InferPy Documentation

Release 1.0

Rafael Cabañas

Nov 13, 2018

General Information


AMiDST
TOOLBOX

CHAPTER 1

What is AMIDST?

AMIDST is an open source Java toolbox for scalable probabilistic machine learning with a special focus on (massive) streaming data. The toolbox allows specifying probabilistic graphical models with latent variables and temporal dependencies.

The main features of the tooblox are listed below:

- **Probabilistic Graphical Models**: Specify your model using probabilistic graphical models with latent variables and temporal dependencies. AMIDST contains a large list of predefined latent variable models:
- **Scalable inference**: Perform inference on your probabilistic models with powerful approximate and scalable algorithms.
- **Data Streams**: Update your models when new data is available. This makes our toolbox appropriate for learning from (massive) data streams.
- **Large-scale Data**: Use your defined models to process massive data sets in a distributed computer cluster using Apache Flink or (soon) Apache Spark.
- **Extensible**: Code your models or algorithms within AMiDST and expand the toolbox functionalities. Flexible toolbox for researchers performing their experimentation in machine learning.
- **Interoperability**: Leverage existing functionalities and algorithms by interfacing to other software tools such as Hugin, MOA, Weka, R, etc.

1.1 About AMIDST

1.1.1 What is AMIDST?

AMIDST is an open source Java toolbox for scalable probabilistic machine learning with a special focus on (massive) streaming data. The toolbox allows specifying probabilistic graphical models with latent variables and temporal dependencies.

For start using AMIDST, just visit the Getting-started section.

The main features of the toolbox are listed below:

- **Probabilistic Graphical Models**: Specify your model using probabilistic graphical models with latent variables and temporal dependencies. AMIDST contains a large list of predefined latent variable models:
- **Scalable inference**: Perform inference on your probabilistic models with powerful approximate and scalable algorithms.
- **Data Streams**: Update your models when new data is available. This makes our toolbox appropriate for learning from (massive) data streams.
- **Large-scale Data**: Use your defined models to process massive data sets in a distributed computer cluster using Apache Flink or (soon) Apache Spark.
- **Extensible**: Code your models or algorithms within AMiDST and expand the toolbox functionalities. Flexible toolbox for researchers performing their experimentation in machine learning.
- **Interoperability**: Leverage existing functionalities and algorithms by interfacing to other software tools such as Hugin, MOA, Weka, R, etc.

1.1.2 Scalability

Multi-Core Scalablity using Java 8 Streams

Scalability is a main concern for the AMIDST toolbox. Java 8 streams are used to provide parallel implementations of our learning algorithms. If more computation capacity is needed to process data, AMIDST users can also use more CPU cores. As an example, the following figure shows how the data processing capacity of our toolbox increases given the number of CPU cores when learning an a probabilistic model (including a class variable C, two latent variables (LM, LG), multinomial (M1,. . . ,M50) and Gaussian (G1,. . . ,G50) observable variables) using the AMIDST's learning engine. As can be seen, using our variational learning engine, AMIDST toolbox is able to process data in the order of gigabytes (GB) per hour depending on the number of available CPU cores with large and complex PGMs with latent variables. Note that, these experiments were carried out on a Ubuntu Linux server with a x86_64 architecture and 32 cores. The size of the processed data set was measured according to the Weka's ARFF format.

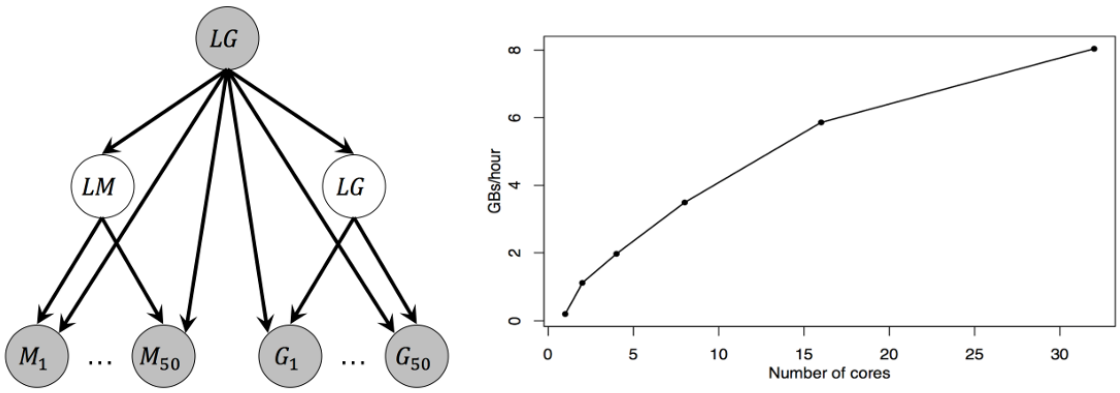
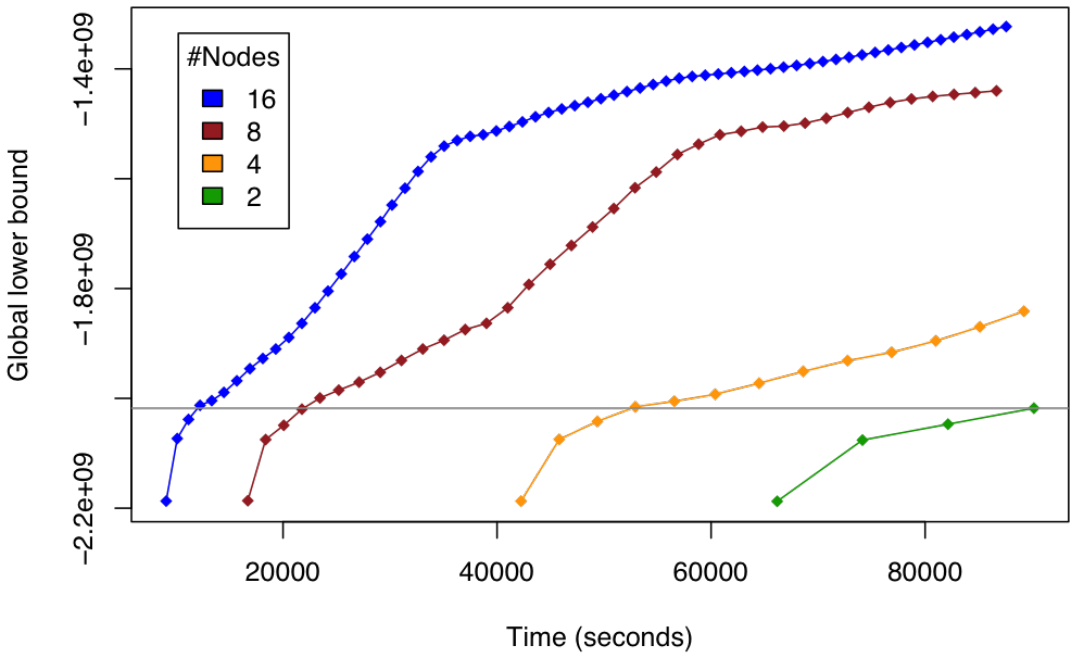


Fig. 1: Results in a multicore CPU

Distributed Scalablity using Apache Flink

If your data is really big and can not be stored in a single laptop, you can also learn your probabilistic model on it by using the AMIDST distributed learning engine based on a novel and state-of-the-art distributed message passing scheme implemented on top of Apache Flink. we were able to perform inference in a billion node (i.e. 10^9) probabilistic model in an Amazon's cluster with 2, 4, 8 and 16 nodes, each node containing 8 processing units. The following figure shows the scalability of our approach under these settings.



1.1.3 Related Software

Numerous tools for graphical models have been published during the last three decades. See this link for an updated list. In the following table, the main features of the AMIDST toolbox are compared against other related tools.

The vast majority of the similar tools do not support scalable inference and learning algorithms. To the best of our knowledge, this is the first PGM based software tool which is able to learn general Bayesian networks from streaming data: the rest of existing software for learning PGMs tools focus on stationary data sets. Moreover, AMIDST is able to exploit modern computing distributed processing tools like Apache Flink and Apache Spark. In the era of Big Data, this is a relevant and worthy feature. Even though MOA and ML Lib can process data streams in a distributed way, these tools do not allow learning PGMs.

On the other hand, most of these toolboxes only deal with discrete-multinomial variables, and only few of them also support Gaussian and conditional lineal Gaussian random variables such as Hugin. However, AMIDST Toolbox can deal with general conjugate exponential Bayesian networks. In the current version, we support multinomial, Gaussian, Dirichlet, and Gamma distributed random variables, and the implemented Bayesian learning algorithms are general enough to accommodate any other distribution belonging to the conjugate exponential family. Infer.net can deal with distributions in this family, however this tool is a proprietary non-portable software which cannot be run in a cluster.

1.2 Sparklink: Code Examples

1.2.1 Input/output

Reading data

In this example we show how can we read a dataset using sparklink. This module supports reading files in formats *json* and *parquet*. Note that the method *DataSparkLoader::open* automatically detects the format of the file (the indicated path should contain the extension). Finally all the instances in the data set are printed.

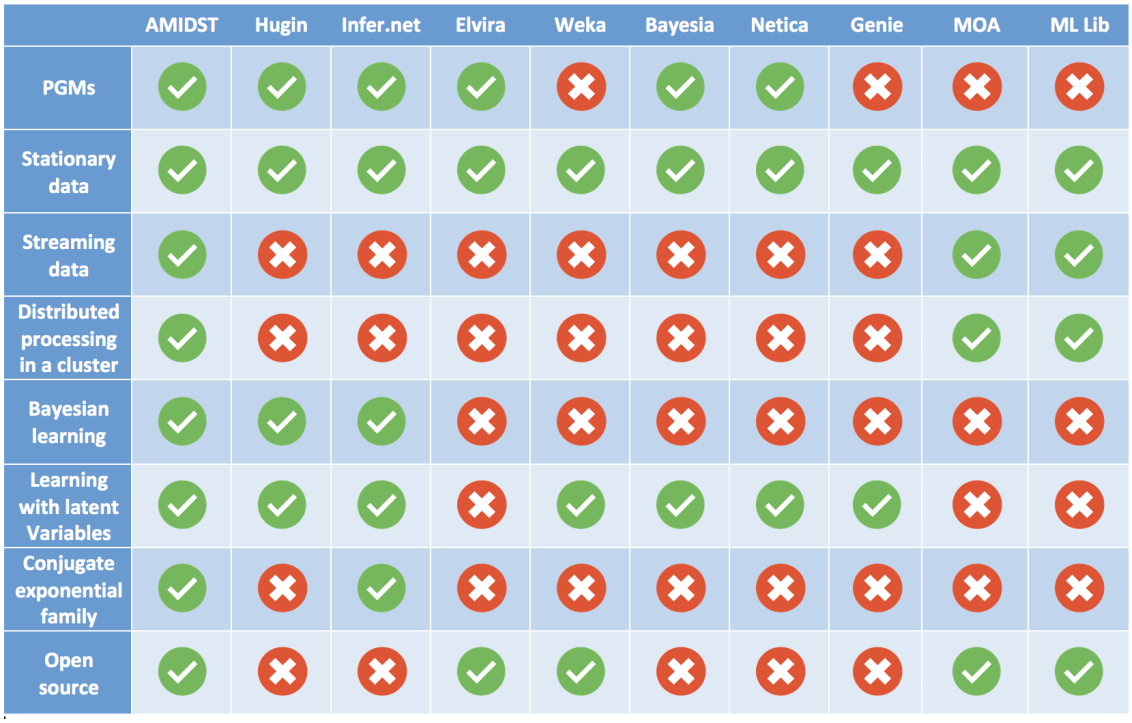


Fig. 2: Features of the related tools

```java
package eu.amidst.sparklink.examples.io;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import org.apache.spark.SparkContext;
import org.apache.spark.SparkConf;
import org.apache.spark.sql.DataFrame;
import org.apache.spark.sql.SQLContext;
import eu.amidst.sparklink.core.data.DataSpark;
import eu.amidst.sparklink.core.io.DataSparkLoader;

/**
 * Created by rcabanas on 10/06/16.
 */
public class DataStreamLoaderExample {
    public static void main(String[] args) throws Exception {
        SparkConf conf = new SparkConf().setAppName("SLink!").setMaster("local");
        SparkContext sc = new SparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);

        //Path to dataset
        String path ="datasets/simulated/WI_samples.json";

                //Create an AMIDST object for managing the data
        DataSpark dataSpark = DataSparkLoader.open(sqlContext, path);


                //Print all the instances in the dataset
        dataSpark.collectDataStream()
```

(continues on next page)

(continued from previous page)

```
                .forEach(
                        dataInstance -> System.out.println(dataInstance)
                );


    }
}
```

Writing data

Here we show how can we save spark data into a file. First a random data set is generated using the method *DataSetGenerator::generate*. Afterwards, such data is save using the method *DataSparkWriter::writeDataToFolder*

```
package eu.amidst.sparklink.examples.io;

import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.sparklink.core.data.DataSpark;
import eu.amidst.sparklink.core.io.DataSparkWriter;
import eu.amidst.sparklink.core.util.BayesianNetworkSampler;
import eu.amidst.sparklink.core.util.DataSetGenerator;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.SQLContext;

/**
 * Created by rcabanas on 30/09/16.
 */
public class DataStreamWriterExample {

        public static void main(String[] args) throws Exception {

                //Setting up spark
                SparkConf conf = new SparkConf().setAppName("SparkLink!").setMaster(
↪"local");
                JavaSparkContext jsc = new JavaSparkContext(conf);
                SQLContext sqlContext = new SQLContext(jsc);


                //Generate random data
                int seed = 1234;
                int nInstances = 1000;
                int nDiscreteAtts=3;
                int nContinuousAttributes = 2;

                DataSpark data = DataSetGenerator
                                .generate(        jsc,
                                                  seed,
                                                  nInstances,
                                                  nDiscreteAtts,
                                                  nContinuousAttributes );


                // Save it as a json and parquet file
                DataSparkWriter.writeDataToFolder(data, "datasets/simulated/
↪randomData.json", sqlContext);
```

(continues on next page)

(continued from previous page)

```
                DataSparkWriter.writeDataToFolder(data, "datasets/simulated/
↪randomData.parquet", sqlContext);

        }



}
```

1.2.2 Parameter learning

AMIDST provides parameter learning using spark with the *Maximum Likelihood* algorithm. In the following example, we load a data set in format json and we use it for learning a simple naive bayes (more complex DAGs can also be learnt).

```
package eu.amidst.sparklink.examples.learning;

import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.utils.DAGGenerator;
import eu.amidst.sparklink.core.data.DataSpark;
import eu.amidst.sparklink.core.io.DataSparkLoader;
import eu.amidst.sparklink.core.learning.ParallelMaximumLikelihood;
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.sql.DataFrame;
import org.apache.spark.sql.SQLContext;

/**
 * Created by rcabanas on 10/06/16.
 */
public class MaximumLikelihoodLearningExample {
    public static void main(String[] args) throws Exception {
        SparkConf conf = new SparkConf().setAppName("SparkLink!").setMaster("local");;
        SparkContext sc = new SparkContext(conf);
        SQLContext sqlContext = new SQLContext(sc);

        //Path to dataset
        String path ="datasets/simulated/WI_samples.json";

        //Create an AMIDST object for managing the data
        DataSpark dataSpark = DataSparkLoader.open(sqlContext, path);

        //Learning algorithm
        ParallelMaximumLikelihood parameterLearningAlgorithm = new␣
↪ParallelMaximumLikelihood();


        //We fix the BN structure
        DAG dag = DAGGenerator.getNaiveBayesStructure(dataSpark.getAttributes(), "W");

        parameterLearningAlgorithm.setDAG(dag);

        //We set the batch size which will be employed to learn the model in parallel
        parameterLearningAlgorithm.setBatchSize(100);
```

(continues on next page)

(continued from previous page)

```
        //We set the data which is going to be used for leaning the parameters
        parameterLearningAlgorithm.setDataSpark(dataSpark);
        //We perform the learning
        parameterLearningAlgorithm.runLearning();
        //And we get the model
        BayesianNetwork bn = parameterLearningAlgorithm.getLearntBayesianNetwork();

        System.out.println(bn);


    }
}
```

1.3 Wekalink: using an AMIDST classifier in Weka

One of the greatests points of AMIDST is the integration with other tools for data analysis. This is the case of Weka whose integration fuctionality is provided by the module *wekalink*. We will be able to create a wrapper for evaluating an AMDIST classifier with Weka.

1.3.1 Prepare your project

The first thing we have to do is to load the required AMIDST dependencies in a Maven project. In this case, we will have to load the modules **wekalink** and **latent-variable-models**. For that, add the following code to the file pom.xml of your project.

```
<dependencies>

  <dependency>
    <groupId>eu.amidst</groupId>
    <artifactId>wekalink</artifactId>
    <version> 0.7.2 </version>
    <scope>compile</scope>
 </dependency>

  <dependency>
    <groupId>eu.amidst</groupId>
    <artifactId>latent-variable-models</artifactId>
    <version> 0.7.2 </version>
    <scope>compile</scope>
  </dependency>

<!-- ... -->
</dependencies>
```

Further details for creating a project using AMIDST fuctionality is given in the Getting Started section.

1.3.2 Create the wrapper class

A custom classifier that could be handled by weka should inherit from class *weka.classifiers.AbstractClassifier* and implement interface *weka.core.Randomizable*. Thus we should override at least the following methods:

- *void buildClassifier(Instances data)*: builds the classifier from scratch with the given dataset.
- *double[] distributionForInstance(Instance instance)*: returns a vector containing the probability for each label or state of the class.

  Here below we show a minimal example where the Naive Bayes classifier provided by AMIDST is used.

```
import eu.amidst.core.datastream.Attributes;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataOnMemoryListContainer;
import eu.amidst.core.datastream.filereaders.DataInstanceFromDataRow;
import eu.amidst.latentvariablemodels.staticmodels.classifiers.NaiveBayesClassifier;
import eu.amidst.wekalink.converterFromWekaToAmidst.Converter;
import eu.amidst.wekalink.converterFromWekaToAmidst.DataRowWeka;
import weka.classifiers.AbstractClassifier;
import weka.core.Instance;
import weka.core.Instances;


public class AmidstNaiveBayes extends AbstractClassifier  {

    private NaiveBayesClassifier model = null;
    private Attributes attributes;

    @Override
    public void buildClassifier(Instances data) throws Exception {

        attributes = Converter.convertAttributes(data.enumerateAttributes(),
                                data.classAttribute());
        DataOnMemoryListContainer<DataInstance> dataAmidst =
                    new DataOnMemoryListContainer(attributes);

        data.stream()
            .forEach(instance -> dataAmidst.add(
                new DataInstanceFromDataRow(
                    new DataRowWeka(instance, attributes)))
        );

        model = new NaiveBayesClassifier(attributes);
        model.updateModel(dataAmidst);
    }


    @Override
    public double[] distributionForInstance(Instance instance) throws Exception {
        DataInstance amidstInstance =
            new DataInstanceFromDataRow(new DataRowWeka(instance,
                                this.attributes));
        return model.predict(amidstInstance).getParameters();

    }

}
```

Note that previous code does not implement neither the learning nor the classification processes, it simply calls to the corresponding methods *eu.amidst.latentvariablemodels.NaiveBayesClassifier* performing such tasks.

1.3.3 Testing the AMIDST classifier in Weka

Now we can evaluate an AMIDST classifier using only calls to functions from Weka. Here we show an example where we load a dataset in format .arff, we learn a naive Bayes classifier and we show the confusion matrix:

```
//Load the dataset
BufferedReader reader =
    new BufferedReader(new FileReader("exampleDS_d5_c0.arff"));
Instances data = new Instances(reader);
data.setClassIndex(6);

//Learn and evaluate the classifier
Evaluation eval = new Evaluation(data);
Debug.Random rand = new Debug.Random(1);
int folds = 10;
Classifier cls = new AmidstNaiveBayes();
eval.crossValidateModel(cls, data, folds, rand);

//Print the confusion matrix
System.out.println(eval.toMatrixString());
```

1.4 Tutorial: Easy Machine Learning with Latent Variable Models in AMIDST

In AMIDST toolbox 0.4.2 the module *latent-variable-models*, that contains a wide range of predefined latent variable models (see table below), was included. In this tutorial we will show how the use of this module simplifies the learning and inference processes. In fact, you will be able to learn your model and to perform inference with a few lines of code.

	STATIC	DYNAMIC
Maximum Likelihood	• Naïve Bayes • TAN • (G)AODE/HODE	• Dynamic NB
Bayesian learning	• Gaussian Discriminant Analysis • Latent Classification Models (LCM) • Multivariate Gaussian Mixture • Multivariate Gaussian Distribution • Bayesian Linear Regression • Factor Analysis • Mixture of FA	• Dynamic LCM • Hidden Markov Model (HMM) • Kalman Filter (KF) • Switching KF • Factorial HMM • Auto-regressive HMM • Input-Output HMM

Fig. 3: Set of predefined latent variable models in AMIDST

Besides of the simplicity, the required code for learning a latent variable model is also flexible: you will be able to change the learnt model or the inference algorithm just with some slight modfications in the code. Another advantage of using AMIDST for learning one of the predefined models is that the procedure is traparent to the format of your training data: you will use the same methods regardless of learning from a local or a distributed dataset (with Flink). Note that this last feature was included in the version 0.5.0 of the toolbox.

1.4.1 Setting up

In order to follow this tutorial, you will need to have the java 8 (i.e. SDK 1.8 or higher) installed in your system. For more details about the system requirements, see this link. Additionally, you can download a ready-to-use IntelliJ maven project with all the code examples in this tutorial. For that, use the following command:

```
$ git clone https://github.com/amidst/tutorial.git
```

Alternativelly, you would rather create a new maven project or use an existing one. For that, you might check the Getting Started page.

