NOVA IMS Information Management School

MGI

Master Degree Program in

Information Management

Radar Emitter Classification based on Deep Ensemble

Tiago Pedro Giesta Martins

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

RADAR EMITTER CLASSIFICATION BASED ON DEEP ENSEMBLE

By

Tiago Pedro Giesta Martins

Master Thesis presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Business Intelligence.

Supervisor: Roberto André Pereira Henriques

July 2023

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisboa, July 05, 2023

DEDICATION

To my family, who has always supported me and without them nothing could be accomplished. Thank You!

ACKNOWLEDGEMENTS

I would like to start by thanking the Portuguese Navy for giving me the chance to complete a master program in Information Management. Focusing the present project, a big thank you to Eng. Michiel Poel (Royal Netherlands Navy), who have provided all the data used, and Eng. Adam Leitch (Thales UK) for giving me all the necessary explanations about Electronic Support Measurement. Finally, I would like to express gratitude to my supervisor.

ABSTRACT

Electronic Support Measures (ESM) systems are designed to classify radar signals, providing information about the presence of threats. This function aids in battlefield situational awareness and the commander's decision on which countermeasures to employ. This dissertation aims to develop a deep ensemble model, recognizing the importance of a fast and precise classification based on a deep forest as an alternative to the parameter matching method. Four deep ensemble models and six of its base learners were built and evaluated to classify 52 emitters, using seven train/test datasets and two test datasets with noise, totalling 420 measurements of accuracy and classification speed. After analyzing these results, two deep ensemble models and their base learners were optimized, each for a different dataset, achieving 100% accuracy in a feature-engineered dataset and up to 98.358% in the original dataset. Regarding classification speed, the fastest models can classify 1000 records in 64ms, which may be acceptable in the real world. The experimental results of this approach reveal several advantages, making it a feasible alternative, including reduced dependency on ESM experts, ease of maintenance, quick to update, and high accuracy.

KEYWORDS

Ensemble methods; Deep architecture; Electronic Support Measures; Classification

Sustainable Development Goals (SDG):

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AMOP	Amplitude MOP
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CPON	Class Probability Output Network
CRISP-DM	Cross Industry Standard Process for Data Mining
CW	Continuous Wave
D-SVM	Deep Support Vector Machine
DOA	Direction of Arrival
DS	Design Science
DSRM	Design Science Research Methodology
EA	Electronic Attack
EP	Electronic Protection
ES	Electronic Support
ESM	Electronic Support Measures
EW	Electronic Warfare
ExTrees	Extremely Randomized Trees
FMOP	Frequency MOP
GBDT	Gradient Boost Decision Tree
GMM	Gaussian Mixture Model
HRRP	High-resolution range profile
IS	Information System
KDD	Knowledge Discovery in Databases
KNN	K-Nearest Neighbor
LASSO	Least Absolute Shrinkage and Selection Operator
LPI	Low Probability of Intercepts
LSTM	Long Short-Term Memory

MARS	Multivariate Adaptive Regression Splines
MLP	Multi-layer Perceptron
MOP	Modulation on Pulse
PCA	Principal Component Analysis
PDW	Pulse Descriptor Word
PMOP	Phase MOP
POP	Pulse On Pulse
POPed	Pulse On Pulse, interrupted pulse
PRF	Pulse Repetition Frequency
PRI	Pulse Repetition Interval
PRT	Pulse Repetition Time
PW	Pulse Width
RADAR	Radio Detecting and Ranging
RESM	Radar ESM
RF	Radiofrequency
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Network
SAR	Synthetic Aperture Radar
SDG	Sustainable Development Goals
SEMMA	Sampling, Exploring, Modifying, Modelling, and Assessing
SGD	Stochastic Gradient Descent
SIGINT	Signal Intelligence
SMOTE	Synthetic Minority Oversampling Technique
SOM	Self Organizing Maps
SVM	Support Vector Machine
TOA	Time of Arrival
XGBoost	Extreme Gradient Boost

1. INTRODUCTION

A radar (radio detecting and ranging) illuminates its surroundings, like a searchlight, and picks up part of the energy scattered by the objects it illuminates, as shown in Figure 1.1 (Meikle, 2008).

Considering a pulse radar, the energy is transmitted by short pulses of electromagnetic energy in the radiofrequency (RF) spectrum, which are reflected when it hits an object. The reflected pulses are used to determine the direction and distance of that object (Sebastian, 2017).

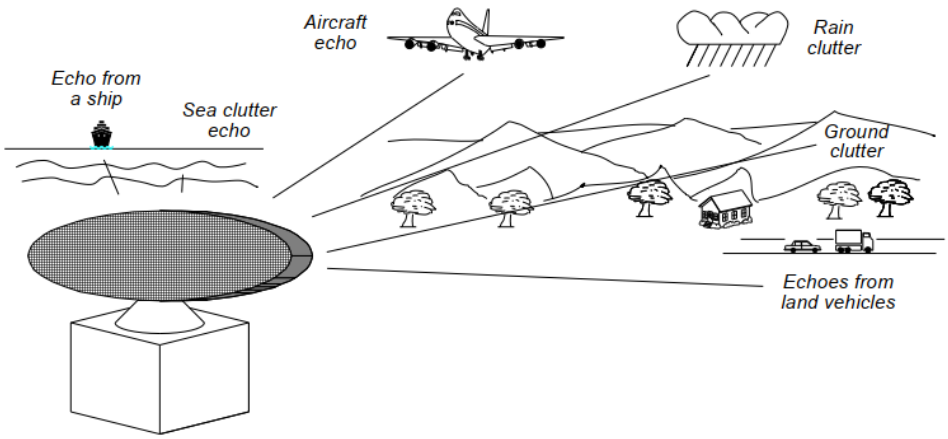


Figure 1.1 - Ground radar and its environment (Meikle, 2008, p. 1)

The time between two consecutive transmitted pulses is named Pulse Repetition Time (PRT) or Pulse Repetition Interval (PRI), and the number of pulses transmitted per second is called Pulse Repetition Frequency (PRF). Another essential characteristic of radar is each transmission's Pulse Width (PW), as represented in Figure 1.2 (Sebastian, 2017). Furthermore, the energy delivered to a target is determined by the product of PW with the transmitter output power (height of a pulse) (Revillon, 2019).

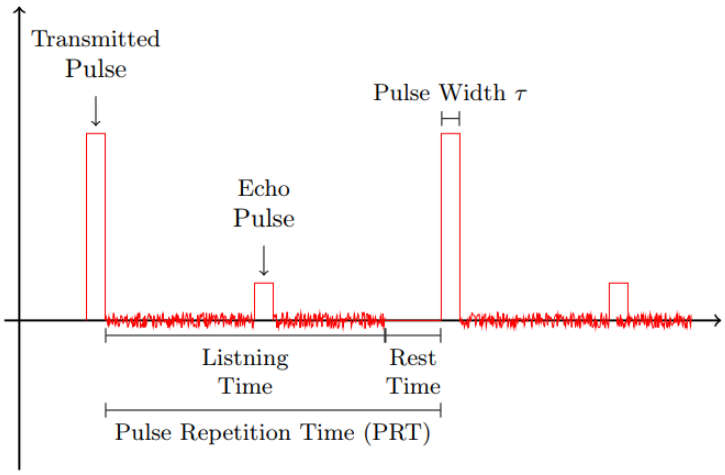


Figure 1.2 - Illustration of transmission of a pulse (Sebastian, 2017, p. 9)

These characteristics, mainly RF and PRI, can be fixed or modulated pulse-to-pulse; that is, they can have the same parameter overtime or change from one pulse to another. When a radar can change RF, it is said to have frequency agility or hopping. A radar is frequency agile if its modulation varies within fixed bounds around a central frequency; in the case of frequency hopping, the frequency has a systematic variation on a set of different frequencies. Regarding PRI, it can have the following modulations, as shown in Figure 1.3: Constant, when it has a fixed value; Slide, when the value is linearly sliding; Dwell and Switch, when the value is fixed for a number *n* of pulses and then changes; Stagger, when an *n* number of pulses, with different values, have a periodic sequence; Wobble, when the values have a periodic pattern, usually a sinus or a triangular wave; or Jitter, when the value is randomly generated, usually with a normal distribution (Revillon, 2019).

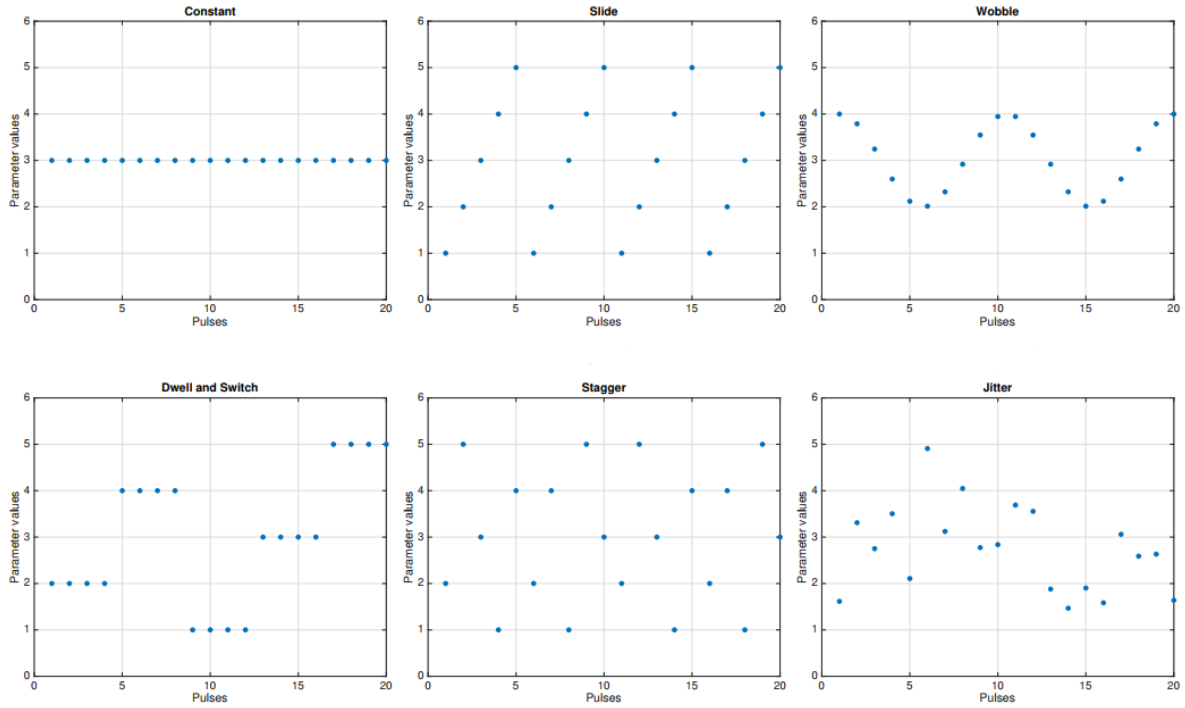


Figure 1.3 - Examples of different PRI modulations (Revillon, 2019, p. 55)

Typically, a radar signal is conceived as a pulse-to-pulse modulation to perform a specific role, such as missile guidance, surveillance, short-range tracking or other. Besides pulse radars, Continuous Wave (CW) radars also play significant roles. The main difference is that a CW radar continuously transmits a (fixed) high-frequency signal while processing the received echoes. As there are no pulses, it can only measure the target speed and not its distance. Though frequency-modulated CW radars can measure distance, as transmitting frequency is constantly shifting, the frequency can be extracted from the echo and, knowing when that particular frequency was sent out in the past, distance can be calculated (Revillon, 2019).

Electronic Warfare (EW) aims to react to threats detected in the electromagnetic spectrum. EW can be divided into three sectors: Electronic Attack (EA), Electronic Support (ES), and Electronic Protection (EP). While the aim of *attack* and *protection* is implied, *support* requires further detail – it comprises actions to search, intercept, identify and locate emitters. ES can be divided into Signal Intelligence (SIGINT) and Electronic Support Measures (ESM) (Aldossary, 2017).

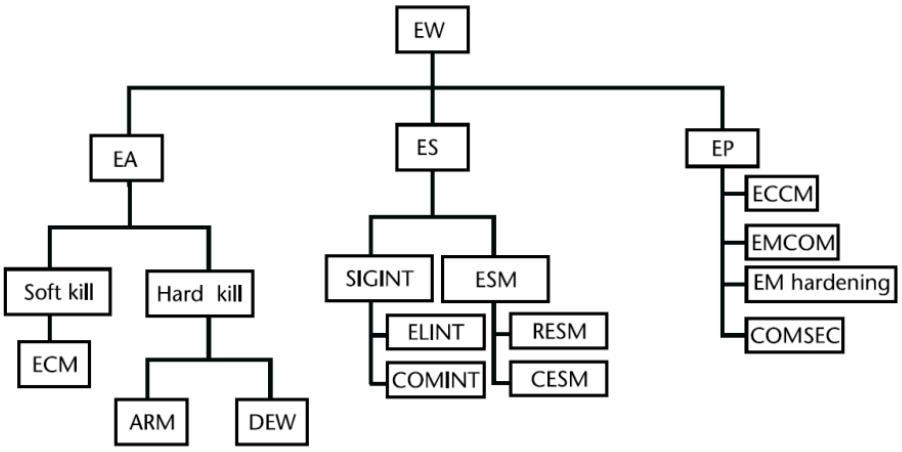


Figure 1.4 - Electronic Warfare divisions (Aldossary, 2017, p. 8)

ESM is the focus of this dissertation, specifically Radar ESM (RESM), whose ultimate goal is to classify radar emitters (Aslan, 2006). To achieve it, ESM systems must intercept pulses, characterize them, and record their characteristics in a Pulse Descriptor Word file (PDW). Some of those characteristics can be measured based on one pulse, called monopulse parameters: RF, PW, power, direction of arrival (DOA), and time of arrival (TOA). While others can be derived from it, such as PRI (Bildøy, 2006). In addition, modulation on pulse (MOP) is another parameter that becomes common in the current construction of the PDW and can be represented by a flag (Aslan, 2006). As seen previously, this modulation can be present in frequency (FMOP) but can also be in amplitude (AMOP), phase (PMOP) or even denote the presence of CW and other characteristics.

Before proceeding to classification, pulses are deinterleaved; this is, pulses are sorted to form pulse cells that are assumed to belong to a specific emitter designated as a track (Gençol, 2015). The accuracy of deinterleaving can have a significant impact on threat assessment and operations on the battlefield. Deinterleaving algorithms can be categorized into three groups: clustering algorithms, time-based algorithms, and combined algorithms, each with many algorithms (Aldossary, 2017). Still, as radars get more complex, for instance, with Low Probability of Intercepts (LPI) radars, detection gets more complicated and, consequently, the deinterleaving task.

The two main methods to classify radar emitters are the expert system method and the parameter matching method. The expert system method develops radar signal rules based on the experts' knowledge of radar properties. This method is known to have a low classification accuracy and a slow classification speed. In parameter matching method, which is widely used, pulse characteristics are directly compared with a radar parameter database, as in Figure 1.5. Although it has the advantage of being simpler to implement and having a fast classification speed, this method is too dependent on prior knowledge/records (J. Wang et al., 2022).

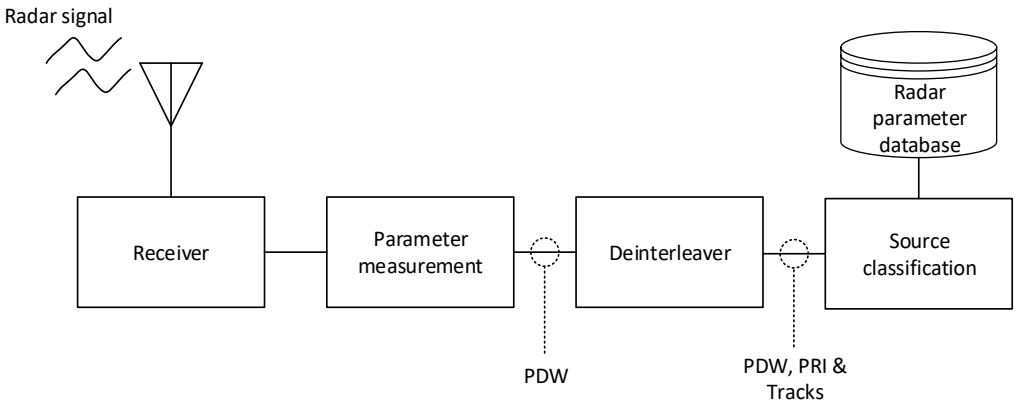


Figure 1.5 - General block diagram of radar ESM data processing

In ideal scenarios, emitters and signals are properly isolated from one another in time, frequency, and space, allowing PDW-based classification to work well. In congested scenarios, numerous emitters are degraded by noise, interference, fading, clutter, multipath and pulse overlap (Aslan, 2006; Buchenroth, 2015). In addition to environment and propagation degradations, the receiver has limitations, such as instantaneous band-width, automatic gain control and sensitivity (Aldossary, 2017; Meikle, 2008). Following these harsh conditions, several researchers have developed and applied cutting-edge machine learning and deep learning techniques to radar classification in the last decade. Some of those techniques are Feedforward Neural Networks (Petrov et al., 2013), Fuzzy Logic (Y. M. Chen et al., 2013), Weighted Extreme Gradient Boost (W. Chen et al., 2017), Support Vector Machine (SVM) (Sebastian, 2017), Unidimensional Convolutional Neural Network (CNN) (Sun et al., 2018), Deep Forest (Y. Wang et al., 2019; Zhang et al., 2020), Gaussian Mixture Model (GMM) (Revillon, 2019), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) (Notaro et al., 2019), Naïve Bayes (Kvasnov, 2020; Xiao & Yan, 2020), LSTM and Markov Chain (Apfeld & Charlish, 2021), Hybrid Deep Neural Network with dynamic CNN and LSTM (Feng et al., 2021), and Intuitionistic Fuzzy Information Tri-training (J. Wang et al., 2022). To validate the effectiveness of previous proposed models, Relevant Vector Machine, Deep Belief Network, Gradient Boost Decision Tree (GBDT), Extreme Gradient Boost (XGBoost), Random Forest, Extremely Randomized Trees (ExTrees), and K-Nearest Neighbor (KNN) have also been used. As machine learning is a data-driven discipline that comprises two essential steps, feature extraction and model training, some studies have also proposed different feature engineering approaches (Ahmed et al., 2018; W. Chen et al., 2017; Gençol, 2015; Notaro et al., 2019; Petrov et al., 2013; Sun et al., 2018; Y. Zhou et al., 2020).

Deep learning is a machine learning category that employs multiple layers of processing units, where each layer input comes from the previous layer output. Deep neural networks, an architecture of deep learning, have been successfully applied in several fields and achieved high accuracy rates (Young et al., 2018). Nevertheless, it also has some drawbacks: it requires large amounts of data to train, does not converge as fast as traditional machine learning algorithms, are complicated models, requires powerful computational facilities, has numerous hyperparameters, and the learning performance depends seriously on careful tuning (Young et al., 2018; Z.-H. Zhou & Feng, 2017).

Zhou & Feng (2017) argue that learning models will undoubtedly need to go deep to handle complex learning problems; however, deep models are always neural networks, with the disadvantages already known. As traditional machine learning algorithms are relatively simple to tune and their output may

provide interpretable results, novel deep ensemble models based on deep learning and using machine learning algorithms have recently been proposed (Qi et al., 2016; Young et al., 2018; Z.-H. Zhou & Feng, 2017). Moreover, following the research of Grinsztajn et al. (2022), when these algorithms are tree-based and used on medium-sized tabular data, with and without categorical features, it outperforms neural networks.

"*Reliable and real-time identification of radar signals is of crucial importance for timely threat detection, threat avoidance, general situation awareness and timely deployment of counter-measures*" (Petrov et al., 2013, p. 1194). Recognizing the importance of ESM classification, the novelty of deep ensemble models, and its performance, the following research gap is considered: Classification of emitters in PDW using a deep ensemble model. Accordingly, this study aims to develop a deep ensemble model based on deep forest, to classify radar emitters in PDW as an alternative to standard classification methods. Therefore, the following research question is: How can a deep ensemble model, based on deep forest, be used to classify radar emitters in PDW of ESM systems?

Since this investigation aims to design an artifact – a deep ensemble model – a proper methodology is required, leading to the adoption of design science research methodology (DSRM). This methodology has six activities – problem identification and motivation, the definition of objectives for a solution, design and development (of the artifact), demonstration, evaluation, and communication, developed along the dissertation. In general, deep ensemble models and their base learners presented such a competitive performance that a practical implementation may accurately classify emitters degraded by noise while achieving the main goal – being an alternative to standard classification methods, among other benefits and contributions.

This dissertation is divided into six chapters, beginning with the presentation of the context, main objective and research question, a summary of the methodology, results, and contributions, and ends with a brief description of its structure. Then, a review of deep ensemble models, pertinent base learners, and some application studies are presented. The third chapter describes the methodology, and the fourth chapter outlines the procedures followed from data collection to model evaluation on different datasets. The results are then presented and analyzed, in the fifth chapter, along with a comparison to earlier research and an evaluation of effectiveness. The final chapter summarizes conclusions and limitations before restating the most relevant contributions and suggesting future developments.

2. LITERATURE REVIEW

This chapter aims to enhance the knowledge of deep ensemble models and gain a perspective on previous research organized chronologically.

2.1. CASCADE GENERALIZATION

The ability of an algorithm to induce a good generalization depends on the appropriateness of its representation to express generalizations for a given task. This can be achieved in two ways: selecting the most appropriate algorithm or combining predictions of algorithms with different search heuristics, for instance, stacked generalization (Gama & Brazdil, 2000).

Stacked generalization refers to any scheme of forwarding information from one layer of models to another before forming the final prediction. This is a suitable method for correcting the constituent models' biases towards a learning set (Wolpert, 1992). In this sense, Gama & Brazdil (2000) proposed a family of algorithms under the generic name of cascade generalization, which is considered a particular case of stacking generalization. The basic idea of cascade generalization is to use learning algorithms in sequence and perform the following two-step process at each iteration: first, a model is built using base classifiers, and then the original data is extended by inserting new attributes. This is, all classifiers have access to the original attributes (independent variables) and produce new continuous attributes with the class probability (dependent variable), which are appended to the original data and used on the next iteration/layer (Gama & Brazdil, 2000).

The research concluded that these models could improve the accuracy of the base classifiers but could not state why it occurred, how many base classifiers are needed, or what classifiers to combine. It was also concluded that in some datasets, these models could not improve the accuracy (Gama & Brazdil, 2000). In short, the authors could not predict under what circumstances cascade generalization would improve performance.

2.2. DEEP SUPER LEARNER

The deep super learner model uses a super learning ensemble on each layer. Thus, the super learner must be introduced prior to its cascade version.

2.2.1. Base Learner

The super learner is a stacking ensemble learner of heterogeneous base learners that uses v-fold cross-validation to protect against overfitting and optimizes the weights of the base learners by minimizing a loss function (Laan et al., 2007). As represented in Figure 2.1, this model works as follows:

1. Split data in equal size *v*-fold blocks, mutually exclusive.
2. Build and train each *m*-base learning algorithm on each training block.
3. Predict the class probability of the validation block for each base learner, and get a Z matrix of *v***m* shape.
4. Build the minimum cross-validated risk predictor, for instance, a linear regression to find the linear combination of the Z predictions that minimize the error against true values Y. This function provides the weights to apply at each base learners.

5. Re-train each base learning algorithm on the entire dataset, represented as step 0 in Figure 2.1, and combine their predictions by applying the cross-validated risk minimizer.

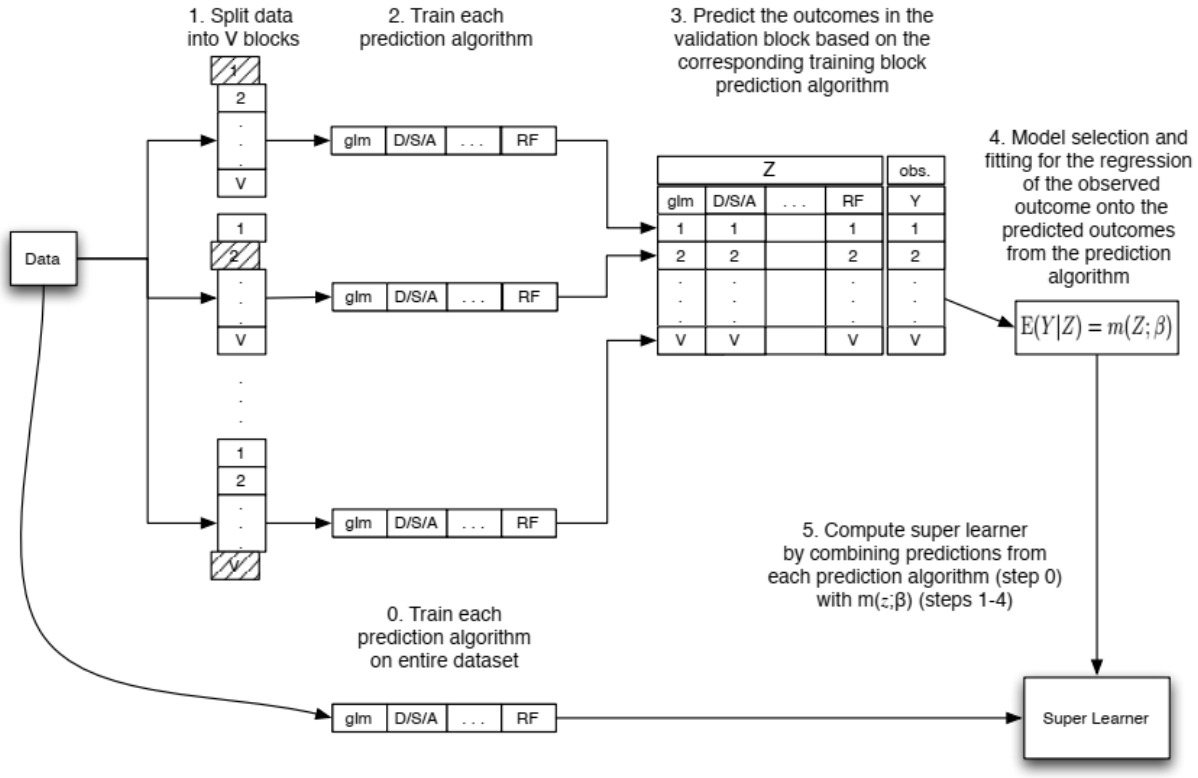


Figure 2.1 - Overall procedure of Super Learner (Laan et al., 2007)

Previous developments have shown that the super learner would theoreticaly perform asymptotically as well or better than any of its base learners, which was verified in practice by Laan et al. (2007) – when applied to generated dataset the super learner was able to outperform its base learner, but when applied to diabetes dataset and to HIV-1 drug resistance dataset it wasn't able to outperform its base learners but also did not any worse; which prompted the researchers to claim that when a base learner makes accurate predictions it leaves little room for the super learner to improve. Additionally, as it is *"unlikely to know apriori which candidate learner will work best, the super learner is a natural choice for prediction"* (Laan et al., 2007, p. 17).

