Coupling digital simulation and machine learning metamodel through an active learning approach in Industry4.0 context

Sylvain Chabanet, Hind Bril El-Haouzi, Philippe Thomas

Coupling digital simulation and Machine Learning metamodel through an Active Learning approach in Industry4.0 context

Sylvain Chabanet[1], Hind Bril El-Haouzi[1], and Philippe Thomas[1]

[1]Université de Lorraine, CNRS, CRAN, F-88000 Epinal, France (e-mail: {sylvain.chabanet, hind.el-haouzi, philippe.thomas}@univ-lorraine.fr

Abstract

Although digital simulations are becoming increasingly important in the industrial world owing to the transition toward Industry 4.0, as well as the development of digital twin technologies, they have become increasingly computationally intensive. Many authors have proposed the use of Machine Learning (ML) metamodels to alleviate this cost and take advantage of the enormous amount of data that are currently available in industry. In an industrial context, it is necessary to continuously train predictive models integrated into decision support systems to ensure the consistency of their prediction quality over time. This led the authors to investigate Active Learning (AL) concepts in the particular context of the sawmilling industry. In this paper, a method based on AL is proposed to combine simulation and an ML metamodel that is trained incrementally using only selected data (smart data). A case study based on the sawmilling industry and experiments are shown, the results of which prove the possible advantages of this approach.

Keywords: Stream based Active Learning, K-Nearest Neighbors, simulation metamodel, Sawmill, Smart Data, Artificial Intelligence

1 Introduction

The use of simulation and digital prediction tools to sustain both the integrated design of production systems and factory management is of crucial importance when planning to transition toward an industry of the future framework (European Factories of the Future Research Association and others (2012)). On the one hand, market constraints and a highly competitive commercial environment increase the need for such tools, whereas on the other hand, the increased complexity of the production system induced by the development of Industry 4.0 makes it necessary for the production facility to stay predictable, analyzable, able to be simulated at all times, flexible and adaptable (figure 1). This should

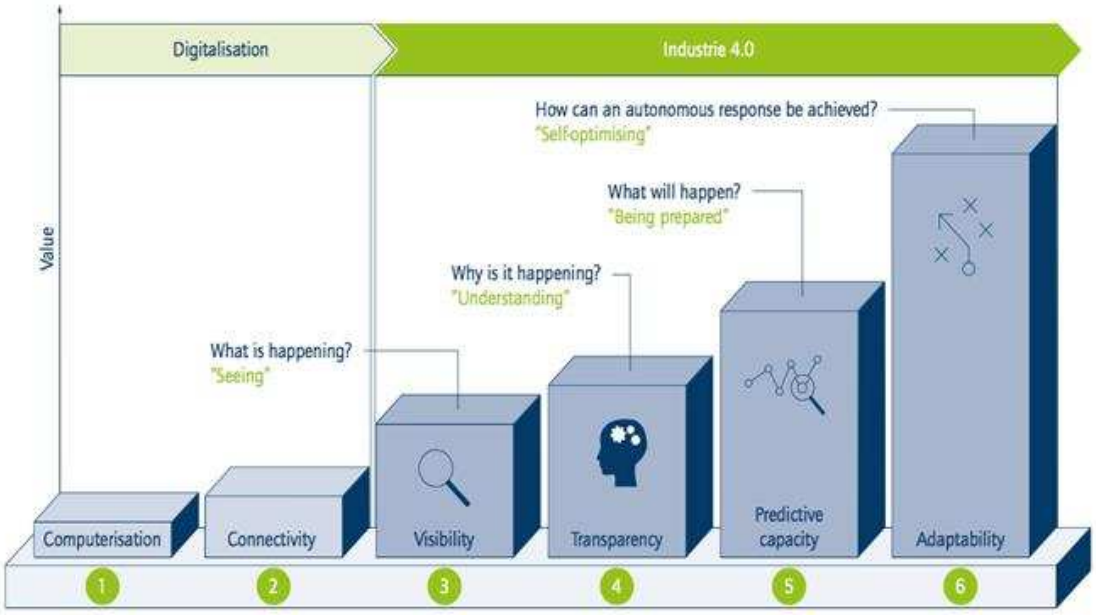

FIGURE 1: Development stages toward Industry 4.0. Source: FIR e.V. at RWTH Aachen University

be managed despite an increasing tendency to decentralization induced by the adoption of Cyber Physical Production System technologies (Monostori (2014)) and more generally by modular manufacturing systems (Rossgoderer (2015)). In addition to modularity, these systems introduce notions of connectivity, autonomy, and a dual view (physical and virtual).

Owing to these properties, as well as to the variety of virtual representations used in the life cycle of both products and processes, the co-simulation of smart factories has become increasingly credible. Authors such as Wu et al. (2015) demonstrate that this virtualization and IoT technologies will facilitate information sharing through a virtualization of a closed loop centered around the concept of digital twins (Bril El Haouzi (2017)).

Simulations are currently a widely accepted tool in the industrial world, and in particular, Masood and Sonntag (2020) identify it as a key Industry 4.0 technology with low complexity and high benefit for SMEs. Authors such as Salama and Eltawil (2018), however, have noted their ever-increasing computational costs and have thus proposed a coupling with predictive artificial intelligence metamodels, which can benefit from historical data and advances in Machine Learning (ML) technologies. Similarly, Thomas and Thomas (2011) proposed the use of a multilayer perceptron to reduce the simulation model of a sawmill workshop. In addition, Nesrine and Henri (2019) used a simulation to completely generate a dataset and train a neural network to drive decision making in the card based control system of a manufacturing installation. All of these studies demonstrate in different ways the interest of coupling simulations and ML models in an industrial context.

However, if unlabeled data are increasingly easy to gather, the labeled data necessary to

train ML metamodels can still be a limiting factor because their production often implies the intervention of an expert team or the use of a computationally intensive, gold standard simulation. This is even more problematic when multiple classifiers must be trained to correspond to multiple settings of the simulation or of the environment. A linked example, albeit not in a simulation context, was presented by Htike and Hogg (2016), who considered the problem of training multiple, context-dependent, pedestrian detectors needed for autonomous cars. Among other solutions, they pointed toward Active Learning (AL) as a promising research direction. This goal of efficiently using the data to train ML models is also linked with the transition of a paradigm from big data to smart data, which aims to leverage only useful data among the mass gathered by big data technologies (Iafrate (2014)).

Active Learning is a field of Machine Learning, and deals with the selection of a small subset of to-be-labeled individuals in an unlabeled database when the cost of labeling the entire base is intractable. However, despite showing excellent results on multiple benchmark datasets, Settles (2011) noted that Active Learning strategies had rarely been used in the industrial world. They, additionally, noticed that empirical studies with real annotators had mixed or negative results. They attribute this dichotomy to open challenges that were rarely studied by the literature. An example of such problem is to consider settings where the different classes in the classification problem are unknown. Recently, Abraham and Dreyfus-Schmidt (2020) have interpreted the mistrust of the industry toward AL application as being partially caused by the literature striving for ever increasing prediction scores, while the industry has a higher interest in robust methodologies, performing at least better than random strategies.

This paper explores a method, based on Active Learning concepts, to combine two main technologies associated to industry 4.0: a simulator, considered as exact in this study but computationally costly, and an ML classifier, less costly to use online but whose predictions are only approximations of the simulator. More precisely, this study considers a case inspired from the sawmill industry where sawing simulators are used to simulate the sawing of logs using their 3D scans, that is, point clouds sampled from the log surface, and an ML classifier that directly uses those scans to predict the results of the simulator. This method is tested on a dataset provided by the Canadian forest wood industry, which contains a particularly large number of labels with respect to the number of instances. Three elements stemming from an industrial setup are important to consider when designing an Active Learning strategy (figure 2). The first is the way it accesses the unlabeled data. Is there an existing massive database or are the data generated by a stream? The second problem is the actual classification problem that one wants to solve. Is it a binary or multi-class problem? In addition, are the input data structured? The third is the notion of budget, that is, which proportion of the unlabeled instances can be labeled. This proposed workflow outline general principles that should be considered in any AL study and will guide us to design the solution implemented in our simulated experiments. The remainder of this

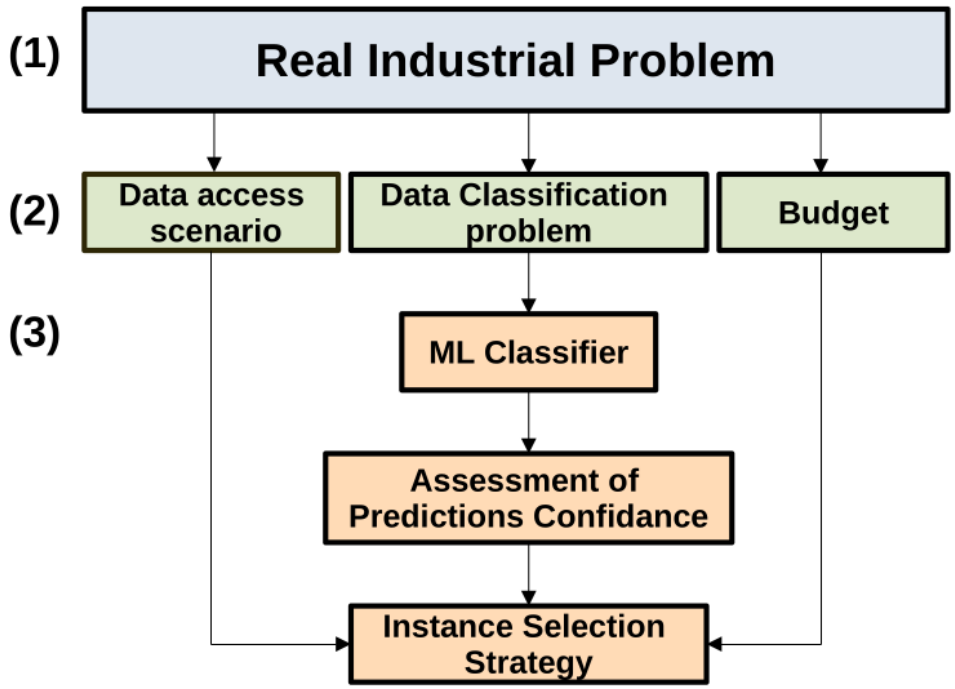

FIGURE 2: Proposed workflow

paper is structured as follows. Section 2 discusses the industrial context along with previous studies on the development of ML metamodels for sawing simulators. Section 3 provides an overview of the Active Learning concepts presented in figure 2. The AL method studied in this paper is then described in Section 4. Section 5 provides details about the experimental setup, the results of which are presented in Section 6. Finally, section 7 provides some concluding remarks along with some perspectives regarding this research.