Note that this project does not contain any AMIDST source or binary, it only has some .java files using the AMIDST functionallity. Instead, each of the AMIDST modules are provided through maven. Doing that, the transitive dependences of the AMIDST toolbox are also downloaded in a transparent way for the user. An scheme of this idea is shown below:

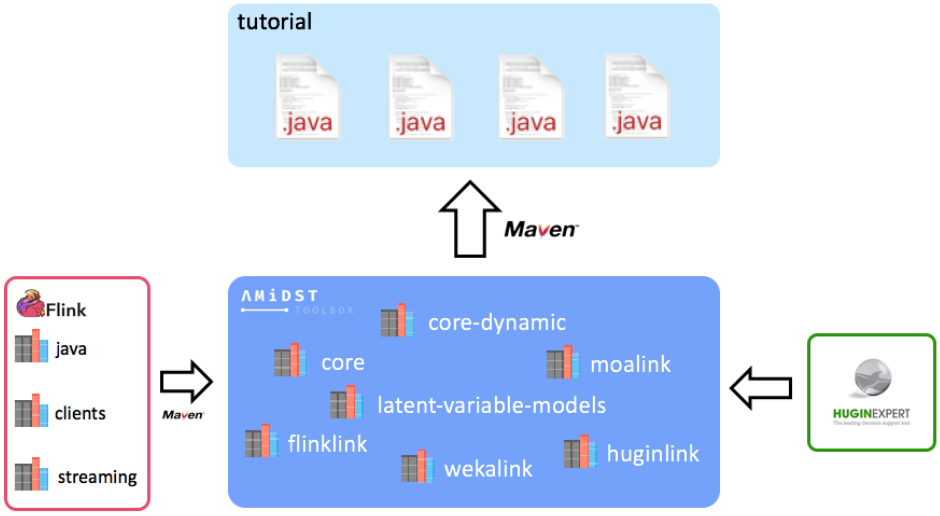


Fig. 4: Set of predefined latent variable models in AMIDST

If we open the downloaded project, we will see that it contains the following relevant folders and files:

- **datasets**: folder with local and distributed datasets used in this tutorial in ARFF format.
- **doc**: folder containing documentation about this tutorial.
- **lib**: folder for storing those libraries not available through maven.
- **src/main/java**: folder with all the code example.
- **pom.xml**: this is the maven configuration file where the AMIDST dependencies are defined.

In the pom.xml file of the downloaded project, only the module called *latent-variable-models* is linked. However some other AMIDST are also loaded as *latent-variable-models* module depends on them. This is the case of the modules called *core*, *core-dynamic*, *flinklink*, etc. You can see the full list of dependencies in the **maven project panel**, usually located on the right side of the window (see image below). If dependencies are not automatically downloaded, click on **Upload** button. It is recommended to download the sources and java

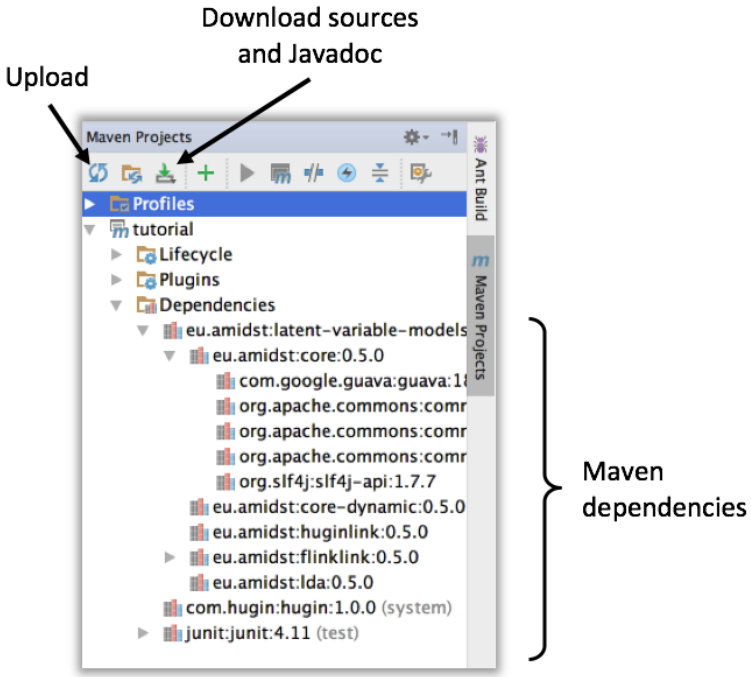


Fig. 5: Maven project panel showing the list of dependencies

1.4.2 Static Models

Learning and saving to disk

Here we will see how can we learnt a static model from a local dataset (non-distributed). In particular, we will use the financial-like dataset **datasets/simulated/cajamar.arff** containing 4 continuous (normal distributed) variables. From this data, a *Factor Analysis* model will be learnt. In short, this model aims to reduce the dimensionality of a set of observed continuous variables by expressing them as combination of gaussians. A synthesis of this process is shown in the image below where: $X1, X2, X3$ and $X4$ are the observed variables and $H1, H2$ and $H3$ are the latent variables representing the combination of gaussians.

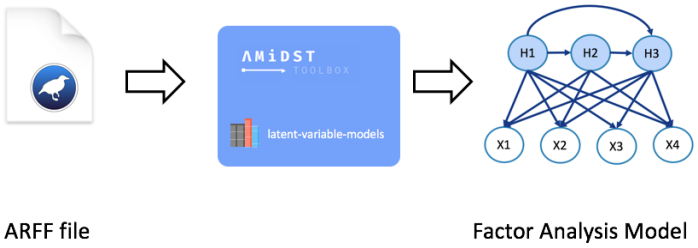


Fig. 6: Static learning process

Note that the required functionality for learning the predefined model is provided by the module *latent-variable-models*. A code-example for learning a factor analysis model is shown below.

```
package eu.amidst.tutorial.usingAmidst.examples;



import COM.hugin.HAPI.ExceptionHugin;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.BayesianNetworkWriter;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.huginlink.io.BayesianNetworkWriterToHugin;
import eu.amidst.latentvariablemodels.staticmodels.FactorAnalysis;
import eu.amidst.latentvariablemodels.staticmodels.MixtureOfFactorAnalysers;
import eu.amidst.latentvariablemodels.staticmodels.Model;

import java.io.IOException;

/**
 * Created by rcabanas on 23/05/16.
 */
public class StaticModelLearning {
    public static void main(String[] args) throws ExceptionHugin, IOException {

        //Load the datastream
        String filename = "datasets/simulated/cajamar.arff";
        DataStream<DataInstance> data = DataStreamLoader.open(filename);

        //Learn the model
        Model model = new FactorAnalysis(data.getAttributes());
       // ((MixtureOfFactorAnalysers)model).setNumberOfLatentVariables(3);
```

(continues on next page)

(continued from previous page)

```
        model.updateModel(data);
        BayesianNetwork bn = model.getModel();

        System.out.println(bn);

        // Save with .bn format
        BayesianNetworkWriter.save(bn, "networks/simulated/exampleBN.bn");

        // Save with hugin format
        //BayesianNetworkWriterToHugin.save(bn, "networks/simulated/exampleBN.net");
    }

}
```

See on GitHub

For learning any of the available static models, we create an object of any of the classes inheriting from the class *Model*. These classes encapsulates all the fuctionality for learning/updating a latent-variable model. For example, in the code above we create an object of the class *FactorAnalysis* which is actually stored as *Model* object. The flexibility of the toolbox is due to this hierarchical desing: if we aim to change the model learnt from our data, we simply have to change the constructor used (assuming that our data also fits the constraints of the new model). For example, if we aim to learn a mixture of factor analysers instead, we simply have to replace the line

```
Model model = new FactorAnalysis(data.getAtributes());
```

by

```
Model model = new MixtureOfFactorAnalysers(data.getAtributes());
```

Note that the method for learning the model, namely *Model::updateMode(DataStream<DataInstance>)* will always be the same regardless of the particular type of static model.

The actual learnt model is an object of the class *BayesianNetwork* which is stored as a member variable of *Model*. Thus, for using the network, we simply have to extract with the method *Model::getModel()*. One of the actions we can perform with it is saving it into the local file system. For saving it in *.bn* format:

```
BayesianNetworkWriter::save(BayesianNetwork bn, String path)
```

Alternatively, and assuming that we have the hugin library available, we can also save it in *.net* format:

```
BayesianNetworkWriteToHuginr::save(BayesianNetwork bn, String path)
```

Learning from Flink

In previous section we showed how the AMIDST toolbox can be used for learning a static model from a non-distributed dataset. In addition, you can use the pre-defined models to process massive data sets in a distributed computer cluster using **Apache Flink**. In particular, the model can be learnt from a *distributed ARFF folder* or from a file accesible via a HDFS url. A scheme of the learning process is shown below.

A code example for learning from Flink is shown below. Note that it only differs from the one in previous section in lines 25 to 33. In these lines, the Flink session is configurated and the stream is loaded, which is managed with an object of the class *DataFlink* (instead of *DataStream*).

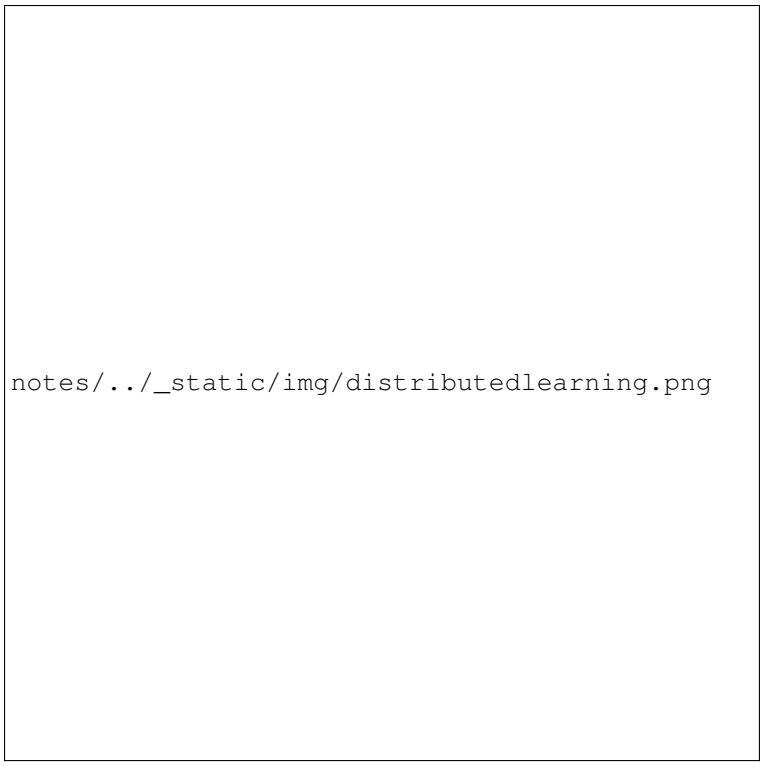


Fig. 7: Distributed learning process scheme

```
package eu.amidst.tutorial.usingAmidst.examples;

import COM.hugin.HAPI.ExceptionHugin;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.io.BayesianNetworkWriter;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.io.DataFlinkLoader;
import eu.amidst.latentvariablemodels.staticmodels.FactorAnalysis;
import eu.amidst.latentvariablemodels.staticmodels.Model;
import eu.amidst.tutorial.usingAmidst.Main;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

import java.io.IOException;

/**
 * Created by rcabanas on 23/05/16.
 */
public class StaticModelFlink {
    public static void main(String[] args) throws IOException, ExceptionHugin {


        //Set-up Flink session.
       // Configuration conf = new Configuration();
      //  conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
        final ExecutionEnvironment env = ExecutionEnvironment.
↪getExecutionEnvironment();
```

(continues on next page)

(continued from previous page)

```
        //env.getConfig().disableSysoutLogging();
       // env.setParallelism(Main.PARALLELISM);

        //Load the datastream
        String filename = "datasets/simulated/cajamarDistributed.arff";
        DataFlink<DataInstance> data = DataFlinkLoader.open(env, filename, false);

        //Learn the model
        Model model = new FactorAnalysis(data.getAttributes());
        model.updateModel(data);
        BayesianNetwork bn = model.getModel();

        System.out.println(bn);

        // Save with .bn format
        BayesianNetworkWriter.save(bn, "networks/simulated/exampleBN.bn");

        // Save with hugin format
        //BayesianNetworkWriterToHugin.save(bn, "networks/simulated/exampleBN.net");

    }

}
```

See on GitHub

In previous example, the distributed dataset is stored in our local file system. Instead, we might need to load from a distributed file system. For that, simply replace the string indicating the path. That is, replace

```
String filename = "datasets/simulated/cajamarDistributed.arff"
```

by

```
String filename = "hdfs://distributed-server/path-to-file"
```

Inference

Making probabilistic inference in BNs (a.k.a *belief updating*) consists of the computation of the posterior probability distribution for a set of variables of interest given some evidence over some other variables (see image below).

Fig. 8: Learning process scheme

The inference process is the same regardless of the way we have learnt our model: we simply have to obtain the BN learnt (stored as an object of the class *BayesianNetwork*), set the target variables and the evidence. As an example, let us consider the following code:

```java
package eu.amidst.tutorial.usingAmidst.examples;

import eu.amidst.core.distribution.Distribution;
import eu.amidst.core.inference.InferenceAlgorithm;
import eu.amidst.core.inference.messagepassing.VMP;
import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.HashMapAssignment;
import eu.amidst.core.variables.Variable;
import eu.amidst.core.variables.Variables;

import java.io.IOException;

/**
 * Created by rcabanas on 23/05/16.
 */
public class StaticModelInference {

    public static void main(String[] args) throws IOException, ClassNotFoundException
↪{

        BayesianNetwork bn  = BayesianNetworkLoader.loadFromFile("networks/simulated/
↪exampleBN.bn");
        Variables variables = bn.getVariables();

        //Variabeles of interest
        Variable varTarget = variables.getVariableByName("LatentVar1");
        Variable varObserved = null;

        //we set the evidence
        Assignment assignment = new HashMapAssignment(2);
        varObserved = variables.getVariableByName("Income");
        assignment.setValue(varObserved,0.0);

        //we set the algorithm
        InferenceAlgorithm infer = new VMP(); //new HuginInference(); new␣
↪ImportanceSampling();
        infer.setModel(bn);
        infer.setEvidence(assignment);

        //query
        infer.runInference();
        Distribution p = infer.getPosterior(varTarget);
        System.out.println("P(LatentVar1|Income=0.0) = "+p);

        //Or some more refined queries
        System.out.println("P(0.7<LatentVar1<6.59 |Income=0.0) = " + infer.
↪getExpectedValue(varTarget, v -> (0.7 < v && v < 6.59) ? 1.0 : 0.0 ));

    }

}
```

See on GitHub

Note that the learning algorithm can be easily changed by simply modifying line 35 where *VMP algorithm*. If we aim to use *Importance Sampling algorithm*, replace such line with:

```
InferenceAlgorithm infer = new ImportanceSampling();
```

Alternatively, we can use *Hugin Inference algorithm* (assuming that we have the corresponding libraries):

```
InferenceAlgorithm infer = new HuginInference();
```

Custom static model

It could happend that your model of interest is not predifined. In that case you can implement it yourself. For that purpose, create a new class inheriting the class *Model*. Then, add the code to the constructor with an object *Attributes* as input parameter, and the code of the method *void buildDAG()*. This last method is called before learning process and creates the object representing the DAG. As an example, the code below shows how to create a custom Gaussian Mixture.

```
package eu.amidst.tutorial.usingAmidst.practice;

import eu.amidst.core.datastream.Attributes;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.variables.Variable;
import eu.amidst.core.variables.Variables;
import eu.amidst.latentvariablemodels.staticmodels.FactorAnalysis;
import eu.amidst.latentvariablemodels.staticmodels.Model;
import eu.amidst.latentvariablemodels.staticmodels.exceptions.
↪WrongConfigurationException;

/**
 * Created by rcabanas on 23/05/16.
 */

public class CustomGaussianMixture extends Model{

    Attributes attributes;

    public CustomGaussianMixture(Attributes attributes) throws␣
↪WrongConfigurationException {
        super(attributes);
        this.attributes=attributes;
    }

    @Override
    protected void buildDAG() {

        /** Create a set of variables from the given attributes**/
        Variables variables = new Variables(attributes);

        /** Create a hidden variable with two hidden states*/
        Variable hiddenVar = variables.newMultinomialVariable("HiddenVar",2);

        //We create a standard naive Bayes
        DAG dag = new DAG(variables);
```

(continues on next page)

(continued from previous page)

```
        for (Variable variable: variables){
            if (variable==hiddenVar)
                continue;

            dag.getParentSet(variable).addParent(hiddenVar);
        }

        //This is needed to maintain coherence in the Model class.
        this.dag=dag;
        this.vars = variables;

    }

    //Method for testing the custom model
    public static void main(String[] args) {
        String filename = "datasets/simulated/cajamar.arff";
        DataStream<DataInstance> data = DataStreamLoader.open(filename);

        //Learn the model
        Model model = new CustomGaussianMixture(data.getAttributes());

        model.updateModel(data);
        BayesianNetwork bn = model.getModel();

        System.out.println(bn);
    }


}
```

See on GitHub

1.4.3 Dynamic Models

Learning and saving to disk

When dealing with temporal data, it might be advisable to learn a dynamic model. The module *latent-variable-models* in AMDIST also suports such kinds of models. For that the classes inheriting *DynamicModel* will be used. A synthesis of the learning process is shown below.

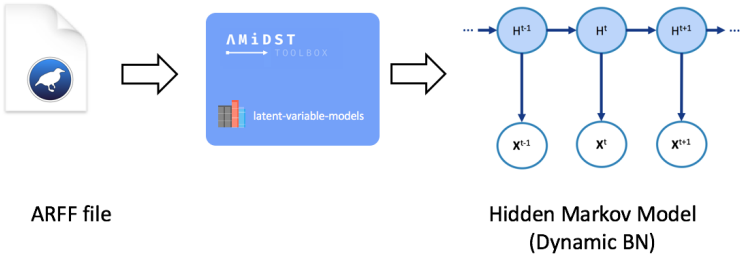


Fig. 9: Learning process scheme of a dynamic model

A code-example is given below. Here, a *Hidden Markov Model (HMM)* is learnt from a financial dataset with temporal information. Note the difference within the static learning is the way the dataset is loaded: now, it is

handled with an object of the class *DataStream<DynamicDataInstance>*. Finally, the DBN learnt is saved to disk with the method *DynamicBayesianNetworkWriter::save(String pathFile)*.

```
package eu.amidst.tutorial.usingAmidst.examples;

import COM.hugin.HAPI.ExceptionHugin;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.io.DynamicBayesianNetworkWriter;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.latentvariablemodels.dynamicmodels.DynamicModel;
import eu.amidst.latentvariablemodels.dynamicmodels.HiddenMarkovModel;

import java.io.IOException;

/**
 * Created by rcabanas on 23/05/16.
 */
public class DynamicModelLearning {

    public static void main(String[] args) throws IOException, ExceptionHugin {

        //Load the datastream
        String filename = "datasets/simulated/cajamar.arff";
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.
↪loadFromFile(filename);


        //Learn the model
        DynamicModel model = new HiddenMarkovModel(data.getAttributes());
        model.updateModel(data);
        DynamicBayesianNetwork dbn = model.getModel();


        System.out.println(dbn);


        // Save with .bn format
        DynamicBayesianNetworkWriter.save(dbn, "networks/simulated/exampleDBN.dbn");

        // Save with hugin format
        //DynamicBayesianNetworkWriterToHugin.save(dbn, "networks/simulated/
↪exampleDBN.net");

    }


}
```

See on GitHub

Inference

In the following code-example, the inference process of dynamic models is illustrated. First, a DBN is loaded from disk (line 24). Then, a dynamic dataset is loaded for testing our model (lines 29 to 30). Then the inference algorithm and target variables are set. In the final loop, the inference is perform for each data instance.

```
package eu.amidst.tutorial.usingAmidst.examples;


import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.distribution.UnivariateDistribution;
import eu.amidst.core.inference.ImportanceSampling;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.inference.FactoredFrontierForDBN;
import eu.amidst.dynamic.inference.InferenceAlgorithmForDBN;
import eu.amidst.dynamic.io.DynamicBayesianNetworkLoader;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;

import java.io.IOException;

/**
 * Created by rcabanas on 23/05/16.
 */
public class DynamicModelInference {

    public static void main(String[] args) throws IOException, ClassNotFoundException
↪{

        DynamicBayesianNetwork dbn = DynamicBayesianNetworkLoader.loadFromFile(
↪"networks/simulated/exampleDBN.dbn");

        System.out.println(dbn);

        //Testing dataset
        String filenamePredict = "datasets/simulated/cajamar.arff";
        DataStream<DynamicDataInstance> dataPredict = DynamicDataStreamLoader.
↪open(filenamePredict);

        //Select the inference algorithm
        InferenceAlgorithmForDBN infer = new FactoredFrontierForDBN(new␣
↪ImportanceSampling()); // new ImportanceSampling(),  new VMP(),
        infer.setModel(dbn);


        Variable varTarget = dbn.getDynamicVariables().getVariableByName(
↪"discreteHiddenVar");
        UnivariateDistribution posterior = null;

        //Classify each instance
        int t = 0;
        for (DynamicDataInstance instance : dataPredict) {
            if (instance.getSequenceID()>0)
                break;

            infer.addDynamicEvidence(instance);
            infer.runInference();

            posterior = infer.getFilteredPosterior(varTarget);
            System.out.println("t="+t+", P(discreteHiddenVar | Evidence)  = " +␣
↪posterior);
```

(continues on next page)

