POLITECNICO
MILANO 1863

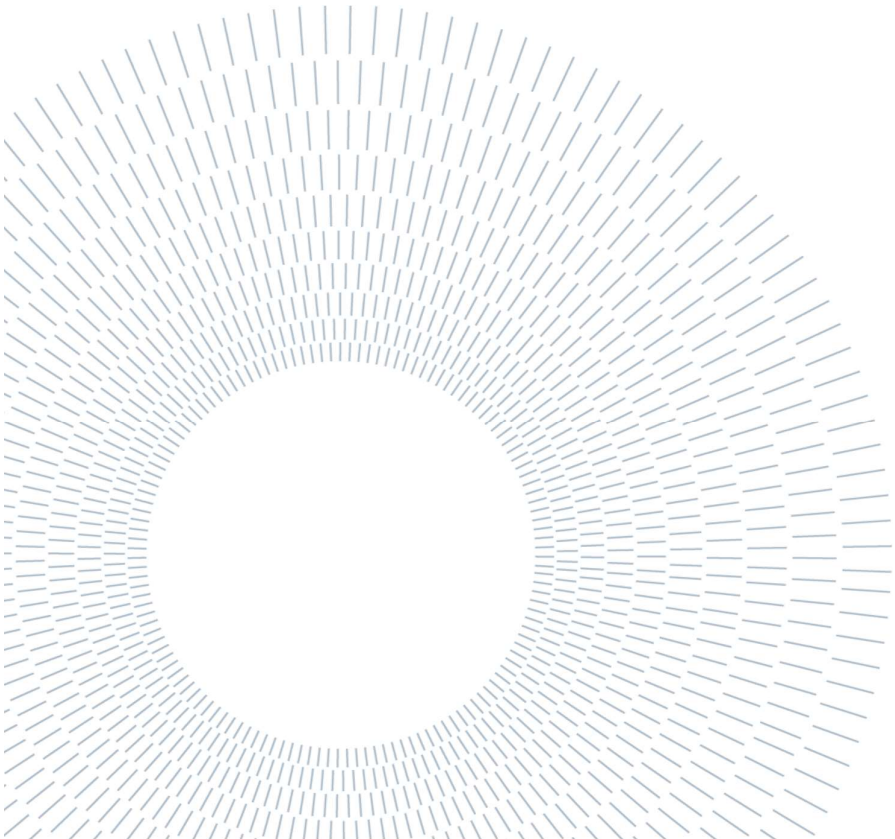
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Predicting the accuracy for streaming machine learning systems

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING
INGEGNERIA INFORMATICA

Author: **Priyanka Rajendra**

Student ID:	945298
Advisor:	Danilo Ardagna
Co-advisor:	Bruno Guindani
Academic Year:	2022-2023



Abstract

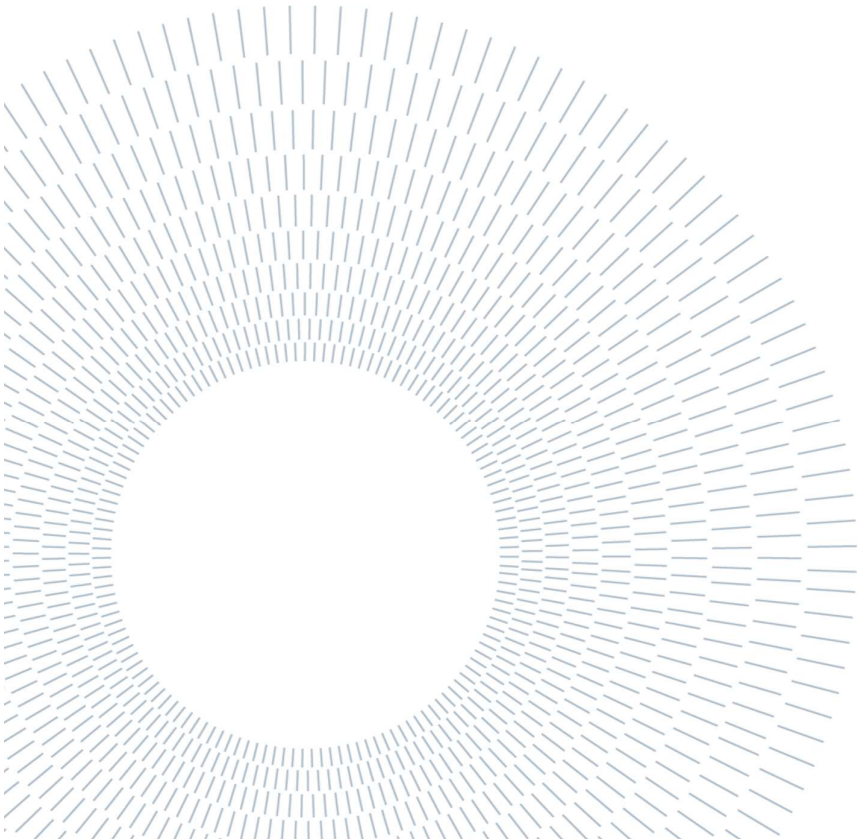
Machine learning (ML) algorithms provide better results when their hyperparameters are fine-tuned. However, finding the suitable algorithms and parameters for a given task is a time-consuming effort. AutoML techniques solve this issue by automating data pre-processing, feature engineering, model selection and hyperparameter tuning in ML workflows. However, AutoML doesn't seem to perform well with streaming data because of concept drift. The EvoAutoML algorithm was proposed as a solution to the underlying problem. EvoAutoML uses natural selection inspired by Genetic Algorithms, which at each iteration identify the best and the worst element in their model ensembles. It then replaces the worst model with a new model which is mutated using the best model under the assumption that it will provide better results in the next iteration. Though in some cases, the newly replaced models are not good performers. To avoid this, we may conduct accuracy prediction for each model, based on the available historic data, in order to assess which model should be replaced. By using the accuracy data available from the experimental results of the EvoAutoML algorithm, it is possible to rank each type of model according to their performance, including the number of appearances per ensemble throughout the iterations. This way, we can assess whether some model will perform better for the upcoming data by choosing the higher ranked model to replace the worst model identified by the EvoAutoML algorithm.

Keywords: Evolutionary algorithms, data analysis, model ranking, accuracy prediction, streaming data.

Abstract in lingua italiana

Gli algoritmi di machine learning (ML) forniscono risultati migliori quando i loro iperparametri sono regolati con precisione. Trovare gli algoritmi e i parametri adatti per un determinato problema richiede però molto tempo. Le tecniche AutoML risolvono questo problema automatizzando il pre-processing dei dati, l'elaborazione delle loro feature, la selezione del modello e la regolazione degli iperparametri nei workflow ML. Tuttavia, l'AutoML non funziona bene con i dati in streaming a causa del cosiddetto fenomeno del "concept drift". L'algoritmo EvoAutoML è stato proposto come soluzione a questo problema. EvoAutoML utilizza la selezione naturale ispirata dai Genetic Algorithms, i quali ad ogni iterazione identificano il modello migliore e il peggiore nella loro ensemble di modelli. Esso sostituisce quindi il modello peggiore con un nuovo modello che viene mutato a partire dal migliore, nell'ipotesi che fornirà risultati migliori nell'iterazione successiva. Tuttavia, in alcuni casi i nuovi modelli non hanno buone prestazioni. Per evitare questo problema, possiamo tentare di prevedere l'accuratezza di ogni modello, partento dai dati storici disponibili, per individuare con precisione qual è il modello da sostituire. Utilizzando i dati di accuratezza derivanti dai risultati sperimentali dell'algoritmo EvoAutoML, è possibile stilare una classifica dei tipi di modello in base al loro livello di performance, inclusa la loro frequenza di apparizione nell'ensemble nel corso delle iterazioni. In questo modo, possiamo valutare se un modello avrà prestazioni migliori sui dati successivi, scegliendo il modello con il rango più alto per sostituire il modello peggiore identificato dall'algoritmo EvoAutoML.

Parole chiave: Algoritmi evolutivi, analisi dati, classifica dei modelli, previsione dell'accuratezza, dati in streaming.



Contents

1 Introduction

In the digital era, there is an exceptional need for machine learning (ML) techniques to predict useful information from the raw data. However, machine learning algorithms operating on huge data are quite computationally expensive since it invloves lot of training. Selecting the correct parameters is an important part of ML model selection, to get better accuracy in training and prediction. Human intervention in selecting the right parameters for the ML algorithm to achieve good results is a tedious and time-consuming process since it is a trial-and-error process until the expected performance is achieved [9].

1.1. AutoML

Automated Machine Learning (AutoML) techniques are introduced to solve the problem of selecting the correct parameters. AutoML is a collection of methods and resources that streamline the creation of machine learning models by minimizing the amount of human work needed to complete processes like feature selection, algorithm selection, hyperparameter tuning, and model validation. AutoML aims to streamline and speed up the machine learning workflow [2]. However, these techniques perform well only with offline data and often cannot handle streaming data.

1.2. What is Streaming data?

Streaming data are data generated continuously by different sources. It consists of different types of data like transaction data, multimedia information generated by social networks, purchases made on e-commerce websites etc. AutoML techniques cannot handle this type of data naturally. When data is non-stationary, patterns and correlations vary with time, and the distribution of the underlying data may also change. When the statistical characteristics of the attributes and their target classes alter over time, a process called concept drift takes place. Concept drift frequently occurs in an environment that is changing dynamically. Applications including anomaly detection, credit risk analysis,

biometric authentication, and antibiotic resistance prediction all experience concept drift.

Therefore, classifiers utilized in these applications must be able to perform effectively in presence of concept drift.

1.3. EvoAutoML

Evolutionary algorithms are inspired by the biological process of natural selection and used to find the best solution by sorting through a population of candidates using mutation, crossover, and selection operators. Evolutionary Automated Machine Learning (EvoAutoML) uses this approach to select hyperparameters thus avoiding retraining of an AutoML learner which is very expensive [1]. By finding the joint algorithm combination and hyperparameter setting which reduces the predefined loss over a data stream, it provides solution to the Online CASH (Combined Algorithm Selection and Hyperparameter) [11] problem for a ML classification task.

The input to this algorithm consists of a data stream S, a population size P, and a sampling rate f_{SS}. The length of the input data stream S provides the number of updates $\lfloor t/f_{SS} \rfloor$ and is probably infinite. The sampling rate f_{SS} regulates the rate at which a mutation is applied. The loss function L estimates the performance of a pipeline configuration on the examples provided e_i.

The algorithm is initialized with a random population of algorithms, called the ensemble. This ensures that the EvoAutoML algorithm is able to give predictions at any given time. When the data stream is started, mutation at the rate of f_{SS} is applied. The algorithm selects P^{best}, P^{weak} pipeline configurations and the mutation are applied based on the P^{best} pipeline configuration and its corresponding hyperparameters are changed randomly. It is then added to p and P^{weak} is removed from the population. Now, the population will be trained on the new instance, e_i.. The algorithm searches for suitable configurations to be used for prediction from the population that is updated in an ensemble manner during the training process. It uses the hard majority voting approach of the algorithm configurations to predict an unlabeled instance.

$$\hat{y}_i = mode\{P.predict(e_i)\} \in p$$

$\hat{y}_i$ is the predicted label and p being the set of algorithmic configurations.

EvoAutoML uses different datasets with different complexities and generated using different synthetic data stream generators. An algorithm

configuration contains both a scaler and a classifier (see Chapter 2.1) having 12 possible configurations. The population size, P=10 which is equal to the ensemble learners and a sampling rate of $f_{SS} = 1000$ is used. It uses interleaved *test-then-train* evaluation where the new data is first used to test the current performance of the algorithm and then to train and update the algorithm.

Predictive performance, total running time and memory consumption are used to evaluate the algorithm with other baseline algorithms like Online Bagging (OB)[12], Hoeffding Tree (HT) [13], Leveraging Bagging (LB) [14], Adaptive Random Forest (ARF) [15], K-Nearest Neighbors (KNN) and Gaussian Naïve Bayes (GNB). From the results provided in [1] it is known that EvoAutoML outperforms all the baseline algorithms in all the evaluation metrics considered. The accuracy that is given is measured as the final percentage of correctly classified examples.

1.4. Need for Accuracy Prediction

The main objective of this work is to predict the accuracy of the streaming machine learning systems by in turn using machine learning techniques. The model that performs better has the highest accuracy and tends to survive in almost every new instance, e_i. This model is assigned the highest rank. Accuracy prediction for every model or the algorithm configuration used in the EvoAutoML algorithm is necessary to assign the ranks. The model with the worst predicted accuracy or the one with the lowest rank will be ultimately removed, because it is likely to provide the worst results.

The experimental results of EvoAutoML, the prediction accuracy for the datasets under consideration and the features of the dataset that are used to predict the classification target (unlabelled instance) are used for the prediction of accuracy in streaming machine learning systems. By predicting the accuracy that is likely to be achieved by the algorithm, this could be a valuable metric for choosing the best models for replacement, ultimately leading to improved performance and better results.

2 Unleashing the Insights: A Thorough Analysis of the Dataset

The primary step to begin with any prediction task is to analyze the data to get some meaningful insights to understand the data trend and relationship between the different features of the data. This helps to derive well-informed decisions based on the findings.

To predict the accuracy of the streaming data it is necessary to choose wisely the datasets for consideration. Let us take the Agrawal dataset that has many features, thus representing high complexity and large datasets, and the Sine dataset that has only two features, thus representing low complexity and small datasets.

The Sine dataset is generated by a data stream with abrupt concept drift as described in [26]. It generates 4 numerical features, two features relevant for classification task and two noises (optional). The two relevant features are generates as a value from 0 to 1 and the classification function determines whether to classify the instance as class_0 or class_1 [24]. The data trend for feature_0 and feature_1 fluctuates from 0.47 to 0.53, with no discernible increasing or decreasing trend.

The data exploration step involves visualizing the data trends. The following are the trends of each feature in the Sine dataset (See Figure 1 and Figure 2). It has almost 1 million data points, but for the purpose of visualization each feature value is split into chunks of 1000 data and the average of each chunk is represented. We therefore have about 1000 points in the visualization.

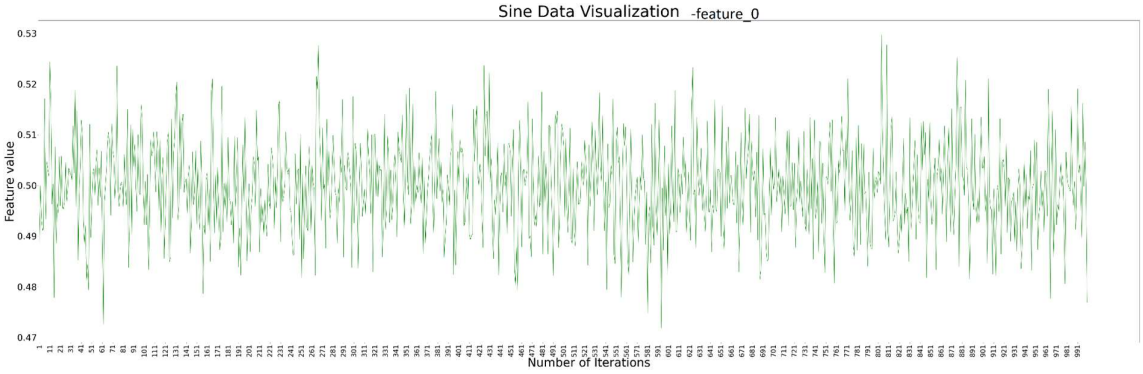


Figure 1: Data trend for feature_0 of Sine dataset.

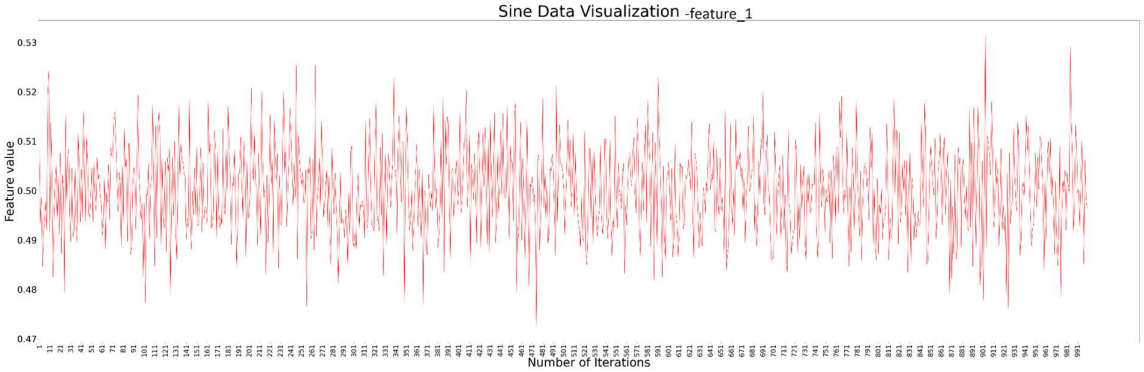


Figure 2: Data trend for feature_1 of Sine dataset.

The Agrawal dataset is produced by a stream generator that contains nine features (age, car, commission, elevel, hvalue, hyears, loan, zipcode and salary), six of which numerical and three categorical, and the classification target is either 0 or 1 which determines whether the loan should be approved or not.

The salary values are distributed form 20K to 150K, commission from 10K to 75K if salary is greater than 75K else 0, age from 20 to 80, education level (elevel) from 0 to 4, car maker from 1 to 20, zipcode from 0 to 8, house value (hvalue) from 50K*zipcode to 100K*zipcode, years of house owned (hyears) from 1 to 30, total loan amount from 0 to 500K [25]. The data trend for all the features fluctuates, and we cannot observe neither an increasing nor a decreasing trend.

The following are the trends of each feature in Agrawal dataset (See Figures 3 to Figure 11). As before, we have about 1000 points in the visualization.

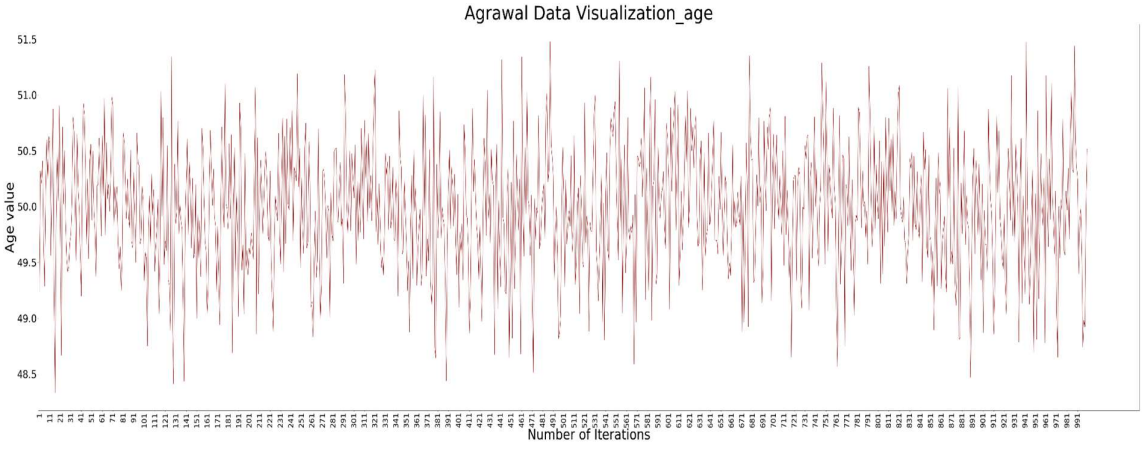


Figure 3: Data trend for feature age of Agrawal dataset.

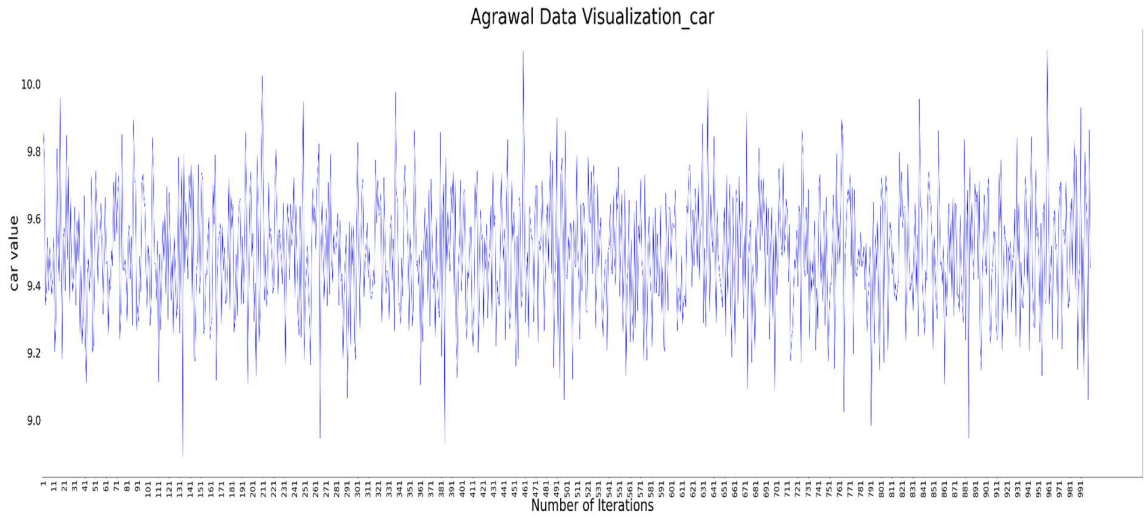


Figure 4: Data trend for feature car of Agrawal dataset.

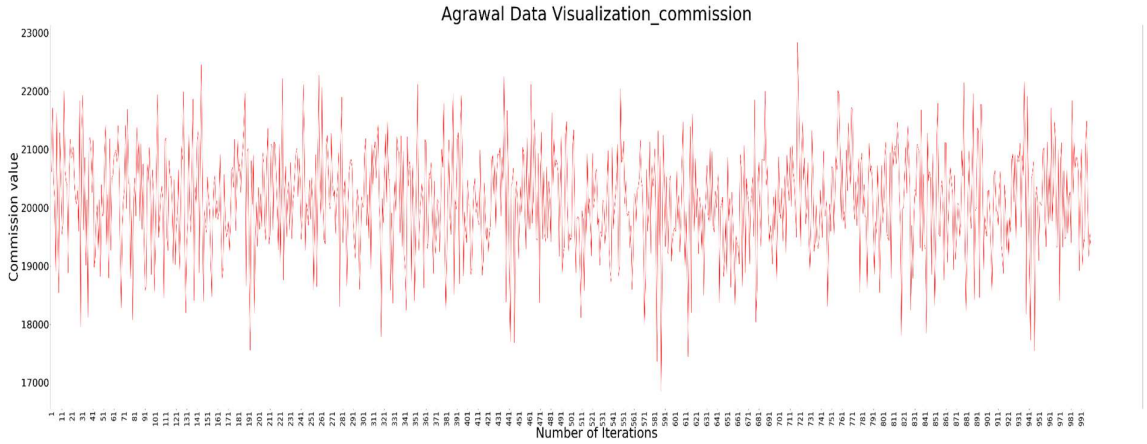


Figure 5: Data trend for feature commission of Agrawal dataset.

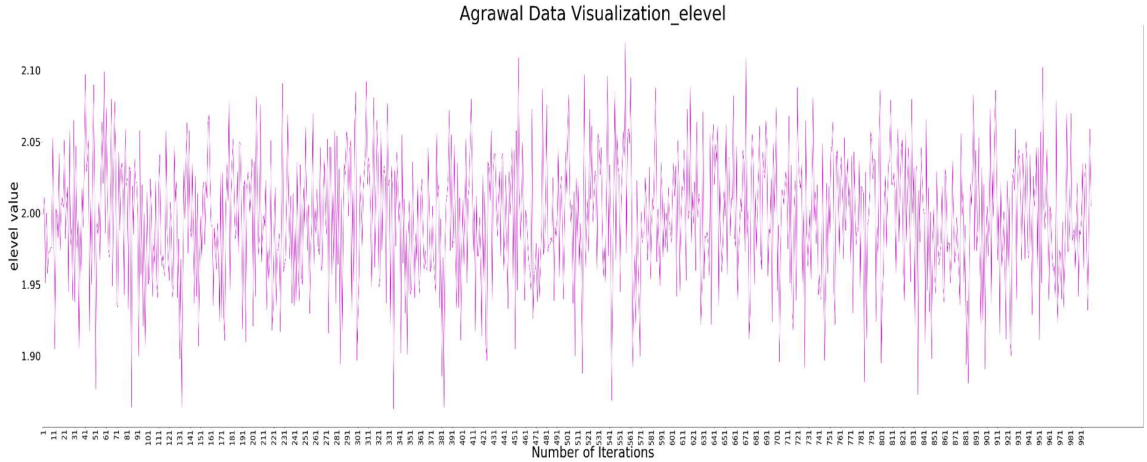


Figure 6: Data trend for feature elevel of Agrawal dataset.

