int roomIndex
                =Integer.parseInt(tempSplit[2].substring(tempSplit[2].indexOf("e")+1));
            toReturn += "view:"+rooms.get(roomIndex).getId()+";";

        } else if(tempSplit[0].equals("walk")){
            int roomIndexOr =
                Integer.parseInt(tempSplit[2].substring(tempSplit[2].indexOf("e")+1));
            int roomIndexDest =
                Integer.parseInt(tempSplit[3].substring(tempSplit[3].indexOf("e")+1));
```

```
                toReturn +=
                    "walk:"+rooms.get(roomIndexOr).getId()+","+rooms.get(roomIndexDest).getId()+";";
            }
        }

        return toReturn.substring(0,toReturn.length()-1);
    }
}
```

13 com/company/Server.java

```
import org.apache.commons.lang3.tuple.Pair;
import org.json.JSONObject;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.bayes.NaiveBayes;

import java.net.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import java.io.*;

public class Server
{
    String currentBID;
    ArrayList<Pair<String,BayesNet>> classifiersBN;
    ArrayList<Pair<String,NaiveBayes>> classifiersNB;
    public Server()
    {
        classifiersBN = new ArrayList<>();
        classifiersNB = new ArrayList<>();

        try {
            ServerSocket sSocket = new ServerSocket(5000);
            System.out.println("Server started!");
            currentBID = "";

            while(true) {
                Socket socket = sSocket.accept();
                ClientThread cT = new ClientThread(socket);
                new Thread(cT).start();
            }
        } catch(IOException exception) {
            System.out.println("Error: " + exception);
        }
    }

    class ClientThread implements Runnable
    {
        Socket socket;
        public ClientThread(Socket socket)
        {
            this.socket = socket;
        }

        public void run()
        {
```

```
try {
    PrintWriter pw = new
        PrintWriter(socket.getOutputStream(), true);
    BufferedReader br = new BufferedReader(new
        InputStreamReader(socket.getInputStream()));
    String rec = br.readLine();
    while (rec == null) {
        rec = br.readLine();
    }
    System.out.println(rec);
    JSONObject recJSON = new JSONObject(rec);
    if(recJSON.getString("command").equals("learn")){
        BayesNet bn =
            Learner.learnFromJSON_BN(recJSON.getString("building_id"),recJSON.getJSONArray('
        NaiveBayes nb =
            Learner.learnFromJSON_NB(recJSON.getString("building_id"),recJSON.getJSONArray('
        int i =
            buildingClassifierBNInitialised(recJSON.getString("building_id"));
        int j =
            buildingClassifierNBInitialised(recJSON.getString("building_id"));
            //not really needed.
        if(i>-1){
            classifiersBN.set(i,Pair.of(recJSON.getString("building_id"),bn));
        } else {
            classifiersBN.add(Pair.of(recJSON.getString("building_id"),bn));
        }

        if(i>-1){
            classifiersNB.set(i,Pair.of(recJSON.getString("building_id"),nb));
        } else {
            classifiersNB.add(Pair.of(recJSON.getString("building_id"),nb));
        }
        pw.write("Done!");
    } else
        if(recJSON.getString("command").equals("classify")){
        int i =
            buildingClassifierBNInitialised(recJSON.getString("building_id"));
        int j =
            buildingClassifierNBInitialised(recJSON.getString("building_id"));
            //same

        if(i>-1){
            String res
                ="BN:"+Learner.classify_BN(recJSON.getString("building_id"),
                    recJSON.getJSONArray("learning_set"),classifiersBN.get(i).getRight());
            res +=
                ",NB:"+Learner.classify_NB(recJSON.getString("building_id"),
                    recJSON.getJSONArray("learning_set"),classifiersNB.get(j).getRight());
            pw.write(res);
            System.out.println(res);
```

