Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

MACHINE LEARNING-BASED METAMODELS FOR SAWING SIMULATION

Michael Morin
Frédérik Paradis
Amélie Rolland
Jean Wery
Jonathan Gaudreault
François Laviolette

FORAC Research Consortium / Department of Computer Science and Software Engineering
1065, av. de la Médecine
Université Laval
Québec, QC, G1V 0A6, CANADA

ABSTRACT

We use machine learning to generate metamodels for sawing simulation. Simulation is widely used in the wood industry for decision making. These simulators are particular since their response for a given input is a structured object, i.e., a basket of lumbers. We demonstrate how we use simple machine learning algorithms (e.g., a tree) to obtain a good approximation of the simulator's response. The generated metamodels are guaranteed to output physically realistic baskets (i.e., there exists at least one log that can produce the basket). We also propose to use kernel ridge regression. While having the power to exploit the structure of a basket, it can predict previously unseen baskets. We finally evaluate the impact of possibly predicting unrealistic baskets using ridge regression jointly with a nearest neighbor approach in the output space. All metamodels are evaluated using standard machine learning metrics and novel metrics especially designed for the problem.

1 INTRODUCTION

We propose to tackle a raw matter transformation problem from the wood-products industry using metamodels generated using machine learning techniques. Felled trees are first transformed into logs and taken to a sawmill. Each log is then processed by the mill, the log breakdown resulting in a set of lumbers we call the basket (or the mix) of products. In the industry, it is capital to foresee the lumbers resulting from the transformation of a log or several logs. This information is used in several decision-making processes, from strategic choices on the design of the plant, to tactical decisions regarding the introduction of a new product. Since sawmilling is influenced by many factors, simulation is often the preferred method to support these decisions. Sawing simulators take three-dimensional scans of the logs and determine the lumbers resulting from the sawing. Unfortunately, sawing simulators tend to be slow at processing large volumes of wood.

In this paper, we generate metamodels for sawing simulation using machine learning algorithms. These metamodels are particular in that a sawing simulator response is not a single value. It is the types and numbers of lumbers to be produced (basket of products) which is a structured object. We evaluate the fitted metamodels using novel problem-specific metrics as well as traditional machine learning metrics. Such a metamodel can be used to complement simulation as presented in (Hara et al. 2014) or to replace it in a decision-making context (Barton and Meckesheimer 2006).

As a first specific contribution, we generate metamodels using the *k-nearest neighbors* (*k-NN*) (Fix and Hodges Jr 1951), the *decision tree* (*DT*) (Breiman et al. 1996), and the *random forest* (*RF*) (Breiman 2001) algorithms. All three algorithms are different methods to determine to which known log an input

(and possibly unknown) log resembles the most. Taking a vector that represents the characteristics of a log as input, the fitted metamodel output is a known product basket. These techniques, although they predict a realistic basket, do not fully exploit the simulation problem's structure. A basket of lumbers is physically constrained by the machines used for the transformation of an input log and by the characteristics of that log. In the machine learning community, structured output problems are recognized to be more difficult than their unstructured counterpart (Bakĭr et al. 2007). For that reason, we propose using an advanced *kernel ridge regression* (*KRR*) algorithm (Cortes et al. 2007) as a second specific contribution. The main advantage of the method being that it has the power to predict unknown structured objects (i.e., structures it has never encountered during the training phase). We first exploit this property. The downside of the approach is that it might produce an unrealistic basket from a given log. That is, there might exist no physically realistic log for which the plant can produce that basket. In order to evaluate the impact of this last observation, we developed a second version of the algorithm we called *kernel ridge regression with nearest output* (*KRR-NO*). This version first predicts a basket using *KRR* to finally choose, among known and seen baskets, the one closest to the predicted basket.

The paper is structured as follows. We first introduce background concepts related to sawing simulation, metamodeling, and machine learning (Section 2). Then, in Section 3 we introduce the proposed methods to fit the metamodel of a sawing simulator using machine learning. Experiments are presented in Section 4. We conclude in Section 5.

2 PRELIMINARY NOTIONS

This section has three parts. We first introduce sawing simulation and review important simulators from the sawmilling industry (Section 2.1). Then, we review the background notions of metamodeling and discuss the importance of metamodels in decision making and simulation-optimization (Section 2.2). Finally, we describe machine learning as a method to generate metamodels (Section 2.3).

2.1 Sawmilling and Log Breakdown Simulation

We are interested in the sawmilling process which consists in determining the set of *lumbers* (called *basket* or *mix*) resulting from the *sawing* of a log at a given *sawmill* (plant). The sawing process is a divergent process with co-production (Figure 1). It is *divergent* in that it results in multiple lumbers (outputs) with different characteristics, e.g., thickness and width (Figure 1). The lumbers are *co-produced* in that their output is simultaneous for a single input (e.g., two 2 inches by 3, two 2 by 4, and one 2 by 6). Felled trees are first transformed into logs. These logs are taken to a lumber mill log yard where they wait to be processed by the sawmill. Given a log to process, the sawing equipments decide in real time how that specific log will be transformed. A log is sawn according to physically constrained patterns determined by the configuration of the saws as well as by the physical characteristics of the log. The curvature of a log is, for instance, an important factor that determines its position in the machine and, as a result, influences the set of feasible cuts and thus the outputted lumber set. North American softwood lumber products are normalized according to NLGA (National Lumber Grades Authority) grading rules. The *type* of a product is defined solely by its thickness, its width, and its length. The *grade* of a product is an ordering relation of products of the same type. Even though the North American market allows for any produced lumber to be sold, the prices vary in time. The sawmill equipment is configured to select a feasible pattern that maximizes the basket value. The exact same log sawn by different sawmill equipment would generate a different basket of products.