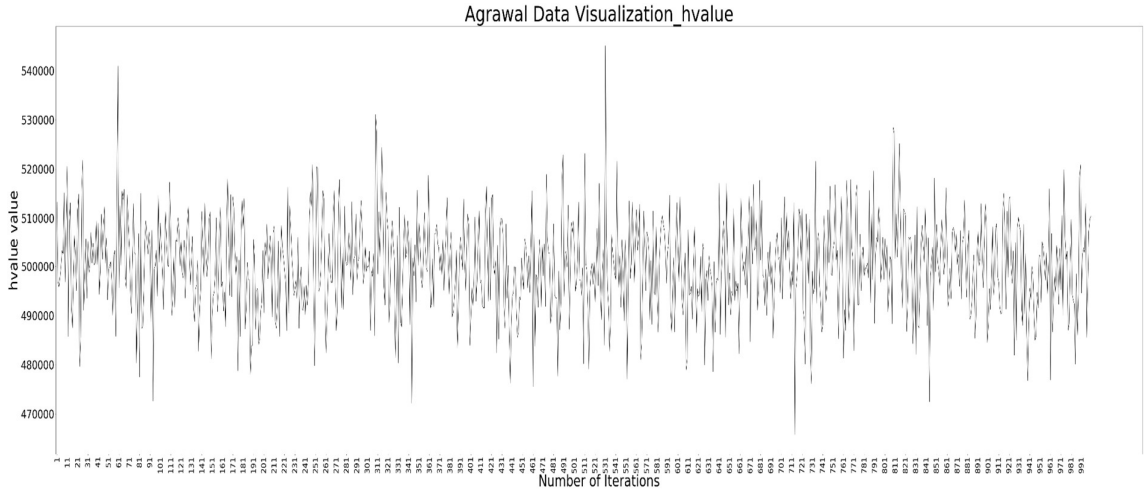


Figure 7: Data trend for feature hvalue of Agrawal dataset.

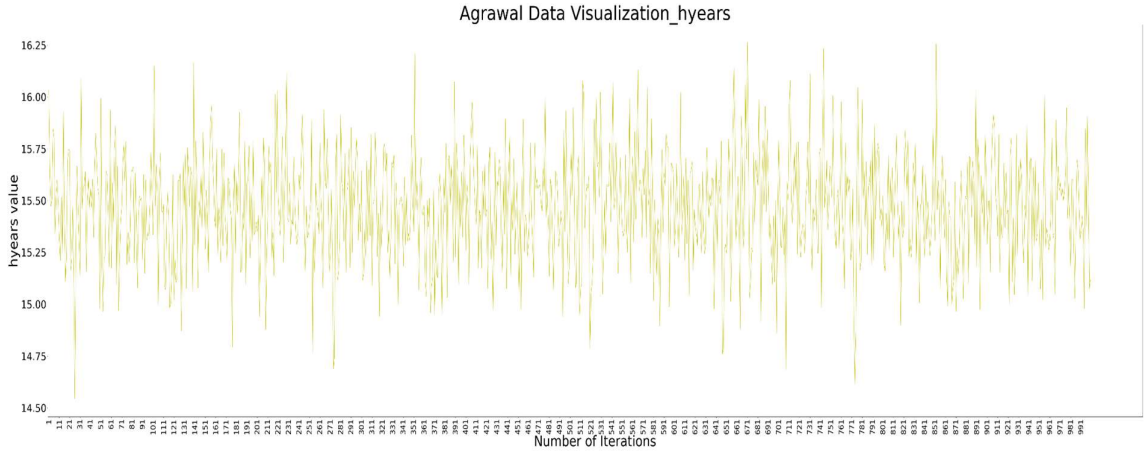


Figure 8: Data trend for feature hyears of Agrawal dataset.

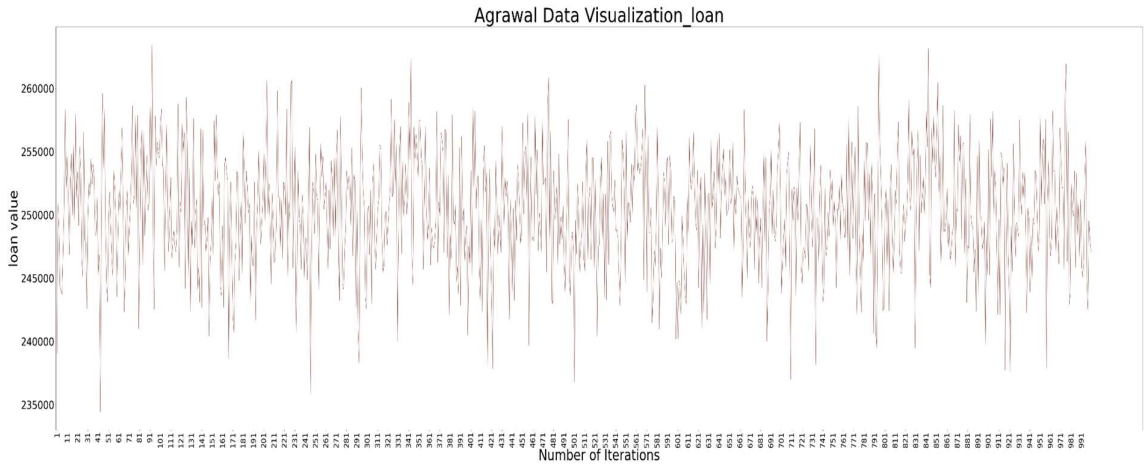


Figure 9: Data trend for feature loan of Agrawal dataset.

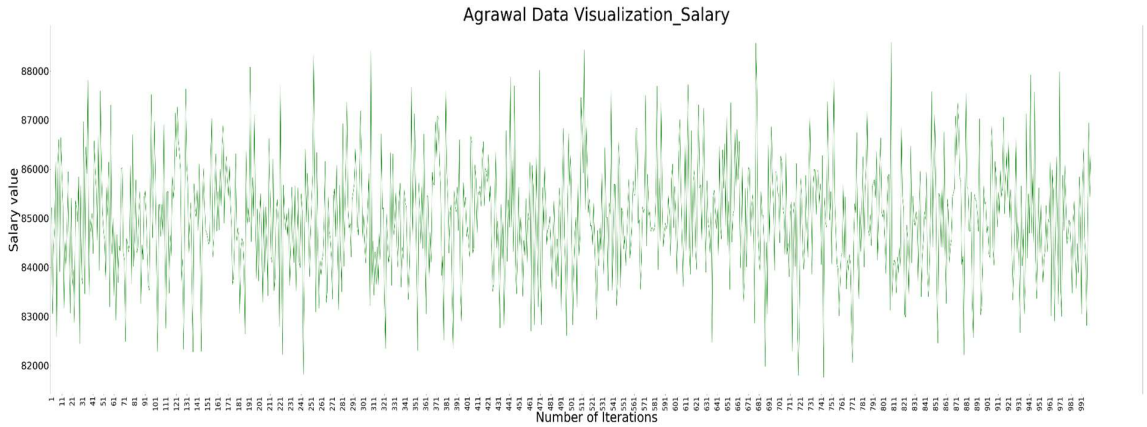


Figure 10: Data trend for feature salary of Agrawal dataset.

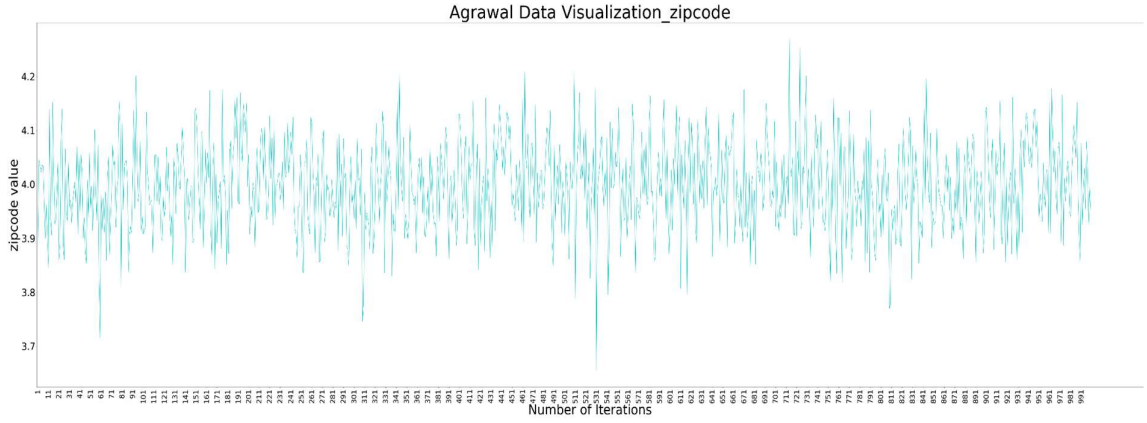


Figure 11: Data trend for feature zipode of Agrawal dataset.

2.1. Scaler and Classifiers

The data provided are for classification problems so the machine learning model has classifiers and some scalers to preprocess and normalize the data. In the Sine dataset we have feature_0 and feature_1, while in the Agrawal dataset we have age, car, commission, elevel, hvalue, hyears, loan, zipcode and salary as features. In both cases, the classification target is either 0 or 1. The type of scalers used are as follows:

2.1.1. Minmax scaler

A prominent data normalization method that is used in machine learning to scale a dataset's values to a specified range between 0 and 1. The MinMaxScaler formula is the following:

$$X_{normalized} = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

where X is the original feature value, X_{min} is the feature's minimum and $X_{max}$is the feature's maximum value [4].

When the range of feature values varies greatly, the MinMaxScaler is frequently used. By minimizing the impact of outliers in the dataset, it can assist machine learning algorithms perform better.

2.1.2. Maxabs scaler

A data normalization method that scales a dataset's feature values to a range between -1 and 1. Here is the MaxAbsScaler formula:

$$X_{normalized} = \frac{X}{max(|X|)}$$

where X is the value of the original feature and $max(|X|)$ is the feature's largest absolute value found in the dataset [16].

It is helpful when the dataset contains features with a mix of positive and negative values. It does not alter the sign or the relationship between the feature values; it just scales them so that they are all between -1 and 1. By reducing the influence of outliers and features with a wide range of values, MaxAbsScaler can aid in enhancing the performance of machine learning algorithms.

2.1.3. Standard scaler

A common data normalization method that scales the data to have a mean of 0 and a standard deviation of 1 (unit variance) [4]. The following formula is for the StandardScaler:

$$X_{normalized} = \frac{(X - mean(X)}{std(X)}$$

where X is the original feature value, $mean(X)$ is the feature's average value throughout the dataset, and $std(X)$ is the feature's standard deviation.

When the features in the dataset have multiple sizes and their distribution is roughly Gaussian (i.e., normal distribution), the StandardScaler is an effective approach [17]. The StandardScaler converts the feature's data into a standardized distribution that is simpler for machine learning algorithms to understand by scaling the features to have a mean of 0 and a standard deviation of 1.

The type of classifiers used are as follows:

2.1.4. Hoeffding tree classifier

It is an incremental learning algorithm. Hoeffding's inequality, a mathematical concept that enables the estimation of the probability distribution of a random variable, serves as the foundation for this algorithm.

In applications where the distribution of the data may change over time, the Hoeffding classifier is especially helpful. It is made for analyzing massive amounts of streaming data. Based on the incoming data stream, a decision tree is built, with each node representing a test on a different feature of the data.

The Hoeffding classifier is based on the idea that by just looking at a small subset of the data, it is possible to predict the classification accuracy with a high degree of confidence. The Hoeffding classifier dynamically changes the decision tree to take new information into account and modify its predictions as fresh data is received.

When compared to other machine learning methods, the Hoeffding classifier has the benefit of requiring extremely minimal memory and computational power. It is robust to noisy data and outliers and can handle data with a lot of features [18].

2.1.5. Gaussian Naive Bayes (GNB)

A straightforward probabilistic classifier that is frequently used in machine learning for classification problems is called Gaussian Naive Bayes (GNB). The Bayes' theorem, which asserts that the probability of a hypothesis (such as a classification label) can be revised based on additional information, forms the basis of the method (such as features of the data).

In Gaussian Naive Bayes, the probability of each class is determined given the observed features under the assumption that the features are normally distributed. Each feature's mean and variance are estimated using the algorithm, which then utilizes those values to determine the class probabilities for each feature. Because it assumes that the features are independent of one another given the class name, GNB is a "naive" algorithm. This assumption makes the computations easier and increases the computational efficiency of the method. However, as the features are frequently connected with one another in practice, this assumption might rarely hold in real scenarios [7,19].

2.1.6. Logistic regression

A statistical technique called logistic regression is used for binary classification, or for predicting one of two possible outcomes, often denoted as 0 or 1. Because it is comparatively straightforward, understandable, and effective, logistic regression is a widely used technique in machine learning.

The likelihood of the first positive outcome (i.e., outcome 1) as a function of the predictor variables (i.e., the features of the data) is modeled using the logistic regression algorithm. The logistic function, which converts any input to a value between 0 and 1, is commonly used to model this function.

The logistic regression algorithm uses a technique called maximum likelihood estimation to estimate the logistic function's parameters based on the training data. By determining the probability of a positive result given the values of the predictor variables, the logistic regression model may be used to make predictions on new data once the parameters have been determined.

Providing a measure of prediction uncertainty is one of the benefits of logistic regression, which can be helpful in a variety of applications. Additionally, by utilizing techniques like one-vs-all or softmax regression, logistic regression can be expanded to handle multiclass classification tasks [20,21,22].

2.1.7. KNN classifier

In machine learning, the KNN (K-Nearest Neighbors) classifier is a simple, non-parametric technique used for classification and regression problems.The classifier's predictions relies on the closest training examples in the feature space since it uses an instance-based learning technique.

Given a new data point, the KNN method determines the k nearest neighbors in the training dataset using a distance measure like Euclidean distance. The new data point's class designation is then decided by a majority voting by the k closest neighbors [23].

2.2. Model Distribution in Sine Dataset

Each model is a combination of a classifier (Hoeffding Tree, KNN, Logistic regression, Gaussian NB) and a scaler (Minmax, Maxabs, Standard). Therefore we have 4*3 = 12 possible model configurations [1].

All the algorithms can be parameterized with hyperparameters, which influence the algorithm's behavior. For instance, the KNN classifier can be parameterized by the number of neighbors, or the Hoeffding Tree (HT) classifier by its maximum depth or the tie threshold, as well as the binary parameters if a binary split technique should be used or if unfavorable attributes should be eliminated. Thus the domain space has 174 different algorithm or model configurations overall when all the hyperparameters combinations are considered [1].

On analyzing the Sine dataset case, we can see it uses nine algorithm configurations or models in the whole process. The sampling rate f_{ss}= 1000 and the

population size P=10 which is equal to the size of the ensemble learners. So there are 1000 iterations each with 10 different ensembles. Below are the unique set of algorithm configurations or models used without considering the hyperparameters [1].

1. Standard Scaler KNN Classifier.
2. MaxAbs Scaler GaussianNB.
3. MinMax Scaler Hoeffding Tree Classifier.
4. MaxAbs Scaler KNN Classifier.
5. MaxAbs Scaler Logistic Regression.
6. MaxAbs Scaler Hoeffding Tree Classifier.
7. MinMax Scaler GaussianNB.
8. MinMax Scaler KNN Classifier.
9. Standard Scaler Hoeffding Tree Classifier.

The idea is to find the most popular model among the nine models used. The number of times each model occurs in a single iteration along 1000 population steps is plotted below.

2.2.1. Standard Scaler KNN Classifier

This model (Figure 12) appears only in the first 20 iterations and it is never used in later iterations. The maximum number of times used in a single iteration is 2 (excluding the hyperparameters) in the whole process.

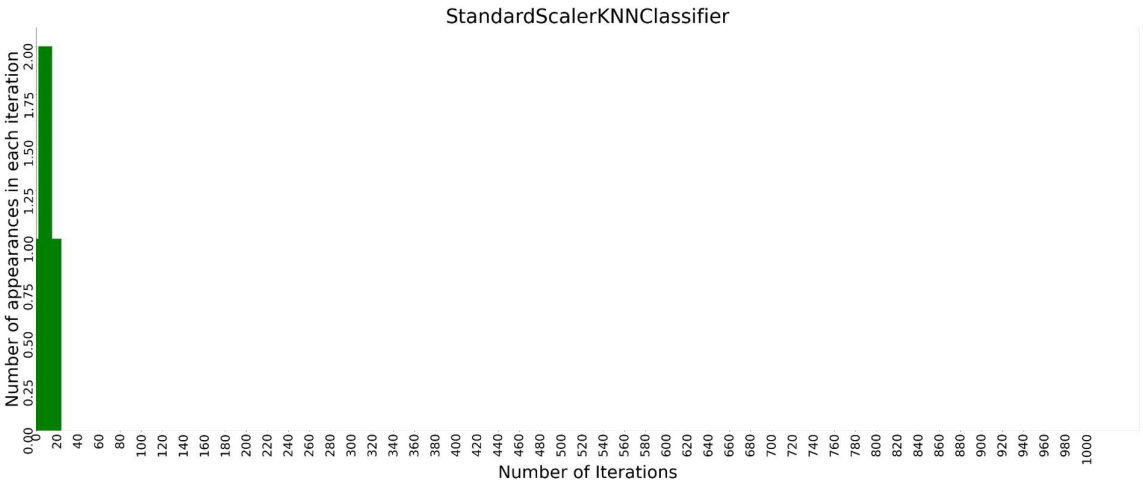


Figure 12: Usage of Standard Scaler KNN Classifier model throughout the process

2.2.2. MaxAbs Scaler GaussianNB

This model (Figure 13) randomly appears only once per iteration. It appears 4 to 5 times in the whole population steps.

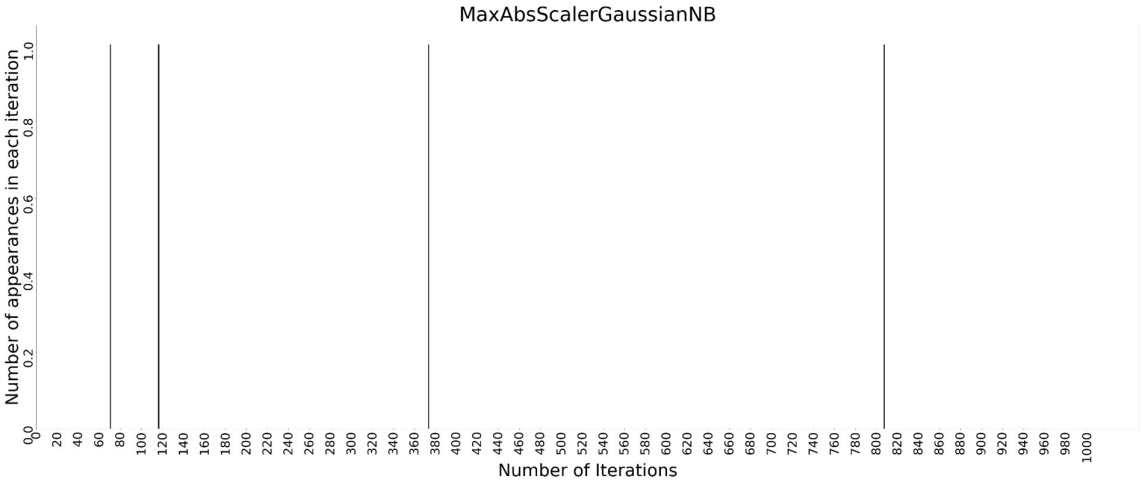


Figure 13: Usage of MaxAbs Scaler Gaussian NB model throughout the process.

2.2.3. MinMax Scaler Hoeffding Tree Classifier

The maximum number of times this model (Figure 14) appears in a single ensemble is five; it's used at least once in the ensemble for most iterations. However it is never used in a few iterations in between.

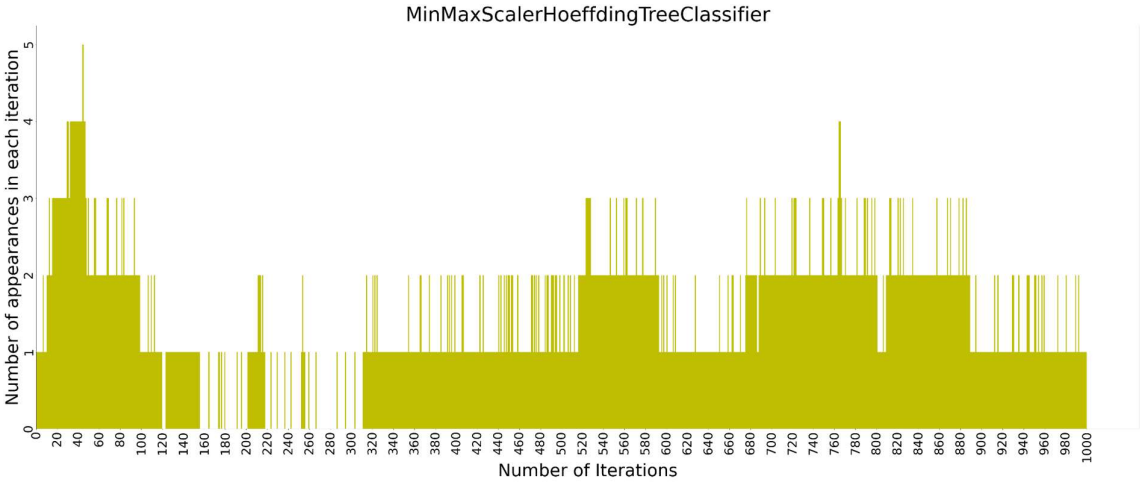


Figure 14: Usage of Minmax Scaler Hoeffding Tree Classifier model throughout the process.

2.2.4. MaxAbs Scaler KNN Classifier

This model (Figure 15) appears utmost once per each iteration. It is not even consistent throughout the iterations.

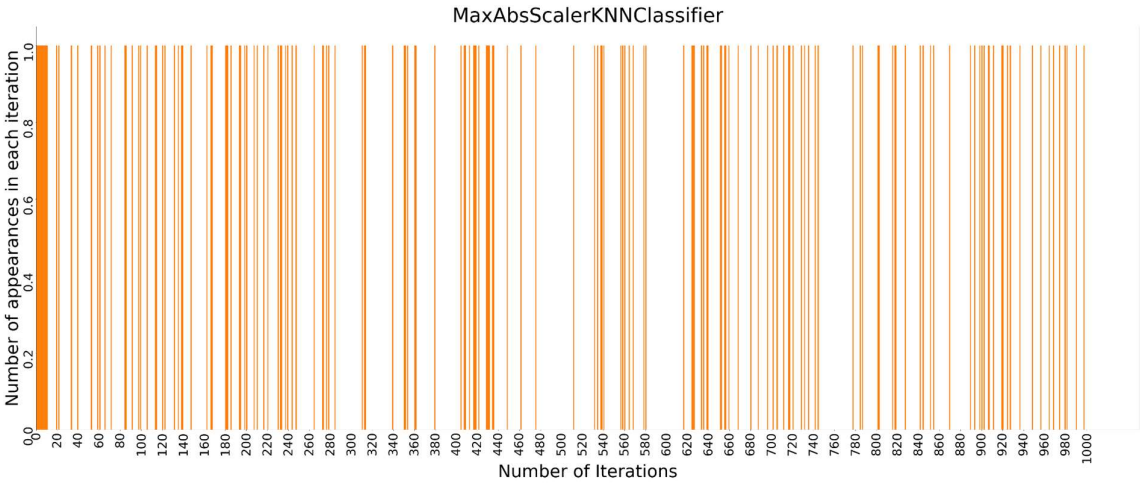


Figure 15: Usage of MaxAbs Scaler KNN Classifier model throughout the process.

2.2.5. MaxAbs Scaler Logistic Regression

This model (Figure 16) appears randomly only once per iteration and only very few times in the whole process. It doesn't seem to be a popular choice among the nine models used, since it is not consistently selected. However, it does appear in a few iterations, even though very randomly.

Figure 16: Usage of MaxAbs Scaler Logistic Regression model throughout the process

2.2.6. MaxAbs Scaler Hoeffding Tree Classifier

It is the most popular model (Figure 17) among all the models. It appears in almost all of the iterations and also it is selected more than 3 times in a single ensemble (probably with different hyperparameter settings). The highest number of appearances in a single ensemble is 10. Most of the iterations have 9 or 10 selections per ensemble. This indicates that the MaxAbs Scaler Hoeffding Tree Classifier is a highly effective model for the sine dataset. The consistent usage of this model across a

large number of iterations also shows that it is reliable for the problem under consideration. On the whole, the popularity of this model and its frequent selection in ensembles makes it an excellent choice for the most effective model in this study.

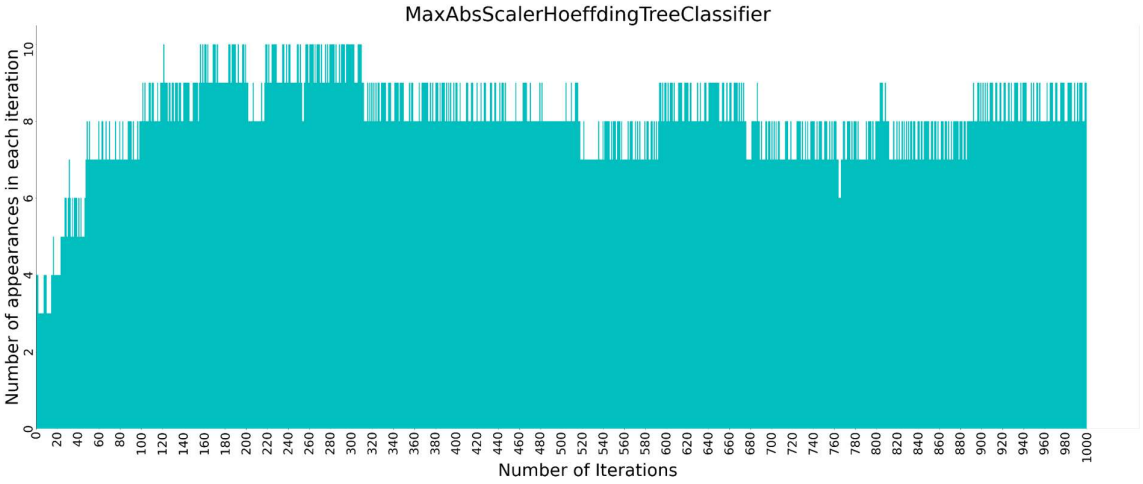


Figure 17: Usage of MaxAbs Scaler Hoeffding Tree Classifier model throughout the process.

