Chapter 8
A Decision-Support System for the Digitization of Circular Supply Chains

Dimitris Ntalaperas, Iosif Angelidis, Giorgos Vafeiadis, and Danai Vergeti

Abstract As it has been already explained, it is very important for circular economies to minimize the wasted resources, as well as maximize the utilization value of the existing ones. To that end, experts can evaluate the materials and give an accurate estimation for both aspects. In that case, one might wonder, why is a decision support system employing machine learning necessary? While a fully automated machine learning model rarely surpasses a human's ability in such tasks, there are several advantages in employing one. For starters, human experts will be more expensive to employ, rather than use an algorithm. One could claim that research towards developing an efficient and fully automated decision support system would end up costing more than employing actual human experts. In this instance, it is paramount to think long-term. Investing in this kind of research will create systems which are reusable, extensible, and scalable. This aspect alone more than remedies the initial costs. It is also important to observe that, if the number of wastes to be processed is more than the human experts can process in a timely fashion, they will not be able to provide their services, even if employment costs were not a concern. On the contrary, a machine learning model is perfectly capable of scaling to humongous amounts of data, conducting fast data processing and decision making. For power plants with particularly fast processing needs, an automated decision support system is an important asset. Moreover, a decision support system can predict the future based on past observations. While not always entirely spot on, it can give a future estimation about aspects such as energy required, amounts of wastes produced etc. in the future. Therefore, processing plants can plan of time and adapt to specific needs. A human expert can provide this as well to some degree, but on a much smaller scale. Especially in time series forecasting, it is interesting to note that, even if a decision support model does not predict exact values, it is highly likely to predict trends of the value increasing or decreasing in certain ranges. In the next sections, we are going to describe the four machine learning models that were developed and which compose the Decision Support System of FENIX. Section 8.1 describes how we predict the quality of the extracted materials based on features such as temperature, extruder speed, etc. Section 8.2 describes the process of extracting heuristic rules based on

D. Ntalaperas (✉) · I. Angelidis · G. Vafeiadis · D. Vergeti
SingularLogic, Achaias 3 & Trizinias st., 14564 Kifissia, Greece

P. Rosa and S. Terzi (eds.), *New Business Models for the Reuse of Secondary Resources from WEEEs*, PoliMI SpringerBriefs,
https://doi.org/10.1007/978-3-030-74886-9_8

existing results. Section 8.3 describes how FENIX provides time-series forecasting to predict the future of a variable based on past observations. Finally, Sect. 8.4 describes the process of classifying materials based on images.

8.1 Extracted Materials Quality Prediction

The first model of the proposed decision support system predicts the quality of the produced material based on input features. More specifically, a Logistic Regression model is used. This model is well-documented [1] and considered a standard in deep learning. Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences, while it also provides invaluable predictions in market applications. Before explaining the finer details of FENIX's model, we will explain the fundamentals of logistic regression in general. The goal of logistic regression is to find the best fitting but biologically reasonable model to describe the relationship between the binary characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients, its standard errors as well as the significance levels of a formula to predict a logit transformation of the probability of presence of the characteristic of interest:

$$logit(p) = b_0 + b_1 X_1 + b_2 X_2 + b_3 X_3 + \cdots + b_k X_k$$

It originates from statistics and, while the most basic type of LR is the binary LR, which classifies inputs into one of two categories as explained above, its generalization classifies inputs into arbitrarily many categories. It is important to note, however, that machine learning does computations in terms of numerical matrices which are composed of features and weights. Since all features must be numbers, any input feature which is not a number must be properly processed to remedy this. A typical approach to this is one-hot categorical encoding. All non-numerical features in FENIX such as the name of the material have a finite set of possible values, enabling the use of one-hot categorical encoding. What this encoding does is it maps each string value into a vector of zeros, as many as the possible values for that feature, while one of them is one for the position that represents the original string. For example, let us assume that feature "f" has possible values "A", "B", "C", then we would map those to [1, 0, 0], [0, 1, 0] and [0, 0, 1], respectively. When encountering an input of "B", we would immediately convert it to [0, 1, 0]. While one-hot encoding solves the issue, it creates another one. Since other features are already numerical, we need to somehow "merge" the dimensions of the features for input X. For example, let's assume that X has the features "mean temperature" (value 34), "process energy" (value 25) and "f" (value "B"). Then, 34, 25 and [0, 1, 0] need to be fed into the model, but their dimensions do not match. This is solved by assuming each feature being a $1 \times N$ vector and then concatenating these vectors to form the final input X. In this instance, X would be [34, 25, 0, 1, 0]. Another important aspect of the model

to discuss involves its activation function. The activation function defines the output of that node given an input or set of inputs. For a binary LR, the activation function would be a sigmoid, because it outputs a value in [0, 1], expressing a probability.