Naimi & Balzer (2018) demonstrate the implementation of the super learner to estimate a dose-response curve and to build a classifier for a binary outcome. Within this study, a few essential considerations were highlighted:

- The number of v-fold cross-validation is critical for the super learner, and its choice is not always clear. So, increasing the number of folds as the sample size decreases is recommended.
- A wide variety of base learners can be included in the super learner, and it is recommended to include standard parametric models along with more complex data-adaptative models.

- Since hyperparameter tunning is a critical step in optimizing machine learning algorithms, it is recommended to replicate the same algorithm with different parameter values.
- As it is possible to choose any loss functions to determine the optimal combination of predictions directly impacting performance, it is recommended to choose that function based on the analysis objective.

Although this model appears large and complex, the computation cost does not exceed the computation required to train each base learner on v-folds and the full dataset (Laan et al., 2007).

2.2.1.1. Application example

Taghizadeh-Mehrjardi et al. (2021) used the super learner to predict several soil properties, typically obtained by measuring soil samples from related environmental covariates (groundwater data, terrain attributes, geomorphic map, and remote sensing data). A major challenge is quantifying each environmental covariate's importance to the final prediction. To address this issue, a model-agnostic interpretation tool was used. Furthermore, permutation feature importance analysis was selected mainly for its ability to rank features considering the interaction with other features.

Before modeling, to test accuracy, the dataset was randomly partitioned into two sets – 80% for training and 20% for testing. The training dataset was then split in five-fold (step 1 of the procedure) to train 12 base learners of the super learner: least absolute shrinkage and selection operator (LASSO), KNN, genetic programming, adaptive-network-based fuzzy inference systems, multi-linear regression, multivariate adaptive regression splines (MARS), support vector regression, artificial neural network (ANN), cubist, random forest, ExTrees, and XGBoost. The selected cross-validated risk minimizer, a meta-learner that uses the cross-validation predictions, was a generalized linear model (step 4 of the procedure). Moreover, to compare the results, an equal-weight combiner was used (1/12).

The super learner was consistently the best model to predict all soil properties, and the equal-weight combiner was the second. The researchers concluded that no single algorithm could be the 'best' for every region and soil property. Another interesting conclusion relates to base learners' contribution/weights for final prediction. Although the performance of most base learners varies from one soil property to another, the multi-linear regression was consistently the worst, but it still contributed to super learner predictions. This led researchers to conclude that super learner is exceptionally flexible in incorporating different base learners and evaluating their individual contributions to the final prediction.

2.2.2. Deep model

The deep super learner is an extension of the super learner ensemble, using a super learner on each layer, with some nuances. The initial four steps are similar to the super learner, the fifth is changed, and two additional steps are implemented (Young et al., 2018). As represented in Figure 2.2, this model works as follows:

1. Split data in equal size *k*-fold blocks, mutually exclusive.
2. Build and train each m-base learning algorithm on each training block, and save the *k***m* trained models.

3. Predict (*j*) classes probability of the validation block for each base learner.
4. Build and minimize a loss function on cross-validated predictions, and save the loss function weights.
5. (Option1, as proposed in super learner) Re-train each base learning algorithm on the entire dataset and combine their predictions. Then save the *m*-trained models.
   (Option 2, a novel approach) Calculate the weighted average of the cross-validated predictions for each record.
6. Append the overall predictions to the original training data.
7. Repeat the previous steps with the augmented data until the loss function value no longer decreases and save the number of iterations.

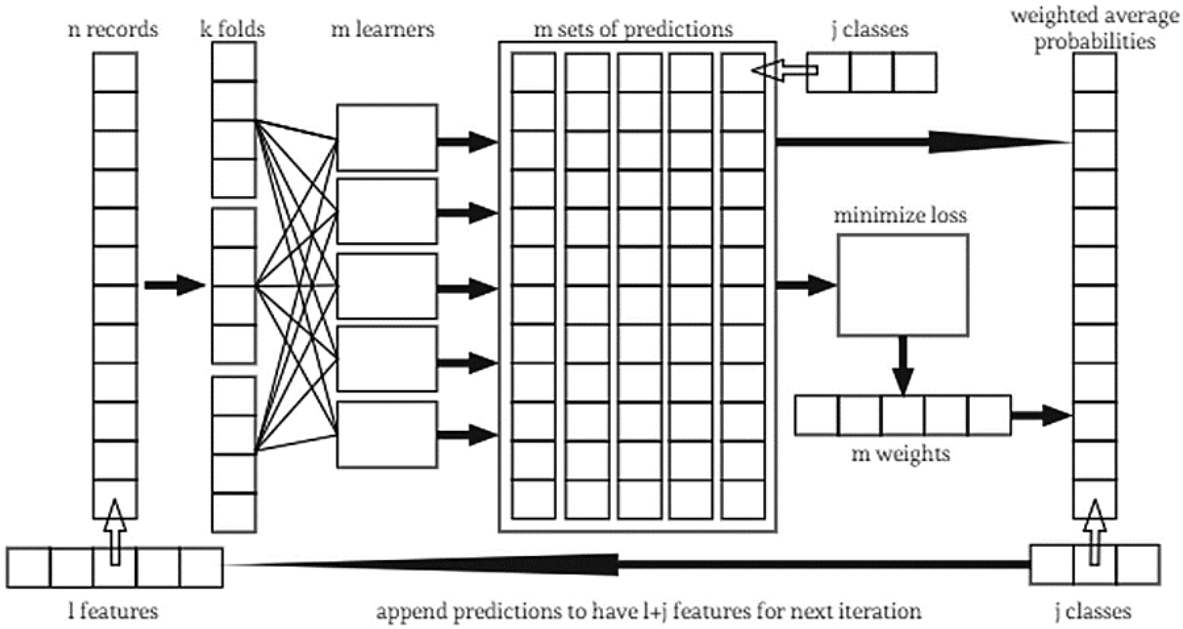


Figure 2.2 - Overall procedure of Deep Super Learner (Young et al., 2018, p. 87)

The unseen data must pass through a similar process (of the training), using the models and weights of each iteration to make predictions. If the base learners are trained on the entire training dataset (option 1 of step 5), use those models to make the predictions. If the base learners are trained on k-folds (option 2 of step 5), use each model trained on each fold to make the predictions.

Young et al. (2018) implemented a new approach in step 5, arguing that with sufficient cross-validation folds the base learners will have enough data to train, and additional data does not improve the goodness of fit. However, authors recommend the original approach when using few folds or when making predictions is computationally more expensive than training, as with KNN.

In this research, Young et al. (2018) tested the deep super learner on the IMDB movie review sentiment classification dataset and MNIST handwritten digits dataset. Deep super learner was built on three-fold cross-validation with five heterogeneous base learners: logistic regression, KNN, random forest, ExTrees, and XGBoost. For loss function or *cross-validated risk minimizer*, log loss was chosen. To compare results, the five base learners are also tested individually, as well as three ensembles – a stacked ensemble where the output of the base learners is fed into XGBoost, an equal-weighted

average of the base learners, and a super learner; plus two deep neural network architectures – multi-layer perceptron (MLP) and a CNN. Regarding results, on the IMDB dataset, the deep super learner finished training after the third iteration and outperformed all other models. On MNIST, the deep super learner finished training after the fifth iteration and could not perform better than CNN. As a runtime reference, on the IMDB dataset, deep super learner took 50 minutes to converge; 46 of those were spent by KNN, and on MNIST took 86 minutes, where 70 of those were spent by KNN.

2.2.3. Application example

A WebShell file is an executable program written with web scripts, such as PHP, to be used as a website backdoor. These malicious files allow attackers to obtain database information. Consequently, finding website backdoors is essential for data security. In this sense, Ai et al. (2020) propose to use deep super learner to detect WebShell.

This application's dataset comprises PHP files with instructions or fields for a computer program. Thus, in data preprocessing, some feature extraction operations are executed and feature vectorization, using Word2Vec. Then, to remove irrelevant or redundant features and keep the most effective ones, feature selection is performed based on a genetic algorithm. Moreover, to fix dataset class imbalance, data sampling with the synthetic minority oversampling technique (SMOTE) was implemented. The modeling of deep super learner uses logistic regression, MLP and random forest as base learners and sequential least squares programming to calculate the models' weight value (Ai et al., 2020).

As noted by Naimi & Balzer (2018), the value of *k*-fold is a parameter of utmost importance in the (deep) super learner; therefore, Ai et al. (2020) tested different values, from three to ten-fold cross-validation, achieving the best performance on seven-fold.

To verify and compare the deep super learner performance, two base learners are used – logistic regression and random forest, complemented with XGBoost, LightGBM, Adaboost and a stacking ensemble model. Deep super learner achieved the best performance, yet, in terms of time efficiency on the test dataset, was the second slowest model, with 2.89 seconds, close to XGBoost, with 2.47 seconds, and far from the slowest model – LightGBM, with 8.43 seconds.

2.3. DEEP SVM

Deep learning is a sophisticated machine learning method that identifies lower-level features and inputs them to the next layer to identify higher-level features, improving overall classification performance (Young et al., 2018). These architectural designs have succeeded in various applications, particularly visual and speech information (Pang et al., 2022).

Inspired by deep belief networks, Abdullah et al. (2009) presented a deep support vector machine (D-SVM) and an ensemble of D-SVM for image categorization. The D-SVM is obtained through training one SVM in the standard way, and creating a new training dataset with labels and kernel activations, to be used on the next layer SVM. Concerning the proposed ensemble (see Figure 2.3), this calculates probability estimates for each class using a set of D-SVM, each receiving a distinct set of features as input. Then, class probabilities are combined with the product rule, multiplying each class probability and using the highest probability as the final class label. Experiments with Corel and Caltech-101 datasets revealed that D-SVM has a higher accuracy than standard SVM, but not quite significant.

Moreover, regarding the ensemble a similar result was obtained – the ensemble of D-SVM performed slightly better than an ensemble of SVMs.

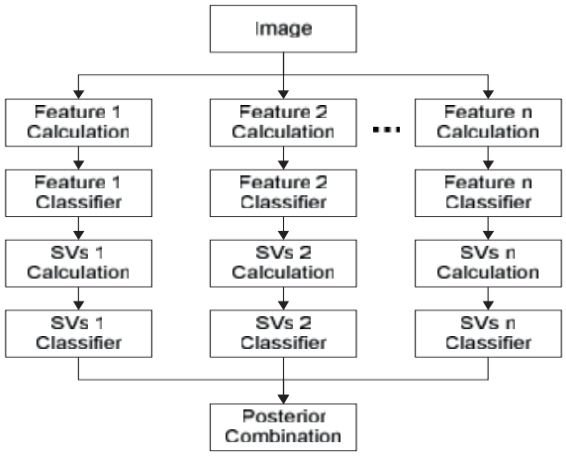


Figure 2.3 - Ensemble of D-SVM (Abdullah et al., 2009, p. 303)

As the work of Abdullah et al. (2009) and other researchers had a fixed structure of just two layers of SVMs and actually did not have a deep structure, Kim et al. (2015) proposed a new deep network model. This new structure consists of (*k*) SVMs layers to perform k-class classification, selecting the label with the maximum output as the final prediction (see Figure 2.4). At each layer, a class probability output network (CPON) measures uncertainty when the classification decision is made. If the required certainty is met, the classification is performed in that layer; if not, the CPON output continues for the next layer. Experiments with four datasets from the UCI Machine Learning and the MNIST dataset, comparing the proposed model with Naïve Bayes, single-layer SVM, single-layer SVM with CPON, and deep belief networks, showed its competitiveness – outperforming the other models in all datasets, except in Ionosphere.

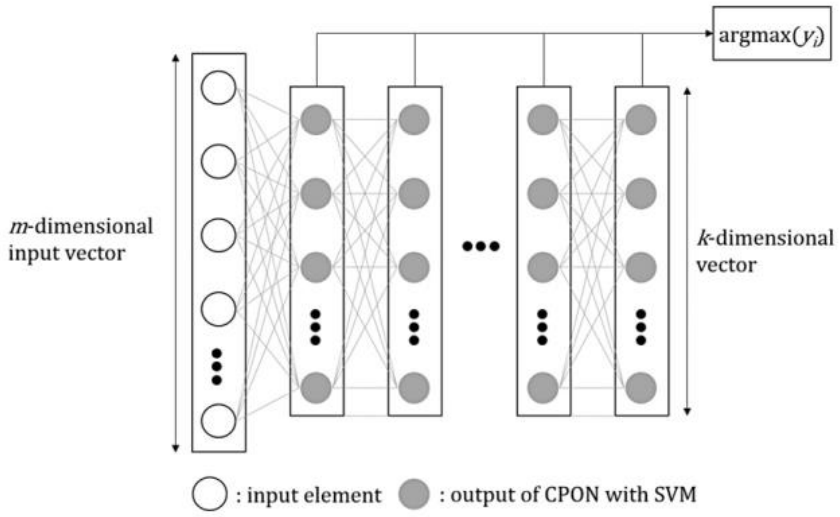


Figure 2.4 - Structure of deep SVM (Kim et al., 2015, p. 23)

Later, Qi et al. (2016) proposed a new deep support vector machine called DeepSVM. This model has a multi-layer architecture and extracts features via Adaboost (Ex-Adaboost) on each layer. This Ex-

Adaboost can adjust kernel parameters, classifier weight coefficients and feature's weight for the next layer. Experiments on eight UCI Machine Learning datasets, comparing the proposed model with a version of himself without Ex-Adaboost, showed that DeepSVM achieves a slightly better performance, which proves that Ex-Adaboost promotes data representation. When comparing DeepSVM accuracy with the model proposed by Kim et al. (2015), DeepSVM underperformed in all three common datasets.

2.4. DEEP FOREST

As in other research above, Zhou & Feng (2017) presented a deep ensemble model intending to endow deep learning properties without shortcomings. The proposed model, gcForest (multi-Grained Cascade Forest), comprises two parts – multi-grained scanning and cascade forest. The first part seeks to facilitate spatial or sequential interpretation, while the second does representation learning. To encourage diversity gcForest uses two types of forest – random forest and completely-random tree forest. This last forest contains a defined number of completely-random trees produced with randomly selected features at each node and growing until each leaf becomes pure.

As represented in Figure 2.5, this model works as follows:

Multi-Grained Scanning:

1. Take *d*-dimension raw input features and slide multiple-size windows (suggested [d/16], [d/8] and [d/4]) to generate feature vectors. If d=400 and window=100, a total of 301 feature vectors are produced.
2. Each forest produces a probabilistic estimate of class distribution for each feature vector. Considering the previous 301 feature vectors, if the dataset has three classes, each forest will produce 301 three-dimensional class vectors (leading to a 903-dimensional transformed feature vector).

Cascade Forest:

3. The transformed feature vector will then be used to train the first layer of the cascade forest and produce class estimates generated by *k*-fold cross-validation using *k*-1 folds. Considering a layer with four forests and the previous dataset with three classes, a 12-dimension class vector is produced.
4. The generated class vector, augmented alternately with the transformed feature vectors, is then used to train the next layer.
5. After adding a new level, the holdout fold (validation set) is used to estimate the performance of the entire cascade, and if there is no significant improvement in performance, the training procedure ends.
6. Reaching the last layer, an average class vector is produced, and the class with the highest value is taken as the final prediction.

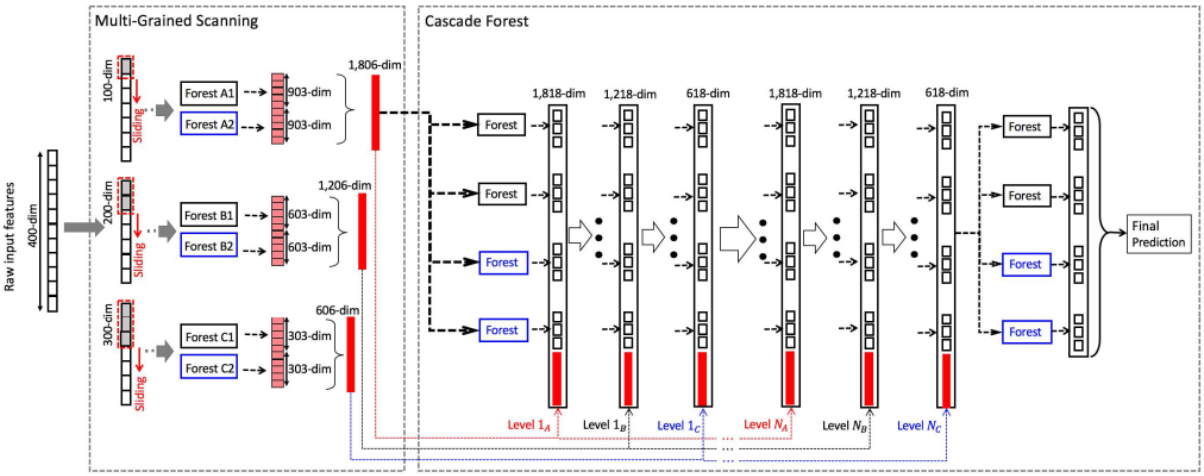


Figure 2.5 - Overall procedure of Deep Forest (Z.-H. Zhou & Feng, 2017, p. 3555)

Data must pass a similar process, starting with the multi-grained scanning procedure to obtain the corresponding transformed feature representation and proceed with the cascade until the final prediction is obtained.

Zhou & Feng (2017) compared gcForest with deep neural networks and several other popular learning algorithms on five different datasets for music classification (GTZAN), image categorization (MNIST), hand movement recognition (sEMG), face recognition (ORL), and sentiment classification (IMDB), plus three low dimensional datasets (LETTER, ADULT, YEAST). The proposed model achieved a highly competitive performance obtaining the best accuracy in all experiments. Besides having fewer hyperparameters than deep neural networks, this model can get excellent performance across various domains with default settings. Moreover, to exemplify the runtime efficiency, gcForest took 40 minutes to converge on the IMDB dataset, while MLP required 77 minutes.

2.4.1. New versions

Since the introduction of gcForest numerous researchers have suggested new deep forest configurations to address a specific purpose or a specific challenge. Guo et al. (2018) proposed BCDForest (boosting cascade deep forest) to be suitable for small-scale class-imbalanced biology data. This model has a multi-class-grained scanning approach (instead of multi-grained scanning) to encourage the diversity of ensemble forests and an additional output feature in each forest of the cascade (the standard deviation of the top-k most important features) to boost important features. As result, BCDForest consistently outperformed gcForest on most cancer datasets, demonstrating that the proposed boosting strategy has improved the models' classification ability.

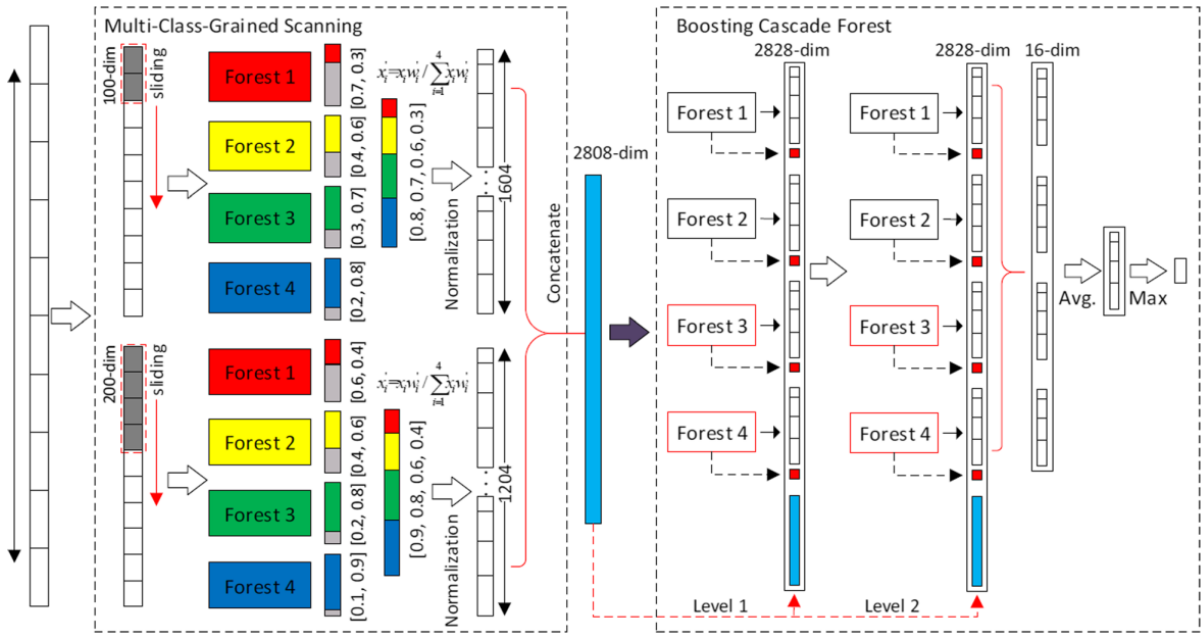


Figure 2.6 - Overall procedure of BCDForest (Guo et al., 2018, p. 6)

In essence, gcForest can automatically determine its complexity and perform well across various tasks, but it can also be computationally expensive. This happens because the transformed feature vector can be significantly bigger than the original input feature vector, and all instances must pass through all cascade layers. Consequently, multi-grained scanning demands time and memory while cascade forest linearly increases time with the number of layers in the cascade. Even though users can balance this by choosing the number of grains, forests, and trees, Pang et al. (2018) introduced a confidence screening mechanism to reduce time and memory costs. This new model, called gcForestCS, has the following modifications (subsampling multi-grained scanning, confidence screening mechanism, and variable model complexity):

1. Feature vectors formed by sliding windows in multi-grained scanning are randomly sampled (10%) before being used to produce class estimates.
2. Measure prediction confidence on each cascade layer, and if an instance requires a higher level of learning, it is pushed to the next layer; otherwise, it is predicted using the model at the current layer.
3. As the instances that reach higher cascade layers are harder to predict, forests must increase complexity, requiring more trees in each forest.

The proposed model was compared to gcForest, and instead of using the same model on all datasets, dependent if datasets held spatial or sequential relationships, the models were tested with or without multi-grained scanning. With the results, it is possible to conclude that gcForestCS uses an order of magnitude less memory than gcForest while achieving comparable or better accuracy and faster runtime.

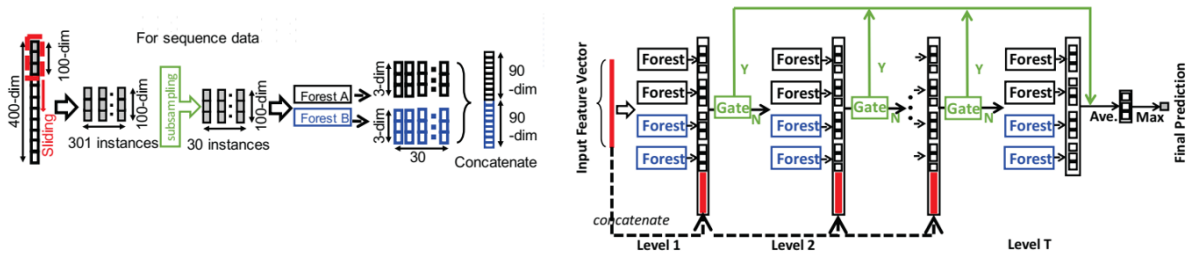


Figure 2.7 - Illustration of gcForestCS (Pang et al., 2018, p. 1195)

Attempting to improve the efficiency and performance of gcForest, Ni & Kao (2020) proposed a PSForest model consisting of two parts – multi-grained pooling and gate-cascade forest. In multi-grained pooling, the output of each sliding window is filtered to keep the feature with the higher value. Then, all class vectors generated are concatenated with the raw feature vector and delivered to a gate-cascade forest. In this part, instead of using all class vectors in all cascade layers, some class vectors are filtered out at each layer based on the out-of-bag error of each forest. Thus, the input vector of each layer is made of the initial transformed feature vector concatenated with the filtered output of the preceding layer. Experiments on various datasets demonstrate that PSForest achieves comparable or higher accuracy than gcForest while using less memory and time.