2.2.7. MinMax Scaler GaussianNB

In comparison to the other models, the MinMax Scaler GaussianNB model (Figure 18) is a little bit of an outlier. It only appears once, in a single iteration in the whole process. This indicates that this model is particularly not useful for this dataset.

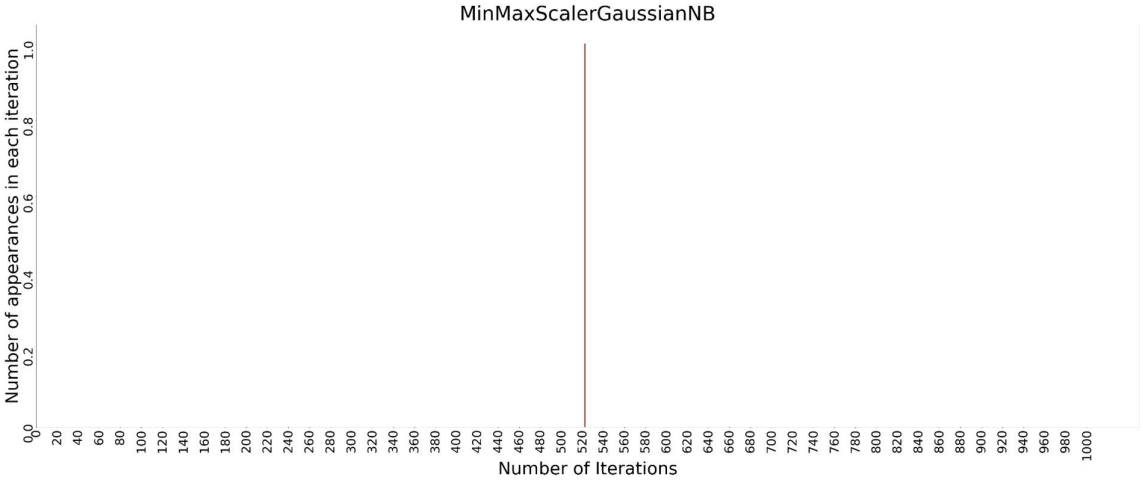


Figure 18: Usage of MinMax Scaler Gaussian NB model throughout the process.

2.2.8. MinMax Scaler KNN Classifier

This model (Figure 19) is evident in the early iterations consistently, where it is selected once or twice in a single ensemble. After it only appears at a few iterations,

notably in the middle and once towards the conclusion. It is not a constant model throughout the whole process.

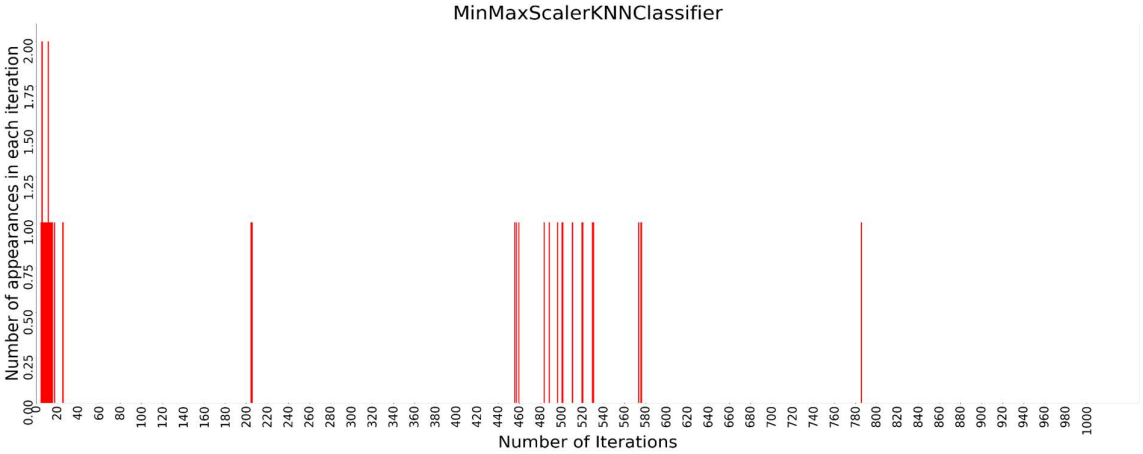


Figure 19: Usage of MinMax Scaler KNN Classifier model throughout the process.

2.2.9. Standard Scaler Hoeffding Tree Classifier

This model (Figure 20) is selected in every ensemble until somewhere before the 30th iteration appearing at least once and sometimes twice or thrice, suggesting that it is a common choice in the early iterations. As in the later iterations, it loses consistency, and in some iterations, this model is completely absent. Nevertheless, it continues to recur again throughout the process, demonstrating that it is still regarded as a viable option in some circumstances.

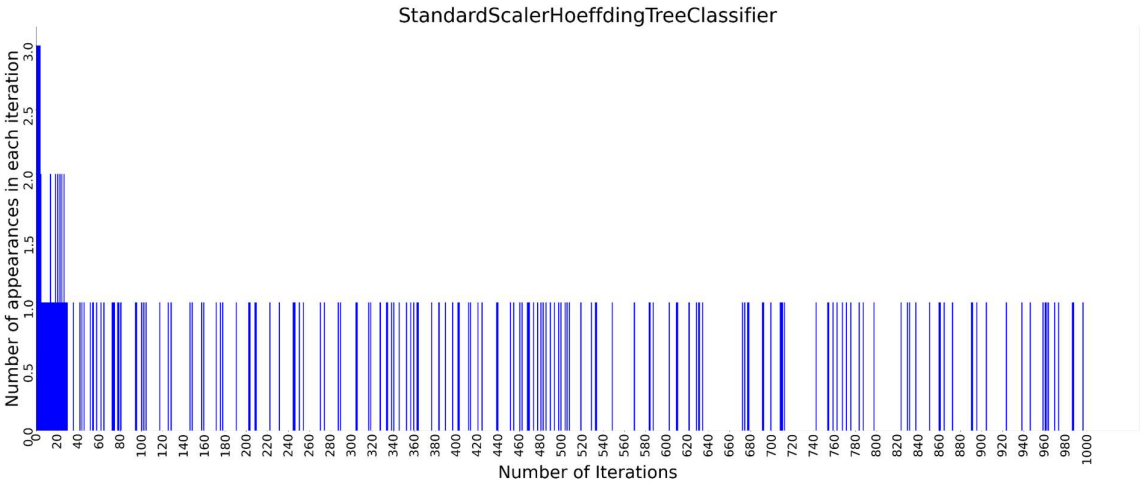


Figure 20: Usage of Standard Scaler Hoeffding Tree Classifier model throughout the process.

2.2.10. Analysis on Distribution of Hyperparameter Values

Hyperparameters are parameters that are set before a model is trained; and are not determined by the data. They may significantly affect how well a model performs. The following table displays the values selected for each hyperparameter.

Classifier	**Hyperparameter**	**Hyperparameter value**
Hoeffding Tree Classifier	Maximum depth	10, 30, 60
Hoeffding Tree Classifier	Grace period	10, 100, 200
Hoeffding Tree Classifier	Maximum size	5, 10
Logistic Regression	L2	0.000, 0.001, 0.010
KNN Classifier	Number of neighbors	1, 5, 20
KNN Classifier	Window size	100, 500, 1000
KNN Classifier	Weighted	1- True 0- False
KNN Classifier	P	1- Euclidean distance 2- Manhattan distance

Table 1: Values for different hyperparameters used.

2.2.10.1. Hyperparameters for Hoeffding Tree Classifier

Maximum depth, grace period, and maximum size are the three hyperparameters for the Hoeffding tree classifier. The following plots (Figure 21 to Figure 28) display which hyperparameter value is chosen for each iteration of the model's training.

The size of the marker is proportional to the number of times the model used the parameter in the particular iteration. Larger the marker size model chooses it very often and smaller the marker size very few selections by the model in that iteration.

A maximum depth of 10 seems to have been the most often chosen value. The most often chosen values for grace period and maximum size, respectively, were 10 and 5. The value that is chosen more often indicates that this value provided the best performance on the data.

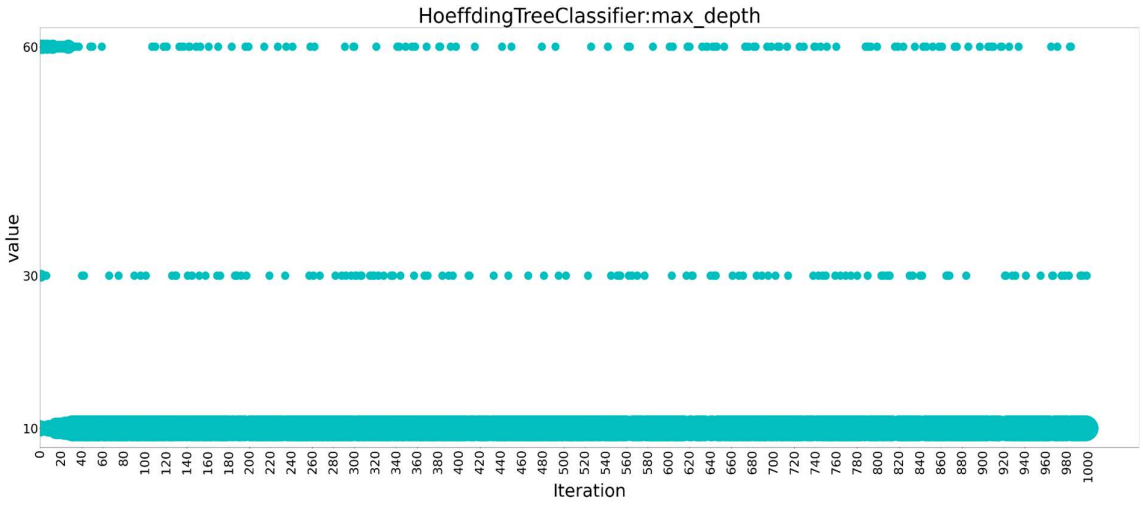


Figure 21: Hyperparameter maximum depth values for Hoeffding Tree of Sine Dataset.

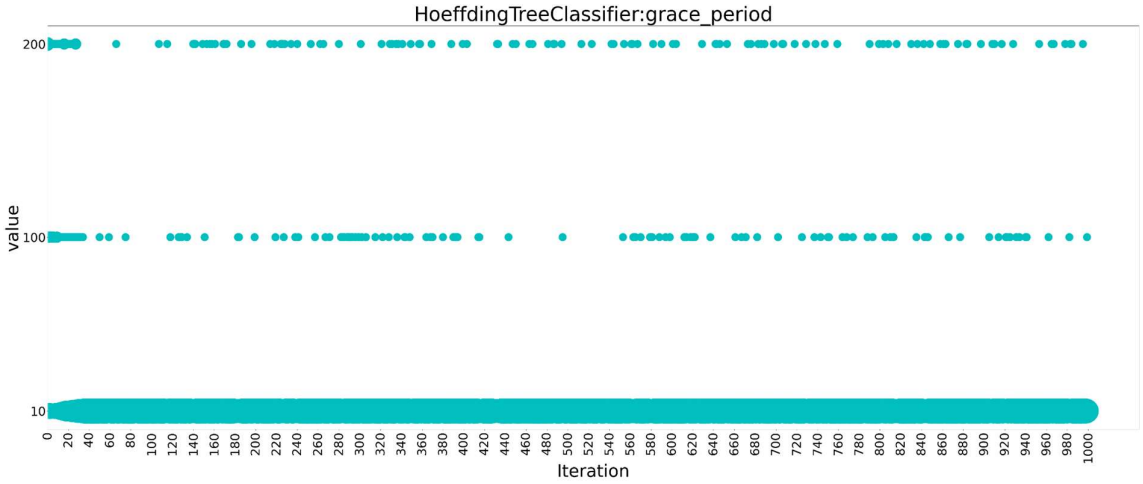


Figure 22: Hyperparameter grace period values for Hoeffding Tree of Sine Dataset.

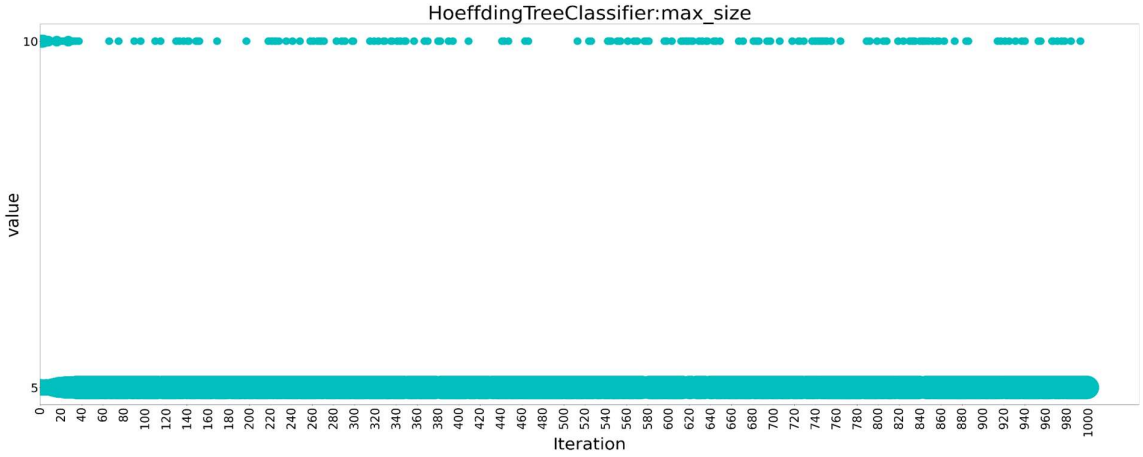


Figure 23: Hyperparameter maximum size values for Hoeffding Tree of Sine Dataset.

2.2.10.2. Hyperparameter for Logistic Regression

The only hyperparameter in the logistic regression model is L2 regularization. The plots show that no particular value of L2 regularization consistently performed better than other values.

Figure 24: Hyperparameter L2 values for Logistic Regression of Sine Dataset

2.2.10.3. Hyperparameters for KNN Classifier

Eventually, for the KNN classifier, there are four hyperparameters: number of neighbors, window size, weighted, and p (See Table 2.2.1 for details).

However, like the Hoeffding tree classifier, none of the hyperparameters appear to be particularly desirable, indicating that their values may not significantly influence the performance for both the logistic regression and KNN classifier.

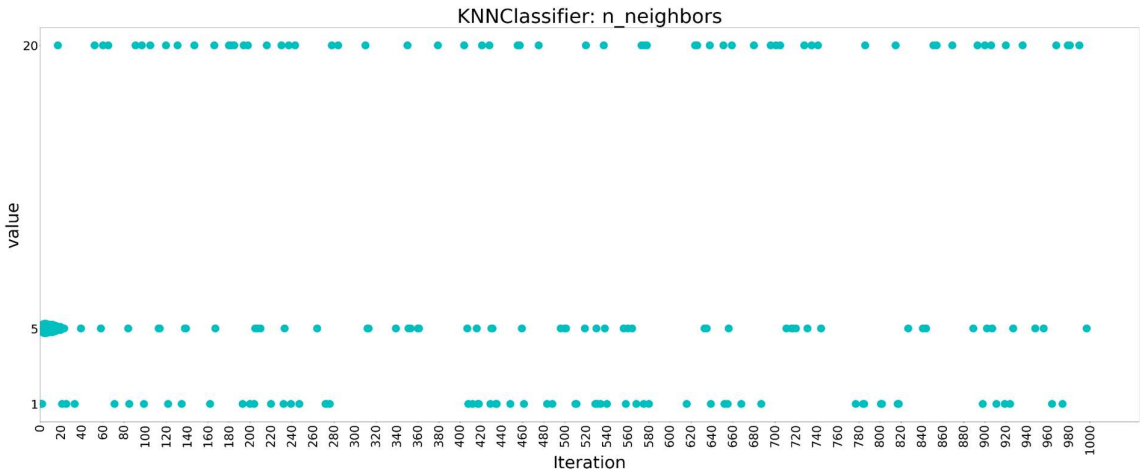


Figure 25: Hyperparameter n_neighbors values for KNN Classifier of Sine Dataset.

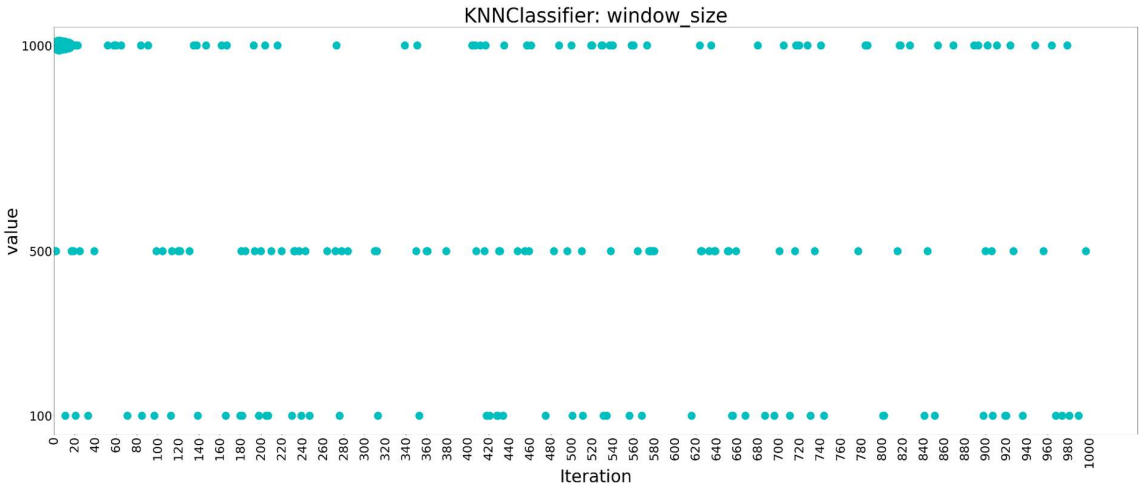


Figure 26: Hyperparameter window size values for KNN Classifier of Sine Dataset.

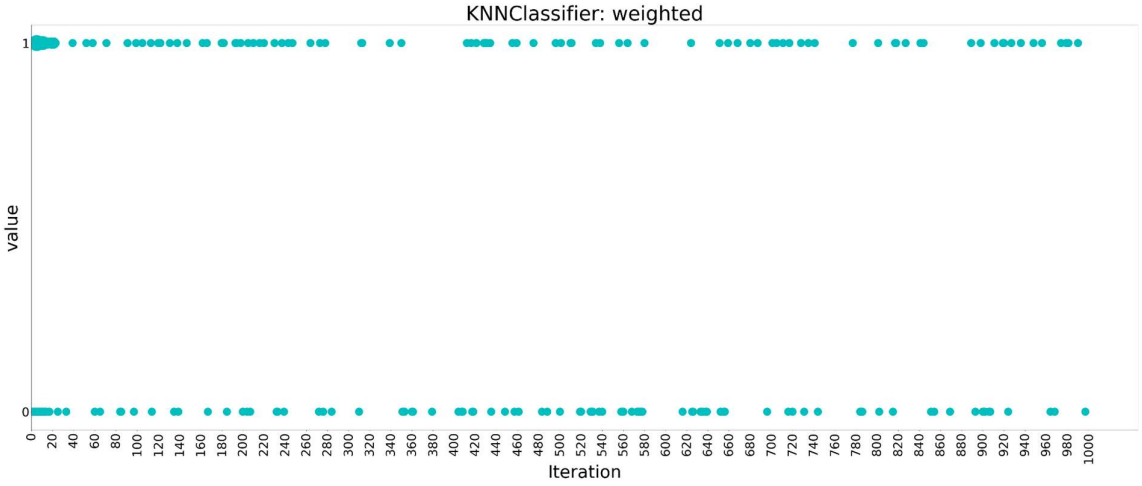


Figure 27: Hyperparameter weighted values for KNN Classifier of Sine Dataset.

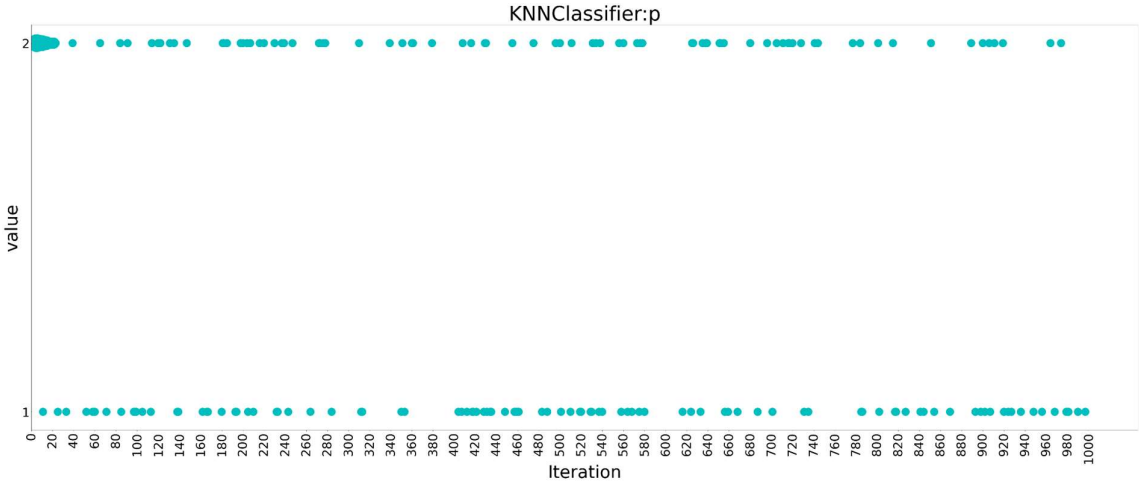


Figure 28: Hyperparameter p values for KNN Classifier of Sine Dataset.

2.3. Model Distribution in Agrawal Dataset

Now, let us consider the Agrawal dataset which has more features and high complexity with the same sampling frequency f_{ss}= 1000 and the population size P=10. There are 1000 iterations each with 10 different ensembles. The unique set of algorithm configurations or models used by Agrawal dataset without considering the hyperparameters are as follows [1].

1. MaxAbs Scaler Hoeffding Tree Classifier.
2. Standard Scale Hoeffding Tree Classifier.
3. Standard Scaler Logistic Regression.
4. MinMax Scaler KNN Classifier.
5. Standard Scaler KNN Classifier.
6. MinMax Scaler Hoeffding Tree Classifier.
7. MinMax Scaler Logistic Regression.
8. MaxAbs Scaler KNN Classifier
9. MinMax Scaler Gaussian NB.

The idea is to find the most popular model among the nine models used. The number of times each model occurs in a single iteration along 1000 population steps is plotted below.

2.3.1. MaxAbs Scaler Hoeffding Tree Classifier

This model (Figure 29) appears frequently in many iterations and is typically chosen at least once per ensemble. Until the earlier iterations of 300s, it consistently appears, at least one option per ensemble and occasionally even twice. It occurs in a single ensemble at least twice in the first 60 iterations, and occasionally even thrice. However, in some iterations, it does not appear at all. The fact that it keeps being chosen repeatedly shows that it is still regarded as a possible option in some situations.

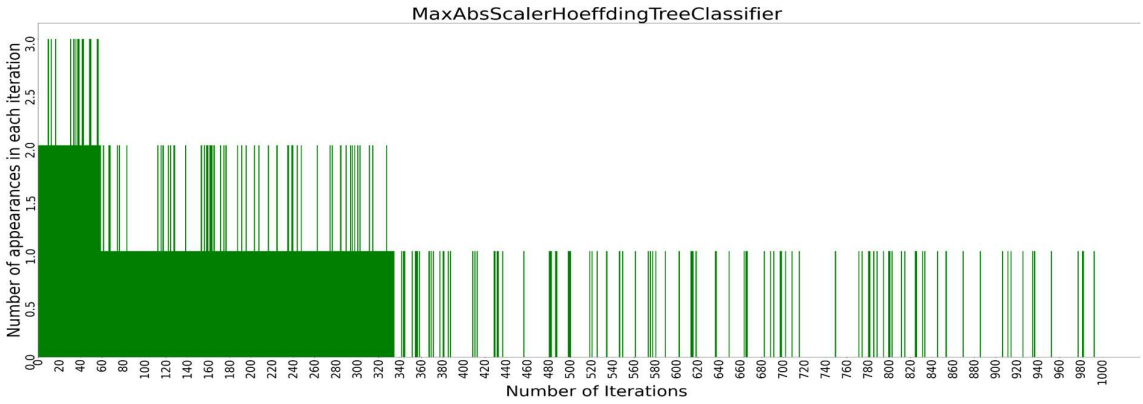


Figure 29: Usage of MaxAbs Scaler Hoeffding Tree Classifier throughout the process.