$$\varphi(u_i) = (1 + e^{-u_i})^{-1}$$

While this works well when a problem has only 2 possible classes, we need a different way to achieve the same result for arbitrarily many classes. The solution lies in using the softmax activation function.

$$f_i(x^{\rightarrow}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \quad j = 1, \ldots, J$$

The softmax activation function has a form that forces the resulting numbers to be in [0, 1] as well, but with one additional property: their sum is always 1, so all outputs express a probability distribution for the initial input. For example, if we have three possible classes, C1, C2, C3 and an input X, the output would be something like [0.3, 0.3, 0.4]. All values are in [0, 1] and their sum is 1. For FENIX's purposes, we try to predict the Satisfactory status of a material based on features such as material input, process mean temperature, extruder speed (mm/min). Each material is classified into one of three categories "Yes", "No", "Printable". Features which are not initially numerical are converted into one-hot categorical encodings and then all features for each input are concatenated to form the input vector for the model, following the procedure that was explained above. After the models' computations, we obtain an output of the format [p1, p2, p3], where p1 indicates the estimated probability for "Yes", p2 for "No" and p3 for "Printable". Obviously, the highest among the values is declared the model's prediction of class for the specified input. Before the model can be used, however, it needs to be trained. To that end, we conducted supervised learning. In general, machine learning uses two general learning strategies during training, depending on the task at hand: supervised or unsupervised learning. Unsupervised learning is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision. In contrast to supervised learning that usually makes use of human-labelled data, unsupervised learning allows for modelling of probability densities over inputs. Supervised learning is the task of learning a function that maps an input to an output based on example input–output pairs. For FENIX, we already know that each material will have one of three "Satisfactory" status, we only need to learn to predict it. To that end, we labelled a generous amount of input with the correct satisfactory status. These samples were further split into test, training, and validation datasets. The split is necessary, because we want the model to train under part of the data and not overfit, which is why we need the test-train split. However, we also want to make sure its performance is sufficient even on samples it sees for the first time (which will be the case in deployment as well), as well as ensure the model does not create correlations of any kind between test and training data. This

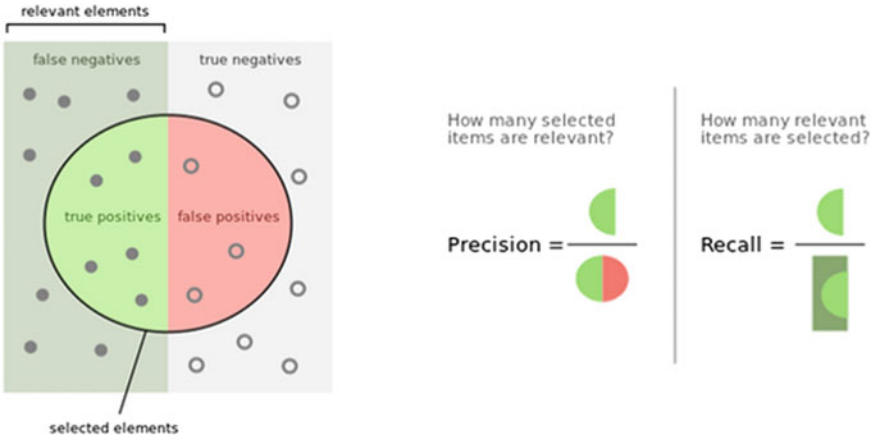

Fig. 8.1 Precision and recall

justifies the further split of the test data into test and validation. In order to increase the credibility of the results, we used state-of-the-art metrics to evaluate our model, namely precision, recall and F1-score. In pattern recognition, information retrieval and classification (machine learning), precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of the total amount of relevant instances that were retrieved. Both precision and recall are therefore based on an understanding and measure of relevance (Fig. 8.1).