In the industry, it is capital to be able to determine efficiently the output of the transformation of the logs for a given mill at different planning stages whether it be at a *strategic* (long-term decision), at a *tactical* (mid-term decision), or at an *operational* (short-term decision) level (Genin et al. 2007). Simulation is one of the possible and widely used techniques for such purposes (Sampson 1990, Lindner 2014). There exist multiple sawing simulators, e.g., Autosaw, Optitek, Optsaw, Saw2003, SAWSIM, SIMSAW, and

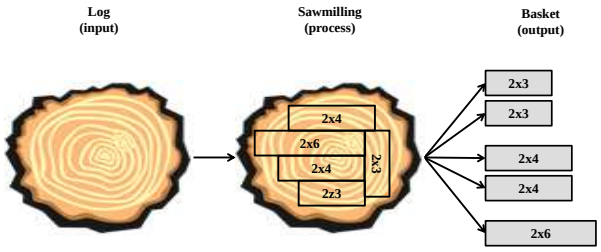

Figure 1: Sawing (process) a log (input) produces a basket (output) with different products types (e.g., thickness and width).

WoodCIM. The input of these simulators is the three-dimensional scans of the available logs. Given the model of a sawmill, the simulator virtually processes each log and generates a *basket* (*mix*) of products resulting from its breakdown. The basket is part of the simulator's *response*. Sophisticated simulators take into account the machines used in the mill's design along with their configuration. They also take into account the physical characteristics of the log and the physical constraints of the sawing process. This leads to a log-specific sawing pattern implying an input-contextual response which includes various reports along with the specific basket.

One example of a strategic usage of sawing simulation is sawmill design. A sawing simulator can be used jointly with other simulators, e.g., a discrete-event simulator, to simulate the sawmill and to verify its productivity. By using various scenarios of sawmill design (including machines specifications and configuration) and multiple virtual log sets, it is possible, for instance, to use a simulator to make decisions related to the "ideal" design of a mill. Sawmill design decisions are strategic-level decisions that might be taken over a lengthy period. As a result, the simulation of multiple scenarios is run off-line and the total running time of each simulation is not of great importance. Examples of such researches are found in the work of (Lundahl 2007) (with Saw2003), (Verret 1997, Tong and Zhang 2006, Liu et al. 2007, Goulet 2007) (with Optitek), (Orbay 1984) (with SAWSIM).

This off-line use is not the only context in which we might require sawing simulation. It happens that a sawmill receives a demand for specific products which are non-standard. Choosing the configuration or the log(s) to process is a difficult task in a co-production context. Many products are generated at once and a small modification might influence the overall production and reduce the profit. This is a case of tactical planning where sawing simulation can help in evaluating multiple scenarios with different sets of logs and configurations (Wery et al. 2014). Finally, for a company owning many sawmills, predicting log breakdown also improves deciding which logs should be sent to which sawmills.

2.2 Metamodeling

Actually, large companies around the world can handle a volume of more than 5 millions m^3 of wood a year (M. and Friberg 2007). Even when considering a single sawmill, the volume of wood processed per year can easily be close to a million of m^3. Simulation of the sawing of the logs involved in a tactical (mid-term) or operational (short-term) decision-making context has tremendous computational costs. Even strategic planning with such a large volume of wood to handle poses computational issues. Besides reducing computational costs, metamodels are used to get insights on a phenomenon at a low cost (design space exploration), to enhance a problem formulation, and as optimization support (Wang and Shan 2007). We now formalize the contexts in which simulation can be used as decision aid. To do so, we discuss the framework of simulation-optimization and describe the concept of metamodel (Barton and Meckesheimer 2006, Wang and Shan 2007, Banks et al. 2010).

In general, simulators can be considered as a black-box function f from an input space Θ to a response space $\mathcal{Y}$ (Banks et al. 2010). Suppose a discrete-event simulator (Barton and Meckesheimer 2006). We

further suppose that the simulator response is a real number representing the expected value of some key performance indicator (KPI) or objective (Barton and Meckesheimer 2006). The goal of the simulation-optimization problem is to find the design parameters $\theta \in \Theta$ that minimize or maximize $\mathrm{f}(\theta)$ where the response function $\mathrm{f}(\theta)$ form is not known (Barton and Meckesheimer 2006). The goal of a *metamodel*, which is a simpler mathematical relationship, is to approximate $\mathrm{f} : \Theta \rightarrow \mathcal{Y}$ (Banks et al. 2010).

Metamodels for optimization are either global or local (Barton and Meckesheimer 2006). A *global* metamodel is fitted once whereas a *local* metamodel is fitted as the optimization progresses. Regression analysis and Kriging interpolation (Van Beers and Kleijnen 2004, Kleijnen 2009), and neural networks (Barton and Meckesheimer 2006) are among the widely used metamodeling techniques. One of the recent applications of metamodels for KPIs predictions is found in the work of (Hara et al. 2014) in (multi-)agent-based simulation (Macal and North 2009). The authors demonstrate how prediction-based metamodels (here Kriging interpolation) can be used to infer a KPI value by using the design parameters $\theta \in \Theta$ along with the results of a partial simulation as input. This additional information proved to be useful at improving the accuracy of the prediction on the KPI. We find, among other recently used metamodeling techniques, Bayesian networks (Poropudas et al. 2011) and least-square regression (Salemi et al. 2012).

Simulators, however, do not only output a single value response. The principal output of the sawing simulators described in Section 2.1 is the basket which is a structured object. Structured object can be used, just as other responses, as part of an optimization (or decision) process, e.g., sawing simulators are used in discrete-event simulators (Orbay 1984, Verret 1997, Lundahl 2007, Tong and Zhang 2006, Liu et al. 2007, Goulet 2007). Just as discrete-event simulators, physical processes simulators require lengthy runtime to virtually reproduce the process to simulate. The machine learning-based metamodels we present in this paper are meant for structured predictions, e.g., they quickly approximate the simulator's structured response which is, in our context, a basket of products.