2.3.2. Standard Scaler Hoeffding Tree Classifier

Almost all iterations use this model (Figure 30) makes it a popular model. In all iterations, it occurs at least once per ensemble, and occasionally twice. It appears once or twice per ensemble in the early 300 iterations, and from the early 300 to the middle of the 400 iterations, it continues to appear with an increment by one time per ensemble until it reaches nine times. It then occurs almost nine or ten more times per ensemble (excluding hyperparameters).

As a result, it can be inferred that the Standard Scaler Hoeffding Tree Classifier is a reliable and effective model that is constantly applied throughout the whole process. Its ability to process big datasets with high-dimensional features and the ability to learn incrementally from data streams, and high classification accuracy could be the reasons for its popularity.

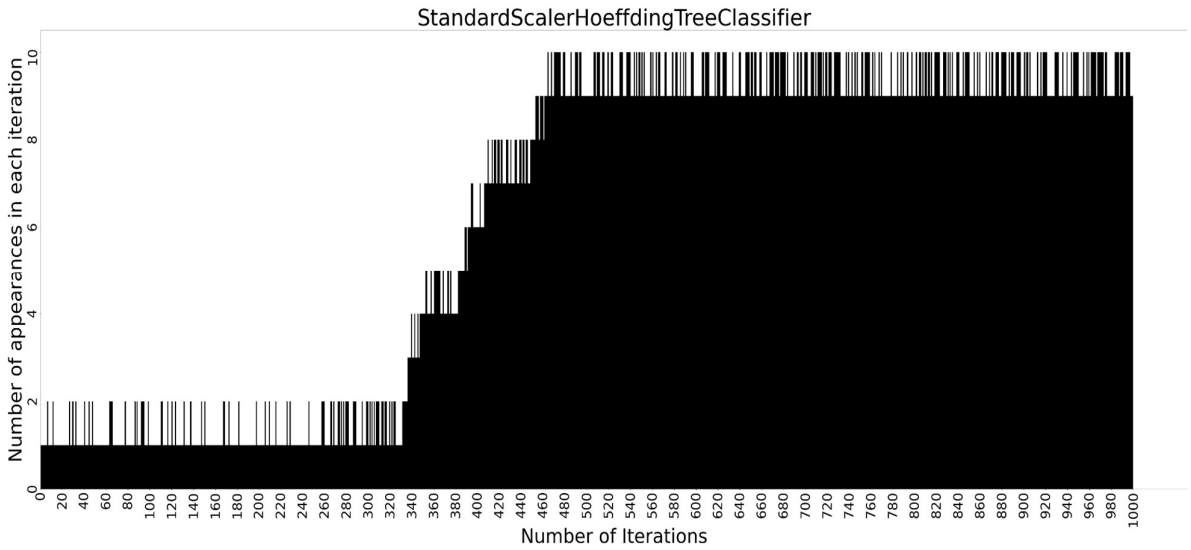


Figure 30: Usage of Standard Scaler Hoeffding Tree Classifier throughout the process.

2.3.3. Standard Scaler Logistic Regression

The scattered appearance of this model (Figure 31) throughout the whole process indicates that it is not a frequently used model. It appears only once per ensemble and only once per iteration. In many iterations, it does not appear at all featuring a few random appearances, but only once per ensemble throughout the process.

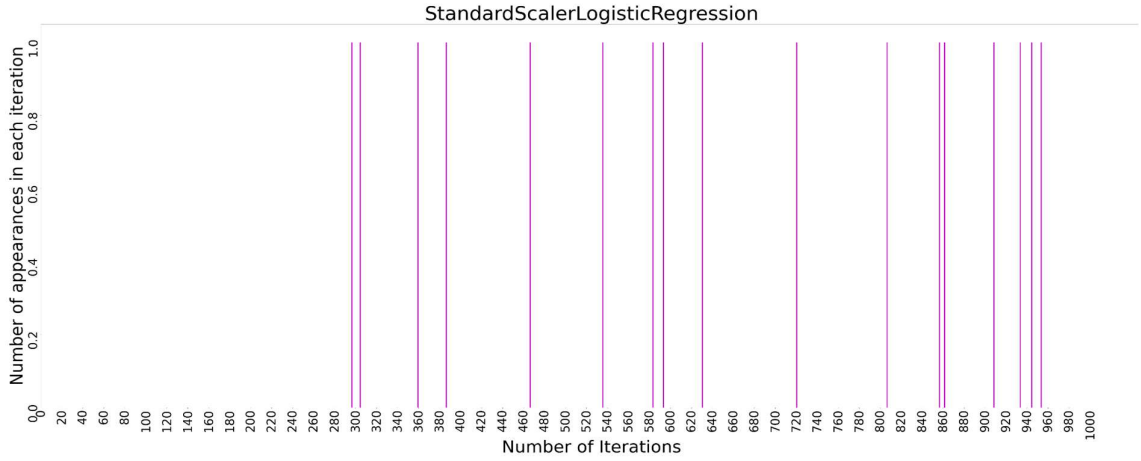


Figure 31: Usage of Standard Scaler Logistic Regression throughout the process.

2.3.4. MinMax Scaler KNN Classifier

This model (Figure 32) shows little consistency throughout the iterations. Until the latter iterations of 200s, it only makes a single appearance per ensemble for only a few iterations. After that the model disappears. It is picked twice for each ensemble in the first five iterations, but neither choice is reliable.

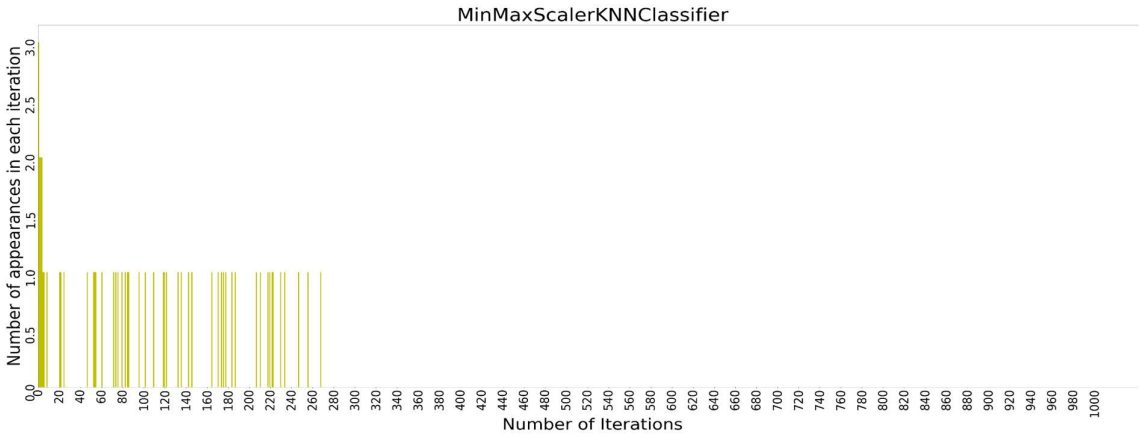


Figure 32: Usage of Minmax Scaler KNN Classifier throughout the process.

2.3.5. Standard Scaler KNN Classifier

This model (Figure 33) consistently appears in the first very few iterations with only one selection per ensemble. However, it is not utilized regularly throughout the latter iterations and only appears once per ensemble after the mid iterations of 200s until which it has no single appearance. It does appear in many iterations despite its seemingly random appearances, suggesting that it is a popular option among the ensembles throughout the whole process.

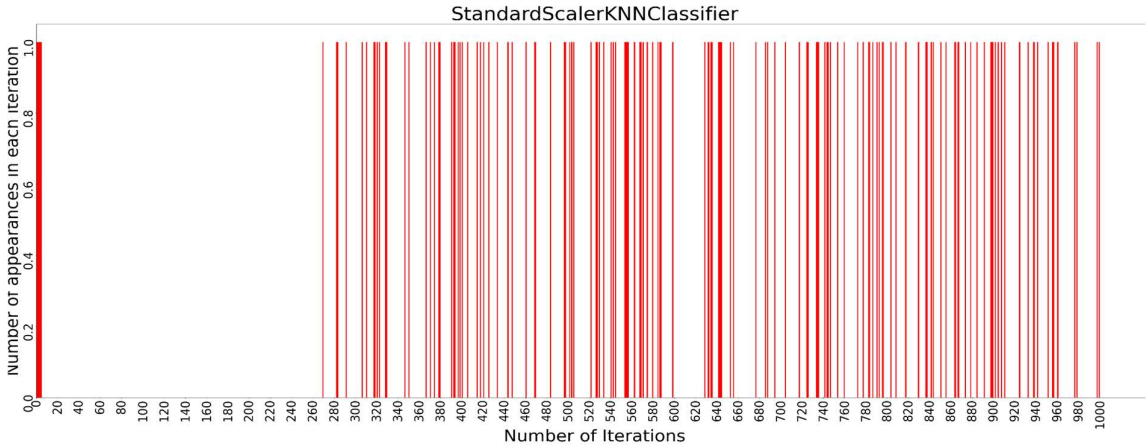


Figure 33: Usage of Standard Scaler KNN Classifier throughout the process.

2.3.6. MinMax Scaler Hoeffding Tree Classifier

This model (Figure 34) exhibits a greater frequency of appearances till the early 300 iterations of the process. Typically, 6 to 8 times per ensemble. The frequency of appearance then steadily declines by one until the middle of the 400 iterations. The model then makes random appearances but only once per ensemble in the latter iterations but it is not consistent. However, as the iterations go on, there is a trend shift in the underlying dataset that affects the model's consistency.

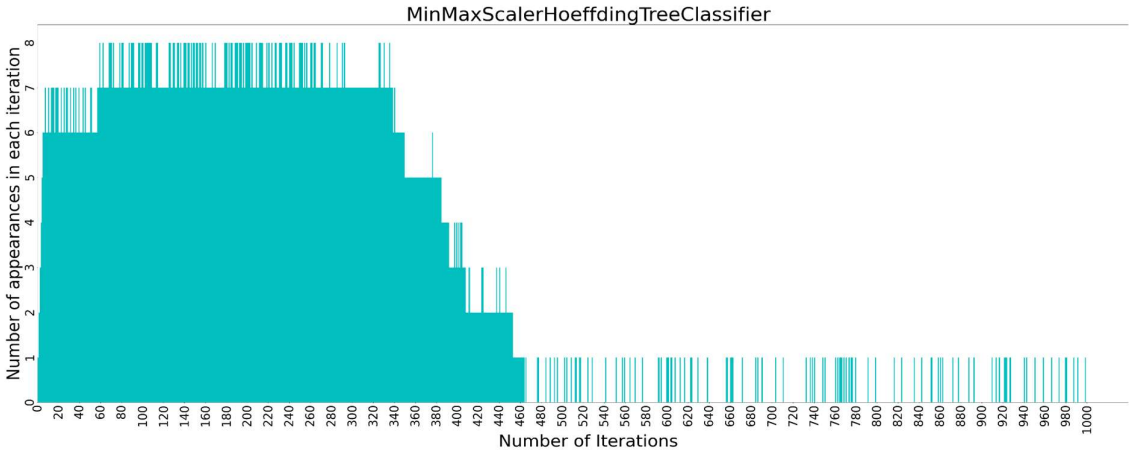


Figure 34: Usage of Minmax Scaler Hoeffding Tree Classifier throughout the process.

2.3.7. MinMax Scaler Logistic Regression

This model (Figure 35) is not a frequently used model in the whole process by appearing only twice in the whole process, and each time, it is selected only once per ensemble. It doesn't appear in any other iteration till the end. This suggests that the MinMax Scaler Logistic Regression model is not a particularly strong or effective

model for this dataset. Overall, the infrequent appearance of this model indicates that it may not contribute to predicting accuracy of the overall ensemble model.

Figure 35: Usage of Minmax Scaler Logistic Regression throughout the process.

2.3.8. MaxAbs Scaler KNN Classifier

This model (Figure 36) appears twice per ensemble in the first two iterations, indicating that it is one of the more popular models at the start of the process. However, its appearance reduces to once per ensemble in the third iteration, and it does not appear again in the latter iterations. This indicates that this model is not appropriate for this dataset, as it is not consistently selected throughout the iterations.

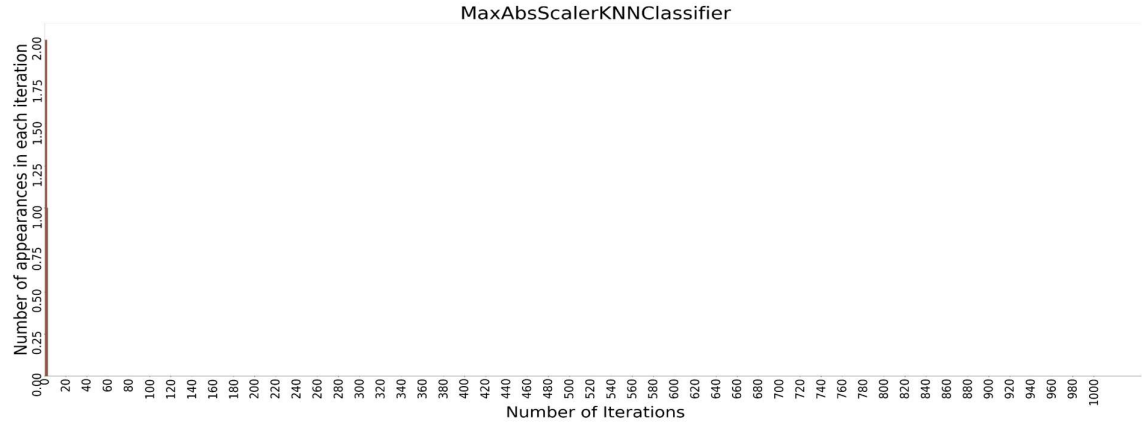


Figure 36: Usage of Maxabs Scaler KNN Classifier throughout the process.

2.3.9. MinMax Scaler Gaussian NB

Throughout the entire process, this model (Figure 37) only makes an appearance once per ensemble, and that too in a single iteration. It doesn't exhibit any recurring patterns or a tendency to be picked once more in consecutive iterations.

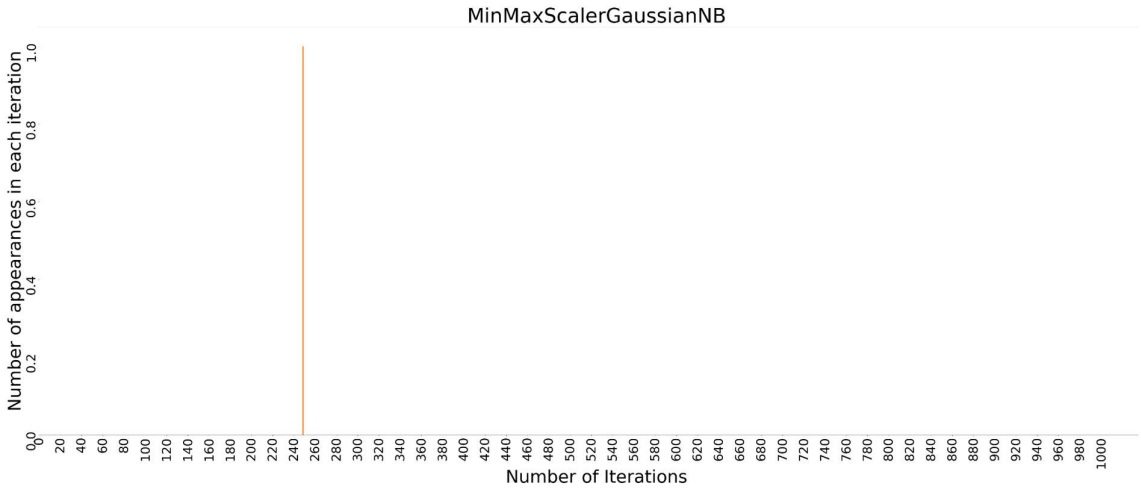


Figure 37: Usage of Minmax Scaler Gaussian NB throughout the process.

2.3.10. Analysis on Distribution of Hyperparameter Values

The plots (Figure 38 to Figure 45) below display the hyperparameter values for each classifier and how well it is used in each iteration.

2.3.10.1. Hyperparameters for Hoeffding Tree Classifier

The maximum depth of the decision tree that a Hoeffding classifier can generate is determined by the hyperparameter "max depth." The success of the classifier may be significantly impacted by this parameter. To see the values for the hyperparameters used (See Table 1 for reference).

In the early stages of the process, the values of 10 and 30 are consistently chosen for the max depth parameter but its popularity decreases in the latter stages. This could indicate that the initial dataset was very small, or that a simple decision tree was sufficient to achieve the desired result. Throughout the process, the "max depth" value of 60 is consistently used. This shows that the classifier has decided that the best decision tree for the task at hand has a depth of 60.

The “grace period” hyperparameter is used to determine the minimum number of instances or leaves that should be observed to gather information about the distribution of data before splitting a leaf node.

In the early stages, values like 100 and 200 are frequently used for the grace period because they offer a better balance between exploration and exploitation. In streaming data environments where the data distribution may change over time, these bigger values enable the algorithm to investigate the data more thoroughly before making a decision. Smaller values, like 10, might, however, be adequate once the algorithm has collected enough knowledge about the data, and the larger values might

not be selected as frequently. Because of this, 10 is frequently selected and gains popularity over time.

The "max size" hyperparameter is used to determine the maximum number of leaf nodes that the tree can have. It is used to control complexity of the model and prevent overfitting. The value 10 is popular in the early stages and has a decreasing trend towards the end indicates that it is used to capture all the complex relationships at the beginning and a value of 5 is more reasonable after that. Lesser the value of max size, lesser the complexity and less prone to overfitting.

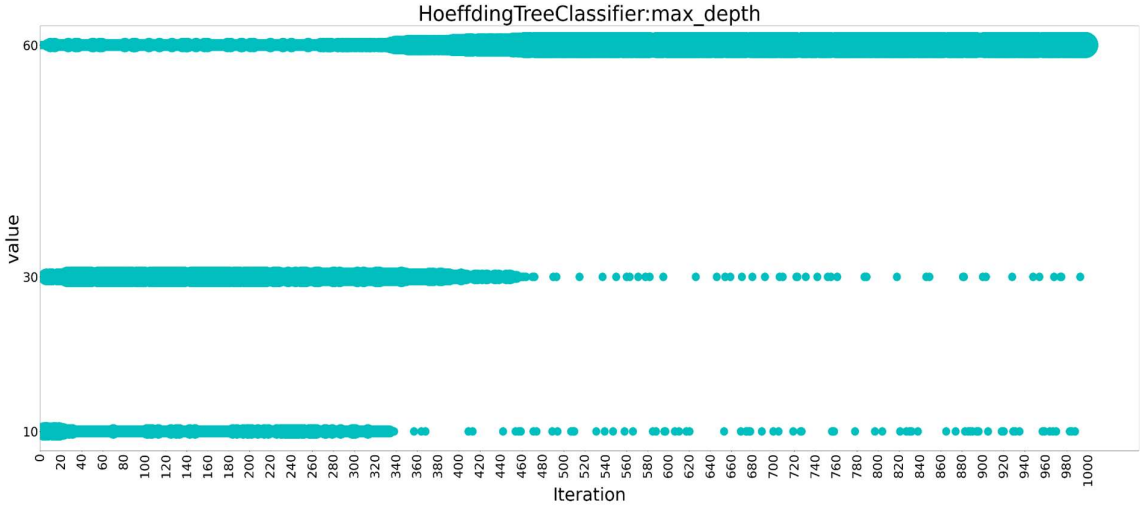


Figure 38: Hyperparameter maximum depth values for Hoeffding Tree of Agrawal Dataset

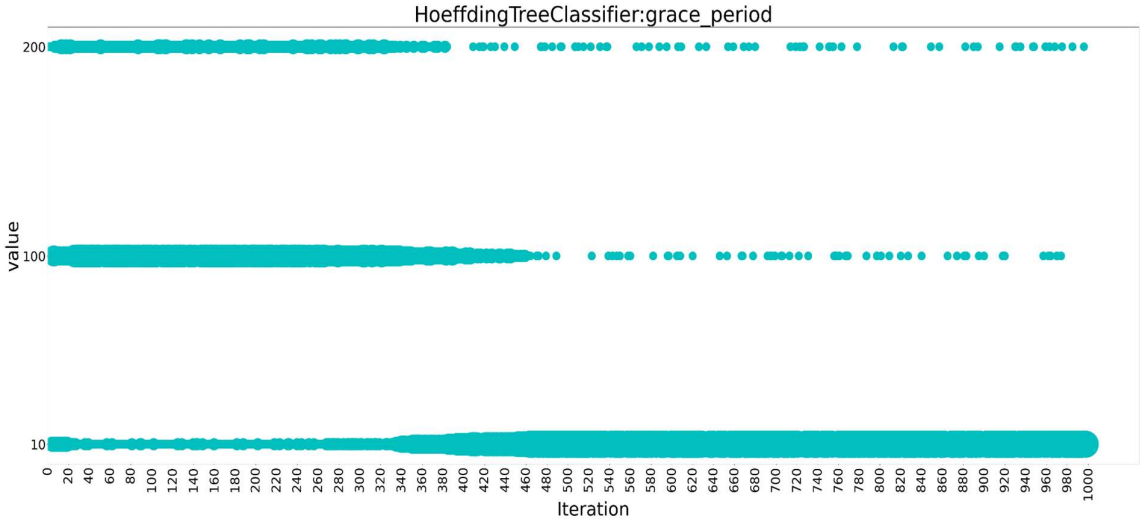


Figure 39: Hyperparameter grace period values for Hoeffding Tree of Agrawal Dataset

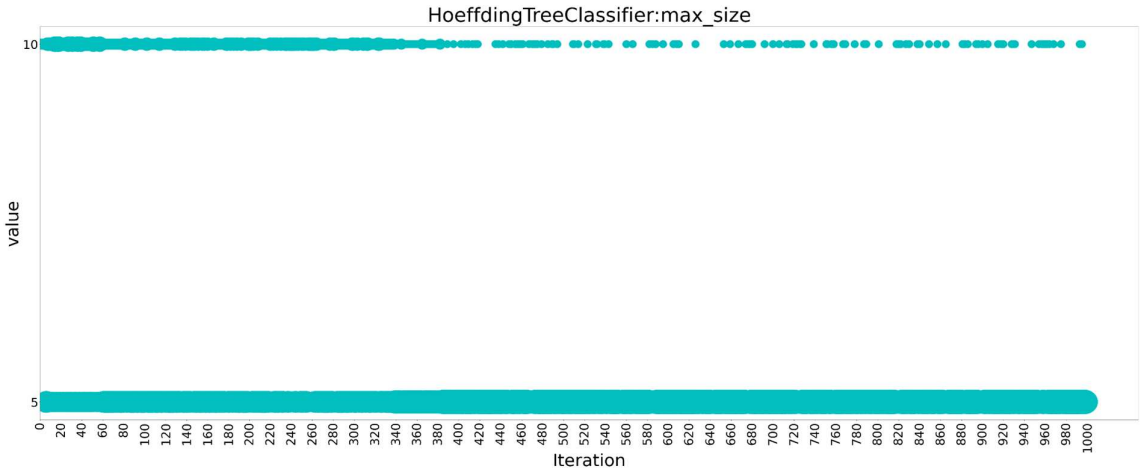


Figure 40: Hyperparameter maximum size values for Hoeffding Tree of Agrawal Dataset.

2.3.10.2. Hyperparameter for Logistic Regression

Logistic regression with L2 regularization where L2 is the regularization parameter. This hyperparameter establishes the trade-off between maintaining small model weights to avoid overfitting and providing good fit for training data.

If the hyperparameter lambda has values of 0.000, 0.001, and 0.010 scattered between values, it has been tested and determined that these values offer adequate performance on the dataset. The larger values of 0.001 and 0.010 exhibit stronger regularization, while the smaller value of 0.000 indicates no regularization at all.

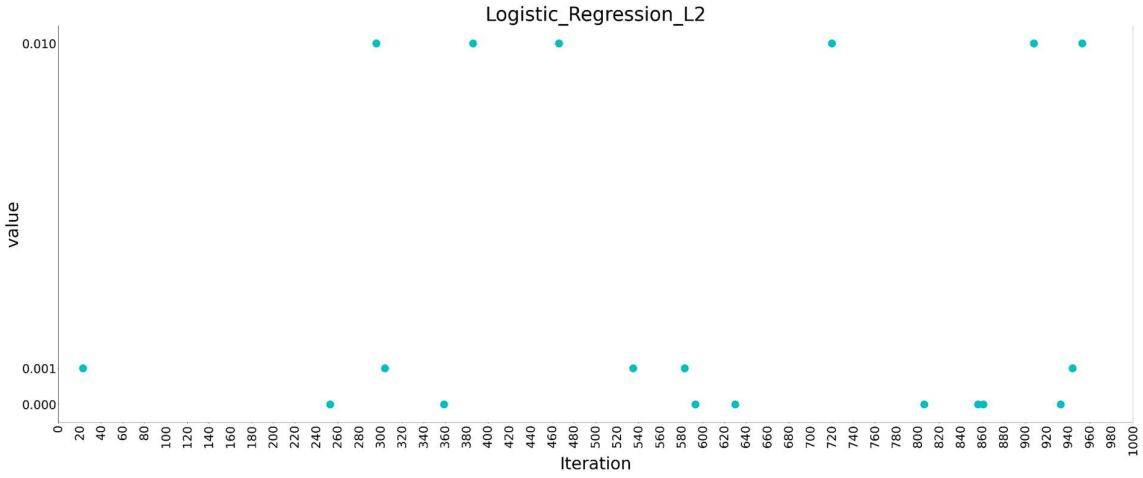


Figure 41: Hyperparameter L2 values for Logistic Regression of Agrawal Dataset.