2 Sawing simulation in the wood industry

Sawmills process logs of varied lengths and shapes to produce lumber. This process is described as divergent with a co-production because several products, each with potentially different dimensions and grades, are produced at the same time from the sawing of a single log (figure 3).

Owing to these characteristics, predicting in advance the set of lumber (called a basket of products in the following) resulting from the sawing of a specific log, is a difficult problem. This complicates the optimization of operational or strategic planning, as well as the running of simulations needed to make long-term decisions, such as the remodeling of a sawmill, or simply the acceptation of a command containing unusual products, since acceptation of such a command would require the modification of mill setup and impact the whole mix of lumber produced in a way difficult to predict (Wery et al. (2018)). However, the wood

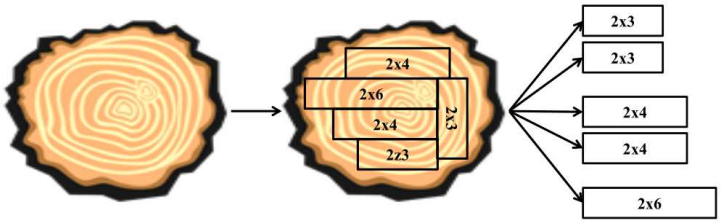

FIGURE 3: Breaking a log into a set of lumber. Morin et al. (2015)

industry has several sawing simulation tools at its disposal to tackle this issue. Examples of such simulators include Optitek (FPInnovation), SAWSIM (HALCO software systems ltd.) and Autosaw (Todoroki et al. (1990)). Considering 3D scans of the exterior shape of the logs, these simulators are able to digitally break it into a basket of products. However, depending on the simulation setting required and the log considered, simulators may need from a few seconds to 3 hours or more CPU time to determine the optimal breaking of a single log, which renders them impractical in cases in which a large number of simulations are needed. As an example of such an issue, Morneau-Pereira et al. (2014) proposed a mixed integer programming (MIP) model using Optitek simulation results on a few sampled logs to optimize the allocation of logs from cutblocks to sawmills. They reported that, although the MIP can be solved within a few seconds, the simulation itself takes several hours.

Considering this problem, Morin et al. (2015) proposed replacing the simulators with Machine Learning metamodels, which learn from the results of previous simulations and are able to approximate the simulator output. They used six features describing the logs to train several ML models, including random forest and k-Nearest Neighbor (kNN) classifiers. In Morin et al. (2020), the same authors used those classifiers to generate data later applied in a MIP model to optimize the log allocation from cutblocks to sawmills.

However, considering that these ML models use only six features describing the logs while the simulator makes use of a full 3D scan, Selma et al. (2018) and Chabanet et al. (2021) proposed an algorithm based on the iterative closest point algorithm (ICP) to compute the similarity between log scans. These similarities were then used in a kNN classifier.

Whereas previous studies on both structured and unstructured data offer interesting perspectives in terms of the computing time and prediction scores, they consider disposing from the start of a labeled database used to train an ML model with the objective of fully replacing the simulator. However, such ML metamodels are only applicable to a single sawmill using one configuration. Its efficiency also depends on the input log distribution staying stationary over time. Therefore, it is preferable to couple the simulator and the ML classifier using AL concepts to continually improve the latter while alleviating the computational cost of the former.

3 Literature review on Active Learning

In this section, Active Learning concepts and scenarios common in the literature are reviewed, along with popular sampling strategies.

3.1 Definition

Active Learning can be defined as a field of artificial intelligence that considers the problem of selecting only a few individuals from an unlabeled database. These instances are then iteratively labeled and used to train an ML model, that is, a function that can yield a prediction $\hat{y}$ for the label of an instance x. The main motivation is that, while huge, unlabeled databases have become increasingly easier to produce with the development of big data technologies, creating labeled databases remains time consuming and costly in some cases, particularly when the labeling process involves human experts or computationally intensive simulations. However, in the AL framework, only a few specific individuals are selected to be labeled by an expert, called an Oracle, to train or improve an ML classifier.

From our reading, an efficient AL strategy should be able to train such a classifier so that it is nearly as good as the model trained on the whole database by using only a fraction of the available data. Here, "nearly as good" refers to the occurrence of an expected error. Considering a loss function $l(y, \hat{y})$ that measures the cost of making an erroneous prediction $\hat{y}$ instead of the true label y, an AL strategy should be able to train a predictive model h_{AL} such that:

$$\mathbf{E}_{(x,y)}[l(y, h_{AL}(x)] < (1 + \epsilon)\mathbf{E}_{(x,y)}[l(y, h_{ML}(x)]\,, \tag{1}$$

with $\mathbf{E}$ the probabilistic expected value of the loss l over new data instances x with label y, h_{ML} a predictive model trained on the whole dataset and ϵ a small value.

3.2 Data access scenarios

Various studies have explored several ways in which an Active Learning engine can gain access to a dataset, which are referred to as data access scenarios in this study for easy differentiation from other AL concepts. Occasionally, however, they are also simply called Active Learning scenarios in the literature. As identified in Kumar and Gupta (2020), the data access scenario considered in the literature can be classified into three categories: pool-based, stream-based, and membership query synthesis.

- *Pool-based Active Learning* (figure 4) occurs when the AL engine has access from the start to a huge base of unlabeled data, and potentially of a small set of labeled data. This database is not supposed to evolve, except for the addition of labels to existing instances. Here, the AL engine goal is to select only a subset of the unlabeled data to be labeled by the Oracle and included in the labeled database. This labeled

database is simultaneously used by the ML engine to train and improve the ML model. For example, Rastogi and Sharma (2019) propose to carefully select points from a database to label them and train a Laplacian twin support vector machine, which is a classifier able to learn from both labeled and unlabeled data. Similarly, Martinez Arellano and Ratchev (2019) propose an application of pooled-based active learning in an Industry 4.0 context. They study the use of Bayesian Neural Networks for tool wear detection. The uncertainty of the prediction is, interestingly, measured using Bayesian inference over the network parameters.

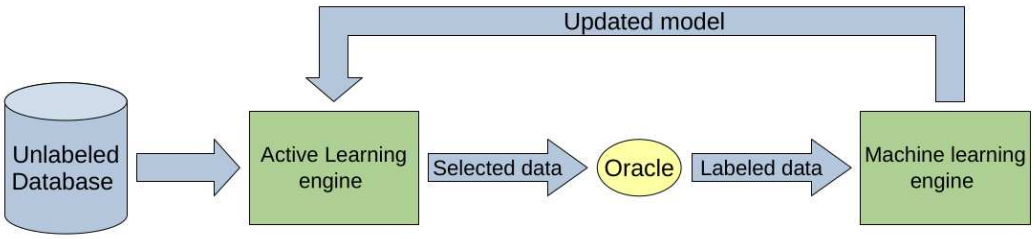

FIGURE 4: Explanatory diagram of pool-based Active Learning scenario

- Contrary to what occurs in a pool-based case, *stream-based Active Learning* (figure 5) strategies do not consider having access to the whole unlabeled database at once. The new data instances arrive iteratively from an unlabeled data stream. The goal of the AL engine considering the current state of the ML model is to then decide whether to call the Oracle or not.

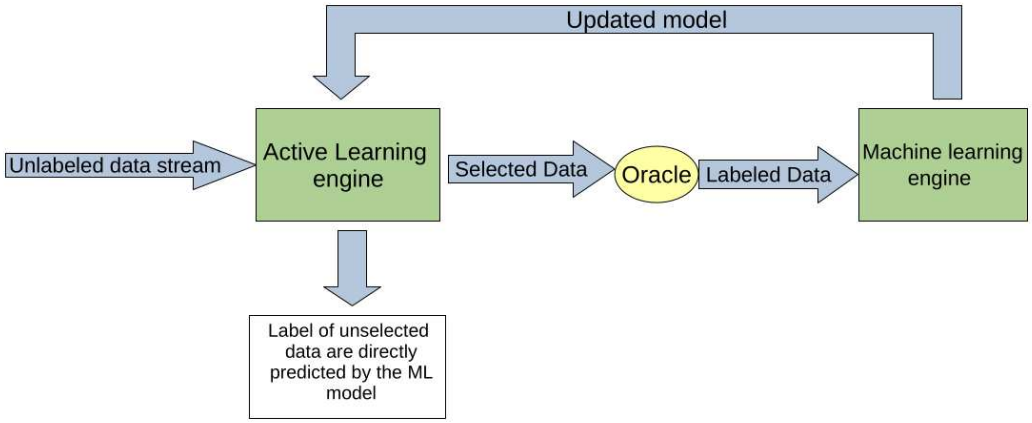

FIGURE 5: Explanatory diagram of the stream based Active Learning scenario

For example, Lughofer et al. (2015) proposed an active learning application based on fuzzy classifiers to classify visual defects on micro fluid chips. An important point of their study is the ability of the proposed model to incorporate on the fly new classes, that is, new type of defects never observed before. More recently, Lowrance and

Lauf (2019) proposed a comprehensive framework for developing the online Active Learning of the wireless connection quality for mobile robots.

- In *membership query synthesis based Active Learning* (figure 6), no stream or existing database is used. The AL engine generates query instances that are labeled by the Oracle. This framework was first introduced by Angluin (1988), who considered the problem of identifying an unknown subset from a larger universal set. However, a disadvantage of this strategy is that it may be difficult for human experts to classify artificially generated instances. To solve this issue, Wang et al. (2015) proposed a framework merging a membership query synthesis scenario and a pool-based scenario, where the AL engine generates an artificial query, but the one being labeled is its nearest neighbor in a real database.

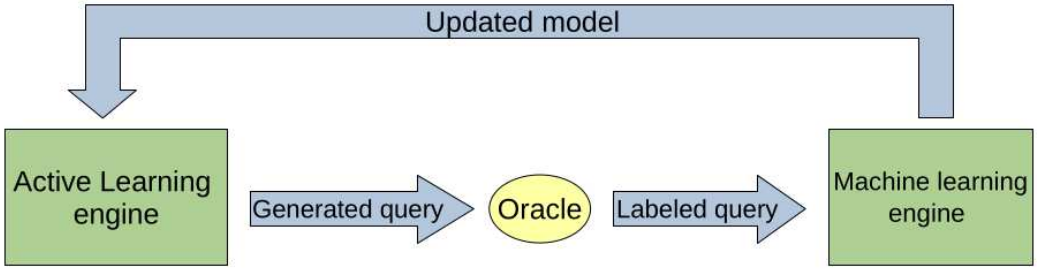

FIGURE 6: Explanatory diagram of the query synthesis

In addition to these scenarios, the AL process can either start from a small subset of already labeled data or with no labeled data. The latter case is referred to as cold start.