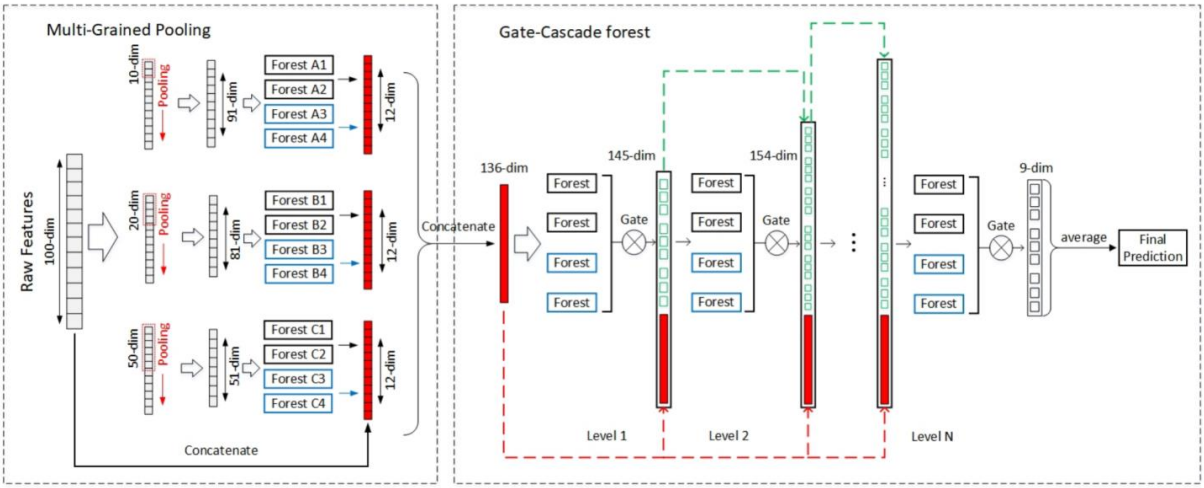


Figure 2.8 - Overall procedure of PSForest (Ni & Kao, 2020, p. 775)

Y.-H. Chen et al. (2021) argue that gcForest requires a large amount of memory to store multi-layered forest models, and it is very time-consuming to make predictions. Moreover, as forests are stable models, an ensemble of forests may produce similar predictions, resulting in redundant feature representation and reduced diversity. With this drive, researchers proposed a novel (cascade) model named high-order interaction deep forest (hiDF). This model operates in three steps: it starts by extracting decision rules from random forests, which are then processed by *generalized random intersection trees* to select the features with better generalization. Secondly, a new feature is generated by *activated linear combinations*, concatenated to input features, and moved to the next layer. Then (third step), the current structure is evaluated on a validation set to avoid overfitting, meaning the training stops if there is no performance improvement. The result of experiments on ten benchmark datasets, comparing the proposed model to gcForest, gcForestCS, random forest, XGBoost,

GBDT, and SVM, demonstrates a superior accuracy performance of hiDF on all datasets. Additionally, hiDF requires less time on the test dataset and about one order of magnitude less memory.

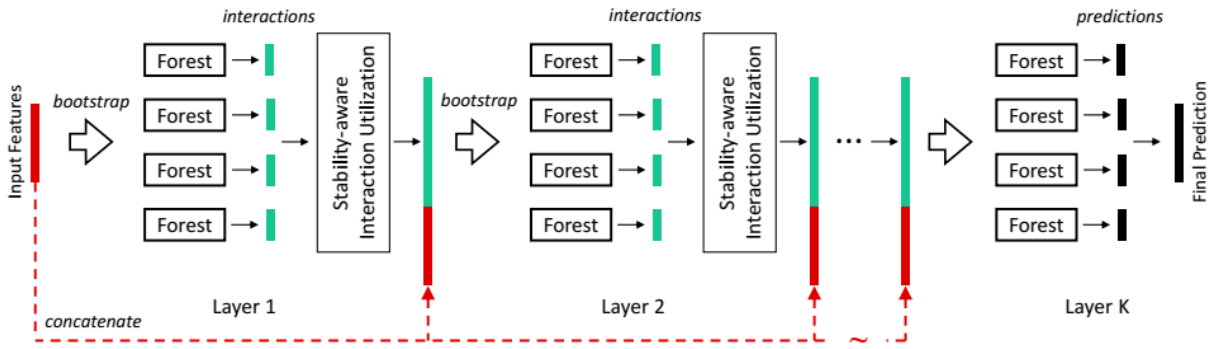


Figure 2.9 - Illustration of hiDF (Y.-H. Chen et al., 2021, p. 1031)

Ma et al. (2022) claim that gcForestCS, with default hyperparameters, achieves lower prediction accuracy than the original deep forest model, and mis-partitioned instances cause that. To overcome this issue, the authors proposed a new deep binning confidence screening forest (DBC-Forest) model. The difference between this model and gcForestCS resides in the strategy used to comply with the confidence threshold. The new model groups the instances into bins based on their confidence values and determines the bins' accuracy to select which will move to the final prediction and which will continue for the next layer. Experimental results on nine datasets, comparing DBC-Forest with gcForest and gcForestCS provide the basis for the following remarks: DBC-Forest is more robust to hyperparameter settings; DBC-Forest generally achieves higher accuracy; on low-dimensional datasets, the results between models are not very different. Regarding time efficiency, on training, DBC-Forest was the fastest on seven datasets, gcForestCS was the fastest on the EMNIST dataset, and gcForest was the fastest on YEAST dataset.

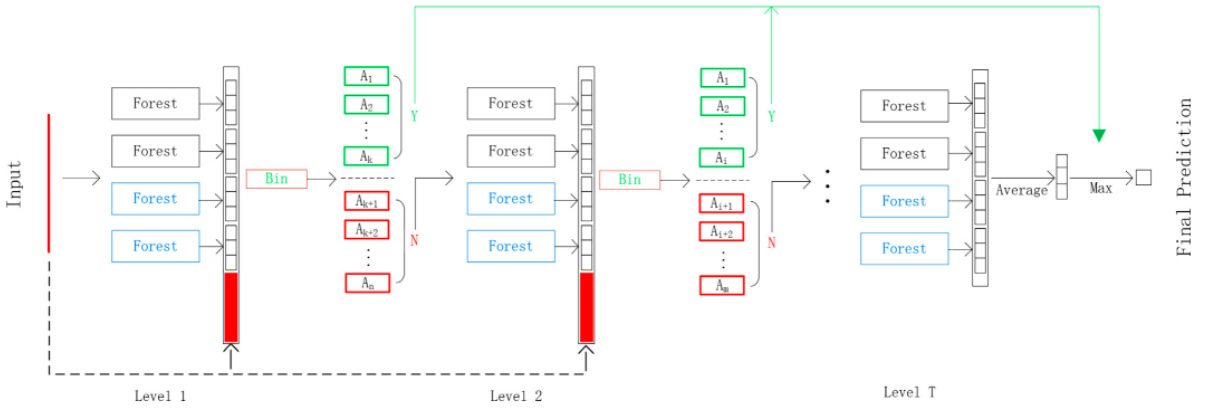


Figure 2.10 - Illustration of DBC-Forest (Ma, Wu, Li, Guo, & Li, 2022, p. 115)

Likewise, Ma, Wu, Li, Guo, Jiang, et al. (2022) argue that gcForestCS produces redundant feature vectors in multi-grained scanning and that confidence screening depends on hyperparameter tuning. The authors propose an HW-Forest to improve its performance, which adopts two screening mechanisms – window screening and hashing screening. This last uses perceptual hashing to evaluate the similarity between feature vectors, then removes the redundant vectors generated in multi-grain scanning. Window screening is an improvement to the DBC-Forest strategy that uses windows with

variable sizes to calculate the confidence threshold. For the performance evaluation of HW-Forest, the authors compared ten benchmark datasets with four analogous models – gcForest, gcForestCS, DBC-Forest, and AWDF (adaptive weighted deep forest). Experiments show that HW-Forest achieves higher accuracy than DBC-Forest and the other models. Concerning time cost, gcForestCS was the fastest model on seven datasets and HW-Forest on three.

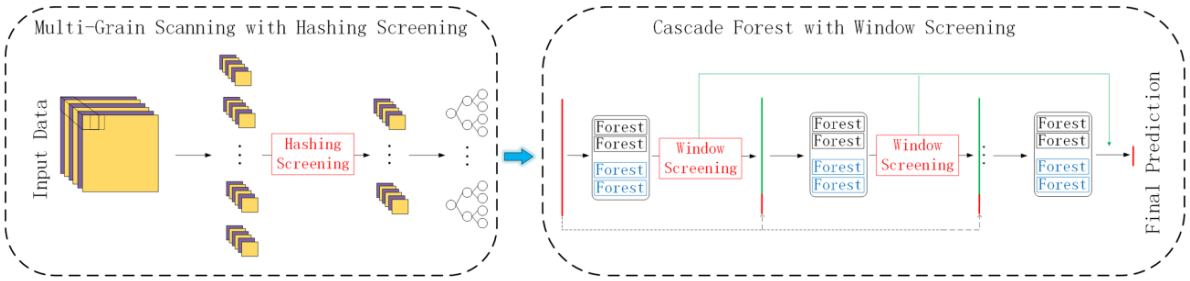


Figure 2.11 - Overall procedure of HW-Forest (Ma, Wu, Li, Guo, Jiang, et al., 2022, p. 10)

To further improve the previous model (gcForestCS), Pang et al. (2022) propose replacing the multi-grained scanning with a new version designated completely-random forest transformation, which uses a completely-random forest to get a smaller transformed feature vector. This new proposed model, gcForestS, keeps the cascade modifications of gcForestCS (variable model complexity and confidence screening) and adds a feature screening to select features that are significant for performance improvement at each level. Experiments with and without multi-grained scanning on eight different datasets demonstrate that gcForestS improves accuracy while reducing runtime and memory usage. Also, experiments with and without completely-random forest transformation on gcForest and gcForestCS enabled those models to achieve better accuracy in 2 (out of 3) datasets.

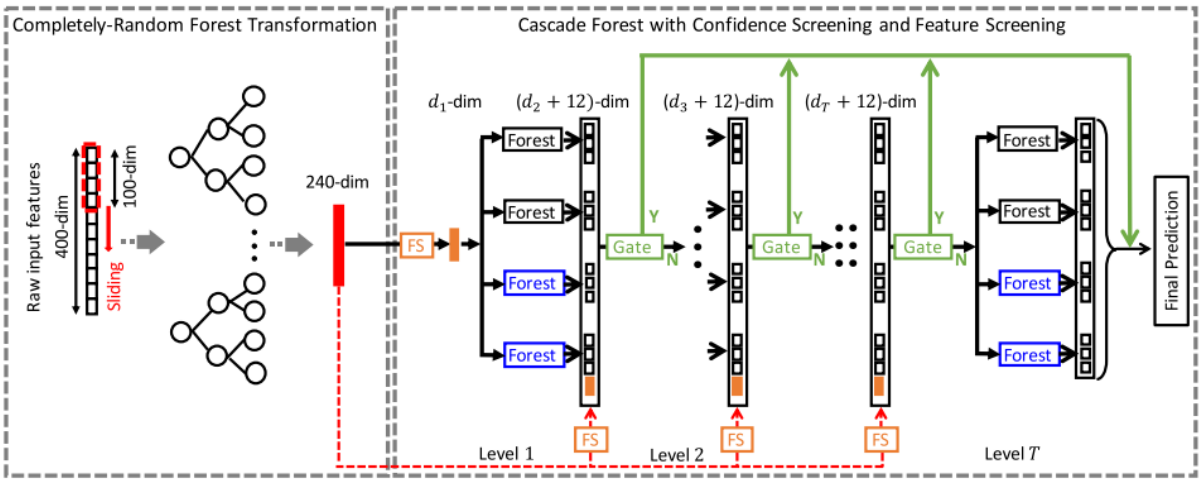


Figure 2.12 - Overall procedure of gcForestS (Pang et al., 2022, p. 4303)

Although some standard datasets and models haven been used in the research mentioned above, mainly for comparison, the accuracy they presented has variations. Table 2.1 presents the difference in accuracy to gcForest to understand the performance between models.

Table 2.1 - Comparison of accuracy between models on six datasets

	sEMG	**MNIST**	**CIFAR10**	**LETTER**	**ADULT**	**IMDB**
gcForestCS	+1.29	0	+0.84	0*	+0.05*	+0.37*
PSForest	-	+0.28	+0.05	-	-	-
hiDF	-	-	-	-	**+0.73**	-
DBC-Forest	-	+0.26	+1.30	+0.05	+0.12	+0.58
HW-Forest	-	**+0.30**	+1.64	**+0.17***	+0.19*	**+0.77***
gcForestS	**+6.11**	+0.06	**+4.43**	+0.01*	+0.24*	+0.36*

*: without multi-grained scanning

2.4.2. Application example

Radars have become an extremely important sensor in various applications, including unmanned aerial vehicle detection, perimeter surveillance, and autonomous driving, which requires methods for target recognition. Towards this objective, a radar high-resolution range profile (HRRP) is a one-dimensional feature used to characterize a target. In this context, Y. Wang et al. (2019) tested deep forest (gcForest) for HRRP recognition. Without multi-grained scanning, the model consists of two random forests, two ExTree, two XGBoost, and two GBDT. Although raw HRRP has one dimension, the authors could extract 36-dimensional features and input them into the model. The performance of deep forest was compared with one SVM and all base learners, which are ensembles. As a result, deep forest reached the highest accuracy, and the second best was XGBoost (2.44% less accuracy).

Accordingly, this research demonstrates the added value of deep forest in the context of radar recognition. The results show a significant improvement over the traditional approach, which is not surprising given the forementioned research.

3. METHODOLOGY

This chapter is divided into two sections, with the first portion providing a theoretical overview of design science (DS) research to ease the comprehension of its application in the following section.

3.1. DESIGN SCIENCE RESEARCH

Science and Technology have different goals, processes, and drives, but both advance and evolve in complex interactions. While science aims to grow descriptive knowledge based on the natural world and human behavior, technology aims to grow prescriptive knowledge based on purposefully designed theories and artifacts (methods, models, constructs, or instantiations). The interactions between science and technology happen in two ways: science informs technology via solid foundations in domain knowledge, and technology informs science via innovative solutions to practical problems (Baskerville et al., 2018; Hevner et al., 2004).

"New technologies are driven and enabled by science, but, more often, scientific advances are driven and enabled by the emerging use of technology" (Baskerville et al., 2018, p. 361).

DS paradigm has its roots in engineering and is oriented to solve problems. When applied to information systems (IS), build and evaluate artifacts intended to solve organizational problems. Due to the science-technology complementary relation, Hevner et al. (2004) proposed a framework to position IS research within that relation (see Figure 3.1).

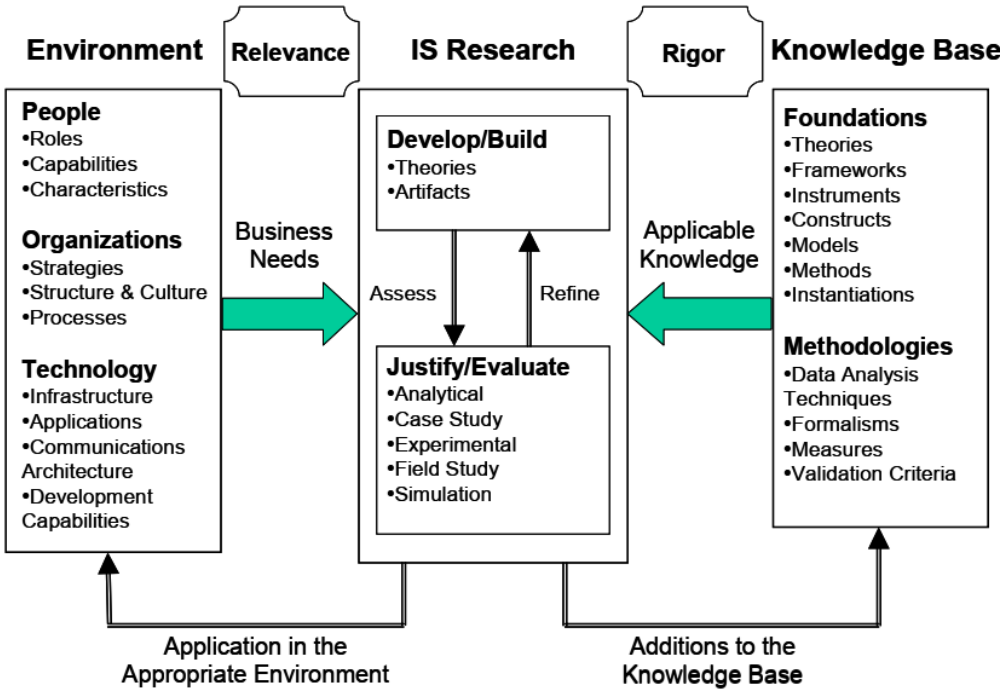


Figure 3.1 - IS research framework (Hevner et al., 2004, p. 80)

Several researchers have promoted the use of DS in IS since the early 1990s; however, 15 years later, few research has been published with a DS approach. Peffers et al. (2006) have proposed a DS research process for production and presentation of DS research in IS to address this issue. This was latter enhanced to be designated as a design science research methodology (DSRM) process, presented by

Peffers et al. (2007), incorporating principles, practices, and procedures. DSRM was developed using a consensus-building strategy combining seven representative papers on DS. The result, as represented in Figure 3.2, is a process model made up of six activities in sequential order, with four possible entry points and an iterative process to improve the artifact (Peffers et al., 2007):

1. Problem identification and motivation – Define the research problem, capturing its complexity and the importance of a solution. This activity requires knowledge of the problems' nature and the value of a solution.
2. Define the objectives for a solution – Establish the objectives of a solution (either quantitative or qualitative) from the problem description and an assessment of what is appropriate. Resources required for this activity include knowledge of problems' current state and existing solutions.
3. Design and development – Determine the required functionality and architecture of the artifact and then build it. Resources required for this activity include knowledge of theories that can be applied to a solution.
4. Demonstration – Identify how the artifact can solve the problem or some of its instances. Resources required include an adequate understanding of the artifact and its use.
5. Evaluation – Compare the objectives from activity 2 to the observed results of the artifact in activity 4. Depending on the nature of the problem, this activity could take many forms, such as satisfaction surveys, client feedback, response time, availability, simulations, system performance, and empirical evidence or logical proof. At the end of this activity, the researchers can decide whether to iterate back to *design and development* or continue for the next activity and leave further improvement to subsequent projects. Resources required for this activity include knowledge of relevant metrics and analysis.
6. Communication – Describe the problem and its importance, the design of the proposed artifact and its effectiveness, novelty, and utility.

As referred, there are four possible entry points in the DSRM process:

1. In activity 1, problem-centered approach – If the inspiration for the research came from the observation of a problem or a suggestion made in a previous project.
2. In activity 2, objective-centered solution – If an industry or a research need triggers the research.
3. In activity 3, design- and development-centered approach – If the research uses an artifact developed for a different problem.
4. In activity 4, client-/context-initiated solution – If the research seeks to improve the process of a solution already developed.

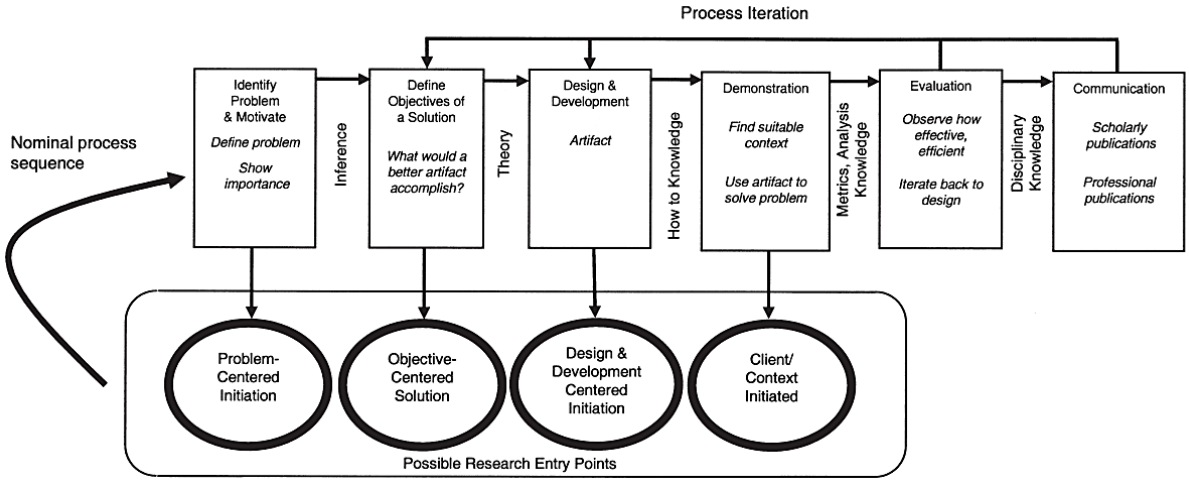


Figure 3.2 - DSRM process model (Peffers et al., 2007, p. 54)

As seen in Figure 3.1, DS research has two main outcomes: design artifacts and design theories. The first can be categorized as (Baskerville et al., 2018):

- Construct – Establish the basic concepts and language used to describe and communicate problems and solutions.
- Model – Use constructs to represent real-world contexts (problems and solutions).
- Method – Define processes such as solution algorithms.
- Instantiation – Embodies methods, models and constructs in a working system, enabling concrete assessment.

Designing artifacts allow researchers to learn about the real-world and how it is affected, leading to the development of design theories. Consequently, it is expected that some reflection of the form and functions of an artifact but not a fully formed theory on any project or article (Baskerville et al., 2018).

Gregor & Hevner (2013) analyzed DS research development and its different contributions to knowledge, urging the proposal of a DS research knowledge contribution framework (see Figure 3.3). This framework focuses attention on the start-point knowledge. It has two dimensions – the existing maturity of the problem (x-axis) and the existing maturity of potentially usable artifact (y-axis) for the research question, producing a 2x2 matrix with the following quadrants (Brendel et al., 2021; Gregor & Hevner, 2013):

- Invention – Proving new solutions for new problems. Projects in this quadrant are rare and involve research on novel applications where the problem is poorly understood and there are no effective artifacts available.
- Improvement – Presenting new solutions for known problems. The intention is to develop improved solutions, which can be in the form of more effective and efficient technologies, products, services, processes, or ideas.
- Exaptation – Extending known solutions to new problems. In this quadrant, knowledge that already exists in one field is extended or refined to another.

- Routine design – Proving known solution for known problems. In this quadrant no major knowledge is expected, but some research opportunities may arise when the application of existing knowledge in a routine way leads to surprises and discoveries, requiring the research to move to another quadrant.

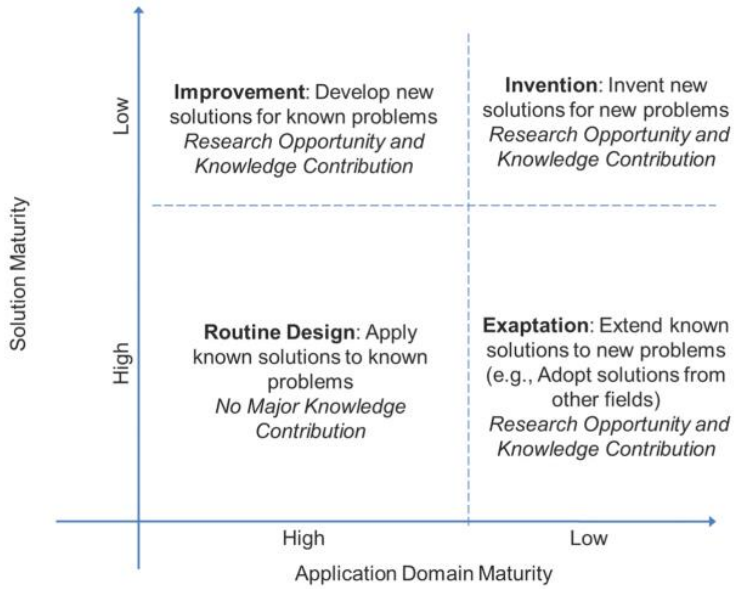


Figure 3.3 - DS research knowledge contribution framework (Gregor & Hevner, 2013, p. 345)

Gregor & Hevner (2013) also argue that prior guidelines for presenting DS research do not offer enough guidance to researchers and propose a publication schema in comparison to a conventional behavioral science paper:

1. Introduction – Contents are similar but should also include the goals for the artifact, specifying the scope and purpose.
2. Literature review – Should include relevant literature on knowledge or artifacts related to the problem at hand.
3. Method – The adopted research approach should be described and supported with reference to existing authorities.
4. Artifact description – This section differs markedly from conventional articles in the behavioral science. It is expected to have a description of the artifact and (may also include) the development process, at the appropriate level of abstraction.
5. Evaluation – Reports any evidence that the artifact does what it is meant to do, in accordance with the goals established.
6. Discussion – Presents an interpretation of the results, the novelty of the artifact and its contributions to knowledge.
7. Conclusions – Restates the main contributions to knowledge and why they are important.

3.2. Design Science Research Methodology

Relating DSRM activities to the publication schema and the present dissertation structure, it's possible to understand how to comply with DS research.

Table 3.1 - DSRM activities relation with chapters

DSRM activity:	Publication schema:	Dissertation structure:
Problem identification and motivation	Introduction	Introduction
Define the objectives for a solution		
	Literature review	Literature review
-	Method	Methodology
Design and development	Artifact description	Empirical study
Demonstration	Evaluation	Results and discussion
Evaluation	Discussion	
Communication	Conclusions	Conclusions and future works

It is clear at this point what to expect from each activity and where it is described. Still, a brief review of the first two activities is provided to ensure clarity.

Problem identification – Current radar classification methods heavily rely on the knowledge of experts to label records performed by ESM systems (e.g. PDW), to build rules of radar attributes when using the expert system method, or to characterize and input radar parameters in databases, when using the parameter matching method. Then, a system expert should test and tune these rules or parameters. It should be noticed that due to sensitivity, congestion management, and other system-specific characteristics, any record created by an ESM system and used to develop rules or parameters may not be suitable for another ESM system.

Objectives for a solution – The primary objective is to create a machine learning model, specifically a deep ensemble, that can classify radar emitters. To be an alternative, it must comply with three requirements:

- Be quick – As most radars can transmit thousands of pulses per second and may be traveling fast (more than 1000km/h in case of missiles or fighters), the classification process is required to be as quick as possible.
- Highly precise – Errors in classification can lead to higher threat assessment processing time, hampering timely reactions. Therefore, precision is required for better and faster responses.
- And robust – ESM operates in harsh conditions, affected by several environment, system, and propagation degradation, with impact on performance. To sustain the precision of classification and lessen this issue, the proposed machine learning model should be robust, this is, should be noise resistant.