2.3.10.3.Hyperparameters for KNN Classifier

KNN Classifier has a "n_neighbors" hyperparameter that decides the decision boundary. The values 1, 5 and 20 are not consistent throughout the whole process which indicates that the values are adjusted with the change in data trend.The hyperparameter "window size" specifies the area where the KNN algorithm searches for the nearest neighbors to a given data point. The values 100, 500 and 1000 are chosen randomly till the end indicates there may be a shift in the data trend.

The hyperparameter "weighted" is a boolean flag and has values 1 for True and 0 for False, indicating whether to assign weights to the n_neighbors based on their distance from the new sample that helps in majority voting where the data is distributed. There is always a shift between 0 and 1 that might help the classifier to fit to the trend of the data. The hyperparameter "p" refers to the distance metric used by the classifier to calculate the distance between two samples. There is always a shift between the values 1 and 2 depending on the properties of the data.

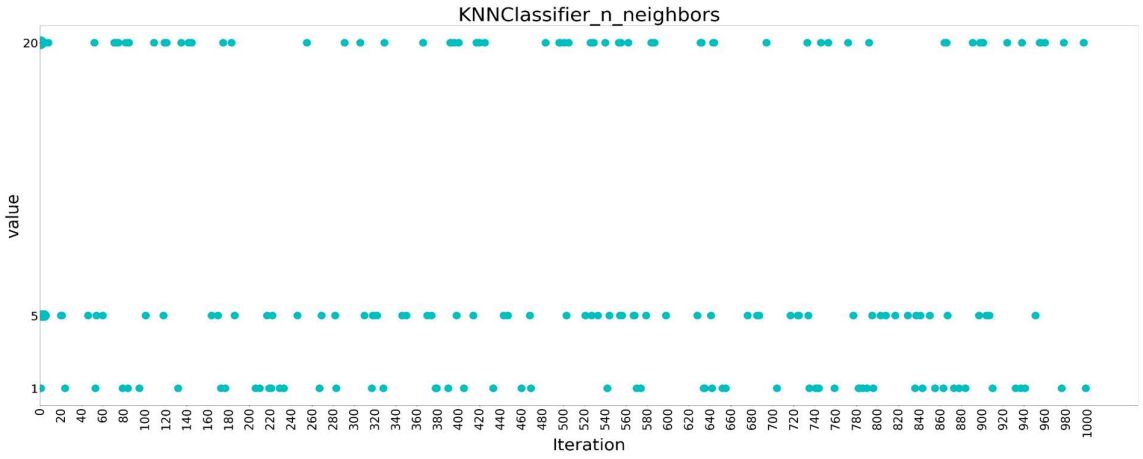


Figure 42: Hyperparameter n_neighbors values for KNN Classifier of Agrawal Dataset.

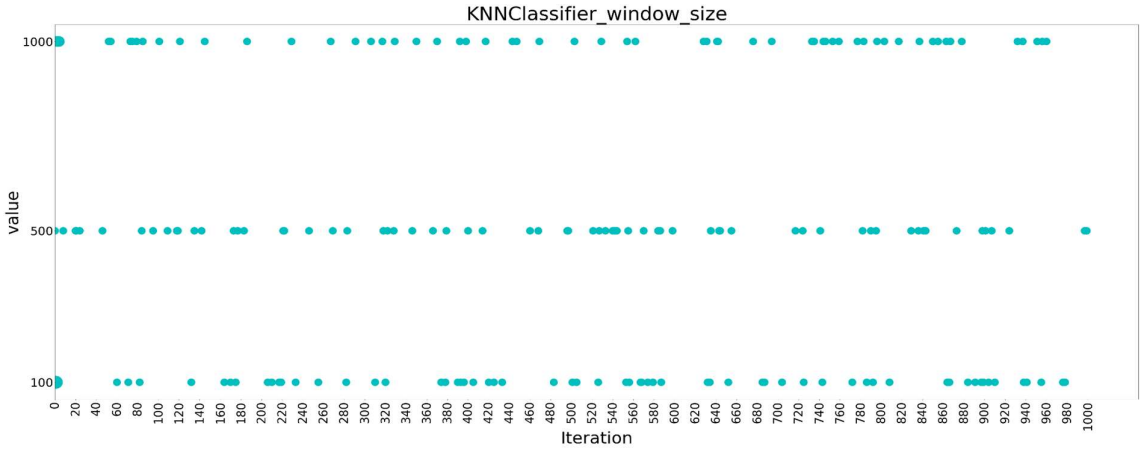


Figure 43: Hyperparameter window size values for KNN Classifier of Agrawal Dataset.

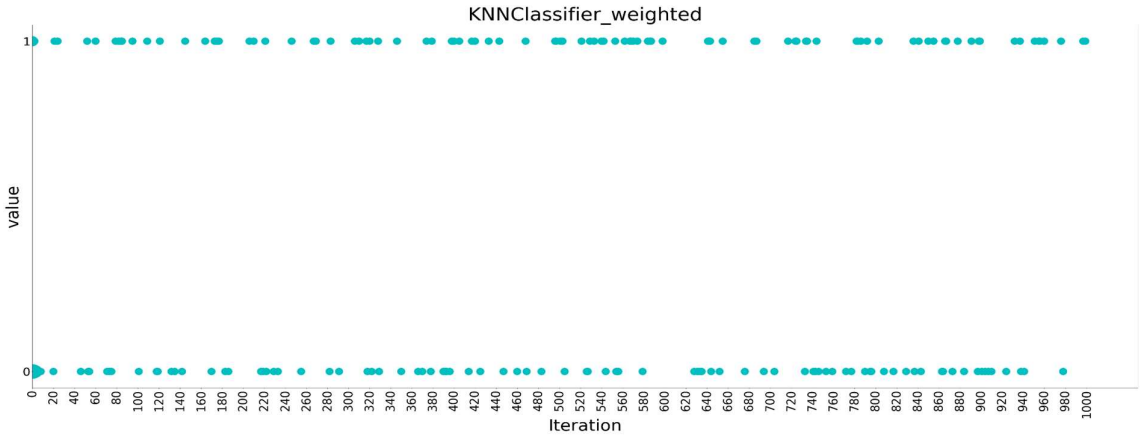


Figure 44: Hyperparameter weighted values for KNN Classifier of Agrawal Dataset.

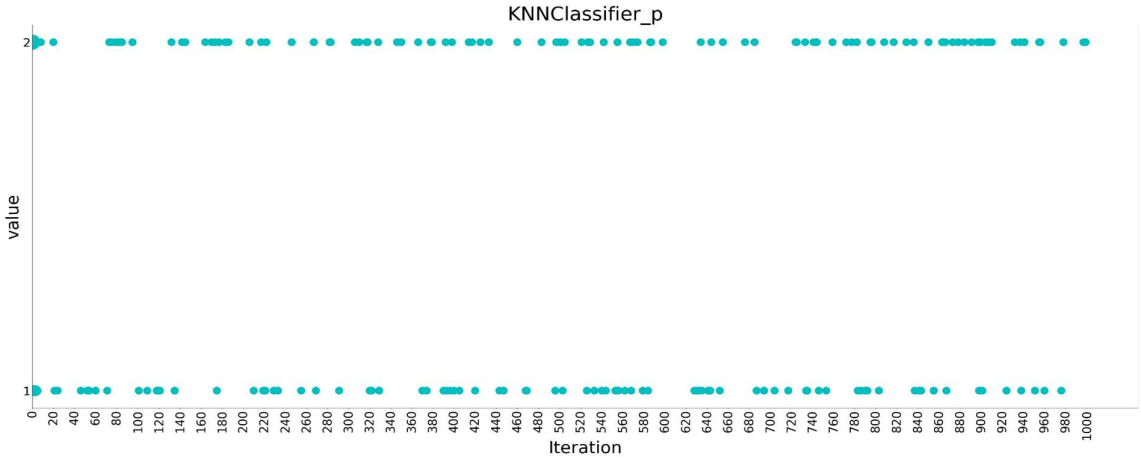


Figure 45: Hyperparameter p values for KNN Classifier of Agrawal Dataset.

3 Accuracy Prediction-From Data to Forecasting Future

3.1. Machine Learning Techniques & Hyperparameters

We use automated machine learning, also known as AutoML, to automate the machine learning pipeline and produce predictions without human intervention [2]. The machine learning technique and hyperparameters to use are based on the particular problem we are trying to solve.

For example, if we are dealing with a regression problem, one of the techniques that we can use is linear regression. To achieve the best results in this case, we will need to tune the regularization parameter and the optimization algorithm. We can use logistic regression, decision trees, random forests, support vector machines (SVM), gradient boosting, or neural networks to solve classification problems. Each of these methods will have its own set of hyperparameters that must be optimized in order to achieve the best results.

We have accuracy of each model in an ensemble for a classification problem evaluated by the EvoAutoML algorithm [1]. In general, a regression problem involves predicting a continuous numerical value as the output, such as predicting the price of a house based on its characteristics. Since accuracy depends on different features, we can consider it as a regression problem. The aMLLibrary (for regression models) [3] provides a variety of machine learning methods and hyperparameters to predict the accuracy of streaming machine learning systems. Let us consider the following machine learning techniques to predict the accuracy of each model given the features:

- Linear Ridge Regression (LRRidge).
- eXtreme Gradient Boosting (XGBoost).
- Decision Tree (DT).
- Random Forest (RF).
- Support Vector Regression (SVR).

3.1.1. Linear Ridge Regression (LRRidge)

It is Linear least squares with L2 regularization where regularization is a technique to prevent overfitting and increases the generalization of the model.

Minimizes the objective function: $||y - X_w||_2^2 + alpha * ||w||_2^2$

This model solves a regression model where the loss function is the linear least squares function and the L2-norm for regularization. It is also known as Tikhonov regularization [4].

The hyperparameter **alpha** is used to control the regularization by adding a penalty term to the loss function. Higher values of the alpha hyperparameter lead to greater regularization, which is how the regularization penalty is controlled. Using the loguniform distribution, which samples values logarithmically between a minimum and maximum value, you can specify the search area for alpha for alpha.

In particular, Hyperopt is a Python library that allows the usage of distributions on hyperparameters, and is used by aMLLibrary for hyperparameter optimization [7]. Hyperopt exploits a form of Bayesian optimization for parameter tuning which provides the best parameters for a given model.eXtreme Gradient Boosting (XGBoost)

It is an efficient and scalable version of the gradient boosting framework [5][6]. The package contains an efficient linear model solver and tree learning algorithm.

The following are the hyperparameters of XGBoost that are used to train and predict the accuracy:

- **min_child_weight:**This hyperparameter determines the minimum sum of total weights of all observations required in a child. By requiring a certain minimum amount of samples in each split will help prevent overfitting.
- **gamma:** This hyperparameter determines the minimum loss reduction necessary to make a split. When this value is large, we obtain fewer splits and therefore a simple model.
- **n_estimators:** This hyperparameter provided the number of trees in the model. More trees may improve performance but also lengthen training periods and raise the risk of overfitting.
- **learning_rate:** This hyperparameter determines the step size or controls the shrinkage applied in each boosting iteration. A learning rate that is smaller can improve generalization but needs more iterations.
- **max_depth:** This hyperparameter gives the tree's maximum depth in the model. Trees that are deeper trees capture more complex interactions but also have the tendency to overfit.

3.1.2. Decision Tree (DT)

It is a type of non-parametric supervised learning technique that can be used for classification and regression. The aim is to build a model that predicts the value of a target variable using simple decision rules derived from data features [4]. A tree is an example of a piecewise constant approximation.

The following are the hyperparameters of decision trees that are used to train and predict the accuracy:

- **criterion:** This hyperparameter provides us with a function that is used to measure the quality of a split. We use mean squared error (MSE) as the function to determine the quality of the split.
- **max_depth:** This hyperparameter gives the maximum depth of the decision tree. Deeper the tree more possibility to capture complex interactions but it might tend to overfit.
- **max_features:** This hyperparameter provides the number of features that needs to be considered while looking for the best split.
- **min_samples_split:** This hyperparameter gives the minimum number of samples necessary to split an internal node. By imposing a minimum number of sample criteria in each split, it helps to avoid overfitting.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node. By imposing a minimum number of sample criteria in each leaf, it helps to avoid overfitting.

3.1.3. Random Forest (RF)

Each tree in a random forest ensemble is constructed from a sample drawn with replacement (i.e., a bootstrap sample) from the training set [4]. Furthermore, when splitting each node during tree construction, the best split is determined by selecting either all input features or a random subset of size max_features.

These two randomness factors tend to reduce the variance of the forest estimator. Individual decision trees do, in fact, often exhibit high variance and a tendency for overfitting. Decision trees with somewhat decoupled prediction errors are produced by injecting randomness into forests. Some errors can be eliminated by averaging these predictions. By combining different trees, random forests decrease variance, sometimes at the expense of a slight rise in bias. In practice, the variance reduction is frequently substantial, producing an improved model as a whole [4].

The following are the hyperparameters of random forest that are used to train and predict the accuracy:

- **n_estimators:** This hyperparameter determines the number of trees in the forest. The accuracy of the model can be increased with a higher number of trees. However, it can also increase training time and memory usage.
- **criterion:** This hyperparameter provides a function that is used to measure the quality of a split. We use mean squared error (MSE) as the function to determine the quality of the split.
- **max_depth:** This hyperparameter provides the maximum depth of every decision tree in the forest. A deeper tree can capture more complex interactions but may also overfit.
- **max_features:** This hyperparameter provides the number of features to be considered while looking for the best split.
- **min_samples_split:** This hyperparameter provides the minimum number of samples necessary to split an internal node. By imposing a minimum number of sample criteria in each split, it helps to avoid overfitting.
- **min_samples_leaf:** This hyperparameter gives the minimum number of samples necessary to be at a leaf node. By imposing a minimum number of sample criteria in each leaf, it helps to avoid overfitting.

3.1.4. Support Vector Regression (SVR)

It is a supervised machine learning algorithm that is used for regression tasks. The main aim is to find a hyperplane in the high-dimensional feature space that maximizes the margin (distance) between the hyperplane and the nearest data points.

The following are the hyperparameters of random forest that are used to train and predict the accuracy:

- **C:** This is a regularization parameter. It trades off between correctly classifying training samples and maximizing the margin of the decision function.
- **epsilon:** The region of the epsilon-tube is where errors are not penalized. In other words, any prediction that falls within this range of the true value is regarded as accurate.
- **gamma:** This acts as the kernel coefficient for 'rbf', 'poly', and 'sigmoid' kernels. The decision boundary tightly fits around the training samples when it is a smaller value, while a larger value will produce a more flexible model.
- **kernel:** provides the type of kernel to be used in the algorithm.
- **degree:** It is the degree of the polynomial kernel function ('poly'). All other kernels disregard this option.

3.2. A Comparative Analysis of the Popular Models

The most popular model in the Sine dataset is MaxAbs Scaler Hoeffding Tree Classifier and the second most popular model is MinMax Scaler Hoeffding Tree Classifier. Now, let's dissect these models considering the hyperparameters.

3.2.1. Model 1: MaxAbs Scaler Hoeffding Tree Classifier

When the hyperparameters max_depth, grace period, and max_size consider various values, there are a total of 18 possibilities for MaxAbs Scaler Hoeffding Tree Classifier model.

Each plot lists the appearance of the model along with how frequently it occurs in a given iteration. The number of iterations is represented by the x-axis. The number of times the model appears in each iteration is shown on the y-axis. This plot (Figure 46) helps visualize the popularity and consistency throughout the whole process and which hyperparameter combinations are more frequently used.

The plot below (Figure 46) with hyperparameters value 10 for max_depth, 10 for grace period and 5 for max_size is the most popular among all the combinations of hyperaparameters used by model 1.

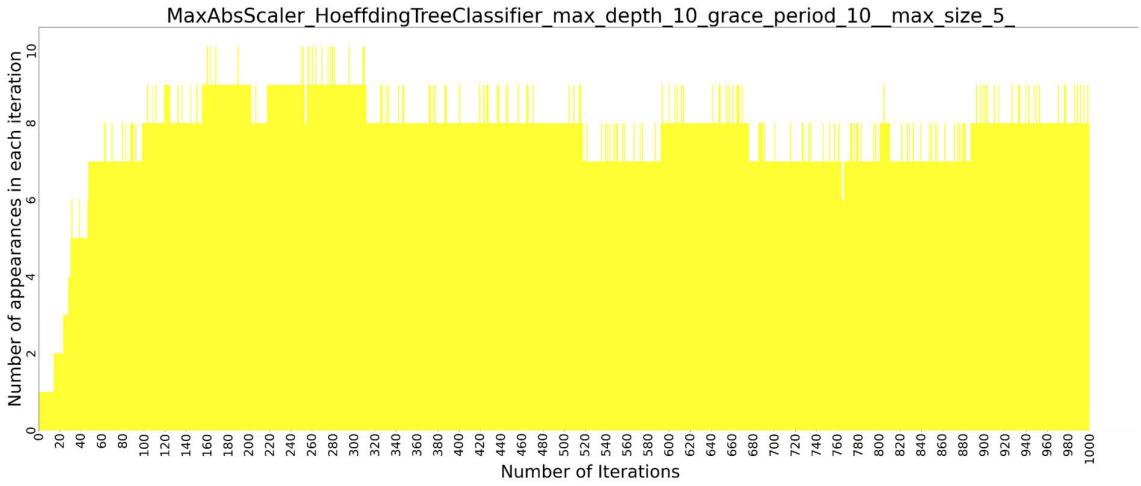


Figure 46: Usage of Maxabs Scaler Hoeffding Tree Classifier with hyperparameters max_depth: 10, grace_period: 10 and max_size: 5 throughout the process

All other combinations of model 1 rarely appear in the process and that too with very few appearances that are not consistent like the one shown below (Figure 47).

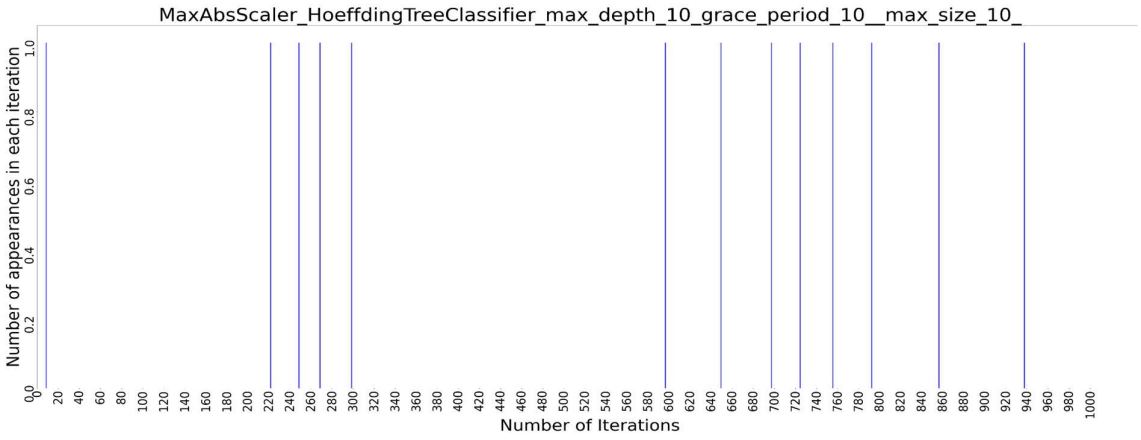


Figure 47: Usage of Maxabs Scaler Hoeffding Tree Classifier with hyperparameters max_depth: 10, grace_period: 10 and max_size: 10 throughout the process.

3.2.2. Model 2: MinMax Scaler Hoeffding Tree Classifier

When the hyperparameters max_depth, grace period, and max_size consider various values, there are a total of 15 possibilities for MinMax Scaler Hoeffding Tree Classifier model.Each plot lists the appearance along with how frequently it occurs in a given iteration. The number of iterations is represented by the x-axis. The number of times the model appears in each iteration is shown on the y-axis. This plot (Figure 48) helps visualize the popularity and consistency throughout the whole process and which hyperparameter combinations are more frequently used.

The plot (Figure 48) below with hyperparameters value 10 for max_depth, 10 for grace period and 5 for max_size is the most popular among all the combinations of hyperaparameters used by model 2.

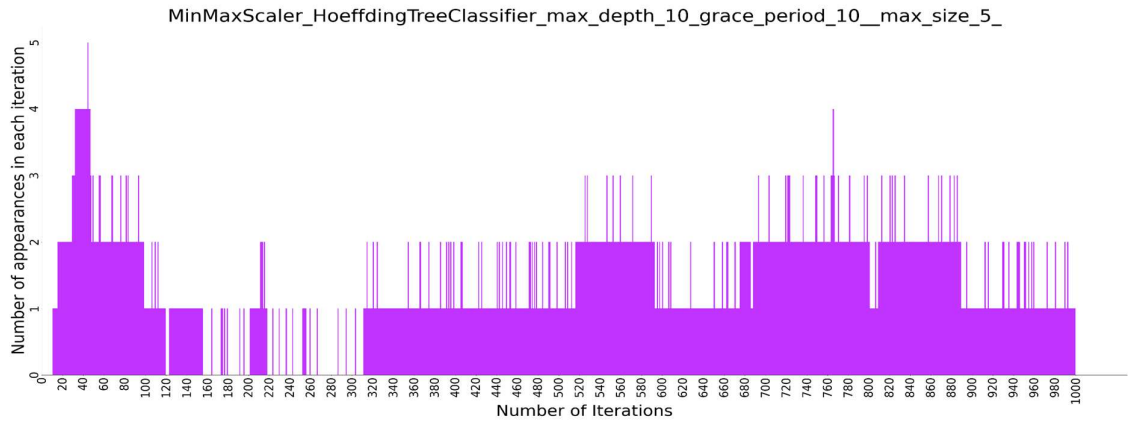


Figure 48: Usage of Minmax Scaler Hoeffding Tree Classifier with hyperparameters max_depth: 10, grace_period: 10 and max_size: 10 throughout the process.

All other combinations of model 1 rarely appear in the process and that too with very few appearances that are not consistent like the one shown below (Figure 49).

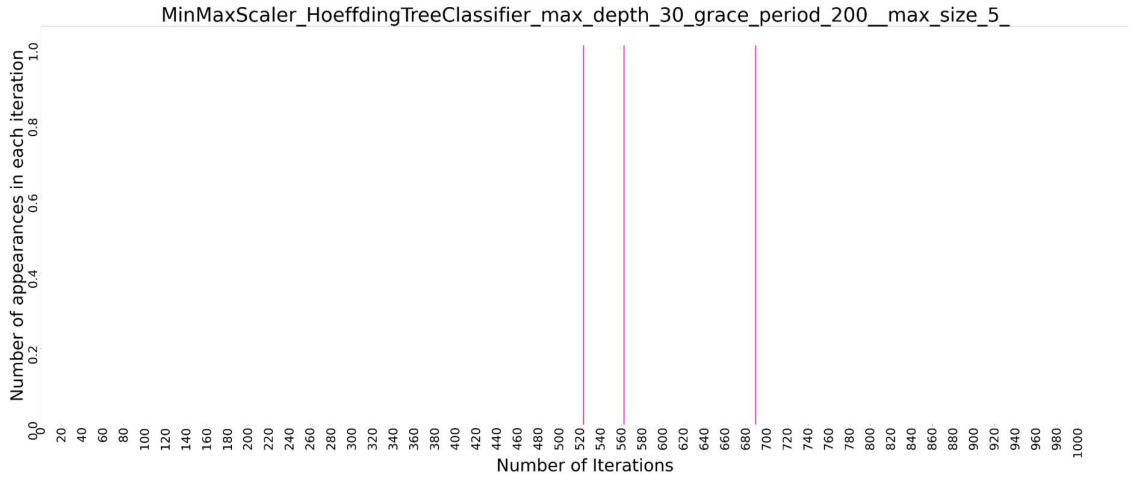


Figure 49: Usage of Minmax Scaler Hoeffding Tree Classifier with hyperparameters max_depth: 30, grace_period: 200 and max_size: 5 throughout the process.

Now, let's try to predict the accuracy for the two best model combination for each model. To do this, first we want to conduct data preparation. The input features are the averaged values of feature_0 and feature_1 (See Chapter 2) in numbers of 1000 which is the amount of data each ensemble processes. The target variable is the accuracy taken by filtering out the accuracy data corresponding to the model in each ensemble. We have 4 cases in terms of averaging the features data and the ratio of data used for training to prediction. The following are the 4 cases:

Case (i): Averaging 200 data per ensemble giving 5 sets of features- Training- 50% Prediction- 50%

Case (ii): Averaging 200 data per ensemble giving 5 sets of features- Training- 80% Prediction- 20%.

Case (iii): Averaging 100 data per ensemble giving 10 sets of features- Training- 50% Prediction- 50%.

Case (iv): Averaging 100 data per ensemble giving 10 sets of features- Training- 80% Prediction- 20%.

