Automated decision-making with DMN: from decision trees to decision tables

D. Etinger*, S.D. Simić* and L. Buljubašić*  
* Juraj Dobrila University of Pula/Faculty of Informatics, Pula, Croatia  
{darko.etinger, ssimic, lea.buljubasic}@unipu.hr

Abstract - Recent advances in artificial intelligence, especially the subfield of machine learning, is commonly cited as one of the driving forces for digital transformation and innovative business models. Ongoing research is focusing on embedding solutions based on machine learning into business processes which are commonly modelled using the BPMN standard. The Object Management Group has recently adopted the Decision Model and Notation standard. By using the Decision Model and Notation (DMN) it is possible to replace multiple decision points embedded in business processes. The purpose of this research is to provide a method to derive DMN decision tables from the corresponding machine learning model generated by the decision tree classifier. The development is conducted using the Python machine learning library scikit-learn and Camunda Modeler. This approach facilitates and automates the process of converting machine learning models into DMN tables.

Keywords - Business Process Management (BPM); Decision Model and Notation (DMN); decision trees

I. INTRODUCTION

Business process management (BPM) tools have been extensively used for digital transformation within companies: recourse orchestration, work routing and automation of routine manual tasks are just some examples of business/IT alignment that enable companies to work more efficiently. As business process management (BPM) systems become more advanced, there is a shift from pre-defined process models to intelligent systems that support and automate processes. Modern BPM systems are general-purpose application development platforms that allow integration of other technologies through e.g. RESTful API calls. These innovative and augmented processes usually include artificial intelligence applications, powered by machine learning algorithms. Technologies that are being integrated into modern BPM systems include business rules, predictive analytics, machine learning and robotic process automation (RPA), Internet of Things (IoT), process mining algorithms etc.

Advances in machine learning algorithms include image recognition systems, recommender systems and natural language processing automation tasks. As business processes are generally great candidates for automation, combining BPM systems and machine learning together can take automation to another level. By connecting machine learning applications to existing BPM tools and feeding process data to machine learning applications, companies could decrease the user tasks even more, while at the same time also deliver a better product to market.

The content of this paper is structured as follows. Section II presents selected relevant literature and establishes the theoretical framework for the development of the proposed method. Section III details the conceptual modelling process and the results of the development process. The method is presented and described in detail. Finally, section IV includes the summary of the results, guidelines for further actions, limitations of the study and concluding remarks.

II. LITERATURE REVIEW AND THEORETICAL FRAMEWORK

The DMN model is primarily designed for business people who are modeling business decisions. Its central construct is the decision table. Relevant objects in DMN tables are business knowledge models that relate to implementation and sources of knowledge that serve for documentation. DMN tables are based on the FEEL (Friendly Enough Expression Language) language, a standard DMN language. FEEL language has the following characteristics [1]: a) side-effect free; b) simple data model with numbers, dates, strings, lists, and contexts; c) simple syntax designed for a wide audience; d) three-valued logic (true, false, null) based on Structured Query Language (SQL) and Predictive Model Markup Language (PMML). The elements of the DMN decision table are [1]: Decision, Business Knowledge Model, Input Data, Knowledge Source and Decision Service.

The purpose of DMN is to provide constructs that are needed to model decisions, so that organizational decision-making can be readily depicted in diagrams, accurately defined by business analysts, and (optionally) automated [2]. When using a DMN table, it is desirable to understand clearly the business objective and specific business rules to avoid storing large amounts of data. The DMN decision table contains columns and rows that indicate the input/output logic and rules.
The basic elements and the structure of a decision model are shown in Figure 1. In this case, the decision is obtained using the logic of the business knowledge model that is derived from the input values.

Due to the increasing use of DMN decision tables in the business world, there is a growing need for analysis of these tables [2]. Rules for defining task analysis over this tables and a general approach for analysis and processing of decision tables based on geometric interpretation is proposed [3]. The authors provide a solution to simplify the DMN tables by merging the rules when they have the same expression and conclude that the variant with a single permutation is sufficient for practical application. The complexity of the Decision Model and Notation was assessed, and the results were compared with previous studies for other modelling notations, such as Unified Modeling Language (UML) activity diagrams, Business Process Model & Notation (BPMN), and Case Management Model & Notation (CMMN) [4]. The authors note that the complexity of the process models decreases as the process is reduced to its essence, i.e. without hard-coding the decision logic within the process. It is also suggested that employing DMN in organizational decision making might be specifically useful in a setting where business rules change frequently and where decisions have high risk for the operations [5].

The decision tree is a recursive graphical algorithm that uses the branching method to illustrate the possible out-come of a decision. It works on a Boolean algorithm principle divide and conquer. It divides data into smaller subcategories and if those unambiguous algorithms cease to work and we get the result. However, if subcategories are not uniquely determined, then the algorithm is the next subcategory of the other attributes as long as it does not get a sub-category that is uniquely determined. Using the decision tree, we get new data that we can use to build a DMN table.