As shown in figure 2, the determination of which scenario to use is highly dependent on the application context. Pooled-based strategies are suitable when a huge unlabeled database is already present and no concept drift or concept shift is expected. Concept drift refers to the emergence of a new class or a change in the data generation process slowly over time, while concept shift refers to a brutal change in the data. In such a case, the predictor can be trained off line at one point in time and used consistently with no loss of prediction accuracy.

However, if concept drift is expected, if new unlabeled data are continuously generated, or if a single pass over the dataset is considered preferable due to computational limitations, a stream-based strategy might be preferable. This study focuses on a stream-based scenario, since it is more generally applicable than the pool-based case. However, while concept drift is to be expected in the forest-wood industry on which this study is based, no drift or shift was naturally present on the studied dataset. Therefore, in the last set of experiments presented in this paper, concept shift has been simulated to assess its effect on the proposed solution.

Lastly, a membership query synthesis might be an interesting choice when even unlabeled data are difficult to gather. However, in the case of a human the Oracle, a hybrid strategy is preferable.

3.3 Budget

As emphasized in Settles (2011), the notion of budget is crucial for an Active Learning strategy to be practical and accepted in an industrial context. What defines the budget and the strategies proposed to avoid overspending may vary depending on the data access scenario and industrial setup being considered. In a pool-based scenario, the budget is often expressed as the maximum number of calls to the Oracle. A popular strategy, as used in Gikunda and Jouandeau (2020), for example, is to iteratively select instances from the unlabeled dataset, which maximize some criteria and adds them to the labeled dataset until the budget is exhausted. In a stream-based scenario, it might be argued that a budget over time, that is, a maximum proportion $b \in [0, 1]$ of the stream instances to be selected per time period, is more practical than a fixed budget. Indeed, if the stream is infinite but the budget is fixed, the AL process will stop at some point in time when the budget is exhausted, whereas the stream will continue indefinitely, rendering the adaptation of the ML classifier to structural changes in the stream difficult. In Kottke et al. (2015), such a budget is considered and the use of an adaptive threshold is proposed on some computed quantity, which ensures that the Oracle budget is met within some tolerance window.

In addition, depending on the chosen ML classifier, an increase in the labeled database size might also increase its training time (e.g., random forest classifier) or prediction time (e.g., kNN). It might then be necessary to limit the size of the labeled database by dropping the least interesting instances. In Kurlej and Woźniak (2011), such a design is implemented by systematically dropping the oldest labeled instance, with the idea of helping a kNN model adapt to sudden drifts in the data stream.

3.4 Overview of AL methods

Once the scenario, budget constraints, and classification problem have been identified, it is necessary to develop an appropriate sampling strategy to select a subset of the available instances. As shown in Figure 2, the final choice of this strategy should be motivated by those characteristics. In an industrial context, it might also be important to choose an *interpretable* model.

Numerous AL strategies have been reported in the literature. For example, uncertainty sampling methods, introduced in Lewis and Gale (1994), is a natural heuristic for performing AL when the ML classifier output can be interpreted as a probability distribution over the labels given the features. An AL engine using uncertainty sampling queries instances that are more uncertain, that is, when $\max_y p(y|x)$ is below a certain threshold. For example, Schmidt et al. (2016) propose an application in text classification. An unlabeled document

is first classified by a precise but fixed SVM ensemble, which attributes a probability to each of the document possible classes. If, however, the most probable class for the document is under a *decision confidence threshold*, the document is classified by a single SVM which is regularly trained on a database built using active learning principles. When this class probability is additionally under a *annotation confidence threshold*, the document is added to this classifier training database. A natural variant of uncertainty sampling, margin sampling, is used in Schein and Ungar (2007) for example. Consider y and y' as the most probable and second-most probable classes for x. The margin is defined as $p(y|x) - p(y'|x)$, and the instances sampled are those with the lowest margin.

Similarly, query-by-committee methods were introduced in Seung et al. (1992). This method maintains a committee of several ML models and queries the Oracle in case of disagreement among its members. Several methods have been used in the literature to measure disagreement among committees. According to Settles (2009), two popular methods are vote entropy and Kullback-Leibler divergence. A more straightforward methodology, used for example in Rajpathak et al. (2020), is to train several classifiers, 8 in their case, and to select instances leading to tie in the vote, that is, in this example, instances with 4 vote for each class.

Whereas the former methods can be said to search for *informative* data points, other methods search for *representative* points. For example, in Wu et al. (2006), an AL strategy combining several criteria are proposed to measure the interest in selecting an instance, including a measure of representativeness pushing for the selection of points in a dense area of the input data space. Similarly, Wang et al. (2017) proposed an algorithm to build clusters from an unlabeled database and select representative points for each cluster.

In Lughofer (2017), a family of methods based on soft computing is identified, which has been gaining attention during the last few years. Similar to uncertainty sampling methods, these strategies aim to quantify prediction uncertainty, but do not necessarily use the ML model outputs to do so. This allows them to consider other sources of uncertainty. For example, Kumar and Halder (2020) proposed an Active Learning strategy based on fuzzy rough logic for the classification of cancer samples.

Whereas most of the above methods propose a heuristic to select the points, the ultimate goal of the AL engine is to select data to reduce the prediction error of the ML model being trained, as defined in equation 1. This is a difficult task, however, as it requires an accurate estimation of the output probabilities without access to a test set of labeled instances Chapelle (2005). In addition, as described in Dasgupta (2011), the Active Learning engine may induce sampling bias in the labeled database, which means that the data distribution in this database does not reflect the real data distribution. To bypass these problems, several authors have proposed the use of Bayesian inference and sample weighting. See Kottke et al. (2020) for an example of a pool-based model and Mohamad et al. (2018) for a stream-based model.

In addition, an AL strategy should consider at least two sources of uncertainty. In

Lughofer (2012), conflict and ignorance concepts are presented:

- A conflict occurs when an instance appears near the decision boundary between classes.
- Ignorance occurs when an instance appears in a region of the input space that has not been sufficiently explored or has not been explored at all.

In Bondu et al. (2010), an interpretation of similar concepts is proposed under the exploration-exploitation framework. The AL engine should compromise between sampling from unknown areas of the input space, that is, *exploration*, and sampling from an area already known to increase the ML classifier performance there, that is, *exploitation*. A model performing only exploitation might be extremely precise on a small area of the input space and extremely imprecise everywhere else, whereas a model performing only exploration might miss areas of the input space where the classifier performs poorly and where more sampling is necessary.

4 Proposed method

Even if the proposed approach considers a particular problem from the sawmilling industry, a generalization of that problem can be made, with minimal assumptions on the data structures and class labels.

The data access scenario considered here is a stream-based one. Instances sampled from a possibly ill-defined space $\mathbf{B}$ arrive one by one from a stream, and an AL engine has to decide whether to call the Oracle and add the instances to a labeled database used to train an ML classifier. The instances from the set $\mathbf{B}$ are not supposed to have any structure, and therefore, can only be compared using a dissimilarity function, that is, a function $d : \mathbf{B} \times \mathbf{B} \mapsto \mathbf{R}$, which intuitively measures the difference between two instances. No other assumption was made for the dissimilarity function. In particular, neither symmetry nor triangular inequality, which are properties of a valid distance, are required. In addition, there is no consideration of having prior knowledge on the set Y of possible classes, neither on its size nor on the class probability distribution.

In this study, the Oracle is a simulator. Therefore, the instances whose classes were attributed by the Oracle are considered to be the gold standard and stored in a database used to further improve the ML model. A k-nearest neighbor classifier was used. A kNN is well suited for the problem considered in this study because it allows the direct use of a non-metric dissimilarity, and do not considers the extraction of a context dependent vector of features.

A second advantage of kNN is that this algorithm belongs to the family of instance-based classifiers. This means that it does not require any training phase, but directly uses a database of known examples to make a prediction. Hence, the task of the ML engine is only to add the instances selected by the AL engine to this example database. This also means,

however, that the computational cost of making a prediction increases with the size of the example database.

A last noticeable advantage is that a kNN classifier does not require prior knowledge of the number of classes or their probability distribution. This is useful in the setup described in the present paper because it does not consider any prior knowledge on the composition of realistic baskets of products before running the simulation.

However, considering that this kNN classifier makes use of dissimilarities, which in some cases can be relatively costly to compute, there is a real need to limit the growth of the example database.

4.1 The kNN model

The kNN rule (Fix (1951)) is well known and commonly used in the industry owing to its simple implementation and overall results. Given a set of labeled instances and a new instance x with an unknown label, the kNN rule searches for the k instances $(x_1, ..., x_k)$ that are more similar to x, that is, the k instances with minimal dissimilarity. The class predicted for x is as follows:

$$\hat{y} = \underset{y \in Y}{\operatorname{argmax}} \frac{1}{k} \sum_{i=1}^{k} \mathbf{1}_{\{y_i = y\}}\,, \tag{2}$$

where $\mathbf{1}$ is the indicator function and $(y_1, ..., y_k)$ are the respective labels of $(x_1, ..., x_k)$. Therefore, the kNN rule predicts the most frequent class among k instances that are most similar to x.

It is expected, however, that instances farther from the query will have a lower probability of sharing its class than the closer ones. Some authors, for example Dudani (1976), rectify this by weighting the instance votes with a function decreasing with the dissimilarity. In this study, a similar approach was explored, which only considers a neighbor if its dissimilarity with x is below some threshold t_{nn}. Such neighbors are later referred to as *acceptable*. The prediction $\hat{y}$ then becomes the following:

$$\hat{y} = \underset{y \in Y}{\operatorname{argmax}} \frac{\sum_{i=1}^{k} \mathbf{1}_{\{d(x,x_i) \leq t_{nn} \text{ and } y_i = y\}}}{\sum_{i=1}^{k} \mathbf{1}_{\{d(x,x_i) \leq t_{nn}\}}}\,, \tag{3}$$

where d is the dissimilarity. Therefore, the indicative function $\mathbf{1}_{\{d(x,x_i) \leq t_{nn}\}}$ as a weight can be seen as an extreme case of weighting scheme.

In this study, the dissimilarity introduced in Selma et al. (2018) is used. Figure 7 shows the ratio of logs sharing the same basket as a function of this dissimilarity. As expected, this ratio decreased rapidly with increasing dissimilarity.

In the remainder of this paper, the threshold t_{nn} is tuned to 250, which is approximately the value at which the ratio is lower than 0.5.