3.2.3. Metrics for Evaluating the Model Performance

We need some metrics to measure the performance and to derive the expected result. Regression model effectiveness is assessed using the measures MAPE, RMSE, and R2. Every metric measures distinct aspects of the model's accuracy. Here's an overview of each:

(i) Mean Absolute Percentage Error (MAPE): It is calculated by taking the difference between the real value (target/true value) and the predicted value. It is then divided by the real value. To average it across the dataset, an absolute percentage is applied. MAPE increases when the error increases. So, to get a better model performance a smaller MAPE is desired [7].

$$MAPE = \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i} * 100\%$$

where y is the real value, $\hat{y}$ is the predicted value, and n is the number of data points.

(ii) Rooted Mean Square Error (RMSE): It is calculated by taking the square root of the average of squared differences between the real and predicted value. A RMSE value of '0' indicates the model is fitted perfectly (overfitting). The lower the RMSE, the better the model and its predictions [7].

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y is the real value, $\hat{y}$ is the predicted value, and n is the number of data points.

(iii) Coefficient of Determination (R^2): It is a statistical measure that gives information about how well the regression line approximates the real value.

$$R^2 = 1 - \frac{RSS\ (Residual\ Sum\ of\ Squares)}{TSS\ (Total\ Sum\ of\ Squares)}$$

$$= \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

where y is the real value, $\hat{y}$ is the predicted value, $\bar{y}$ is the mean of all values. When, R2 is 1, the regression line perfectly fits the data and when it is 0, the regression line doesn't fit any data.

3.2.4. Hyperparameter values for each Machine Learning Technique

Hyperopt is used as the hyperparameter tuning method where the aMLLibrary can be given probability distributions on the hyperparameters, and it will use them to sample a few values when trying to find the best one.

The values for hyperparameters for each machine learning technique has been set as follows:

Machine Learning Technique	Hyperparameter values
[LRRidge]	alpha = ['loguniform(0.01,1)']
[XGBoost]	min_child_weight = [1] gamma = ['loguniform(0.1,10)'] n_estimators = [1000] learning_rate = ['loguniform(0.01,1)'] max_depth = [100]
[DecisionTree]	criterion = ['mse'] max_depth = [3] max_features = ['auto'] min_samples_split = ['loguniform(0.01,1)'] min_samples_leaf = ['loguniform(0.01,0.5)']
[RandomForest]	n_estimators = [5] criterion = ['mse'] max_depth = ['quniform(3,6,1)'] max_features = ['auto'] min_samples_split = ['loguniform(0.1,1)'] min_samples_leaf = [1]
[SVR]	C = ['loguniform(0.001,1)'] epsilon = ['loguniform(0.01,1)'] gamma = [1e-7] kernel = ['linear'] degree = [2]

Table 2: Hyperparameter values set for each ML Technique

3.2.5. Model 1: Results

For training and validation we take the most common 80-20 split with a hold out ratio of 0.2. Below are the results for each case of model 1.

	Model1-Case(i)	**Model1-Case(ii)**	**Model1-Case(iii)**	**Model1-Case(iv)**
Best Technique	Random Forest	Decision Tree	LRRidge	Random Forest
MAPE for best result	Training-0.009330 HP Selection-0.024728 Validation-0.007016	Training-0.016473 HP Selection-0.016620 Validation-0.002652	Training-0.236447 HP Selection-0.009524 Validation-0.008018	Training-0.071537 HP Selection-0.156130 Validation-0.003189
RMSE for best result	Training-0.018866 HP Selection-0.024639 Validation-0.051075	Training-0.032234 HP Selection-0.019833 Validation-0.005000	Training-0.042253 HP Selection-0.011970 Validation-0.010485	Training-0.021447 HP Selection-0.020733 Validation-0.017352
R^2 for best result	Training-0.660301 HP Selection-0.056258 Validation -294.418132	Training-0.021164 HP Selection-0.906751 Validation-1.630041	Training-0.067677 HP Selection-4.853547 Validation-11.045693	Training-0.479989 HP Selection- -0.227771 Validation- -39.693978

Table 3: Best technique, MAPE, RMSE and R^2 for all cases of Model 1.

3.2.5.1. Model 1- Case(i): Training and Validation

MaxAbs Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 5 sets of features. The dataset has been split into 50-50 where the first 500 rows are used for training and the later for prediction. (See Figure 50) for the distribution of training data.

The validation metrics for Random Forest show that it is the best model during the training phase of this case. (See Figure 51).

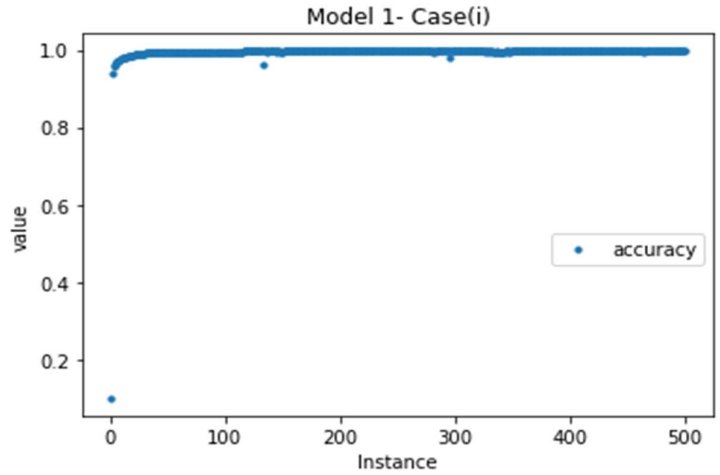


Figure 50: Distribution of training data for Model 1- case(i).

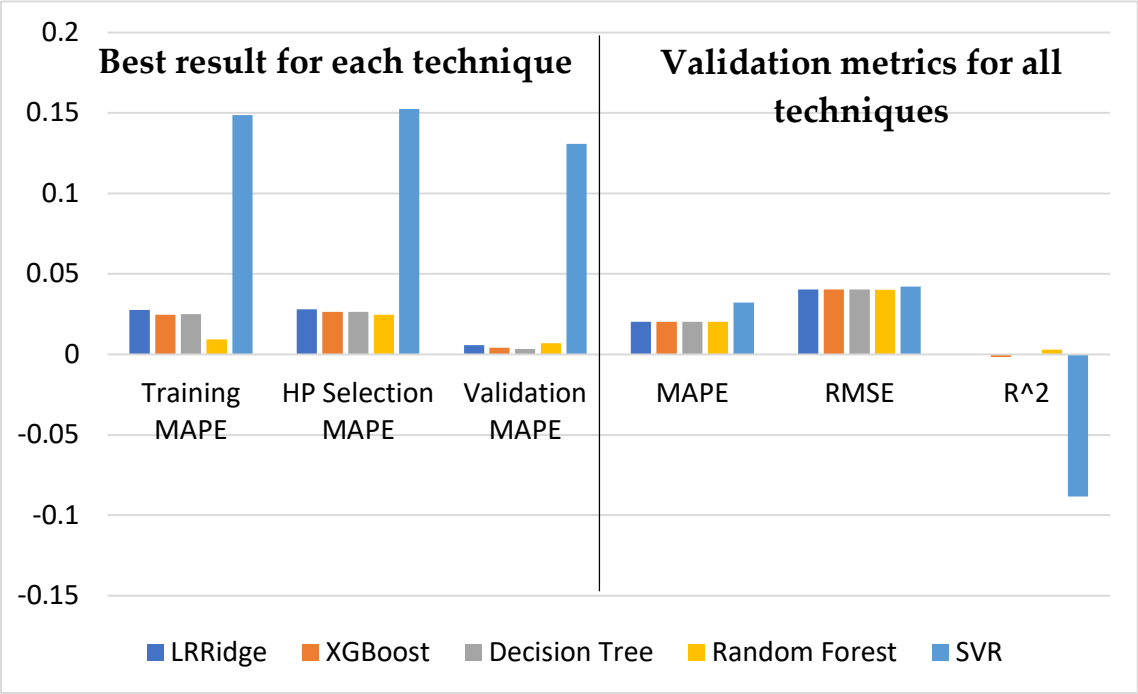


Figure 51: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	0.02745	0.024645	0.025134	0.00933	0.148651
HP Selection MAPE	0.028034	0.026301	0.026346	0.024728	0.152538
Validation MAPE	0.005639	0.004106	0.003464	0.007016	0.130821
MAPE	0.020128	0.020134	0.020115	0.020103	0.032071
RMSE	0.04029	0.040314	0.040289	0.040222	0.042025
R^2	-0.00041	-0.00162	-0.000355	0.002974	-0.08844

Table 4: Best results and Validation metrics for all techniques.

3.2.5.2. Model 1- Case(i): Prediction

The accuracy prediction for case(i) are as follows:

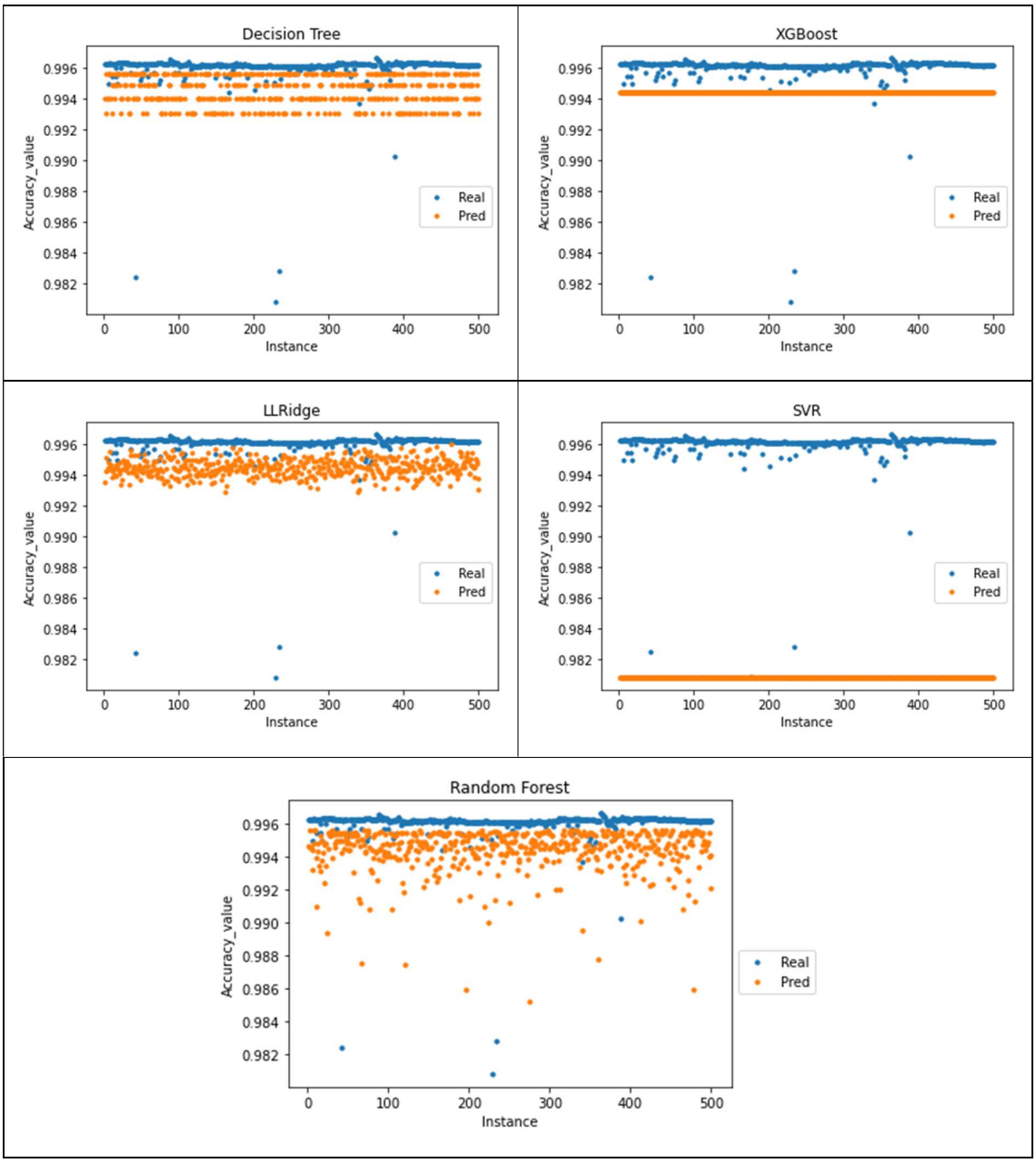


Table 5: Model 1- Case(i)- Accuracy prediction plots (real Vs predicted) for each technique.

Random forest performs better in prediction as well. It generalizes well with the prediction data by having some predictions towards the lowest accuracy value 0.982. All other models stay at the top and SVR barely predicts.

3.2.5.3. Model 1- Case(ii): Training and Validation

MaxAbs Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 5 sets of features.The dataset has been split into 80-20 where the first 800 rows are used for training and the later for prediction. Below (Figure 52) is the distribution of training data:

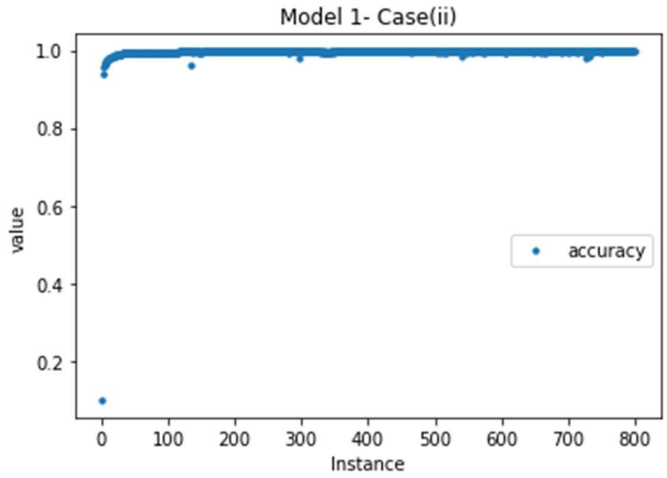


Figure 52: Distribution of training data for Model 1- case(ii).

The validation metrics for Decision Tree shows that it is the best model during the training phase of this case. (see Figure 53).

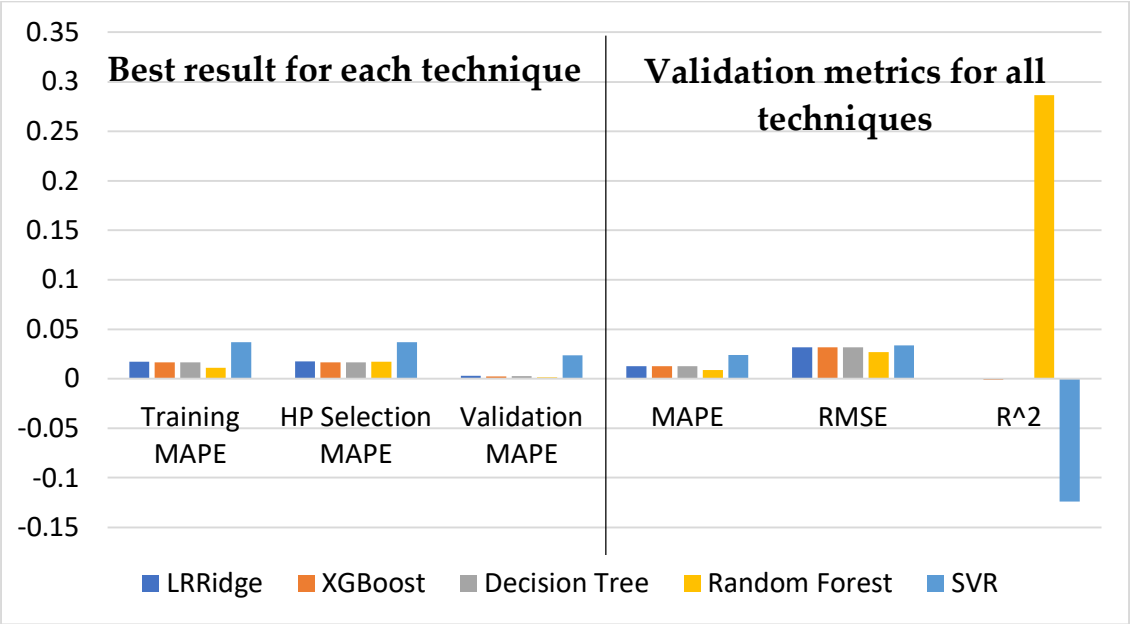


Figure 53: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	0.017497	0.016776	0.016473	0.010966	0.037104
HP Selection MAPE	0.017761	0.016823	0.01662	0.017507	0.037105
Validation MAPE	0.003083	0.002588	0.002652	0.001521	0.023776
MAPE	0.012724	0.01274	0.012742	0.008786	0.023907
RMSE	0.031898	0.031913	0.031896	0.026941	0.03381
R^2	-0.00026	-0.00119	-0.000153	0.286442	-0.12379

Table 6: Best results and Validation metrics for all techniques.

3.2.5.4. Model 1- Case(ii): Prediction

The accuracy prediction for case(ii) are as follows:

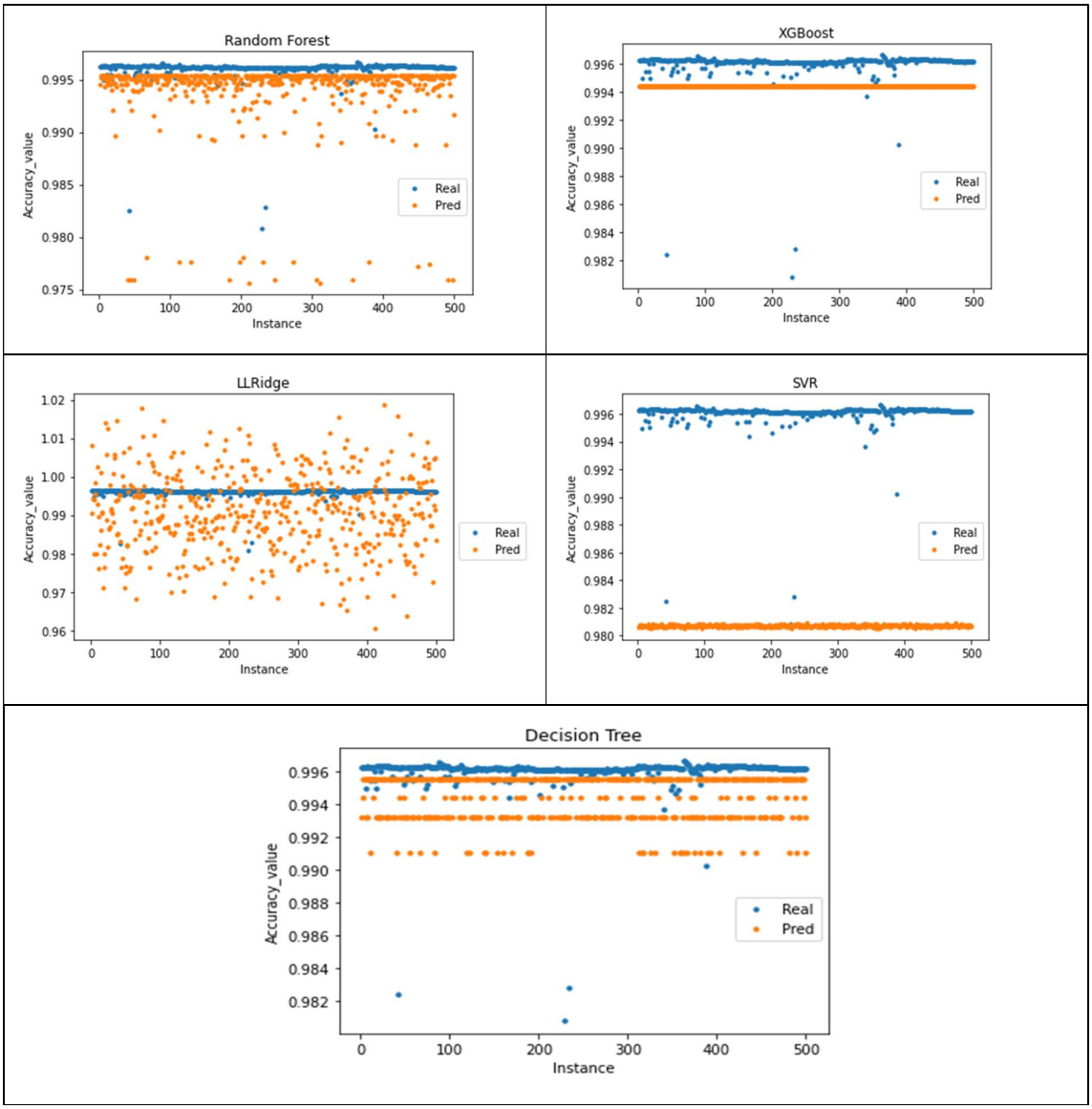


Table 7: Model 1- Case(ii)- Accuracy prediction plots (real Vs predicted) for each technique.

Random Forest and Decision tree performs comparitively better in prediction. Random Forest generalizes well with the prediction data by having some predictions towards the lowest accuracy value 0.982. Decision Tree still have some predictions towards the lower accuracy values. XGBoost predictions stay at the top and SVR barely predicts. LRRidge predicts value greater than 1 also it tries to generalize its predictions.

3.2.5.5. Model 1- Case(iii): Training and Validation

MaxAbs Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 10 sets of features. The dataset has been split into 50-50 where the first 500 rows are used for training and the later for prediction. Below (Figure 54) is the distribution of training data:

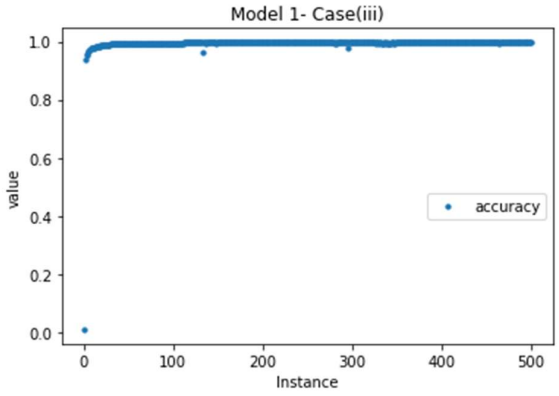


Figure 54: Distribution of training data for Model 1- case(iii).

The validation metrics for LRRidge shows that it is the best model during the training phase of this case. (see Figure 55).

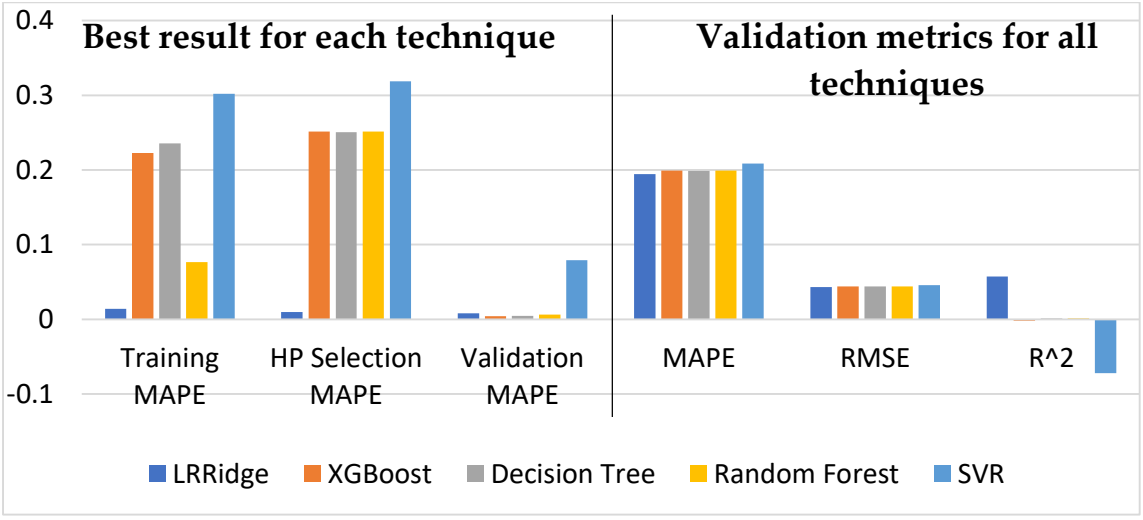


Figure 55: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	0.014063	0.222542	0.235446	0.076523	0.3018
HP Selection MAPE	0.009524	0.251024	0.250511	0.251248	0.318344
Validation MAPE	0.008018	0.00439	0.004535	0.006108	0.079082
MAPE	0.194282	0.199133	0.198919	0.199388	0.208695
RMSE	0.042979	0.044311	0.044248	0.044248	0.045848
R^2	0.05763	-0.00166	0.001178	0.001178	-0.07236

Table 8: Best results and Validation metrics for all techniques.

