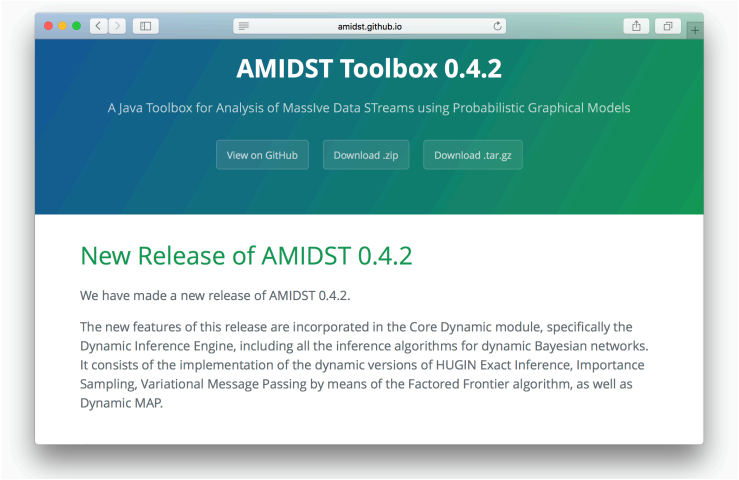
AMiDST

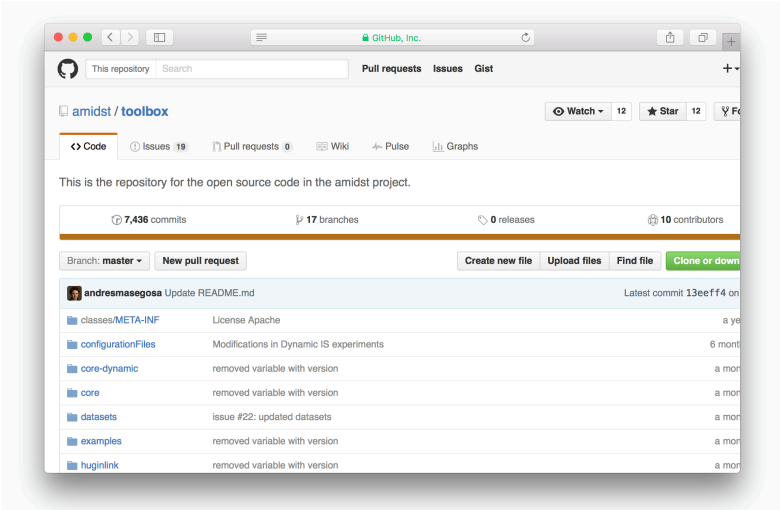
Analysis of Massive Data STreams

Tutorial: initial steps for setting up AMIDST toolbox and using it

June 3 2016

Important URLs





http://amidst.github.io/toolbox/
(Documentation, tutorials and more)

https://github.com/amidst/toolbox
(source code)

http://amidst.github.io/toolbox/tutorial-huginsa.zip
(material for this tutorial)