Intuitively, precision shows the ratio of the outputs we predicted correctly compared to the sum of correct predictions and false positives (values that were classified as being correct by mistake). Recall shows the ratio of the outputs we predicted correctly compared to the total amount of correct predictions and false negatives (values that should be classified as the specific class but did not by mistake).

Precision formula

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{8.1.1}$$

Recall formula

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{8.1.2}$$

Finally, F1-score belongs to the family of F metrics, which indicate a weighted mean calculation of precision and recall, with the goal of providing a final "overall score" for a model.

$$F_\beta = (1+\beta) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} (in\ precision - recall\ terms)$$

$$F_\beta = (1+\beta) \cdot \frac{(1+\beta)^2 \cdot TP}{(1+\beta)^2 \cdot TP + \beta^2 \cdot FN + FP} (in\ TP - FP - FN\ terms)$$

When β = 1 we get the F1-score, which is the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

Finally, to make the evaluation even more thorough, we conducted k-fold cross validation. This is a powerful technique when training a model under a dataset because it tries to reduce overfitting as much as possible. Simply put, let us assume that all the samples for training (test, train, validation) are split into k sets (where k > 3). Then, if we shuffle the k sets and assign them to test, train and validation (with correct proportions, train should be about 60% of the samples and test and validation 20% each), we force the model to better generalize its learning capabilities. To make the model more useful, we provide it as a service to the FENIX platform via a REST API. As a matter of fact, all four models documented in this chapter are provided in that way. This isolates each model for potential future extensions, addition of features etc., while they seamlessly work together with the platform. While it is important to document the model's pipeline from training to deploying, it is also necessary to justify its usefulness for a circular economy of reusing materials, especially after going through all this trouble. As already explained, a human expert can do the same as LR does, probably with more precise results. However, there are a few things to consider. For starters, if more variables need to be taken into consideration for decision-making, the model can be very easily adapted (to the point of barely changing its code even). In contrast, a human expert will need to adapt his strategy and heuristics, perhaps even do research on new variables, and learn their role and how they affect the result, to achieve similar results. Considering many variables can also lead to mistakes as a single wrong calculation would lead an expert to make the wrong call. The model will never make such mistakes. Furthermore, when facing an industry with ever-increasing needs for fast and efficient processing, it's hard to argue against a fully automated decision support system that can almost instantaneously notify about the quality of the produced material just by taking into consideration the initial parameters it is going to be processed. This knowledge can even be used in artificial experiments to save thousands or perhaps millions of dollars by experimenting with optimal values, instead of trying them and potentially failing.

8.2 Rules Extraction

The next model we are going to present is responsible for extracting rules based on input parameters and pre-classified results. Its function not only complements the LR classifier, it also provides powerful insight on greatly reducing the cost of resources

when recycling materials. This will be further expanded upon near the end of the section. The goal of this model is simple: given a set of sample inputs with specified variables and the actual result that is a direct result of these variables, it creates a hierarchy of rules, starting from the most prevailing one and/or selecting the top N rules which guarantee with maximal accuracy a desired result. A careful reader might ask, why maximal and not maximum? Remember that every machine learning model makes mistakes (no matter how small), they merely try to maximize the accuracy they have. This means that the extracted rules will ensure that the model is correct for as many times as it is possible given the data it was trained with. In general, rules extraction consists of a family of powerful inductive algorithms which are based on the principle of separate and conquer. There is a wide area of applications these kinds of models can be applied on: stock investment, finances, text processing and so on. An investor can analyse the global market and use extracted rules to decide when it is wise to invest on a specific stock based on past economic trends. Text processing for a specific purpose can utilize specific rules to decide on thresholds certain heuristics work. Such models are based on decision trees [2, 3] and random forests [4], which generate a hierarchical structure of rules based on their maximal accuracy, then combine them to maximize the overall accuracy. It is important to note that, while other combinations of rules of similar total accuracy may exist, the ones extracted are also the simplest. This is very interesting, because the heuristic criteria needed to fulfil a specific outcome are as few and simple as possible. The model used for the purposes of the project offers a trade-off between the interpretability of a Decision Tree and the modelling power of a Random Forest. Its hybrid architecture consists of both decision and regression trees. After filtering a set of logical rules according to precision and recall thresholds, the higher performance rules are extracted and, after deduplication (it is possible to reach the same rules from different routes), the final set of the best heterogenous rules is generated. The implementation is based on SkopeRules, which in turn is based on the works of RuleFit [5], Slipper [6], LRI [7], MLRules [8]. In our case, the model utilizes the same training samples as the LR model. The difference is that both the input features and the labels are now given; we now want to extract the rules which maximize the accuracy for each label type. So, since "Satisfactory" status can be "Yes", "No" or "Printable", we need to extract 3 sets of rules, one for each label. This time, we split the dataset a bit differently since rule extraction only works for a single label. 3 separate model trainings take place, each model training under sample data for its respective label. A major advantage of this approach is that, once the set of rules for each label is generated, it does not have to be computed again. It can be stored and just returned on demand. As with all models, this is also offered as a separate service via a REST API, allowing future updates or retraining of the model while the FENIX platform is live. As before, we need to discuss the value of the model compared to preferring a more traditional approach, such as employing a human expert. While a human is capable of providing a set of heuristic rules for maximizing the accuracy for a specific label, it is unlikely they will be the simplest possible and it is difficult to provide weights on the importance of the rules' aspects they picked, they can only weigh their importance based on their own experience in the field. On the contrary,