To be presented in the next chapter, the design and development activity follows a set of steps adapted from Ferreira (2020) – the six phases of the data science loop.

1. Data exploration: A general assessment of the dataset is performed in this initial step. It enables familiarization with the data, verifying its quality, checking features correlation, and any issue to be solved in the data cleaning step. The following features are available in the dataset: TOA, DOA, RF, power, AMOP, FMOP, PMOP, CW, POP (Pulse On Pulse, interrupting pulse), POPed (Pulse On Pulse, interrupted pulse), and label.

2. Data cleaning: As in CRISP-DM (Cross Industry Standard Process for Data Mining), SEMMA (Sampling, Exploring, Modifying, Modelling, and Assessing) or KDD (Knowledge Discovery in Databases), this step aims to perform noise removal and transformations on data.

3. Feature engineering: This step creates new features from original data and transformations to enhance models' performance.

New Features:

- PRI, using the first difference of TOA between sequential pulses of each emitter[1].
- The second difference of TOA, based on Ahmed et al. (2018).
- The minimum, maximum and mean values of RF, PW and PRI, based on W. Chen et al. (2017) and Petrov et al. (2013).
- Rolling minimum, maximum, and mean values of RF, PW and PRI, based on the sequence normalization of Notaro et al. (2019).
- Binning RF, PW and PRI, based on Sun et al. (2018).

Transformations:

- Normalization with Min-Max scaling [0,1], based on W. Chen et al. (2017) and Petrov et al. (2013).
- Standardization scaling, based on Petrov et al. (2013).
- Normalization scaling [-1,1], based on Notaro et al. (2019).
- Sequence/rolling normalization [-1,1], based on Notaro et al. (2019).
- Principal Component Analysis (PCA), based on Gençol (2015) and Y. Zhou et al. (2020).

4. Model build: Deep forest is the foundation to build several deep ensemble models that replicate some research mentioned in the Literature review. Additionally, to compare results across datasets, their base learners must be built separately.

5. Hyperparameter optimization: This step aims to finetune each model for superior performance. Models can also be tested with different features, following a feature selection procedure.

[1] The use of labels in preprocessing shouldn't be considered data leakage because deinterleaving occurs before classification, grouping pulses of the same emitter to produce tracks.

6. Evaluate and compare: The evaluation and comparison of models' performance in all datasets must score time and accuracy to value the first two requirements – be quick and highly precise. The third requirement, the models' robustness, is evaluated through the performance of noisy data.

3.2.1. Dataset and resources

The dataset (PDW) was produced with a virtual EW simulator from Leonardo DRS. This software can simulate different radar characteristics, including antenna patterns and external interference, resembling a real-world scenario. In total, 52 different emitters were simulated, from simple navigation radars to multifunction radars, including fixed frequency and RF agility, a wide variety of PW, several PRI modulations, and different types of antenna polarization.

The algorithms used to develop the artifact were implemented in Python, and the experiments were performed using a computer with an Intel® Core™ i5-8265U processor and 16GB of memory.

4. EMPIRICAL STUDY

This chapter describes the design and development of the proposed artifact, following the procedures described in 3.2.

4.1. DATASET PREPARATION

The dataset includes the PDW of 52 emitters generated separately using the virtual EW simulator from Leonardo DRS. To use this data in supervised learning, all PDWs were concatenated and labeled with an alphanumeric label[2] (one letter for each type of radar and a sequential number). Within this preparation, a few changes to features' names and measurement units were performed, ending as TOA(ns), DOA(deg), RF(kHz), PW(ns), POWER(dB), AMOP, FMOP, PMOP, POP, POPed, CW.

Two noisy datasets were produced using 20% of the pulses from each emitter to evaluate the models' robustness:

1. Gaussian noise on continuous features:
   Revillon (2019) have noised continuous features applying a multivariate Gaussian noise with a diagonal covariance matrix comprising RF, PW, PRI = [1MHz, 50ns, 1µs]. Using identical approach the following matrix was taken into consideration – TOA(ns), RF(Khz), PW(ns), POWER(dB) = [1000, 1000, 50, 0.1]. TOA was used instead of PRI because this last is not available in the original dataset and TOA will be used to determine PRI. Furthermore, to affect all available continuous features, power was also noised.
2. Uniform noise on categorical features:
   Y.-H. Chen et al. (2021) used uniform random noise on 25% of the synthetic test data in order to evaluate the hiDF model (presented in the Literature review). Similarly, all flag features (AMOP, FMOP, PMOP, POP, POPed and CW) were uniform random noised.

4.2. DATA EXPLORATION AND DATA CLEANING

4.2.1. Data exploration

6.740.247 records/pulses in the PDW dataset are available without missing data. Checking the descriptive statistics, as Figure 4.1 shows, no incoherent values exist in any feature. On the other hand, TOA(ns), DOA(deg), POP, and POPed are environment-related features and should not be used to build the models, consequently, can be dropped. Thou, as PRI is determined with the difference in TOA between sequential pulses, this feature can only be dropped in the Feature engineering phase.

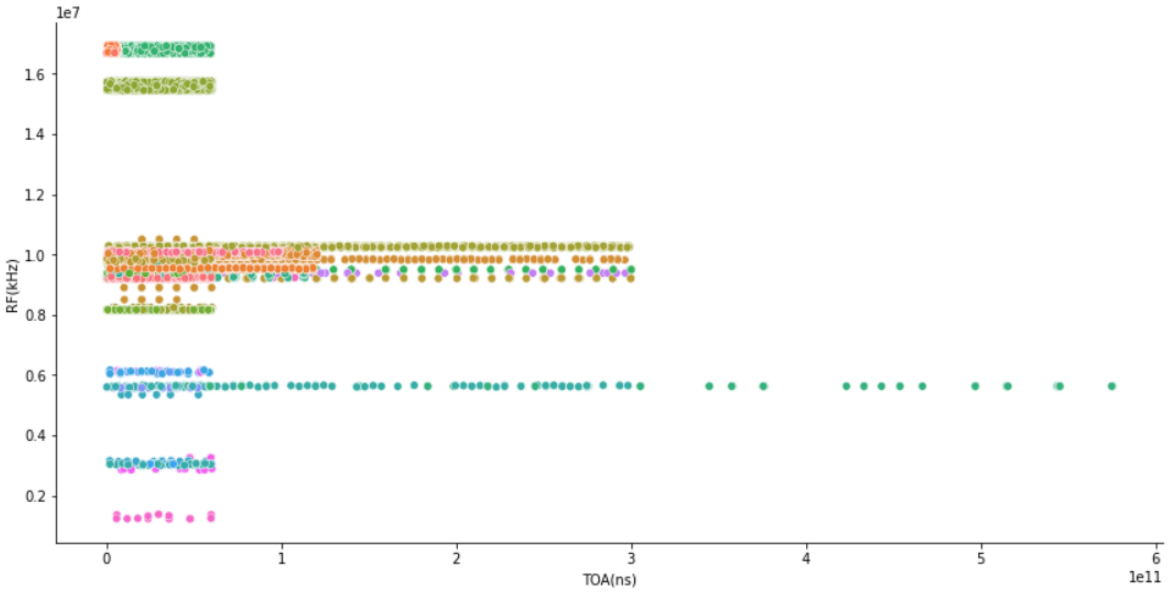
Spearman's correlation was calculated to test for feature correlation and revealed a high negative correlation between TOA(ns) and DOA(deg), which makes no logic because the time when a certain pulse arrives at an ESM system has no relation with the direction from where it arrives; besides, both features will be dropped and will not impact the models.

[2] A second label feature must also be created because XGBoost, since version 1.3.2, requires it to start from 0.

	count	mean	std	min	25%	50%	75%	max
TOA(ns)	6740247.0	3.711060e+10	4.254516e+10	48000000.0	4.466767e+09	2.518418e+10	5.633845e+10	5.753504e+11
DOA(deg)	6740247.0	7.546658e+01	3.862399e-01	68.8	7.530000e+01	7.560000e+01	7.570000e+01	7.580000e+01
RF(kHz)	6740247.0	1.209213e+07	3.366365e+06	1214000.0	9.550000e+06	1.006000e+07	1.669600e+07	1.694300e+07
PW(ns)	6740247.0	2.233413e+03	6.607071e+03	50.0	5.000000e+02	7.000000e+02	1.600000e+03	1.000000e+05
POWER(dB)	6740247.0	2.146573e+01	8.571501e+00	3.0	1.580000e+01	2.200000e+01	2.630000e+01	6.050000e+01
AMOP	6740247.0	0.000000e+00	0.000000e+00	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
FMOP	6740247.0	2.461794e-02	1.549577e-01	0.0	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
PMOP	6740247.0	3.543891e-02	1.848864e-01	0.0	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
POP	6740247.0	0.000000e+00	0.000000e+00	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
POPed	6740247.0	0.000000e+00	0.000000e+00	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
CW	6740247.0	0.000000e+00	0.000000e+00	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00

Figure 4.1 - PDW dataset descriptive statistics

To enhance the familiarization with the PDW dataset[3], Figure 4.2 displays the three main continuous features overtime, colored by the emitter.



[3] More figures are included in Appendix A.1 and A.2.

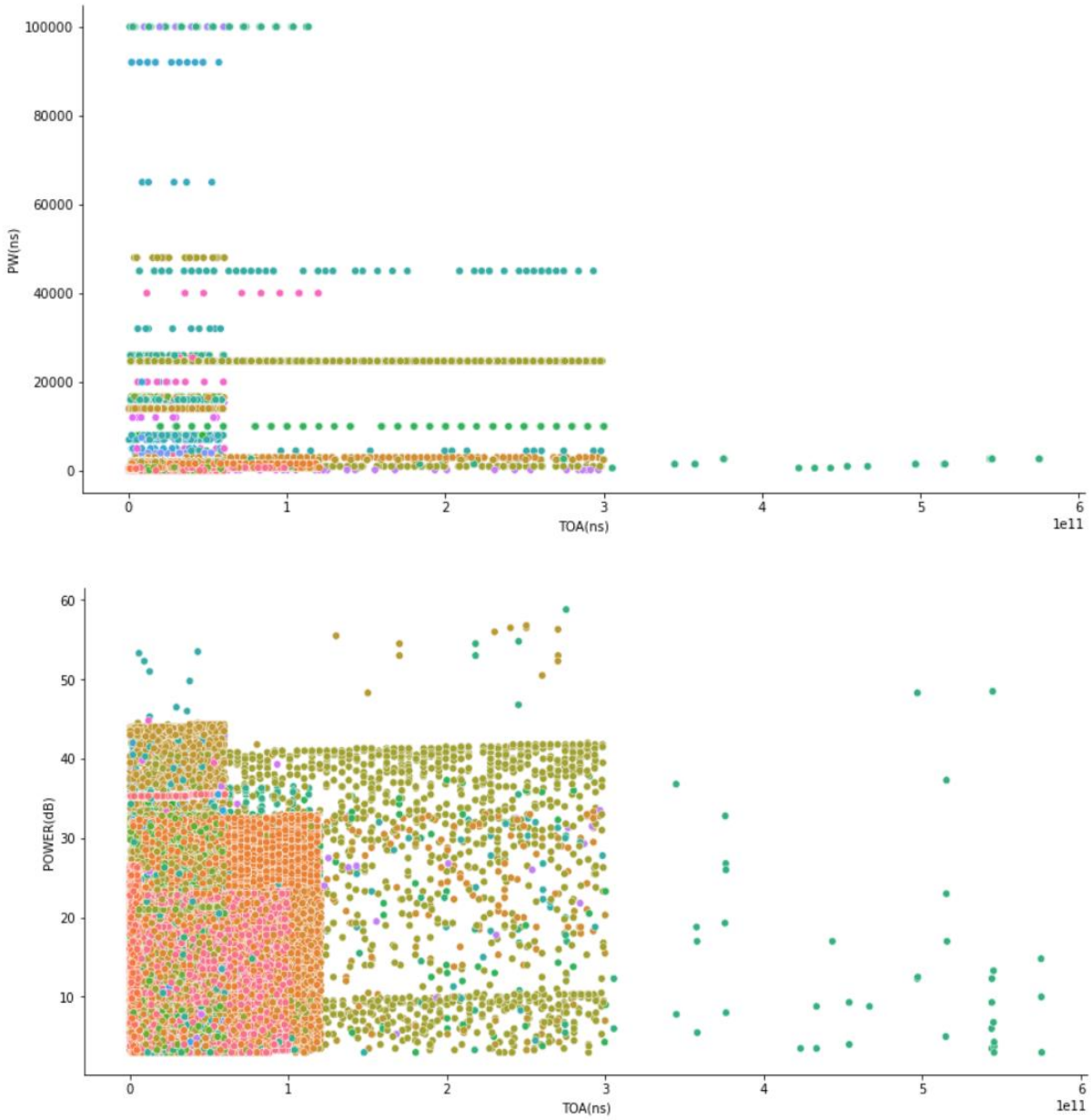


Figure 4.2 - Continuous features evolution overtime

4.2.2. Data cleaning

Over time, a radar transmits pulses with the same characteristics, even when they have the agility and change their parameters for a certain period. So, it is expected to have a substantial number of duplicate records (6.737.766). Nevertheless, as some new features will be based on rolling windows and sequential pulses (reducing that number), duplicate values can only be dropped in the Feature engineering phase before transformations are performed.

From Figure 4.2 and Figure 4.3, it is difficult to identify which records could be considered outliers. On the other hand, since the base model (deep forest) is designed to cope with raw data, there is no place for outliers' removal. Additionally, the exclusion of records may altogether remove emitters from the dataset.

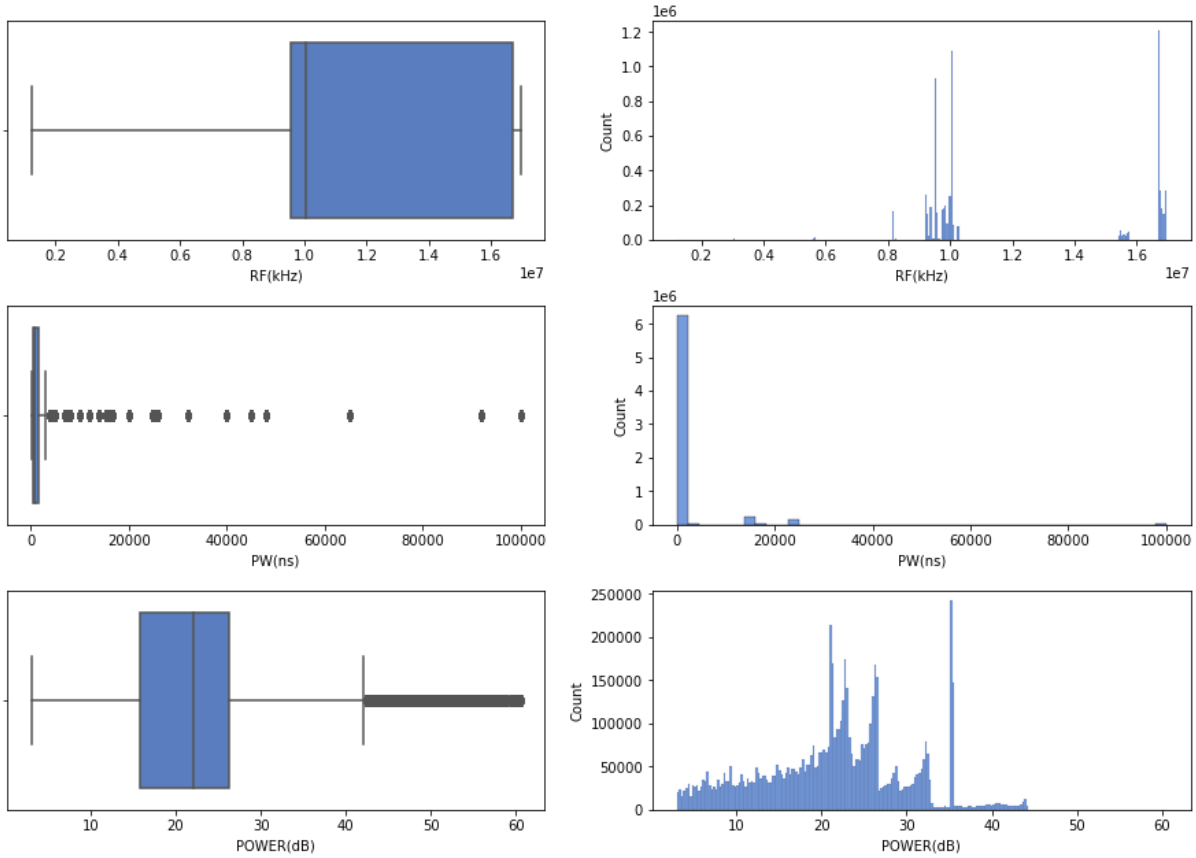


Figure 4.3 - Continuous features distribution

The train-test split of the PDW dataset follows what previous research on deep forest and super learner have implemented (80-20% split), as in Z.-H. Zhou & Feng (2017), Naimi & Balzer (2018), Y. Wang et al. (2019) and Taghizadeh-Mehrjardi et al. (2021).

4.3. FEATURE ENGINEERING

The Feature engineering phase mainly implements what earlier research has proposed and tested regarding new features and transformations. The first new feature is PRI, using the first difference of TOA(ns). This is the primary feature to determine the emitters' modulation in time. Following Ahmed et al. (2018), a second difference of TOA(ns) is also implemented.

In the research of Petrov et al. (2013), the dataset has pulse trains, and this is a summary of the emitter characteristics, which includes the minimum and maximum values of RF, PW and PRI. On the other hand, W. Chen et al. (2017) also have the mean value of the same features. Based on this earlier research, the minimum, maximum, and mean values of RF(kHz), PW(ns) and PRI of all recorded pulses per emitter were determined and included in the dataset. Considering these new features and one of the transformations presented below, the sequence normalization of Notaro et al. (2019), rolling minimum, maximum, and mean values of RF(kHz), PW(ns) and PRI were also included, using a window of 20 pulses.

The next new set of features apply binning to RF, PW and PRI, resembling Sun et al. (2018). This last have implemented a (binary) sequence encoding method dividing the range of each feature in bins. If a bin range is within the minimum and maximum value of a specific feature, this bin will have the value

1 and 0 otherwise. The authors have used RF, PW, PRI = [700, 800, 600] bins and get 2100 features per observation: {RF1, …, RF700, PRI1, …, PRI600, PW1, …, PW800}. Instead of implementing such a vast number of features, each RF, PW and PRI feature already available (original record, overall minimum, maximum and mean, as well as rolling versions) were characterized by a bin. While Sun et al. (2018) have detailed the range applicable to RF, from 200 to 10000 MHz, PW and PRI have not. Consequently, it is possible to replicate the RF bin width in the considered RF range (500 to 18000 MHz), obtaining 1250 bins, but not for the remaining features. PW was defined to use 1501 bins, considering a range from 0 to 75000ns and a bin from this last to infinite (due to CW emitters). Moreover, PRI uses 1000 bins, considering a range from 0 to 3996000ns and a bin from this last to infinite.

Although these new features were based on several individual research, they were combined in the same dataset and will go through feature selection later.

The dataset currently has the following features, excluding labels:

Table 4.1 - List of features

Feature name:	Short description:
RF(kHz)	Frequency of the pulse, in kilohertz
PW(ns)	Duration of the pulse, in nanoseconds
POWER(DB)	Power amplitude of the transmission, in decibel
AMOP	Amplitude modulation detection flag
FMOP	Frequency modulation detection flag
PMOP	Phase modulation detection flag
CW	Continuous wave emitter detection flag
PRI_1	Pulse repetition interval (time lapse between sequential pulses)
PRI_2	Second difference of TOA (the difference of PRI)
RF_min	Minimum RF(khz) value of the emitter
RF_max	Maximum RF(khz) value of the emitter
RF_mean	Average RF(khz) value of the emitter
PRI_min	Minimum PRI_1 value of the emitter
PRI_max	Maximum PRI_1 value of the emitter
PRI_mean	Average PRI_1 value of the emitter
PW_min	Minimum PW(ns) value of the emitter
PW_max	Maximum PW(ns) value of the emitter
PW_mean	Average PW(ns) value of the emitter
RF_rolmin	Rolling minimum RF(khz) value of the emitter
RF_rolmax	Rolling maximum RF(khz) value of the emitter
RF_rolmean	Rolling average RF(khz) value of the emitter
PRI_rolmin	Rolling minimum PRI_1 value of the emitter
PRI_rolmax	Rolling maximum PRI_1 value of the emitter
PRI_rolmean	Rolling average PRI_1 value of the emitter
PW_rolmin	Rolling minimum PW(ns) value of the emitter
PW_rolmax	Rolling maximum PW(ns) value of the emitter
PW_rolmean	Rolling average PW(ns) value of the emitter

RF_bin	Binning of the RF(khz) value
RF_min_bin	Binning of the RF_min value
RF_max_bin	Binning of the RF_max value
RF_mean_bin	Binning of the RF_mean value
RF_rolmin_bin	Binning of the RF_rolmin value
RF_rolmax_bin	Binning of the RF_rolmax value
RF_rolmean_bin	Binning of the RF_rolmean value
PW_bin	Binning of the PW(ns) value
PW_min_bin	Binning of the PW_min value
PW_max_bin	Binning of the PW_max value
PW_mean_bin	Binning of the PW_mean value
PW_rolmin_bin	Binning of the PW_rolmin value
PW_rolmax_bin	Binning of the PW_rolmax value
PW_rolmean_bin	Binning of the PW_rolmean value
PRI_bin	Binning of the PRI_1 value
PRI_min_bin	Binning of the PRI_min value
PRI_max_bin	Binning of the PRI_max value
PRI_mean_bin	Binning of the PRI_mean value
PRI_rolmin_bin	Binning of the PRI_rolmin value
PRI_rolmax_bin	Binning of the PRI_rolmax value
PRI_rolmean_bin	Binning of the PRI_rolmean value

Now, having the PDW dataset enhanced with new features, new datasets can be generated by applying different transformations.

W. Chen et al. (2017) preprocess data by scaling continuous features with Min-Max scaling, ranging those features on [0,1]. Likewise, Petrov et al. (2013) have experimented with three different scaling forms: without scaling, Min-Max scaling, and standardization scaling. These three forms, which create three datasets, are also used in this dissertation.

Another couple of transformations are based on the research of Notaro et al. (2019): normalization scaling with range [-1,1] and the sequence/rolling normalization by the emitter, also with range [-1,1], using a window of 20 pulses. Similarly to Petrov et al. (2013), these authors have experimented with three different forms: without scaling, normalization scaling [-1,1], and concatenating this last with sequence normalization. Following this approach, two new datasets are generated.

The last transformation is based on the research of Gençol (2015) and Y. Zhou et al. (2020), which uses PCA to reduce the dimensionality of Self Organizing Maps (SOM) and images, respectively. Five components accounted for 80% of the variance in the present context were created.

Figure 4.4 depicts a high-level workflow used to gather all (9) datasets and subsequent steps to facilitate understanding and provide status.

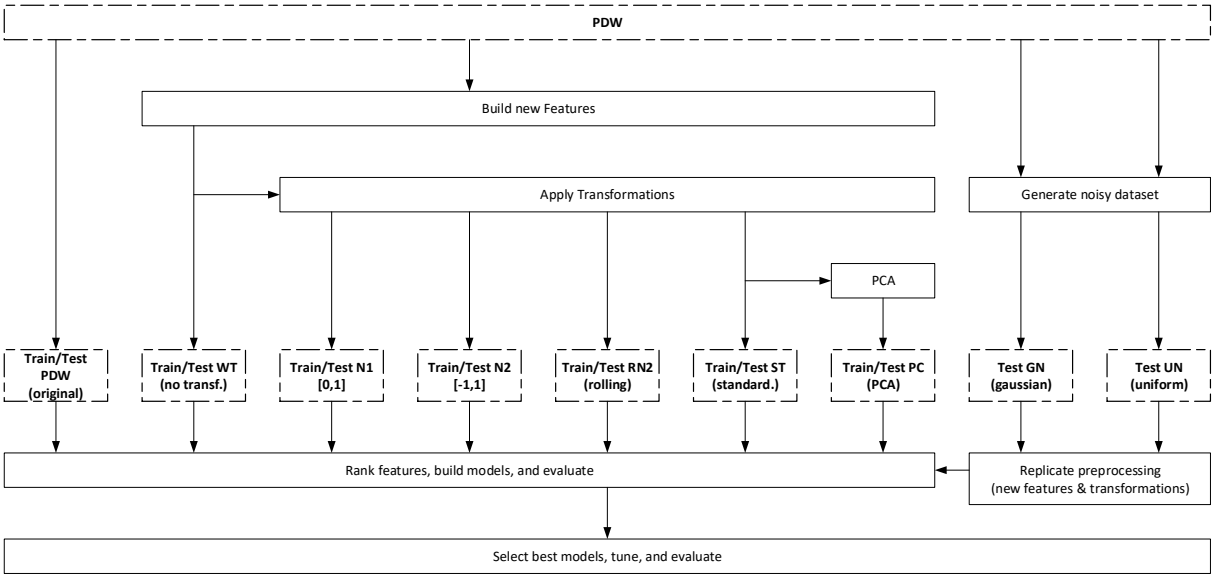


Figure 4.4 - Datasets produced

Feature selection was performed using the wrapper method with recursive feature elimination (RFE) and the filter method with mutual information. The goal was to avoid a single perspective over feature importance and take a more robust approach. Then, cross-referencing the top common features of both methods, the following set of features was obtained: RF_mean_bin, PRI_max, RF_mean, RF_min_bin, PRI_mean, RF_max_bin, RF_rolmin, PRI_min_bin, RF_max, PRI_rolmin, RF_rolmean, PW_mean, RF_min, PW_max, RF_rolmax, PRI_min, RF_rolmax_bin, RF_rolmin_bin. These features are considered in Model optimization.