2.3 Machine Learning Concepts and Methodology

Machine learning algorithms aim at approximating a phenomenon from data observation (Bakĭr et al. 2007). More precisely, in the supervised learning setting, the goal is to learn a predictor $h : \mathcal{X} \rightarrow \mathcal{Y}$ that accurately predicts the output $\mathbf{y} \in \mathcal{Y}$ of an input $\mathbf{x} \in \mathcal{X}$. In our problem, this corresponds to a prediction of the output basket $\mathbf{y}$ given a log $\mathbf{x}$ as input. To do so, the learner (or learning algorithm) has access to a training set of m examples $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_m, \mathbf{y}_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, ..., x_{in}]$ is the feature representation (or feature vector) of the example i, and $\mathbf{y}_i$ is the corresponding output. We assume the examples to be drawn iid (independently and identically distributed) from an unknown distribution D, meaning they are generated using the same distribution (identically distributed) and the occurrence of any example does not affect the probability of encountering any other example (independently distributed).

The goal of the learning algorithm is to find a predictor h, among all predictors in $\mathcal{H}$, achieving the lowest expected loss (minimum number of errors):

$$r(h) = \mathop{\mathrm{E}}_{(\mathbf{x},\mathbf{y})\sim D} l(h(\mathbf{x}), \mathbf{y}), \tag{1}$$

where $l : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ defines the loss incurred when the prediction $h(\mathbf{x})$ differs from the observed value $\mathbf{y}$. Since the distribution D generating the examples is unknown, the expected loss can be estimated with the empirical loss $r_{emp}(h)$ on the training set $\mathcal{S}$:

$$r_{emp}(h) = \frac{1}{m} \sum_{i=1}^{m} l(h(\mathbf{x}_i), \mathbf{y}_i). \tag{2}$$

However, a learning algorithm that only minimizes the empirical loss can result in over-fitting. Over-fitting occurs when the predictor approximates the training set so perfectly and in such a complex fashion that generalization to new examples is no longer possible. Over-fitted predictors usually have a poor

performance on new data. For that reason, multiple learning algorithms minimize a regularized empirical loss that compromises between the performance of the predictor on the training data and its complexity. Moreover, the performances of a predictor are evaluated on an unseen testing set to insure it can accurately predict the output value of new examples.

Since the output space $\mathcal{Y}$ is arbitrary, supervised learning can be applied to a wide variety of tasks, namely classification, regression and structured prediction. The output $\mathbf{y}$ is a discrete value (class) of a finite set for classification tasks, a real value for regression tasks, and a structure like a sequence, a list or a tree for structured output problems. Thereby, the prediction of the basket $\mathbf{y}$ given a log $\mathbf{x}$ can be seen as a structured prediction problem, where the output basket is represented as a list of products. However, structured output algorithms are often computationally more expensive than classification algorithms. To overcome this problem, we can reduce the basket prediction problem to a classification task, where each class corresponds to a basket in the training set. In this setting, the predictor h would only be able to output one of the previously seen baskets. Note that this could be problematic since the correct output of a new log might not be in $\mathcal{S}$. For these reasons, in Section 3, we present both classification and structured output algorithms for metamodels generation in sawmilling simulation.

3 MACHINE LEARNING ALGORITHMS FOR METAMODELS GENERATION

In this section, we describe four machine learning algorithms that were used to train (fit) a metamodel for sawmill simulation. Sections 3.1 to 3.3 respectively describe the use of nearest neighbors and tree-based classification algorithms. Section 3.4 describes the kernel ridge regression algorithm for structured prediction. We describe each algorithm in terms of training phase and prediction phase. The training phase consists in fitting the metamodel. The prediction phase consists in approximating the response of the simulator on a given input.

3.1 K-Nearest Neighbors Classifier

The *k-nearest neighbors* (*k-NN*) algorithm (Fix and Hodges Jr 1951) predicts the output class of a new example by averaging the vote of the k closest examples in the training set. The training phase consists of storing the training examples in a search-efficient data structure. Thereby, the k closest training examples of a new example can be found using a distance function like the Euclidean distance over their feature vectors, i.e., over the known inputs of the training set. The prediction of the output class $\mathbf{y}$ of an unknown example $\mathbf{x}$ is often done by taking the majority class among the k nearest examples found. An alternative approach is to weight each of the k nearest examples according to their distance from $\mathbf{x}$.

3.2 Decision Tree Classifier

The *decision tree* (*DT*) algorithm (Breiman et al. 1996) learns a predictor in the form of a tree structure. This tree consists of decision nodes and leaf nodes. A *decision node* contains a branching rule based on a single feature ($feature \leq value$) which decides if an example $\mathbf{x}$ should take the left or the right path of the tree. Each time an example takes the left path, it respects the rule. Otherwise it does not. A *leaf node* contains the value to predict when $\mathbf{x}$ reaches it.

In the training phase, the tree is constructed in a greedy fashion. The first branching rule is the one that gives the best partitioning of the training examples according to some metric. One of the most commonly used metrics is the *gini impurity* that computes the probability of an example to be misclassified given the branching rule. Next, the training examples take the left path of the tree if they respect the chosen rule, and the right path otherwise. A new decision node is then created on each path by using only the examples reaching that node. This greedy process stops, thus creating a leaf node, when all the examples reaching a node belong to the same class, or as soon as a specified condition to avoid overfitting is met, e.g., maximum depth, maximum number of nodes. Finally, the value of a leaf node is set to the majority class of the training examples reaching it.

In the testing phase, the output class of an unseen example $\mathbf{x}$ is predicted by passing this example through the tree until a leaf is reached. The value of the reached leaf is predicted.

3.3 Random Forest Classifier

The *random forest* (*RF*) algorithm (Breiman 2001) is similar to the *DT* algorithm, but it creates k trees instead of only one. In the training phase, k new datasets of m examples are created by sub-sampling the training examples uniformly with replacement. A tree is then learned on each of the k datasets. In contrast with *DT*, only a random subset of features is used to select the branching rule of each decision node. The output class $\mathbf{y}$ of an unseen example $\mathbf{x}$ can be predicted by passing the example through each tree and returning the most frequently predicted class.

