2016 IEEE 16th International Conference on Data Mining Workshops

Financial Data Analysis with PGMs using AMIDST

Rafael Cabañas* Aalborg University Email: rcabanas@cs.aau.dk

Ana M. Martínez* Aalborg University Email: ana@cs.aau.dk

Andrés R. Masegosa* Norwegian University of Science and Technology Email: andresrm@idi.ntnu.no

Darío Ramos-López* University of Almería Email: dramoslopez@ual.es

Antonio Salmerón University of Almería Email: antonio.salmeron@ual.es

Thomas D. Nielsen Aalborg University Email: tdn@cs.aau.dk

Helge Langseth Norwegian University of Science and Technology Email: helgel@idi.ntnu.no

Anders L. Madsen Hugin Expert A/S and Aalborg University Email: anders@hugin.com

***Abstract*—The AMIDST Toolbox is an open source Java 8 library for scalable learning of probabilistic graphical models (PGMs) based on both batch and streaming data. An important application domain with streaming data characteristics is the banking sector, where we may want to monitor individual customers (based on their financial situation and behavior) as well as the general economic climate. Using a real financial data set from a Spanish bank, we have previously proposed and demonstrated a novel PGM framework for performing this type of data analysis with particular focus on concept drift. The framework is implemented in the AMIDST Toolbox, which was also used to conduct the reported analyses. In this paper, we provide an overview of the toolbox and illustrate with code examples how the toolbox can be used for setting up and performing analyses of this particular type.**

I. Introduction

The AMIDST Toolbox is an open source library implemented in Java 8 for scalable machine learning of *probabilistic graphical models* (PGMs). PGMs constitute a principled modeling framework for learning and reasoning under uncertainty, and are defined by two parts: first, a qualitative component in the form of graph representing independence relations between the variables (i.e., nodes) in the domain being modeled; secondly, a quantitative component consisting of a set of probability distributions quantifying the relations specified in the graph. In particular, we consider hybrid *Bayesian networks (BNs)* [6], [3], [4], where the qualitative part is a direct acyclic graph (DAG) and the quantitative part is a set of conditional probability distributions, one for each variable given its parents. We will also consider the dynamic extension of BNs, namely *dynamic Bayesian networks (DBNs)* in the form of 2-Timeslice BNs. The AMIDST Toolbox supports the specification of both BNs and DBNs, including models containing *latent variables*, thereby also facilitating a fully Bayesian approach for doing model learning [5].

The financial sector represents an important application domain, where data usually have a streaming nature, i.e., new data is continuously being generated. When analyzing this kind of data, even a modest updating frequency can produce huge volumes of data, thereby making efficient and accurate data analysis and prediction extremely difficult. To handle these big data streams, current systems for PGMs often employ simplistic solution techniques that, e.g., only consider the most recently generated data or only store the data at a much lower frequency than with which it is generated. By doing so, potentially valuable data, which could otherwise have contributed to an improved system accuracy, are being ignored. By contrast, the AMIDST Toolbox has the possibility of handling large data streams. To achieve that, the algorithms implemented in AMIDST are scalable and can leverage both multi-core and distributed architectures, the latter through integration with Apache Flink[1]. To the best of our knowledge, the AMIDST Toolbox is the first system to directly support scalable mining and analysis of data streams based on PGMs.

A salient aspect of streaming data in general, and financial data in particular, 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* [2] and if not carefully taken into account, the result can be a failure to capture and interpret intrinsic properties of the data. This detection is especially crucial when analyzing financial data as the economic cycles and changes in the general economic climate can undermine the learned models.

We have previously addressed the issue of concept drift detection in the financial domain based on real customer data from a Spanish bank [1]. The results reported in [1] were obtained using the AMIDST Toolbox, which also implements the concept drift detection framework that was proposed. In this paper we first provide an overview of the AMIDST Toolbox with focus on its main features and architecture. To illustrate the use and significance of the toolbox we next describe and provide complete code examples for setting up and performing the concept drift analyses reported in [1]. The code examples have been chosen to not only illustrate how to do financial data analysis using AMIDST, but also to provide insight into the more general toolbox design and functionality. The toolbox is open source and can be downloaded from http://www.amidsttoolbox.com.

*These four authors are considered as first authors and contributed equally.