(continued from previous page)

```
            posterior = infer.getPredictivePosterior(varTarget, 2);
            //Display the output
            System.out.println("t="+t+"+5, P(discreteHiddenVar | Evidence)  = " +
↪posterior);


            t++;
        }




    }


}
```

See on GitHub

Note that the inference algorithm can be easily change if line 33 is modified by replacing it with:

```
InferenceAlgorithmForDBN = new FactoredFrontierForDBN(new VMP());
```

Custom dynamic model

Like for static models, we might be interested in creating our own dynamic models. In this case, you will have to create a class inheriting *DynamicModel*. Here below an example of a custom *Kalman Filter* is given.

```
package eu.amidst.tutorial.usingAmidst.practice;

import COM.hugin.HAPI.ExceptionHugin;
import eu.amidst.core.datastream.Attributes;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.io.DynamicBayesianNetworkWriter;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.models.DynamicDAG;
import eu.amidst.dynamic.variables.DynamicVariables;
import eu.amidst.latentvariablemodels.dynamicmodels.DynamicModel;
import eu.amidst.latentvariablemodels.dynamicmodels.HiddenMarkovModel;
import eu.amidst.latentvariablemodels.dynamicmodels.KalmanFilter;
import eu.amidst.latentvariablemodels.staticmodels.FactorAnalysis;
import eu.amidst.latentvariablemodels.staticmodels.Model;
import eu.amidst.latentvariablemodels.staticmodels.exceptions.
↪WrongConfigurationException;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
```

(continues on next page)

(continued from previous page)

```
/**
 * Created by rcabanas on 23/05/16.
 */

public class CustomKalmanFilter extends DynamicModel {

    Attributes attributes;

    public CustomKalmanFilter(Attributes attributes) throws␣
↪WrongConfigurationException {
        super(attributes);
        this.attributes=attributes;
    }


    @Override
    protected void buildDAG() {

        /*number of continuous hidden variable*/
        int numHiddenVars=3;

        /** Create a set of dynamic variables from the given attributes**/
        variables = new DynamicVariables(attributes);

        /* List of continuous hidden vars*/
        List<Variable> gaussianHiddenVars = new ArrayList<>();

        for(int i=0; i<numHiddenVars; i++) {
            Variable Hi = variables.newGaussianDynamicVariable("gaussianHiddenVar" +␣
↪i);
            gaussianHiddenVars.add(Hi);
        }

        DynamicDAG dynamicDAG = new DynamicDAG(this.variables);


        for (Variable h : gaussianHiddenVars) {
            dynamicDAG.getParentSetTimeT(h).addParent(h.getInterfaceVariable());
        }

        for (Variable variable: variables) {
            if (gaussianHiddenVars.contains(variable))
                continue;

            for (Variable h : gaussianHiddenVars) {
                dynamicDAG.getParentSetTimeT(variable).addParent(h);
            }
        }

        //This is needed to maintain coherence in the DynamicModel class.
        this.variables = variables;
        this.dynamicDAG = dynamicDAG;
    }


    public static void main(String[] args) throws IOException, ExceptionHugin {
```

(continues on next page)

(continued from previous page)

```
        //Load the datastream
        String filename = "datasets/simulated/cajamar.arff";
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.
↪loadFromFile(filename);


        //Learn the model
        DynamicModel model = new CustomKalmanFilter(data.getAttributes());
        model.updateModel(data);
        DynamicBayesianNetwork dbn = model.getModel();


        System.out.println(dbn);

    }


}
```

See on GitHub

1.5 Flinklink: Code Examples

1.5.1 Input/output

Reading data

In this example we show how can we read a dataset using Flink. Note that the process is the same regardless being a single or a distributed file.

```
package eu.amidst.flinklink.examples.io;

import eu.amidst.Main;
```

(continues on next page)

(continued from previous page)

```
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.io.DataFlinkLoader;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by rcabanas on 10/06/16.
 */
public class DataStreamLoaderExample {
    public static void main(String[] args) throws Exception {

        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;

        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
            env.getConfig().disableSysoutLogging();
        }
        //Paths to datasets
        String simpleFile = "datasets/simulated/syntheticData.arff";
        String distriFile = "datasets/simulated/distributed.arff";

        //Load the data
        DataFlink<DataInstance> dataSimple = DataFlinkLoader.open(env, simpleFile,␣
↪false);
        DataFlink<DataInstance> dataDistri = DataFlinkLoader.open(env,distriFile,␣
↪false);

        //Print the number of data samples
        System.out.println(dataSimple.getDataSet().count());
        System.out.println(dataDistri.getDataSet().count());

    }
}
```

[Back to Top]

Writing data

Below we generate a random Flink dataset with 1000 instances, 2 discrete variables and 3 continuous ones. The seed used is 1234. Eventually, we save it as a distributed dataset (format ARFF folder).

```java
package eu.amidst.flinklink.examples.io;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.io.DataFlinkWriter;
import eu.amidst.flinklink.core.utils.DataSetGenerator;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by rcabanas on 09/06/16.
 */
public class DataStreamWriterExample {
    public static void main(String[] args) throws Exception {

        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;

        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
            env.getConfig().disableSysoutLogging();
        }

        //generate a random dataset
        DataFlink<DataInstance> dataFlink = new DataSetGenerator().generate(env,1234,
↪1000,2,3);

        //Saves it as a distributed arff file
        DataFlinkWriter.writeDataToARFFFolder(dataFlink, "datasets/simulated/
↪distributed.arff");
    }
}


//TODO: Write to standard arff --> convert to datastream??
```

[Back to Top]

1.5.2 Parametric learning

Here give examples of the provided algorithms by AMiDST for learning the probability distributions from a Flink data set. For shake of simplicity, we will consider the Naive Bayes DAG structure. Note that the code is almost the

same of each of the algoritms, they only differ on the constructor used (e.g. *new ParallelMaximumLikelihood()*, *new dVMP()*, etc.)

Parallel Maximum Likelihood

```
package eu.amidst.flinklink.examples.learning;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.utils.DAGGenerator;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.learning.parametric.ParallelMaximumLikelihood;
import eu.amidst.flinklink.core.learning.parametric.ParameterLearningAlgorithm;
import eu.amidst.flinklink.core.utils.DataSetGenerator;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by rcabanas on 14/06/16.
 */
public class ParallelMLExample {
    public static void main(String[] args) throws Exception {

        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;

        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
            env.getConfig().disableSysoutLogging();
        }


        //generate a random dataset
        DataFlink<DataInstance> dataFlink = new DataSetGenerator().generate(env,1234,
↪1000,5,0);

        //Creates a DAG with the NaiveBayes structure for the random dataset
        DAG dag = DAGGenerator.getNaiveBayesStructure(dataFlink.getAttributes(),
↪"DiscreteVar4");
        System.out.println(dag.toString());
```

(continues on next page)

(continued from previous page)

```
        //Create the  Learner object
        ParameterLearningAlgorithm learningAlgorithmFlink =
                new ParallelMaximumLikelihood();

        //Learning parameters
        learningAlgorithmFlink.setBatchSize(10);
        learningAlgorithmFlink.setDAG(dag);

        //Initialize the learning process
        learningAlgorithmFlink.initLearning();

        //Learn from the flink data
        learningAlgorithmFlink.updateModel(dataFlink);

        //Print the learnt BN
        BayesianNetwork bn = learningAlgorithmFlink.getLearntBayesianNetwork();
        System.out.println(bn);



    }
}
```

[Back to Top]

Distributed Variational Message Passing

```
package eu.amidst.flinklink.examples.learning;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.utils.DAGGenerator;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.learning.parametric.ParameterLearningAlgorithm;
import eu.amidst.flinklink.core.learning.parametric.dVMP;
import eu.amidst.flinklink.core.utils.DataSetGenerator;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by rcabanas on 14/06/16.
 */
public class dVMPExample {
    public static void main(String[] args) throws Exception {

        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;
```

(continues on next page)

(continued from previous page)

```
        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
            env.getConfig().disableSysoutLogging();
        }
        //generate a random dataset
        DataFlink<DataInstance> dataFlink = new DataSetGenerator().generate(env,1234,
↪1000,5,0);

        //Creates a DAG with the NaiveBayes structure for the random dataset
        DAG dag = DAGGenerator.getNaiveBayesStructure(dataFlink.getAttributes(),
↪"DiscreteVar4");
        System.out.println(dag.toString());


        //Create the  Learner object
        ParameterLearningAlgorithm learningAlgorithmFlink =
                new dVMP();

        //Learning parameters
        learningAlgorithmFlink.setBatchSize(10);
        learningAlgorithmFlink.setDAG(dag);

        //Initialize the learning process
        learningAlgorithmFlink.initLearning();

        //Learn from the flink data
        learningAlgorithmFlink.updateModel(dataFlink);

        //Print the learnt BN
        BayesianNetwork bn = learningAlgorithmFlink.getLearntBayesianNetwork();
        System.out.println(bn);



    }
}
```

[Back to Top]

Distributed VI

```
package eu.amidst.flinklink.examples.learning;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.models.BayesianNetwork;
```

(continues on next page)

(continued from previous page)

```
import eu.amidst.core.models.DAG;
import eu.amidst.core.utils.DAGGenerator;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.learning.parametric.DistributedVI;
import eu.amidst.flinklink.core.learning.parametric.ParameterLearningAlgorithm;
import eu.amidst.flinklink.core.utils.DataSetGenerator;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by rcabanas on 14/06/16.
 */
public class DistributedVIExample {
    public static void main(String[] args) throws Exception {
        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;

        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
            env.getConfig().disableSysoutLogging();
        }
        //generate a random dataset
        DataFlink<DataInstance> dataFlink = new DataSetGenerator().generate(env,1234,
↪1000,5,0);

        //Creates a DAG with the NaiveBayes structure for the random dataset
        DAG dag = DAGGenerator.getNaiveBayesStructure(dataFlink.getAttributes(),
↪"DiscreteVar4");
        System.out.println(dag.toString());


        //Create the  Learner object
        ParameterLearningAlgorithm learningAlgorithmFlink =
                new DistributedVI();

        //Learning parameters
        learningAlgorithmFlink.setBatchSize(10);
        learningAlgorithmFlink.setDAG(dag);

        //Initialize the learning process
        learningAlgorithmFlink.initLearning();

        //Learn from the flink data
        learningAlgorithmFlink.updateModel(dataFlink);
```

(continues on next page)

(continued from previous page)

```
        //Print the learnt BN
        BayesianNetwork bn = learningAlgorithmFlink.getLearntBayesianNetwork();
        System.out.println(bn);



    }
}
```

[Back to Top]

Stochastic VI

An example of the learning algorithm Stochastic VI is given below. Note that two specific parameters must be set, namely the *learning factor* and the *data size*.

```
package eu.amidst.flinklink.examples.learning;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.utils.DAGGenerator;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.learning.parametric.ParameterLearningAlgorithm;
import eu.amidst.flinklink.core.learning.parametric.StochasticVI;
import eu.amidst.flinklink.core.utils.DataSetGenerator;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by rcabanas on 14/06/16.
 */
public class StochasticVIExample {
    public static void main(String[] args) throws Exception {

        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;

        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
```

(continues on next page)

(continued from previous page)

```
            env.getConfig().disableSysoutLogging();
        }
        //generate a random dataset
        DataFlink<DataInstance> dataFlink = new DataSetGenerator().generate(env,1234,
↪1000,5,0);

        //Creates a DAG with the NaiveBayes structure for the random dataset
        DAG dag = DAGGenerator.getNaiveBayesStructure(dataFlink.getAttributes(),
↪"DiscreteVar4");
        System.out.println(dag.toString());


        //Create the  Learner object
        ParameterLearningAlgorithm learningAlgorithmFlink =
                new StochasticVI();

        //Learning parameters
        learningAlgorithmFlink.setBatchSize(10);
        learningAlgorithmFlink.setDAG(dag);

        //Initialize the learning process
        learningAlgorithmFlink.initLearning();

        //Learn from the flink data
        learningAlgorithmFlink.updateModel(dataFlink);

        //Specific parameters for the algorithm
        ((StochasticVI)learningAlgorithmFlink).setLearningFactor(0.7);
        ((StochasticVI)learningAlgorithmFlink).setDataSetSize((int) dataFlink.
↪getDataSet().count());


        //Print the learnt BN
        BayesianNetwork bn = learningAlgorithmFlink.getLearntBayesianNetwork();
        System.out.println(bn);



    }
}
```

[Back to Top]

1.5.3 Extensions and applications

Latent variable models with Flink

The module *latent-variable-models* contains a large set of classes that allow to easily learn some of the standard models with latent variables. These models can be learnt from not only from local datasets (e.g. a single ARFF file) but also from distributed ones (e.g. ARFF folder). These last ones are managed using Flink. In code example shown below the model *Factor Analysis* is learnt from a distributed dataset.

```
package eu.amidst.flinklink.examples.extensions;

import eu.amidst.Main;
```

(continues on next page)

(continued from previous page)

```
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.io.DataFlinkLoader;
import eu.amidst.latentvariablemodels.staticmodels.FactorAnalysis;
import eu.amidst.latentvariablemodels.staticmodels.Model;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

import java.io.FileNotFoundException;

/**
 * Created by rcabanas on 14/06/16.
 */
public class LatentModelsFlink {
    public static void main(String[] args) throws FileNotFoundException {
        boolean hadoop_cluster = false;

        if (args.length>1){
            hadoop_cluster = Boolean.parseBoolean(args[0]);
        }

        final ExecutionEnvironment env;

        //Set-up Flink session.
        if(hadoop_cluster){
            env = ExecutionEnvironment.getExecutionEnvironment();
            env.getConfig().disableSysoutLogging();
        }else{
            Configuration conf = new Configuration();
            conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
            conf.setInteger("taskmanager.numberOfTaskSlots",Main.PARALLELISM);
            env = ExecutionEnvironment.createLocalEnvironment(conf);
            env.setParallelism(Main.PARALLELISM);
            env.getConfig().disableSysoutLogging();
        }
        //Load the datastream
        String filename = "datasets/simulated/exampleDS_d0_c5.arff";
        DataFlink<DataInstance> data = DataFlinkLoader.loadDataFromFile(env, filename,
↪ false);

        //Learn the model
        Model model = new FactorAnalysis(data.getAttributes());
        ((FactorAnalysis)model).setNumberOfLatentVariables(3);
        model.updateModel(data);
        BayesianNetwork bn = model.getModel();

        System.out.println(bn);

    }
}
```

[Back to Top]

Concept drift detection

A salient aspect of streaming data is that the domain being modeled is often *non-stationary*. That is, the distribution governing the data changes over time. This situation is known as *concept drift* and if not carefully taken into account, the result can be a failure to capture and interpret intrinsic properties of the data during data exploration. The AMIDST toolbox can be used for detecting this situation as shown in the example below.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.flinklink.examples.reviewMeeting2015;

import eu.amidst.Main;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.flinklink.core.conceptdrift.IDAConceptDriftDetector;
import eu.amidst.flinklink.core.data.DataFlink;
import eu.amidst.flinklink.core.io.DataFlinkLoader;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.configuration.Configuration;

/**
 * Created by ana@cs.aau.dk on 18/01/16.
 */
public class ConceptDriftDetector {

    //public int NSETS = 15;


    public static void learnIDAConceptDriftDetector(int NSETS) throws Exception {
        //Set-up Flink session.
        Configuration conf = new Configuration();
        conf.setInteger("taskmanager.network.numberOfBuffers", 12000);
        final ExecutionEnvironment env = ExecutionEnvironment.
↪createLocalEnvironment(conf);
        env.getConfig().disableSysoutLogging();
        env.setParallelism(Main.PARALLELISM);
```

(continues on next page)

(continued from previous page)

```
      //  DataFlink<DataInstance> data0 = DataFlinkLoader.loadDataFromFolder(env,
      //          "hdfs:///tmp_conceptdrift_data0.arff", false);

          DataFlink<DataInstance> data0 = DataFlinkLoader.open(env,
                  "./datasets/simulated/tmp_conceptdrift_data0.arff", false);

        long start = System.nanoTime();
        IDAConceptDriftDetector learn = new IDAConceptDriftDetector();
        learn.setBatchSize(1000);
        learn.setClassIndex(0);
        learn.setAttributes(data0.getAttributes());
        learn.setNumberOfGlobalVars(1);
        learn.setTransitionVariance(0.1);
        learn.setSeed(0);

        learn.initLearning();

        System.out.println(learn.getGlobalDAG().toString());

        double[] output = new double[NSETS];

        System.out.println("--------------- LEARNING DATA " + 0 + " ------------------
↪--------");
        double[] out = learn.updateModelWithNewTimeSlice(data0);
        //System.out.println(learn.getLearntDynamicBayesianNetwork());
        output[0] = out[0];

        for (int i = 1; i < NSETS; i++) {
            System.out.println("--------------- LEARNING DATA " + i + " --------------
↪-------------");
            DataFlink<DataInstance> dataNew = DataFlinkLoader.open(env,
                    "./datasets/simulated/tmp_conceptdrift_data"+i+".arff", false);
            out = learn.updateModelWithNewTimeSlice(dataNew);
            //System.out.println(learn.getLearntDynamicBayesianNetwork());
            output[i] = out[0];

        }
        long duration = (System.nanoTime() - start) / 1;
        double seconds = duration / 1000000000.0;

        System.out.println("Running time" + seconds + " seconds");

        //System.out.println(learn.getLearntDynamicBayesianNetwork());

        for (int i = 0; i < NSETS; i++) {
            System.out.println("E(H_"+i+") =\t" + output[i]);
        }

    }

    public static void main(String[] args) throws Exception {

        int NSETS = Integer.parseInt(args[0]);

        learnIDAConceptDriftDetector(NSETS);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

[Back to Top]

1.6 Dynamic Bayesian Networks: Code Examples

1.6.1 Data Streams

In this example we show how to use the main features of a *DataStream* object. More precisely, we show how to load a dynamic data stream and how to iterate over the *DynamicDataInstance* objects.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
```

(continues on next page)

(continued from previous page)

```
 *
 *
 */

package eu.amidst.dynamic.examples.datastream;

import eu.amidst.core.datastream.Attribute;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.utils.DataSetGenerator;

/**
 * An example showing how to load an use a DataStream object. For more options refer␣
↪to class
 * eu.amidst.core.examples.datastream and simply change DataInstance by␣
↪DynamicDataInstance
 *
 * Created by ana@cs.aau.dk on 02/12/15.
 */
public class DataStreamsExample {
    public static void main(String[] args) throws Exception {

        //Open the data stream using the class DynamicDataStreamLoader
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.loadFromFile(
↪"datasets/simulated/exampleDS_d2_c3.arff");


        //Access the attributes defining the data stream
        System.out.println("Attributes defining the data set");
        for (Attribute attribute : data.getAttributes()) {
            System.out.println(attribute.getName());
        }
        Attribute discreteVar0 = data.getAttributes().getAttributeByName("DiscreteVar0
↪");

        //Iterate over dynamic data instances
        System.out.println("1. Iterating over samples using a for loop");
        for (DynamicDataInstance dataInstance : data) {
            System.out.println("SequenceID = "+dataInstance.getSequenceID()+", TimeID␣
↪= "+dataInstance.getTimeID());
            System.out.println("The value of attribute discreteVar0 for the current␣
↪data instance is: " +
                    dataInstance.getValue(discreteVar0));
        }

    }
}
```

1.6.2 Dynamic Random Variables

This example show the basic functionalities related to dynamic variables.

```
/*
 *
```

(continues on next page)

(continued from previous page)

```
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.variables;

import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.variables.DynamicVariables;

import java.util.Arrays;

/**
 * This example show the basic functionalities related to dynamic variables.
 */
public class DynamicVariablesExample {

    public static void main(String[] args) throws Exception {

        //Create an empty DynamicVariables object
        DynamicVariables variables = new DynamicVariables();

        //Invoke the "new" methods of the object DynamicVariables to create new␣
↪dynamic variables.