3.4 Structured Kernel Ridge Regression

Given an input $\mathbf{x}$, the structured kernel ridge regression algorithm predicts a vectorial representation of the corresponding output $\mathbf{y}$, and then transforms the predicted vector into a structure of the output space $\mathcal{Y}$. More formally, let $\phi_{\mathcal{X}} : \mathcal{X} \rightarrow \mathcal{H}_{\mathcal{X}}$ be a function that maps an input $\mathbf{x}$ into a high-dimensional vector space $\mathcal{H}_{\mathcal{X}}$, and let $\phi_{\mathcal{Y}} : \mathcal{Y} \rightarrow \mathcal{H}_{\mathcal{Y}}$ be a function that maps an output $\mathbf{y}$ into a high-dimensional vector space $\mathcal{H}_{\mathcal{Y}}$. The goal of the learning algorithm is to find a linear operator $\mathbf{W}$ that transforms a vector of $\mathcal{H}_{\mathcal{X}}$ into a vector of $\mathcal{H}_{\mathcal{Y}}$. Given the predicted vector $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x}) \in \mathcal{H}_{\mathcal{Y}}$, the corresponding output in $\mathcal{Y}$ is predicted by finding the closest $\mathbf{y} \in \mathcal{Y}$ once mapped into the $\mathcal{H}_{\mathcal{Y}}$ space:

$$h(\mathbf{x}) = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmin}} ||\phi_{\mathcal{Y}}(\mathbf{y}) - \mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})||. \tag{3}$$

For the log breakdown problem, $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x}) \in \mathbb{R}^p$ represents the frequency of each of the p products in the basket when the input log $\mathbf{x}$ is sent to a sawmill. However, since the predicted product counts are regression values, $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})$ is an unrealistic basket. The conversion of $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})$ into the $\mathcal{Y}$ space is called the pre-image problem and is often computationally expensive. We first describe how to learn $\mathbf{W}$ and then show how we solve the pre-image problem in our case.

Let $\mathbf{X}$ and $\mathbf{Y}$ be the input and output matrix respectively, where $\mathbf{X}_i = \phi_{\mathcal{X}}(\mathbf{x}_i)$ and $\mathbf{Y}_i = \phi_{\mathcal{Y}}(\mathbf{y}_i)$. By minimizing a regularized l^2 loss, (Cortes et al. 2007) show that the predictor $\mathbf{W}$ is obtained in the following way:

$$\mathbf{W} = \mathbf{Y}(\mathbf{X} \cdot \mathbf{X}^T + \lambda \mathbf{I})^{-1}\mathbf{X}^T, \tag{4}$$

where λ is a parameter controlling the penalty on the complexity of the predictor, $\mathbf{I}_{m \times m}$ is the identity matrix, $()^{-1}$ is the inverse matrix function, and $()^T$ is the transposed function. Note however that there is no need to consider the vectors $\phi_{\mathcal{X}}(\mathbf{x})$ and $\phi_{\mathcal{Y}}(\mathbf{y})$ (which can be of infinite dimensions), but only scalar products of the form $\phi_{\mathcal{X}}(\mathbf{x}) \cdot \phi_{\mathcal{X}}(\mathbf{x}')$ and $\phi_{\mathcal{Y}}(\mathbf{y}) \cdot \phi_{\mathcal{Y}}(\mathbf{y}')$. In many cases, there exists a function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ that emulates the scalar product $\phi_{\mathcal{X}}(\mathbf{x}) \cdot \phi_{\mathcal{X}}(\mathbf{x}')$ by taking as input any $\mathbf{x}$ and $\mathbf{x}'$, and returning a real value representing how similar $\mathbf{x}$ and $\mathbf{x}'$ are in $\mathcal{H}_{\mathcal{X}}$. We call such function a *kernel*. Hence, we need two kernels, to emulate respectively the scalar product $\mathcal{H}_{\mathcal{X}}$ and $\mathcal{H}_{\mathcal{Y}}$. In our experiments, we considered the simplest kernel for the output by using $\mathcal{H}_{\mathcal{Y}} = \mathcal{Y}$, but an output kernel that encodes specific properties of the sawmill basket production problem could also be designed. For the input kernel, we used the *radial basis function* (RBF) (Bishop 2006, Ben-Hur and Weston 2010). The vector $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})$ can now be predicted with:

$$\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x}) = \mathbf{Y}(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{k}_{\mathbf{x}}^T, \tag{5}$$

where $\mathbf{K}$ is an $m \times m$ matrix with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $\mathbf{k}_{\mathbf{x}} = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_m, \mathbf{x})]$.

Once the vector $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})$ is predicted, the corresponding output $h(\mathbf{x})$ can be predicted by finding the basket, over all possible ones, that is the closest to $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})$. This could easily be done by rounding the

values of $\mathbf{W}\phi_{\mathcal{X}}(\mathbf{x})$ to the nearest integer. However, this basket would not be guaranteed to be feasible since the volume of the products could be higher than the volume of the input log. A second version of the algorithm, we call *ridge regression with nearest output* (*KRR-NO*), performs a search in the baskets of the training set to find the nearest.

4 EXPERIMENTS: FITTING MACHINE LEARNING-GENERATED METAMODELS

One goal of the experiment is to compare the sawing metamodels generated using various machine learning algorithms and to illustrate, along the way, the proposed methodology. We describe, in the next two sections, the data we used to train (fit) the metamodels and problem-specific performance measures. The results are presented and discussed in Section 4.3. All algorithms were implemented in Python. All experiments were run using the help of the scikit-learn module (Pedregosa et al. 2011).