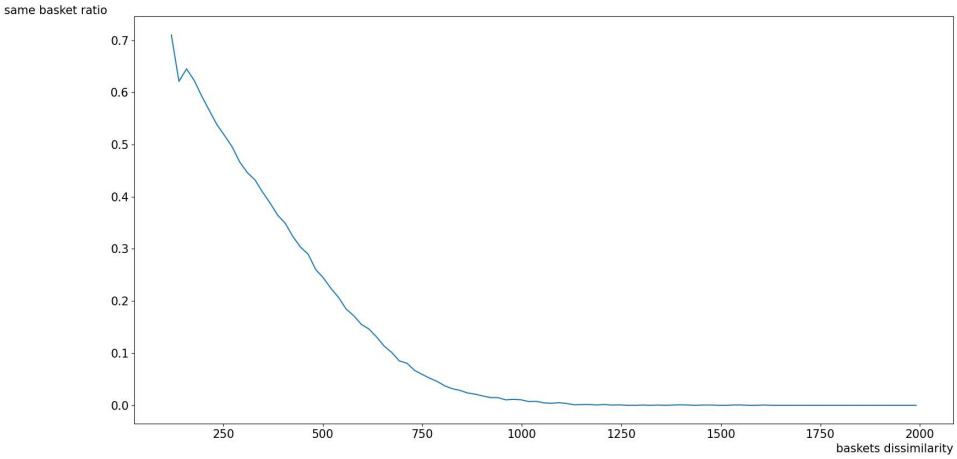

FIGURE 7: Ratio of pair of logs having the same basket, as a function of the ICP dissimilarity between the logs

Furthermore, as the specificity of this dataset, each log belongs to one of three categories depending on its length. These categories are different from the class labels. Therefore, the neighbor search is restricted to logs from the same length category only.

When the set of acceptable neighbors is empty but for some reasons a prediction is still needed for x, the modified kNN rule falls back to a simple 1NN rule, that is, the predicted basket is the basket of the known instance that is more similar to the query x. The conflict measure used in this paper as a measure of the predictor uncertainty cannot, indeed, take more than k values. k needs, therefore, to be high enough to allow this measure to differentiate multiple situations. However, the size of the example database used by the kNN model to make its prediction is expected to remain small in our experiments. Therefore, the value of k should be chosen to remain small compared with the size of this database. No further optimization of k had been made. More experiments, however, have been done, changing the value of k to 5 or 15. This had very little impact on the overall results. Consequently, in the following, the parameter k is fixed to 10 as a reasonable value considering the expected size of the built database and the way the prediction confidence is measured.

4.2 Prediction confidence assessment

The kNN algorithm is not, by nature, probabilistic. However, numerous heuristics have been proposed for measuring the uncertainty of its output. For example, Delany et al. (2005) proposed 5 confidence measures, and Cheetham and Price (2004) proposed 12.

In this study, two measures of uncertainty were chosen, one adapted to measure the *ignorance* and the other to measure the *conflict*.

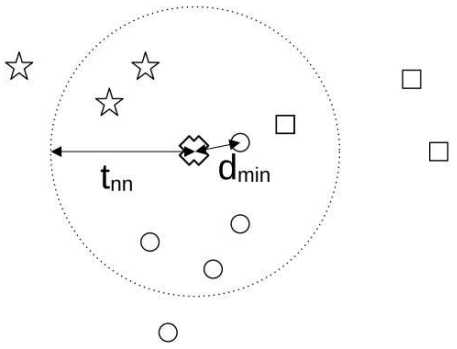

FIGURE 8: Example of an application of the AL strategy.

The ignorance measure is the inverse of the distance from x to its nearest neighbor in the example database.

$$u_{ign}(x) = \frac{1}{\min_{1<i<k}(d(x, x_i))}, \tag{4}$$

The conflict measure is defined as the margin between the two most frequent classes among the k-Nearest Neighbors of x used by the modified kNN classifier, that is, those with a dissimilarity lower than t_{nn}. More precisely, first consider p_1 as follows:

$$p_1 = \frac{\sum_{i=1}^{k} \mathbf{1}_{\{y_i=\hat{y} \text{ and } d(x,x_i)<t_{nn}\}}}{\sum_{i=1}^{k} \mathbf{1}_{\{d(x,x_i)<t_{nn}\}}}. \tag{5}$$

Similarly, consider p_2 as the score of the second most frequent class:

$$p_2 = \max_{y \in Y \setminus \{\hat{y}\}} \frac{\sum_{i=1}^{k} \mathbf{1}_{\{y_i=y \text{ and } d(x,x_i)<t_{nn}\}}}{\sum_{i=1}^{k} \mathbf{1}_{\{d(x,x_i)<t_{nn}\}}}, \tag{6}$$

The conflict measure is as follows:

$$u_{conf} = p_1 - p_2 \tag{7}$$

This measure is used as follows. Consider two thresholds t_{ign} and t_{conf} (these thresholds can either be fixed in advance based on prior knowledge or adjusted over time to ensure that the budget is respected). When $u_{ign}(x) < t_{ign}$ or $u_{conf}(x) < t_{conf}$, the Oracle is called for the new instance x, and the result is added to the example database. If the Oracle is not called, the class predicted for x is given by the modified kNN presented in the previous section.

Figure 8 presents an example of the threshold usage. The query instance is represented by a cross and is surrounded by instances from three classes, circle, square, and star. Among these instances, seven are acceptable, i.e., have a dissimilarity lower than the threshold t_{nn}. Here, d_{min} is the minimum dissimilarity among the neighbors. The two more frequent

Algorithm 1 Application of the use of the ignorance and conflict measures to decide whether to send a new instance to the Oracle.

Input New instance x, a database $\mathbf{B}$
Output toAdd, Boolean, whether to add x to $\mathbf{B}$ or not

kNearestNeighbors ← list of k elements in $\mathbf{B}$ most similar to x
acceptableNeighbors ← ∅
toAdd ← FALSE
for nn in kNearestNeighbors **do**
  if Dissimilarity(x, nn) < t_{nn} **then**
    add nn to acceptableNeighbors
  end if
end for
$u\_ign \leftarrow u_{ign}(x, \mathbf{B})$
$u\_conf \leftarrow u_{conf}(x,$ acceptableNeighbors$)$
if $u\_ign < t_{ign}$ OR $u\_conf < t_{conf}$ **then**
  toAdd ← TRUE
end if
return toAdd

classes are circle and star, with a margin of $\frac{4}{7} - \frac{2}{7} = 0.286$. The cross query is sent to the Oracle if and only if $0.286 < t_{conf}$ or $\frac{1}{d_{min}} < t_{ign}$.

4.3 Budget

This study runs four set of experiments. The first two sets of experiments do not consider any notion of budget and use fixed ignorance and conflict thresholds. However, this would be considered unsatisfactory in most industrial contexts. As previously described, budget management is, indeed, particularly important in such a case. The third and fourth set of experiments will, therefore, introduce a simulation budget, which will also be considered in the last set of experiments assessing the impact of concept shift. More precisely, the coupling of an ML classifier and a simulator requires to take into consideration the maximum simulator capacity. Inspired by the methodology presented in Kottke et al. (2015), the following extension is proposed to ensure that the budget will not be exceeded, that is, that no more instances will be sent to the simulator-oracle than can be simultaneously managed. Consider b_t as the ratio of free slots in the simulator at time step t. The thresholds t_{ign} and t_{conf} will be adjusted such that, for an incoming instance x, the probability of sending it to the Oracle is lower than b_t; that is, $\mathbf{P}(u_{ign}(x) < t_{ign}$ or $u_{conf}(x) < t_{conf}) < b_t$. Therefore, when no free slot is available, $b_t = 0$ and the probability of sending a new instance to the Oracle is null. However, it is also desirable to keep $\mathbf{P}(u_{ign}(x) < t_{ign}$ or $u_{conf}(x) < t_{conf})$ sufficiently high such that the simulator does not remain unused. Multiple choices of t_{ign} and t_{conf} satisfy these constraints. To select only one, a hyperparameter $\alpha \in [0, 1]$ is introduced, which can be seen as a measure of the relative importance accorded to the

ignorance measure, compared with the conflict measure. The thresholds t_{ign} and t_{conf} are then chosen as the quantiles at level αb_t over the distribution of u_{ign} and at level $(1-\alpha)b_t$ over the distribution of u_{conf}. Then, by definition of the quantile,

$$\begin{aligned}\mathbf{P}(u_{ign}(x) < t_{ign} \text{ or } u_{conf}(x) < t_{conf}) &\leq \mathbf{P}(u_{ign}(x) < t_{ign}) + \mathbf{P}(u_{conf}(x) < t_{conf}) \\ &\leq \alpha b_t + (1-\alpha)b_t \\ &\leq b_t.\end{aligned} \tag{8}$$

As in Kottke et al. (2015), the quantiles were estimated on a sliding window on the past data from the stream. It is expected that this estimate may be biased, as the distributions of u_{ign} and u_{conf} are expected to vary over time with the addition of new instances to the example database. However, this strategy ensures that the simulator will not be overloaded.

However, it should be noted that the conflict measure u_{conf} can only take values in a finite set. Considering that the conflict threshold being defined has a quantile that often takes one of those values, the strict inequality comparison seemingly leads to an under sampling of the stream, particularly for small values of α. This behavior was corrected using the following scheme. When $u_{conf}(x) = t_{conf}$, consider $dp = (1-\alpha)b_t - \mathbf{P}(u_{conf}(x) < t_{conf})$. The instance x is sent to the simulator according to the value of a Bernoulli random variable of parameter dp.

An overall summary of the complete proposed method is presented figure 9. In this figure, the budget is represented as a number of free slots in the simulator, or free capacity c_t. The two notions are linked by the formula $b_t = \frac{c_t}{N_{slots}}$ where N_{slots} is the maximal capacity of the simulator.

