PONTIFÍCIA UNIVERSIDADECATÓLICA DO PARANÁ

ANDRÉ SEIJI SUZUKI KUSAKARIBA

ENHANCING STATE-OF-THE-ART ENSEMBLES IN VARYING FEATURE SPACES IN DATA STREAMS THROUGH FEATURE IMPORTANCE MONITORING

CURITIBA

2025

Pontifícia Universidade Católica do Paraná
Programa de Pós-Graduação em Informática

ENHANCING STATE-OF-THE-ART ENSEMBLES IN VARYING FEATURE SPACES IN DATA STREAMS THROUGH FEATURE IMPORTANCE MONITORING

André Seiji Suzuki Kusakariba

Dissertation presented to the *Programa de Pós-Graduação em Informática* as a partial requirement for the degree of Master in Informatics.

Major Field: Computer Science

Supervisor: Fabrício Enembreck

Co-supervisor: Jean Paul Barddal

Curitiba
2025

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Edilene de Oliveira dos Santos CRB 9 / 1636

K97e
2025

Kusakariba, André Seiji Suzuki
Enhancing state-of-the-art ensembles in varying feature spaces in data streams through feature importance monitoring / André Seiji Suzuki Kusakariba ; supervisor: Fabrício Enembreck ; co-supervisor: Jean Paul Barddal. -- 2025
79 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2025
Bibliografia: f.75-79

1. Informática. 2. Mineração de dados (Computação). 3. Fluxo de dados (Computadores). 4. Aprendizado do computador. 5. Monitoramento. I. Enembreck, Fabrício. II. Barddal, Jean Paul. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título

CDD 20. ed. – 004

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

Curitiba, 14 de agosto de 2025.

87-2025

DECLARAÇÃO

Declaro para os devidos fins, que **ANDRÉ SEIJI SUZUKI KUSAKARIBA** defendeu a dissertação de Mestrado intitulada "**ENHANCING STATE-OF-THE-ART ENSEMBLES IN VARYING FEATURE SPACES IN DATA STREAMS THROUGH FEATURE IMPORTANCE MONITORING**", na área de concentração Ciência da Computação no dia 13 de maio de 2025, no qual foi aprovado.

Declaro ainda, que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

Documento assinado digitalmente
JEAN PAUL BARDDAL
Data: 15/08/2025 14:49:52-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Jean Paul Barddal
Coordenador do Programa de Pós-Graduação em Informática

Acknowledgements

I would like to thank my supervisor, Fabrício Enembreck, and my co-supervisor, Jean Paul Barddal, for their guidance throughout this journey. The knowledge and insights they have shared with me are immeasurable.

"Overtime, as the parts of Theseus ship were replaced, one by one, the question arose: Is it still the same ship, or has it become something new?" — Paraphrase of Plutarch's "Life of Theseus"

Abstract

In data stream mining, the availability of features can oscillate along the data stream. This requires the algorithm to be able to adapt, so its prediction performance does not decay, since some features may no longer be present in the stream.

This problem is called Varying Feature Spaces in Data Streams, and previous works either considered batch learning (a small number of instances compared to attributes) or assumed that information was known before the beginning of the stream. No works have been made about the impact of Varying Feature Spaces in Streaming Random Patches (SRP), an ensemble classifier that typically produces deep trees.

This project presents an analysis and proposal of a meta-ensemble method designed for varying feature spaces in data streams, termed Feature Importance Streaming Random Patches (FISRP). The proposed approach is empirically evaluated against Streaming Random Patches (SRP) and Adaptive Random Forest (ARF), both benchmark algorithms recognized among the state-of-the-art methods for data stream classification.

FISRP is a meta-ensemble that combines the base learners from SRP with a method of prioritizing features into their subspaces based on their importance ranking from Feature Scoring using Tree-Based Ensembles for Evolving Data Streams. This ranking is made by an isolated learner that is trained in the same instances.

Experiments indicate that in situations where SRP's trees grow extensively, FISRP uses shallower trees, and the classification performance is higher, while the processing times and memory usage are lower.
Key-words: Data Stream Classification, Streaming Random Patches, Adaptive Random Forest, Varying Feature Spaces in Data Streams, Feature Importance, Feature Scoring for Evolving Data Streams

Resumo

Em mineração de dados a disponibilidade de variáveis pode oscilar ao longo do fluxo de dados. É necessário que o algoritmo seja apto a se adaptar para que sua performance de classificação não diminua, já que algumas variáveis podem não mais estar presentes no fluxo de dados.