        //Create a Gaussian dynamic variables
        Variable gaussianVar = variables.newGaussianDynamicVariable("GaussianVar");

        //Create a Multinomial dynamic variable with two states
        Variable multinomialVar = variables.newMultinomialDynamicVariable(
↪"MultinomialVar", 2);

        //Create a Multinomial dynamic variable with two states: TRUE and FALSE
        Variable multinomialVar2 = variables.newMultinomialDynamicVariable(
↪"MultinomialVar2", Arrays.asList("TRUE, FALSE"));

        //We must block the object before we start to query.
        variables.block();

        //All dynamic Variables have an interface variable
```

(continues on next page)

(continued from previous page)

```
        Variable gaussianVarInt = gaussianVar.getInterfaceVariable();
        Variable multinomialVarInt = multinomialVar.getInterfaceVariable();

        //Get the "main" Variable associated with each interface variable through the␣
↪DynamicVariable object
        Variable mainMultinomialVar = variables.
↪getVariableFromInterface(multinomialVarInt);

        //Check whether a variable is an interface variable
        System.out.println("Is Variable "+gaussianVar.getName()+" an interface␣
↪variable? "
                +gaussianVar.isInterfaceVariable());
        System.out.println("Is Variable "+gaussianVarInt.getName()+" an interface␣
↪variable? "
                +gaussianVarInt.isInterfaceVariable());

        //Check whether a variable is a dynamic variable
        System.out.println("Is Variable "+multinomialVar.getName()+" a dynamic␣
↪variable? "
                +gaussianVar.isDynamicVariable());
    }
}
```

1.6.3 Dynamic Bayesian networks

Creating Dynamic Bayesian networks

This example creates a dynamic BN, from a dynamic data stream, with randomly generated probability distributions, then saves it to a file.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */
```

(continues on next page)

(continued from previous page)

```
package eu.amidst.dynamic.examples.models;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.io.DynamicBayesianNetworkWriter;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.models.DynamicDAG;
import eu.amidst.dynamic.variables.DynamicVariables;

/**
 * This example creates a dynamic BN from a dynamic data stream, with randomly␣
↪generated probability distributions, then saves it to a file.
 */
public class CreatingDBNs {

    public static void main(String[] args) throws Exception{

        //Open the data stream using the static class DynamicDataStreamLoader
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.loadFromFile(
                "datasets/simulated/syntheticDataDiscrete.arff");

        /**
         * 1. Once the data is loaded, we create a random variable for each of the␣
↪attributes (i.e. data columns)
         * in our data.
         *
         * 2. {@link DynamicVariables} is the class for doing that. It takes a list␣
↪of Attributes and internally creates
         * all the variables. We create the variables using DynamicVariables class to␣
↪guarantee that each variable
         * has a different ID number and make it transparent for the user. Each␣
↪random variable has an associated
         * interface variable.
         *
         * 3. We can extract the Variable objects by using the method␣
↪getVariableByName();
         */
        DynamicVariables dynamicVariables = new DynamicVariables(data.
↪getAttributes());
        DynamicDAG dynamicDAG = new DynamicDAG(dynamicVariables);

        Variable A = dynamicVariables.getVariableByName("A");
        Variable B = dynamicVariables.getVariableByName("B");
        Variable C = dynamicVariables.getVariableByName("C");
        Variable D = dynamicVariables.getVariableByName("D");
        Variable E = dynamicVariables.getVariableByName("E");
        Variable G = dynamicVariables.getVariableByName("G");

        Variable A_Interface = dynamicVariables.getInterfaceVariable(A);
        Variable B_Interface = dynamicVariables.getInterfaceVariable(B);

        //Note that C_Interface and D_Interface are also created although they will␣
↪not be used
        //(we will not add temporal dependencies)
```

(continues on next page)

(continued from previous page)

```
        Variable E_Interface = dynamicVariables.getInterfaceVariable(E);
        Variable G_Interface = dynamicVariables.getInterfaceVariable(G);

        // Example of the dynamic DAG structure
        // Time 0: Parents at time 0 are automatically created when adding parents at␣
↪time T
        dynamicDAG.getParentSetTimeT(B).addParent(A);
        dynamicDAG.getParentSetTimeT(C).addParent(A);
        dynamicDAG.getParentSetTimeT(D).addParent(A);
        dynamicDAG.getParentSetTimeT(E).addParent(A);
        dynamicDAG.getParentSetTimeT(G).addParent(A);
        dynamicDAG.getParentSetTimeT(A).addParent(A_Interface);
        dynamicDAG.getParentSetTimeT(B).addParent(B_Interface);
        dynamicDAG.getParentSetTimeT(E).addParent(E_Interface);
        dynamicDAG.getParentSetTimeT(G).addParent(G_Interface);

        System.out.println(dynamicDAG.toString());

        /**
         * 1. We now create the Dynamic Bayesian network from the previous Dynamic␣
↪DAG.
         *
         * 2. The DBN object is created from the DynamicDAG. It automatically looks␣
↪at the distribution type
         * of each variable and their parents to initialize the Distributions objects␣
↪that are stored
         * inside (i.e. Multinomial, Normal, CLG, etc). The parameters defining these␣
↪distributions are
         * properly initialized.
         *
         * 3. The network is printed and we can have a look at the kind of␣
↪distributions stored in the DBN object.
         */
        DynamicBayesianNetwork dbn = new DynamicBayesianNetwork(dynamicDAG);
        System.out.printf(dbn.toString());

        /**
         * Finally teh Bayesian network is saved to a file.
         */
        DynamicBayesianNetworkWriter.save(dbn, "networks/simulated/DBNExample.dbn");
    }

}
```

Creating Dynamic Bayesian Networks with Latent Variables

This example shows how to create a BN model with hidden variables. We simply create a BN for clustering, i.e., a naive Bayes like structure with a single hidden variable acting as parant of all the remaining observable variables.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
```

(continues on next page)

(continued from previous page)

```
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.models;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.io.DynamicBayesianNetworkWriter;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.models.DynamicDAG;
import eu.amidst.dynamic.variables.DynamicVariables;

/**
 *
 * In this example, we show how to create a DBN model with latent variables. We␣
↪create a DBN a class and a hidden
 * variable acting as parent of all the observable variables.
 *
 * Created by ana@cs.aau.dk on 02/12/15.
 */
public class CreatingDBNsWithLatentVariables {

    public static void main(String[] args) throws Exception{

        //We can open the data stream using the static class DynamicDataStreamLoader
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.loadFromFile(
                "datasets/simulated/syntheticDataDiscrete.arff");

        /**
         * 1. Once the data is loaded, we create a random variable for each of the␣
↪attributes (i.e. data columns)
         * in our data.
         *
         * 2. {@link DynamicVariables} is the class for doing that. It takes a list␣
↪of Attributes and internally creates
         * all the variables. We create the variables using DynamicVariables class to␣
↪guarantee that each variable
         * has a different ID number and make it transparent for the user. Each␣
↪random variable has an associated
         * interface variable.
         *
```

(continues on next page)

(continued from previous page)

```
         * 3. We can extract the Variable objects by using the method␣
↪getVariableByName();
         */
        DynamicVariables dynamicVariables = new DynamicVariables(data.
↪getAttributes());

        Variable A = dynamicVariables.getVariableByName("A");
        Variable B = dynamicVariables.getVariableByName("B");
        Variable C = dynamicVariables.getVariableByName("C");
        Variable D = dynamicVariables.getVariableByName("D");
        Variable E = dynamicVariables.getVariableByName("E");
        Variable G = dynamicVariables.getVariableByName("G");

        Variable A_Interface = dynamicVariables.getInterfaceVariable(A);
        Variable B_Interface = dynamicVariables.getInterfaceVariable(B);

        //Note that C_Interface and D_Interface are also created although they will␣
↪not be used
        //(we will not add temporal dependencies)

        Variable E_Interface = dynamicVariables.getInterfaceVariable(E);
        Variable G_Interface = dynamicVariables.getInterfaceVariable(G);

        /*
         * We add a hidden multinomial variable (with 2 states) as parent of all␣
↪variables except A
         */

        Variable H = dynamicVariables.newMultinomialDynamicVariable("H", 2);
        Variable H_Interface = dynamicVariables.getInterfaceVariable(H);

        /**
         * 1. Once we have defined your {@link DynamicVariables} object, including␣
↪the latent variable,
         * the next step is to create a DynamicDAG structure over this set of␣
↪variables.
         *
         */

        DynamicDAG dynamicDAG = new DynamicDAG(dynamicVariables);

        // EXAMPLE OF THE DAG STRUCTURE

        /*

        DAG Time 0
        H : {  }
        A : {  }
        B : { A, H }
        C : { A, H }
        D : { A, H }
        E : { A, H }
        G : { A, H }

        DAG Time T
        H : { H_Interface }
        A : { A_Interface }
```

(continues on next page)

(continued from previous page)

```
        B : { H, A, B_Interface }
        C : { H, A }
        D : { H, A }
        E : { H, A, E_Interface }
        G : { H, A, G_Interface }

        */

        /*
         * 1. To add parents to each variable, we first recover the ParentSet object by the method
         * getParentSet(Variable var) and then call the method addParent(Variable var).
         *
         * 2. We just put the hidden variable as parent of all the other variables (except A), and
         * link it temporally.
         *
         */

        // Time 0: Parents at time 0 are automatically created when adding parents at time T
        // Time t
        dynamicDAG.getParentSetsTimeT().stream()
                .filter(pset -> pset.getMainVar().getVarID() != A.getVarID())
                .filter(pset -> pset.getMainVar().getVarID() != H.getVarID())
                .forEach(pset -> {
                    pset.addParent(A);
                    pset.addParent(H);
                });
        dynamicDAG.getParentSetTimeT(A).addParent(A_Interface);
        dynamicDAG.getParentSetTimeT(B).addParent(B_Interface);
        dynamicDAG.getParentSetTimeT(E).addParent(E_Interface);
        dynamicDAG.getParentSetTimeT(G).addParent(G_Interface);
        dynamicDAG.getParentSetTimeT(H).addParent(H_Interface);

        System.out.println(dynamicDAG.toString());

        /**
         * 1. We now create the Dynamic Bayesian network from the previous Dynamic DAG.
         *
         * 2. The DBN object is created from the DynamicDAG. It automatically looks at the distribution type
         * of each variable and their parents to initialize the Distributions objects that are stored
         * inside (i.e. Multinomial, Normal, CLG, etc). The parameters defining these distributions are
         * properly initialized.
         *
         * 3. The network is printed and we can have a look at the kind of distributions stored in the DBN object.
         */
        DynamicBayesianNetwork dbn = new DynamicBayesianNetwork(dynamicDAG);
        System.out.printf(dbn.toString());

        /**
```

(continues on next page)

(continued from previous page)

```
         * Finally teh Bayesian network is saved to a file.
         */
        DynamicBayesianNetworkWriter.save(dbn, "networks/simulated/DBNExample.dbn");
    }
}
```

Modifying Dynamic Bayesian Networks

This example shows how to create a BN model with hidden variables. We This example shows how to access and modify the conditional probabilities of a Dynamic Bayesian network model.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.models;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.distribution.Multinomial;
import eu.amidst.core.distribution.Multinomial_MultinomialParents;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.io.DynamicDataStreamLoader;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.models.DynamicDAG;
import eu.amidst.dynamic.variables.DynamicVariables;

/**
 * In this example we show how to access and modify the conditional probabilities of␣
↪a Dynamic Bayesian network model.
 *
 * Created by ana@cs.aau.dk on 02/12/15.
 */
public class ModifyingDBNs {
    public static void main(String[] args) throws Exception {
```

(continues on next page)

(continued from previous page)

```
        //We can open the data stream using the static class DynamicDataStreamLoader
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.loadFromFile(
                "datasets/simulated/syntheticDataDiscrete.arff");

        /**
         * 1. Once the data is loaded, we create a random variable for each of the␣
↪attributes (i.e. data columns)
         * in our data.
         *
         * 2. {@link DynamicVariables} is the class for doing that. It takes a list␣
↪of Attributes and internally creates
         * all the variables. We create the variables using DynamicVariables class to␣
↪guarantee that each variable
         * has a different ID number and make it transparent for the user. Each␣
↪random variable has an associated
         * interface variable.
         *
         * 3. We can extract the Variable objects by using the method␣
↪getVariableByName();
         */
        DynamicVariables dynamicVariables = new DynamicVariables(data.
↪getAttributes());
        DynamicDAG dynamicDAG = new DynamicDAG(dynamicVariables);

        Variable A = dynamicVariables.getVariableByName("A");
        Variable B = dynamicVariables.getVariableByName("B");
        Variable C = dynamicVariables.getVariableByName("C");
        Variable D = dynamicVariables.getVariableByName("D");
        Variable E = dynamicVariables.getVariableByName("E");
        Variable G = dynamicVariables.getVariableByName("G");

        Variable A_Interface = dynamicVariables.getInterfaceVariable(A);
        Variable B_Interface = dynamicVariables.getInterfaceVariable(B);

        //Note that C_Interface and D_Interface are also created although they will␣
↪not be used
        //(we will not add temporal dependencies)

        Variable E_Interface = dynamicVariables.getInterfaceVariable(E);
        Variable G_Interface = dynamicVariables.getInterfaceVariable(G);

        // EXAMPLE OF THE DAG STRUCTURE

        /*

        DAG Time 0
        A : {  }
        B : { A }
        C : { A }
        D : { A }
        E : { A }
        G : { A }

        DAG Time T
        A : { A_Interface }
        B : { A, B_Interface }
```

(continues on next page)

(continued from previous page)

```
        C : { A }
        D : { A }
        E : { A, E_Interface }
        G : { A, G_Interface }

        */

        // Time 0: Parents at time 0 are automatically created when adding parents at␣
↪time T
        // Time t
        dynamicDAG.getParentSetTimeT(B).addParent(A);
        dynamicDAG.getParentSetTimeT(C).addParent(A);
        dynamicDAG.getParentSetTimeT(D).addParent(A);
        dynamicDAG.getParentSetTimeT(E).addParent(A);
        dynamicDAG.getParentSetTimeT(G).addParent(A);
        dynamicDAG.getParentSetTimeT(A).addParent(A_Interface);
        dynamicDAG.getParentSetTimeT(B).addParent(B_Interface);
        dynamicDAG.getParentSetTimeT(E).addParent(E_Interface);
        dynamicDAG.getParentSetTimeT(G).addParent(G_Interface);

        /**
         * 1. We now create the Dynamic Bayesian network from the previous Dynamic␣
↪DAG.
         *
         * 2. The DBN object is created from the DynamicDAG. It automatically looks␣
↪at the distribution type
         * of each variable and their parents to initialize the Distributions objects␣
↪that are stored
         * inside (i.e. Multinomial, Normal, CLG, etc). The parameters defining these␣
↪distributions are
         * properly initialized.
         *
         * 3. The network is printed and we can have a look at the kind of␣
↪distributions stored in the DBN object.
         */
        DynamicBayesianNetwork dbn = new DynamicBayesianNetwork(dynamicDAG);
        System.out.printf(dbn.toString());

        /*
         * We now modify the conditional probability distributions to assigned the␣
↪values we want
         */

        /* IMPORTANT: The parents are indexed according to Koller (Chapter 10. Pag.␣
↪358). Example:
           Parents: A = {A0,A1} and B = {B0,B1,B2}.
           NumberOfPossibleAssignments = 6

           Index   A    B
             0     A0   B0
             1     A1   B1
             2     A0   B2
             3     A1   B0
             4     A0   B1
             5     A1   B2
        */
```

(continues on next page)

(continued from previous page)

```
        //␣
↪*****************************************************************************************
        // ********************************** TIME 0␣
↪*****************************************
        //␣
↪*****************************************************************************************

        // Variable A
        Multinomial distA_Time0 = dbn.getConditionalDistributionTime0(A);
        distA_Time0.setProbabilities(new double[]{0.3, 0.7});

        // Variable B
        Multinomial_MultinomialParents distB_Time0 = dbn.
↪getConditionalDistributionTime0(B);
        distB_Time0.getMultinomial(0).setProbabilities(new double[]{0.4, 0.1, 0.5});
        distB_Time0.getMultinomial(1).setProbabilities(new double[]{0.2, 0.5, 0.3});

        // Variable C
        Multinomial_MultinomialParents distC_Time0 = dbn.
↪getConditionalDistributionTime0(C);
        distC_Time0.getMultinomial(0).setProbabilities(new double[]{0.4, 0.6});
        distC_Time0.getMultinomial(1).setProbabilities(new double[]{0.2, 0.8});

        // Variable D
        Multinomial_MultinomialParents distD_Time0 = dbn.
↪getConditionalDistributionTime0(D);
        distD_Time0.getMultinomial(0).setProbabilities(new double[]{0.7, 0.3});
        distD_Time0.getMultinomial(1).setProbabilities(new double[]{0.1, 0.9});

        // Variable E
        Multinomial_MultinomialParents distE_Time0 = dbn.
↪getConditionalDistributionTime0(E);
        distE_Time0.getMultinomial(0).setProbabilities(new double[]{0.8, 0.2});
        distE_Time0.getMultinomial(1).setProbabilities(new double[]{0.1, 0.9});

        // Variable G
        Multinomial_MultinomialParents distG_Time0 = dbn.
↪getConditionalDistributionTime0(G);
        distG_Time0.getMultinomial(0).setProbabilities(new double[]{0.6, 0.4});
        distG_Time0.getMultinomial(1).setProbabilities(new double[]{0.7, 0.3});

        //␣
↪*****************************************************************************************
        // ********************************** TIME T␣
↪*****************************************
        //␣
↪*****************************************************************************************

        // Variable A
        Multinomial_MultinomialParents distA_TimeT = dbn.
↪getConditionalDistributionTimeT(A);
        distA_TimeT.getMultinomial(0).setProbabilities(new double[]{0.15, 0.85});
        distA_TimeT.getMultinomial(1).setProbabilities(new double[]{0.1, 0.9});

        // Variable B
        Multinomial_MultinomialParents distB_TimeT = dbn.
↪getConditionalDistributionTimeT(B);
```

(continues on next page)

(continued from previous page)

```
        distB_TimeT.getMultinomial(0).setProbabilities(new double[]{0.1, 0.2, 0.7});
        distB_TimeT.getMultinomial(1).setProbabilities(new double[]{0.6, 0.1, 0.3});
        distB_TimeT.getMultinomial(2).setProbabilities(new double[]{0.3, 0.4, 0.3});
        distB_TimeT.getMultinomial(3).setProbabilities(new double[]{0.2, 0.1, 0.7});
        distB_TimeT.getMultinomial(4).setProbabilities(new double[]{0.5, 0.1, 0.4});
        distB_TimeT.getMultinomial(5).setProbabilities(new double[]{0.1, 0.1, 0.8});

        // Variable C: equals to the distribution at time 0 (C does not have temporal␣
↪clone)
        Multinomial_MultinomialParents distC_TimeT = dbn.
↪getConditionalDistributionTimeT(C);
        distC_TimeT.getMultinomial(0).setProbabilities(new double[]{0.4, 0.6});
        distC_TimeT.getMultinomial(1).setProbabilities(new double[]{0.2, 0.8});

        // Variable D: equals to the distribution at time 0 (D does not have temporal␣
↪clone)
        Multinomial_MultinomialParents distD_TimeT = dbn.
↪getConditionalDistributionTimeT(D);
        distD_TimeT.getMultinomial(0).setProbabilities(new double[]{0.7, 0.3});
        distD_TimeT.getMultinomial(1).setProbabilities(new double[]{0.1, 0.9});

        // Variable E
        Multinomial_MultinomialParents distE_TimeT = dbn.
↪getConditionalDistributionTimeT(E);
        distE_TimeT.getMultinomial(0).setProbabilities(new double[]{0.3, 0.7});
        distE_TimeT.getMultinomial(1).setProbabilities(new double[]{0.6, 0.4});
        distE_TimeT.getMultinomial(2).setProbabilities(new double[]{0.7, 0.3});
        distE_TimeT.getMultinomial(3).setProbabilities(new double[]{0.9, 0.1});

        // Variable G
        Multinomial_MultinomialParents distG_TimeT = dbn.
↪getConditionalDistributionTimeT(G);
        distG_TimeT.getMultinomial(0).setProbabilities(new double[]{0.2, 0.8});
        distG_TimeT.getMultinomial(1).setProbabilities(new double[]{0.5, 0.5});
        distG_TimeT.getMultinomial(2).setProbabilities(new double[]{0.3, 0.7});
        distG_TimeT.getMultinomial(3).setProbabilities(new double[]{0.8, 0.2});

        /*
         * We print the new DBN
         */
        System.out.println(dbn.toString());

    }
}
```

1.6.4 Sampling from Dynamic Bayesian Networks

This example shows how to use the *DynamicBayesianNetworkSampler* class to randomly generate a dynamic data stream from a given Dynamic Bayesian network.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
```

(continues on next page)

(continued from previous page)

```
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.utils;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamWriter;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkSampler;

import java.util.Random;

/**
 * This example shows how to use the DynamicBayesianNetworkSampler class to randomly␣
↪generate a data sample
 * for a given Dynamic Bayesian network.
 *
 * Created by ana@cs.aau.dk on 02/12/15.
 */
public class DynamicBayesianNetworkSamplerExample {
    public static void main(String[] args) throws Exception{

        //We first generate a DBN with 3 continuous and 3 discrete variables with 2␣
↪states
        DynamicBayesianNetworkGenerator dbnGenerator = new␣
↪DynamicBayesianNetworkGenerator();
        dbnGenerator.setNumberOfContinuousVars(3);
        dbnGenerator.setNumberOfDiscreteVars(3);
        dbnGenerator.setNumberOfStates(2);

        //Create a NB-like structure with temporal links in the children (leaves) and␣
↪2 states for
        //the class variable
        DynamicBayesianNetwork network = DynamicBayesianNetworkGenerator.
↪generateDynamicNaiveBayes(
                new Random(0), 2, true);

        //Create the sampler from this network
        DynamicBayesianNetworkSampler sampler = new␣
↪DynamicBayesianNetworkSampler(network);
```

(continues on next page)

(continued from previous page)

```
        sampler.setSeed(0);

        //Sample a dataStream of 3 sequences of 1000 samples each
        DataStream<DynamicDataInstance> dataStream = sampler.sampleToDataBase(3,1000);

        //Save the created data sample in a file
        DataStreamWriter.writeDataToFile(dataStream, "./datasets/simulated/dnb-
↪samples.arff");
    }
}
```

1.6.5 Inference Algorithms for Dynamic Bayesian Networks

The Dynamic MAP Inference

This example shows how to use the Dynamic MAP Inference algorithm.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.inference;

import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;
import eu.amidst.dynamic.variables.DynamicAssignment;
import eu.amidst.dynamic.variables.HashMapDynamicAssignment;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
```

(continues on next page)