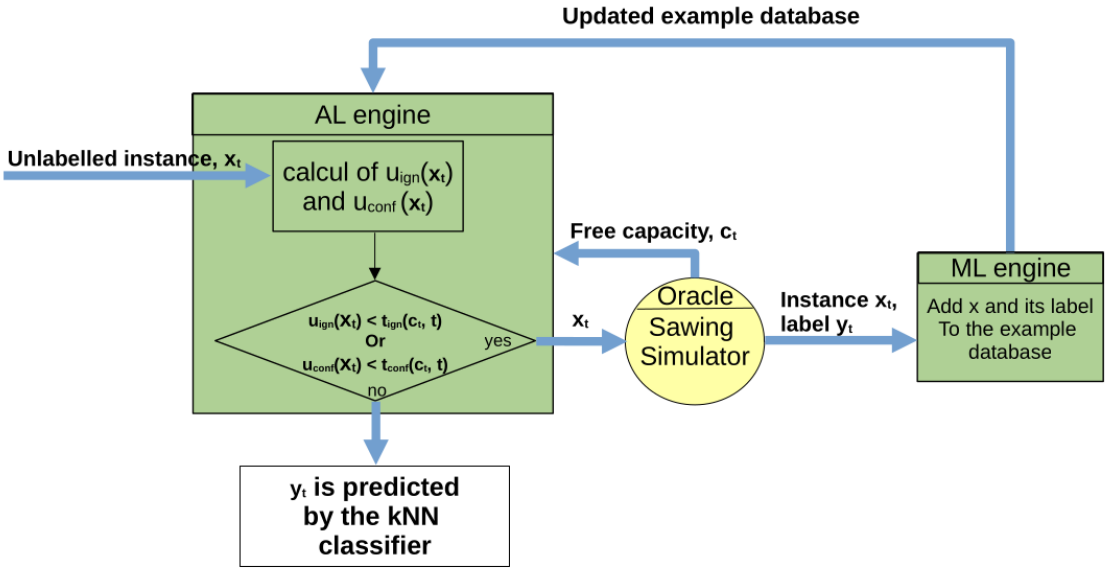

FIGURE 9: Outline summary of our proposed method

5 Experiments setup

The strategy described in the previous section was simulated using the programming language Python, with a 3D scan dataset supplied by the Canadian wood industry. The following is a detailed description of the database and an evaluation of the scores.

5.1 Log database

The database used in this paper contains 1207 3D scans of logs (figure 10), as well as their associated baskets of products simulated by the software Optitek. As said earlier, these logs can be separated into three length classes. The *very short logs* have a length under 2.48 m. *shorts* logs are defined as having a length between 2.48 m and 3.5 m. The vast majority of them, however, have a standard length of 2.5 m. Similarly, *long* logs are defined has having a length above 3.5 m and have, in nearly all cases, a standard length of 5 m. The 19 different types of lumber appearing in the database are numbered from 1 to 19, and each basket is represented by a vector of length 19, where the element in position i is the amount of product i resulting from the breaking of the log. It should be noted that, because no basket contains more than five different kinds of products, such vectors are sparse.

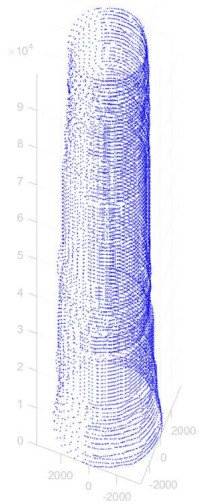

FIGURE 10: Example of a 3D scan of a log.

The base contains up to 105 different types of baskets, including 72 that appear only once because of their rarity and the size of the database. Each basket was considered a class in the classification problem. Because our goal is to create a metamodel of the original simulator, that is, the Oracle, these rare labels are not considered as outliers.

The scans are composed of a succession of ellipsoids, which together sample the log surface. However, some scans initially contained missing sections, that is, parts without ellipsoids. These sections were filled by repeating the ellipsoid that immediately preceded them.

5.2 Evaluation scores

To evaluate and compare the performances of the different ML classifiers, a score must be defined. In Morin et al. (2015), the prediction-production score is specifically introduced for the problem of evaluating ML models that predict the log baskets of products.

Consider y as the real basket of products of a log, and $\hat{y}$ as the predicted basket of products. Because both y and $\hat{y}$ are sparse, including all $(0, 0)$ real/predicted pairs would bias the score optimistically. Hence, all such $(0, 0)$ values are removed. The new length of the vectors is denoted by p.

The production score is introduced as follows:

$$s^{pro} = \frac{1}{p} \sum_{i=1}^{p} \min(1, \frac{y_i}{\max(\epsilon, \hat{y}_i)}), \tag{9}$$

where y_i and $\hat{y}_i$ are the i^{th} components of vectors y and $\hat{y}$, respectively, and ϵ is a small value to avoid dividing by zero.

The prediction score is similarly defined as follows:

$$s^{pre} = \frac{1}{p} \sum_{i=1}^{p} \min(1, \frac{\hat{y}_i}{\max(\epsilon, y_i)}). \tag{10}$$

The production score can be seen as the proportion of the prediction that is effectively produced, whereas the prediction score is the proportion of the production that is correctly predicted.

The prediction-production score is then naturally defined by the following:

$$s^{pre \times pro} = s^{pre} \times s^{pro}. \tag{11}$$

The evolution of the kNN performance is monitored as the stream progresses using the accumulated $s^{pre \times pro}$: $AS^{pre \times pro}$. This accumulated score allows following the evolution of the performance profile of the classifier with respect to the stream advancement, as recommended in Lughofer (2017) for short stream simulations.

The value of $AS^{pre \times pro}$ is initially set to zero. Then, whenever a new log appears in the stream, a prediction $\hat{y}$ is always performed using the current example database, before potentially completing this database with the new log depending on the result of the AL rules.

The accumulated score is then updated using the following formula:

$$AS^{pre \times pro} = \frac{AS^{pre \times pro}(i-1) + s^{pre \times pro}}{i}, \tag{12}$$

with i being the number of previous updates.

When not working under budget, $AAS^{pre \times pro}$ is introduced as a version of the accumulated score and is only updated when the log is not sent to the Oracle and is therefore

not added to the database. In this case, the growth of the database was also monitored. Similarly, when working under budget, $BAS^{pre \times pro}$ is introduced, which is either updated with 1 when the log is sent to the Oracle, and $s^{pre \times pro}$ when it is not. A strategy with a higher $BAS^{pre \times pro}$ is better at dispatching incoming instances between the simulator and the ML classifier under budget constraints.

The second performance measure considered in this study is the number of classes present on the example dataset at the end of the stream. This measure is not directly linked to the prediction score. It is, however, a measure of the effectiveness of the exploration of the input data space by the AL strategy. This stems from the idea that a strategy with a better exploration is expected to be, in general, more robust against a potential concept shift. Such a measure is considered in Mohamad et al. (2018) under section *class discovery performance*.

6 Results

In this section, four sets of experiments are described. First, no budget restrictions are considered. Therefore, the thresholds are set to fixed values. The example database used by the kNN classifier is initially empty, and the logs are considered individually in random order to simulate a stream. Three models were considered in this study. One model adds all instances from the stream to the example database. One model adds only the instances whose ignorance measure is lower than t_{ign}. The third model selects instances when the ignorance measure is lower than t_{ign} or when the conflict measure is lower than t_{conf}.

A second set of experiments repeated the previous experiment 100 times and considered the influence of the stream order.

The third set of experiments considers the budget with the strategy presented in Section 4. To avoid problems at the beginning of the stream, the example database is initialized with one log at random per category of length. These logs are, therefore, removed from the stream. This experiment was repeated 100 times for different stream orders.

The last set of experiments consider a set up similar to the third set of experiments. In particular, budget is considered and managed with the same strategy, presented in section 4. Contrary to what is done in the third set of experiment, however, concept shift is introduced here. More particularly, long logs are only included in the stream after the first 600 time steps.

6.1 Experiment with no budget

This section focuses on the results of the simulation for one random ordering of the stream. Rather than an independent set of experiments over an implementable solution, this section should be seen as an example of what happen in a particular case to gain insight over the mechanics at play. Budget is not considered yet, so that the base can freely grow.

The conflict threshold is set to $t_{conf} = 0.3$, as previous experiments make it appears an interesting compromise between database growth and prediction score. Further details about these experiments and the roles of the thresholds will be given in the next section. The ignorance threshold is, similarly, set to $t_{ign} = \frac{1}{250}$, that is, $t_{ign} = \frac{1}{t_{nn}}$. In this particular case, the modified kNN rule never has to fall back to the 1NN rule because the Oracle is called whenever no neighbors have a dissimilarity lower than t_{nn}.

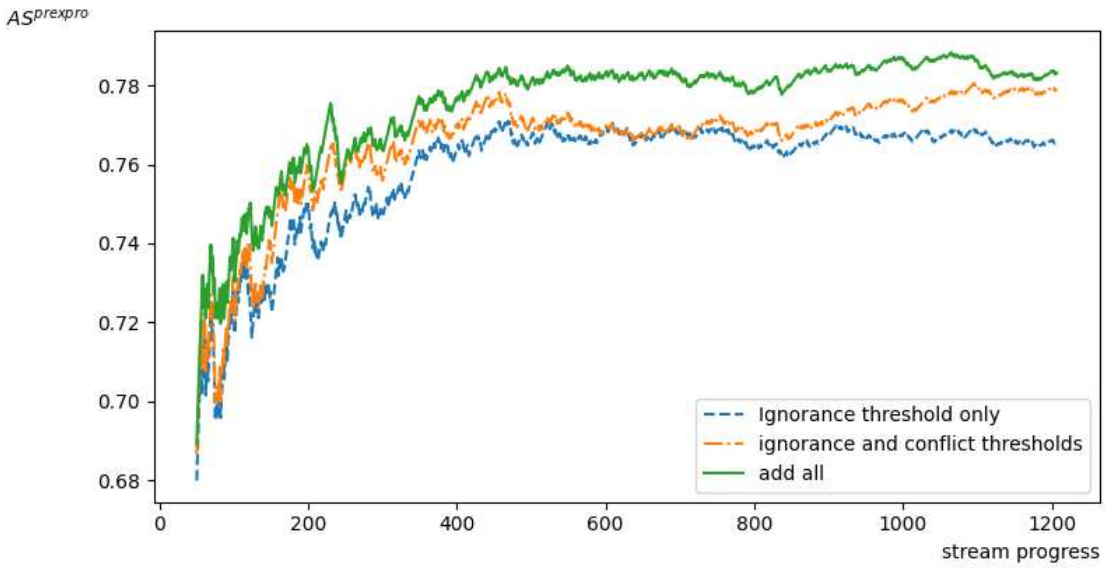

FIGURE 11: Evolution of $AS^{pre \times pro}$ with the progress of the stream for three AL models.

Figure 11 shows the evolution of the accumulated score with the progress of the stream. Here, the next instance is always predicted by the modified kNN rule, whether sent to the Oracle or not. For readability, the plot starts only after the first 50 instances have passed in the stream. The experiment was conducted on the three models. The first considers only the ignorance measure and therefore performs only an *exploration* on the input data space. The second experiment considers both ignorance and conflict thresholds, whereas the third experiment adds all instances from the stream to the example database for future predictions. As can be seen, there is little difference between the three accumulated score curve profiles. They all grow rapidly at the beginning of the stream before stabilizing around similar final values. However, the curves for the models using thresholds are slightly lower than those of the model that adds all examples to the labeled database. A symmetric Wilcoxon signed rank test between the scores of the last 500 predictions of the model when adding all instances of the model using the ignorance threshold shows a p-value of 0.02, which is statistically significant at the 5% confidence level. However, the p-value of the test between the model when adding all instances and the model using both ignorance and the conflict threshold was 0.51.