4.4. Model build and optimization

With datasets already created, it is time to build the models. As previously stated, these models use the deep forest as the foundation and follow some of the research mentioned in the Literature review:

1. Deep forest cascade with default parameters, following Z.-H. Zhou & Feng (2017) implements four random forests and four ExTrees on each layer. This model will be designated as *DF* from now on.
2. Deep forest cascade with two XGBoost, two random forests, two ExTrees, and two GBDT on each layer, following Y. Wang et al. (2019). This model will be designated as *DF-Stack* from now on.
3. Deep forest cascade with two SVM, based on Kim et al. (2015). This model will be designated as *DF-SVM* from now on.
4. Deep forest cascade with two super learners, each with one logistic regression, one KNN, one random forest, one ExTrees, and one XGBoost, following Young et al. (2018), and a linear model as meta-learner (cross-validated risk minimizer), based on Laan et al. (2007). This model will be designated as *DF-SL* from now on.

To improve the comparison of results, deep models' base learners are also built separately: random forest, ExTrees, XGBoost, GBDT, SVM and super learner.

All (10) models are trained and tested with each dataset (Train/Test PDW, Train/Test WT, Train/Test ST, Train/Test N1, Train/Test N2, Train/Test RN2, Train/Test PC) and then evaluated on noisy datasets (Test GN, Test UN). The performance of the models across all datasets is presented in Table 4.3, Table 4.4 and Table 4.5.

Since the shape of the dataset varies with the feature engineering step it comprises, Table 4.2 states the number of records and features present in each test dataset.

Table 4.2 - Datasets shape

Train Dataset	**Train PDW**	**Train WT**	**Train ST**	**Train N1**	**Train N2**	**Train RN2**	**Train PC**
Test Dataset	**Test PDW**	**Test WT**	**Test ST**	**Test N1**	**Test N2**	**Test RN2**	**Test PC**
Records	4933	33830	33830	33830	33830	33830	33830
Features	7	48	48	48	48	96	5
Test Dataset	**Test GN-PDW**	**Test GN-WT**	**Test GN-ST**	**Test GN-N1**	**Test GN-N2**	**Test GN-RN2**	**Test GN-PC**
Records	1348052	1347948	1347948	1347948	1347948	1347948	1347948
Features	7	48	48	48	48	96	5
Test Dataset	**Test UN-PDW**	**Test UN-WT**	**Test UN-ST**	**Test UN-N1**	**Test UN-N2**	**Test UN-RN2**	**Test UN-PC**
Records	143249	294253	294253	294253	294253	294253	294253
Features	7	48	48	48	48	96	5

4.4.1. Model optimization

As the best models already achieve 100% accuracy in primary and noisy datasets, improving them through feature selection and hyperparameter optimization is impossible. Tough, two developments can be made:

1. Reduce the number of features the deep model uses to achieve 100% accuracy. In this sense, and to increase efficiency, only datasets without transformation should be used.
2. Improve the performance of a deep model using raw data (dataset without feature engineering).

The first line of development makes use of DF, whereas the second makes use of DF-SL. This decision is based on the models' performance for the specific development context (accuracy and prediction time with or without new features, respectively). As previously, deep models' base learners are also built in separately to promote the comparison of results.

DF optimization:

Recalling the feature selection previously performed; it was confirmed that DF only requires the first four features (RF_mean_bin, PRI_max, RF_mean, RF_min_bin) to maintain 100% accuracy in Test WT, Test GN and Test UN dataset.

DF-SL optimization:

To start, as the dataset for this optimization contains a different set of features, a new rank of features is obtained: RF(kHz), PW(ns), POWER(dB), FMOP, PMOP, CW, and AMOP. The performance of DF-SL and DF-SL base learners are tested over different sets of features in all three datasets and reported in Table 4.7 and Table 4.8. Falling the choice only to use the three continuous features.

The hyperparameter optimization procedure is based on Zhang et al. (2020), for deep forest cascade optimization, and Ai et al. (2020) for the super learner, comprising the following steps:

1. Test and select the best parameters for the base learner (super learner).
    a. Test and select the best number of v-fold cross-validation [2,10].
    b. Test and select the best loss function (cross-validated risk minimizer).
2. Test and select the best number of base learners [2,4].

All accuracies reached in these tests are registered in Table 4.9 and Table 4.10, allowing to set of an optimized version of DF-SL which implements three super learners, with 7-fold cross-validation and stochastic gradient descent (SGD) loss function, on each cascade layer.

The performance of the optimized DF-SL and its base learner across the three datasets is recorded in Table 4.11. In order to further detail the accuracy here achieved, the confusion matrix of both models, in all datasets, are available in Appendix A.3 and A.4.

4.5. EVALUATE AND COMPARE

The models' performance must be evaluated to compare the models and determine whether the problem can be solved. In this context, and following the latest deep forest research, the accuracy, in percentage, and the test prediction time, in seconds, are both considered (Pang et al., 2018, 2022).

The models performed for all train/test datasets as shown in Table 4.3. Then, these models were evaluated using both noisy datasets, performing as shown in Table 4.4 and Table 4.5.

Table 4.3 - Models' performance on train/test dataset

Train Dataset	**Train PDW**	**Train WT**	**Train ST**	**Train N1**	**Train N2**	**Train RN2**	**Train PC**
Test Dataset	**Test PDW**	**Test WT**	**Test ST**	**Test N1**	**Test N2**	**Test RN2**	**Test PC**
Model:	**Accuracy:**						
DF	98.358%	100%	100%	100%	100%	100%	99.985%
DF-Stack	97.203%	100%	100%	100%	100%	100%	99.988%
DF-SVM	86.621%	99.693%	99.693%	99.693%	97.351%	95.516%	98.696%
DF-SL	96.757%	100%	100%	100%	100%	100%	99.988%
RandForest	94.567%	100%	100%	100%	100%	100%	99.991%
ExTrees	94.263%	100%	100%	100%	100%	100%	99.994%
XGBoost	94.952%	100%	100%	100%	100%	100%	99.982%
GBDT	92.601%	100%	100%	100%	100%	100%	95.998%
SVM	45.713%	70.183%	99.985%	98.424%	98.424%	97.824%	95.936%
SuperLearner	97.567%	100%	100%	100%	1000%	100%	99.994%
Model:	**Prediction time (seconds):**						
DF	4.700	10.400	13.300	19.600	16.900	13.900	28.600
DF-Stack	17.500	12.200	17.200	12.100	14.500	27.600	12.100

DF-SVM	35.700	315.000	313.000	310.000	313.000	1280.000	250.000
DF-SL	4.060	159.000	152.000	160.000	163.000	216.000	8.870
RandForest	0.318	1.090	1.050	1.070	1.080	1.580	1.100
ExTrees	0.336	1.090	1.080	1.090	1.060	1.660	1.200
XGBoost	0.042	0.208	0.200	0.160	0.167	0.216	0.248
GBDT	0.149	5.500	3.580	3.370	3.370	201.000	0.630
SVM	21.000	559.000	121.000	178.000	179.000	253.000	162.000
SuperLearner	0.843	10.600	10.900	10.600	10.800	16.300	2.050

Table 4.4 - Models' performance on Gaussian noise dataset

Train Dataset	**Train PDW**	**Train WT**	**Train ST**	**Train N1**	**Train N2**	**Train RN2**	**Train PC**
Test Dataset	**Test GN-PDW**	**Test GN-WT**	**Test GN-ST**	**Test GN-N1**	**Test GN-N2**	**Test GN-RN2**	**Test GN-PC**
Model:	**Accuracy:**						
DF	90.277%	100%	100%	100%	100%	100%	99.995%
DF-Stack	89.673%	99.989%	99.989%	99.989%	99.987%	97.071%	99.998%
DF-SVM	79.075%	97.017%	97.017%	97.017%	98.058%	89.236%	99.748%
DF-SL	91.621%	100%	100%	100%	100%	99.998%	99.996%
RandForest	94.432%	100%	100%	100%	100%	100%	99.892%
ExTrees	94.028%	100%	100%	100%	100%	100%	99.999%
XGBoost	92.744%	98.413%	98.413%	98.413%	78.136%	75.756%	99.191%
GBDT	87.049%	99.565%	99.565%	99.565%	80.368%	77.282%	98.107%
SVM	39.637%	17.143%	99.997%	84.822%	84.827%	69.464%	97.301%
SuperLearner	94.433%	100%	100%	100%	100%	100%	99.997%
Model:	**Prediction time (seconds):**						
DF	1387.000	525.000	360.000	544.000	406.000	749.000	1187.000
DF-Stack	3120.000	645.000	470.000	692.000	487.000	981.000	596.000
DF-SVM	10050.000	13799.00	12488.00	13080.00	12643.00	19424.00	11465.00
DF-SL	733.000	8108.000	6073.000	6439.000	6437.000	6623.000	270.000
RandForest	99.000	38.800	36.800	36.700	40.000	51.800	39.200
ExTrees	702.000	37.700	37.800	37.200	40.100	53.500	40.700
XGBoost	8.960	4.970	5.310	4.850	4.810	10.300	6.680
GBDT	22.100	129.000	123.000	114.000	120.000	127.000	21.400
SVM	5553.000	22602.00	4727.000	7072.000	7137.000	7922.000	7400.000
SuperLearner	85.000	415.000	418.000	405.000	407.000	617.000	65.000

Table 4.5 - Models' performance on Uniform noise dataset

Train Dataset	**Train PDW**	**Train WT**	**Train ST**	**Train N1**	**Train N2**	**Train RN2**	**Train PC**
Test Dataset	**Test UN-PDW**	**Test UN-WT**	**Test UN-ST**	**Test UN-N1**	**Test UN-N2**	**Test UN-RN2**	**Test UN-PC**
Model:	**Accuracy:**						
DF	73.843%	100%	100%	100%	100%	100%	62.818%
DF-Stack	77.397%	100%	100%	100%	86.364%	91.280%	61.729%
DF-SVM	87.915%	99.909%	99.909%	99.909%	99.900%	98.487%	61.733%

DF-SL	89.031%	100%	100%	100%	100%	100%	64.374%
RandForest	69.829%	100%	100%	100%	100%	100%	56.661%
ExTrees	41.954%	100%	100%	100%	100%	100%	63.178%
XGBoost	86.635%	74.234%	74.234%	74.393%	60.415%	59.297%	63.584%
GBDT	49.109%	99.049%	99.049%	99.049%	74.567%	74.710%	45.903%
SVM	48.095%	39.587%	92.386%	39.161%	39.577%	56.889%	65.876%
SuperLearner	93.066%	100%	99.500%	83.126%	82.551%	91.239%	72.918%
Model:	**Prediction time (seconds):**						
DF	134.000	123.000	72.000	83.000	86.000	102.000	197.000
DF-Stack	332.000	146.000	101.000	101.000	106.000	147.000	104.000
DF-SVM	1053.000	5005.000	3743.000	2711.000	3907.000	6474.000	2209.000
DF-SL	39.200	2848.000	1830000	1399.000	1935.000	1942.000	63.000
RandForest	6.450	18.600	11.400	7.950	12.200	11.200	8.460
ExTrees	6.640	18.800	12.300	8.050	12.400	11.800	8.890
XGBoost	1.080	1.190	1.400	1.070	1.380	1.410	1.360
GBDT	2.160	34.400	30.500	26.100	36.600	29.200	4.940
SVM	905.000	7298.000	1534.000	1546.000	1979.000	2244.000	1354.000
SuperLearner	15.300	86.000	87.000	88.000	91.000	128.000	17.800

The following considerations can be taken by analyzing and comparing the models' performance:

- Models' accuracy increased with new features (created in feature engineering).
- In general, the models had better accuracy with the normalization scaling [-1,1] dataset (N2) than the concatenation of normalization scaling [-1,1] with the sequence normalization (RN2).
- Deep models, random forest, ExTrees, XGBoost and GBDT had the same accuracy whether the dataset had no transformation (WT), Min-Max scaling (N1) or standardization scaling (ST).
- DF-SVM improved the accuracy of its base learner (SVM).
- DF-SL improved the accuracy of its base learner (super learner) when using datasets with transformation and underperformed when using original data (PDW) and principal components (PC).
- While DF-SL had the best overall accuracy, DF had the second-best accuracy and the best time performance within the deep forest models (almost eight times faster than DF-SL).

Since DF only employs random forests and ExTrees as base learners, its performance may be justified by the same top performance of its base learners.

4.5.1. Evaluate optimization

The optimization of DF is obtained through feature selection, which requires evaluating its performance over a reduced sequence of features previously ranked. As presented in Table 4.6, DF maintained 100% accuracy in all test datasets using four features.

Table 4.6 - DF and base learners' performance with four features

Train Dataset	**Train WT**	**Train WT**	**Train WT**
Test Dataset	**Test WT**	**Test GN-WT**	**Test UN-WT**
Features	RF_mean_bin, PRI_max, RF_mean, RF_min_bin		
Model:	**Accuracy:**		
DF	100%	100%	100%
RandForest	100%	100%	100%
ExTrees	100%	100%	100%
Model:	**Prediction time (seconds):**		
DF	21.900	746.000	165.000
RandForest	2.530	86.000	19.300
ExTrees	2.430	85.000	19.100

Regarding prediction time, DF increased 42% while random forest and ExTrees increased 84%, yet DF is still eight times slower.

The DF-SL optimization includes feature selection and hyperparameter optimization. The first optimization is executed as in DF, which records the performance over different sets of features, as follows.

Table 4.7 - DF-SL and super learner performance with five features

Train Dataset	**Train PDW**	**Train PDW**	**Train PDW**
Test Dataset	**Test PDW**	**Test GN-PDW**	**Test UN-PDW**
Features	RF(kHz), PW(ns), POWER(dB), FMOP, PMOP		
Model:	**Accuracy:**		
DF-SL	96.777%	91.618%	90.123%
SuperLearner	97.588%	94.423%	93.392%
Model:	**Prediction time (seconds):**		
DF-SL	2.860	351.000	38.200
SuperLearner	0.806	88.000	9.160

Table 4.8 - DF-SL and super learner performance with three features

Train Dataset	**Train PDW**	**Train PDW**	**Train PDW**
Test Dataset	**Test PDW**	**Test GN-PDW**	**Test UN-PDW**
Features	RF(kHz), PW(ns), POWER(dB)		
Model:	**Accuracy:**		
DF-SL	96.817%	91.610%	96.179%
SuperLearner	97.567%	94.357%	96.112%
Model:	**Prediction time (seconds):**		
DF-SL	3.180	284.000	38.100
SuperLearner	1.080	67.000	7.750

Comparing both tables, using three features improves the accuracy of Test UN[4] in both models. Furthermore, the overall prediction time is faster. Consequently, this set of features should be selected and continued for hyperparameter optimization.

The first step of hyperparameter optimization requires testing and selecting the best parameters for the base learner. Table 4.9 comprises the accuracy of the super learner (DF-SL base learner) over a different number of v-fold cross-validation and loss functions.

Table 4.9 - Super Learner accuracy over fold and loss function

Loss Function	**Logistic Regression**	**Ridge Classifier**	**SGD Classifier**
Folds:	**Train/Test PDW accuracy:**	**Train/Test PDW accuracy:**	**Train/Test PDW accuracy:**
2-folds	96.751% / 97.567%	96.766% / 97.790%	96.563% / 97.810%
3-folds	96.563% / 98.135%	96.619% / 97.993%	96.345% / 98.317%
4-folds	96.436% / 98.297%	96.512% / 98.135%	96.543% / 98.317%
5-folds	96.401% / 98.378%	96.467% / 98.256%	96.264% / 98.439%
6-folds	96.447% / 98.276%	96.502% / 98.216%	96.507% / 98.317%
7-folds	96.340% / 98.398%	96.447% / 98.317%	96.071% / 98.439%
8-folds	96.365% / 98.357%	96.472% / 98.216%	96.431% / 98.357%
9-folds	96.355% / 98.378%	96.457% / 98.256%	96.021% / 98.337%
10-folds	96.381% / 98.378%	96.477% / 98.236%	96.330% / 98.357%

The second step requires testing and selecting the best number of (optimized) base learners – 7-fold cross-validation super learner with SGD loss function; to be used in each layer of the deep forest cascade. The accuracy achieved with 2, 3 and 4 base learners is recorded in Table 4.10.

Table 4.10 - DF-SL accuracy over base learner quantity

Models qt:	**Train/Test PDW accuracy:**
2 SL	96.928% / 97.121%
3 SL	96.923% / 97.182%
4 SL	96.913% / 97.161%

After finding the best number of base learners (3) it is possible to re-evaluate the models over the three datasets and compare it with the initial performance, as follows.

Table 4.11 - Optimized models performance

Train Dataset	**Train PDW**	**Train PDW**	**Train PDW**
Test Dataset	**Test PDW**	**Test GN-PDW**	**Test UN-PDW**
Model:	**Accuracy:**		
DF-SL	97.182%	90.310%	95.999%
SuperLearner	98.358%	94.532%	96.073%
Model:	**Prediction time (seconds):**		

[4] As the models aren't using categorical features, the noise introduced in Test UN doesn't affect the models' accuracy.

DF-SL	4.890	786.000	69.000
SuperLearner	0.906	135.000	15.400

The optimized version of DF-SL improved by 2% accuracy on average while the optimized version of super learner improved by 1.29%; however, this last still achieved higher accuracy in all three datasets. DF-SL also worsen by 1.31% accuracy in Test GN, revealing less generalization capability. Regarding prediction time, DF-SL increased by 11%, and super learner increased by 50%, yet DF-SL is still five times slower.

4.6. RESOURCES

Table 4.12 lists the main Python packages and software used to develop the artifact.

Table 4.12 - Summary of software

Software/package:	**Version:**	**Software/package:**	**Version:**
Jupyter Notebook	6.3.0	matplotlib	3.3.4
phyton	3.8.8	seaborn	0.12.0
pandas	1.4.2	deep-forest	0.1.7
numpy	1.23.4	xgboost	1.6.0
scikit-learn	1.1.3	mlens	0.2.3

5. RESULTS AND DISCUSSION

The main objective of this dissertation is to develop a deep ensemble model based on deep forest that can classify radar emitters in PDW. In this sense, four deep ensemble models and six of its base learners were built and tested on several datasets. Figure 5.1 plots the models' performance, measured in accuracy and time, on each test dataset. It is possible to see that XGBoost is the fastest model, DF-SVM is the slowest, and almost every model achieved 100% accuracy (except SVM and DF-SVM). All models underperformed on Test PC and Test PDW (trained on Train PC and Train PDW, respectively), meaning they performed better on datasets with engineered features regardless of the transformation applied.

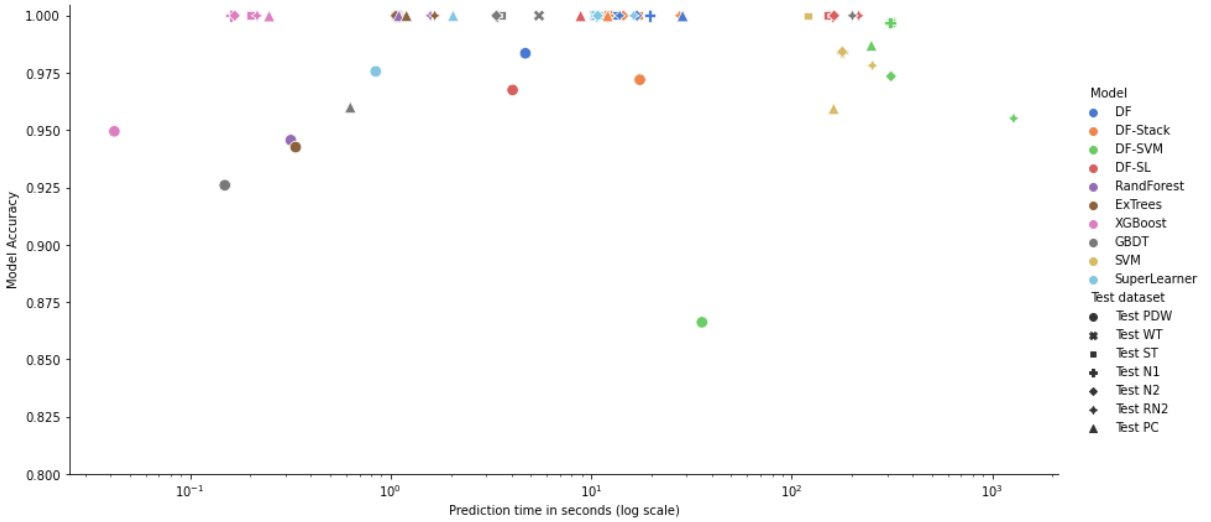


Figure 5.1 - Models' performance on the test dataset

Figure 5.2 plots the models' performance on the Gaussian noise test dataset, where the prediction time increased because this dataset is much bigger. Looking to the right side of the figure, it is possible to see that DF-SVM achieved a higher accuracy than its base learner (SVM).

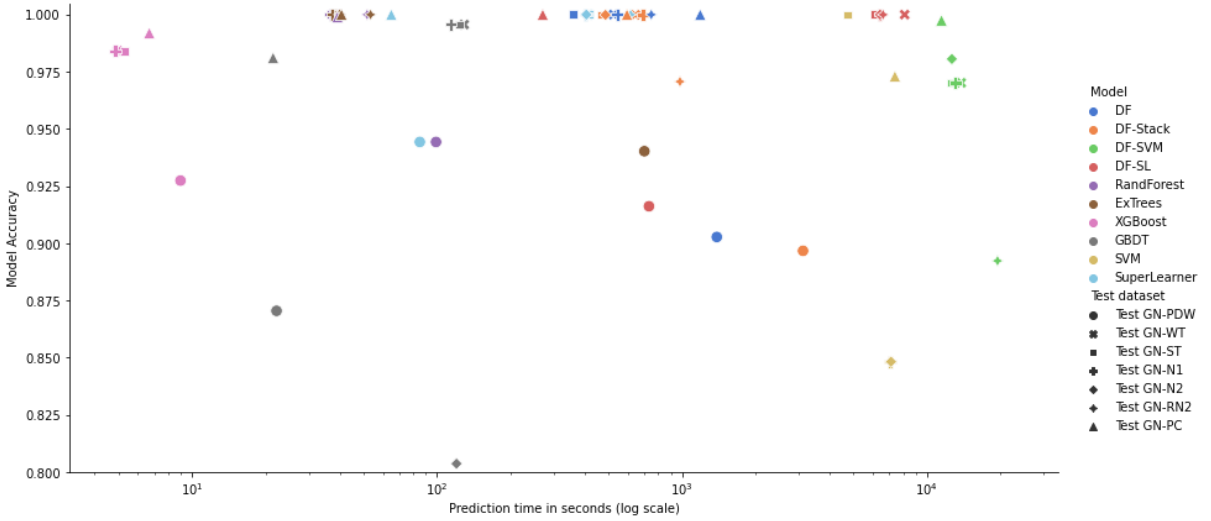


Figure 5.2 - Models' performance on Gaussian noise dataset

Figure 5.3 plots the models' performance[5] on the Uniform noise test dataset, where it is possible to see that DF-SL had lower accuracy than its base learner (super learner) on Test PDW but better on other datasets, as Test N1, Test ST, Test N2 and Test RN2.

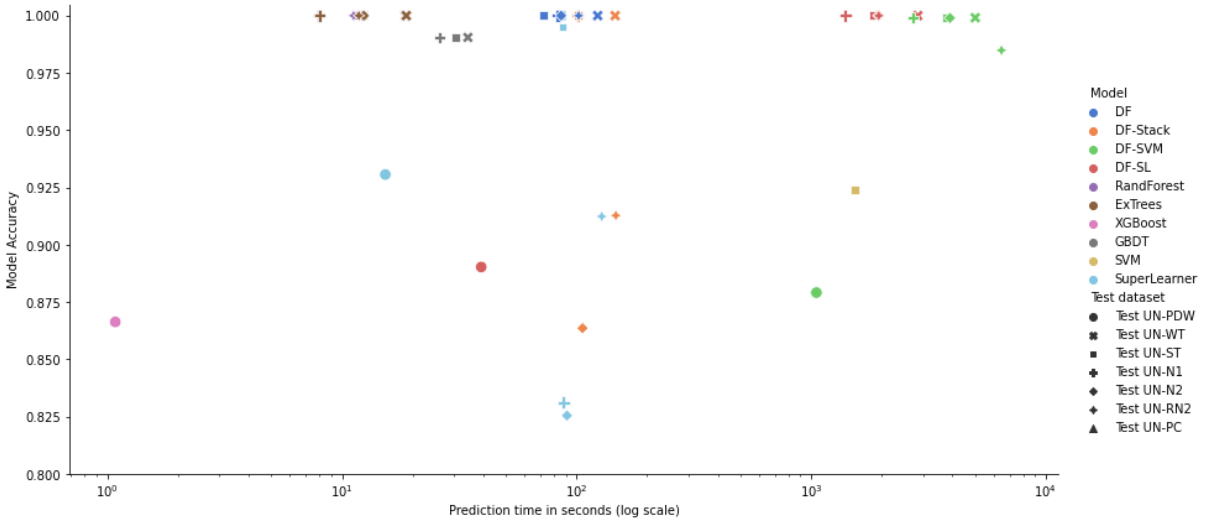


Figure 5.3 - Models' performance on Uniform noise dataset

Given that datasets have a different number of records and models may perform differently on each, Table 5.1 presents the average prediction time of each model per 1000 records, confirming that XGBoost, random forest and ExTrees are significantly faster than the remaining.

Table 5.1 - Average model prediction time

Average prediction time per 1000 records (seconds):				
DF	**DF-Stack**	**DF-SVM**	**DF-SL**	**RandForest**
0.524	0.775	11.785	4.107	0.038
ExTrees	**XGBoost**	**GBDT**	**SVM**	**SuperLearner**
0.061	0.005	0.358	7.334	0.265

Since the best models already achieve 100% accuracy in primary and noisy datasets, improving them through feature selection and hyperparameter optimization is impossible. Tough, it is possible to seek efficiency by reducing the number of features the model uses on datasets without transformation. This DF implementation concluded that using four features (RF_mean_bin, PRI_max, RF_mean, RF_min_bin) was enough to maintain 100% accuracy on Test WT, Test GN and Test UN datasets. However, different from what was expected, the models become substantially slower, especially random forest and ExTrees, with an increase of 84%.