```
                } else
                    if(Files.exists(Paths.get(recJSON.getString("building_id")+".arff"))){
                    BayesNet bn =
                        Learner.getClassifierBN(recJSON.getString("building_id"));
                    NaiveBayes nb =
                        Learner.getClassifierNB(recJSON.getString("building_id"));
                    classifiersBN.add(Pair.of(recJSON.getString("building_id"),bn));
                    classifiersNB.add(Pair.of(recJSON.getString("building_id"),nb));

                    i =
                        buildingClassifierBNInitialised(recJSON.getString("building_id"));
                    j =
                        buildingClassifierNBInitialised(recJSON.getString("building_id"));
                        //same
                    String res
                        ="BN:"+Learner.classify_BN(recJSON.getString("building_id"),
                           recJSON.getJSONArray("learning_set"),classifiersBN.get(i).getRight());
                    res +=
                        ",NB:"+Learner.classify_NB(recJSON.getString("building_id"),
                           recJSON.getJSONArray("learning_set"),classifiersNB.get(j).getRight());
                    pw.write(res);
                } else {
                    pw.write("Weird classify request!");
                }
            }  else if(recJSON.getString("command").equals("route")){
                pw.write(Planner.route(recJSON.getJSONArray("request_set"),recJSON.getInt("deadline"
                       recJSON.getString("building_id")));
            }
            pw.flush();
        } catch(IOException exception) {
            System.out.println("Error: " + exception);
        }
    }
}

public int buildingClassifierBNInitialised(String building_id){
    for(Pair<String,BayesNet> p:classifiersBN){
        if(p.getLeft().equals(building_id))
            return classifiersBN.indexOf(p);
    }
    return -1;
}

public int buildingClassifierNBInitialised(String building_id){
    for(Pair<String,NaiveBayes> p:classifiersNB){
        if(p.getLeft().equals(building_id))
            return classifiersNB.indexOf(p);
    }
    return -1;
}
```

```
}
```

14 com/company/Main.java

```
public class Main {

    public static void main(String[] args) {
        new Server();
    }
}
```

15 com/company/Learner.java

```
import org.json.JSONArray;
import org.json.JSONObject;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instances;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.Set;


public class Learner {

    public static NaiveBayes learnFromJSON_NB(String building_id,
         JSONArray JSONData){
        makeLearnerARFFfromJSON(building_id,JSONData);
        NaiveBayes nb = new NaiveBayes();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            nb.buildClassifier(instances);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return nb;
    }

    public static BayesNet learnFromJSON_BN(String building_id,
         JSONArray JSONData){
        makeLearnerARFFfromJSON(building_id,JSONData);
        BayesNet bn = new BayesNet();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            bn.buildClassifier(instances);
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }

        return bn;
    }

    public static BayesNet getClassifierBN(String building_id){
        BayesNet bn = new BayesNet();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            bn.buildClassifier(instances);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return bn;
    }

    public static NaiveBayes getClassifierNB(String building_id){
        NaiveBayes nb = new NaiveBayes();
        try {
            Instances instances = new Instances(new BufferedReader(new
                FileReader(building_id+".arff")));
            instances.setClassIndex(instances.numAttributes()-1);
            nb.buildClassifier(instances);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return nb;
    }

    public static String classify_NB(String building_id, JSONArray
         JSONdata, NaiveBayes nb){
        makeClassifierARFF(building_id,JSONdata);
        Instances unlabeled;
        Instances labeled = null;
        double clsLabel = 0;
        try {
            unlabeled = new Instances(new BufferedReader(new
                FileReader(building_id+"_temp.arff")));
            unlabeled.setClassIndex(unlabeled.numAttributes()-1);
            labeled = new Instances(unlabeled);
            clsLabel = nb.classifyInstance(unlabeled.firstInstance());
            labeled.firstInstance().setClassValue(clsLabel);
            Files.deleteIfExists(Paths.get(building_id+"_temp.arff"));
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
        if (labeled != null) {
            return
                labeled.instance(0).classAttribute().value((int)clsLabel);
        }
        return "na";

    }

    public static String classify_BN(String building_id, JSONArray
         JSONdata, BayesNet bn){
        makeClassifierARFF(building_id,JSONdata);
        Instances unlabeled;
        Instances labeled = null;
        double clsLabel = 0;
        try {
            unlabeled = new Instances(new BufferedReader(new
                FileReader(building_id+"_temp.arff")));
            unlabeled.setClassIndex(unlabeled.numAttributes()-1);
            labeled = new Instances(unlabeled);
            clsLabel = bn.classifyInstance(unlabeled.firstInstance());
            labeled.firstInstance().setClassValue(clsLabel);
            Files.deleteIfExists(Paths.get(building_id+"_temp.arff"));
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (labeled != null) {
            return
                labeled.instance(0).classAttribute().value((int)clsLabel);
        }
        return "na";
    }

    public static void makeLearnerARFFfromJSON(String building_id,
         JSONArray JSONData){
        Path arffPath = Paths.get(building_id+".arff");
        Path roomsPath = Paths.get(building_id+"_rooms.data");
        Path rpPath = Paths.get(building_id+"_RPs.data");