More importantly, a significant difference can be observed between the model management of the example database in Figure 12. This figure shows the growth of the example

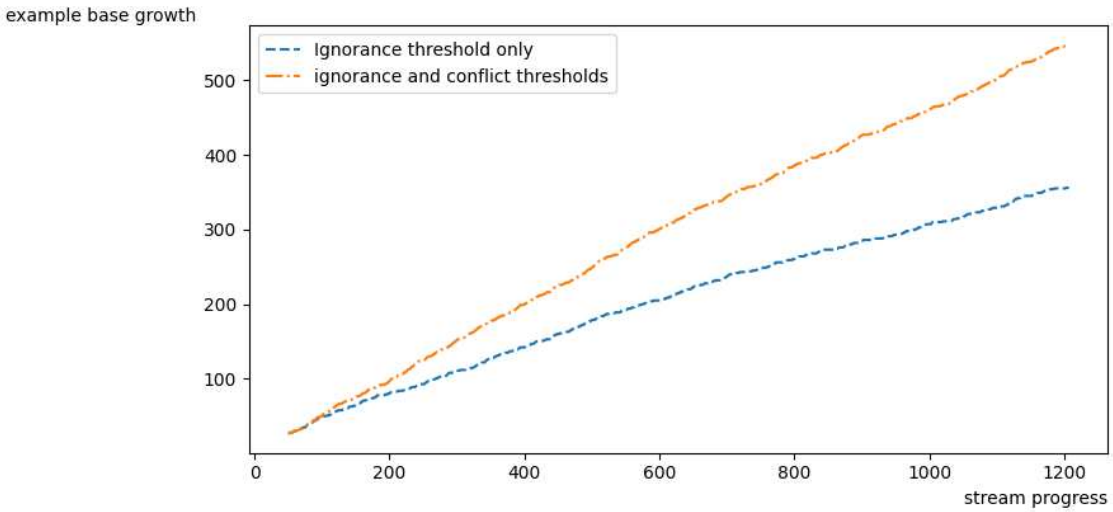

FIGURE 12: Growth of the example database with the progress of the stream for two AL models.

database for the models using either the ignorance rule only, or both the ignorance and conflict rules. Whereas the model adding all examples to the labeled database is not presented for clarity, its curve would be a straight line $y = x$. As can be seen from this figure, however, at the end of the stream, the model using both rules selects only 45% of the instances, whereas the model using the ignorance rule alone selects only 39%. Considering that the nearest neighbor search in this study uses brute force because the data input does not belong to a vector space, and therefore has a linear prediction cost with respect to the size of the example database, reducing this base, as was applied herein, reduces the prediction cost by a similar amount.

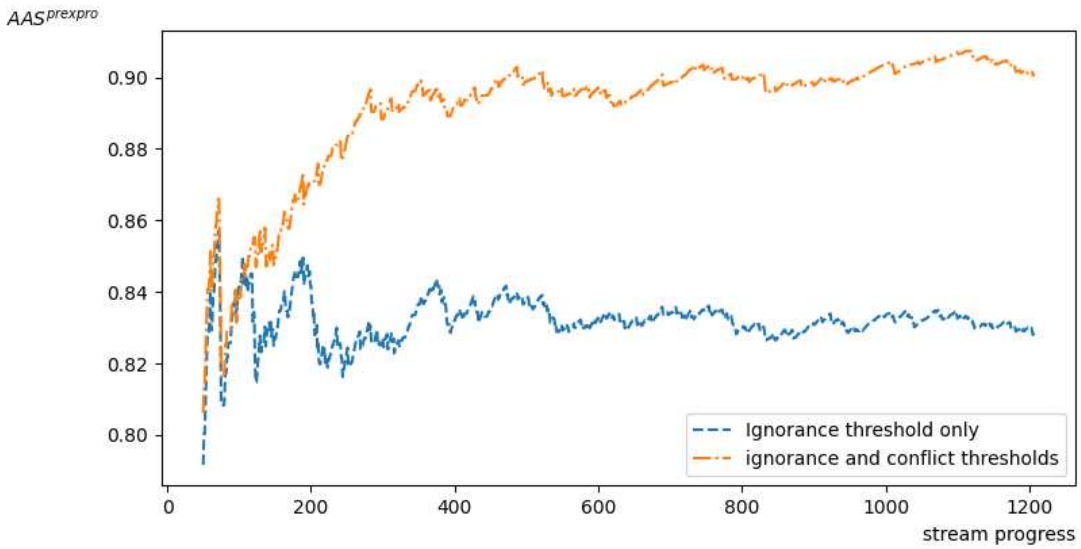

FIGURE 13: Evolution of $AAS^{pre \times pro}$ with the progress of the stream for two AL models.

The benefit of the conflict rule is shown in Figure 13. This figure shows the evolution of $AAS^{pre \times pro}$ as the stream progresses, i.e., the evolution of the accumulated prediction/production score when it is only updated with the prediction/production scores of the predicted class of instances not sent to the Oracle. In this case, the model that adds all aspects to the example database cannot be considered because $AAS^{pre \times pro}$ would never be updated. Here, whereas the evolution of the scores is less stable than in Figure 11, the curves are clearly separated, with the model using the conflict rule having a final score higher than a model that does not. The difference was found to be statistically significant based on a Mann-Whitney U test. Hence, in this case, the conflict rule is effective at detecting instances that would be poorly predicted by the current kNN model.

6.2 Influence of stream order

To study the influence of the stream order, the experiment carried out previously was repeated 100 times for 100 random orders in the stream. The end values of $AS^{pre \times pro}$ of $AAS^{pre \times pro}$ and of the example database size were stored for each repetition. The total number of classes in the example database was also recorded. These values are listed in Table 1.

t_{ign}	t_{conf}	$AS^{pre \times pro}$	$AAS^{pre \times pro}$	Example database size	Number of classes
$\frac{1}{200}$	Not used	77.8 ± 0.1	86.3 ± 0.2	577 ± 1	105.0 ± 0
	0.1	77.9 ± 0.1	88.4 ± 0.2	620 ± 2	105.0 ± 0
	0.3	78.2 ± 0.1	92.4 ± 0.2	707 ± 3	105.0 ± 0
	0.5	78.3 ± 0.1	96.1 ± 0.1	807 ± 2	105.0 ± 0
$\frac{1}{250}$	Not used	76.2 ± 0.2	81.7 ± 0.3	355 ± 1	101.0 ± 0.2
	0.1	77.5 ± 0.2	85.5 ± 0.2	426 ± 3	101.4 ± 0.2
	0.3	77.8 ± 0.1	88.9 ± 0.2	534 ± 4	101.5 ± 0.2
	0.5	78.2 ± 0.1	91.8 ± 0.2	655 ± 4	101.7 ± 0.2
$\frac{1}{300}$	Not used	74.8 ± 0.4	78.7 ± 0.5	234 ± 1	88.4 ± 0.4
	0.1	76.8 ± 0.2	82.5 ± 0.3	300 ± 3	88.5 ± 0.4
	0.3	77.1 ± 0.2	84.7 ± 0.3	392 ± 5	88.6 ± 0.4
	0.5	77.5 ± 0.2	86.5 ± 0.3	504 ± 6	88.8 ± 0.4
add all		78.5 ± 0.1	-	1207	105

TABLE 1: Accumulated scores, example database size, and number of classes discovered at the end of the stream, averaged over 100 runs of the experiment, with random stream ordering.

The experiments were repeated for three different ignorance thresholds t_{ign} and three conflict thresholds t_{conf}. In addition, the table also contains the results for AL models using only the ignorance rule. They correspond to the models marked "Not used" in the t_{conf} column.

As the first remark, the end of the stream accumulated scores of the AL models are on average nearly as good as the model that adds every instance from the stream to the example database. However, they are not *as good as* this last model, as confirmed by the Wilcoxon signed-rank tests between the end of the stream values, with a confidence level of 5%.

However, the worst accumulated score, 74.8%, also corresponds to the model with the most reduced example database. Indeed, 234 logs were selected, which corresponds to only 29% of the total stream. The AL model with the highest example database is, unsurprisingly, the model with the highest ignorance threshold, $t_{ign} = \frac{1}{200}$, and the highest conflict threshold $t_{conf} = 0.5$, as they increase the sensitivity of the rules. The size of the example database in this case is 807 logs, which corresponds to a reduction of approximately 33% with the initial model. Despite selecting fewer samples, the AL models are efficient at detecting new classes. Indeed, by the end of the stream, at least 95 different classes are present in the example database, out of a total of 105 present in the entire dataset. The models with the lowest ignorance threshold systematically selected at least one instance of each class. For comparison, a set of 807 logs sampled randomly from the entire database contains on average only 80 different classes. Similarly, a random set of size 234 contains only 34 classes.

In addition, the average accumulated scores updated only for instances not added to the example database, $AAS^{pre \times pro}$, is always higher than the average accumulated score for all instances from the stream, $AS^{pre \times pro}$. In fact, they are always higher than the maximum value of $AS^{pre \times pro}$ obtained for the model that adds all components to the example database. The only model for which this difference is not statistically significant is the model using $t_{ign} = \frac{1}{300}$ and no t_{conf}, with a p-value of the Wilcoxon test at 0.5. Therefore, the proposed set of rules is efficient at detecting instances from the stream that would be poorly predicted by the current model, sending them to the Oracle.

Concerning the threshold role, a higher conflict threshold for a fixed ignorance threshold coincides on average with a higher end of stream $AAS^{pre \times pro}$, as confirmed using Wilcoxon signed rank tests. Similarly, this results in a higher $AS^{pre \times pro}$. A higher ignorance threshold for a fixed conflict threshold also results in a higher $AAS^{pre \times pro}$ and $AS^{pre \times pro}$.

However, it can be observed that when t_{ign} or t_{conf} increases, the size of the example database also increases because higher thresholds mean that the Oracle is called more often.

6.3 Budget experiment

In this section, a budget limitation is added to the simulation using the strategy defined in Section 4. The number of steps needed to obtain the result of the simulation was fixed to 20, and the number of slots in the simulator was fixed to 10, and thus no more than half of

the instances from the stream could possibly be sent to the simulator. The experiments were run for five values of parameter α. For $\alpha = 0$, only the conflict measure is used. For $\alpha = 1$, only the ignorance measure is used. For $\alpha = 0.3$, 0.5, and 0.7, both measures are used but are more or less important. For comparison, a greedy strategy to dispatch logs between the simulator and classifier is also considered, i.e., whenever a slot is free in the simulator, the next instance is sent to fill it.

