Proceedings of the 6th Győr Symposium and 3rd Hungarian-Polish
and 1st Hungarian-Romanian Joint Conference on Computational Intelligence

Integration of Model Based Prediction into Complex Event Processing Applications

Bogdan Târnaucă[1], Florin Moldoveanu[2], Constantin Suciu[3], Lucian Mircea Sasu[3], Dan Puiu[3]

[1]**Siemens Corporate Technology, Braşov, Romania**
E-mail: bogdan.tarnauca@siemens.com
[2]**Transilvania University of Braşov, Romania**
[3]**Siemens Corporate Technology, Braşov, Romania**

Abstract: This paper reviews two major data–consumers: Complex Event Processing and Machine Learning. Although they have different targets and outputs, they are complementary rather than competing technologies. CEP, through its rich set of operations, may contribute to data integration and data cleansing, which is a vital part of ML process. On the other hand, ML produces inferential models which can be further integrated as user–defined functions in CEP workflows. The ML paradigm enriches CEP by its different vision: the models are data–driven and produce insights which may be hard to be detected, even by experienced human experts.

Keywords: *Complex event processing, machine learning, integration pattern*

1. Introduction

The data deluge phenomenon is omnipresent, and it became a salient leverage for taking decision in business, science, government. The main issue one faces is efficiently accessing, exposing and exploiting the data. Besides this, discovering and exposing the intrinsic knowledge is another prominent goal. This paper discusses two branches — Complex Event Processing (CEP) and Machine Learning (ML) – which complementarily act and share the data stage. The main merit of this paper is showing how these two paradigms can mutually benefit of each other and a use–case showing their synergistic integration.

The field of Complex Event Processing is composed of the techniques and the tools used for processing, in near real-time, high volumes of events. The complex events are

derived by exploiting the structural and temporal relations between events. The Machine Learning paradigm provides algorithms and methodologies for pattern detection and model extraction from data. It is a data–driven approach, mostly relying on non–parametric models.

Both CEP and ML are data–consumers, but their aims are different and complementary to each other: the former offers the framework for expression and execution of human–defined detection statements, while the latter is specialized towards inference, based on training data.

2. CEP and ML: brief presentations

CEP was originally designed for system architecture prototyping with powerful event oriented semantics. Although near real-time event processing platforms and solutions exists for decades [1], CEP has consolidated itself as a separate topic rather recently. The temporal, causal and structural relations between the raw events are exploited with the goal of producing added value information represented as complex events. The usual event–oriented operations [2] implemented by a CEP engine are: filtering, translation, aggregation, composition, pattern detection. The Event Processing Language (EPL) is a crux characteristic for the CEP platform, and represents all the operators which can be used by the developer/domain expert to express the logic which is executed by PNs (such as the operations presented above). Because the EPL of the CEP engines provides operators such as windows, aggregation, joining and analysis functions the domain expert can define EPNs that perform event stream analysis.

Almost all the data oriented applications contain components and mechanisms responsible for data cleansing and preprocessing. A CEP based solution can be recommended for such operations because it often involves processing of high amounts of events with strong temporal constraints. In this case the data preprocessing occurs after the events have entered the processing EPN and the main operations that can be achieved are the following ones: data filtering, missing data detection or constraints/correlation based validation.

Machine learning targets development of algorithms which are able to improve their behavior based on the acquired experience. The models resulted in ML are data driven. There are plenty of proposed ML algorithms: neural networks, support vector machines, Bayesian networks, decision trees, association rule learning, etc. We also witness an upsurge interest in ensemble methods, i.e. combining multiple learners. The usual workflow for a ML–based process is: *a)* start from raw data; *b)* perform data cleansing and data preprocessing; *c)* apply a ML algorithm to build a model from data. The derived model is subsequently used to process upcoming data, mainly for making predictions.