[1] flink.apache.org

II. Toolbox description

A. *Main features*

The main features of the AMIDST Toolbox are summarized as follows:

- **Probabilistic graphical models**: the AMIDST Toolbox supports the specification of PGMs with discrete and continuous variables and temporal dependencies. PGMs handled by AMIDST may contain latent (i.e., hidden) variables. This supports the representation of a large range of problems with complex probabilistic dependencies. Moreover, due to the intuitive nature of PGMs, the models and results can easily be understood by financial experts who are usually not familiar with machine learning methods.
- **Multi-core and distributed processing**: AMIDST provides parallel and distributed implementations of parameter learning [8], [7] and inference [9], [7] algorithms. The implementations are based on flexible and scalable message passing algorithms using Java 8's built-in functionalities, thereby making the toolbox suitable for analyzing streaming data. Additionally, massive data sets can be processed in distributed computer clusters by interfacing with Flink. Integration with Apache Spark[2] is under development.
- **Concept drift detection**: detect frequent changes in the financial data using a novel probabilistic graphical modeling framework that provides an explicit representation of concept drift as an integral part of the model.
- **Interoperability**: leverage existing functionalities and algorithms by interfacing to other software tools such as HUGIN, MOA, Weka, R, etc.

B. *Architecture*

The AMIDST Toolbox has been designed following a modular structure, i.e., all the provided functionality is divided across different modules as depicted in Fig. 1. The most important modules in AMIDST are *core* and *core-dynamic*, which implement the functionality related to BNs and DBNs, repectively (i.e., learning and inference algorithms, classes handling data streams, etc.). In fact, in Fig. 1 it can be observed that, either directly or indirectly, all the modules depend on the core module (the direction of the arrows indicate the dependency). The rest of the modules are: *examples*, containing a set of code-examples of the toolbox; *latent-variable-models* implementing a set of predefined models [5]; the *lda* module, which allows text processing by means of the latent dirichlet allocation model; and finally, the modules that allow for interaction with external software are *huginlink*, *moalink*, *wekalink*, and *flinklink*.

The modular design of the toolbox, with key functionality located in a kernel with few distinct modules (*core* and *core-dynamic*), enables future extensions of the toolbox to be made independently of the core design, thereby leaving the kernel small and robust. Another added value of the modularity is that it enables a more seamlessly interaction with external software.

The AMIDST software has been built around the Maven[3] automation tool for managing software projects. Maven has been chosen because it is widely spread and renowned for its ability to handle dependencies. By leveraging this tool, using AMIDST in a software project is easy and almost transparent to the user: simply include the xml fragment shown in Code Fragment 1 in the corresponding *pom.xml* file (i.e., the configuration file in the Maven project, where dependencies are defined).

```xml
<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>0.5.2</version>
  <scope>compile</scope>
 </dependency>
</dependencies>
```

Code Fragment 1: XML code for loading AMIDST and its transitive dependencies in a Maven project.

The Code Fragment above imports all the AMIDST modules. However, if only a subset of the modules is needed, one simply changes the content of the entry `artifactId` to list the modules that should be imported.

C. *Predefined probabilistic models*

The AMIDST Toolbox contains a wide range of predefined models (see Table I). These models are available in the module *latent-variable-models* (version 0.4.2 or higher).

TABLE I: Predefined models in AMIDST.

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

Learning and applying these models is straight-forward. As an example, Code Fragment 2 shows how a static model, in this case a *Mixture of Gaussians* [5], can be learned from a dataset with three continuous variables (by default 2 mixtures components are learnt, but this parameter can be easily changed with `setNumStatesHiddenVar`). Afterwards the BN is saved to disk.

```java
//Load the datastream
String path = "datasets/simulated/";
String filename = path+"data_0.arff";
DataStream<DataInstance> data =
   DataStreamLoader.open(filename);

```

[2]http://spark.apache.org

[3]https://maven.apache.org

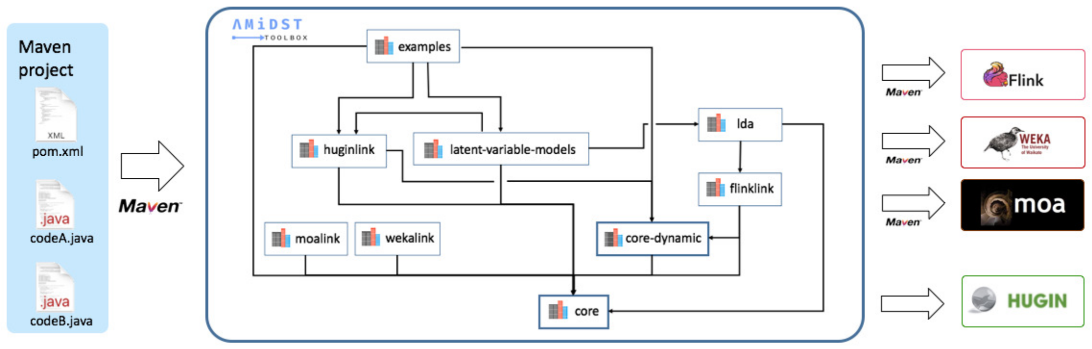

Fig. 1: The modular design of the AMIDST Toolbox, including current external interfaces.

```
 7   //Learn the model
 8   Model model = new GaussianMixture(data.getAttributes());
 9   model.updateModel(data);
10   BayesianNetwork bn = model.getModel();
11
12   // Print the BN and save it
13   System.out.println(bn);
14   BayesianNetworkWriter.save(bn, "networks/simulated/network.bn");
```

Code Fragment 2: Learn a static model from a data stream.