α	**AAS$^{\text{pre}\times\text{pro}}$**	**BAS$^{\text{pre}\times\text{pro}}$**	**Example database size**	**Number of classes**	**Average simulator use**
0	79.5 ± 0.2	86.6 ± 0.1	412.2 ± 0.7	20.2 ± 0.7	6.5 ± 0.1
0.3	83.4 ± 0.2	89.1 ± 0.1	412.2 ± 0.7	69.9 ± 0.7	6.5 ± 0.1
0.5	84.0 ± 0.2	89.5 ± 0.1	409.2 ± 0.78	84.6 ± 0.7	6.5 ± 0.1
0.7	83.8 ± 0.2	89.3 ± 0.2	403.4 ± 0.9	93.4 ± 0.6	6.3 ± 0.1
1	81.6 ± 0.3	87.8 ± 0.2	400.0 ± 0.7	98.7 ± 0.4	6.3 ± 0.1
Greedy strategy	77.8 ± 0.2	88.9 ± 0.2	597.0 ± 0	65.1 ± 1	9.5 ± 0.1

TABLE 2: Accumulated scores, example database size, number of classes discovered at the end of the stream, and average number of occupied slots in the simulator, averaged over 100 runs of the experiment, with random stream ordering.

Table 2 presents the results of the simulations: The average end of stream $AAS^{pre\times pro}$, which measures the performance of the kNN model alone, the average end of stream $BAS^{pre\times pro}$, which measures the global performance of the couple simulator-ML classifier, the average end size of the example database, the average number of classes discovered, and the average number of occupied slots in the simulator along the whole stream.

As can be seen, the strategy using only one of the conflict and ignorance measures, that is, for $\alpha = 0$ or $\alpha = 1$, underperforms in terms of both $AAS^{pre\times pro}$ and $BAS^{pre\times pro}$ when compared with models using both measures. In particular, the model using only the conflict measure has the worst accumulated scores among all non-greedy models. It also discovers far fewer classes than the other, i.e., 20 classes on average against more than 65 for all the others, despite sending in slightly more instances to the simulator and example database on average.

Among the models using both measures, $\alpha = 0.5$ yields slightly better results than the others. This difference was rarely found to be statistically significant, however, because when repeating the Wilcoxon test under the hypothesis in which the "prediction production score for $\alpha = 0.3$ is less than the prediction/production score on the same instance for $\alpha = 0.5$", with simulation prediction scores equal to 1, the p-value was never lower than the (Bonferroni corrected) level of 0.0005; in addition, it was under the non-corrected threshold for only four of the experiments.

All models have a similar management of the number of occupied slots and database growth. Indeed, they all use an average of 6 out of 10 slots of the simulators, whereas the size of the example database at the end of the stream is approximately 400.

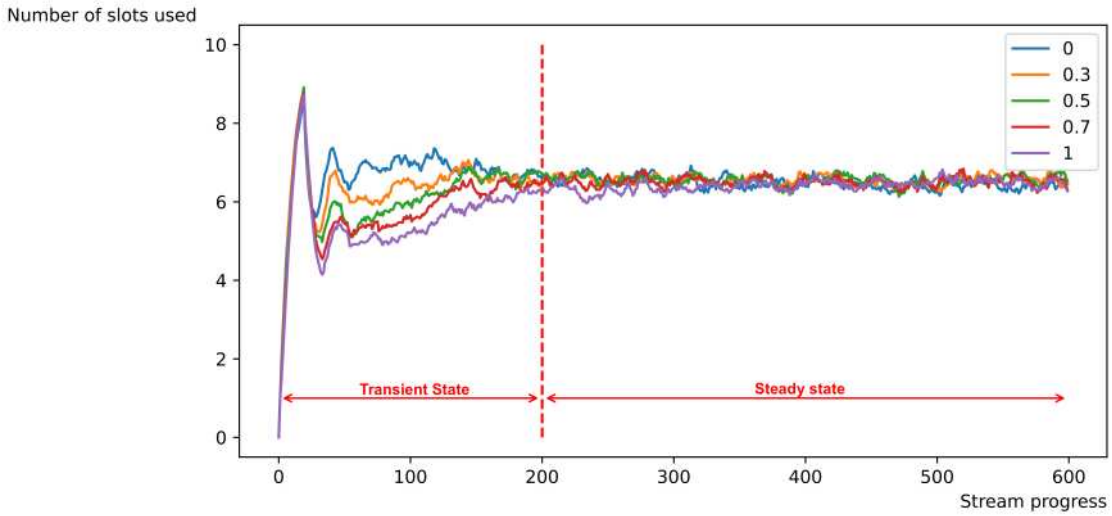

FIGURE 14: Average number of used slots in the simulator as a function of time, for each value of the parameter α

The greedy strategy, despite having a decent average end of stream $BAS^{pre \times pro}$ of higher than $\alpha = 1$ but lower than $\alpha = 0.3$, has the worst $AAS^{pre \times pro}$ among all models considered. Indeed, the example base used by the kNN classifier is built this time at random and therefore does not perform as well as the kNN using the selected example base.

Figure 14 exposes the averaged number of occupied simulator slots for each step of the stream and value of α. For clarity, the plot was stopped after 600 time steps, but the curves keep going as expected. As can be seen, this figure can be separated into a transient state, where the kNN and distributions used for the computation of the quantile threshold are set up, and a steady state. The differences observed between different values of α can be, at least in part, explained by the correction dp. This correction, indeed, push to select instances with a probability closer from b_t during the set up time where the quantiles are ill-defined. However, in the steady state, the average simulator usage is stable and appears independent of the value of α. This might be different, however, in the presence of concept drift, which might be expected to trigger a higher use of the simulator.

6.4 Impact of concept shift

This section presents the results of experiments with concept shift. The notion of budget is considered and managed as in the previous section. Concept shift is similarly introduced, and simulated as follow: As explained in section 5.1, the logs present in the studied dataset belong to three length classes. For the 600 first iterations of the stream, the logs are selected

randomly among the *very shorts* and *shorts* logs only. After the 600^{th} time step, logs are chosen among the all the ones not already passed in the stream, including the *long* logs.

α	**AAS$^{\text{pre}\times\text{pro}}$**	**BAS$^{\text{pre}\times\text{pro}}$**	**Example database size**	**Number of classes**	**Average simulator use**
0	78.7 ± 0.2	86.0 ± 0.1	409.9 ± 0.7	17.0 ± 0.5	6.5 ± 0.1
0.3	82.4 ± 0.2	88.5 ± 0.1	411.9 ± 0.8	60.5 ± 0.7	6.5 ± 0.1
0.5	83.5 ± 0.2	89.2 ± 0.1	402.7 ± 0.8	79.1 ± 0.6	6.5 ± 0.1
0.7	83.5 ± 0.2	89.0 ± 0.2	402.7 ± 0.9	89.5 ± 0.6	6.4 ± 0.1
1	81.9 ± 0.3	88.0 ± 0.2	401.2 ± 0.8	95.8 ± 0.4	6.3 ± 0.1
Greedy strategy	77.6 ± 0.2	88.8 ± 0.1	597.0 ± 0	64.7 ± 0.8	9.5 ± 0

TABLE 3: Accumulated scores, example database size, number of classes discovered at the end of the stream, and average number of occupied slots in the simulator, averaged over 100 runs of the experiment, with random stream ordering. Concept shift is included in these tests.

Table 3 presents the same scores used in the previous section. The difference lays in the simulation of data drift in this set of experiment. These scores are, unsurprisingly, similar to what was obtained in the previous set of experiments, albeit slightly lower in general.

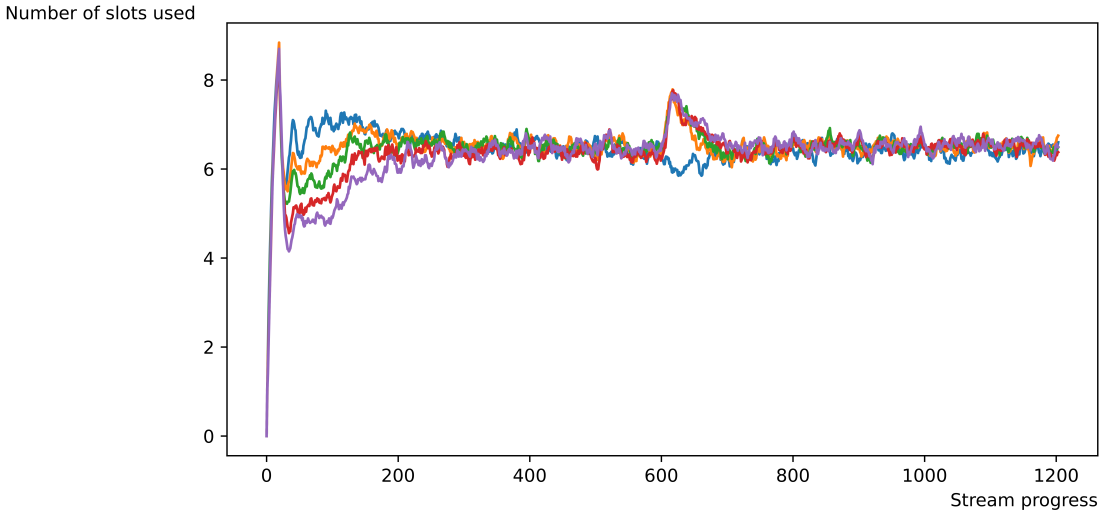

FIGURE 15: Average number of used slots in the simulator as a function of time, for each value of the parameter α. Long logs are introduced from the step 600 only.

Figure 15 appears, in that aspect, of higher interest. This figure is similar to figure 14 and shows the average number of occupied slots in the simulator for each time step. The

influence of the concept shift is visible from the simulator usage increase, for a short period following the appearance of long logs in the stream. This effect is similar for all values of the parameters α, except for $\alpha = 0$, for which no such increase in the number of occupied slots is observed. This is not surprising considering that for $\alpha = 0$, only the conflict measure is used, which will not be able to immediately detect the apparition of logs very dissimilar with the ones previously seen in the stream.

7 Conclusion and limitations

In this paper, an application of an Active Learning concept was proposed to couple two technologies pushed by the transition toward Industry 4.0, by adding one of the mains industry of the future capabilities: predictive capacity to anticipate and react to changes and avoid potential issues. More precisely, a simulator and an ML classifier were coupled. Log scans from a stream were dispatched to one or the other to ensure a high prediction quality of the ML classifier and reduce the number of calls to the costly simulator. Furthermore, the logs sent to the simulator were used to improve the classifier.