Another interesting approach is to improve a deep model using the PDW dataset, which has not any transformation or feature engineering. This implementation on DF-SL and its base learner increased

[5] Appendix A.5. presents the models' performance on test dataset, Gaussian noise dataset, and Uniform noise dataset with linear time scale and accuracy between 0 and 100%.

the accuracy by 2.027% and 1.299%, respectively. Nevertheless, as presented in Figure 5.4, the super learner still achieves higher accuracy in all three datasets and is five times faster.

For both approaches, it was impossible to determine why prediction time increased. While the number of layers in both deep ensembles has remained constant, the training times for all models have increased.

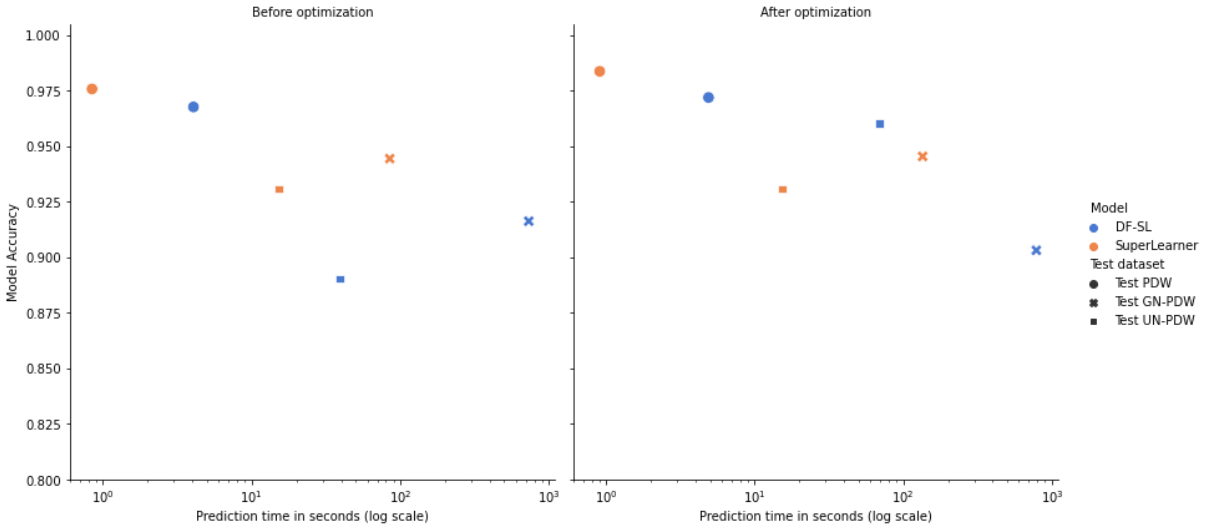


Figure 5.4 - Comparison of models' performance

The accuracy reached in this dissertation appears to be good, but how does it compare to previous research? The answer to this important question is complicated since each research employs its dataset, ranging from a few thousand records to some hundred thousand records, containing different numbers of agile emitters and varying the number of labels from a dozen to a hundred. So, this comparison is only possible with many assumptions.

As expected, the super learner performed asymptotically or better than any of its base learners[6]. The only down performance was observed in classifying the Uniform noise dataset using a model trained with transformations (Train ST, Train N1, Train N2 and Train RN2 dataset). In these occurrences, the deep model (DF-SL) improved the accuracy of the base learner, confirming what Young et al. (2018) concluded – augmented features add depth to the learning and improve accuracy. Likewise, SVM's accuracy increased significantly on the deep model, going from an average of 70.536% to 94.390%, similar to what Kim et al. (2015) and Qi et al. (2016) had verified.

According to Z.-H. Zhou & Feng (2017) deep forest has few hyperparameters and can perform excellently with default settings. In applying this model to classify synthetic aperture radar (SAR) targets, the authors had the same conclusion but also stated that an appropriate selection of hyperparameters would improve the accuracy for a specific classification task (Zhang et al., 2020). Considering the results of this dissertation, it is possible to confirm both statements.

[6] This conclusion comes from Literature review and considers the use of train/test dataset compared to random forest, ExTrees and XGBoost.

In the context of HRRP recognition, as presented in the Literature review, deep forest achieved 94.81% accuracy (Y. Wang et al., 2019). That model was replicated on DF-Stack and, using the PDW dataset, achieved 97.203% accuracy.

Some previous research on radar classification was referred to in Introduction, and the corresponding accuracies are now presented for comparison (see Table 5.2).

Table 5.2 - Accuracy of previous research

Research:	**Model:**	**Highest accuracy:**
Petrov et al., 2013	Feedforward Neural Networks	84.3%
W. Chen et al., 2017	Weighted XGBoost	98.30%
Sun et al., 2018	Unidimensional CNN	97.5%
Notaro et al., 2019	RNN and LSTM	64.98%
Xiao & Yan, 2020	Naïve Bayes	99.50%
Feng et al., 2021	Hybrid Deep Neural Network with dynamic CNN and LSTM	96.24%
J. Wang et al., 2022	Intuitionistic Fuzzy Information Tri-training	99.91%

Regarding prediction time, just a few previous research have measured it, and from those, some comparisons could be made solely for SVM, following Sun et al., 2018 and Xiao & Yan, 2020. Even still, the impact of different hardware and datasets would have to be considered in that comparison.

Some models evaluated are highly competitive in accuracy, but are they appropriate to solve the problem? Considering the use case of accessing data after deinterleave (with cluster labels), it is possible to engineer the four features that enable DF to accomplish 100% accuracy on datasets without transformation. This leads to the conclusion that DF complies with the requirements of precision and robustness. Regarding classification speed, the conclusion is complex – radars can transmit between a few hundred and several hundred thousand pulses per second, and how many are unique to be used in classification is unclear. In this context, on average, DF can classify 1000 records in 557ms (after optimization), which may be acceptable. It is also worth noticing that random forest and ExTrees accomplish the same accuracy and can be eight times faster than DF.

In the use case of working with raw data, the super learner (after optimization) matched the accuracy of DF on Test PDW (98.358%) but can be considered the best performant model due to its superiority on noisy datasets. Though, it cannot be considered to meet the required precision and robustness due to a lack of quantitative requirements. Regarding classification speed, in this context, can classify 1000 records in 101ms, which may be acceptable.

The contributions of this dissertation are summarized as follows:

- For the first time, using a super learner as the deep forest's base learner and using these models separately is proposed to classify emitters in PDW.
- It is verified that transformations do not impact deep forest accuracy.
- A new set of features has been proposed and tested – rolling minimum, maximum, and mean values of RF(kHz), PW(ns) and PRI, have risen to the top of feature selection.

Focusing on the practical implementation of this work, some of the benefits and contributions could be:

- Less dependency on experts to build and tune radar parameters databases, considering future model training with ESM PDW records.
- The robustness to noise is intrinsic to the models' behavior instead of database parameters.
- The four features required to achieve 100% accuracy can be obtained from pulse trains.
- The model's classification process is not human-readable, protecting such classified information.

6. CONCLUSIONS AND FUTURE WORKS

Modern warfare employs information from numerous sources to conduct operations, including ESM systems that intercept, locate, and classify radar emitters to support the battlefield's situational awareness. These systems use the parameter matching method[7] to perform classification, which heavily relies on several experts to update, maintain, tune, and deploy radar parameters databases. Therefore, the present dissertation fosters an alternative – How can a deep ensemble model, based on deep forest, be used to classify radar emitters in PDW of ESM systems? Given the impact of misclassification and the conditions of operation, this model must be quick, highly precise, and robust to noise.

The most suitable methodology was found to be DSRM, complemented by *the six phases of the data science loop* from Ferreira (2020), and the (PDW) dataset was produced with a virtual EW simulator from Leonardo DRS, totaling 52 different emitters. This dataset started with 11 features and after feature engineering, ended with 48. Then, several transformations took place, producing seven datasets to train and test the models, plus two datasets to assess the models' robustness to noise – one is dedicated to Gaussian noise on continuous features and another to Uniform noise on categorical features.

Based on previous research, four deep ensemble models and six of its base learners were built and evaluated on all datasets, counting 420 measurements of accuracy and classification speed (also designated as prediction time). In overall accuracy, DF-SL was the best model (97.227%), and DF was the second-best (96.442%), but regarding speed, DF was almost eight times faster than DF-SL, making it the best performant model. Both models achieved 100% accuracy in datasets with feature engineering, but with raw data, DF-SL achieved 92.470%, and DF achieved 87.493%, on average. Hence, model optimization followed two distinct paths:

1. Improve the efficiency of DF, reducing the number of features used by the model through feature selection while maintaining 100% accuracy in primary and noisy datasets.
2. Improve the performance of DF-SL through feature selection and hyperparameter optimization using raw data.

These developments were compared in deep ensemble models and their base learners. In the first line of development, DF and its base learners were able to maintain the accuracy using four features (RF_mean_bin, PRI_max, RF_mean, RF_min_bin) but became much slower – DF increased prediction time by 42% while base learners increased 84%. Nevertheless, the base learners (random forest and ExTrees) were still eight times faster than DF.

Regarding the second line of development, feature selection leads DF-SL and super learner to use three features (RF(kHz), PW(ns), POWER(dB)), and after hyperparameter optimization, the accuracy has increased 2.027% and 1.299% respectively. However, the super learner achieves higher accuracy in all three datasets and is five times faster than DF-SL.

[7] As the expert system method has a low recognition efficiency and a slow recognition speed, it's not very common.

Further conclusions can be taken from the results:

- Models achieved higher accuracy on datasets with engineered features, regardless of their transformation.
- DF have exceptional performance with default settings.
- DF-SVM improved the accuracy of its base learner (SVM).
- DF-SL improved the accuracy of its base learner (super learner) on datasets with transformation.
- XGBoost, random forest and ExTrees are significantly faster than the remaining models.

Compared with previous research, some of the models evaluated are highly competitive in accuracy – 100% when using datasets with feature engineering and an average of 96.321%[8] using raw data – which may be considered to meet the requirements of precision and robustness. Regarding classification speed, the optimized models can classify 1000 records from 64ms (in the case of random forest and ExTrees) up to 575ms (in the case of DF-SL), which may be acceptable. However, as these models support parallelism, hardware can speed up their prediction time.

Bearing models' performances in mind, the effort required to train them, and recalling the research objective, this can be used as an alternative to the parameter matching method since it is less dependent on experts, easy to maintain, fast to update (train the model on new emitters), and highly accurate.

This dissertation contributed to the knowledge on deep forest and its usage with customized base learners, on classification. It also introduced some novelty by combining deep forest with super learner, and the rolling minimum, maximum, and mean values of RF(kHz), PW(ns) and PRI. Even so, many limitations are also present and should be addressed in future works, such as:

- Perform hyperparameter optimization of the super learner's base learners, including new base learners, and evaluate DF-SL with this newly optimized super learner (as its base learner).
- Evaluate the models on datasets with an incremental margin of error in all records and features (continuous and categorical).
- Evaluate the models on datasets with errors in deinterleaver and cluster label errors.
- Evaluate the models on datasets with more complex scenarios and actual PDW recordings.
- Fix dataset class imbalance with SMOTE, as performed by Ai et al. (2020).
- Grant the proper open-set procedure, such as the classification of unknown emitters, following Apfeld & Charlish (2021).
- Build a super learner that implements deep forest as one of the base learners.

[8] Average accuracy of super learner after optimization, in accordance with Table 4.11.

BIBLIOGRAPHICAL REFERENCES

Abdullah, A., Veltkamp, R. C., & Wiering, M. A. (2009). An Ensemble of Deep Support Vector Machines for Image Categorization. *2009 International Conference of Soft Computing and Pattern Recognition*, 301–306. https://doi.org/10.1109/SoCPaR.2009.67

Ahmed, U. I., Rehman, T. ur, Baqar, S., Hussain, I., & Adnan, M. (2018). Robust pulse repetition interval (PRI) classification scheme under complex multi emitter scenario. *2018 22nd International Microwave and Radar Conference (MIKON)*, 597–600. https://doi.org/10.23919/MIKON.2018.8405297

Ai, Z., Luktarhan, N., Zhou, A., & Lv, D. (2020). WebShell Attack Detection Based on a Deep Super Learner. *Symmetry*, *12*(9), 1406. https://doi.org/10.3390/sym12091406

Aldossary, M. (2017). *De-interleaving of Radar pulses for EW receivers with an ELINT application*. University of Cape Town.

Apfeld, S., & Charlish, A. (2021). Recognition of Unknown Radar Emitters With Machine Learning. *IEEE Transactions on Aerospace and Electronic Systems*, *57*(6), 4433–4447. https://doi.org/10.1109/TAES.2021.3098125

Aslan, M. K. (2006). *Emitter Identification Techniques*. Middle East Technical University.

Baskerville, R., Baiyere, A., Gergor, S., Hevner, A., & Rossi, M. (2018). Design Science Research Contributions: Finding a Balance between Artifact and Theory. *Journal of the Association for Information Systems*, *19*(5), 358–376. https://doi.org/10.17705/1jais.00495

Bildøy, B. E. S. (2006). *Satellite Cluster Consepts, A system evaluation with emphasis on deinterleaving and emitter recognition*. Norwegian University of Science and Technology.

Brendel, A. B., Lembcke, T.-B., Muntermann, J., & Kolbe, L. M. (2021). Toward replication study types for design science research. *Journal of Information Technology*, *36*(3), 198–215. https://doi.org/10.1177/02683962211006429

Buchenroth, A. (2015). *Ambiguity-Based Classification of Phase Modulated Waveforms*. Wright State University.

Chen, W., Fu, K., Zuo, J., Zheng, X., Huang, T., & Ren, W. (2017). Radar Emitter Classification for large data set based on weighted-xgboost. *IET Radar, Sonar & Navigation*, *11*(8), 1203–1207. https://doi.org/10.1049/iet-rsn.2016.0632

Chen, Y. M., Lin, C.-M., & Hsueh, C.-S. (2013). Identification of Highly Jittered Radar Emitters Signals based on Fuzzy Classification. *IOSR Journal of Engineering*, *3*(10), 53–59. https://doi.org/10.9790/3021-031015359

Chen, Y.-H., Lyu, S.-H., & Jiang, Y. (2021). Improving Deep Forest by Exploiting High-order Interactions. *2021 IEEE International Conference on Data Mining (ICDM)*, 1030–1035. https://doi.org/10.1109/ICDM51629.2021.00118

Feng, Y., Wang, G., Liu, Z., Cui, B., Yang, Y., Xu, X., & Han, H. (2021). Recognition of Radar Emitters with Agile Waveform Based on Hybrid Deep Neural Network and Attention Mechanism. *Radioengineering*, *30*(4), 704–712. https://doi.org/10.13164/re.2021.0704

Ferreira, J. (2020, março 26). *Machine Learning: The Data Science Loop*. Feedzai. https://feedzai.com/blog/machine-learning-the-data-science-loop/

Gama, J., & Brazdil, P. (2000). Cascade Generalization. *Machine Learning*, *41*, 315–343. https://doi.org/10.1023/A:1007652114878

Gençol, K. (2015). *New Methods For Radar Emitter Identification*. Anadolu University.

Gregor, S., & Hevner, A. R. (2013). Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, *37*(2), 337–355. https://doi.org/10.25300/MISQ/2013/37.2.01

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). *Why do tree-based models still outperform deep learning on tabular data?* 33. https://doi.org/10.48550/arXiv.2207.08815

Guo, Y., Liu, S., Li, Z., & Shang, X. (2018). BCDForest: A boosting cascade deep forest model towards the classification of cancer subtypes based on gene expression data. *BMC Bioinformatics*, *19*(S5), 118. https://doi.org/10.1186/s12859-018-2095-4

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Kim, S., Yu, Z., Kil, R. M., & Lee, M. (2015). Deep learning of support vector machines with class probability output networks. *Neural Networks*, *64*, 19–28. https://doi.org/10.1016/j.neunet.2014.09.007

Kvasnov, A. V. (2020). Methodology of classification and recognition of the radar emission sources based on Bayesian programming. *IET Radar, Sonar & Navigation*, *14*(8), 1175–1182. https://doi.org/10.1049/iet-rsn.2019.0380

Laan, M. J. van der, Polley, E. C., & Hubbard, A. E. (2007). Super Learner. *The Berkeley Electronic Press*, 22.

Ma, P., Wu, Y., Li, Y., Guo, L., Jiang, H., Zhu, X., & Wu, X. (2022). HW-Forest: Deep Forest with Hashing Screening and Window Screening. *ACM Transactions on Knowledge Discovery from Data*, *16*(6), 1–24. https://doi.org/10.1145/3532193

Ma, P., Wu, Y., Li, Y., Guo, L., & Li, Z. (2022). DBC-Forest: Deep forest with binning confidence screening. *Neurocomputing*, *475*, 112–122. https://doi.org/10.1016/j.neucom.2021.12.075

Meikle, H. (2008). *Modern Radar Systems* (2.ª ed.). Artech House.

Naimi, A. I., & Balzer, L. B. (2018). Stacked generalization: An introduction to super learning. *European Journal of Epidemiology*, *33*(5), 459–464. https://doi.org/10.1007/s10654-018-0390-z

Ni, S., & Kao, H.-Y. (2020). *PSForest: Improving Deep Forest via Feature Pooling and Error Screening*. *129*, 769–781. https://proceedings.mlr.press/v129/ni20a.html

Notaro, P., Paschali, M., Hopke, C., Wittmann, D., & Navab, N. (2019). *Radar Emitter Classification with Attribute-specific Recurrent Neural Networks* (arXiv:1911.07683). arXiv. http://arxiv.org/abs/1911.07683

Pang, M., Ting, K. M., Zhao, P., & Zhou, Z.-H. (2022). Improving Deep Forest by Screening. *IEEE Transactions on Knowledge and Data Engineering*, *34*(9), 4298–4312. https://doi.org/10.1109/TKDE.2020.3038799

Pang, M., Ting, K.-M., Zhao, P., & Zhou, Z.-H. (2018). Improving Deep Forest by Confidence Screening. *2018 IEEE International Conference on Data Mining (ICDM)*, 1194–1199. https://doi.org/10.1109/ICDM.2018.00158

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., & Hui, W. (2006). *The Design Science Research Process: A Model for Producing and Presenting Information Systems Research*. 83–106. http://urn.fi/URN:NBN:fi:jyu-201904092111

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Petrov, N., Jordanov, I., & Roe, J. (2013). Radar Emitter Signals Recognition and Classification with Feedforward Networks. *Procedia Computer Science*, *22*, 1192–1200. https://doi.org/10.1016/j.procs.2013.09.206

Qi, Z., Wang, B., Tian, Y., & Zhang, P. (2016). When Ensemble Learning Meets Deep Learning: A New Deep Support Vector Machine for Classification. *Knowledge-Based Systems*, *107*, 54–60. https://doi.org/10.1016/j.knosys.2016.05.055

Revillon, G. (2019). *Uncertainty in radar emitter classification and clustering*. Université Paris-Saclay.

Sebastian, E. (2017). *Radar target classification using Support Vector Machines and Mel Frequency Cepstral Coefficients*. KTH Royal Institute of Technology.

Sun, J., Xu, G., Ren, W., & Yan, Z. (2018). Radar Emitter Classification based on unidimensional convolutional neural network. *IET Radar, Sonar & Navigation*, *12*(8), 862–867. https://doi.org/10.1049/iet-rsn.2017.0547

Taghizadeh-Mehrjardi, R., Hamzehpour, N., Hassanzadeh, M., Heung, B., Ghebleh Goydaragh, M., Schmidt, K., & Scholten, T. (2021). Enhancing the accuracy of machine learning models using the super learner technique in digital soil mapping. *Geoderma*, *399*, 115108. https://doi.org/10.1016/j.geoderma.2021.115108

Wang, J., Wang, X., Tian, Y., Chen, Z., & Chen, Y. (2022). A Radar Emitter Recognition Mechanism Based on IFS-Tri-Training Classification Processing. *Electronics*, *11*(7), 1078. https://doi.org/10.3390/electronics11071078

Wang, Y., Bi, X., Chen, W., Li, Y., Chen, Q., & Long, T. (2019). Deep Forest for radar HRRP recognition. *The Journal of Engineering*, *2019*(21), 8018–8021. https://doi.org/10.1049/joe.2019.0723

Wolpert, D. H. (1992). Stacked Generalization. *Neural Networks*, *5*(2), 241–259. https://doi.org/10.1016/S0893-6080(05)80023-1

Xiao, Z., & Yan, Z. (2020). Radar Emitter Identification Based on Naive Bayesian Algorithm. *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 1242–1244. https://doi.org/10.1109/ITOEC49072.2020.9141663

Young, S., Abdou, T., & Bener, A. (2018). Deep Super Learner: A Deep Ensemble for Classification Problems. *Advances in Artificial Intelligence*, 84–95. https://doi.org/10.1007/978-3-319-89656-4_7

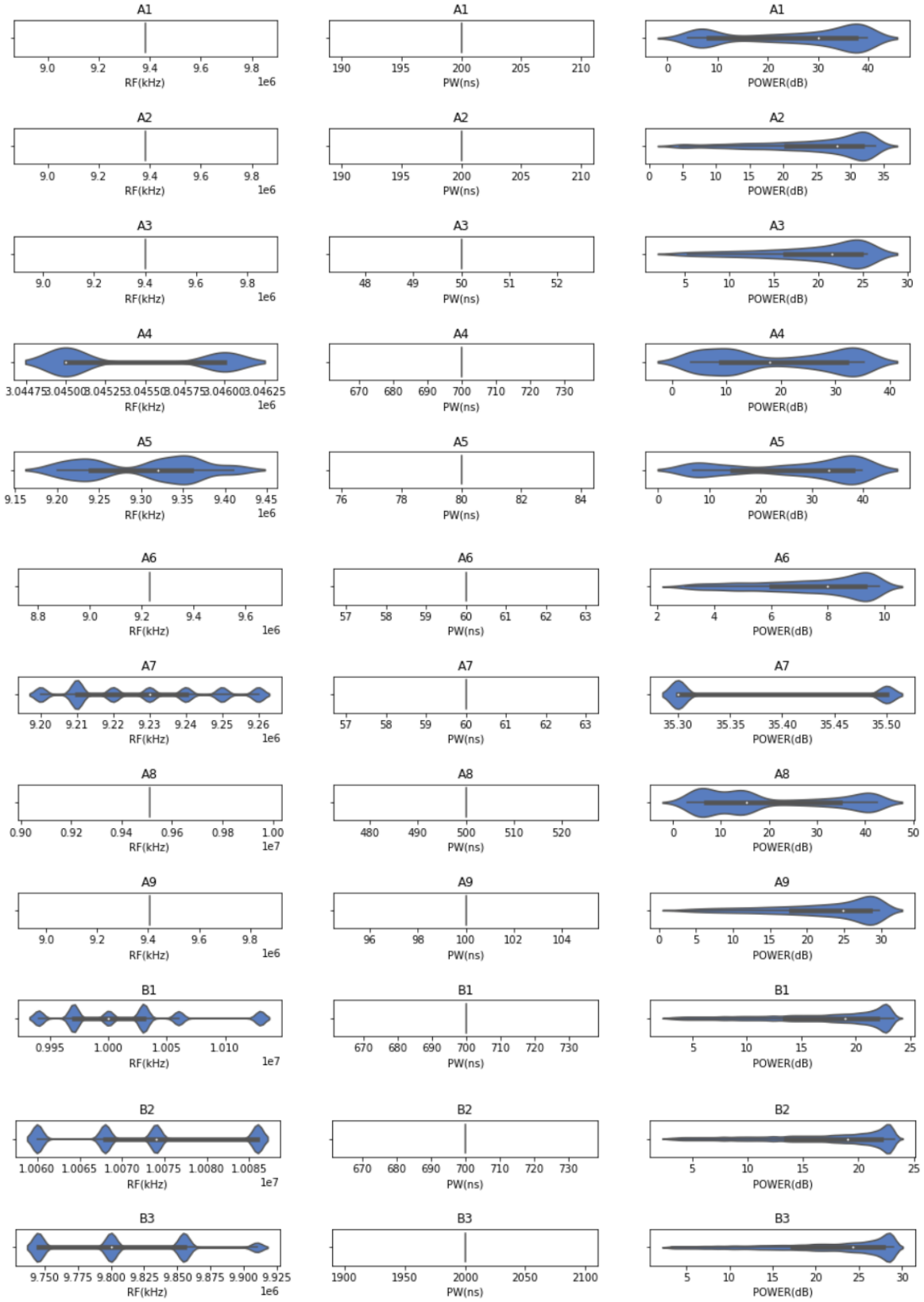
Zhang, J., Song, H., & Zhou, B. (2020). SAR Target Classification Based on Deep Forest Model. *Remote Sensing*, *12*(1), 128. https://doi.org/10.3390/rs12010128