(continued from previous page)

```
/**
 * This example shows how to use the Dynamic MAP Inference algorithm described in␣
↪Deliverable 3.4 (Section 6).
 * Created by dario on 11/11/15.
 */
public class DynamicMAPInference {

    public static void main(String[] arguments) throws IOException,␣
↪ClassNotFoundException {

        /*
         * LOADS THE DYNAMIC NETWORK AND PRINTS IT
         */
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(3);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(2);
        DynamicBayesianNetworkGenerator.setNumberOfLinks(5);

        DynamicBayesianNetwork dynamicBayesianNetwork =␣
↪DynamicBayesianNetworkGenerator.generateDynamicNaiveBayes(new Random(0), 2, true);


        /*
         *  INITIALIZE THE DYNAMIC MAP OBJECT
         */
        int nTimeSteps = 6;
        eu.amidst.dynamic.inference.DynamicMAPInference dynMAP = new eu.amidst.
↪dynamic.inference.DynamicMAPInference();
        dynMAP.setModel(dynamicBayesianNetwork);
        dynMAP.setNumberOfTimeSteps(nTimeSteps);

        Variable mapVariable = dynamicBayesianNetwork.getDynamicVariables().
↪getVariableByName("ClassVar");
        dynMAP.setMAPvariable(mapVariable);


        /*
         * GENERATE AN EVIDENCE FOR T=0,...,nTimeSteps-1
         */
        List<Variable> varsDynamicModel = dynamicBayesianNetwork.
↪getDynamicVariables().getListOfDynamicVariables();

        System.out.println("DYNAMIC VARIABLES:");
        varsDynamicModel.forEach(var -> System.out.println("Var ID " + var.getVarID()␣
↪+ ": " + var.getName()));
        System.out.println();
        int indexVarEvidence1 = 2;
        int indexVarEvidence2 = 3;
        int indexVarEvidence3 = 4;
        Variable varEvidence1 = varsDynamicModel.get(indexVarEvidence1);
        Variable varEvidence2 = varsDynamicModel.get(indexVarEvidence2);
        Variable varEvidence3 = varsDynamicModel.get(indexVarEvidence3);

        List<Variable> varsEvidence = new ArrayList<>(3);
        varsEvidence.add(0,varEvidence1);
        varsEvidence.add(1,varEvidence2);
```

(continues on next page)

(continued from previous page)

```
        varsEvidence.add(2,varEvidence3);

        double varEvidenceValue;

        Random random = new Random(4634);

        List<DynamicAssignment> evidence = new ArrayList<>(nTimeSteps);

        for (int t = 0; t < nTimeSteps; t++) {
            HashMapDynamicAssignment dynAssignment = new␣
↪HashMapDynamicAssignment(varsEvidence.size());

            for (int i = 0; i < varsEvidence.size(); i++) {

                dynAssignment.setSequenceID(12302253);
                dynAssignment.setTimeID(t);
                Variable varEvidence = varsEvidence.get(i);

                if (varEvidence.isMultinomial()) {
                    varEvidenceValue = random.nextInt(varEvidence1.
↪getNumberOfStates());
                } else {
                    varEvidenceValue = -5 + 10 * random.nextDouble();
                }
                dynAssignment.setValue(varEvidence, varEvidenceValue);
            }
            evidence.add(dynAssignment);
        }
        System.out.println("EVIDENCE:");
        evidence.forEach(evid -> {
            System.out.println("Evidence at time " + evid.getTimeID());
            evid.getVariables().forEach(variable -> System.out.println(variable.
↪getName() + ": " + Integer.toString((int) evid.getValue(variable))));
            System.out.println();
        });

        /*
         *  SET THE EVIDENCE AND MAKE INFERENCE
         */
        dynMAP.setEvidence(evidence);
        dynMAP.runInference(eu.amidst.dynamic.inference.DynamicMAPInference.
↪SearchAlgorithm.IS);

        /*
         *  SHOW RESULTS
         */
        Assignment MAPestimate = dynMAP.getMAPestimate();
        double MAPestimateProbability = dynMAP.getMAPestimateProbability();

        System.out.println("MAP sequence over " + mapVariable.getName() + ":");
        List<Variable> MAPvarReplications = MAPestimate.getVariables().stream().
↪sorted((var1,var2) -> (var1.getVarID()>var2.getVarID()? 1 : -1)).collect(Collectors.
↪toList());

        StringBuilder sequence = new StringBuilder();
        MAPvarReplications.stream().forEachOrdered(var -> sequence.append(Integer.
↪toString((int) MAPestimate.getValue(var)) + ", "));
```

(continues on next page)

(continued from previous page)

```
        //System.out.println(MAPestimate.outputString(MAPvarReplications));
        System.out.println(sequence.toString());
        System.out.println("with probability prop. to: " + MAPestimateProbability);

    }
}
```

The Dynamic Variational Message Passing

This example shows how to use the Factored Frontier algorithm with Variational Message Passing for running inference on dynamic Bayesian networks.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.inference;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.distribution.UnivariateDistribution;
import eu.amidst.core.inference.messagepassing.VMP;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.inference.FactoredFrontierForDBN;
import eu.amidst.dynamic.inference.InferenceEngineForDBN;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkSampler;

import java.io.IOException;
import java.util.Random;

/**
 * This example shows how to use the Factored Frontier algorithm with Variational␣
↪Message Passing described in
```

(continues on next page)

(continued from previous page)

```
 * Deliverable 3.4 (Section 6).
 * Created by ana@cs.aau.dk on 16/11/15.
 */
public class DynamicVMP_FactoredFrontier {
    public static void main(String[] args) throws IOException {

        Random random = new Random(1);

        //We first generate a dynamic Bayesian network (NB structure with class and␣
↪attributes temporally linked)
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(3);
        DynamicBayesianNetwork extendedDBN = DynamicBayesianNetworkGenerator.
↪generateDynamicNaiveBayes(random,2,true);

        System.out.println(extendedDBN.toString());

        //We select the target variable for inference, in this case the class variable
        Variable classVar = extendedDBN.getDynamicVariables().getVariableByName(
↪"ClassVar");


        //We create a dynamic dataset with 3 sequences for prediction. The class var␣
↪is made hidden.
        DynamicBayesianNetworkSampler dynamicSampler = new␣
↪DynamicBayesianNetworkSampler(extendedDBN);
        dynamicSampler.setHiddenVar(classVar);
        DataStream<DynamicDataInstance> dataPredict = dynamicSampler.
↪sampleToDataBase(1,10);


        //We select VMP with the factored frontier algorithm as the Inference␣
↪Algorithm
        FactoredFrontierForDBN FFalgorithm = new FactoredFrontierForDBN(new VMP());
        InferenceEngineForDBN.setInferenceAlgorithmForDBN(FFalgorithm);

        //Then, we set the DBN model
        InferenceEngineForDBN.setModel(extendedDBN);

        int time = 0 ;
        UnivariateDistribution posterior = null;
        for (DynamicDataInstance instance : dataPredict) {
            //The InferenceEngineForDBN must be reset at the begining of each␣
↪Sequence.
            if (instance.getTimeID()==0 && posterior != null) {
                InferenceEngineForDBN.reset();
                time=0;
            }
            //We also set the evidence.
            InferenceEngineForDBN.addDynamicEvidence(instance);

            //Then we run inference
            InferenceEngineForDBN.runInference();

            //Then we query the posterior of the target variable
            posterior = InferenceEngineForDBN.getFilteredPosterior(classVar);
```

(continues on next page)

(continued from previous page)

```
            //We show the output
            System.out.println("P(ClassVar|e[0:"+(time++)+"]) = "+posterior);
        }
    }
}
```

The Dynamic Importance Sampling

This example shows how to use the Factored Frontier algorithm with Importance Sampling for running inference in dynamic Bayesian networks.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.inference;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.distribution.UnivariateDistribution;
import eu.amidst.core.inference.ImportanceSampling;
import eu.amidst.core.variables.Variable;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.inference.FactoredFrontierForDBN;
import eu.amidst.dynamic.inference.InferenceEngineForDBN;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkSampler;

import java.io.IOException;
import java.util.Random;

/**
 * This example shows how to use the Factored Frontier algorithm with Importance␣
↪Sampling described in
```

(continues on next page)

(continued from previous page)

```
 * Deliverable 3.4 (Section 6).
 * Created by ana@cs.aau.dk on 16/11/15.
 */
public class DynamicIS_FactoredFrontier {
    public static void main(String[] args) throws IOException {

        Random random = new Random(1);

        //We first generate a dynamic Bayesian network (NB structure with class and␣
↪attributes temporally linked)
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(3);
        DynamicBayesianNetwork extendedDBN = DynamicBayesianNetworkGenerator.
↪generateDynamicNaiveBayes(random, 2, true);

        System.out.println(extendedDBN.toString());

        //We select the target variable for inference, in this case the class variable
        Variable classVar = extendedDBN.getDynamicVariables().getVariableByName(
↪"ClassVar");

        //We create a dynamic dataset with 3 sequences for prediction. The class var␣
↪is made hidden.
        DynamicBayesianNetworkSampler dynamicSampler = new␣
↪DynamicBayesianNetworkSampler(extendedDBN);
        dynamicSampler.setHiddenVar(classVar);
        DataStream<DynamicDataInstance> dataPredict = dynamicSampler.
↪sampleToDataBase(1, 10);

        //We select IS with the factored frontier algorithm as the Inference Algorithm
        ImportanceSampling importanceSampling = new ImportanceSampling();
        importanceSampling.setKeepDataOnMemory(true);
        FactoredFrontierForDBN FFalgorithm = new␣
↪FactoredFrontierForDBN(importanceSampling);
        InferenceEngineForDBN.setInferenceAlgorithmForDBN(FFalgorithm);

        //Then, we set the DBN model
        InferenceEngineForDBN.setModel(extendedDBN);

        int time = 0;
        UnivariateDistribution posterior = null;
        for (DynamicDataInstance instance : dataPredict) {
            //The InferenceEngineForDBN must be reset at the begining of each␣
↪Sequence.
            if (instance.getTimeID() == 0 && posterior != null) {
                InferenceEngineForDBN.reset();
                time = 0;
            }

            //We also set the evidence.
            InferenceEngineForDBN.addDynamicEvidence(instance);

            //Then we run inference
            InferenceEngineForDBN.runInference();

            //Then we query the posterior of the target variable
```

(continues on next page)

(continued from previous page)

```
            posterior = InferenceEngineForDBN.getFilteredPosterior(classVar);


            //We show the output
            System.out.println("P(ClassVar|e[0:" + (time++) + "]) = " + posterior);
        }
    }
}
```

1.6.6 Dynamic Learning Algorithms

Maximum Likelihood for DBNs

This example shows how to learn the parameters of a dynamic Bayesian network using maximum likelihood from a randomly sampled data stream.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.learning;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.learning.parametric.ParallelMaximumLikelihood;
import eu.amidst.dynamic.learning.parametric.ParameterLearningAlgorithm;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkSampler;

import java.io.IOException;
import java.util.Random;

/**
```

(continues on next page)

(continued from previous page)

```
 *
 * This example shows how to learn the parameters of a dynamic Bayesian network using␣
↪maximum likelihood
 * from a sample data.
 *
 * Created by ana@cs.aau.dk on 01/12/15.
 */
public class MLforDBNsampling {

    public static void main(String[] args) throws IOException {
        Random random = new Random(1);

        //We first generate a dynamic Bayesian network (NB structure with class and␣
↪attributes temporally linked)
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(3);
        DynamicBayesianNetwork dbnRandom = DynamicBayesianNetworkGenerator.
↪generateDynamicNaiveBayes(random,2,true);

        //Sample dynamic data from the created dbn with random parameters
        DynamicBayesianNetworkSampler sampler = new␣
↪DynamicBayesianNetworkSampler(dbnRandom);
        sampler.setSeed(0);
        //Sample 3 sequences of 100K instances
        DataStream<DynamicDataInstance> data = sampler.sampleToDataBase(3,10000);

        /*Parameter Learning with ML*/

        //We fix the DAG structure, the data and learn the DBN
        ParameterLearningAlgorithm parallelMaximumLikelihood = new␣
↪ParallelMaximumLikelihood();
        parallelMaximumLikelihood.setWindowsSize(1000);
        parallelMaximumLikelihood.setDynamicDAG(dbnRandom.getDynamicDAG());
        parallelMaximumLikelihood.initLearning();
        parallelMaximumLikelihood.updateModel(data);

        DynamicBayesianNetwork dbnLearnt = parallelMaximumLikelihood.getLearntDBN();

        //We print the model
        System.out.println(dbnLearnt.toString());
    }

}
```

Streaming Variational Bayes for DBNs

This example shows how to learn the parameters of a dynamic Bayesian network using streaming variational Bayes from a randomly sampled data stream.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
```

(continues on next page)

(continued from previous page)

```
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.dynamic.examples.learning;

import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.inference.messagepassing.VMP;
import eu.amidst.dynamic.datastream.DynamicDataInstance;
import eu.amidst.dynamic.learning.parametric.bayesian.SVB;
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkSampler;

import java.io.IOException;
import java.util.Random;

/**
 * Created by ana@cs.aau.dk on 01/12/15.
 */
public class SVBforDBN {
    public static void main(String[] args) throws IOException {
        Random random = new Random(1);

        //We first generate a dynamic Bayesian network (NB structure with class and␣
↪attributes temporally linked)
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(3);
        DynamicBayesianNetwork dbnRandom = DynamicBayesianNetworkGenerator.
↪generateDynamicNaiveBayes(random,2,true);

        //Sample dynamic data from the created dbn with random parameters
        DynamicBayesianNetworkSampler sampler = new␣
↪DynamicBayesianNetworkSampler(dbnRandom);
        sampler.setSeed(0);
        //Sample 3 sequences of 100K instances
        DataStream<DynamicDataInstance> data = sampler.sampleToDataBase(3,10000);

        /*Parameter Learning with Streaming variational Bayes VMP*/
        SVB svb = new SVB();
        //We set the desired options for the svb
```

(continues on next page)

(continued from previous page)

```
        svb.setWindowsSize(100);
        svb.setSeed(0);
        //If desired, we also set some options for the VMP
        VMP vmp = svb.getPlateauStructure().getVMPTimeT();
        vmp.setOutput(false);
        vmp.setTestELBO(true);
        vmp.setMaxIter(1000);
        vmp.setThreshold(0.0001);

        //We set the dynamicDAG, the data and start learning
        svb.setDynamicDAG(dbnRandom.getDynamicDAG());
        svb.setDataStream(data);
        svb.runLearning();

        //We get the learnt DBN
        DynamicBayesianNetwork dbnLearnt = svb.getLearntDBN();

        //We print the model
        System.out.println(dbnLearnt.toString());
    }

}
```

1.7 Bayesian Networks: Code Examples

  - *Naive Bayes with Virtual Concept Drift Detection*
- *HuginLink*
  - *Models conversion between AMIDST and Hugin*
  - *I/O of Bayesian Networks with Hugin net format*
  - *Invoking Hugin's inference engine*
  - *Invoking Hugin's Parallel TAN*
- *MoaLink*
  - *AMIDST Classifiers from MOA*

1.7.1 Data Streams

In this example we show how to use the main features of a DataStream object. More precisely, we show six different ways of iterating over the data samples of a DataStream object.

```
package eu.amidst.core.examples.datastream;


import eu.amidst.core.datastream.Attribute;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataOnMemory;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.utils.DataSetGenerator;

/**
 * An example showing how to use the main features of a DataStream object. More␣
↪precisely, we show six different
 * ways of iterating over the data samples of a DataStream object.
 */
public class DataStreamsExample {

    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        //DataStream<DataInstance> data = DataStreamLoader.open("datasetsTests/data.
↪arff");

        //Generate the data stream using the class DataSetGenerator
        DataStream<DataInstance> data = DataSetGenerator.generate(1,10,5,5);


        //Access to the attributes defining the data set
        System.out.println("Attributes defining the data set");
        for (Attribute attribute : data.getAttributes()) {
            System.out.println(attribute.getName());
        }
        Attribute discreteVar0 = data.getAttributes().getAttributeByName("DiscreteVar0
↪");

        //1. Iterating over samples using a for loop
        System.out.println("1. Iterating over samples using a for loop");
        for (DataInstance dataInstance : data) {
            System.out.println("The value of attribute A for the current data␣
↪instance is: " + dataInstance.getValue(discreteVar0));
```

(continues on next page)

(continued from previous page)

```
        }


        //2. Iterating using streams. We need to restart the data again as a␣
↪DataStream can only be used once.
        System.out.println("2. Iterating using streams.");
        data.restart();
        data.stream().forEach(dataInstance ->
                        System.out.println("The value of attribute A for the current␣
↪data instance is: " + dataInstance.getValue(discreteVar0))
        );


        //3. Iterating using parallel streams.
        System.out.println("3. Iterating using parallel streams.");
        data.restart();
        data.parallelStream(10).forEach(dataInstance ->
                        System.out.println("The value of attribute A for the current␣
↪data instance is: " + dataInstance.getValue(discreteVar0))
        );

        //4. Iterating over a stream of data batches.
        System.out.println("4. Iterating over a stream of data batches.");
        data.restart();
        data.streamOfBatches(10).forEach(batch -> {
            for (DataInstance dataInstance : batch)
                System.out.println("The value of attribute A for the current data␣
↪instance is: " + dataInstance.getValue(discreteVar0));
        });

        //5. Iterating over a parallel stream of data batches.
        System.out.println("5. Iterating over a parallel stream of data batches.");
        data.restart();
        data.parallelStreamOfBatches(10).forEach(batch -> {
            for (DataInstance dataInstance : batch)
                System.out.println("The value of attribute A for the current data␣
↪instance is: " + dataInstance.getValue(discreteVar0));
        });


        //6. Iterating over data batches using a for loop
        System.out.println("6. Iterating over data batches using a for loop.");
        for (DataOnMemory<DataInstance> batch : data.iterableOverBatches(10)) {
            for (DataInstance dataInstance : batch)
                System.out.println("The value of attribute A for the current data␣
↪instance is: " + dataInstance.getValue(discreteVar0));
        }
    }

}
```

1.7.2 Data Streams

This example show the basic functionality of the classes Variables and Variable.

```
package eu.amidst.core.examples.variables;


import eu.amidst.core.variables.Variable;
import eu.amidst.core.variables.Variables;
import eu.amidst.core.variables.stateSpaceTypes.FiniteStateSpace;

import java.util.Arrays;

/**
 *
 * This example show the basic functionality of the classes Variables and Variable.
 *
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class VariablesExample {

    public static void main(String[] args) throws Exception {

        //We first create an empty Variables object
        Variables variables = new Variables();

        //We invoke the "new" methods of the object Variables to create new variables.
        //Now we create a Gaussian variables
        Variable gaussianVar = variables.newGaussianVariable("Gaussian");

        //Now we create a Multinomial variable with two states
        Variable multinomialVar = variables.newMultinomialVariable("Multinomial", 2);

        //Now we create a Multinomial variable with two states: TRUE and FALSE
        Variable multinomialVar2 = variables.newMultinomialVariable("Multinomial2",␣
↪Arrays.asList("TRUE, FALSE"));

        //For Multinomial variables we can iterate over their different states
        FiniteStateSpace states = multinomialVar2.getStateSpaceType();
        states.getStatesNames().forEach(System.out::println);

        //Variable objects can also be used, for example, to know if one variable can␣
↪be set as parent of some other variable
        System.out.println("Can a Gaussian variable be parent of Multinomial variable?
↪ " +
                (multinomialVar.getDistributionType().
↪isParentCompatible(gaussianVar)));

        System.out.println("Can a Multinomial variable be parent of Gaussian variable?
↪ " +
                (gaussianVar.getDistributionType().
↪isParentCompatible(multinomialVar)));

    }
}
```

[Back to Top]

1.7.3 Models

Creating BNs

In this example, we take a data set, create a BN and we compute the log-likelihood of all the samples of this data set. The numbers defining the probability distributions of the BN are randomly fixed.

```
package eu.amidst.core.examples.models;

import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.BayesianNetworkWriter;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.variables.Variable;
import eu.amidst.core.variables.Variables;

/**
 *  In this example, we take a data set, create a BN and we compute the log-
↪likelihood of all the samples
 *  of this data set. The numbers defining the probability distributions of the BN␣
↪are randomly fixed.
 *  Created by andresmasegosa on 18/6/15.
 */
public class CreatingBayesianNetworks {


    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪syntheticData.arff");


        /**
         * 1. Once the data is loaded, we create a random variable for each of the␣
↪attributes (i.e. data columns)
         * in our data.
         *
         * 2. {@link Variables} is the class for doing that. It takes a list of␣
↪Attributes and internally creates
         * all the variables. We create the variables using Variables class to␣
↪guarantee that each variable
         * has a different ID number and make it transparent for the user.
         *
         * 3. We can extract the Variable objects by using the method␣
↪getVariableByName();
         */
        Variables variables = new Variables(data.getAttributes());

        Variable a = variables.getVariableByName("A");
        Variable b = variables.getVariableByName("B");
        Variable c = variables.getVariableByName("C");
        Variable d = variables.getVariableByName("D");
        Variable e = variables.getVariableByName("E");
        Variable g = variables.getVariableByName("G");
```

(continues on next page)

(continued from previous page)

```
        Variable h = variables.getVariableByName("H");
        Variable i = variables.getVariableByName("I");

        /**
         * 1. Once you have defined your {@link Variables} object, the next step is␣
↪to create
         * a DAG structure over this set of variables.
         *
         * 2. To add parents to each variable, we first recover the ParentSet object␣
↪by the method
         * getParentSet(Variable var) and then call the method addParent().
         */
        DAG dag = new DAG(variables);

        dag.getParentSet(e).addParent(a);
        dag.getParentSet(e).addParent(b);

        dag.getParentSet(h).addParent(a);
        dag.getParentSet(h).addParent(b);

        dag.getParentSet(i).addParent(a);
        dag.getParentSet(i).addParent(b);
        dag.getParentSet(i).addParent(c);
        dag.getParentSet(i).addParent(d);

        dag.getParentSet(g).addParent(c);
        dag.getParentSet(g).addParent(d);

        /**
         * 1. We first check if the graph contains cycles.
         *
         * 2. We print out the created DAG. We can check that everything is as␣
↪expected.
         */
        if (dag.containCycles()) {
            try {
            } catch (Exception ex) {
                throw new IllegalArgumentException(ex);
            }
        }

        System.out.println(dag.toString());


        /**
         * 1. We now create the Bayesian network from the previous DAG.
         *
         * 2. The BN object is created from the DAG. It automatically looks at the␣
↪distribution tye
         * of each variable and their parents to initialize the Distributions objects␣
↪that are stored
         * inside (i.e. Multinomial, Normal, CLG, etc). The parameters defining these␣
↪distributions are
         * properly initialized.
         *
         * 3. The network is printed and we can have look at the kind of␣
↪distributions stored in the BN object.
```