4.1 Data

Even if the complete scan of each log could be used as input for the metamodel, the input space can be reduced by identifying the important characteristics of the logs. We chose to use the volume of a log (dm^3), its length (m), its wide-end diameter (cm), its narrow-end diameter (cm), its curvature (cm/m), and its shrinking (cm/m) (a measure of how fast the log shrinks from its wide end to its narrow end) leading to an input vector $\mathbf{x} \in \mathbb{R}^n$ with $n = 6$. Given a log's input, the output is the basket of products defined as a vector $\mathbf{y} \in \mathbb{N}^p$. The length of vector $\mathbf{y}$ depends on the total number of products that can be produced by the sawmill of interest. As mentioned in Section 2.1, the products are standardized according to their type and their grade. Different factors influence the grade including the presence of nodes and singularities, and/or the imperfection of the cut. Nodes and singularities naturally occur in trees. Imperfection of a cut might arise when the lumber is located close to the periphery of the log. We have, in our input vector, few characteristics that could help the machine learning algorithm to identify nodes and singularities. We thus limit ourselves to predicting the type of the products.

The baskets were generated from the three-dimensional scans of real logs using the Optitek sawing simulator (FPInnovations 2014). The dataset we use has 1207 logs generated from 299 trunks with 6 input characteristics and 19 plausible product types.

4.2 Performance Evaluation

The zero-one score and the Hamming distance are two conventional machine learning performance measures for vector outputs. The *zero-one score* s^{Z} is a binary score rewarding only 100% correct predictions only (Bakĭr et al. 2007). The *Hamming distance* d^{H} is a finer metric that penalizes the algorithm for in-basket incorrect predictions, i.e., product by product (Bakĭr et al. 2007). The Hamming distance d^{H} is normalized on the size of the input vector leading to a penalty between 0 and 1.

Definition 1 (Zero-one score) Given a prediction $\hat{\mathbf{y}} \in \mathbb{N}^p$ and an output $\mathbf{y} \in \mathbb{N}^p$, the *zero-one score* s^{Z} gives 1 if $\hat{\mathbf{y}} = \mathbf{y}$ and 0 otherwise:

$$\mathrm{s}^{\mathrm{Z}}(\hat{\mathbf{y}}, \mathbf{y}) = \begin{cases} 1 & \hat{\mathbf{y}} = \mathbf{y}; \\ 0 & \hat{\mathbf{y}} \neq \mathbf{y}. \end{cases} \tag{6}$$

Higher is better. ◁

Definition 2 (Hamming distance) Given a prediction $\hat{\mathbf{y}} \in \mathbb{N}^p$ and an output $\mathbf{y} \in \mathbb{N}^p$, the *Hamming distance* d^{H} is the sum of the prediction errors on each single product j:

$$\mathrm{d}^{\mathrm{H}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{p} \sum_{j=1}^{p} \mathrm{isNotEqual}(\hat{y}_j, y_j) \tag{7}$$

where $\mathrm{isNotEqual}(a, b)$ returns 1 if $a \neq b$ and 0 otherwise. *Lower is better.* ◁

In addition to these classic performance measures, we propose a novel version of the Hamming distance we called the *augmented Hamming distance* $\mathrm{d}^{\mathrm{H}^+}$. The augmented Hamming distance compares two vectors of products (baskets) product by product. For each product, the distance gives a score of 0, if the predictor has correctly predicted the number of this product in the outputted basket. Otherwise, it gives a strictly positive penalty that is proportional to the prediction error with respect to the true output. Finally, we average all these scores to obtain the distance $\mathrm{d}^{\mathrm{H}^+}$. We also propose two scores specific to sawing simulation. We call the first the production ratio $\mathrm{s}^{\mathrm{pro}}$ and the second the prediction ratio $\mathrm{s}^{\mathrm{pre}}$. Here are the precise definitions of these three performance measures.

Definition 3 (Augmented Hamming distance) Given a prediction $\hat{\mathbf{y}} \in \mathbb{N}^p$ and an output $\mathbf{y} \in \mathbb{N}^p$, the *augmented Hamming distance* $\mathrm{d}^{\mathrm{H}^+}$ is the sum of the ratios of the minimum between $\hat{y}_j$ and y_j over the maximum between $\hat{y}_j$ and y_j:

$$\mathrm{d}^{\mathrm{H}^+}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{p} \sum_{j=1}^{p} (1 - f(\hat{y}_j, y_j)) \tag{8}$$

where

$$f(\hat{y}_j, y_j) = \begin{cases} 1 & \text{if } \hat{y}_j = y_j; \\ \frac{\min(\hat{y}_j, y_j)}{\max(\hat{y}_j, y_j)} & \text{otherwise.} \end{cases} \tag{9}$$

Lower is better. ◁

Definition 4 (Production) Let ε be a small positive value. Given $\hat{\mathbf{y}} \in \mathbb{N}^p$ and $\mathbf{y} \in \mathbb{N}^p$, the *production ratio* $\mathrm{s}^{\mathrm{pro}}$ is the average bounded ratio of the real production on the predicted production:

$$\mathrm{s}^{\mathrm{pro}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{p} \sum_{j=1}^{p} \min\left(1, \frac{\max(y_j, \varepsilon)}{\max(\hat{y}_j, \varepsilon)}\right). \tag{10}$$

It corresponds to the percentage of predicted products that are effectively produced. *Higher is better.* ◁

Definition 5 (Prediction) Let ε be a small positive value. Given $\hat{\mathbf{y}} \in \mathbb{N}^p$ and $\mathbf{y} \in \mathbb{N}^p$, the *prediction ratio* $\mathrm{s}^{\mathrm{pre}}$ is the average bounded ratio of the predicted production on the real production:

$$\mathrm{s}^{\mathrm{pre}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{p} \sum_{j=1}^{p} \min\left(1, \frac{\max(\hat{y}_j, \varepsilon)}{\max(y_j, \varepsilon)}\right). \tag{11}$$

It corresponds to the percentage of the real production that is faithfully predicted. *Higher is better.* ◁