automated rule extraction can provide the simplest rules possible given a training dataset, as well as detailed scoring information. We also need to consider the case where more variables need to be considered for decision making. A human expert will need to conduct research on variable thresholds, study their potential in effecting the result etc., not to mention do all the computations again (and running the risk of making a mistake). In the same situation, the model barely needs any change, and, after training, it is ready once more to immediately provide the extracted rule sets. Finally, automated rule extraction can easily enable artificial simulations where power plants can test their results by tweaking the variables around their thresholds. This can lead to cheap discovery of combinations of variables which provide better productivity and with even less cost in resources.

8.3 Time Series Forecasting

Next, we are going to discuss our proposed time series forecasting model. A time series is a series of data points indexed, listed, or graphed, in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus, it is a sequence of discrete-time data. Time Series analysis can be useful to see how a given asset, security, or economic variable changes over time. Obviously, any kind of variable evolving through time is an immediate subject of application, such as stock market values, financial values of houses in the 1970s etc. The power of time series analysis lies in taking into consideration more than just the individual discrete data points. It takes into consideration the correlation between two data points, but it also takes into consideration the overall behaviour of the series throughout the entire past to generate the next point. For our purposes, we wish to analyse the active, reactive, and apparent power series in order to predict the future. Since we already have existing samples, we can just get rid of a few samples after a certain time t. Then, we can try to predict them by training on the past points of the series. A very important parameter that greatly affects the capabilities of a time series prediction model is the window size. Simply put, it is the number of discrete data points the series uses internally to infer correlations from. Selecting a large window can lead to overfitting or completely missing the important correlations (due to the model focusing on the general structure of the series instead of specific patterns and correlations). On the other hand, a small window provides little to no correlation information, making the model perform badly. Therefore, it is important to find a window size value which serves as a good compromise between the two. The model we created gets as initial data all existing data points for each reactor variable, then generates incrementally and in real-time future data points indefinitely. It is important to observe here that, since the time series contains numerical values already (we are inspecting a numerical variable's evolution through time), no transformations for input data is required for this model, a far cry when compared to the previous models. As time advances after the initial data points, the series graph is split into two parts: the series evolving with real values and the series evolving with predicted

values. This dual graph is provided to assist in decision making, so that any major differences between a predicted and actual value can be immediately spotted. To make the information more accessible, the past of the series is coloured in blue, the predicted series in red and the actual future series in dotted blue. The diagrams also offer dynamic capabilities, such as freezing a part of the series for more thorough inspection, zooming in to take a closer look at a set of data points, etc. A thumbnail of the entire series is also provided to make browsing easier. Time series prediction of this kind is particularly useful for FENIX decision support. Reactors' status does not need to be constantly monitored that way, releasing human resources for other, more immediate tasks. If there is a big difference between a prediction and an actual value, an automatic alert can be immediately forwarded so that necessary actions can be taken on time. Furthermore, estimating the future values of the reactors' variables can make it easier to predict the lifespan of the reactors, their energy needs etc. This can increase savings by a large margin. It also offers the opportunity to maintain the reactors much better by managing their condition and greatly expanding their lifespan. Another strength of this approach lies in its lightweight nature. The initial input sets the limit to how much data will be in memory. After that point, only a window size's worth of data will be in memory since new values are incrementally generated and returned to the FENIX platform via REST. Computations are fast and the memory needs will not increase in the future, making this model even more appealing to utilize. Like all models, this is also provided as an independent service via a REST API, allowing future updates or retraining of the model while the FENIX platform is live.