System Requirements

- Check your java version:

```
$ java -version
```

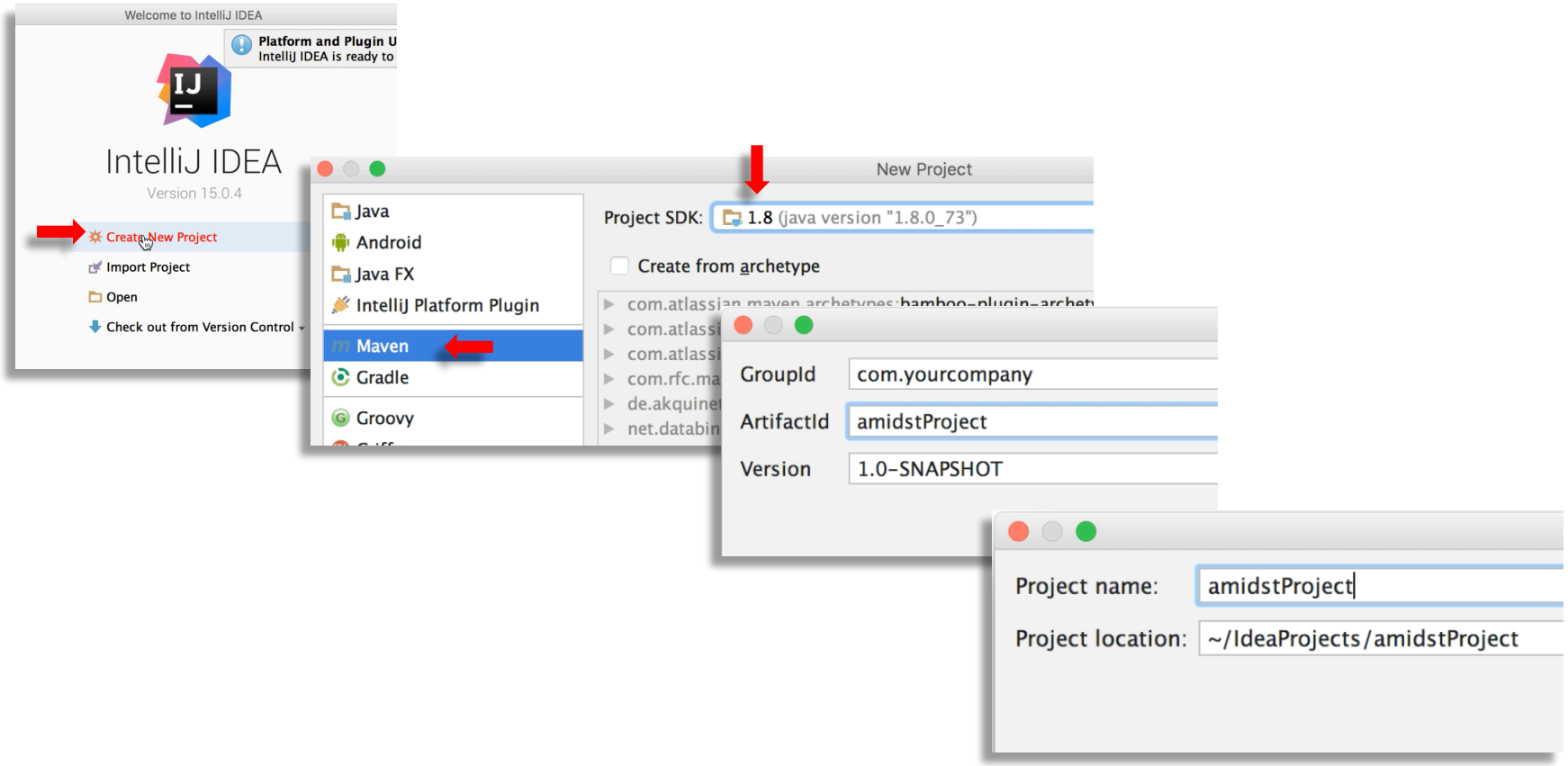
- http://www.oracle.com/technetwork/java/javase/downloads/