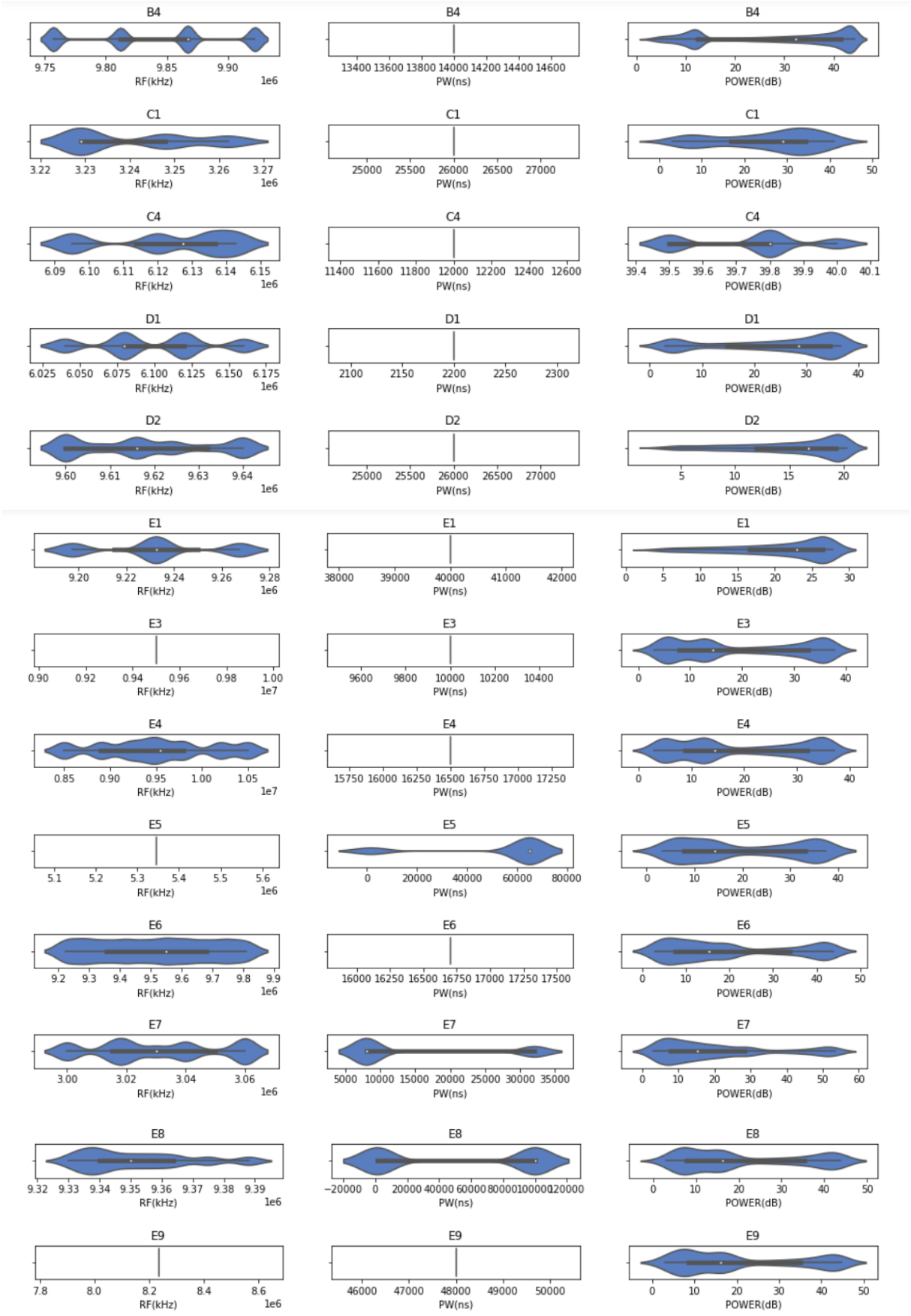
Zhou, Y., Wang, X., Chen, Y., & Tian, Y. (2020). Specific Emitter Identification via Bispectrum-Radon Transform and Hybrid Deep Model. *Mathematical Problems in Engineering*, *2020*, 1–17. https://doi.org/10.1155/2020/7646527

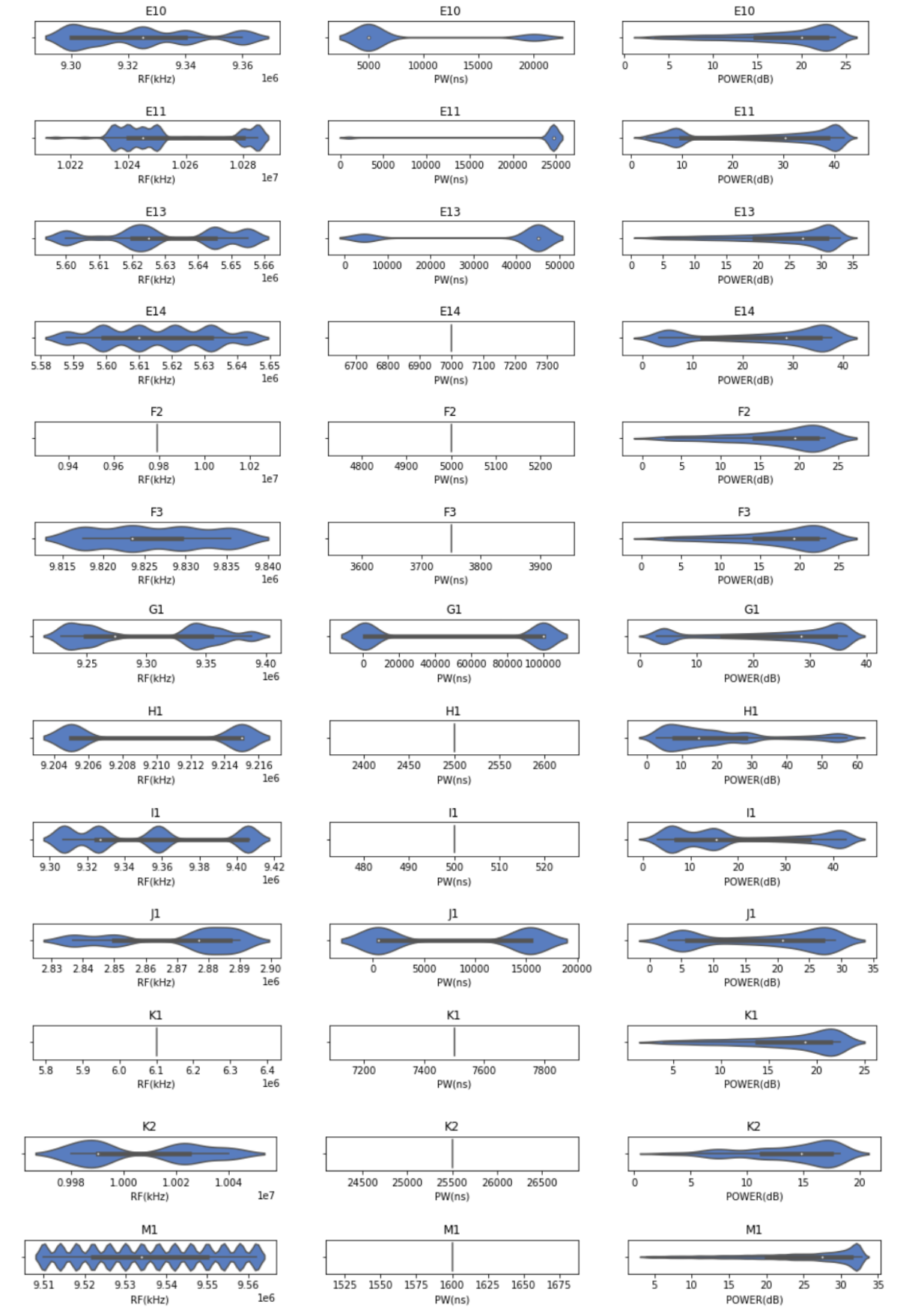
Zhou, Z.-H., & Feng, J. (2017). Deep Forest: Towards An Alternative to Deep Neural Networks. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 3553–3559. https://doi.org/10.24963/ijcai.2017/497

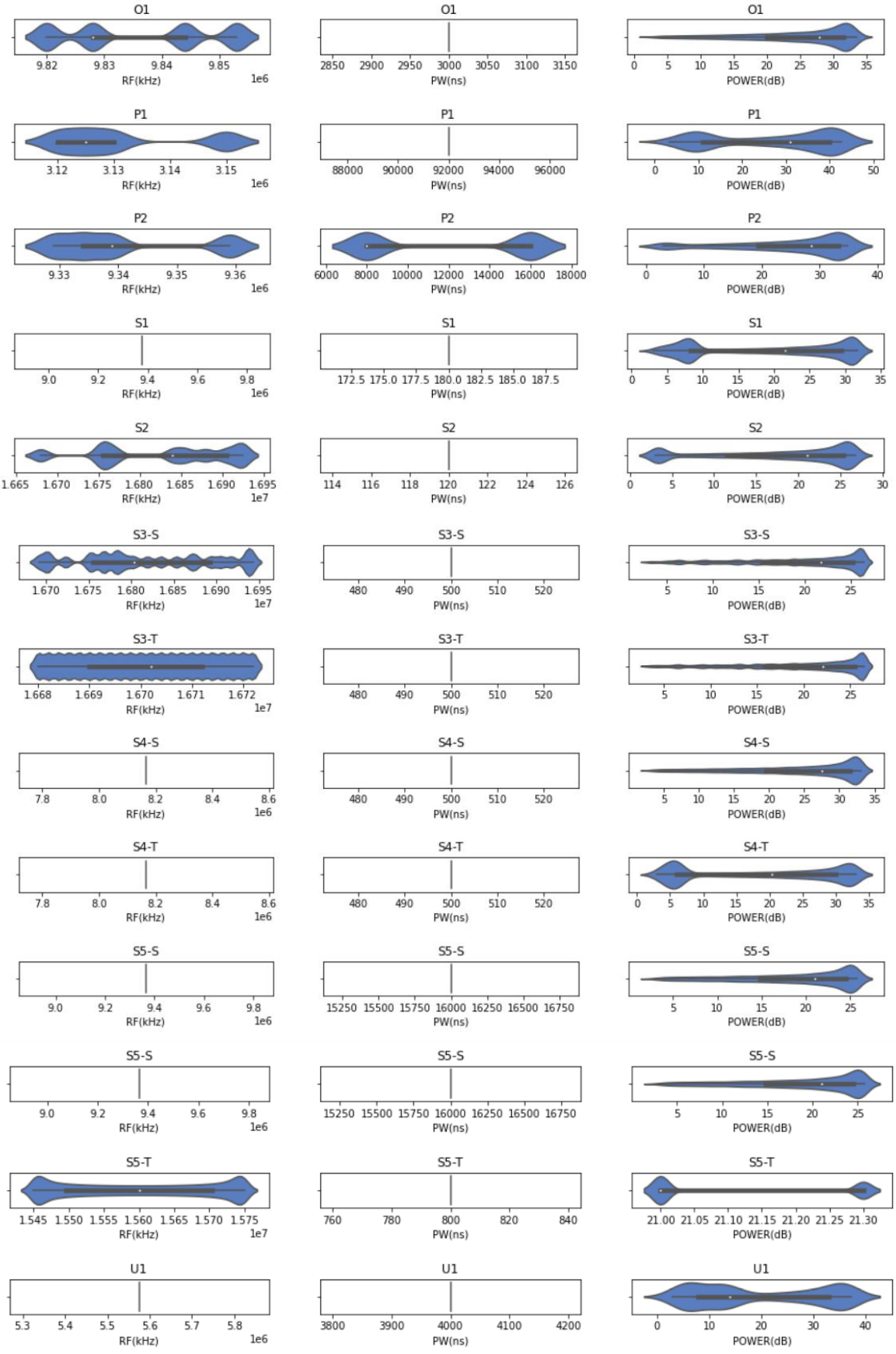
APPENDIX

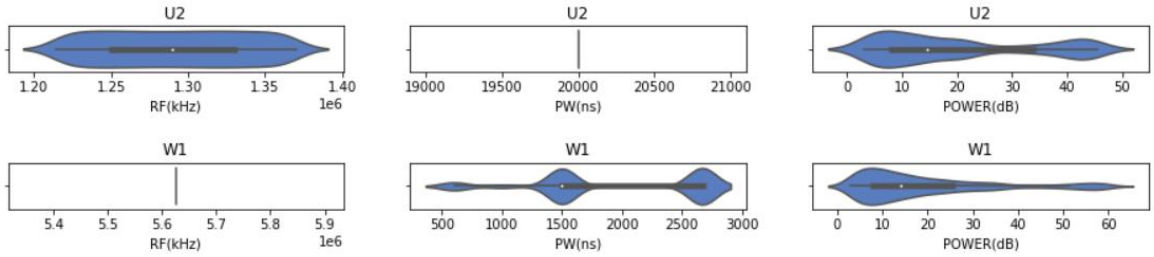
A.1. PDW DATASET - VIOLIN PLOTS OF CONTINUOUS FEATURES BY EMITTER



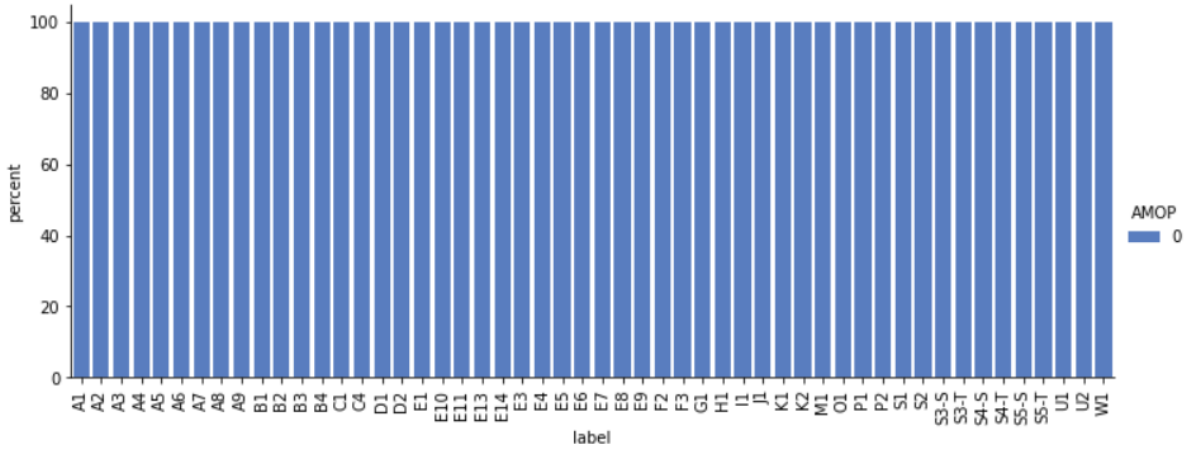

B4
9.75 9.80 9.85 9.90
RF(kHz)
1e6
B4
13400 13600 13800 14000 14200 14400 14600
PW(ns)
B4
0 10 20 30 40
POWER(dB)
C1
3.22 3.23 3.24 3.25 3.26 3.27
RF(kHz)
1e6
C1
25000 25500 26000 26500 27000
PW(ns)
C1
0 10 20 30 40 50
POWER(dB)
C4
6.09 6.10 6.11 6.12 6.13 6.14 6.15
RF(kHz)
1e6
C4
11400 11600 11800 12000 12200 12400 12600
PW(ns)
C4
39.4 39.5 39.6 39.7 39.8 39.9 40.0 40.1
POWER(dB)
D1
6.025 6.050 6.075 6.100 6.125 6.150 6.175
RF(kHz)
1e6
D1
2100 2150 2200 2250 2300
PW(ns)
D1
0 10 20 30 40
POWER(dB)
D2
9.60 9.61 9.62 9.63 9.64
RF(kHz)
1e6
D2
25000 25500 26000 26500 27000
PW(ns)
D2
5 10 15 20
POWER(dB)
E1
9.20 9.22 9.24 9.26 9.28
RF(kHz)
1e6
E1
38000 39000 40000 41000 42000
PW(ns)
E1
0 5 10 15 20 25 30
POWER(dB)
E3
0.90 0.92 0.94 0.96 0.98 1.00
RF(kHz)
1e7
E3
9600 9800 10000 10200 10400
PW(ns)
E3
0 10 20 30 40
POWER(dB)
E4
0.85 0.90 0.95 1.00 1.05
RF(kHz)
1e7
E4
15750 16000 16250 16500 16750 17000 17250
PW(ns)
E4
0 10 20 30 40
POWER(dB)
E5
5.1 5.2 5.3 5.4 5.5 5.6
RF(kHz)
1e6
E5
0 20000 40000 60000 80000
PW(ns)
E5
0 10 20 30 40
POWER(dB)
E6
9.2 9.3 9.4 9.5 9.6 9.7 9.8 9.9
RF(kHz)
1e6
E6
16000 16250 16500 16750 17000 17250 17500
PW(ns)
E6
0 10 20 30 40 50
POWER(dB)
E7
3.00 3.02 3.04 3.06
RF(kHz)
1e6
E7
5000 10000 15000 20000 25000 30000 35000
PW(ns)
E7
0 10 20 30 40 50 60
POWER(dB)
E8
9.32 9.33 9.34 9.35 9.36 9.37 9.38 9.39
RF(kHz)
1e6
E8
−20000 0 20000 40000 60000 80000 100000 120000
PW(ns)
E8
0 10 20 30 40 50
POWER(dB)
E9
7.8 8.0 8.2 8.4 8.6
RF(kHz)
1e6
E9
46000 47000 48000 49000 50000
PW(ns)
E9
0 10 20 30 40 50
POWER(dB)

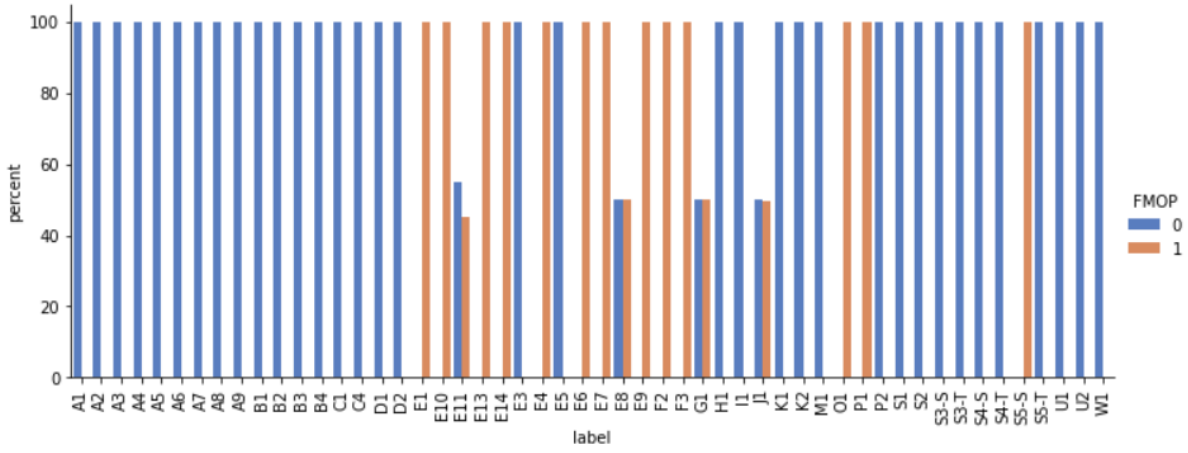

E10
9.30 9.32 9.34 9.36
RF(kHz)
1e6
E10
5000 10000 15000 20000
PW(ns)
E10
0 5 10 15 20 25
POWER(dB)
E11
1.022 1.024 1.026 1.028
RF(kHz)
1e7
E11
0 5000 10000 15000 20000 25000
PW(ns)
E11
0 10 20 30 40
POWER(dB)
E13
5.60 5.61 5.62 5.63 5.64 5.65 5.66
RF(kHz)
1e6
E13
0 10000 20000 30000 40000 50000
PW(ns)
E13
0 5 10 15 20 25 30 35
POWER(dB)
E14
5.58 5.59 5.60 5.61 5.62 5.63 5.64 5.65
RF(kHz)
1e6
E14
6700 6800 6900 7000 7100 7200 7300
PW(ns)
E14
0 10 20 30 40
POWER(dB)
F2
0.94 0.96 0.98 1.00 1.02
RF(kHz)
1e7
F2
4800 4900 5000 5100 5200
PW(ns)
F2
0 5 10 15 20 25
POWER(dB)
F3
9.815 9.820 9.825 9.830 9.835 9.840
RF(kHz)
1e6
F3
3600 3700 3800 3900
PW(ns)
F3
0 5 10 15 20 25
POWER(dB)
G1
9.25 9.30 9.35 9.40
RF(kHz)
1e6
G1
0 20000 40000 60000 80000 100000
PW(ns)
G1
0 10 20 30 40
POWER(dB)
H1
9.204 9.206 9.208 9.210 9.212 9.214 9.216
RF(kHz)
1e6
H1
2400 2450 2500 2550 2600
PW(ns)
H1
0 10 20 30 40 50 60
POWER(dB)
I1
9.30 9.32 9.34 9.36 9.38 9.40 9.42
RF(kHz)
1e6
I1
480 490 500 510 520
PW(ns)
I1
0 10 20 30 40
POWER(dB)
J1
2.83 2.84 2.85 2.86 2.87 2.88 2.89 2.90
RF(kHz)
1e6
J1
0 5000 10000 15000 20000
PW(ns)
J1
0 5 10 15 20 25 30 35
POWER(dB)
K1
5.8 5.9 6.0 6.1 6.2 6.3 6.4
RF(kHz)
1e6
K1
7200 7400 7600 7800
PW(ns)
K1
5 10 15 20 25
POWER(dB)
K2
0.998 1.000 1.002 1.004
RF(kHz)
1e7
K2
24500 25000 25500 26000 26500
PW(ns)
K2
0 5 10 15 20
POWER(dB)
M1
9.51 9.52 9.53 9.54 9.55 9.56
RF(kHz)
1e6
M1
1525 1550 1575 1600 1625 1650 1675
PW(ns)
M1
5 10 15 20 25 30 35
POWER(dB)

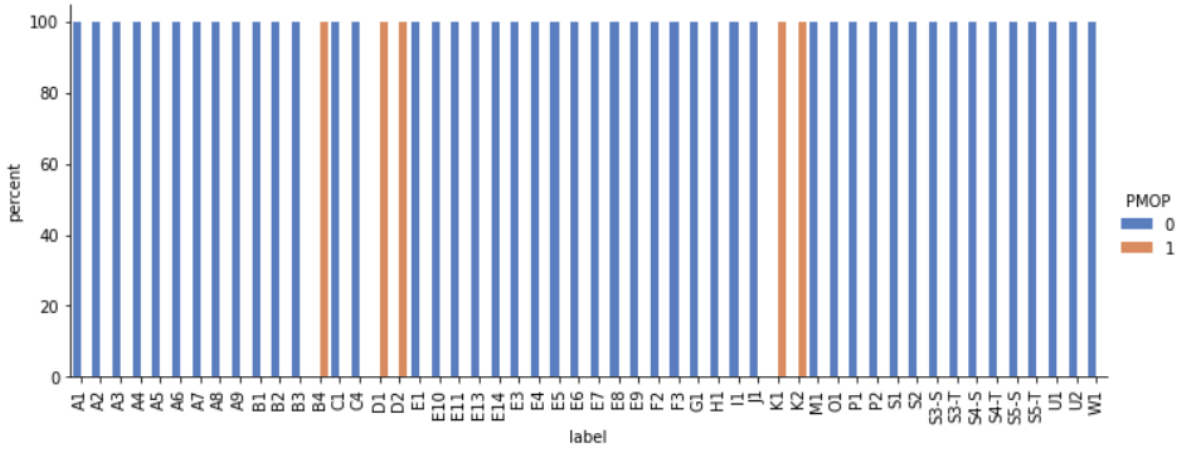

O1
9.82 9.83 9.84 9.85
RF(kHz)
1e6
O1
2850 2900 2950 3000 3050 3100 3150
PW(ns)
O1
0 5 10 15 20 25 30 35
POWER(dB)
P1
3.12 3.13 3.14 3.15
RF(kHz)
1e6
P1
88000 90000 92000 94000 96000
PW(ns)
P1
0 10 20 30 40 50
POWER(dB)
P2
9.33 9.34 9.35 9.36
RF(kHz)
1e6
P2
6000 8000 10000 12000 14000 16000 18000
PW(ns)
P2
0 10 20 30 40
POWER(dB)
S1
9.0 9.2 9.4 9.6 9.8
RF(kHz)
1e6
S1
172.5 175.0 177.5 180.0 182.5 185.0 187.5
PW(ns)
S1
0 5 10 15 20 25 30 35
POWER(dB)
S2
1.665 1.670 1.675 1.680 1.685 1.690 1.695
RF(kHz)
1e7
S2
114 116 118 120 122 124 126
PW(ns)
S2
0 5 10 15 20 25 30
POWER(dB)
S3-S
1.670 1.675 1.680 1.685 1.690 1.695
RF(kHz)
1e7
S3-S
480 490 500 510 520
PW(ns)
S3-S
5 10 15 20 25
POWER(dB)
S3-T
1.668 1.669 1.670 1.671 1.672
RF(kHz)
1e7
S3-T
480 490 500 510 520
PW(ns)
S3-T
5 10 15 20 25
POWER(dB)
S4-S
7.8 8.0 8.2 8.4 8.6
RF(kHz)
1e6
S4-S
480 490 500 510 520
PW(ns)
S4-S
5 10 15 20 25 30 35
POWER(dB)
S4-T
7.8 8.0 8.2 8.4 8.6
RF(kHz)
1e6
S4-T
480 490 500 510 520
PW(ns)
S4-T
0 5 10 15 20 25 30 35
POWER(dB)
S5-S
9.0 9.2 9.4 9.6 9.8
RF(kHz)
1e6
S5-S
15250 15500 15750 16000 16250 16500 16750
PW(ns)
S5-S
5 10 15 20 25
POWER(dB)
S5-S
9.0 9.2 9.4 9.6 9.8
RF(kHz)
1e6
S5-S
15250 15500 15750 16000 16250 16500 16750
PW(ns)
S5-S
5 10 15 20 25
POWER(dB)
S5-T
1.545 1.550 1.555 1.560 1.565 1.570 1.575
RF(kHz)
1e7
S5-T
760 780 800 820 840
PW(ns)
S5-T
21.00 21.05 21.10 21.15 21.20 21.25 21.30
POWER(dB)
U1
5.3 5.4 5.5 5.6 5.7 5.8
RF(kHz)
1e6
U1
3800 3900 4000 4100 4200
PW(ns)
U1
0 10 20 30 40
POWER(dB)