8.4 Materials Classification

Lastly, we present the materials classifier, which is based on image data. In this case, the input is an image of the material at hand, while the model classifies the image (and, consequently, the material) into one of two categories (good, bad material). Like all previous models, extending the possible output categories to a larger set is trivial. Once more, we first need to delve into the actual model's finer details before showcasing its usage within FENIX. Since the input is images this time around, the first hidden layer of the neural net will be convolutional. This means that, instead of having a different weight for each pixel of the image, only a small set of weights (and therefore a significantly a smaller number of neurons are needed) are being applied to small subsets of the image. The reason this is more promising than a plain neural network is that "local features" found in previous layers rather than pixels are being forwarded and, as a result, the network sees progressively larger and more complete parts of the image. In addition, since neurons focus on learning specific local features instead of learning everything over and over (e.g., degree lines in images, small shapes etc.) for each pixel, they gain a considerable speedup. Also, since the focus now is entire subsets of the image and not pixels, keeping all values can be redundant and therefore it is possible to gain even more in speedup if subsampling takes place.

Layers that do that are called pooling layers and together with the convolutional layers they form the fundamental building blocks of Convolutional Neural Networks (CNNs). Since CNNs have been around for many years and have been used in many applications (e.g., object detection in images, face recognition, OCR etc.), many datasets with images have been readily available by the community to train and evaluate such models. A well-known example is MNIST (Modified National Institute of Standards and Technology database), a large database of handwritten digits that is commonly used for training various image processing systems, with the goal of recognizing handwriting and digits. Another useful source for datasets with tasks like our own is Kaggle. It offers a dataset for an almost identical problem to our own: dog versus cat classification. The dataset contains 25,000 of dogs and cats, offering a generous number of samples to split into test-train-validation. However, for tasks such as material classification and especially when speed is essential in industrial workflows, it might not be possible to wait to obtain a good amount of training samples. While this adds an additional challenge to our approach, there are ways to remedy this. We will discuss two here. The first approach is pre-processing of the initial data and augmenting them. What this means is that we take all available samples and apply random basic yet random image transformations such as zooming, rotations, rescaling etc. If the randomness is uniform enough without any bias, this will generate even more data of a similar nature to the original. This technique can essentially multiply the available training samples. That way, even with very few data, we can produce a full training-ready dataset. The second approach is utilizing a pretrained model on a large dataset and using its bottleneck features for our own model, after fine-tuning the top layers. A pretrained model of this kind has already learned the features which are the most useful. Leveraging these features, we can potentially reach a better accuracy than relying only on available data. A well-known example of such models is VGG16, which is pretrained on the ImageNet dataset.

Before incorporating a model like this into a custom model, fine-tuning its top layers can further improve the results. This is done by instantiating VGG16's convolutional base with weights, adding the custom model on top of it (load its weights as well), then finally freeze the layers of the VGG16 model up to the last convolutional block. The result is a fine-tuned model to the specific task at hand. It is also necessary to illustrate a weakness of utilizing a pretrained model. For a model like this to be useful for general tasks, it must have been trained on very large datasets and, to perform well, it is highly likely to be a model with a complex architecture (this is further evident if you turn your attention back to Fig. 8.2). This means that, to obtain the bottleneck features and fine-tune it, significant computational power is required, while it will not be very fast the first time around. In applications where speed is also an important factor, this might not even be an actual option. While the second approach can potentially produce better results as explained before, we opted to use data augmentation for FENIX's purposes. Since we are addressing a real-life problem involving online decision support and often requiring near-immediate decisions on top of that. Therefore, it is paramount to utilize an approach that can give relatively good results even with few data samples. Data augmentation for our purposes is relatively simple. We split images for materials for which we already