The data cleansing step is tightly bound to data quality. One has to analyze the data and to identify the potential errors in the data set. The next step is preparing data for the Machine Learning algorithms. Some of the ML methods specify the data types they are able to process (e.g. neural networks accept input numerical values solely, and only a few of them allow missing values).

There are three approaches to deal with initial data, all of them part of feature engineering. The easiest one is to chose among the existing attributes and restrict to the ones that are considered to be relevant for the task at hand. This step may be supported by a domain expert or automatically performed. The second approach produces new features based on data attributes: one may compute derived features (e.g. lagged values for time series forecasting, or ratio of various quantities), hoping that the newly added values are more relevant for the learning process. The third approach is a recent trend and allows the machine to extract features on its own, without human intervention. We refer here to the so-called "deep learning" branch of ML [3], where successive layers of abstractions are automatically built through learning. The following types of learning mechanisms are used in ML: supervised, unsupervised and reinforcement learning, with some minor hybridization and extensions of these mechanisms, e.g. semisupervised and active – the algorithms makes use of both labeled and unlabeled data – and inductive transfer.

3. Integration Pattern

In Figure 1 we show the integration pattern of ML models in CEP based application. The CEP engine is used at design time in the data preprocessing step for generating the training and validation data sets. At run time, in the CEP engine two EPNs are deployed. The first one is responsible for selecting the sample attributes, which is used by the prediction model for generating a prediction. The second EPN contains a user–defined function (UDF) where the prediction model is embedded.

As discussed in the previous section, data preprocessing is a critical step for the successfulness of a ML task. CEP shows itself as a good candidate to hold the data processing workflow in form of EPNs, based on customized CEP statements. One may use some natively–supported CEP features like computing aggregated values on a window or creating lagged or differenced variables. Hence, the CEP is expected to critically contribute to the feature selection/extraction stage.

In the proposed workflow, after the prediction model is generated and validated, it is serialized in order to be deployed in the CEP engine. Most programming languages and platforms allow for object serialization. The main merit of binary serialization is the support for saving any customized version of the learning model. The main drawback is

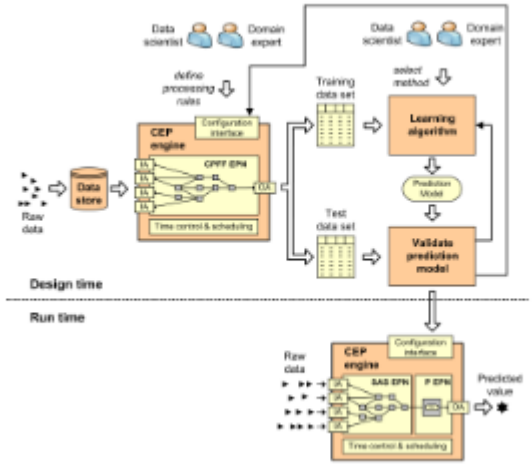

Figure 1. The integration pattern of ML models in CEP applications (CPFF EPN: Cleansing Preprocessing and Feature Filtering EPN; SAS EPN: Sample Attribute Selection EPN; P EPN: Prediction EPN).

that it is challenging to consume a binary object outside the environment which produced it. An alternative with increasing popularity is the open standard Predictive Model Markup Language, an XML–based language which can represent not only predictive and descriptive models, but also data pre– and post– processing steps.

During runtime, the prediction model is deserialized and it is deployed in the CEP engine as a UDF. In addition to that, the native features of the CEP are used for joining data from disparate or asynchronous/unsynchronized data sources, for selecting the sample attributes used by the prediction models to generate the predicted value.