Applications based on machine learning algorithms and other intelligent systems that integrate DMN decision tables with BPM systems can be developed in various ways. One approach is to integrate machine learning applications with microservice architecture, then use it as an external service in BPM systems through signal events (with BPMN represented as the triangles in enclosed circles). A dynamic decision-making framework for BPM is proposed as an extension for BPMS [6]. It aims at guiding the actions and decisions of the BPMS, thus, automatically proposing decisions that have a behavioral character. With this approach, a Bayesian network is derived automatically from an event log and a decision model in DMN is based on historic and current process execution data. A user-friendly graphical user interface that allows for interactive decision enactment on the modelled decisions is proposed [7]. The authors proposed a system for translation of DMN to an extension of first-order logic.

The DMN decision modeling process structure shown in Figure 2 includes two activities addressed by the method proposed in this paper: formulate decision logic (usually a part of the business domain) and create decision services (usually part of the IT domain). The design-time approach taken in our case provides a clear advantage over a runtime approach when using decision trees, rules or linear models that can be produced transparently, represented statically by a decision service and made available to all stakeholders, both in the business and IT domain. Recent studies [8] show that such approach not only provides an explanation for any outcome, but also an understanding of how the decision is made in general, irrespective of any specific input data.

While this approach is not suitable for a broad range of applications, it is very convenient in the way that it takes into account existing DMN standards already included in many BPM systems (e.g. Camunda BPM, Goldman Sacks jDMN, Oracle Process Cloud, Red Hat Drools). This facilitates its implementation and further development and customization to specific needs and scenarios.

We argue that design-time approach may not be suitable in settings where Neural Networks, Support Vector Machines (SVM) or Bayesian networks are used. In such cases, a runtime approach would be preferred.
III. RESEARCH METHOD, DEVELOPMENT PROCESS AND RESULTS ANALYSIS

The development process goal is to create a method that enables the conversion of the decision tree model generated by Python-based machine learning algorithm into a DMN based decision table XML artifact that can be imported into various BPM platforms and used within BPMN process models.

The Python programming language and Camunda modeler were the tools chosen for implementing our solution. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics while Camunda modeler is a desktop application (developed in JavaScript) for modeling BPMN and DMN diagrams which are stored in XML (Extensible Markup Language) format. Scikit-learn, a well-documented and widely popular open source machine learning library, and ElementTreeXML, a simple and efficient API for parsing and creating XML data, are the main libraries used in our script. Other Python libraries used in the development process include: Numpy and Pandas (used for reading csv files), defaultdict class from module collections, random and string (used for creating simple id generator), Graphviz (used for visualization of the decision tree classifier) and Progress (from which we used class Bar to create a simple progress bar).

The script we developed is divided into two classes, the base class (xmlDmn) consisting of functions needed for generating DMN tables and derived class (clfDmn) which contains the machine learning model.

A. Machine learning model development

A clfDmn object is created to which two mandatory parameters are passed: a csv file (our training data) and a dmn file (file that contains the table we want to edit).

The constructor is called and starts processing the csv file by creating and splitting the data frame into needed parts and after the data is ready, the machine learning algorithm (DecisionTreeClassifier) is started. The core function in this class is generateTableFromClf shown in Figure 3. This function first extracts information from the classifier: leaf Ids, decision path, thresholds, features, left and right children.

In the next step, a list of features (featuresForDiagram) is prepared, that will be used for creating the columns in the decision table and for the dictionary (inputOutput), because we have to ensure that the order of features are the same. Next follows the cleaning of the table (clearDecisionTable()), generating new table columns (generateTableColumns()) and the looping over leaves is started. From every leaf the class name and all samples in it (samplesInNode) is collected, but only the first one (sampleId) is used. For that sample, its decision path is tracked and looped over it. By doing that, all features, thresholds and threshold signs (<= or >) from all nodes are extracted, after which the filled dictionary is sent to generate the rules in the table (generateTableRows()).

In the end of the function a new file is created, in which all changes made to existing table are written (writeTree()).

```
def generateTableFromClf(self):
    featuresFromDiagram = list(self.feature)
    featuresFromDmn = list(self.feature)
    for feature in featuresFromDiagram:
        if feature in featuresFromDmn:
            featuresFromDiagram.remove(feature)
    featureDictionary = dict(zip(featuresFromDiagram, featuresFromDmn))
    for feature in featureDictionary:
        if feature not in featuresFromDmn:
            featuresFromDmn.append(feature)
    self.featureDictionary = featuresFromDiagram
    self.featureDictionary = featureDictionary
    # remaining code...
```

Figure 3. generateTableFromClf function from clfDmn class

```
def generateTableColumns(self, names):  
    values = list(names)  
    for name in values:  
        if name in self.attribute:  
            if name in self.attribute:  
                self.attribute[name] = self.attribute[name]  
            else:  
                self.attribute[name] = self.attribute[name]  
    self.attribute['input'] = self.attribute['input']  
    self.attribute['output'] = self.attribute['output']  
    return self.attribute
```