IntelliJ IDEA

- https://www.jetbrains.com/idea/

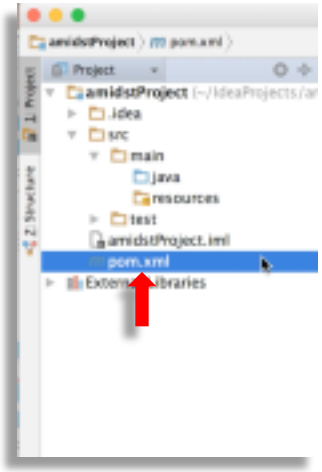
Setting up

- **Step 1**: Create an empty maven project



Setting up

- **Step 2**: Add repository and dependencies in the file pom.xml



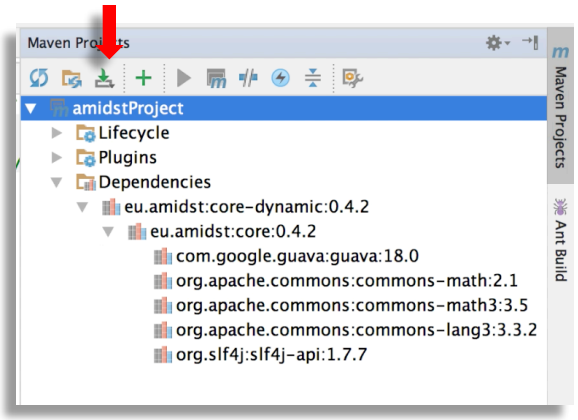
repository

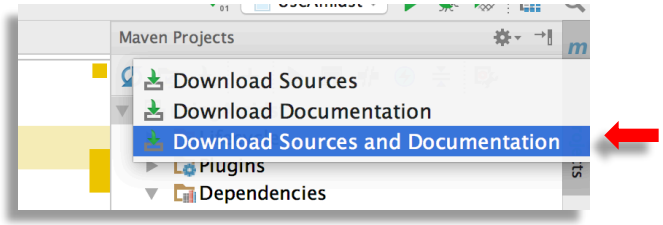
modules

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>eu.amidst.testrepo</groupId>
    <artifactId>testgithubmavenrepo</artifactId>
    <version>1.0-SNAPSHOT</version>

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

    <dependencies>
        <!-- Load one of the modules from AMIDST Toolbox -->
        <dependency>
            <groupId>eu.amidst</groupId>
            <artifactId>latent-variable-models</artifactId>
            <version>0.4.3-alpha</version>
            <scope>compile</scope>
        </dependency>
        <!-- ... -->
    </dependencies>
</project>
```

Setting up

- **Step 3**: Download source code and javadoc





ARFF format

- The toolbox can read datasets saved as .arff (Attribute-Relation File Format) files.
- dynamicDS_d2_c3.arff:

```
@relation dataset

@attribute SEQUENCE_ID real
@attribute TIME_ID real
@attribute DiscreteVar0 {0.0, 1.0}
@attribute DiscreteVar1 {0.0, 1.0}
@attribute GaussianVar0 real
@attribute GaussianVar1 real
@attribute GaussianVar2 real

@data
0,0,0.0,1.0,-11.218928287639379,-9.705288659121928,12.064058511499582
0,1,0.0,0.0,-1.5520768000994851,-4.280081986112549,21.056630499175807
0,2,0.0,0.0,-0.6333905987574826,-2.4527733264023213,37.116863464076864
0,3,0.0,0.0,-1.3737198121876433,-0.23170101600416873,63.662971350021095
. . .
```

- Open dynamicDS_d0_c5.arff and dynamicDS_d5_c0.arff

Static models (learning)

```
public class StaticModelLearning {
   public static void main(String[] args) {

      //Load the datastream
      String filename = "datasets/simulated/exampleDS_d0_c5.arff";
      DataStream<DataInstance> data = DataStreamLoader.openFromFile(filename);

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

   }

}
```

StaticModelLearning.java

Static models (learning from flink)

```
public class StaticModelFlink {
   public static void main(String[] args) throws FileNotFoundException {
      //Load the datastream
      String filename = "datasets/simulated/exampleDS_d0_c5.arff";
      final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
      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);
   }
}
```

StaticModelFlink.java

Static models (save to disk)

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

// Save with hugin format
BNWriterToHugin.saveToHuginFile(bn, "networks/simulated/exampleBN.net");
```

- Note: make sure that you have the following files in your classpath:
  - hgapi83_amidst-64.jar
  - libhgapi83_amidst-64.jnilib
- For adding folders to your class path:

```
-Djava.library.path="…."
```

StaticModelSaveToDisk.java

Static models (inference)