Example 1 Figure 2 plots the $\mathrm{s}^{\mathrm{pre}}$ value against the $\mathrm{s}^{\mathrm{pro}}$ value for a fictive problem on an example $(\mathbf{x}, \mathbf{y})$. We compare the prediction of 11 fictive metamodels (predictors). An ideal metamodel h is one such that $\mathrm{s}^{\mathrm{pro}}(h(\mathbf{x}), \mathbf{y}) = \mathrm{s}^{\mathrm{pre}}(h(\mathbf{x}), \mathbf{y}) = 1$ for all $\mathbf{x}$. A pessimistic metamodel h predicts $h(\mathbf{x}) = \mathbf{0}$ for all $\mathbf{x}$. It obtains $\mathrm{s}^{\mathrm{pro}}(h(\mathbf{x}), \mathbf{y}) = 1$ and $\mathrm{s}^{\mathrm{pre}}(h(\mathbf{x}), \mathbf{y}) = 0$. An optimistic metamodel h is one for which $h(\mathbf{x}) = \infty$ for all $\mathbf{x}$. According to Pareto dominance, a metamodel is non-dominated if there is no other metamodel that is at least equal on all criteria while being strictly better on at least one criteria. The dominance region of each non-dominated metamodel (h_1, h_2, and h_3) is filled in blue on the figure. ◁

According to Pareto optimality theory, non-dominated metamodels are incomparable (Talbi 2009). However, in our specific case, the two criteria are commensurable. This means that we can compare them. We propose as our last metric of performance, the area of the dominance region to compare metamodels on a single example $(\mathbf{x}, \mathbf{y})$.

Definition 6 (Production and prediction area) Given a prediction $\hat{\mathbf{y}} \in \mathbb{N}^p$ and an output $\mathbf{y} \in \mathbb{N}^p$, the *area of the dominance region* is:

$$\mathrm{s}^{\mathrm{pro}\times\mathrm{pre}}(\hat{\mathbf{y}}, \mathbf{y}) = \mathrm{s}^{\mathrm{pro}}(\hat{\mathbf{y}}, \mathbf{y})\, \mathrm{s}^{\mathrm{pre}}(\hat{\mathbf{y}}, \mathbf{y}). \tag{12}$$

Higher is better. ◁

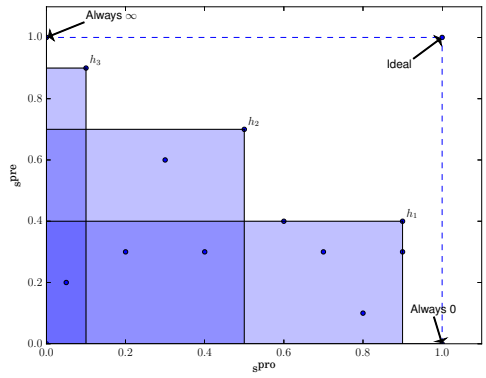

Figure 2: An example of s^{pre} vs s^{pro} plot for a fictive example $(\mathbf{x},\mathbf{y})$ for 11 fictive metamodels; non-dominated metamodels are labeled and their dominance region is filled in blue.

Example 2 The areas of the dominance region of the non dominated metamodels of example 1 are: $s^{pro\times pre}(h_1(\mathbf{x}),\mathbf{y}) = .36$, $s^{pro\times pre}(h_2(\mathbf{x}),\mathbf{y}) = .35$, $s^{pro\times pre}(h_3(\mathbf{x}),\mathbf{y}) = .09$. ◁

We use the expected $s^{pro\times pre}$ value over the sample as our principal quality indicator of a metamodel h. There is a strong parallel to be made between the production ratio, the prediction ratio, and the production and prediction area metrics and the receiver operating characteristic (Fawcett 2004). The former is however closer to our specific problem. The motivation for the proposed metrics can be found in the need, for a wood-industry company, to foresee its production and to evaluate that prediction both in terms of over and under-estimation of the real production. While enabling the proportional comparison of the prediction of the metamodel to the real response of the simulator, the augmented Hamming distance does not give information on the behavior of the metamodel. That is, we do not know whether the metamodel has a tendency to overestimate or to underestimate the production. These insights are given by the prediction and the production ratios.

The baskets are usually sparse. They contain a small number of products. To avoid overestimating the quality of a prediction, it is convenient to filter the outputs prior to evaluation as follows.

Definition 7 (Filtered prediction vector) Let $k_1,k_2,\ldots,k_{p'}$ be the index such that $y_i \neq 0$ or $\hat{y}_i \neq 0$. The filtered vector $\mathbf{y}'$ (resp. $\hat{\mathbf{y}}'$) is such that $\mathbf{y}' = [y_{k_1},\ldots,y_{k_{p'}}]$ (resp. $\hat{\mathbf{y}}' = [\hat{y}_{k_1},\ldots,\hat{y}_{k_{p'}}]$). ◁

4.3 Results and Discussion

Table 1 presents the results of the generated metamodels for various scores (higher is better) on the two datasets described in Section 4.1. To have only metrics for which higher is better in our comparison table, we had considered $1-d$ instead of d itself for both the Hamming distance ($d = \mathrm{d^H}$) and the augmented one ($d = \mathrm{d^{H^+}}$). For each metric, we highlight the single best result in bold. The first dataset does not contain the examples that produce an empty basket. The second one contains all the examples. The former dataset is harder than the latter. The experiment is conducted in three phases. Each dataset is partitioned into a training set (60% of the data) and a test set (40% of the data). The first phase consists in finding the best hyper-parameters for each discussed machine learning algorithm, i.e., *DT*, *RF*, *KRR*, and *KRR-NO*, and *k-NN*. This is done by using a grid search cross-validation procedure on the training set. We use $s^{pro\times pre}$ as our main performance criterion. Once the best parameters are found, the second phase consists in fitting the metamodel using the training set. Finally, in the third phase, we use the test set to assess the quality of the metamodel on unseen logs. The three phases are performed 10 times with different random partitioning of the datasets into a training set and a test set. Each score is computed on filtered baskets (see Def. 7) to avoid overestimating the performance of the metamodels. Without filtering, our fitted metamodels have

a $s^{pro \times pre}$ score of more than 95%. As a basis for comparison, we added a metamodel that predicts the average basket seen in the training set (*MEAN*).

Table 1: Average scores on 10 test set runs (standard deviation); filtered outputs (see Def. 7).