(continues on next page)

(continued from previous page)

```
         */
        BayesianNetwork bn = new BayesianNetwork(dag);
        System.out.println(bn.toString());


        /**
         * 1. We iterate over the data set sample by sample.
         *
         * 2. For each sample or DataInstance object, we compute the log of the␣
↪probability that the BN object
         * assigns to this observation.
         *
         * 3. We accumulate these log-probs and finally we print the log-prob of the␣
↪data set.
         */
        double logProb = 0;
        for (DataInstance instance : data) {
            logProb += bn.getLogProbabiltyOf(instance);
        }
        System.out.println(logProb);

        BayesianNetworkWriter.save(bn, "networks/simulated/BNExample.bn");
    }
}
```

[Back to Top]

Creating Bayesian networks with latent variables

In this example, we simply show how to create a BN model with hidden variables. We simply create a BN for clustering, i.e., a naive-Bayes like structure with a single common hidden variable acting as parant of all the observable variables.

```
package eu.amidst.core.examples.models;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.BayesianNetworkWriter;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.variables.Variable;
import eu.amidst.core.variables.Variables;

import java.util.Arrays;

/**
 * In this example, we simply show how to create a BN model with latent variables. We␣
↪simply
 * create a BN for clustering, i.e.,  a naive-Bayes like structure with a single␣
↪common latent or hidden variable
 * acting as parent of all the observable variables.
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class CreatingBayesianNetworksWithLatentVariables {
```

(continues on next page)

(continued from previous page)

```
    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪syntheticData.arff");

        /**
         * 1. Once the data is loaded, we create a random variable for each of the␣
↪attributes (i.e. data columns)
         * in our data.
         *
         * 2. {@link Variables} is the class for doing that. It takes a list of␣
↪Attributes and internally creates
         * all the variables. We create the variables using Variables class to␣
↪guarantee that each variable
         * has a different ID number and make it transparent for the user.
         *
         * 3. We can extract the Variable objects by using the method␣
↪getVariableByName();
         */
        Variables variables = new Variables(data.getAttributes());

        Variable a = variables.getVariableByName("A");
        Variable b = variables.getVariableByName("B");
        Variable c = variables.getVariableByName("C");
        Variable d = variables.getVariableByName("D");
        Variable e = variables.getVariableByName("E");
        Variable g = variables.getVariableByName("G");
        Variable h = variables.getVariableByName("H");
        Variable i = variables.getVariableByName("I");

        /**
         * 1. We create the hidden variable. For doing that we make use of the method
↪"newMultinomialVariable". When
         * a variable is created from an Attribute object, it contains all the␣
↪information we need (e.g.
         * the name, the type, etc). But hidden variables does not have an associated␣
↪attribute
         * and, for this reason, we use now this to provide this information.
         *
         * 2. Using the "newMultinomialVariable" method, we define a variable called␣
↪HiddenVar, which is
         * not associated to any attribute and, then, it is a latent variable, its␣
↪state space is a finite set with two elements, and its
         * distribution type is multinomial.
         *
         * 3. We finally create the hidden variable using the method "newVariable".
         */

        Variable hidden = variables.newMultinomialVariable("HiddenVar", Arrays.asList(
↪"TRUE", "FALSE"));

        /**
         * 1. Once we have defined your {@link Variables} object, including the␣
↪latent variable,
         * the next step is to create a DAG structure over this set of variables.
         *
```

(continues on next page)

(continued from previous page)

```
         * 2. To add parents to each variable, we first recover the ParentSet object␣
↪by the method
         * getParentSet(Variable var) and then call the method addParent(Variable␣
↪var).
         *
         * 3. We just put the hidden variable as parent of all the other variables.␣
↪Following a naive-Bayes
         * like structure.
         */
        DAG dag = new DAG(variables);

        dag.getParentSet(a).addParent(hidden);
        dag.getParentSet(b).addParent(hidden);
        dag.getParentSet(c).addParent(hidden);
        dag.getParentSet(d).addParent(hidden);
        dag.getParentSet(e).addParent(hidden);
        dag.getParentSet(g).addParent(hidden);
        dag.getParentSet(h).addParent(hidden);
        dag.getParentSet(i).addParent(hidden);

        /**
         * We print the graph to see if is properly created.
         */
        System.out.println(dag.toString());

        /**
         * 1. We now create the Bayesian network from the previous DAG.
         *
         * 2. The BN object is created from the DAG. It automatically looks at the␣
↪distribution type
         * of each variable and their parents to initialize the Distributions objects␣
↪that are stored
         * inside (i.e. Multinomial, Normal, CLG, etc). The parameters defining these␣
↪distributions are
         * properly initialized.
         *
         * 3. The network is printed and we can have look at the kind of␣
↪distributions stored in the BN object.
         */
        BayesianNetwork bn = new BayesianNetwork(dag);
        System.out.println(bn.toString());

        /**
         * Finally the Bayesian network is saved to a file.
         */
        BayesianNetworkWriter.save(bn, "networks/simulated/BNHiddenExample.bn");

    }
}
```

[Back to Top]

Modifying Bayesian networks

In this example we show how to access and modify the conditional probabilities of a Bayesian network model.

```
package eu.amidst.core.examples.models;
import eu.amidst.core.distribution.Multinomial;
import eu.amidst.core.distribution.Normal_MultinomialParents;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.utils.BayesianNetworkGenerator;
import eu.amidst.core.variables.Variable;

/**
 *
 * In this example we show how to access and modify the conditional probabilities of␣
↪a Bayesian network model.
 * Created by andresmasegosa on 24/6/15.
 */
public class ModifiyingBayesianNetworks {

    public static void main (String[] args){

        //We first generate a Bayesian network with one multinomial, one Gaussian␣
↪variable and one link
        BayesianNetworkGenerator.setNumberOfGaussianVars(1);
        BayesianNetworkGenerator.setNumberOfMultinomialVars(1,2);
        BayesianNetworkGenerator.setNumberOfLinks(1);

        BayesianNetwork bn = BayesianNetworkGenerator.generateBayesianNetwork();

        //We print the randomly generated Bayesian networks
        System.out.println(bn.toString());

        //We first access the variable we are interested in
        Variable multiVar = bn.getVariables().getVariableByName("DiscreteVar0");

        //Using the above variable we can get the associated distribution and modify␣
↪it
        Multinomial multinomial = bn.getConditionalDistribution(multiVar);
        multinomial.setProbabilities(new double[]{0.2, 0.8});

        //Same than before but accessing the another variable
        Variable normalVar = bn.getVariables().getVariableByName("GaussianVar0");

        //In this case, the conditional distribtuion is of the type "Normal given␣
↪Multinomial Parents"
        Normal_MultinomialParents normalMultiDist = bn.
↪getConditionalDistribution(normalVar);
        normalMultiDist.getNormal(0).setMean(1.0);
        normalMultiDist.getNormal(0).setVariance(1.0);

        normalMultiDist.getNormal(1).setMean(0.0);
        normalMultiDist.getNormal(1).setVariance(1.0);

        //We print modified Bayesian network
        System.out.println(bn.toString());
    }
}
```

[Back to Top]

1.7.4 Input/Output

I/O of data streams

In this example we show how to load and save data sets from .arff files.

```
package eu.amidst.core.examples.io;


import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.io.DataStreamWriter;

/**
 *
 * In this example we show how to load and save data sets from ".arff" files (http://
↪www.cs.waikato.ac.nz/ml/weka/arff.html)
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class DataStreamIOExample {

    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪syntheticData.arff");

        //We can save this data set to a new file using the static class␣
↪DataStreamWriter
        DataStreamWriter.writeDataToFile(data, "datasets/simulated/tmp.arff");



    }
}
```

[Back to Top]

I/O of BNs

In this example we show how to load and save Bayesian networks models for a binary file with ".bn" extension. In this toolbox Bayesian networks models are saved as serialized objects.

```
package eu.amidst.core.examples.io;


import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.io.BayesianNetworkWriter;
import eu.amidst.core.models.BayesianNetwork;

import java.util.Random;

/**
 *
```

(continues on next page)

(continued from previous page)

```
 * In this example we show how to load and save Bayesian networks models for a binary␣
↪file with ".bn" extension. In
 * this toolbox Bayesian networks models are saved as serialized objects.
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class BayesianNetworkIOExample {

    public static void main(String[] args) throws Exception {

        //We can load a Bayesian network using the static class BayesianNetworkLoader
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //Now we print the loaded model
        System.out.println(bn.toString());

        //Now we change the parameters of the model
        bn.randomInitialization(new Random(0));

        //We can save this Bayesian network to using the static class␣
↪BayesianNetworkWriter
        BayesianNetworkWriter.save(bn, "networks/simulated/tmp.bn");

    }
}
```

[Back to Top]

1.7.5 Inference

The inference engine

This example show how to perform inference in a Bayesian network model using the InferenceEngine static class. This class aims to be a straigthfoward way to perform queries over a Bayesian network model. By the default the *VMP* inference method is invoked.

```
package eu.amidst.core.examples.inference;


import eu.amidst.core.inference.InferenceEngine;
import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.HashMapAssignment;
import eu.amidst.core.variables.Variable;

/**
 * This example show how to perform inference in a Bayesian network model using the␣
↪InferenceEngine static class.
 * This class aims to be a straigthfoward way to perform queries over a Bayesian␣
↪network model.
 *
 * Created by andresmasegosa on 18/6/15.
 */
```

(continues on next page)

(continued from previous page)

```
public class InferenceEngineExample {

    public static void main(String[] args) throws Exception {

        //We first load the WasteIncinerator bayesian network which has multinomial␣
↪and Gaussian variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //We recover the relevant variables for this example: Mout which is normally␣
↪distributed, and W which is multinomial.
        Variable varMout = bn.getVariables().getVariableByName("Mout");
        Variable varW = bn.getVariables().getVariableByName("W");

        //Set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW,0);

        //Then we query the posterior of
        System.out.println("P(Mout|W=0) = " + InferenceEngine.getPosterior(varMout,␣
↪bn, assignment));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<6.59 | W=0) = " + InferenceEngine.
↪getExpectedValue(varMout, bn, v -> (0.7 < v && v < 6.59) ? 1.0 : 0.0 ));

    }

}
```

[Back to Top]

1.7.6 Inference

Variational Message Passing

This example we show how to perform inference on a general Bayesian network using the Variational Message Passing (VMP) algorithm detailed in

Winn, J. M., Bishop, C. M. (2005). Variational message passing. In Journal of Machine Learning Research (pp. 661-694).

```
package eu.amidst.core.examples.inference;

import eu.amidst.core.inference.InferenceAlgorithm;
import eu.amidst.core.inference.messagepassing.VMP;
import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.HashMapAssignment;
import eu.amidst.core.variables.Variable;

/**
 *
 * This example we show how to perform inference on a general Bayesian network using␣
↪the Variational Message Passing (VMP)
```

(continues on next page)

(continued from previous page)

```
 * algorithm detailed in
 *
 * <i> Winn, J. M., and Bishop, C. M. (2005). Variational message passing. In Journal␣
↪of Machine Learning Research (pp. 661-694). </i>
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class VMPExample {

    public static void main(String[] args) throws Exception {

        //We first load the WasteIncinerator bayesian network which has multinomial␣
↪and Gaussian variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //We recover the relevant variables for this example: Mout which is normally␣
↪distributed, and W which is multinomial.
        Variable varMout = bn.getVariables().getVariableByName("Mout");
        Variable varW = bn.getVariables().getVariableByName("W");

        //First we create an instance of a inference algorithm. In this case, we use␣
↪the VMP class.
        InferenceAlgorithm inferenceAlgorithm = new VMP();
        //Then, we set the BN model
        inferenceAlgorithm.setModel(bn);

        //If exists, we also set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW,0);
        inferenceAlgorithm.setEvidence(assignment);

        //Then we run inference
        inferenceAlgorithm.runInference();

        //Then we query the posterior of
        System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.
↪getPosterior(varMout));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<6.59 | W=0) = " + inferenceAlgorithm.
↪getExpectedValue(varMout, v -> (0.7 < v && v < 6.59) ? 1.0 : 0.0 ));

        //We can also compute the probability of the evidence
        System.out.println("P(W=0) = "+Math.exp(inferenceAlgorithm.
↪getLogProbabilityOfEvidence()));


    }
}
```

[Back to Top]

Importance Sampling

This example we show how to perform inference on a general Bayesian network using an importance sampling algorithm detailed in

Fung, R., Chang, K. C. (2013). Weighing and integrating evidence for stochastic simulation in Bayesian networks. arXiv preprint arXiv:1304.1504.

```
package eu.amidst.core.examples.inference;


import eu.amidst.core.inference.ImportanceSampling;
import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.HashMapAssignment;
import eu.amidst.core.variables.Variable;

/**
 *
 * This example we show how to perform inference on a general Bayesian network using␣
↪an importance sampling
 * algorithm detailed in
 *
 * <i> Fung, R., and Chang, K. C. (2013). Weighing and integrating evidence for
 * stochastic simulation in Bayesian networks. arXiv preprint arXiv:1304.1504.
 * </i>
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class ImportanceSamplingExample {

    public static void main(String[] args) throws Exception {

        //We first load the WasteIncinerator bayesian network which has multinomial␣
↪and Gaussian variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //We recover the relevant variables for this example: Mout which is normally␣
↪distributed, and W which is multinomial.
        Variable varMout = bn.getVariables().getVariableByName("Mout");
        Variable varW = bn.getVariables().getVariableByName("W");

        //First we create an instance of a inference algorithm. In this case, we use␣
↪the ImportanceSampling class.
        ImportanceSampling inferenceAlgorithm = new ImportanceSampling();
        //Then, we set the BN model
        inferenceAlgorithm.setModel(bn);

        System.out.println(bn.toString());

        //If it exists, we also set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW,0);
        inferenceAlgorithm.setEvidence(assignment);

        //We can also set to be run in parallel on multicore CPUs
```

(continues on next page)

(continued from previous page)

```
        inferenceAlgorithm.setParallelMode(true);

        //To perform more than one operation, data should be keep in memory
        inferenceAlgorithm.setKeepDataOnMemory(true);

        //Then we run inference
        inferenceAlgorithm.runInference();

        //Then we query the posterior of
        System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.
↪getPosterior(varMout));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<6.59 | W=0) = " + inferenceAlgorithm.
↪getExpectedValue(varMout, v -> (0.7 < v && v < 6.59) ? 1.0 : 0.0 ));

        //We can also compute the probability of the evidence
        System.out.println("P(W=0) = "+Math.exp(inferenceAlgorithm.
↪getLogProbabilityOfEvidence()));

    }
}
```

[Back to Top]

1.7.7 Learning Algorithms

Maximum Likelihood

This other example shows how to learn incrementally the parameters of a Bayesian network using data batches,

```
package eu.amidst.core.examples.learning;



import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataOnMemory;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.learning.parametric.ParallelMaximumLikelihood;
import eu.amidst.core.learning.parametric.ParameterLearningAlgorithm;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.models.DAG;
import eu.amidst.core.variables.Variable;
import eu.amidst.core.variables.Variables;

/**
 *
 * This other example shows how to learn incrementally the parameters of a Bayesian␣
↪network using data batches
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class MaximimumLikelihoodByBatchExample {

```

(continues on next page)

(continued from previous page)

```
    /**
     * This method returns a DAG object with naive Bayes structure for the attributes␣
↪of the passed data stream.
     * @param dataStream object of the class DataStream<DataInstance>
     * @param classIndex integer value indicating the position of the class
     * @return object of the class DAG
     */
    public static DAG getNaiveBayesStructure(DataStream<DataInstance> dataStream, int␣
↪classIndex){

        //We create a Variables object from the attributes of the data stream
        Variables modelHeader = new Variables(dataStream.getAttributes());

        //We define the predicitive class variable
        Variable classVar = modelHeader.getVariableById(classIndex);

        //Then, we create a DAG object with the defined model header
        DAG dag = new DAG(modelHeader);

        //We set the linkds of the DAG.
        dag.getParentSets().stream().filter(w -> w.getMainVar() != classVar).
↪forEach(w -> w.addParent(classVar));

        return dag;
    }


    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪WasteIncineratorSample.arff");

        //We create a ParameterLearningAlgorithm object with the MaximumLikehood␣
↪builder
        ParameterLearningAlgorithm parameterLearningAlgorithm = new␣
↪ParallelMaximumLikelihood();

        //We fix the DAG structure
        parameterLearningAlgorithm.setDAG(getNaiveBayesStructure(data,0));

        //We should invoke this method before processing any data
        parameterLearningAlgorithm.initLearning();


        //Then we show how we can perform parameter learnig by a sequential updating␣
↪of data batches.
        for (DataOnMemory<DataInstance> batch : data.iterableOverBatches(100)){
            parameterLearningAlgorithm.updateModel(batch);
        }

        //And we get the model
        BayesianNetwork bnModel = parameterLearningAlgorithm.
↪getLearntBayesianNetwork();

        //We print the model
```

(continues on next page)

(continued from previous page)

```
        System.out.println(bnModel.toString());

    }

}
```

[Back to Top]

Parallel Maximum Likelihood

This example shows how to learn in parallel the parameters of a Bayesian network from a stream of data using maximum likelihood.

```
package eu.amidst.core.examples.learning;


import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.learning.parametric.ParallelMaximumLikelihood;
import eu.amidst.core.models.BayesianNetwork;

/**
 *
 * This example shows how to learn in parallel the parameters of a Bayesian network␣
↪from a stream of data using maximum
 * likelihood.
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class ParallelMaximumLikelihoodExample {


    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪WasteIncineratorSample.arff");

        //We create a ParallelMaximumLikelihood object with the MaximumLikehood␣
↪builder
        ParallelMaximumLikelihood parameterLearningAlgorithm = new␣
↪ParallelMaximumLikelihood();

        //We activate the parallel mode.
        parameterLearningAlgorithm.setParallelMode(true);

        //We desactivate the debug mode.
        parameterLearningAlgorithm.setDebug(false);

        //We fix the DAG structure
        parameterLearningAlgorithm.setDAG(MaximimumLikelihoodByBatchExample.
↪getNaiveBayesStructure(data, 0));

        //We set the batch size which will be employed to learn the model in parallel
```

(continues on next page)

(continued from previous page)

```
        parameterLearningAlgorithm.setWindowsSize(100);

        //We set the data which is going to be used for leaning the parameters
        parameterLearningAlgorithm.setDataStream(data);

        //We perform the learning
        parameterLearningAlgorithm.runLearning();

        //And we get the model
        BayesianNetwork bnModel = parameterLearningAlgorithm.
↪getLearntBayesianNetwork();

        //We print the model
        System.out.println(bnModel.toString());

    }

}
```

[Back to Top]

Streaming Variational Bayes

This example shows how to learn incrementally the parameters of a Bayesian network from a stream of data with a Bayesian approach using the following algorithm,

> Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013). Streaming variational Bayes. In Advances in Neural Information Processing Systems (pp. 1727-1735).