4. Sample Application

In this section is described an example of how to embed a time series prediction model into an Esper CEP user defined function. The manipulation of the prediction models was done using Weka suite [5]. According to the integration pattern presented in section 3, three steps have to be accomplished in order to embed a prediction model in a PN of a CEP engine. First, the data scientist and the domain expert select the attributes set, the prediction model type and train the prediction model. At the end of first step the resulting

prediction model is serialized. The process of training and serializing the time series prediction model is presented in listing 1. The first two lines from the listing creates the time series prediction object and then, in the third one, the prediction model is trained using the training Data. At the end, the resulting prediction model is serialized to the file *PATH_TO_SERIALIZED_FILE*.

```
1 WekaForecaster wekaForecaster = new WekaForecaster();
  wekaForecaster.setBaseForecaster(new LinearRegression());
3 wekaForecaster.buildForecaster(trainingData);
  SerializationHelper.write(PATH_TO_SERIALIZED_FILE, wekaForecaster);
```

Listing 1: Create, train and serialize the time series prediction model.

In the second step, the prediction model is deserialized and is included in an Esper UDF, which at the end is deployed in the CEP engine. These operations are presented in listing 2. After the time series prediction object is restored from the serialized file (first line), it is included in the custom made UDF: *TimeSeriesPredictionModelUDFClass*. A configuration object (*cepEngineConfiguration*) is further created for the CEP engine. In the line seven of the listing 2, the UDF is registered in the CEP engine using the configuration object. The following parameters have to be included in the configuration object: *operatorName*: the name of the custom made operator, which will be used within the queries deployed in the PN; *udfClassName*: the name of the class, which represents the UDF; *udfMethodName*: the method of the UDF class, which is invoked by the operator. At the end, the Esper CEP engine is instantiated using the configuration file.

```
WekaForecaster wekaForecaster = (WekaForecaster) SerializationHelper.read(
    PATH_TO_SERIALIZED_FILE);
2 TimeSeriesPredictionModelUDFClass.setModel(wekaForecaster);
  Configuration cepEngineConfiguration = new Configuration();
4 cepEngineConfiguration.addPlugInSingleRowFunction("
    predictUsingTimeSeriesModel", "TimeSeriesPredictionModelUDFClass", "
    predict");
  EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider(
    cepEngineConfiguration);
```

Listing 2: Deserialize the Time Series prediction model, load the prediction model in a UDF and start the Esper CEP engine.

Listing 3 presents a sample query which uses the custom made operator defined previously (*predictUsingTimeSeriesModel*). The PN, which executes the query, is triggered every time when an *InputEvent* is generated. The custom made operator *predictUsingTimeSeriesModel* is invoked with the parameter *eventParameter* from the *InputEvent*. At the end a *PredictionEvent* is generated which has the value returned by the prediction model in the *predictedValue* field.

```
insert into PredictionEvent select predictUsingTimeSeriesModel(*) as
    predictedTemperature from TimeSeriesPredictionModelSample
```

Listing 3: Sample Esper based query which uses the time series prediction model embeded in a UDF.

5. Conclusions

The integration pattern presented in this paper exhibits CEP and ML complementarity: CEP supports the data cleansing and feature filtering – needed for model training – and sample attribute selection – for the inference step. The inference step is supported by ML, from which the inference model is embedded as an UDF in a CEP workflow. The CEP language of defining strong temporal relations between events simplifies the process of sample attribute selection, because at this step the raw events have to be synchronized/(aligned in order to generate the sample used by the model to make a prediction. Also this step, in the traditional approach, is accomplished by custom made applications. In addition to these advantages, the integration of a prediction model in a CEP UDF improves the ability of the engine to detect pattern which are beyond the availability of the native CEP language. Due to the recurrence of these scenarios: heterogeneous data to be consumed and exposed to machine learning plus integrating machine learning outcome in the data workflow, we are confident that the described integration can be promoted as a pattern: "a widely recognized and reused solution to a recurring design problem".

References

[1] N. Leavitt, "Complex-Event Processing Poised for Growth," *Computer*, vol. 42, no. 4, pp. 17–20, Apr. 2009.

[2] O. Etzion and P. Niblett, *Event Processing in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2010.

[3] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009, also published as a book. Now Publishers, 2009.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.