3.2.5.6. Model 1- Case(iii): Prediction

The accuracy prediction for case(iii) are as follows:

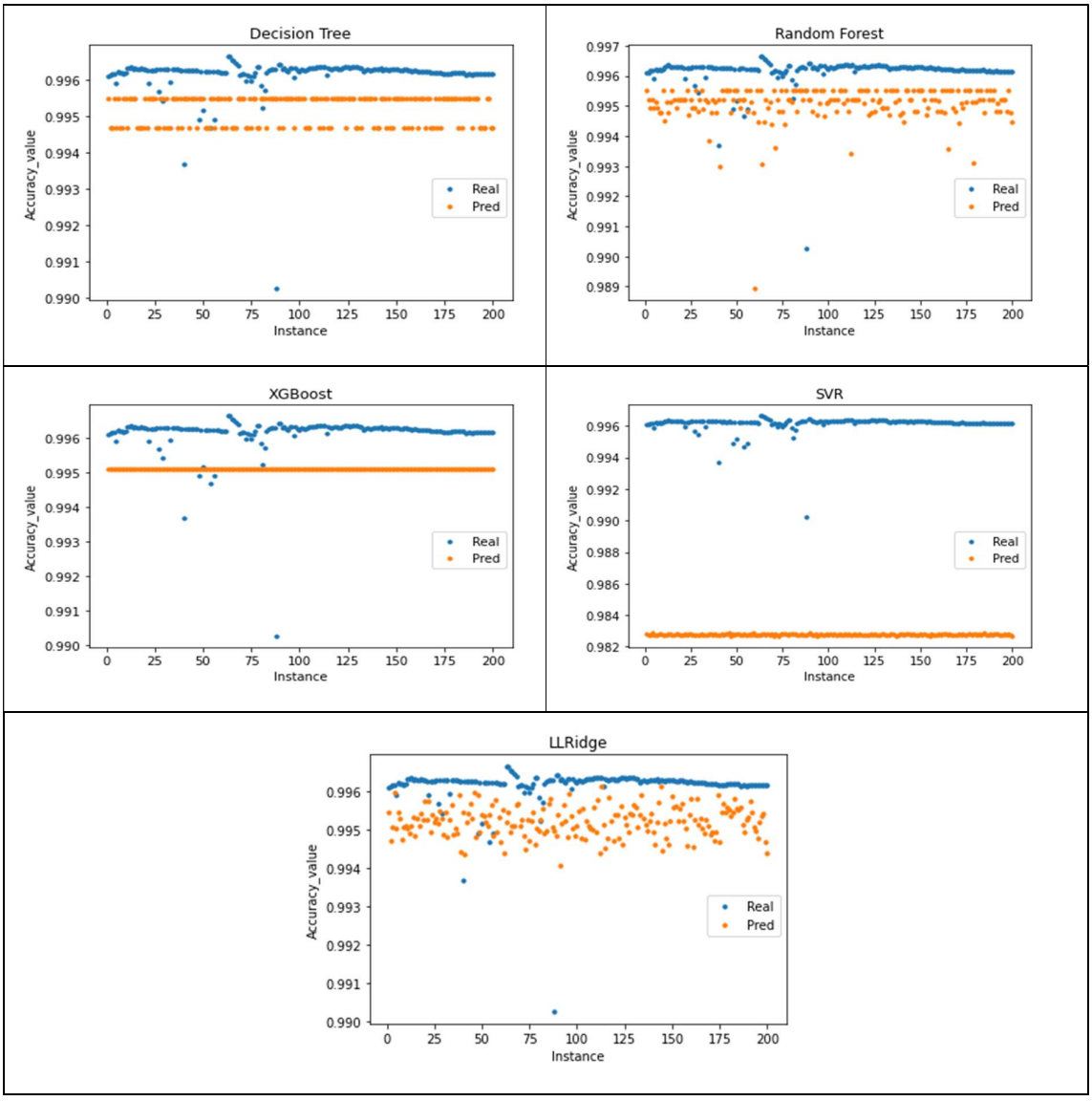


Table 9: Model 1- Case(iii)- Accuracy prediction plots (real Vs predicted) for each technique.

LRRidge, Random forest and Deicion tree performs better in prediction as well. It generalizes well with the prediction data by having some predictions towards the lowest accuracy value. XGBoost predictions stay at the top and SVR barely predicts.

3.2.5.7. Model 1- Case(iv): Training and Validation

MaxAbs Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 10 sets of features. The dataset has been split into 80-20 where the first 800 rows are used for training and the later for prediction. Below (Figure 56) is the distribution of training data:

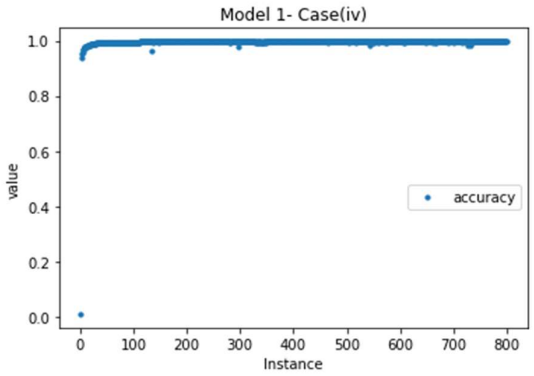


Figure 56: Distribution of training data for Model 1- case(iv).

The validation metrics for Random forest shows that it is the best model during the training phase of this case. (see Figure 57).

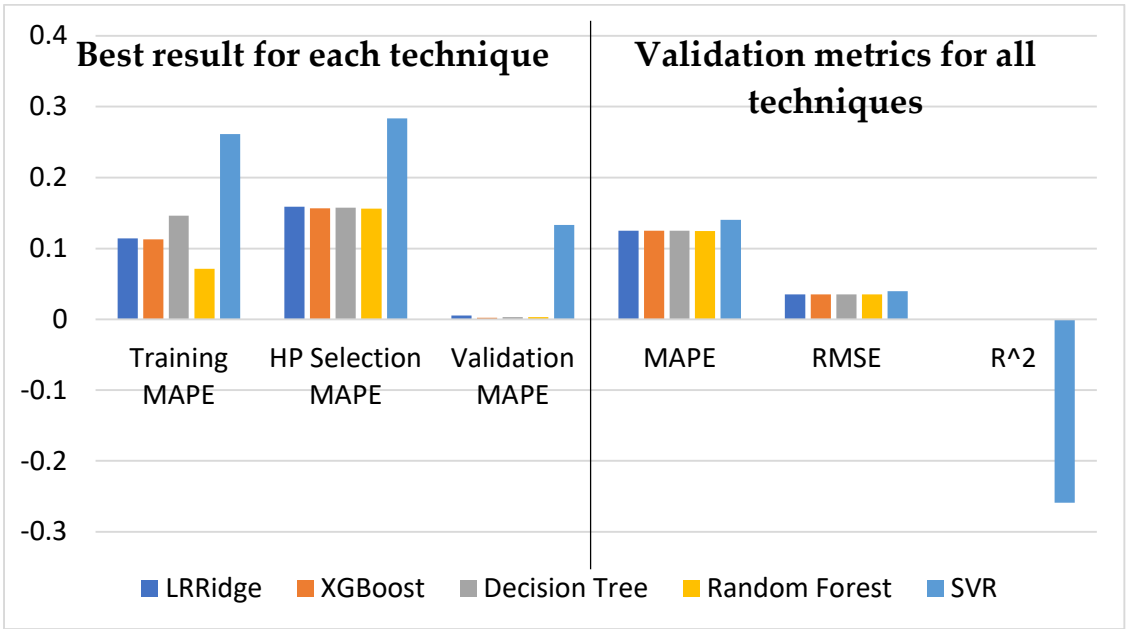


Figure 57: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	0.114349	0.11257	0.145864	0.071537	0.261246
HP Selection MAPE	0.15882	0.156485	0.157338	0.15613	0.28355
Validation MAPE	0.005048	0.002361	0.002958	0.003189	0.132869
MAPE	0.124661	0.124674	0.124657	0.12458	0.140136
RMSE	0.035069	0.035071	0.035055	0.035072	0.039326
R^2	-0.00109	-0.00125	-0.000289	-0.001289	-0.25889

Table 10: Best results and Validation metrics for all techniques.

3.2.5.8. Model 1- Case(iv): Prediction

The accuracy prediction for case(iv) are as follows:

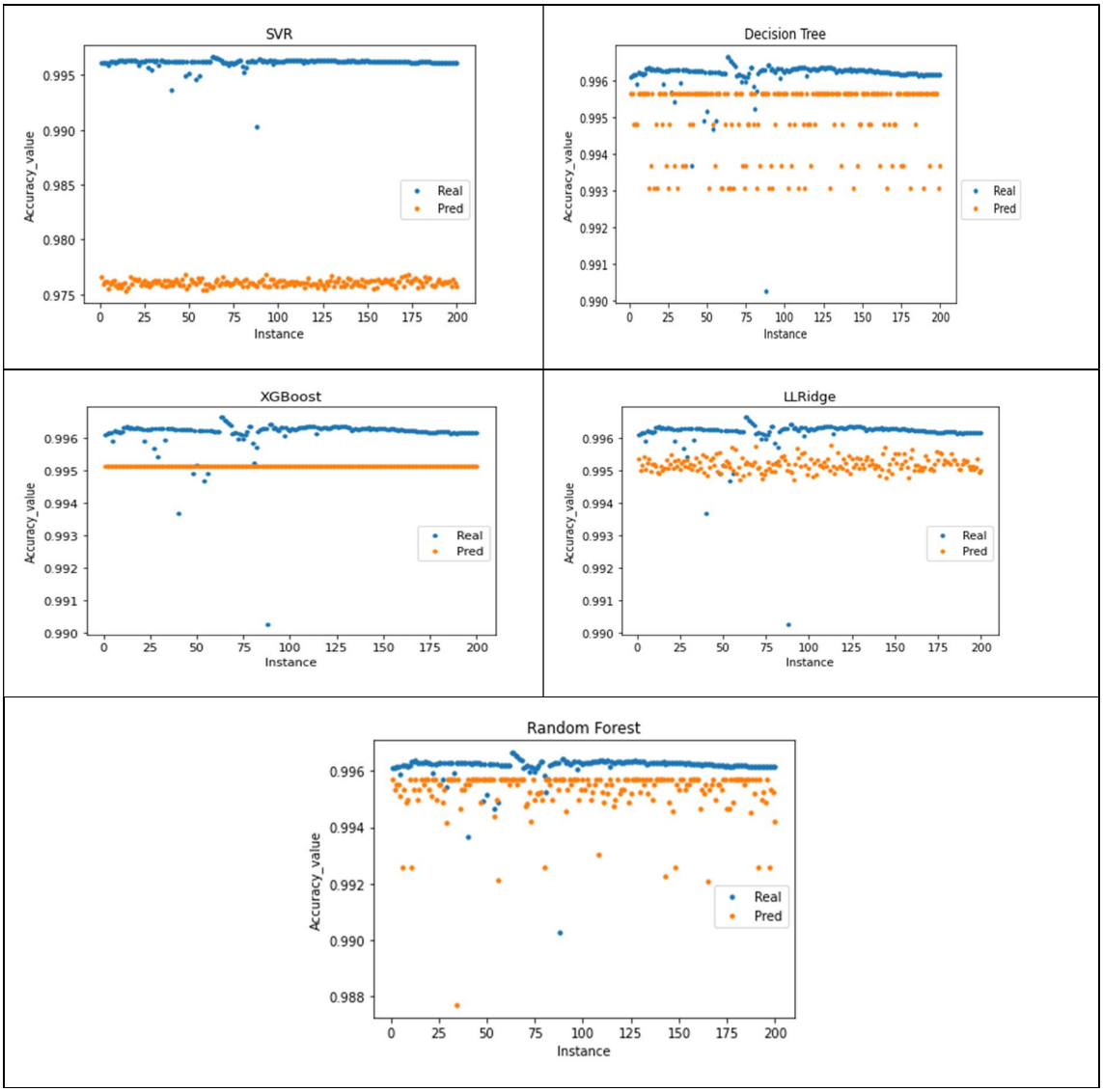


Table 11: Model 1- Case(iv)- Accuracy prediction plots (real Vs predicted) for each technique.

LRRidge, Random forest and Deicion tree performs better in prediction as well. It generalizes well with the prediction data by having some predictions towards the lowest accuracy value. XGBoost predictions stay at the top and SVR barely predicts.

3.2.6. Model 2: Results

For training and validation we take the most common 80-20 split with a hold out ratio of 0.2. Below are the results for each case of model 1.

	Model2-Case(i)	**Model2-Case(ii)**	**Model2-Case(iii)**	**Model2-Case(iv)**
Best Technique	SVR	SVR	Decision Tree	SVR
MAPE for best result	Training: 17.885168 HP Selection: 17.471476 Validation: 15.214929	Training: 12.345402 HP Selection: 12.424598 Validation: 11.67617	Training: 17.410961 HP Selection: 19.507689 Validation: 17.108263	Training: 10.000808 HP Selection: 10.540655 Validation: 9.559241
RMSE for best result	Training: 0.472918 HP Selection: 0.464132 Validation: 0.456896	Training: 12.345402 HP Selection: 12.424598 Validation: 11.676170	Training: 0.411706 HP Selection 0.460758 Validation: 0.445622	Training: 0.465471 HP Selection: 0.478310 Validation: 0.466078
R^2 for best result	Training: 0.176795 HP Selection: 0.171973 Validation: 0.221826	Training: -0.364833 HP Selection: -0.402058 Validation: -0.401614	Training: 0.111004 HP Selection -0.143800: Validation: -0.159244	Training: -0.624766 HP Selection: -0.786275 Validation:-0.71217

Table 12: Best technique, MAPE, RMSE and R^2 for all cases of Model 2.

3.2.6.1. Model 2- Case(i): Training and Validation

MinMax Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 5 sets of features. The dataset has been split into 50-50 where the first 500 rows are used for training and the later for prediction. Below (Figure 56) is the distribution of training data:

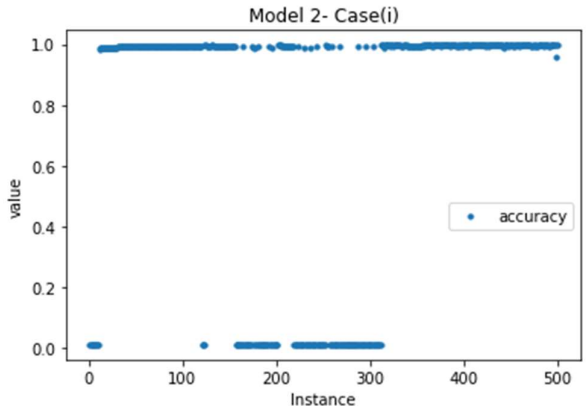


Figure 58: Distribution of training data for Model 2- case(i)

The validation metrics for SVR shows that it is the best model during the training phase of this case. (see Figure 59).

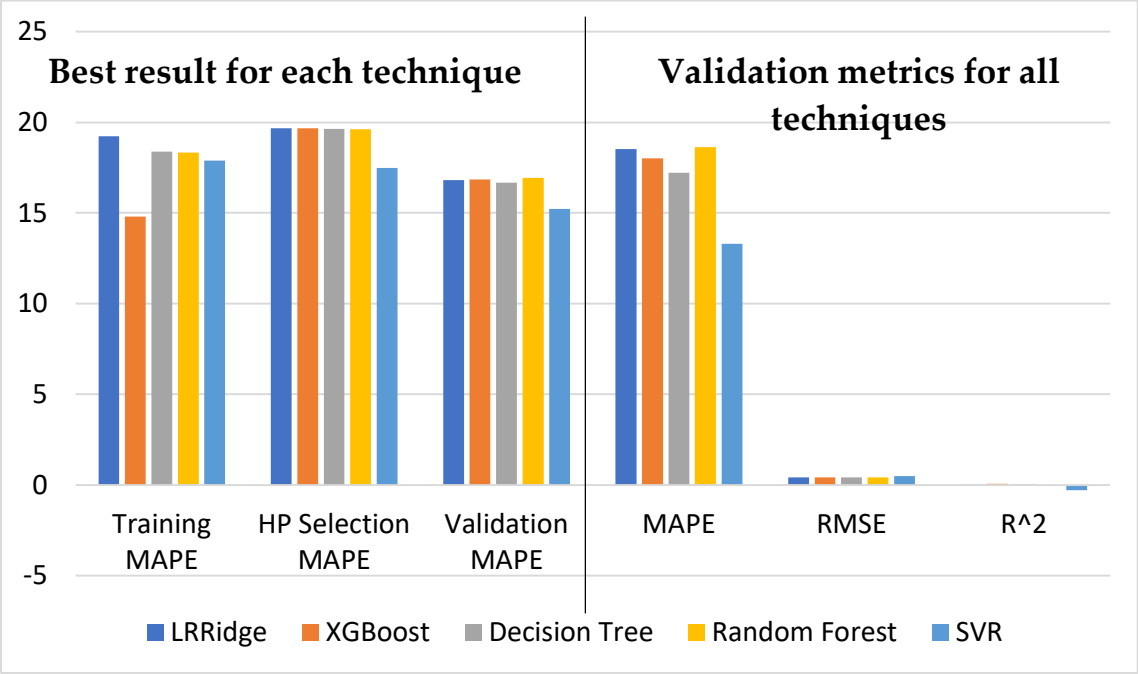


Figure 59: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	19.22463	14.80571	18.381681	18.327856	17.88517
HP Selection MAPE	19.65979	19.664836	19.632512	19.6054	17.47148
Validation MAPE	16.80628	16.84099	16.671066	16.92023	15.21493
MAPE	18.52014	18.00097	17.201945	18.625393	13.2941
RMSE	0.430463	0.420199	0.422221	0.433606	0.491735
R^2	0.011488	0.058064	0.048978	-0.003001	-0.289949

Table 13: Best results and Validation metrics for all techniques.

3.2.6.2. Model 2- Case(i): Prediction

The accuracy prediction for case(i) are as follows:

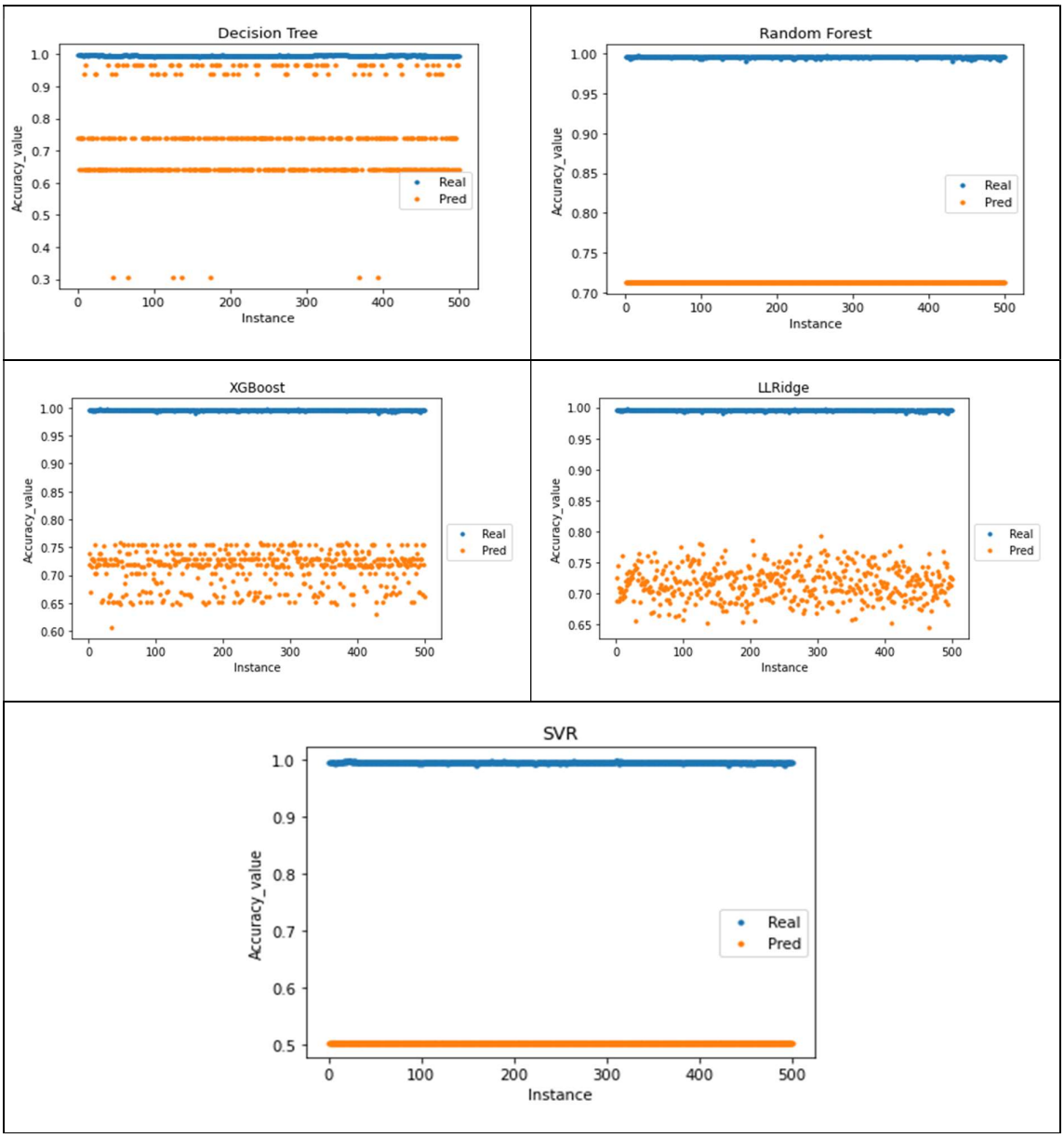


Table 14: Model 2- Case(i)- Accuracy prediction plots (real Vs predicted) for each technique.

Random Forest and SVR predictions stays at the bottom which barely predicts. Decision Tree predicts better than all other models. LRRidge and XGBoost predictions stay at the bottom but tries to move toward upside so it shows a bit of generalization.

3.2.6.3. Model 2- Case(ii): Training and Validation

MinMax Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 5 sets of features. The dataset has been split into 80-20 where the first 500 rows are used for training and the later for prediction. Below (Figure 60) is the distribution of training data:

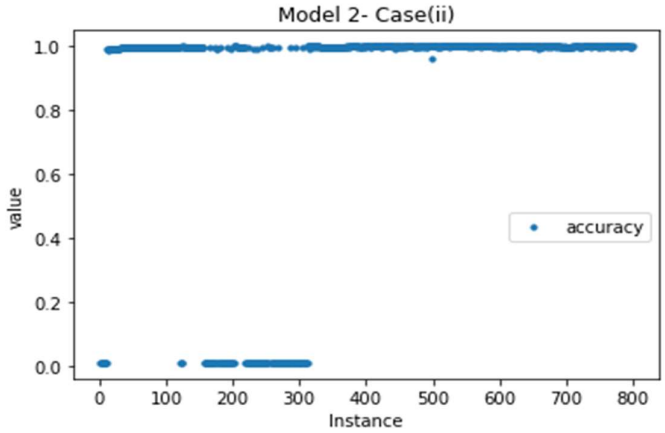


Figure 60: Distribution of training data for Model 2- case(ii).

The validation metrics for SVR shows that it is the best model during the training phase of this case. (see Figure 61).

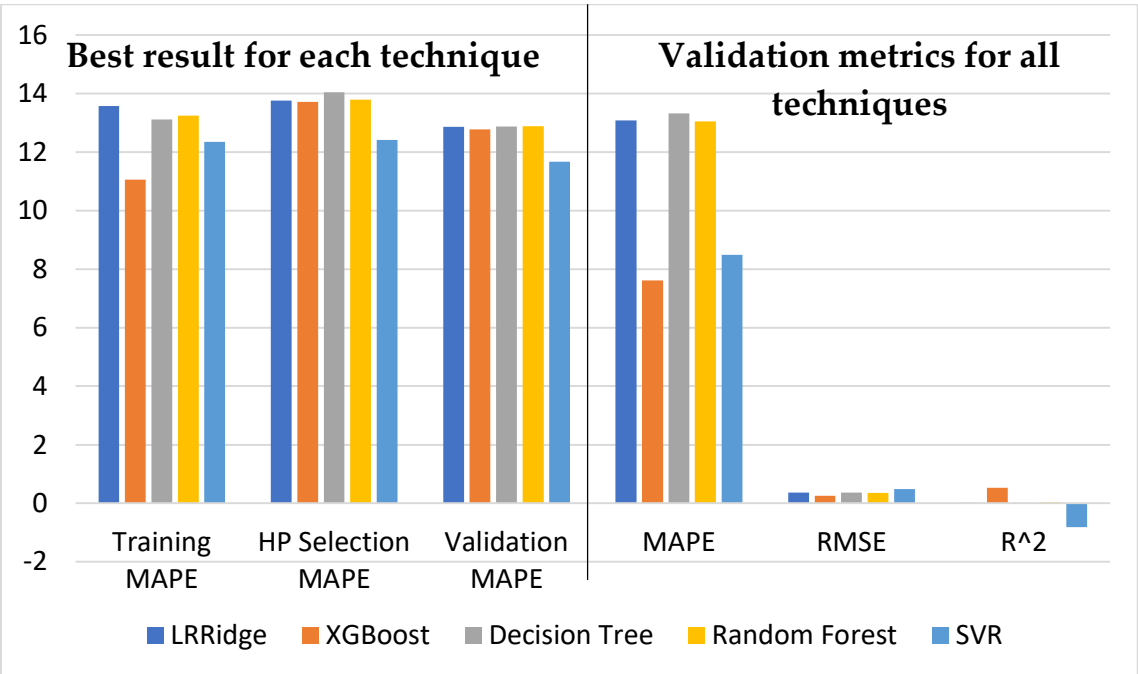


Figure 61: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	13.57347	11.061139	13.117463	13.243066	12.3454
HP Selection MAPE	13.76077	13.716941	14.045788	13.800762	12.4246
Validation MAPE	12.86779	12.786677	12.878937	12.880629	11.67617
MAPE	13.08328	7.614614	13.326696	13.04895	8.494232
RMSE	0.361831	0.250328	0.362777	0.357803	0.492008
R^2	0.015031	0.528554	0.009873	0.036839	-0.82119

Table 15: Best results and Validation metrics for all techniques.