It was shown that, when budget limitations are not considered, the method presented in this paper allows the training of a kNN on a fraction of the log scans with nearly as high of an accumulated score than the kNN which adds all instances from the stream to its example database. In addition, the accumulated score computed only over instances that were not added to the database was significantly higher than the accumulated score over the predictions of all instances from the stream. Therefore, the set of measures used in this study can be considered efficient at detecting instances that would be poorly predicted by the kNN, sending them to the simulator instead.

Similarly, when working under a budget, the method presented in this paper can lead to global accumulated scores of over 89% and kNN scores of over 83% on the effectively predicted examples while reducing the number of logs sent to the simulator by a controlled amount.

However, there are some limitations to this study. In particular, the choice of the hyperparameters α for the AL strategy and t_{nn} for the kNN rule are far from easy to achieve, particularly at the beginning of the stream when no information is known on the log scan properties. Whereas it appears from these experiments that the choice of α has little influence on the results as long as both the ignorance and conflict measures are used, this is not a given and might be different for another application. These parameters could, however, be initialized to reasonable values, for example $\alpha = 0.5$, to balance the exploitation and exploration, and $t_{nn} = +\infty$ to start with a standard kNN rule, and evolve over time according to the system results.

The cost of making a prediction using the kNN model considered in this study is bound to increase as the example database is continuously gathered. Further studies will therefore focus on either limiting the maximum size of the database, using an approximate version of

the classifier, for example with a similarity graph, or by using other ML classifiers able to apply dissimilarity inputs.

Acknowledgement

The authors gratefully acknowledge the financial support of the ANR-20-THIA-0010-01 Projet LOR-AI (lorraine intellgence artificielle) and région Grand EST. We are also extremely grateful to FPInnovation who gathered and processed the dataset we are working with.

References

Abraham, A., Dreyfus-Schmidt, L., 2020. Rebuilding trust in active learning with actionable metrics. arXiv preprint arXiv:2012.11365 .

Angluin, D., 1988. Queries and concept learning. Machine learning 2, 319–342.

Bondu, A., Lemaire, V., Boullé, M., 2010. Exploration vs. exploitation in active learning: A bayesian approach, in: The 2010 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 1–7.

Bril El Haouzi, H., 2017. Contribution à la conception et à l'évaluation des architectures de pilotage des systèmes de production adaptables : vers une approche anthropocentrée pour la simulation et le pilotage .

Chabanet, S., Thomas, P., Bril El-Haouzi, H., Morin, M., Gaudreault, J., 2021. A knn approach based on icp metrics for3d scans matching: an application to thesawing process., in: 17th IFAC Symposium on Information Control Problems in Manufacturing.

Chapelle, O., 2005. Active learning for parzen window classifier., in: AISTATS, Citeseer. pp. 49–56.

Cheetham, W., Price, J., 2004. Measures of solution accuracy in case-based reasoning systems, in: European Conference on Case-Based Reasoning, Springer. pp. 106–118.

Dasgupta, S., 2011. Two faces of active learning. Theoretical computer science 412, 1767–1781.

Delany, S.J., Cunningham, P., Doyle, D., Zamolotskikh, A., 2005. Generating estimates of classification confidence for a case-based spam filter, in: International conference on case-based reasoning, Springer. pp. 177–190.

Dudani, S.A., 1976. The distance-weighted k-nearest-neighbor rule. IEEE Transactions on Systems, Man, and Cybernetics , 325–327.

European Factories of the Future Research Association and others, 2012. Factories of the future ppp fof 2020 roadmap: Consultation document. European Commission, Brussels .

Fix, E., 1951. Discriminatory analysis: nonparametric discrimination, consistency properties. USAF school of Aviation Medicine.

FPInnovation, . Optitek 10. User manual.

Gikunda, P.K., Jouandeau, N., 2020. Cost-based budget active learning for deep learning. arXiv preprint arXiv:2012.05196 .

HALCO software systems ltd. , . The sawsim sawmill simulation tool. URL: https://www.halcosoftware.com/software-1-sawsim.

Htike, K.K., Hogg, D., 2016. Adapting pedestrian detectors to new domains: a comprehensive review. Engineering Applications of Artificial Intelligence 50, 142–158.

Iafrate, F., 2014. A journey from big data to smart data, in: Digital Enterprise Design & Management. Springer, pp. 25–33.

Kottke, D., Herde, M., Sandrock, C., Huseljic, D., Krempl, G., Sick, B., 2020. Toward optimal probabilistic active learning using a bayesian approach. arXiv preprint arXiv:2006.01732 .

Kottke, D., Krempl, G., Spiliopoulou, M., 2015. Probabilistic active learning in datastreams, in: International Symposium on Intelligent Data Analysis, Springer. pp. 145–157.

Kumar, A., Halder, A., 2020. Ensemble-based active learning using fuzzy-rough approach for cancer sample classification. Engineering Applications of Artificial Intelligence 91, 103591.

Kumar, P., Gupta, A., 2020. Active learning query strategies for classification, regression, and clustering: A survey. Journal of Computer Science and Technology 35, 913–945.

Kurlej, B., Woźniak, M., 2011. Learning curve in concept drift while using active learning paradigm, in: International Conference on Adaptive and Intelligent Systems, Springer. pp. 98–106.

Lewis, D.D., Gale, W.A., 1994. A sequential algorithm for training text classifiers, in: SIGIR'94, Springer. pp. 3–12.

Lowrance, C.J., Lauf, A.P., 2019. An active and incremental learning framework for the online prediction of link quality in robot networks. Engineering Applications of Artificial Intelligence 77, 197–211.

Lughofer, E., 2012. Single-pass active learning with conflict and ignorance. Evolving Systems 3, 251–271.

Lughofer, E., 2017. On-line active learning: A new paradigm to improve practical useability of data stream modeling methods. Information Sciences 415, 356–376.

Lughofer, E., Weigl, E., Heidl, W., Eitzinger, C., Radauer, T., 2015. Integrating new classes on the fly in evolving fuzzy classifier designs and their application in visual inspection. Applied Soft Computing 35, 558–582.

Martinez Arellano, G., Ratchev, S., 2019. Towards an active learning approach to tool condition monitoring with bayesian deep learning .

Masood, T., Sonntag, P., 2020. Industry 4.0: Adoption challenges and benefits for smes. Computers in Industry 121, 103261.

Mohamad, S., Sayed-Mouchaweh, M., Bouchachia, A., 2018. Active learning for classifying data streams with unknown number of classes. Neural Networks 98, 1–15.

Monostori, L., 2014. Cyber-physical production systems: Roots, expectations and r&d challenges. Procedia CIRP 17, 9–13.

Morin, M., Gaudreault, J., Brotherton, E., Paradis, F., Rolland, A., Wery, J., Laviolette, F., 2020. Machine learning-based models of sawmills for better wood allocation planning. International Journal of Production Economics 222, 107508.

Morin, M., Paradis, F., Rolland, A., Wery, J., Laviolette, F., Laviolette, F., 2015. Machine learning-based metamodels for sawing simulation, in: 2015 Winter Simulation Conference (WSC), IEEE. pp. 2160–2171.

Morneau-Pereira, M., Arabi, M., Gaudreault, J., Nourelfath, M., Ouhimmou, M., 2014. An optimization and simulation framework for integrated tactical planning of wood harvesting operations, wood allocation and lumber production, in: MOSIM 2014, 10eme Conférence Francophone de Modélisation, Optimisation et Simulation.

Nesrine, A., Henri, P., 2019. Adaptive smart card-based pull control systems in context-aware manufacturing systems: Training a neural network through multi-objective simulation optimization. Applied Soft Computing 75, 46–57.

Rajpathak, D., Xu, Y., Gibbs, I., 2020. An integrated framework for automatic ontology learning from unstructured repair text data for effective fault detection and isolation in automotive domain. Computers in Industry 123, 103338.

Rastogi, R., Sharma, S., 2019. Fast laplacian twin support vector machine with active learning for pattern classification. Applied Soft Computing 74, 424–439.

Rossgoderer, U., 2015. Industrie 4.0-digitalization strategy, in: Plant Simulation Worldwide Use Conference. Siemens AG, Stuttgart.

Salama, S., Eltawil, A.B., 2018. A decision support system architecture based on simulation optimization for cyber-physical systems. Procedia Manufacturing 26, 1147–1158.

Schein, A.I., Ungar, L.H., 2007. Active learning for logistic regression: an evaluation. Machine Learning 68, 235–265.

Schmidt, S., Schnitzer, S., Rensing, C., 2016. Text classification based filters for a domain-specific search engine. Computers in Industry 78, 70–79.

Selma, C., El Haouzi, H.B., Thomas, P., Gaudreault, J., Morin, M., 2018. An iterative closest point method for measuring the level of similarity of 3d log scans in wood industry, in: Service Orientation in Holonic and Multi-Agent Manufacturing. Springer, pp. 433–444.

Settles, B., 2009. Active learning literature survey, in: Computer Sciences Technical Report 1648, University of Wisconsin-Madison Department of Computer Sciences.

Settles, B., 2011. From theories to queries: Active learning in practice, in: Active Learning and Experimental Design workshop In conjunction with AISTATS 2010, JMLR Workshop and Conference Proceedings. pp. 1–18.

Seung, H.S., Opper, M., Sompolinsky, H., 1992. Query by committee, in: Proceedings of the fifth annual workshop on Computational learning theory, pp. 287–294.

Thomas, P., Thomas, A., 2011. Multilayer perceptron for simulation models reduction: Application to a sawmill workshop. Engineering Applications of Artificial Intelligence 24, 646–657.

Todoroki, C., et al., 1990. Autosaw system for sawing simulation. New Zealand Journal of Forestry Science 20, 332–348.

Wang, L., Hu, X., Yuan, B., Lu, J., 2015. Active learning via query synthesis and nearest neighbour search. Neurocomputing 147, 426–434.

Wang, M., Min, F., Zhang, Z.H., Wu, Y.X., 2017. Active learning through density clustering. Expert systems with applications 85, 305–317.

Wery, J., Gaudreault, J., Thomas, A., Marier, P., 2018. Simulation-optimisation based framework for sales and operations planning taking into account new products opportunities in a co-production context. Computers in industry 94, 41–51.

Wu, D., Rosen, D.W., Wang, L., Schaefer, D., 2015. Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation. Computer-Aided Design 59, 1–14.

Wu, Y., Kozintsev, I., Bouguet, J.Y., Dulong, C., 2006. Sampling strategies for active learning in personal photo retrieval, in: 2006 IEEE International Conference on Multimedia and Expo, IEEE. pp. 529–532.