	Without empty baskets						With empty baskets					
Score	*MEAN*	*DT*	*RF*	*KRR*	*KRR-NO*	*k-NN*	*MEAN*	*DT*	*RF*	*KRR*	*KRR-NO*	*k-NN*
s^z	.0722 (.0119)	.5834 (.0221)	**.6088** (.0251)	.5376 (.0181)	.5519 (.0117)	.56 (.0161)	.1905 (.0159)	.7006 (.0172)	**.7265** (.0154)	.6841 (.0177)	.6919 (.0142)	.6979 (.0171)
$1 - d^H$	.1298 (.014)	.6051 (.0235)	**.6331** (.0225)	.5809 (.0184)	.5917 (.0142)	.5874 (.0175)	.2102 (.0165)	.7159 (.0174)	**.7398** (.0148)	.7077 (.0164)	.7138 (.014)	.7137 (.0172)
$1 - d^{H^+}$	.2037 (.013)	.7172 (.0178)	**.7423** (.0173)	.7046 (.0124)	.7118 (.0095)	.6989 (.0134)	.2428 (.0163)	.7839 (.0153)	**.8044** (.0116)	.783 (.0131)	.786 (.0128)	.7813 (.0143)
s^{pre}	.537 (.0089)	.841 (.014)	**.8538** (.0119)	.8402 (.0094)	.8432 (.0057)	.7982 (.0081)	.6332 (.0077)	.8894 (.0146)	**.8945** (.0095)	.8885 (.0075)	.8904 (.0068)	.8635 (.006)
s^{pro}	.6666 (.0105)	.8762 (.0156)	.8886 (.0163)	.8644 (.0129)	.8686 (.0077)	**.9007** (.0092)	.6096 (.0151)	.8944 (.0191)	.9099 (.0128)	.8946 (.0133)	.8957 (.0135)	**.9178** (.0134)
$s^{pro \times pre}$	.3109 (.0101)	.7571 (.0155)	**.7779** (.0147)	.7426 (.0108)	.7511 (.0074)	.7344 (.0106)	.2964 (.0169)	.8056 (.014)	**.8256** (.0099)	.8044 (.0127)	.8077 (.0125)	.8012 (.0125)

The metamodels generated using all the presented machine learning algorithms reach at least 80% accuracy with respect to the $s^{pro \times pre}$ score on the dataset with empty baskets. This is confirmed by low Hamming distance and augmented Hamming distance values. The algorithms are also good at predicting the entire basket (see the s^z score). A s^z of 1 would mean that 100% of the unseen baskets are faithfully predicted. A perfect prediction in 70% of the cases is a good performance considering the fairly low number of features (six) considered in the input vector of the logs compared to the size of a three-dimensional scan in number of points. The standard deviation is low in all cases and the tendency on both the dataset with and without empty baskets is similar. We can see, by comparing the performance of the metamodels generated by the *KRR* algorithm to the one of the metamodels generated by the *KRR-NO* algorithm that the impact of possibly predicting unrealistic basket is fairly low. All metamodels generated using machine learning techniques significantly outperform the *MEAN* procedure.

5 CONCLUSION

We generated metamodels for sawing simulation using machine learning algorithms. We demonstrated how simple machine learning algorithms like k-nearest neighbors (*k-NN*) and decision tree (*DT*) could be used to fit metamodels to output not a single value, but a structured response. Then, we moved on to a more powerful and yet simple metamodel scheme generated by a random forest (*RF*) algorithm. All the metamodels based on these algorithms have the advantage of simplicity. Furthermore, they are guaranteed to produce a physically realistic basket of products meaning that the basket is the result of the transformation of at least one realistic input log. We proposed, as a final fitting method, a kernel ridge regression (*KRR*) approach. Our *KRR* algorithm can be used to exploit the structure of the response of the simulator to predict unseen logs, a feature we will be fully able to exploit by developing problem-specific kernels (especially for the output space). Finally, we developed useful problem-specific metrics along with a general one to evaluate our metamodels: the prediction ratio, the production ratio, and the augmented Hamming distance. The prediction and the production ratios can be aggregated as a single metric we called the production and prediction area. As a future work, we propose to tackle the converse problem using machine learning-generated metamodels. That is, to approximate the inverse of the black-box function of a sawmill simulator which would help in a context where one needs to determine which characteristics of a log are needed to output a given basket. The metamodels we generated using machine learning techniques achieved good performances on our datasets using only the six industrial standard features for a log description which turns out to be an impressive result considering the usual input size in sawing simulation involving a three-dimensional log scan and a detailed model of the plant.

REFERENCES

Bakĭr, G., T. Hofmann, B. Schoölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan. (Eds.) 2007. *Predicting Structured Data*. MIT press.

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2010. *Discrete-Event System Simulation*. Fifth ed. Upper Saddle River, NJ: Prentice Hall.

Barton, R. R., and M. Meckesheimer. 2006. "Metamodel-Based Simulation Optimization". *Handbooks in Operations Research and Management Science* 13:535–574.

Ben-Hur, A., and J. Weston. 2010. "A User's Guide to Support Vector Machines". In *Data Mining Techniques for the Life Sciences*, 223–239.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. New York: Springer.

Breiman, L. 2001. "Random Forests". *Machine Learning* 45 (1): 5–32.

Breiman, L., J. Friedman, R. A. Olshen, and C. J. Stone. 1996. *Classification and Regression Trees*. Belmont, CA: Chapman and Hall.

Cortes, C., M. Mohri, and J. Weston. 2007. "A General Regression Framework for Learning String-to-String Mappings". *Predicting Structured Data*:143–168.

Fawcett, T. 2004. "ROC Graphs: Notes and Practical Considerations for Data Mining Researchers". *Machine Learning* 31:1–38.

Fix, E., and J. L. Hodges Jr. 1951. "Discriminatory Analysis-Nonparametric Discrimination: Consistency Properties". Technical report, DTIC Document.

FPInnovations 2014. "Optitek 10". User's Manual.