Figure 4. generateTableColumns and generateTableRows functions from xmlDmn class
B. DMN tables implementation

When clfDmn object is created xmlDmn constructor gets triggered and starts parsing the mandatory parameter (dmn file) using ElementTreeXML library. Two main functions of class xmlDmn are shown in Figure 4. Both functions are called from generateTableFromClf (see Figure 3). GenerateTableColumns loops over inputNames (sliced list of featuresForDiagram containing only input features) creating a new xml element for each element in the list and at the end the output element is created. At the moment, types from input and output elements are hardcoded. GenerateTableRows loops over dictionary which has specific structure {Class Name : { Feature : {Threshold Sign : Threshold }}}. For each Class Name (variable k in Figure 4) the function creates a new rule after which it loops over all features (variable keyValues in Figure 4) and collects threshold signs (<= or >=) in list (tSignList). Depending on the number of signs in the list we decide to create an empty cell (number of signs = 0), comparison cell (number of signs = 1) or range cell (number of signs = 1). We complete our new business rule with the output element at the end, which contains the class name.

C. Discussion

As we can observe from Figure 5 that every sample in a leaf (node without children) took the same specific path from the root (main node) e.g. node 4 (preorder tree traversal, root node is 0) has 43 samples of class Iris-versicolor. We can conclude that every single sample in that node has “Petal width > 0.8, Petal length <= 4.75 and Petal width <= 1.65”. This knowledge can be extracted from the decision tree and it can be used to create a DMN decision table as we have shown with our script.

Every rule (row) in Figure 5 represents one leaf of the decision tree. By observing Figure 4 we can notice that some cells are left empty which meaning that feature is irrelevant for classification. Furthermore we can notice ranges like “[0.8000..1.6500]” which means that feature “Petal width” needs to be in range 0.8 to 1.65, excluding 0.8 and including 1.65.

IV. CONCLUSION

As BPM systems collect and store detailed process information, paired with machine learning, process data can improve decision making by identifying patterns as the process advances through the workflow, thus automating user tasks that would take significant time (e.g. sales forecasting, demand planning).

The benefits of applying DMN to model business decisions can be summarized as a list that includes: (1) improved speed and faster project cycles, (2) increased participation of business stakeholders, (3) visualization through graphical logic that can reveal hidden relationships, (4) flexibility, transparency and auditability and (5) ease of deployment through REST calls. Benefits also include improved efficiency, both in terms of worker hours and end-to-end cycle time for a transaction, improved quality and compliance due to standardized processes and decisions. All of these contribute to improved customer satisfaction, increased job satisfaction by automating unskilled and semi-skilled tasks that are non-value-added overhead in most workers’ workload.

<table>
<thead>
<tr>
<th>First Decision</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petal length</td>
<td>Sepal length</td>
<td>Petal width</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>U</td>
<td>&lt;= 0.8000</td>
<td>[0.8000..1.6500]</td>
</tr>
<tr>
<td>1</td>
<td>&lt;= 4.7500</td>
<td>[4.7500..5.5900]</td>
</tr>
<tr>
<td>2</td>
<td>[4.7500..5.5900]</td>
<td>[5.5900..5.5900]</td>
</tr>
<tr>
<td>3</td>
<td>[4.7500..5.5900]</td>
<td>[5.5900..5.5900]</td>
</tr>
<tr>
<td>4</td>
<td>&gt; 4.7500</td>
<td>[4.7500..5.5900]</td>
</tr>
<tr>
<td>5</td>
<td>[4.7500..5.5900]</td>
<td>[5.5900..5.5900]</td>
</tr>
<tr>
<td>6</td>
<td>[4.7500..5.5900]</td>
<td>[5.5900..5.5900]</td>
</tr>
<tr>
<td>7</td>
<td>[4.7500..5.5900]</td>
<td>[5.5900..5.5900]</td>
</tr>
<tr>
<td>8</td>
<td>[4.7500..5.5900]</td>
<td>&gt; 5.5900</td>
</tr>
<tr>
<td>9</td>
<td>[4.7500..5.5900]</td>
<td>&gt; 5.5900</td>
</tr>
<tr>
<td>10</td>
<td>&gt; 4.8000</td>
<td>&gt; 4.8000</td>
</tr>
<tr>
<td>11</td>
<td>&gt; 4.8000</td>
<td>&gt; 4.8000</td>
</tr>
</tbody>
</table>

Figure 6. Decision table created from extracted decision path of every leaf
In this paper we presented a method that creates DMN decision tables from a decision tree model. The proposed method automates the process of converting decision tree models (generated by the decision tree classifier) to DMN decision tables, which can be used alongside BPMN process models within various BPM systems.

The main limitation of the presented method is that relevant tasks have many attributes which can be successfully handled by machine learning algorithms but their presentation in the decision table may be a bottleneck. Future work will focus on extending the method to other machine learning algorithms.

REFERENCES


