**“Semi-Lazy” Trajectory Prediction Package**

**Version 0.2**

**Data: 2015.03.06**

**Title:** A Semi-Lazy Approach to Probabilistic Path Prediction in Dynamic Environments

**Author:** zhou jingbo ([jzhousoc@gmail.com](mailto:jzhousoc@gmail.com))

**Maintainer:** zhou jingbo ([jzhousoc@gmail.com](mailto:jzhousoc@gmail.com))

**Description:** This is source code for a project of trajectory prediction in dynamic environments. For more information please refer to:

Jingbo Zhou, [Anthony K. H. Tung](http://www.comp.nus.edu.sg/~atung/), [Wei Wu](https://researchers.a-star.edu.sg/userprofile.aspx?userid=7437), [Wee Siong Ng](http://www1.i2r.a-star.edu.sg/~wsng/); "[A Semi-Lazy Approach to Probabilistic Path Prediction in Dynamic Environments](http://www.comp.nus.edu.sg/~jzhou/paper/Semilazy2013zhou_kdd.pdf)"; Proc. of 2013 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (**KDD 2013**).

Jingbo Zhou, [Anthony K. H. Tung](http://www.comp.nus.edu.sg/~atung/), [Wei Wu](https://researchers.a-star.edu.sg/userprofile.aspx?userid=7437), [Wee Siong Ng](http://www1.i2r.a-star.edu.sg/~wsng/); "[R2-D2: a System to Support Probabilistic Path Prediction in Dynamic Environments via Semi-Lazy Learning](http://www.comp.nus.edu.sg/~jzhou/paper/R2D22013zhou_vldb_demo.pdf)"; Proc. of 2013 Int. Conf. on Very Large Databases (**VLDB 2013**). [Demo paper, [Project website](http://db128gb-b.ddns.comp.nus.edu.sg/jzhou/R2-D2/)]

Website: <http://db128gb-b.ddns.comp.nus.edu.sg/jzhou/R2-D2/>

**License: Private (forbid to distribute without permission)**

**Dependence:** JDK 7 or above, the project is originally created by Eclipse.

**Prediction method – R2D2 (semi-lazy learning approach)**

**Example code:** (see function **static void expPredictionErr\_R2D2()** in class **DTPredictor/src/main/Demo.java**)

**Quick run:**

Run the main() method in class DTPredictor/src/main/Demo.java ( Note: set the SVM heap size as 1024m), You will see a visualization window to show whether the data is correctly loaded ( see Figure 1)

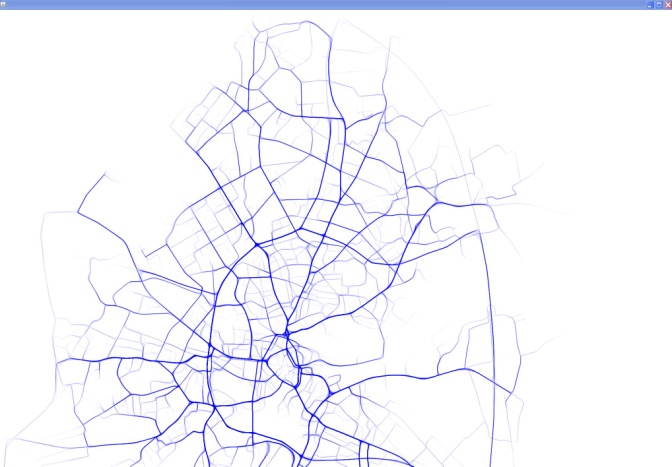


Figure Visualization of the data

Then waiting a while, you can get the output:

**#time count Rate DT\_error**

**1 774 0.8313641245972073 135.75238008645752**

**2 600 0.644468313641246 184.16555482302968**

**3 288 0.30934479054779807 196.6588641071093**

**4 111 0.11922663802363051 197.33903900210606**

**5 31 0.03329752953813104 154.73642376195758**

**….**

In the table, the first column is time step, the second column is the number of predictions made by the program, the third and forth columns show the performance measures- prediction rate and prediction error (refer to section 7.1 in KDD2013 paper).