Learning another type of static model (say, a factor analyzer) just requires changing the constructor used in Line 8 of Code Fragment 2 with e.g.:

```
1   Model model = new FactorAnalysis(data.getAttributes());
```

The learning process is done when method `updateModel` is called. Note that, internally, the data stream is divided and processed into smaller batches (for scalability purposes). The size of these smaller batches can be specified with `setWindowSize`. The model can be easily updated if new data becomes available (see Code Fragment 3).

```
1   //Update the model with new information
2   for(int i=1; i<12; i++) {
3     filename = path+"data_"+i+".arff";
4     data = DataStreamLoader.open(filename);
5     model.updateModel(data);
6     System.out.println(model.getModel());
7   }
```

Code Fragment 3: Update the model when new data becomes available.

III. Analyzing Financial Data: Examples

In this section, we describe and provide code examples[4] to illustrate how our toolbox can be used for modeling concept drift in streaming financial data using the approach of [1].

The financial data set has been provided by Banco de Crédito Cooperativo (BCC). The data contains monthly aggregated information for approximately 50000 BCC clients for the period from April 2007 to March 2014, including a binary variable indicating whether or not the clients *defaulted* on their financial obligations.

[4]A Maven project with these examples can be downloaded from https://github.com/amidst/tutorial.git (branch icdm2016_demo).

Code Fragment 4 shows the code needed for analyzing concept drift. After the dataset is loaded (Line 6), we create an object of class `NaiveBayesVirtualConceptDrift`. Figure 2 shows the graphical model defined by the class. Before any processing is performed, we invoke the `initLearning` method (all parameters have default values). We finally update the model every month to get the corresponding mean of the latent variables (only 1 in this example). Major changes in these values are a sign of concept drift.

```
 1   String path = "datasets/simulated/";
 2   String filename = path+"BCCDefault.arff";
 3   int windowSize = 500; // instances per month
 4
 5   //Open the data stream
 6   DataStream<DataInstance> data =
 7       DataStreamLoader.open(filename);
 8
 9   //Create a NaiveBayesVirtualConceptDriftDetector object
10   NaiveBayesVirtualConceptDriftDetector virtualDriftDetector =
11       new NaiveBayesVirtualConceptDriftDetector();
12
13   //Set class variable
14   virtualDriftDetector.setClassIndex(2);
15
16   //Associate the stream with the detector
17   virtualDriftDetector.setData(data);
18
19   //Fix the size of the window
20   virtualDriftDetector.setWindowsSize(windowSize);
21
22   //Fix the number of global latent variables
23   virtualDriftDetector.setNumberOfGlobalVars(1);
24
25   //Fix the transition variance
26   virtualDriftDetector.setTransitionVariance(0.1);
27
28   //Should invoke this method before processing any data
29   virtualDriftDetector.initLearning();
30
31   //At each iteration, a value for the hidden variable
32   //is obtained
33   for (DataOnMemory<DataInstance> batch :
34       data.iterableOverBatches(windowSize)){
35     double[] H = virtualDriftDetector.updateModel(batch);
36     System.out.println(H[0]);
37   }
```

Code Fragment 4: Concept drift detection in financial data.

The model in Figure 2 corresponds to the code shown in Code Fragment 5. In the figure, $(Y_{i,t}, \mathbf{X}_{i,t})$ describes the behavior of client i at time t, where $Y_{i,t}$ represents the defaulting status of client i at time t and $\mathbf{X}_{i,t}$ the financial indicator variables describing the client. The distributions of $Y_{i,t}$ and $\mathbf{X}_{i,t}$ are parameterized using the parameters θ_y and θ_x, respectively. Concept drift is captured in the model through the latent variables $H_1, H_2, \ldots, H_T$, which are "shared" across clients. When learning from data subject to concept drift, the model responds by "tweaking" H_t over time.