3.2.6.4. Model 2- Case(ii): Prediction

The accuracy prediction for case(ii) are as follows:

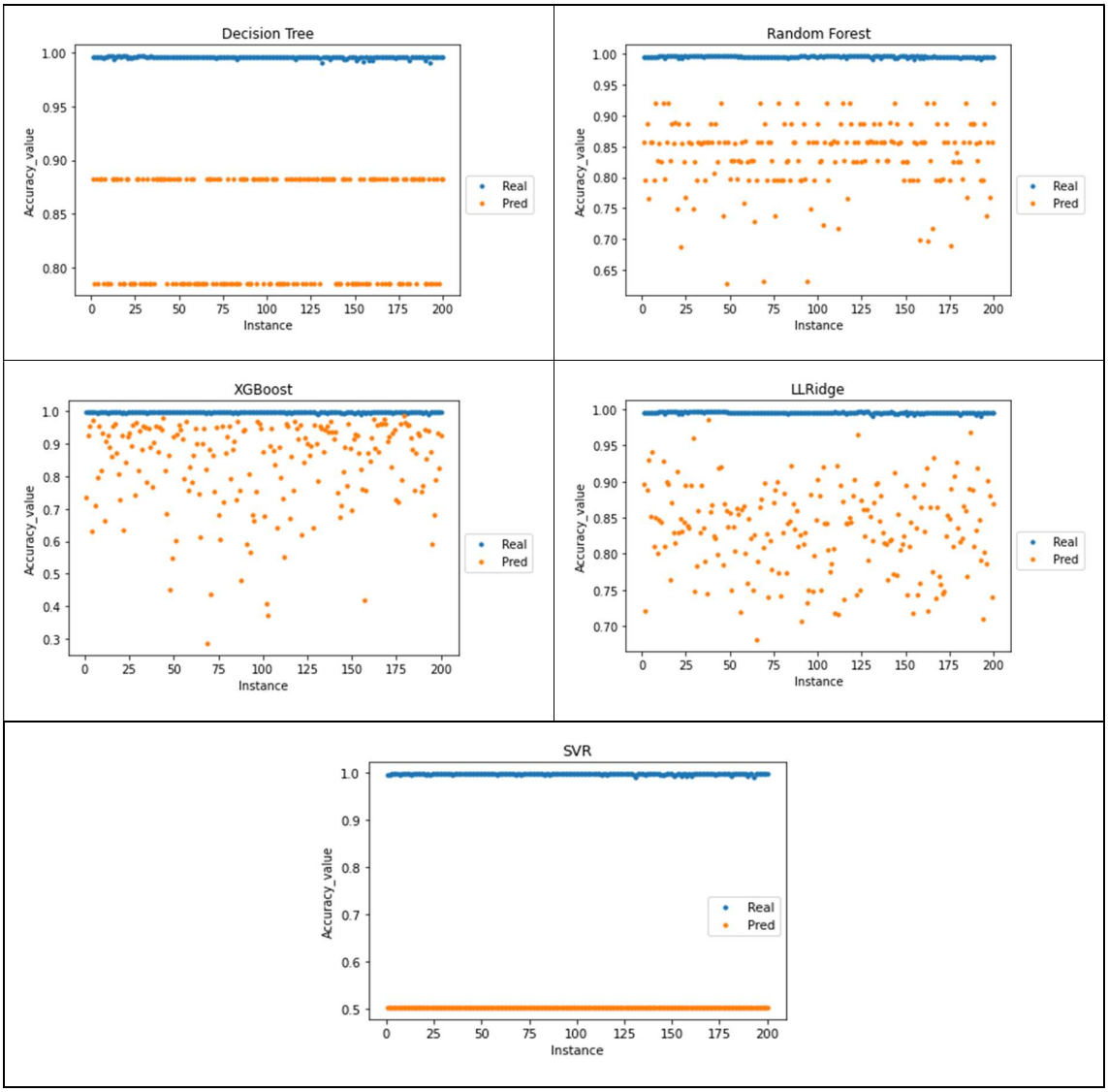


Table 16: Model 2- Case(ii)- Accuracy prediction plots (real Vs predicted) for each technique.

XGBoost performs better than other models. LRRidge and Random Forest tries to generalize whereas SVR and decision tree perform very bad.

3.2.6.5. Model 2- Case(iii): Training and Validation

MinMax Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 10 sets of features. The dataset has been split into 50-50 where the first 500 rows are used for training and the later for prediction. Below (Figure 62) is the distribution of training data:

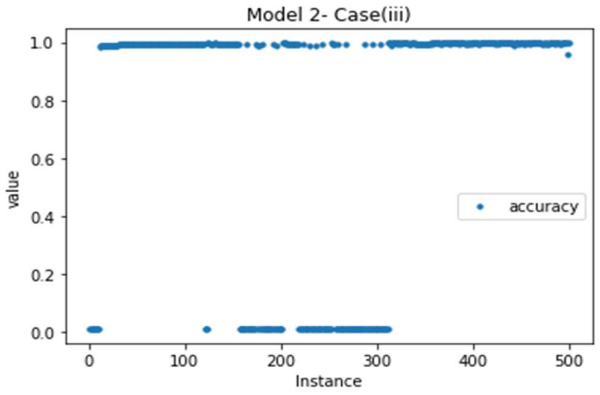


Figure 62: Distribution of training data for Model 2- case(iii).

The validation metrics for SVR shows that it is the best model during the training phase of this case. (see Figure 63).

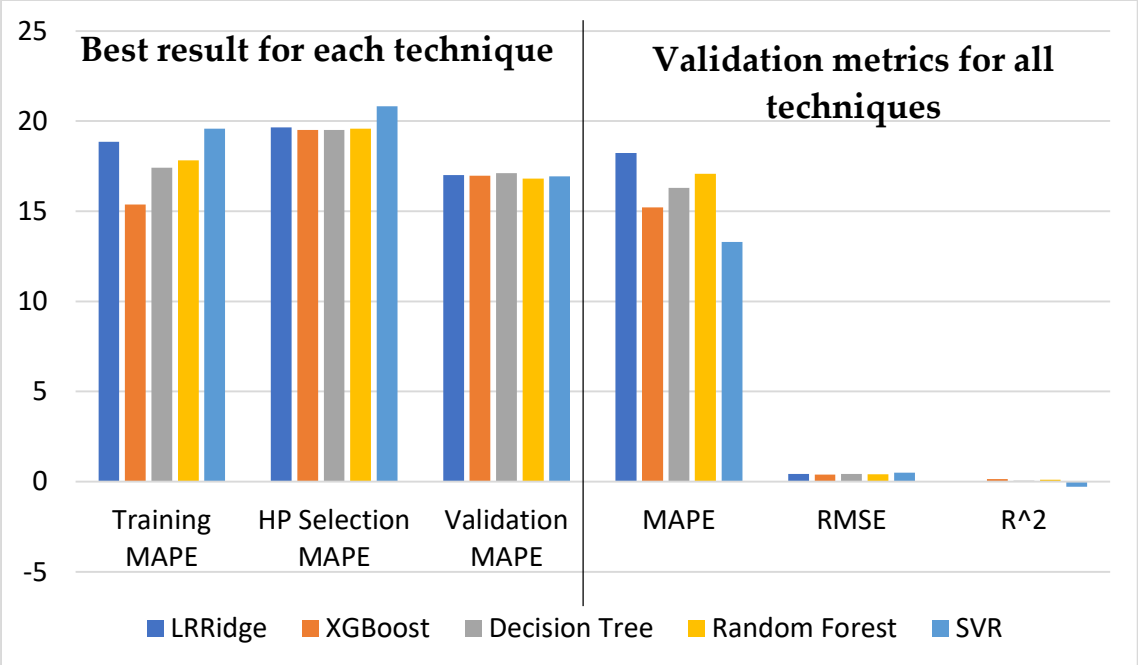


Figure 63: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	18.84055	15.370861	17.410961	17.831168	19.57405
HP Selection MAPE	19.65186	19.5095	19.507689	19.582777	20.81447
Validation MAPE	17.00413	16.98004	17.108263	16.804799	16.92706
MAPE	18.23117	15.208379	16.295243	17.083865	13.2941
RMSE	0.430283	0.398919	0.420371	0.408284	0.491735
R^2	0.012315	0.1520838	0.057297	0.110726	-0.289949

Table 17: Best results and Validation metrics for all techniques.

3.2.6.6. Model 2- Case(iii): Prediction

The accuracy prediction for case(iii) are as follows:

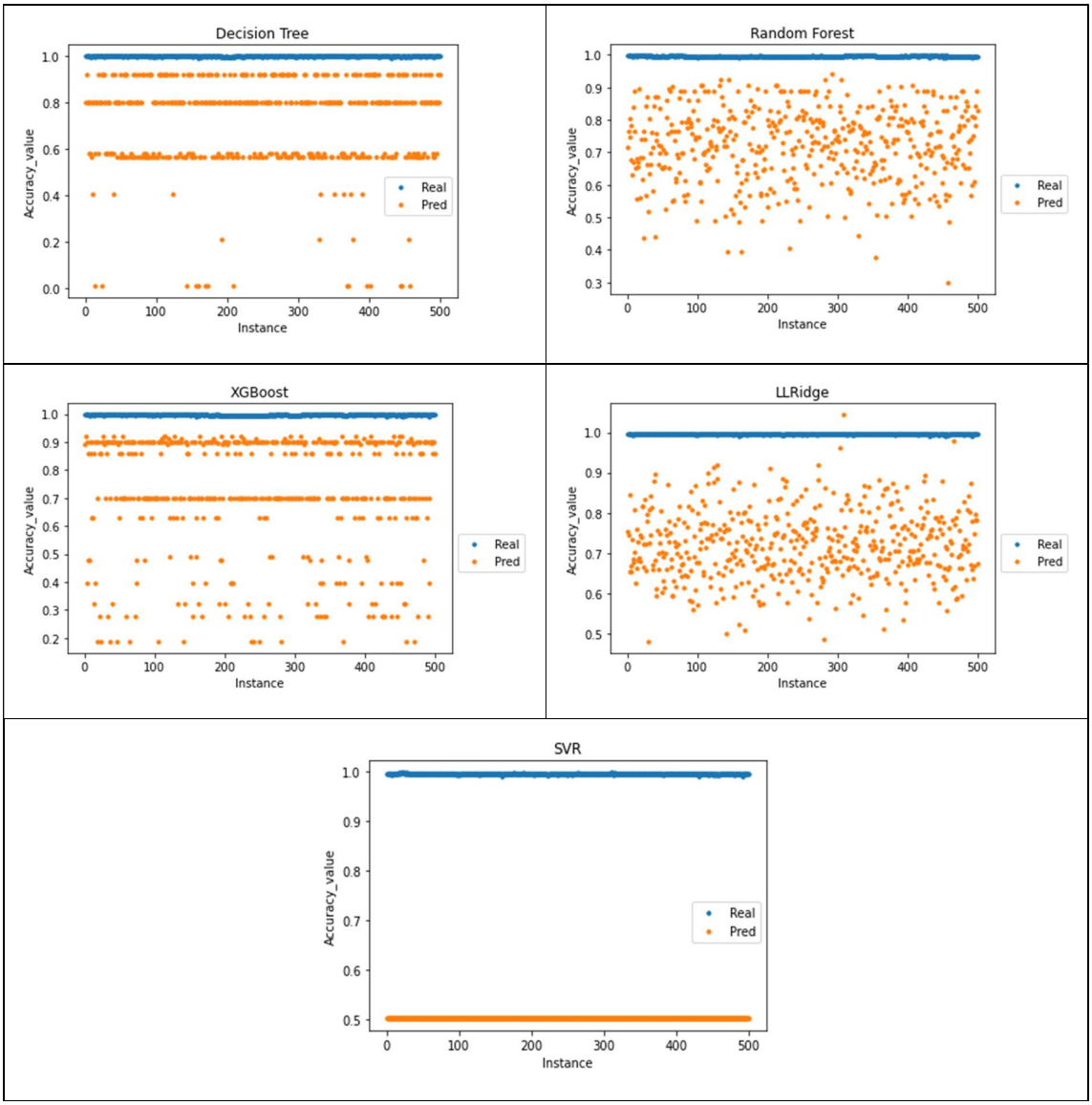


Table 18: Model 2- Case(iii)- Accuracy prediction plots (real Vs predicted) for each technique.

All the models try to generalize its prediction except SVR predictions still stays at the bottom.

3.2.6.7. Model 2- Case(iv): Training and Validation

MinMax Scaler Hoeffding Tree Classifier (max_depth: 10, grace_period: 10, max_size: 5) with 10 sets of features. The dataset has been split into 50-50 where the first 500 rows are used for training and the later for prediction. Below is the distribution of training data:

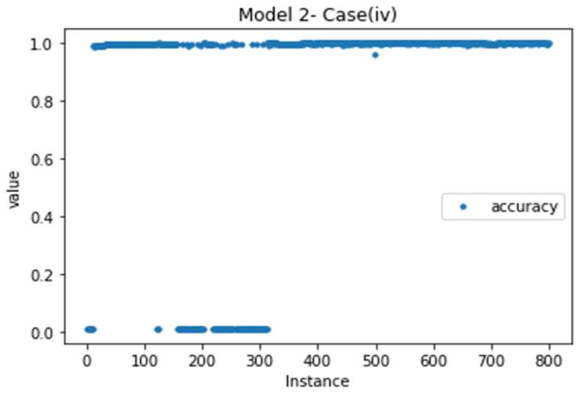


Figure 64: Distribution of training data for Model 2- case(iv).

The validation metrics for SVR shows that it is the best model during the training phase of this case. (see Figure 65).

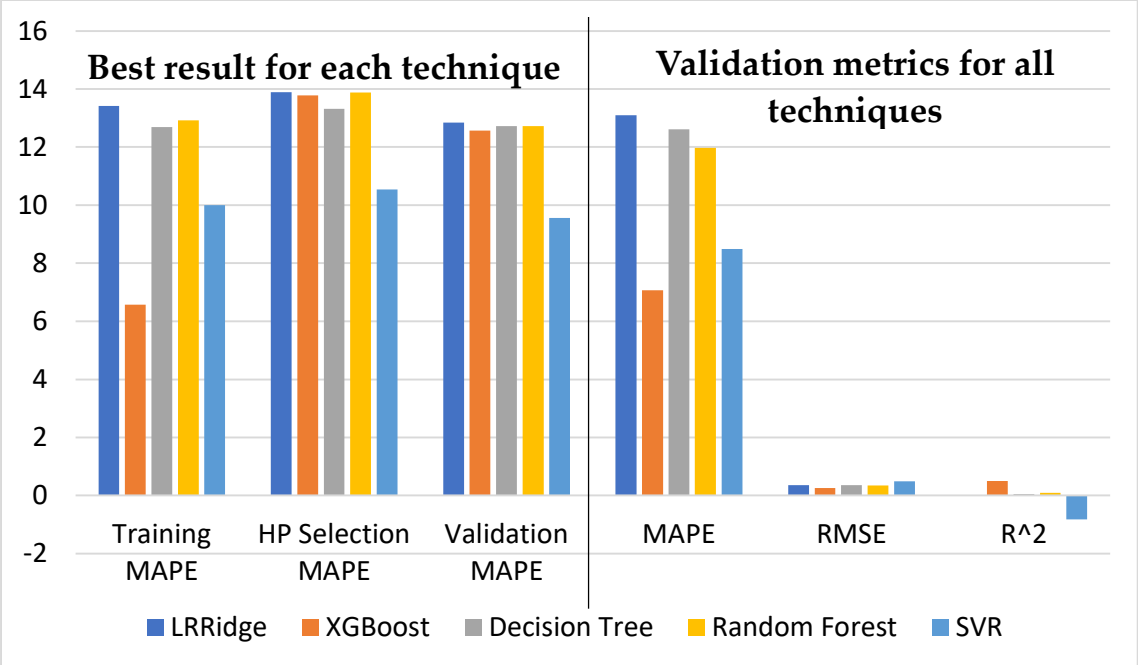


Figure 65: Best results and Validation metrics for all techniques.

	LRRidge	XGBoost	Decision Tree	Random Forest	SVR
Training MAPE	13.4119	6.575098	12.689751	12.927854	10.00081
HP Selection MAPE	13.88889	13.782451	13.314182	13.879215	10.54066
Validation MAPE	12.84111	12.570727	12.733736	12.718655	9.559241
MAPE	13.09653	7.071582	12.610544	11.969572	8.494232
RMSE	0.36144	0.257041	0.356329	0.348895	0.492008
R^2	0.017157	0.502932	0.044759	0.084197	-0.82119

Table 19: Best results and Validation metrics for all techniques.

3.2.6.8. Model 2- Case(iv): Prediction

The accuracy prediction for case(iv) are as follows:

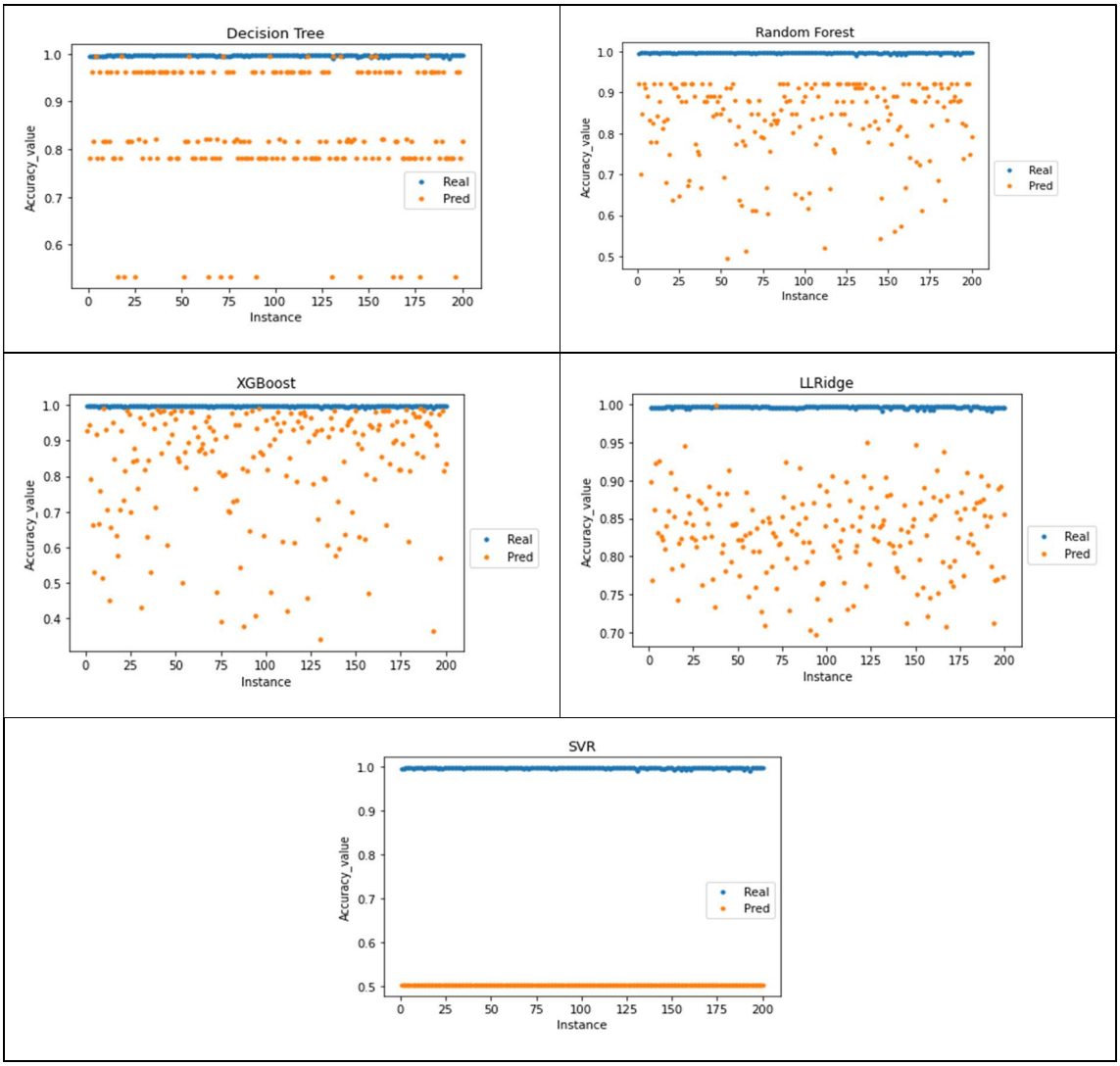


Table 20: Model 2- Case(iv)- Accuracy prediction plots (real Vs predicted) for each technique.

All the models try to generalize its prediction except SVR predictions still stays at the bottom.

4 Conclusion

The results show that the accuracy prediction is crucial for selecting the best performing model for replacement. On evaluating the performance of Model 1 and Model 2 using the Sine dataset, it is found that Model 1 outperfoms Model 2 in all the scenarios (i.e., training, validation and testing). This study recommends that ranking the models on the basis of the accuracy predictions for identifying the best model to replace the worst one identified by EvoAutoML. This approach can help in improving the overall performance of the process and can be used on different datasets and problems, enabling better decision-making and performance optimization.

Bibliography

[1] Kulbach, C., Montiel, J., Bahri, M., Bifet, A., Heyden, M., "Evolution-based Online Automated Machine Learning," *HAL Open Science,* HAL Id: hal-03667231.

[2] Feurer, M., Klein, A., Eggensperger, e.a.: Efficient and Robust Automated Machine Learning. In: Cortes, C., Lawrence, N.D., Lee, D.D. (eds.) *Advances in Neural Information Processing Systems 28: NIPS*. pp. 2962–2970 (2015)

[3] https://github.com/aMLLibrary/aMLLibrary

[4] https://scikit-learn.org/stable/

[5] Friedman J, Hastie T, Tibshirani R, et al. (2000). "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)." *The annals of statistics*, 28(2), 337–407.

[6] Friedman JH (2001). "Greedy function approximation: a gradient boosting machine." *Annals of Statistics*, pp. 1189–1232.

[7] https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/

[8] https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html

[9] Lonut A, "Automated Machine Learning Techniques for Data Streams." *arXiv:2106.07317*.

[10] Alshalali, T., Josyula, DP., "ICISDM 2020: Proceedings of the 2020 the 4th International Conference on Information System and Data Mining." *ISBN:9781450377652.*

[11] Thornton, C., Hutter, F., Hoos, HH., Brown, KL., "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms.",*arXiv:1208.3719.*

[12] Oza, N.C.: Online bagging and boosting. In: *ICSMC. pp. 2340–2345. IEEE (2005*).

[13] Domingos, P.M., Hulten, G.: *Mining high-speed data streams*. In: Ramakrishnan, R., Stolfo, S.J., Bayardo, R.J., Parsa, I. (eds.) *SIGKDD. pp. 71–80. ACM (2000*).

[14] Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: Balc´azar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD. vol. 6321, pp. 135–150. Springer (2010).

[15] Gomes, H.M., Bifet, A., Read, J., Barddal, J.P., Enembreck, F., Pfahringer, B., Holmes, G., Abdessalem, T.: Adaptive random forests for evolving data stream classification. *ML 106(9-10), 1469–1495 (2017).*

[16] https://www.oreilly.com/library/view/python-feature-engineering/9781789806311/45ab74c1-2d3d-4775-b1c7-a832233c528b.xhtml.

[17] https://thecleverprogrammer.com/2020/09/22/standardscaler-in-machine-learning/#.

[18] https://riverml.xyz/dev/api/tree/HoeffdingTreeClassifier/.

[19] https://www.upgrad.com/blog/gaussian-naive-bayes/.

[20] https://www.kdnuggets.com/2022/04/logistic-regression-classification.html.

[21] https://www.kaggle.com/code/prashant111/logistic-regression-classifier-tutorial.

[22] https://blog.kambria.io/logistic-regression-for-machine-learning/.

[23] https://www.ibm.com/topics/knn.

[24] https://riverml.xyz/dev/api/datasets/synth/Sine/.

[25] https://riverml.xyz/dev/api/datasets/synth/Agrawal/.

[26] Gama, João & Medas, Pedro & Castillo, Gladys & Rodrigues, Pedro. (2004). Learning with Drift Detection. Intelligent Data Analysis. *8. 286-295. 10.1007/978-3-540-28645-5_29*.

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to Politecnico di Milano for providing me with the resources, support, and opportunity to pursue this work. Without which, this work would not be feasible.

I am very much grateful to Professor Danilo Ardagna for his significant guidance, expertise, and encouragement throughout my research. His constant feedback and support have been instrumental in shaping the direction of my work.

I also extend my gratitude to co-advisor Bruno Guindani for his valuable insights, contributions and constant support throughout my work which helped me a lot in completing this work.

My heartfelt thanks go to my parents and my sister for their unwavering love, support, and motivation. Their sacrifices and encouragement have been the driving force behind my academic achievements.

I would also like to thank my cousins and friends for their emotional support and unconditional love throughout this journey. Their words of encouragement have helped me overcome challenges and stay motivated.

Once again, I express my sincere gratitude to everyone who supported and encouraged me throughout my research.