Genin, P., A. Thomas, and S. Lamouri. 2007. "How to Manage Robust Tactical Planning with an APS (Advanced Planning Systems)". *Journal of Intelligent Manufacturing* 18 (2): 209–221.

Goulet, P. 2007. *Utilisation de la simulation du flux d'une scierie pour déterminer les gains en productivité associés au triage des tiges par dimensions*. Master's thesis, Université Laval, Québec, Canada.

Hara, S., R. Raymond, T. Morimura, and H. Muta. 2014. "Predicting Halfway Through Simulation: Early Scenario Evaluation Using Intermediate Features of Agent-Based Simulations". In *Proceedings of the 2014 Winter Simulation Conference*, 334–343.

Kleijnen, J. P. C. 2009. "Kriging Metamodeling in Simulation: A Review". *European Journal of Operational Research* 192 (3): 707–716.

Lindner, B. G. 2014. *Determining Optimal Primary Sawing and Ripping Machine Settings in the Wood Manufacturing Chain*. Master's thesis, Stellenbosch: Stellenbosch University.

Liu, C., S. Y. Zhang, A. Cloutier, and T. Rycabel. 2007. "Modeling Lumber Value Recovery in Relation to Selected Tree Characteristics in Black Spruce Using the Optitek Sawing Simulator". *Forest Products Journal* 57 (4): 57–65.

Lundahl, C. G. 2007. *Optimized Processes in Sawmills*. Licentiate thesis, Luleå University of Technology, Luleå, Sweden.

Nylinder M. and P. Friberg 2007. "The Sawmill Database". http://www.sawmilldatabase.com.

Macal, C. M., and M. J. North. 2009. "Agent-Based Modeling and Simulation". In *Proceedings of the 2009 Winter Simulation Conference*, 86–98.

Orbay, L. 1984. *Computer Simulation of Overseas Product Manufacturing Potentials of a Redesigned Multipass Headrig Mill in Coastal British Columbia*. (Ph.D.) thesis, University of British Columbia, Vancouver, Canada.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. "Scikit-Learn: Machine Learning in Python". *The Journal of Machine Learning Research* 12:2825–2830.

Poropudas, J., J. Pousi, and K. Virtanen. 2011. "Multiple Input and Multiple Output Simulation Metamodeling Using Bayesian Networks". In *Proceedings of the 2011 Winter Simulation Conference*, 569–580.

Salemi, P., B. L. Nelson, and J. Staum. 2012. "Moving Least Squares Regression for High Dimensional Simulation Metamodeling". In *Proceedings of the 2012 Winter Simulation Conference*.

Sampson, S. 1990. *Modélisation graphique du débitage du bois*. Master's thesis, Université Laval.

Talbi, E.-G. 2009. *Metaheuristics: From Design to Implementation*, Volume 74. John Wiley & Sons.

Tong, Q. J., and S. Y. Zhang. 2006. "Modelling Jack Pine Lumber Value Recovery in Relation to Tree Characteristics Using Optitek Simulation". *Forest Products Journal* 56 (1): 66–72.

Van Beers, W. C. M., and J. P. C. Kleijnen. 2004. "Kriging Interpolation in Simulation: A Survey". In *Proceedings of the 2004 Winter Simulation Conference*.

Verret, D. 1997. *Modélisation du flux des bois dans les usines de sciage*. Master's thesis, Université Laval, Québec, Canada.

Wang, G. G., and S. Shan. 2007. "Review of Metamodeling Techniques in Support of Engineering Design Optimization". *Journal of Mechanical Design* 129 (4): 370–380.

Wery, J., P. Marier, J. Gaudreault, and A. Thomas. 2014. "Decision-Making Framework for Tactical Planning Taking into Account Market Opportunities (New Products and New Suppliers) in a Co-production Context". In *10th International Conference on Modeling, Optimization & Simulation*.

AUTHOR BIOGRAPHIES

MICHAEL MORIN is a Ph.D. candidate in combinatorial optimization at the Department of Computer Science and Software Engineering of Université Laval. He was awarded the NSERC and the FQRNT post-graduate scholarships. He is a student member of three research groups of Université Laval (PPC, MONADE, and GRAAL) and of the FORAC Research Consortium. His email address is michael.morin.3@ulaval.ca.

FRÉDÉRIK PARADIS is a computer science student at the Department of Computer Science and Software Engineering of Université Laval. His email address is frederik.paradis.1@ulaval.ca.

AMÉLIE ROLLAND is an M.S. student in machine learning at the Department of Computer Science and Software Engineering of Université Laval. She is a student member of the machine learning research group of Université Laval (GRAAL). Her email address is amelie.rolland.1@ulaval.ca.

JEAN WERY is a Ph.D. student in industrial engineering at Laval University. He holds a Master's degree from National School of Wood Science and Timber Engineering (ENSTIB) of Lorraine University, in France. Previously, he had been working for a few years in the industry as a production engineer in Poland. His research interests include simulation and optimization applied to wood processing. He is currently a student member of FORAC Research Consortium and CIRRELT (Interuniversity Research Center On Enterprise Networks, Logistics and Transportation). His email address is jean.wery@cirrelt.ca.

JONATHAN GAUDREAULT is co-director of FORAC, a research consortium dedicated to the development of decision support systems for the forest products industry. Over the last ten years he has led a number of key projects related to operations planning and scheduling, bringing together ideas from the artificial intelligence and operations research fields. In 2012, Dr. Gaudreault and other researchers from the FORAC Research Consortium were recognized with the Brockhouse Canada Prize for Interdisciplinary Research in Science and Engineering, Canada's premier award for interdisciplinary research. His email address is jonathan.gaudreault@ift.ulaval.ca.

FRANÇOIS LAVIOLETTE is a professor at the Department of Computer Science and Software Engineering of Université Laval. He is the director of the Université Laval Big Data Research Center, and the co-director of the machine learning research group of Université Laval (GRAAL). He is member of the FORAC Research Consortium. His email address is francois.laviolette@ift.ulaval.ca.