        ArrayList<String> arffData = new ArrayList<>();
        ArrayList<String> rpids = getRPs(JSONData);
        ArrayList<String> rooms = getRooms(JSONData);

        arffData.add("@relation room"); arffData.add("");
        for(String s:rpids){
            arffData.add("@attribute "+s+" NUMERIC");
        }
        arffData.add("@attribute class {");
        for(int i=0;i<rooms.size()-1;++i){
            arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(i)+",");
```

```
        }
        arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(rooms.size()-1)+"}");
        arffData.add("@data");
        arffData.addAll(getOrderedReadings(JSONData,rpids));
        try {
            Files.write(arffPath,arffData, Charset.forName("UTF-8"));
            Files.write(roomsPath,rooms, Charset.forName("UTF-8"));
            Files.write(rpPath,rpids, Charset.forName("UTF-8"));

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public static void makeClassifierARFF(String building_id,JSONArray
         JSONData){

        Path tempPath = Paths.get(building_id+"_temp.arff");
        Path roomsPath = Paths.get(building_id+"_rooms.data");
        Path rpPath = Paths.get(building_id+"_RPs.data");

        ArrayList<String> arffData = new ArrayList<>();
        ArrayList<String> rpids = new ArrayList<>();
        ArrayList<String> rooms = new ArrayList<>();

        try {
            BufferedReader br =
                 Files.newBufferedReader(roomsPath,Charset.forName("UTF-8"));
            String line;
            while((line = br.readLine()) != null){
                rooms.add(line);
            }
            br = Files.newBufferedReader(rpPath,Charset.forName("UTF-8"));
            while((line = br.readLine()) != null){
                rpids.add(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        arffData.add("@relation room"); arffData.add("");
        for(String s:rpids){
            arffData.add("@attribute "+s+" NUMERIC");
        }
        arffData.add("@attribute class {");
        for(int i=0;i<rooms.size()-1;++i){
            arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(i)+",");
        }
        arffData.set(arffData.size()-1,arffData.get(arffData.size()-1)+rooms.get(rooms.size()-1)+"}");
        arffData.add("@data");
```

```
        arffData.addAll(getOrderedReadings(JSONData,rpids));
        try {
            Files.write(tempPath,arffData, Charset.forName("UTF-8"));
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public static ArrayList<String> getRPs(JSONArray JSONdata){
        ArrayList<String> rpids = new ArrayList<>();
        for(Object reading: JSONdata){
            for(Object pair:
                 ((JSONObject)reading).getJSONArray("rpv_pair")){
                rpids.add(((JSONObject) pair).getString("RPID"));
            }
        }
        Set<String> noDuplicates = new LinkedHashSet<>(rpids);
        rpids.clear();
        rpids.addAll(noDuplicates);
        return rpids;
    }

    public static ArrayList<String> getRooms(JSONArray JSONdata){
        ArrayList<String> rooms = new ArrayList<>();
        for(Object reading: JSONdata){
            rooms.add(((JSONObject) reading).getString("room_id"));
        }
        Set<String> noDuplicates = new LinkedHashSet<>(rooms);
        rooms.clear();
        rooms.addAll(noDuplicates);
        return rooms;
    }

    public static ArrayList<String> getOrderedReadings(JSONArray
         jsonData, ArrayList<String> rpids){
        ArrayList<String> toReturn= new ArrayList<>();
        for(Object o:jsonData){
            String curLine = "";
            for(String rp:rpids){
                curLine += Double.toString(getValueByRPID(((JSONObject)
                     o),rp))+",";

            }
            curLine += ((JSONObject) o).getString("room_id");
            toReturn.add(curLine);
        }

        return toReturn;
    }
```

```
    public static double getValueByRPID(JSONObject reading, String RPID){
        JSONArray pairs = reading.getJSONArray("rpv_pair");
        for(Object pair:pairs){
            if(RPID.equals(((JSONObject) pair).getString("RPID")))
                return ((JSONObject) pair).getDouble("value");
        }

        return 0;
    }
}
```