- Add the following code after learning the model

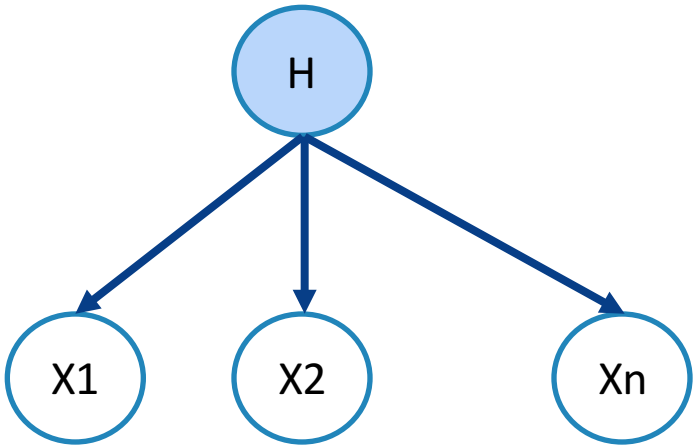
```
//Variabeles of interest
Variable varTarget = bn.getVariables().getVariableByName("LatentVar2");
Variable varObserved = null;

//we set the evidence
Assignment assignment = new HashMapAssignment(2);
varObserved = bn.getVariables().getVariableByName("GaussianVar1");
assignment.setValue(varObserved,6.5);

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

//query
infer.runInference();
Distribution p = infer.getPosterior(varTarget);
System.out.println("P(LatentVar2|GaussianVar1=6.5) = "+p);
```

new HuginInference();
new ImportanceSampling();

StaticModelInference.java

Static models (practice)

- Create your custom model: gaussian mixture



Discrete hidden variable

Continuous variables

- Assume that observed variables are not connected

Static models (practice)

- Some tips:

```
public class CustomGaussianMixture  extends Model{

   public CustomGaussianMixture(Attributes  attributes) throws WrongConfigurationException {
      super(attributes);
      //TODO: Write the contructor code here
   }

   @Override
   protected void buildDAG() {
      //TODO: Write the code building  a DAG for your custom model
   }
}
```

- Useful methods:

```
public Variable Variables::newMultinomialVariable(String  name, int nOfStates)

public List<ParentSet> DAG::getParentSets()
```

Dynamic models (learning)

```
public class DynamicModelLearning {
    public static void main(String[] args) {

        //Load the datastream
        String filename = "datasets/simulated/exampleDS_d0_c5.arff";
        DataStream<DynamicDataInstance> data = DynamicDataStreamLoader.loadFromFile(filename);

        //Learn the model
        DynamicModel model = new HiddenMarkovModel(data.getAttributes());
        ((HiddenMarkovModel)model).setNumStatesHiddenVar(4);
        model.setWindowSize(200);
        model.updateModel(data);
        DynamicBayesianNetwork dbn = model.getModel();


        System.out.println(dbn);

    }
}
```

DynamicModelLearning.java

Dynamic models (save to disk)

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

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

- Note: make sure that you have the following files in your classpath:
  - hgapi83_amidst-64.jar
  - libhgapi83_amidst-64.jnilib
- For adding folders to your class path:

```
-Djava.library.path="…."
```

DynamicModelSaveToDisk.java

Dynamic models (inference)

- Add the following code after learning the process

```
//Testing dataset
String filenamePredict = "datasets/simulated/exampleDS_d0_c5_small.arff";
DataStream<DynamicDataInstance> dataPredict = DynamicDataStreamLoader.loadFromFile(filenamePredict);
//Select the inference algorithm
InferenceAlgorithmForDBN infer = new FactoredFrontierForDBN(new VMP());
infer.setModel(dbn);
Variable varTarget = dbn.getDynamicVariables().getVariableByName("discreteHiddenVar");
UnivariateDistribution posterior = null;

//Classify each instance
int t = 0;
for (DynamicDataInstance instance : dataPredict) {

   infer.addDynamicEvidence(instance);
   infer.runInference();
   posterior = infer.getFilteredPosterior(varTarget);
   System.out.println("t="+t+", P(discreteHiddenVar | Evidence) = " + posterior);

}
```

new HuginInference();
new ImportanceSampling();