In this second example we show a alternative implementation which explicitly updates the model by batches by using the class SVB.

```
package eu.amidst.core.examples.learning;



import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataOnMemory;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.learning.parametric.bayesian.SVB;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.utils.DAGGenerator;

/**
 *
 * This example shows how to learn incrementally the parameters of a Bayesian network␣
↪from a stream of data with a Bayesian
 * approach using the following algorithm
 *
 * <i> Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013).
↪ Streaming variational bayes.
 * In Advances in Neural Information Processing Systems (pp. 1727-1735). </i>
 *
 *
```

(continues on next page)

(continued from previous page)

```
 * Created by andresmasegosa on 18/6/15.
 */
public class SVBByBatchExample {


    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪WasteIncineratorSample.arff");

        //We create a SVB object
        SVB parameterLearningAlgorithm = new SVB();

        //We fix the DAG structure
        parameterLearningAlgorithm.setDAG(DAGGenerator.
↪getHiddenNaiveBayesStructure(data.getAttributes(),"H",2));

        //We fix the size of the window, which must be equal to the size of the data␣
↪batches we use for learning
        parameterLearningAlgorithm.setWindowsSize(100);

        //We can activate the output
        parameterLearningAlgorithm.setOutput(true);

        //We should invoke this method before processing any data
        parameterLearningAlgorithm.initLearning();


        //Then we show how we can perform parameter learning by a sequential updating␣
↪of data batches.
        for (DataOnMemory<DataInstance> batch : data.iterableOverBatches(100)){
            double log_likelihood_of_batch = parameterLearningAlgorithm.
↪updateModel(batch);
            System.out.println("Log-Likelihood of Batch: "+ log_likelihood_of_batch);
        }

        //And we get the model
        BayesianNetwork bnModel = parameterLearningAlgorithm.
↪getLearntBayesianNetwork();

        //We print the model
        System.out.println(bnModel.toString());

    }

}
```

[Back to Top]

Parallel **Streaming** **Variational** **Bayes**

This example shows how to learn in the parameters of a Bayesian network from a stream of data with a Bayesian approach using the parallel version of the SVB algorithm,

Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013). Streaming variational

Bayes. In Advances in Neural Information Processing Systems (pp. 1727-1735).

```
package eu.amidst.core.examples.learning;


import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.learning.parametric.bayesian.ParallelSVB;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.utils.DAGGenerator;

/**
 *
 * This example shows how to learn the parameters of a Bayesian network from a stream␣
↪of data with a Bayesian
 * approach using a **parallel** version of the following algorithm
 *
 * <i> Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013).
↪ Streaming variational Bayes.
 * In Advances in Neural Information Processing Systems (pp. 1727-1735). </i>
 *
 *
 * Created by andresmasegosa on 18/6/15.
 */
public class ParallelSVBExample {

    public static void main(String[] args) throws Exception {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("datasets/simulated/
↪WasteIncineratorSample.arff");

        //We create a ParallelSVB object
        ParallelSVB parameterLearningAlgorithm = new ParallelSVB();

        //We fix the number of cores we want to exploit
        parameterLearningAlgorithm.setNCores(4);

        //We fix the DAG structure, which is a Naive Bayes with a global latent␣
↪binary variable
        parameterLearningAlgorithm.setDAG(DAGGenerator.
↪getHiddenNaiveBayesStructure(data.getAttributes(), "H", 2));

        //We fix the size of the window
        parameterLearningAlgorithm.getSVBEngine().setWindowsSize(100);

        //We can activate the output
        parameterLearningAlgorithm.setOutput(true);

        //We set the data which is going to be used for leaning the parameters
        parameterLearningAlgorithm.setDataStream(data);

        //We perform the learning
        parameterLearningAlgorithm.runLearning();

        //And we get the model
        BayesianNetwork bnModel = parameterLearningAlgorithm.
↪getLearntBayesianNetwork();
```

(continues on next page)

(continued from previous page)

```
        //We print the model
        System.out.println(bnModel.toString());

    }

}
```

[Back to Top]

1.7.8 Concept Drift Methods

Naive Bayes with Virtual Concept Drift Detection

This example shows how to use the class NaiveBayesVirtualConceptDriftDetector to run the virtual concept drift detector detailed in

Borchani et al. Modeling concept drift: A probabilistic graphical model based approach. IDA 2015.

```
/*
 *
 *
 *    Licensed to the Apache Software Foundation (ASF) under one or more contributor␣
↪license agreements.
 *    See the NOTICE file distributed with this work for additional information␣
↪regarding copyright ownership.
 *    The ASF licenses this file to You under the Apache License, Version 2.0 (the
↪"License"); you may not use
 *    this file except in compliance with the License.  You may obtain a copy of the␣
↪License at
 *
 *            http://www.apache.org/licenses/LICENSE-2.0
 *
 *    Unless required by applicable law or agreed to in writing, software distributed␣
↪under the License is
 *    distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,␣
↪either express or implied.
 *    See the License for the specific language governing permissions and limitations␣
↪under the License.
 *
 *
 */

package eu.amidst.core.examples.conceptdrift;


import eu.amidst.core.conceptdrift.NaiveBayesVirtualConceptDriftDetector;
import eu.amidst.core.datastream.DataInstance;
import eu.amidst.core.datastream.DataOnMemory;
import eu.amidst.core.datastream.DataStream;
import eu.amidst.core.io.DataStreamLoader;
import eu.amidst.core.variables.Variable;

/**
 * This example shows how to use the class NaiveBayesVirtualConceptDriftDetector to␣
↪run the virtual concept drift
```

(continues on next page)

(continued from previous page)

```
 * detector detailed in
 *
 * <i>Borchani et al. Modeling concept drift: A probabilistic graphical model based␣
↪approach. IDA 2015.</i>
 *
 */
public class NaiveBayesVirtualConceptDriftDetectorExample {
    public static void main(String[] args) {

        //We can open the data stream using the static class DataStreamLoader
        DataStream<DataInstance> data = DataStreamLoader.open("./datasets/DriftSets/
↪sea.arff");

        //We create a NaiveBayesVirtualConceptDriftDetector object
        NaiveBayesVirtualConceptDriftDetector virtualDriftDetector = new␣
↪NaiveBayesVirtualConceptDriftDetector();

        //We set class variable as the last attribute
        virtualDriftDetector.setClassIndex(-1);

        //We set the data which is going to be used
        virtualDriftDetector.setData(data);

        //We fix the size of the window
        int windowSize = 1000;
        virtualDriftDetector.setWindowsSize(windowSize);

        //We fix the so-called transition variance
        virtualDriftDetector.setTransitionVariance(0.1);

        //We fix the number of global latent variables
        virtualDriftDetector.setNumberOfGlobalVars(1);

        //We should invoke this method before processing any data
        virtualDriftDetector.initLearning();

        //Some prints
        System.out.print("Batch");
        for (Variable hiddenVar : virtualDriftDetector.getHiddenVars()) {
            System.out.print("\t" + hiddenVar.getName());
        }
        System.out.println();


        //Then we show how we can perform the sequential processing of
        // data batches. They must be of the same value than the window
        // size parameter set above.
        int countBatch = 0;
        for (DataOnMemory<DataInstance> batch : data.iterableOverBatches(windowSize)){

            //We update the model by invoking this method. The output
            // is an array with a value associated
            // to each fo the global hidden variables
            double[] out = virtualDriftDetector.updateModel(batch);

            //We print the output
            System.out.print(countBatch + "\t");
```

(continues on next page)

(continued from previous page)

```
            for (int i = 0; i < out.length; i++) {
                System.out.print(out[i]+"\t");
            }
            System.out.println();
            countBatch++;
        }
    }
}
```

[Back to Top]

1.7.9 HuginLink

Models conversion between AMiDST and Hugin

This example shows how to use the class BNConverterToAMIDST and BNConverterToHugin to convert a Bayesian network models between Hugin and AMIDST formats

```
package eu.amidst.core.examples.huginlink;

import COM.hugin.HAPI.Domain;
import COM.hugin.HAPI.ExceptionHugin;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.huginlink.converters.BNConverterToAMIDST;
import eu.amidst.huginlink.converters.BNConverterToHugin;
import eu.amidst.huginlink.io.BNLoaderFromHugin;

/**
 * Created by rcabanas on 24/06/16.
 */
public class HuginConversionExample {
    public static void main(String[] args) throws ExceptionHugin {
        //We load from Hugin format
        Domain huginBN = BNLoaderFromHugin.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //Then, it is converted to AMIDST BayesianNetwork object
        BayesianNetwork amidstBN = BNConverterToAMIDST.convertToAmidst(huginBN);

        //Then, it is converted to Hugin Bayesian Network object
        huginBN = BNConverterToHugin.convertToHugin(amidstBN);

        System.out.println(amidstBN.toString());
        System.out.println(huginBN.toString());

    }
}
```

[Back to Top]

I/O of Bayesian Networks with Hugin net format

This example shows how to use the class BNLoaderFromHugin and BNWriterToHugin classes to load and write Bayesian networks in Hugin format

```
package eu.amidst.core.examples.huginlink;

import COM.hugin.HAPI.Domain;
import COM.hugin.HAPI.ExceptionHugin;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.huginlink.converters.BNConverterToAMIDST;
import eu.amidst.huginlink.io.BNLoaderFromHugin;
import eu.amidst.huginlink.io.BayesianNetworkWriterToHugin;

/**
 * Created by rcabanas on 24/06/16.
 */
public class HuginIOExample {
    public static void main(String[] args) throws ExceptionHugin {
        //We load from Hugin format
        Domain huginBN = BNLoaderFromHugin.loadFromFile("networks/asia.net");

        //We save a AMIDST BN to Hugin format
        BayesianNetwork amidstBN = BNConverterToAMIDST.convertToAmidst(huginBN);
        BayesianNetworkWriterToHugin.save(amidstBN,"networks/tmp.net");

    }
}
```

[Back to Top]

Invoking Hugin's inference engine

This example we show how to perform inference using Hugin inference engine within the AMiDST toolbox

```
package eu.amidst.core.examples.huginlink;

import eu.amidst.core.inference.InferenceAlgorithm;
import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.HashMapAssignment;
import eu.amidst.core.variables.Variable;
import eu.amidst.huginlink.inference.HuginInference;

import java.io.IOException;

/**
 * Created by rcabanas on 24/06/16.
 */
public class HuginInferenceExample {
    public static void main(String[] args) throws IOException, ClassNotFoundException
↪{
        //We first load the WasteIncinerator bayesian network
        //which has multinomial and Gaussian variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //We recover the relevant variables for this example:
        //Mout which is normally distributed, and W which is multinomial.
        Variable varMout = bn.getVariables().getVariableByName("Mout");
```

(continues on next page)

(continued from previous page)

```
        Variable varW = bn.getVariables().getVariableByName("W");

        //First we create an instance of a inference algorithm.
        //In this case, we use the ImportanceSampling class.
        InferenceAlgorithm inferenceAlgorithm = new HuginInference();

        //Then, we set the BN model
        inferenceAlgorithm.setModel(bn);

        //If exists, we also set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW, 0);
        inferenceAlgorithm.setEvidence(assignment);

        //Then we run inference
        inferenceAlgorithm.runInference();

        //Then we query the posterior of
        System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.
↪getPosterior(varMout));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<3.5 | W=0) = "
                + inferenceAlgorithm.getExpectedValue(varMout, v -> (0.7 < v && v < 3.
↪5) ? 1.0 : 0.0));

    }
}
```

[Back to Top]

Invoking Hugin's Parallel TAN

This example we show how to perform inference using Hugin inference engine within the AMIDST toolbox.

This example shows how to use Hugin's functionality to learn in parallel a TAN model. An important remark is that Hugin only allows to learn the TAN model for a data set completely loaded into RAM memory. The case where our data set does not fit into memory, it solved in AMIDST in the following way. We learn the structure using a smaller data set produced by Reservoir sampling and, then, we use AMIDST's ParallelMaximumLikelihood to learn the parameters of the TAN model over the whole data set.

For further details about the implementation of the parallel TAN algorithm look at the following paper:

> Madsen, A.L. et al. A New Method for Vertical Parallelisation of TAN Learning Based on Balanced Incomplete Block Designs. Probabilistic Graphical Models. Lecture Notes in Computer Science Volume 8754, 2014, pp 302-317.

```
package eu.amidst.core.examples.huginlink;

import eu.amidst.core.inference.InferenceAlgorithm;
import eu.amidst.core.io.BayesianNetworkLoader;
import eu.amidst.core.models.BayesianNetwork;
import eu.amidst.core.variables.Assignment;
import eu.amidst.core.variables.HashMapAssignment;
import eu.amidst.core.variables.Variable;
import eu.amidst.huginlink.inference.HuginInference;
```

(continues on next page)

(continued from previous page)

```
import java.io.IOException;

/**
 * Created by rcabanas on 24/06/16.
 */
public class HuginInferenceExample {
    public static void main(String[] args) throws IOException, ClassNotFoundException
↪{
        //We first load the WasteIncinerator bayesian network
        //which has multinomial and Gaussian variables.
        BayesianNetwork bn = BayesianNetworkLoader.loadFromFile("./networks/simulated/
↪WasteIncinerator.bn");

        //We recover the relevant variables for this example:
        //Mout which is normally distributed, and W which is multinomial.
        Variable varMout = bn.getVariables().getVariableByName("Mout");
        Variable varW = bn.getVariables().getVariableByName("W");

        //First we create an instance of a inference algorithm.
        //In this case, we use the ImportanceSampling class.
        InferenceAlgorithm inferenceAlgorithm = new HuginInference();

        //Then, we set the BN model
        inferenceAlgorithm.setModel(bn);

        //If exists, we also set the evidence.
        Assignment assignment = new HashMapAssignment(1);
        assignment.setValue(varW, 0);
        inferenceAlgorithm.setEvidence(assignment);

        //Then we run inference
        inferenceAlgorithm.runInference();

        //Then we query the posterior of
        System.out.println("P(Mout|W=0) = " + inferenceAlgorithm.
↪getPosterior(varMout));

        //Or some more refined queries
        System.out.println("P(0.7<Mout<3.5 | W=0) = "
                + inferenceAlgorithm.getExpectedValue(varMout, v -> (0.7 < v && v < 3.
↪5) ? 1.0 : 0.0));

    }
}
```

[Back to Top]

1.7.10 MoaLink

AMIDST Classifiers from MOA

The following command can be used to learn a Bayesian model with a latent Gaussian variable (HG) and a multinomial with 2 states (HM), as displayed in figure below. The VMP algorithm is used to learn the parameters of these two non-observed variables and make predictions over the class variable.

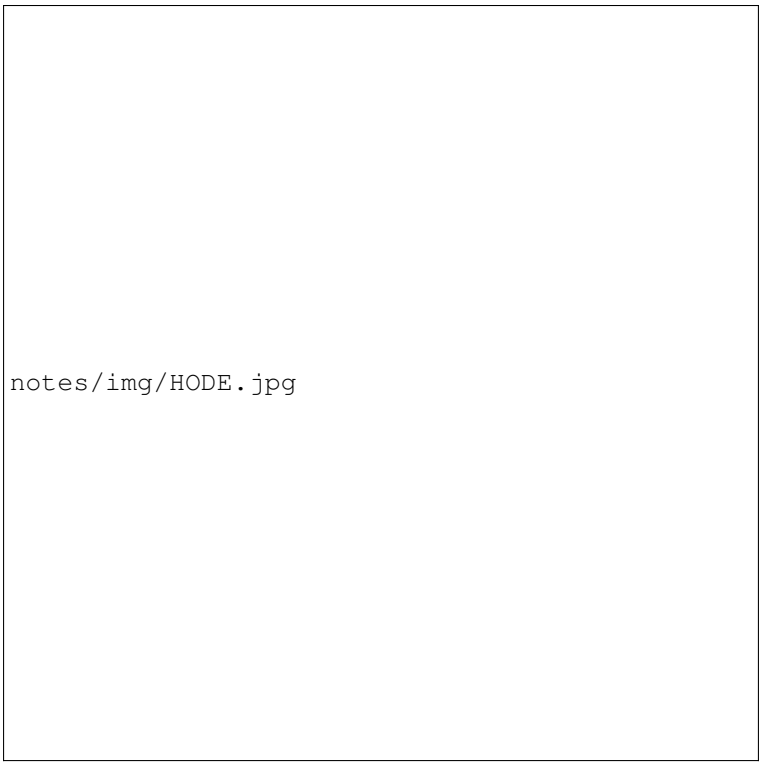


Fig. 10: HODE example

```
java -Xmx512m -cp "../lib/*" -javaagent:../lib/sizeofag-1.0.0.jar
moa.DoTask EvaluatePrequential -l \(bayes.AmidstClassifier -g 1
-m 2\) -s generators.RandomRBFGenerator -i 10000 -f 1000 -q 1000
```

[Back to Top]

AMIDST Classifiers from MOA

It is possible to learn an enriched naive Bayes model for regression if the class label is of a continuous nature. The following command uses the model in Figure *2* on a toy dataset from WEKA's collection of regression problems.

```
java -Xmx512m -cp "../lib/*" -javaagent:../lib/sizeofag-1.0.0.jar
moa.DoTask EvaluatePrequentialRegression -l bayes.AmidstRegressor
-s (ArffFileStream -f ./quake.arff)
```

Note that the simpler the dataset the less complex the model should be. In this case, `quake.arff` is a very simple and small dataset that should probably be learn with a more simple classifier, that is, a high-bias-low-variance classifier, in order to avoid overfitting. This aims at providing a simple running example.

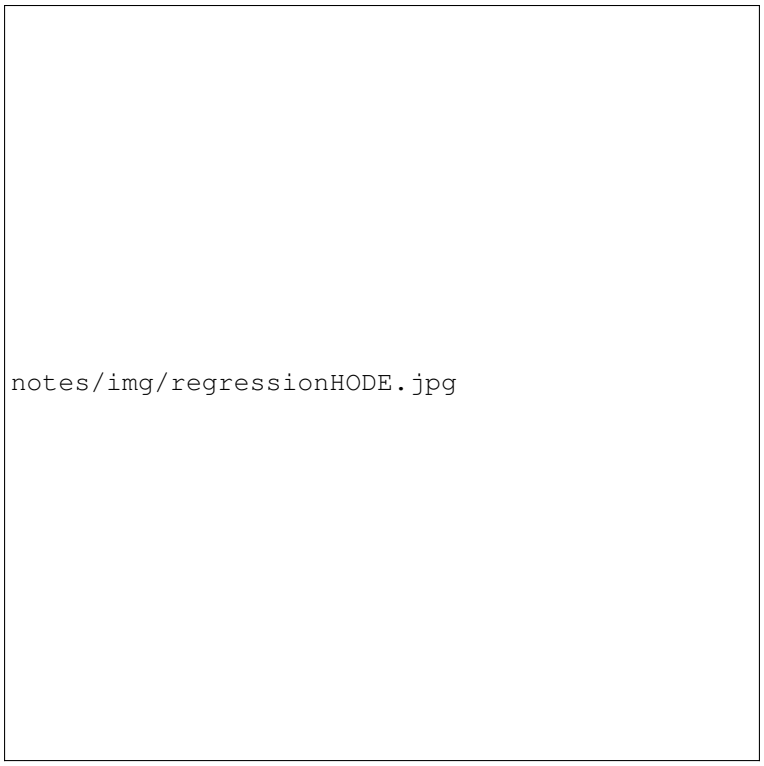


Fig. 11: HODE regression example

1.8 Getting Started!

1.8.1 Quick start

Here we explain how to download and run an example project that uses the AMIDST functionality. You will need to have *java 8, mvn* and *git* installed. For more information, read the requirements section. First, download the example project code:

```
$ git clone https://github.com/amidst/example-project.git
```

Enter in the downloaded folder:

```
$ cd example-project/
```

A code example illustrating the use of the toolbox is provided in the file **./src/main/java/BasicExample.java**.

Compile and build the package:

```
$ mvn clean package
```

Finally, run the code example previously mentioned:

```
$ java -cp target/example-project-full.jar BasicExample
```

Each time that our model is updated, the following output is shown:

```
Processing batch 1:
    Total instance count: 1000
    N Iter: 2, elbo:2781.727395615198
Processing batch 2:
    Total instance count: 2000
    N Iter: 2, elbo:2763.7884038634625

. . .

Processing batch 89:
    Total instance count: 88565
    N Iter: 2, elbo:1524.8632699545146

Bayesian Network:
P(codrna_X1 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = 2.5153648505301884, beta1 = -6.47078042377021, var = 0.012038840802392285 ]
↪| {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X2 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = -1.4100844769398433, beta1 = 6.449118564273272, var = 0.09018732085219959 ]
↪| {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X3 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = 0.5004820734231348, beta1 = -0.7233270338873005, var = 0.02287282091577493
↪] | {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X4 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = -3.727658229972866, beta1 = 15.332997451530298, var = 0.035794031399428765
↪] | {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X5 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = -1.3370521440370204, beta1 = 7.394413026859823, var = 0.028236889224165312
↪] | {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X6 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = -3.3189931551027154, beta1 = 13.565377369009742, var = 0.007243019620713637
↪] | {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X7 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = -1.3216192169520564, beta1 = 6.327466251964861, var = 0.01677087665403506 ]
↪| {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_X8 | M, Z0) follows a Normal|Multinomial,Normal
[ alpha = 2.235639811622681, beta1 = -5.927480690695894, var = 0.015383139745907676 ]
↪| {M = 0}
[ alpha = 0.0, beta1 = 0.0, var = 1.0 ] | {M = 1}

P(codrna_Y) follows a Multinomial
[ 0.3333346978625332, 0.6666653021374668 ]
P(M | codrna_Y) follows a Multinomial|Multinomial
```

(continues on next page)

(continued from previous page)

```
[ 0.9999877194382871, 1.2280561712892748E-5 ] | {codrna_Y = 0}
[ 0.9999938596437365, 6.1403562634704065E-6 ] | {codrna_Y = 1}
P(Z0 | codrna_Y) follows a Normal|Multinomial
Normal [ mu = 0.2687114577360176, var = 6.897846922968294E-5 ] | {codrna_Y = 0}
Normal [ mu = 0.2674517087293682, var = 5.872354808764403E-5 ] | {codrna_Y = 1}


P(codrna_Y|codrna_X1=0.7) = [ 0.49982925627218583, 0.5001707437278141 ]
```

The output shows: the current batch number; the total number of instances that has been processed until now; the required number of iterations for learning from the current batch; and the *elbo (evidence lower bound)*. Finally, distributions in the learnt Bayesian network are given.