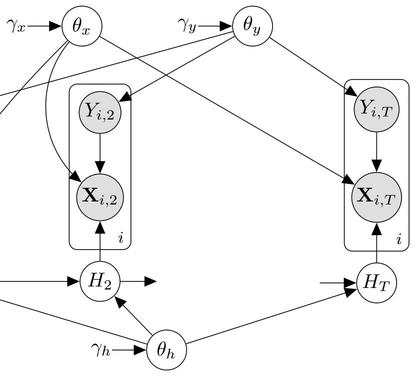

Fig. 2: Model of concept drift [1].

```
//Represent the list of attributes in the dataset (expect the class)
Variables variables = new Variables(data.getAttributes());

//Represent the list of hidden vars modelling concept drift
List<Variable> hiddenVars = new ArrayList<Variable>();

//Create numberOfGlobalVars hidden variables
for (int i = 0; i < this.numberOfGlobalVars ; i++) {
 hiddenVars.add(variables.
          newGaussianVariable("GlobalHidden_"+i));
}
Variable classVariable = variables.getVariableByName(className);

//Create a DAG from all the variables
DAG dag = new DAG(variables);

// Add links into indicator variables. Special attributes
// are omitted (i.e. those with meta−information)
for (Attribute att : data.getAttributes()
         .getListOfNonSpecialAttributes()) {
 if (att.getName().equals(className))
  continue;

 Variable variable = variables.getVariableByName(att.getName());
 //Set the class as parent of all observed variables
 dag.getParentSet(variable).addParent(classVariable);

 //Set the latent variables as parents of all observed variables
 for (int i = 0; i < this.numberOfGlobalVars ; i++) {
  dag.getParentSet(variable).addParent(hiddenVars.get(i));
 }
}
```

Code Fragment 5: Build Naive Bayes DAG for concept drift modeling

Note that the `updateModel` call in Code Fragment 4 (Line 36) encapsulates the updating calls in smaller batches to the learning algorithm. It also retrieves the expected means of the latent variables selected.

The result of conducting the data analysis with the financial data provided by BCC corresponds to the monthly expected value for a single latent variable as shown in Fig. 3. This results highly correlates (the Pearson correlation coefficient is 0.96) with the unemployment rate in the region of the bank during the same period.

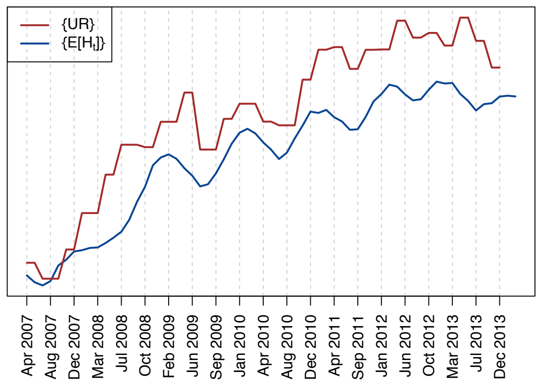

Fig. 3: Evolution of the latent variable for the financial data set modeling concept drift and unemployment rate in Almería region for the same period.

IV. Conclusions and Future Work

We have presented how the AMIDST toolbox can be used to analyze financial data with PGMs, in particular, its application to concept drift detections. With the code-examples, it has been shown that it is a flexible and easy-to-use library. In the future, we aim to integrate it with Apache Spark.

Acknowledgment

This work was performed as part of the AMIDST project. AMIDST has received funding from the EU Seventh Framework Programme for research, technological development and demonstration under grant agreement no 619209. We thank BCC for providing access to their data.

References

[1] Hanen Borchani, Ana M Martínez, Andrés R Masegosa, Helge Langseth, Thomas D Nielsen, Antonio Salmerón, Antonio Fernández, Anders L Madsen, and Ramón Sáez. Modeling concept drift: A probabilistic graphical model based approach. In *International Symposium on Intelligent Data Analysis*, pages 72–83. Springer, 2015.

[2] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014.

[3] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer Verlag, Berlin, Germany, 2007.

[4] Uffe B. Kjærulff and Anders L. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. 2nd Edition. Springer Verlag New York, 2013.

[5] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[6] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA., 1988.

[7] Antonio Salmerón, Darío Ramos-López, Hanen Borchani, Ana M Martínez, Andrés R Masegosa, Antonio Fernández, Helge Langseth, Anders L Madsen, and Thomas D Nielsen. Parallel importance sampling in conditional linear gaussian networks. In *Conference of the Spanish Association for Artificial Intelligence*, pages 36–46. Springer, 2015.

[8] FW Scholz. Maximum likelihood estimation. *Encyclopedia of statistical sciences*, 1985.

[9] John Winn and Christopher M Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(Apr):661–694, 2005.