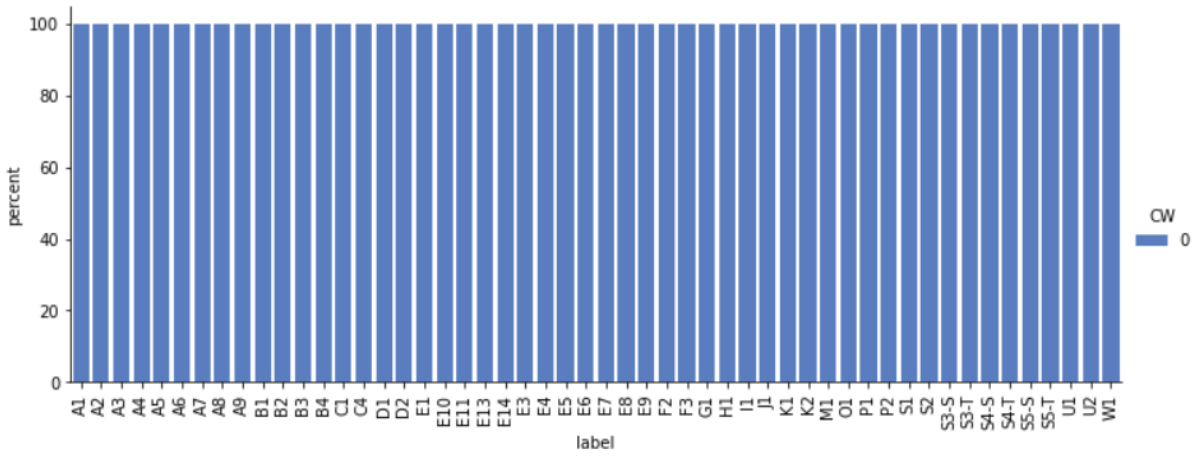


A.2. PDW DATASET - CATEGORICAL FEATURES RELATION WITH EMITTER

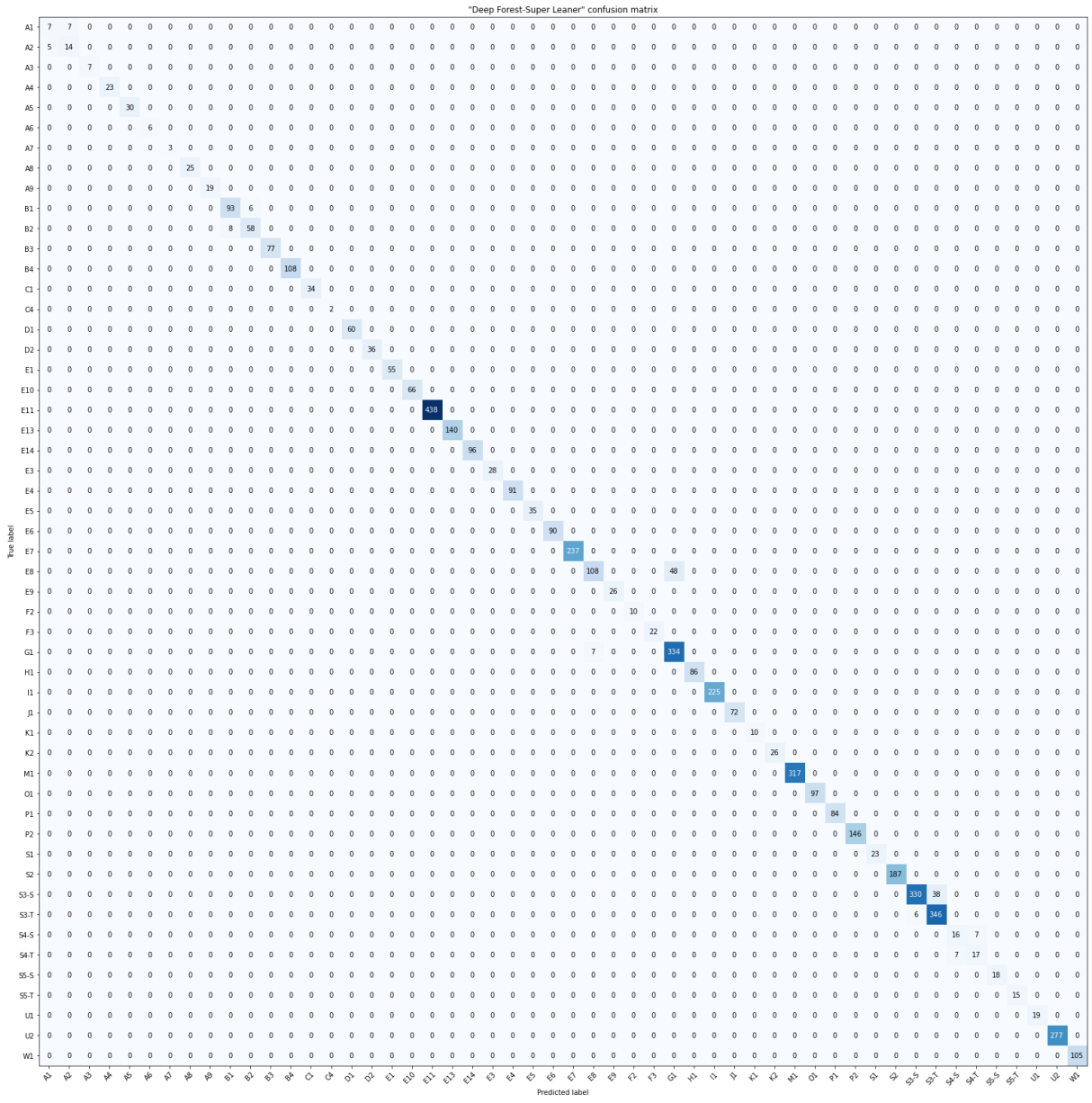


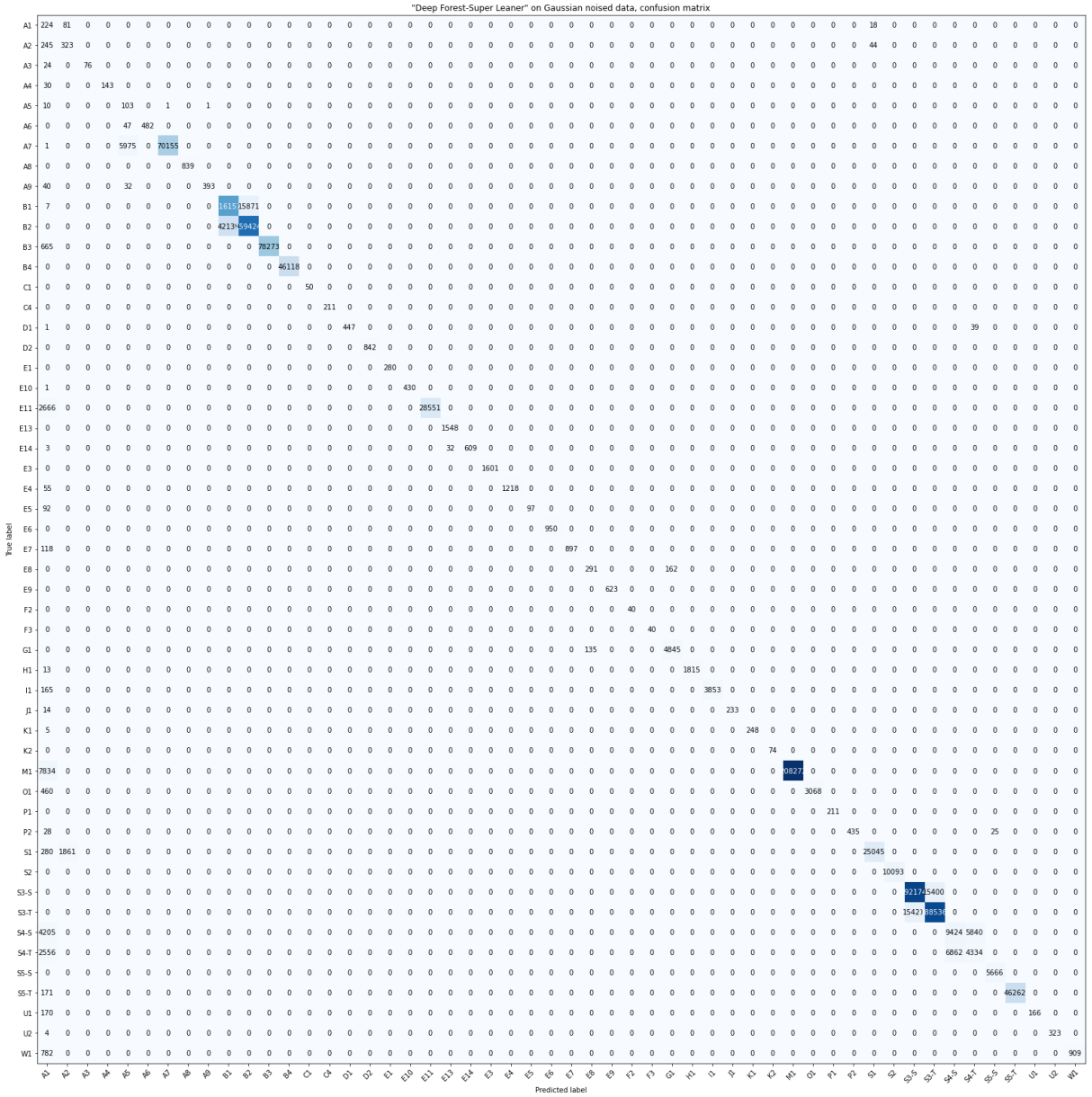


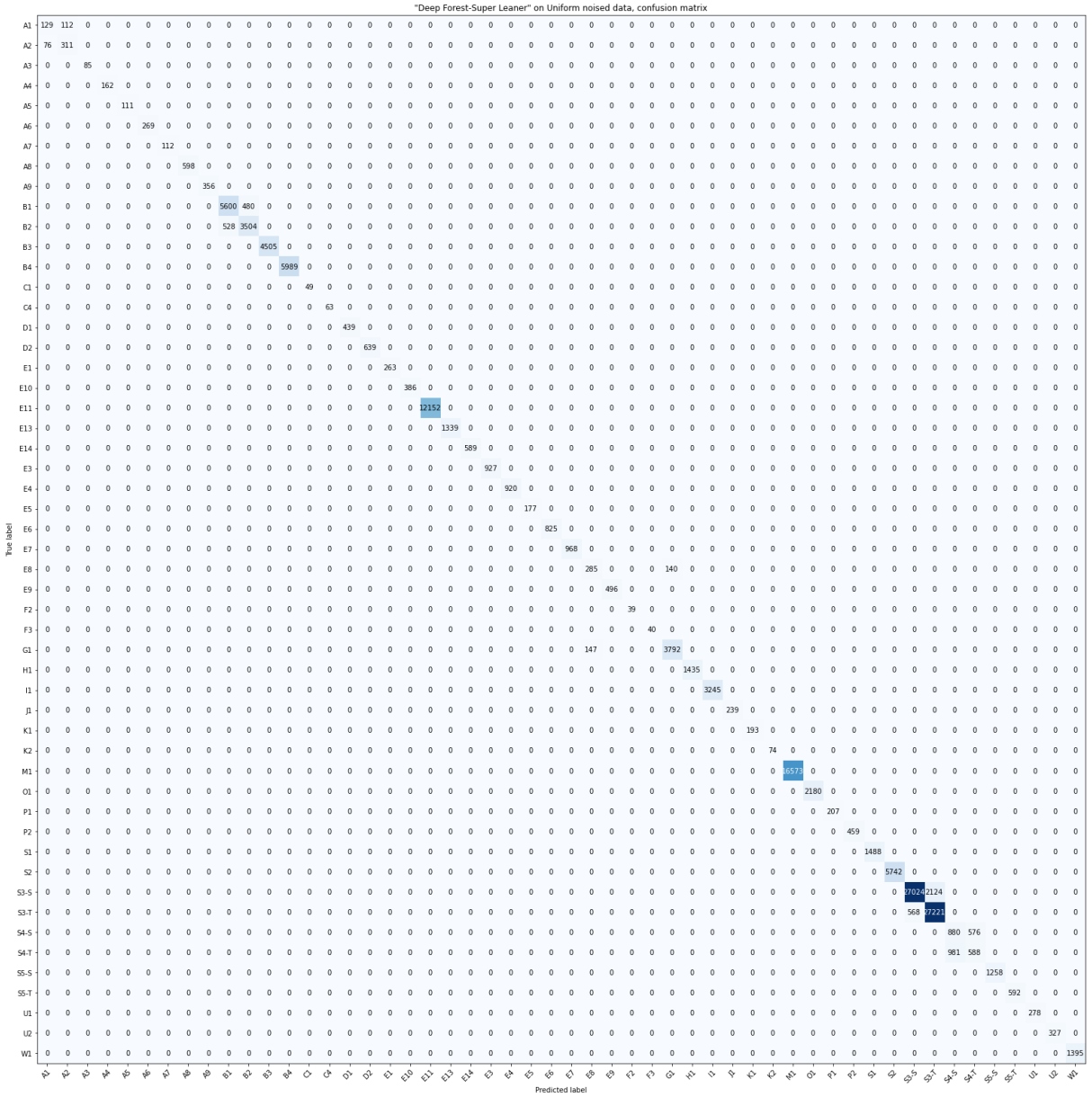




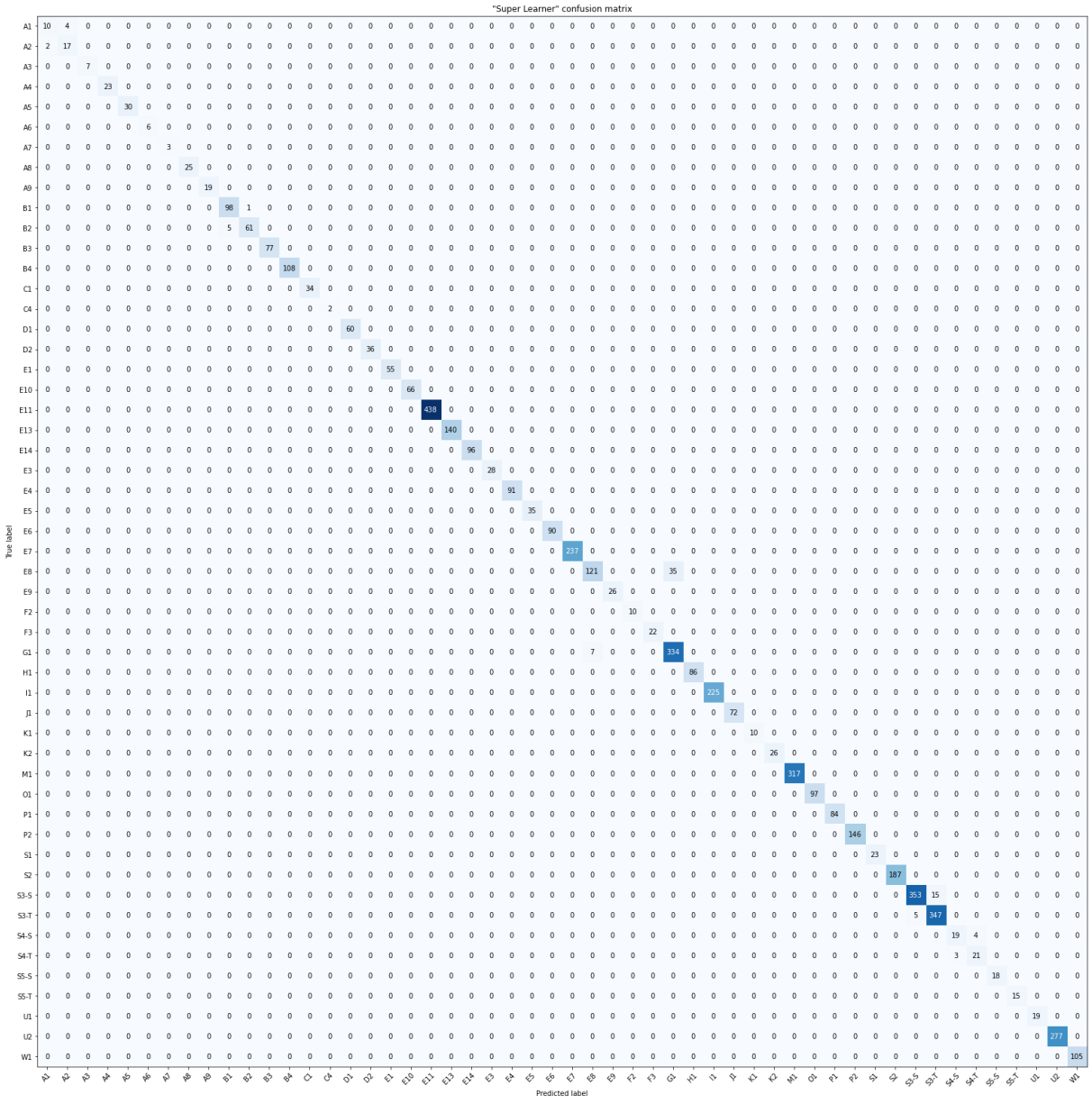
A.3. OPTIMIZED DF-SL CONFUSION MATRIX

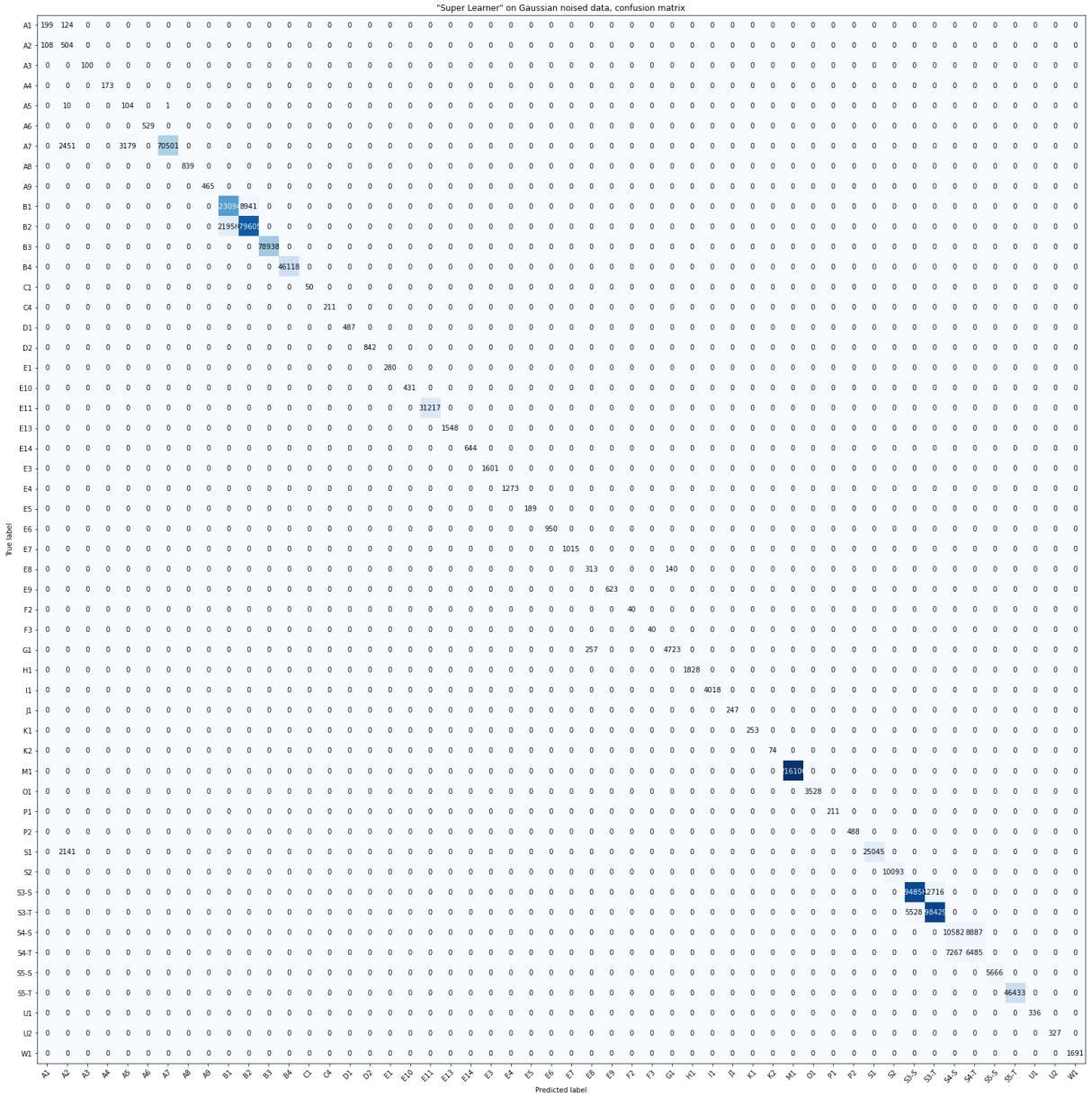


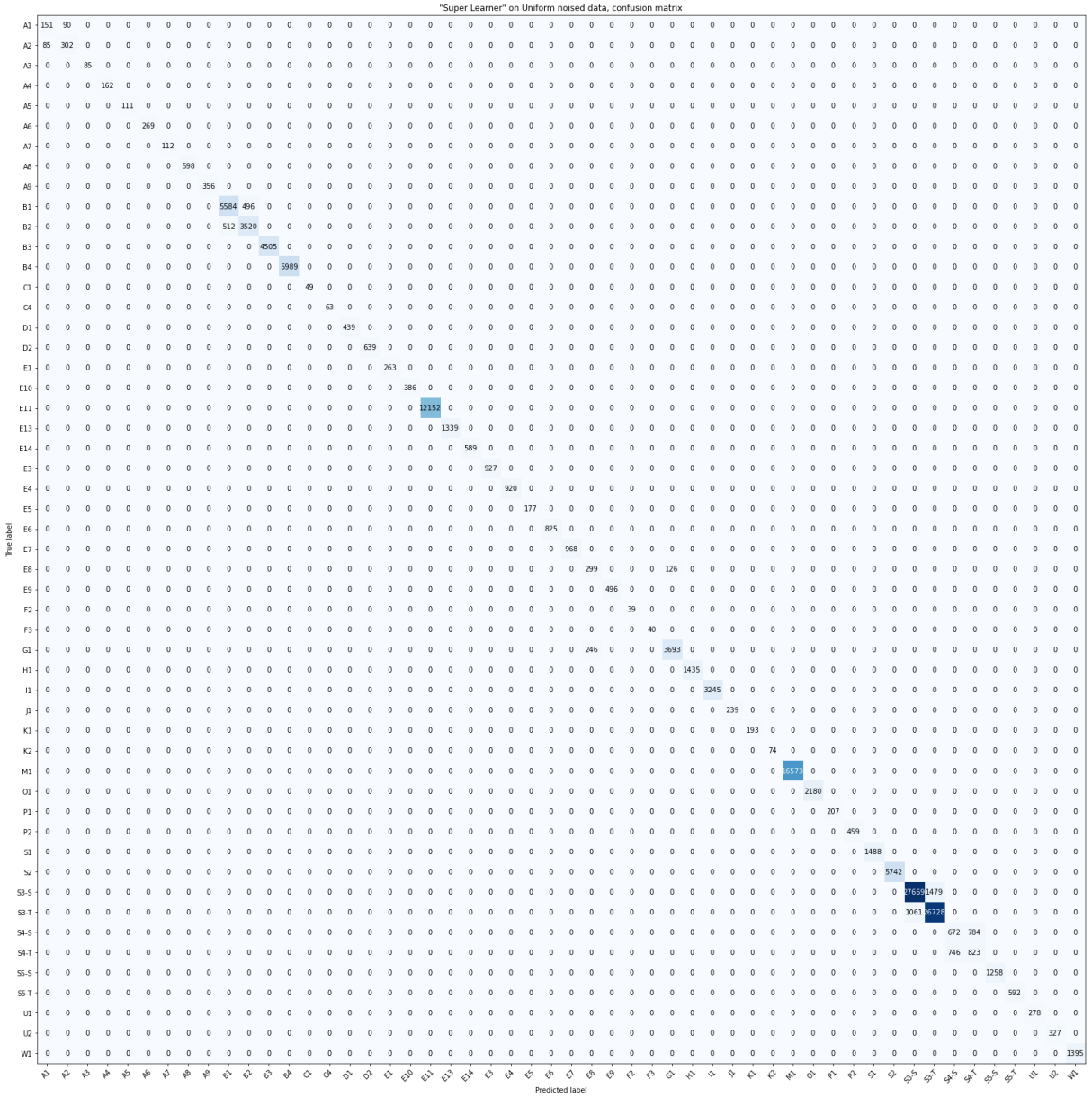

"Deep Forest-Super Leaner" on Gaussian noised data, confusion matrix
True label
Predicted label
A1 A2 A3 A4 A5 A6 A7 A8 A9 B1 B2 B3 B4 C1 C4 D1 D2 E1 E10 E11 E13 E14 E3 E4 E5 E6 E7 E8 E9 F2 F3 G1 H1 I1 J1 K1 K2 M1 O1 P1 P2 S1 S2 S3-S S3-T S4-S S4-T S5-S S5-T U1 U2 W1


"Deep Forest-Super Leaner" on Uniform noised data, confusion matrix
True label
Predicted label
A1 A2 A3 A4 A5 A6 A7 A8 A9 B1 B2 B3 B4 C1 C4 D1 D2 E1 E10 E11 E13 E14 E3 E4 E5 E6 E7 E8 E9 F2 F3 G1 H1 I1 J1 K1 K2 M1 O1 P1 P2 S1 S2 S3-S S3-T S4-S S4-T S5-S S5-T U1 U2 W1

A.4. OPTIMIZED SUPER LEARNER CONFUSION MATRIX




"Super Learner" on Gaussian noised data, confusion matrix
True label
Predicted label


"Super Learner" on Uniform noised data, confusion matrix
True label
Predicted label
A1 A2 A3 A4 A5 A6 A7 A8 A9 B1 B2 B3 B4 C1 C4 D1 D2 E1 E10 E11 E13 E14 E3 E4 E5 E6 E7 E8 E9 F2 F3 G1 H1 I1 J1 K1 K2 M1 O1 P1 P2 S1 S2 S3-S S3-T S4-S S4-T S5-S S5-T U1 U2 W1

A.5. MODELS' PERFORMANCE ACROSS ALL DATASETS