In general, for start using the AMIDST toolbox, add the following lines to the pom.xml file of your maven project:

```
<repositories>
  <repositories>
    <repository>
  <id>amidstRepo</id>
  <url>https://raw.github.com/amidst/toolbox/mvn-repo/</url>
  </repository>
</repositories>

<dependencies>
  <dependency>
    <groupId>eu.amidst</groupId>
    <artifactId>module-all</artifactId>
    <version> 0.7.2 </version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

1.8.2 Getting started in detail

Before starting using the AMDIST, you might check that your system fits the requirements of the toolbox.

Toolbox users (i.e. those interested in simply using the functionality provided by AMIDST) might find useful the following tutorials:

- Loading AMIDST dependencies from a remote maven repository.
- Installing a local AMIDST repository
- Generating the packages for each module and its dependencies (command line).

Additionally, for those developers interested in colaborating to AMIDST toolbox could read the following tutorials:

- Basic steps for contributing

1.9 Requirements for AMIDST Toolbox

1.9.1 For toolbox users

This toolbox has been specifically designed for using the functional-style features provided by the Java 8 release. You can check the Java version installed in your system with the following command:

```
$ java -version
```

If everything is right, the following output (or similar) will be generated:

```
java version "1.8.0_73"
Java(TM) SE Runtime Environment (build 1.8.0_73-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.73-b02, mixed mode)
```

Make sure that the Java version is 1.8 or higher. Otherwise you must install a more recent release of Java.

The second requirement consists of having maven installed your system: http://maven.apache.org/download.cgi (follow specific instructions for your OS). Use the following command for verifying that everything works properly:

```
$ mvn -v
```

which should generate an output similar to

```
Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8fdf4; 2014-08-
↪11T22:58:10+02:00)
Maven home: /sw/share/maven
Java version: 1.8.0_73, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.11.3", arch: "x86_64", family: "mac"
```

Having git installed is advisable for downloading relevant material (i.e., source code, example project, datasets, etc.) Further information can be found here.

1.9.2 For AMIDST developers

AMIDST toolbox is hosted on GitHub. To work with AMIDST code you should follow the Fork & Pull collaboration model. Read this guide for full details about how to fork a project and make a pull request. Once you have forked the project, you can clone to you computer and open it with Intellij by pointing at the pom.xml file, and everything will be ready to start. Further details about how to contribute to this project are given this link.

1.10 Loading AMIDST dependencies from a remote maven repository

Here we explain how to add the AMIDST dependencies in a maven project with Java 8 or higher. Alternatively, you might prefer following the video-tutorial in this link .

In this example, we will use a project containing only one class, though the procedure here explain could be used in any other maven project. You can check this link for getting more information about how to create a new mavenproject.

For using the AMIDST Toolbox, the **pom.xlm** file will be modified. First, in the Project view (located on the left) select the file pom.xml of your project and open it:

Add the AMIDST repository by including the following code to your pom:

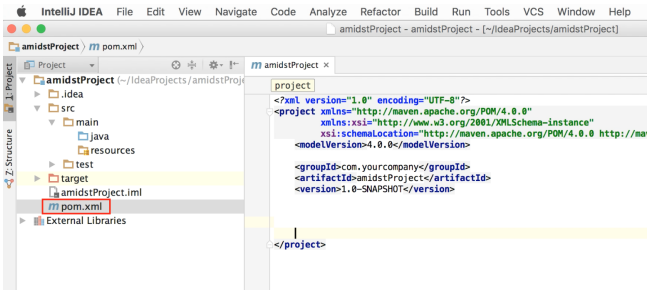


Fig. 12: Capture of Maven project in IntelliJ and the initial pom.xml file

```
<repositories>
<!-- AMIDST repository in github -->
<repository>
<id>amidstRepo</id> <!-- local identifier, it can be anything -->
<url>https://raw.github.com/amidst/toolbox/mvn-repo/</url>
</repository>
<!-- ... -->
</repositories>
```

Then, add the dependencies of modules in AMIDST you want to use. For each module, add an element **<dependency>...</dependency>** inside the labels **<dependencies></dependencies>**. For each one, we have to indicate the following information:

- **groupId** is an identifier of the project's module. In this case it should containt the value *"eu.amidst"*.
- **artifactId** is the name of the module we want to use. More precisely, it is the name of the jar file containing such module. You can see the list of AMIDST modules here.
- **version** is the identifier of AMIDST Toolbox release. You can see here the list of all versions available.
- **scope** allows you to only include dependencies appropriate for the current stage of the build. We will set this to *"compile"*.

For example, for using the *core-dynamic* module, include the following code:

```
<dependencies>
<!-- Load any of the modules from AMIDST Toolbox -->
<dependency>
<groupId>eu.amidst</groupId>
<artifactId>core-dynamic</artifactId>
<version> 0.7.2 </version>
<scope>compile</scope>
</dependency>

<!-- ... -->
</dependencies>
```

Note that for using another module, simply change the value of the element artifactId (i.e. the content between the tags <artifactId> and <artifactId>). Now you can check in the **Maven Projects panel** that all the dependencies have been loaded:

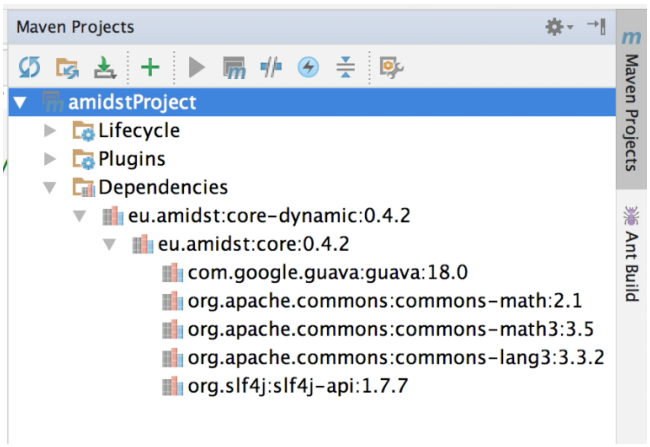


Fig. 13: List of loaded dependencies

Note that the *core-dynamic module* depends on corethat has been loaded as well. We recomend you to download the sources and the javadoc:

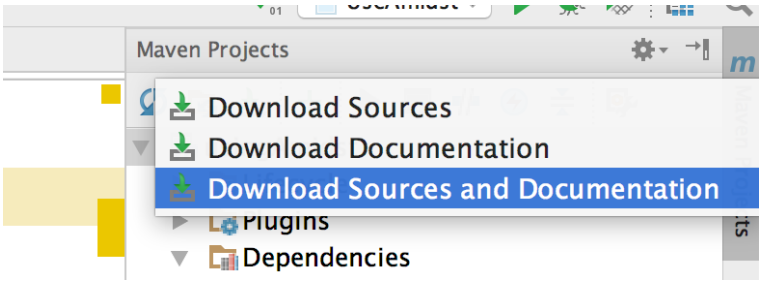


Fig. 14: Download the JavaDoc and the source code

Finally, for testing purposes, we can run the code shown below that generates a random dynamic bayesian network (DBN) and prints its parameters.

```
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;

public class TestingAmidst {
  public static void main(String[] args) throws WrongConfigurationException {
    DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
    DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
    DynamicBayesianNetworkGenerator.setNumberOfStates(3);

    DynamicBayesianNetwork extendedDBN =
    DynamicBayesianNetworkGenerator.generateDynamicBayesianNetwork();

    System.out.println(extendedDBN.toString());


  }

}
```

If everything goes right, the following output will be generated:

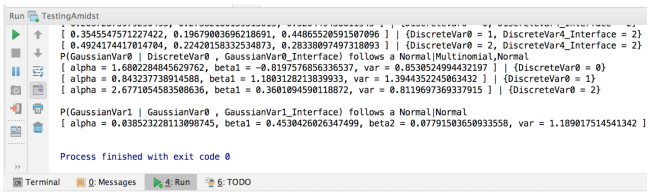


Fig. 15: Ouput of the testing code that generates a random DBN

1.11 Installing a local AMIDST repository

Here we explain how to install a local AMIDST repository and how to add them in a maven project with Java 8 or higher. Alternatively, you can follow this video tutorial. In summary, we will download the source and use the appropriate maven commands for installing it.

First, you can download the source code from the github repository using the following command:

```
$ git clone https://github.com/amidst/toolbox.git
```

Depending on your internet connection, the process may take several minutes. Once the it has finished, enter into the downloaded folder:

```
$ cd toolbox
```

Now, we should install the AMIDST artifact in our local Maven repository (this repository is automatically created when installing Maven). For that, type the following command:

```
$ mvn clean install -Dmaven.test.skip=true
```

Note that you can avoid the argument **-Dmaven.test.skip=true**. In that case, the process will not skip the unitary tests and hence the installation process will take much longer (its default value is false). Once the process has finished, an output similar to the following one will be generated:

```
[INFO] Reactor Summary:
[INFO]
[INFO] AmidstToolbox ..................................... SUCCESS [  0.348 s]
[INFO] core .............................................. SUCCESS [ 12.300 s]
[INFO] core-dynamic ...................................... SUCCESS [  5.352 s]
[INFO] huginlink ......................................... SUCCESS [  3.255 s]
[INFO] standardmodels .................................... SUCCESS [  3.128 s]
[INFO] examples .......................................... SUCCESS [  4.530 s]
[INFO] moalink ........................................... SUCCESS [  3.944 s]
[INFO] wekalink .......................................... SUCCESS [  2.388 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 35.681 s
[INFO] Finished at: 2016-05-10T15:58:14+02:00
[INFO] Final Memory: 63M/539M
[INFO] ------------------------------------------------------------------------
```

Now we can check that the libraries has been placed into the local maven repository, which is usually placed in **~/.m2/repository/**. Thus, type the following command:

```
$ ls ~/.m2/repository/eu/amidst/
```

And you will find a folder for each module:

```
AmidstToolbox     core-dynamic     huginlink     standardmodels
core          examples     moalink      wekalink
```

Now we will see how can we use this local repository from a maven project (using IntelliJ IDEA). In this example, we will use a project containing only one class, though the procedure here explain could be used in any other maven project. You can check this link for getting more information about how to create a new mavenproject.

For using the AMIDST Toolbox, the **pom.xlm** file will be modified. First, in the Project view (located on the left) select the file pom.xml of your project and open it:

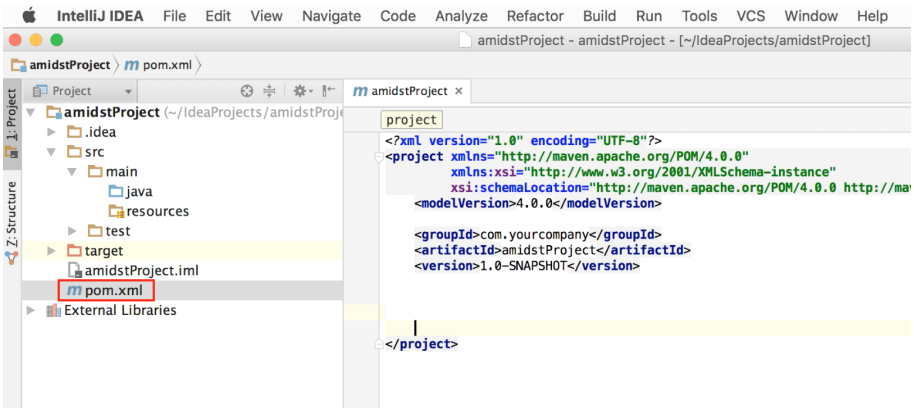


Fig. 16: Initial view of an empty maven project in IntelliJ IDEA

In the file pom.xml, add the dependencies to the modules in AMIDST you want to use. For each module, add an element **<dependency>...</dependency>** inside the labels **<dependencies></dependencies>**. For each one, we have to indicate the following information:

- groupId is an identifier of the project's module. In this case it should containt the value *"eu.amidst"*.
- artifactIdis the name of the module we want to use. More precisely, it is the name of the jar file containing such module. You can see the list of AMIDST modules here.
- **version** is the identifier of AMIDST Toolbox release. You can see here the list of all versions available.
- **scope** allows you to only include dependencies appropriate for the current stage of the build. We will set this to *"compile"*.

For example, for using the core-dynamicmodule, include the following code:

```
<dependencies>
<!-- Load any of the modules from AMIDST Toolbox -->
<dependency>
<groupId>eu.amidst</groupId>
<artifactId>core-dynamic</artifactId>
<version> 0.7.2 </version>
<scope>compile</scope>
</dependency>
```

```
<!-- ... -->
</dependencies>
```

Note that for using another module, simply change the value of the element artifactId(i.e. the content between the tags <artifactId> and <artifactId>). Now you can check in the **Maven Projects panel** that all the dependencies have been loaded:

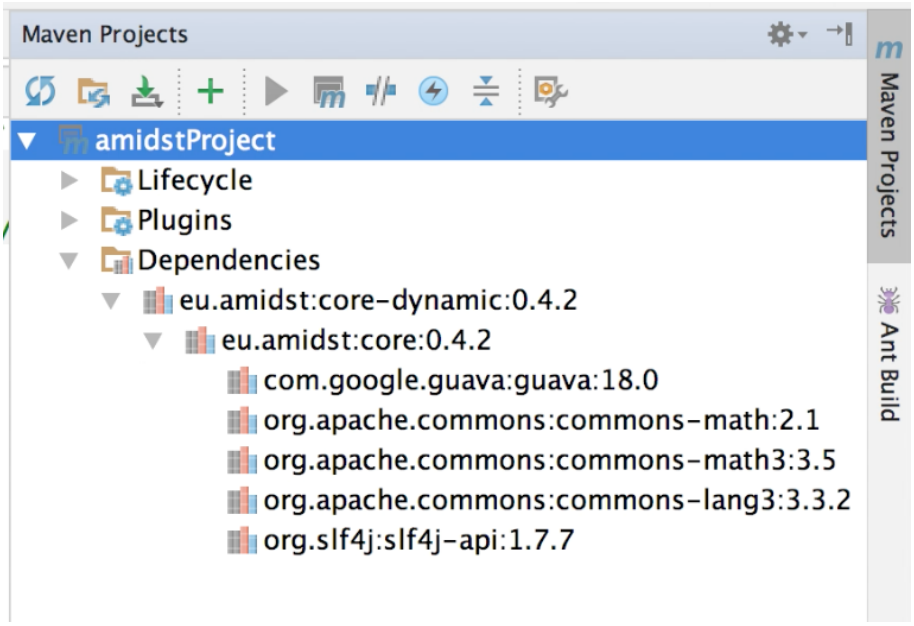


Fig. 17: Loaded dependencies

Note that the *core-dynamic module* depends on corethat has been loaded as well. We recomend you to download the sources and the javadoc as shown below.

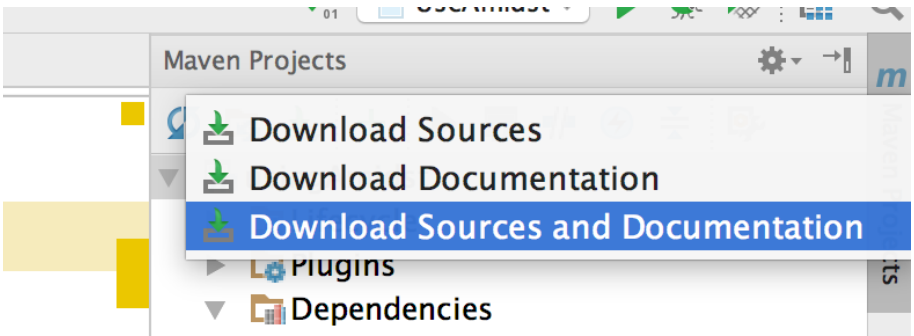


Fig. 18: Downloading JavaDoc and source code

Finally, for testing purposes, we can run the following code:

```
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;

public class TestingAmidst {
 public static void main(String[] args) throws WrongConfigurationException {
  DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
  DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
  DynamicBayesianNetworkGenerator.setNumberOfStates(3);

  DynamicBayesianNetwork extendedDBN =
  DynamicBayesianNetworkGenerator.generateDynamicBayesianNetwork();

  System.out.println(extendedDBN.toString());


 }

}
```

If everything goes right, the following output will be generated:

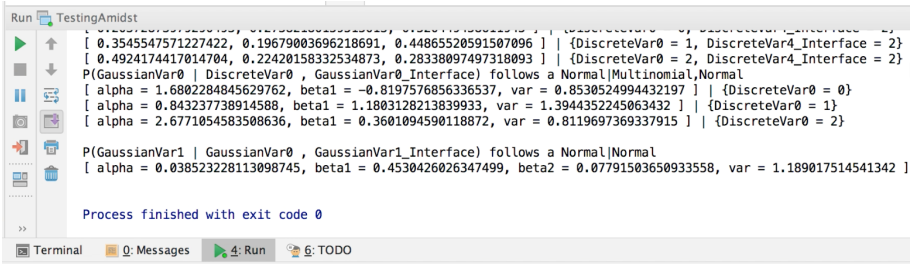


Fig. 19: Output generated when running the example code

1.12 Generating the packages for each module and for its dependencies

Here we explain how can we generate the packages (i.e. jar files) for each module and also those corresponding to the dependencies. This will allow to use AMIDST toolbox in any java project. Alternatively, you can watch this video-tutorial. First, you can download the source code using the following command:

```
$ git clone https://github.com/amidst/toolbox.git
```

Once the download has finished, enter into the downloaded folder:

```
$ cd toolbox
```

For generating the packages for all the modules, use the following command:

```
$ mvn clean dependency:copy-dependencies package -Dmaven.test.skip=true
```

Note that the argument **-Dmaven.test.skip=true** is optional (its default value is false). If omitted, maven will run the unitary tests and hence the process will take much more time. If everything goes right, you will eventually obtain an output similar to the following one:

```
[INFO] Reactor Summary:
[INFO]
[INFO] AmidstToolbox ..................................... SUCCESS [  1.177 s]
[INFO] core .............................................. SUCCESS [ 11.891 s]
[INFO] core-dynamic ...................................... SUCCESS [  5.081 s]
[INFO] huginlink ......................................... SUCCESS [  3.031 s]
[INFO] standardmodels .................................... SUCCESS [  2.976 s]
[INFO] examples .......................................... SUCCESS [  4.563 s]
[INFO] moalink ........................................... SUCCESS [  2.609 s]
[INFO] wekalink .......................................... SUCCESS [  2.515 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 34.504 s
[INFO] Finished at: 2016-05-10T16:41:23+02:00
[INFO] Final Memory: 68M/670M
[INFO] ------------------------------------------------------------------------
```

Now, at each module folder, a new folder called **target** containing the generated packages has been created. In addition the sub-folder target/dependencycontains the dependencies for the given package. Now simply copy all these packages into a folder of your classpath and you will be able to use the functionality provided by AMIDST.

1.13 Basic steps for contributing

Here we will explain the basic github commands for modifying the AMIDST source code and for uploading such changes to the server. Alternatively, you can watch this video-tutorial. The general scheme to follow for contributing to the AMIDST source code is shown below:

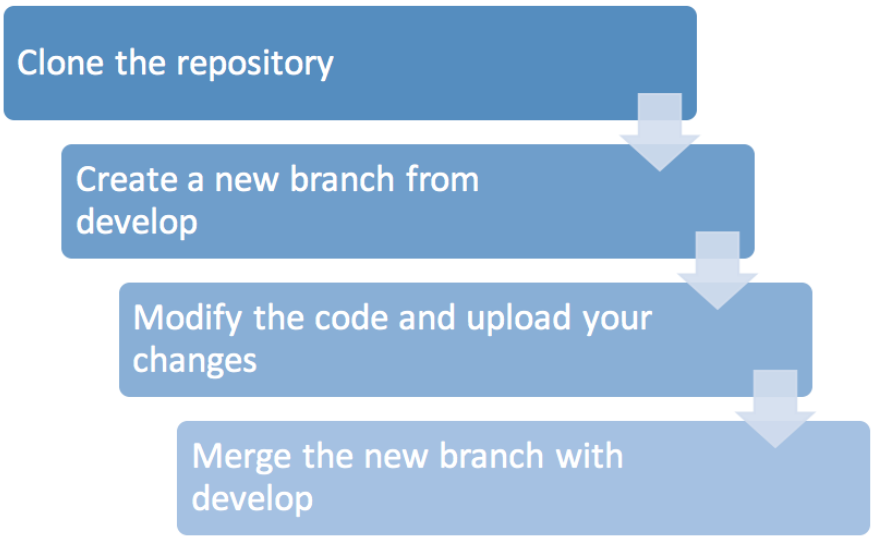


Fig. 20: Scheme for contributing to AMIDST toolbox.

Now we will explain each of these steps and the involved commands.

1.13.1 Clone the repository

First, you can download the source code from github using the following command:

```
$ git clone https://github.com/amidst/toolbox.git
```

Once the download has finished, enter into the downloaded folder:

```
$ cd toolbox
```

1.13.2 Create a new branch from develop

All the new development will be done in the branch develop. **Do not modify the master branch** because it should always contain the source of the very last release. Thus we change into the develop branch with the following command:

```
$ git checkout develop
```

The terminal replies with the following output:

```
Checking out files: 100% (469/469), done.
Branch develop set up to track remote branch develop from origin.
Switched to a new branch 'develop'
```

To avoid overlapping, we advise you to perform all your developments or changes in a new branch created from develop. The name of this new branch will be newfeature and it can be created with the following command

```
$ git branch newfeature
```

We move into the recently created branch:

```
$ git checkout newfeature
```

```
Switched to branch 'newfeature'
```

At any moment, we can verify which is the current branch with the command show below. Make sure that your current branch is always the one you have created.

```
$ git branch
```

Previous command shows a list with all the local branches, being the current branch the one with the symbol *, for example:

```
develop
master
* newfeature
```

The new branch is not yet a remote branch. For uploading it to the server, we will use the following command:

```
$ git push --set-upstream origin newfeature
```

```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/amidst/toolbox.git
* [new branch]      newfeature -> newfeature
Branch newfeature set up to track remote branch newfeature from origin.
```

1.13.3 Modify the code and upload your changes

As an example, we will simply create a new text file and upload it to the server. For creating such file run:

```
$ echo "file to be deleted" > newfile.txt
```

Now, we have to set the new file as a tracked file, for that purpose:

```
$ git add newfile.txt
```

Which generates the output shown below indicating which of the tracked files contain changes to be committed

```
$ git status
```

that generates the following output:

```
On branch newfeature
Your branch is up-to-date with 'origin/newfeature'.
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

new file:   newfile.txt

Untracked files:
(use "git add <file>..." to include in what will be committed)

...
```

Now we will do a commit including the message “added newfile.txt” with the following command:

```
$ git commit -m "added newfile.txt"
```

```
[newfeature f256d1e] added newfile.txt
1 file changed, 1 insertion(+)
create mode 100644 newfile.txt
```

Finally, we have upload all the changes to the server:

```
$ git push
```

1.13.4 Merge the new branch with develop

Until now, the changes done are only present in the branch newfeature. However, we should integrate these changes with the develop branch. Thus, we first change to the branch develop:

```
$ git checkout develop
```

Now, we will merge both branches:

```
$ git merge newfeature
```

If there is not conflicts, an output similar to the following one will be generated:

```
Updating 20ff914..f256d1e
Fast-forward
newfile.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 newfile.txt
```

Finally, we will upload the result of merging both branches to the server:

```
$ git push
```

Now, if we go to the AMIDST github website (https://github.com/amidst/toolbox), we can verify that the branch develop contains the changes in the code:

Fig. 21: View of the result of the contribution in the github website.