java-genetic-programming Documentation

Release 1.0.3

Xianshun Chen

Oct 04, 2017

Contents

java-genetic-programming is a library of java implementation for algorithms in the fields of Genetic Programming.

The main purpose of this library is to provide java developers with a tool set of genetic programming techniques

CHAPTER 1

Installation:

To install the package using maven, add the following dependency to your POM file:

```xml
<dependency>
  <groupId>com.github.chen0040</groupId>
  <artifactId>java-genetic-programming</artifactId>
  <version>1.0.7</version>
</dependency>
```

CHAPTER 2

Usage

To use the algorithms or data structures in your java code:

```
import com.github.chen0040.gp.lgp.LGP;
import com.github.chen0040.gp.commons.BasicObservation;
import com.github.chen0040.gp.commons.Observation;
import com.github.chen0040.gp.lgp.gp.Population;
import com.github.chen0040.gp.lgp.program.operators.*;

LGP lgp = new LGP();
lgp.getOperatorSet().addAll(new Plus(), new Minus(), new Divide(), new Multiply(),␣
↪new Power());
lgp.getOperatorSet().addIfLessThanOperator();
lgp.addConstants(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0);
lgp.setRegisterCount(6);

lgp.setCostEvaluator((program, observations)->{
 double error = 0;
 for(Observation observation : observations){
    program.execute(observation);
    error += Math.pow(observation.getOutput(0) - observation.getPredictedOutput(0), 2.
↪0);
 }

 return error;
});

Program program = lgp.fit(trainingData);

logger.info("best solution found: {}", program);
```

CHAPTER 3

Features

- Linear Genetic Programming
- Tree Genetic Programming (Coming Soon)
- Grammatical Evolution (Coming Soon)
- Gene Experssion Programming (Coming Soon)
- NSGA-II Genetic Programming for multi-objective optimization

CHAPTER 4

Tests:

the unit tests of all algorithms and data structures can be run with the following command from the root folder:

```
$ /mvnw test -Punit-test
```

CHAPTER 5

Contributing:

Contributions are always welcome. Check out the contributing guidelines to get started.

CHAPTER 6

Table of Contents:

Linear Genetic Programming

Supported Features in Linear Genetic Programming

- Linear Genetic Programming
  - Crossover
    - Linear
    - One-Point
    - One-Segment
  - Mutation
    - Micro-Mutation
    - Effective-Macro-Mutation
    - Macro-Mutation
  - Replacement
    - Tournament
    - Direct-Compete
  - Default-Operators
    - Most of the math operators
    - if-less, if-greater
    - Support operator extension

Usage: Create training data

The sample code below shows how to generate data from the “Mexican Hat” regression problem:

```
import com.github.chen0040.gp.commons.BasicObservation;
import com.github.chen0040.gp.commons.Observation;
import java.util.ArrayList;
import java.util.List;
import java.util.function.BiFunction;

private List<Observation> ProblemCatalogue.mexican_hat(){
  List<Observation> result = new ArrayList<>();

  BiFunction<Double, Double, Double> mexican_hat_func = (x1, x2) -> (1 - x1 * x1 / 4 -
↪ x2 * x2 / 4) * Math.exp(- x1 * x2 / 8 - x2 * x2 / 8);

  double lower_bound=-4;
  double upper_bound=4;
  int period=16;

  double interval=(upper_bound - lower_bound) / period;

  for(int i=0; i<period; i++)
  {
     double x1=lower_bound + interval * i;
     for(int j=0; j<period; j++)
     {
        double x2=lower_bound + interval * j;

        Observation observation = new BasicObservation(2, 1);

        observation.setInput(0, x1);
        observation.setInput(1, x2);
        observation.setOutput(0, mexican_hat_func.apply(x1, x2));

        result.add(observation);
     }
  }

  return result;
}
```

We can split the data generated into training and testing data:

```
import com.github.chen0040.gp.utils.CollectionUtils;

List<Observation> data = ProblemCatalogue.mexican_hat();
CollectionUtils.shuffle(data);
TupleTwo<List<Observation>, List<Observation>> split_data = CollectionUtils.
↪split(data, 0.9);
List<Observation> trainingData = split_data._1();
List<Observation> testingData = split_data._2();
```

Usage: Create and train the LGP

The sample code below shows how the LGP can be created and trained:

The last line prints the linear program found by the LGP evolution, a sample of which is shown below:

```
instruction[1]: <If<        r[4]    c[0]    r[4]>
instruction[2]: <If<        r[3]    c[3]    r[0]>
instruction[3]: <-  r[2]    r[3]    r[2]>
instruction[4]: <*  c[7]    r[2]    r[2]>
instruction[5]: <If<        c[2]    r[3]    r[1]>
instruction[6]: </  r[1]    c[4]    r[2]>
instruction[7]: <If<        r[3]    c[7]    r[1]>
instruction[8]: <-  c[0]    r[0]    r[0]>
instruction[9]: <If<        c[7]    r[3]    r[4]>
instruction[10]: <- r[2]    c[3]    r[1]>
instruction[11]: <+ c[4]    r[4]    r[5]>
instruction[12]: <If<       c[2]    r[5]    r[1]>
instruction[13]: <+ c[7]    r[0]    r[5]>
instruction[14]: <^ c[7]    r[4]    r[3]>
instruction[15]: <If<       c[3]    r[1]    r[3]>
instruction[16]: <If<       r[1]    r[0]    r[5]>
instruction[17]: <* c[7]    r[2]    r[2]>
instruction[18]: <^ r[1]    c[6]    r[3]>
instruction[19]: <If<       r[0]    c[5]    r[0]>
instruction[20]: <- c[3]    r[1]    r[3]>
instruction[21]: <If<       r[3]    c[8]    r[0]>
instruction[22]: </ c[2]    r[4]    r[5]>
instruction[23]: <If<       r[3]    c[7]    r[3]>
instruction[24]: <+ r[0]    c[1]    r[0]>
instruction[25]: <* r[0]    c[6]    r[0]>
instruction[26]: <- r[3]    c[7]    r[1]>
instruction[27]: <- r[4]    c[7]    r[4]>
instruction[28]: <If<       c[1]    r[4]    r[4]>
instruction[29]: <- c[1]    r[0]    r[2]>
instruction[30]: </ c[3]    r[4]    r[3]>
instruction[31]: <If<       c[8]    r[2]    r[2]>
instruction[32]: </ r[1]    c[2]    r[3]>
instruction[33]: <If<       r[0]    c[2]    r[1]>
instruction[34]: <- c[2]    r[2]    r[5]>
instruction[35]: <If<       c[7]    r[5]    r[1]>
instruction[36]: <If<       r[2]    c[5]    r[2]>
instruction[37]: <- r[5]    c[7]    r[3]>
instruction[38]: <- c[8]    r[3]    r[3]>
instruction[39]: <^ c[3]    r[0]    r[5]>
```

Usage: Test the program obtained from the LGP evolution

The best program in the LGP population obtained from the training in the above step can then be used for prediction, as shown by the sample code below:

```java
logger.info("global: {}", program);

for(Observation observation : testingData) {
 program.execute(observation);
 double predicted = observation.getPredictedOutput(0);
 double actual = observation.getOutput(0);

 logger.info("predicted: {}\tactual: {}", predicted, actual);
}
```