DynamicModelInference.java

Dynamic models (inference)

- For predicting 5 steps ahead, replace:

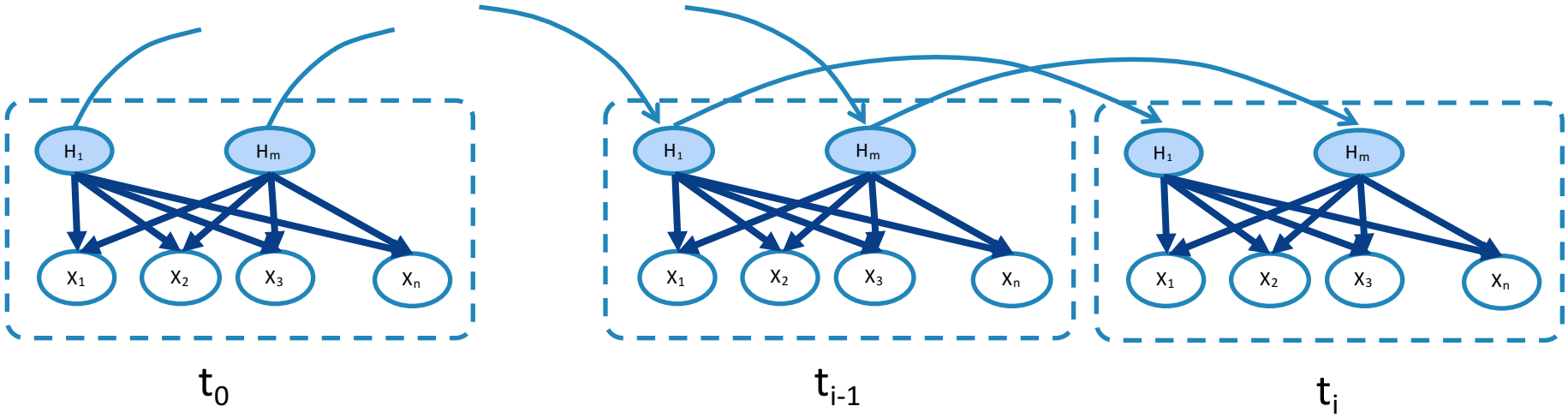
```
posterior = infer.getFilteredPosterior(varTarget);
```

```
posterior = infer.getPredictivePosterior(varTarget, 5);
```

DynamicModelInference.java

Dynamic models (practice)

- Create your custom dynamic model: Kalman filter



- Assume that hidden variables are not connected among them
- All the variabels are continuous

Dynamic models (practice)

- Some tips:

```
public class CustomKalmanFilter extends DynamicModel {

  public CustomKalmanFilter(Attributes attributes) throws WrongConfigurationException {
    super(attributes);
    //TODO: Write the constructor code here
  }
  @Override
  protected void buildDAG() {

  }
}
```

- Useful methods:

```
public DynamicVariable DynamicVariables::newGaussianDynamicVariable (String name)

public Variable DynamicVariable ::getInterfaceVariable()

public ParentSet DynamicDAG::getParentSetTimeT(Variable var)
```

Parallel TAN (Hugin/AMIDST)

- **Hugin:** learn the structure with a subsample of the data
- **AMIDST:** learn the parameters in AMIDST using the whole data.

```
ParallelTAN tan = new ParallelTAN();
tan.setNumCores(4);
tan.setNumSamplesOnMemory(1000);
tan.setNameRoot(var01);
tan.setNameTarget(classVar);
BayesianNetwork model  = tan.learn(data);
```