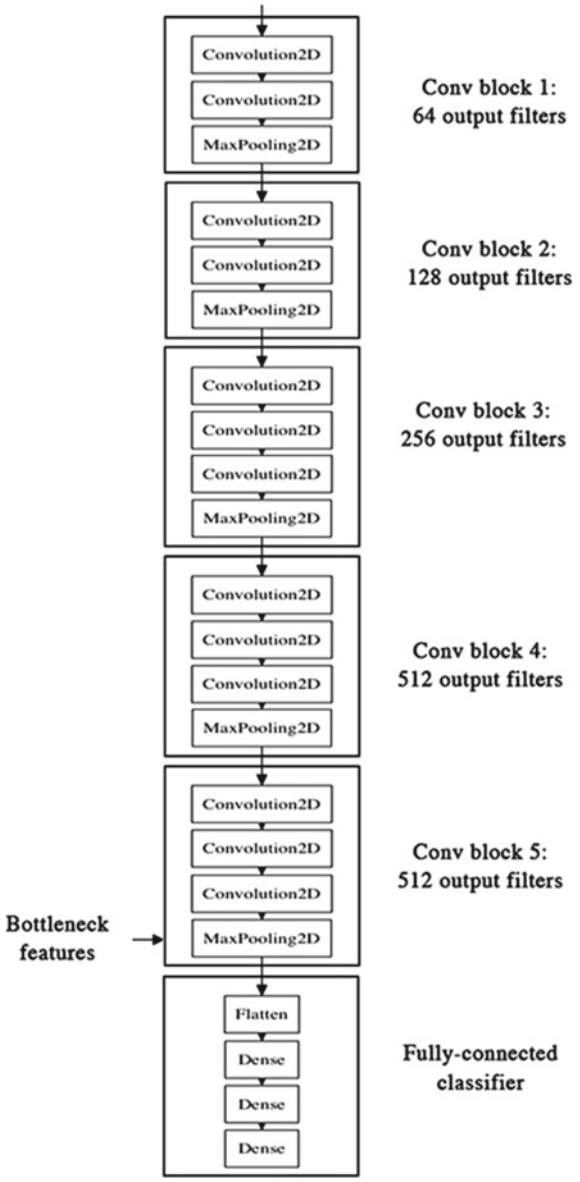

Fig. 8.2 VGG16 architecture

know their class (good, bad) into their respective folders. Then, we augment the data by randomly applying transformations on these images. An image produced by a pre-classified as good or bad will obviously belong to the same class. Taking care to produce an equal number of samples for all classes (two in our case), we further split the data into train, test, validation, splitting each class folder into approximately 60%, 20% and 20%, respectively. Training is relatively fast on a CNN model, especially

when leveraging the power of a strong GPU. The result is a model which we offer as a service via a REST API to the FENIX platform. The platform submits a material image and the model almost instantly returns a response saying if it is a good or a bad material.

8.5 Conclusions

Finally, it is interesting to observe that CNN models extract generic features from an image such as lines, objects etc. which help them conduct classifications. While a human expert can do something equivalent, a model can potentially extract features which may not be intuitively responsible for biasing the classification towards one category or another, yet they do. Even better, with a few adjustments, the model can be tweaked to even show how the features it extracted affected its decision-making.

References

1. McCullagh, P., & Nelder, J. A. (1989). *Generalized linear models* (Vol. 37). CRC press.
2. Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees.*
3. Belson, W. (1959). Matching and prediction on the principle of biological classification. *JRSS, Series C, Applied Statistics, 8*(2), 65–75
4. Ho, T. K. (1995). Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, QC, 14–16 August 1995 (pp. 278–282).
5. Friedman, J. H., & Popescu, B. E. (2005). *Predictive learning via rule ensembles.* Technical Report.
6. Cohen, W. W., & Singer, Y. (1999). A simple, fast, and effective rule learner. In *National Conference on Artificial Intelligence.*
7. Weiss, S. M., & Indurkhya, N. (2000). Lightweight rule induction. In *ICML.*
8. Dembczyński, K., Kotłowski, W., & Słowiński, R. (2008). Maximum likelihood rule ensembles. In *ICML.*