**Method**

**Update Process:** In this process, we load the data from database into our Trajectory Grid (TG) structures. (Refer to section 4 in KDD2013 paper.)

**Load data:** The trajectory data is stored in the Sqlite3 database, the schema of the table is:

CREATE TABLE BBFOldTest(type varchar(20), id INTEGER, seq INTEGER, class INTEGER, t INTEGER, x FLOAT, y FLOAT, speed FLOAT, nextX INTEGER, nextY INTEGER)

An example code for loading the data into the system is as follows:

//========

**System.*out*.println("#start loading data and sample queries ");**

**Grid g=tl.Load2Grid("data/BigBrinkhoff/bigBrinkhoff.db", "BBFOldTest",timeStart ,timeEnd);**

**System.*out*.println("#finished loading data");**

**//========**

After this, we load the data from sqlite database into our Trajectory Grid structure, which is designed to support quickly search of similar trajectories. (At the same time, we also get some sample queries as test data.)

**Prediction Process:** The “Prediction” process makes path prediction. This process has two sub-processes: “Lookup” and “Construction”. (Refer to section 3.1 in KDD2013 paper.)

**Lookup:** we use the trajectory of predicted objects in the last few time steps as query trajectory to retrieve reference trajectories from TG (refer to section 5). An example code of “Lookup” is:

**//=====**

**//get the trajectory of predicted object**

**ArrayList<RoICell> ref=new ArrayList<RoICell>(rcGridList.subList(refTime-DBBackStep+1, refTime+1));**

**//lookup, retrieve the similar trajectories.**

**GridLeafTraHashItem>> testBres=g.queryRangeTimeSeqCells(ref); //=====**

**Construction:** In the “Construction” process, the reference trajectories are used to construct a model for making path prediction (refer to section 6). An example code of “**Construction**” is:

**//======**

**// make prediction**

**StateGridFilter sgf=pdr.PathPrediction(testBres,g,Configuration.*ProDown*,Configuration.*MAPPro*,Configuration.*MicroStateRadius*);**

**//retrieve predicted path**

**ArrayList<MacroState> mp=sgf.gfStates.getMacroStatePath();**

**//=====**

**Parameter configuration:** All the important parameters are configured in src/grid/Configuration.java. Here I give a brief description of the parameters.

**//=====**

**//define how to create the Trajectory Grid, refer to KDD2013 paper section 4**

**Configuration.*BITS\_PER\_GRID*=4;**

**Configuration.*MAX\_LEVEL*=3;**

**Configuration.*GridDivided*=2048;**

**//define the min and max value of the prediction area**

**Configuration.*BBFOldXMin*=292.0;**

**Configuration.*BBFOldYMin*=3935.0;**

**Configuration.*BBFOldXMax*=23056.0;**

**Configuration.*BBFOldYMax*=30851.0;**

**//the period of trajectories which we think they are not expired, refer to the last paragraph in section 4 in KDD2013.**

**Configuration.*T\_period*=200;**

**Configuration.BrinkConstraintRoI=16;**

**//the support of reference trajectories. How many minimum trajectories are required, refer to Algorithm 1 in KDD2013.**

**Configuration.*TraSupport*=1;**

**//define parameter related with state space tree (refer to Section 6.2 in KDD2013 paper)**

**Configuration.*MaxRadius*=5000; //maximum size of state, not important parameters. It is defined on real distance**

**Configuration.*MaxStateDis*=500;//same above not important parameters.**

**Configuration.*AlphaRadius*=1.5;//should be larger than 1,**

**Configuration.*AlphaScore*=1/16.0;// alpha, refer to section 6.2.2 in KDD2013**

**Configuration.*ProDown*=0.2;//theta, refer to the last paragraph in section 6.2.2 in KDD2013**

**Configuration.*MAPPro*=0.2;// probability conﬁdence threshold, see Algorithm 1 in KDD2013**

**Configuration.*MicroStateRadius*=Configuration.*cellRadius*\*2;//size of micro cluster, see section 6.2.3 in KDD2013.**

**//====**

**Example code:** (see function **static void expPredictionErr\_R2D2()** in class **DTPredictor/src/main/Demo.java**)