Este problema é conhecido de Espaço Variável de Características em mineração de dados e trabalhos anteriores foram feitos considerando base de dados estáticas (com número de instâncias menores do que de atributos) ou considerando que informações são conhecidas antes do início do fluxo de dados. Não há referências sobre o impacto do Espaço Variável de Características no Streaming Random Patches (SRP), um classificador que tipicamente produz árvores demasiadamente profundas.

Este projeto propõe e analisa um meta ensemble em suas capacidades de lidar no cenário de Espaço de Características Variável. Esse algoritmo modificado, "Feature Importance Streaming Random Patches (FISRP)", é comparado com os SRP e Adaptive Random Forest (ARF), ambos apresentados como algoritmos estado-da-arte para classificação em fluxos de dados.

FISRP é uma variação de agrupamento de classificadores que combina os classificadores do "Streaming Random Patches (SRP)" com um método de priorizar características dentro de seus subespaços de características, baseado num ranking de importância vindo de "Feature Scoring using Tree-Based Ensembles for Evolving Data Streams". Esse ranking é feito separadamente dos classificadores principais do agrupamento mas é treinado nos mesmos exemplos.

Experimentos feitos neste trabalho indicam que em situações nas quais as árvores do SRP crescem demasiadamente, o FISRP usa árvores mais rasas e sua performance de classificação é maior, enquanto seu tempo de processamento e uso de memória são menores.

Palavras-chave: Classificação em Fluxo de dados, Streaming Random Patches, Adaptive Random Forest, Espaço de Características Variável, Importância de Características, Ranking de Características em Fluxo de dados

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction and Literature Review

1

Introduction

Machine learning algorithms tend to perform well using static datasets since all information is known beforehand. On the other hand, in real-world problems such as online learning (real-time data stream mining) knowing the information in advance is not possible. This makes the classification task much harder and requires adaptability and efficient knowledge extraction from the data.

The complexity of online learning lies within the constant changing of the stream's characteristics. As the stream is updated, new instances can bring new information: some features can disappear and others can start appearing. This means that the availability of each feature may change over time and the model must adapt to it. This problem is called Varying Feature Spaces in Data Streams and real-world applications are: environment monitoring, CCTV streaming images, intrusion detection, trending topic analysis on Twitter and email spam filtering (WANG; YOU, 2020).

Streaming Random Patches (SRP) is an ensemble classifier that typically employs deep trees, which can lead to increased processing times and higher memory usage without enhancing classification performance. This drawback can be pronounced in data streams with varying feature spaces.

This project proposes and evaluates a modified version of SRP that uses a feature importance method to handle Varying Feature Spaces in Data Streams. The modified ensemble, called Feature Importance Streaming Random Patches (FISRP) is tested against its original version to analyze if selecting relevant features outperforms random selection of features in Varying Feature Spaces in Data Streams. The relevant features are selected by a learner responsible for extracting all feature importances. This learner works isolated from the rest of the ensemble and it is trained on the same instances.

The reason for selecting features to be included in the subspaces is to allow the ensemble to better adapt to the changes in Varying Feature Spaces in Data Streams scenarios. At any given time, if a learner does not contain a minimum amount of selected features in its subspace, it is replaced by another learner created prioritizing selected features. This is different from the state-of-the-art ensembles, such as SRP, which uses drift detection to replace its learners.

1.1 Objectives

The objective is to propose and evaluate a modified ensemble classifier that uses a feature importance method to handle Varying Feature Spaces in Data Streams. The specific objectives are:

- Create a metagenerator used for simulating Varying Feature Spaces in Data Streams.
- Propose and evaluate a modified ensemble that uses Feature Importance for the selection of its features while the stream is running. This ensemble (FISRP) is compared with its original version (SRP).

1.2 Hypothesis

The project hypothesis is as follows.

- In Varying Feature Spaces in Data Streams scenario where the SRP's trees grow extensively, the proposed Feature Importance Streaming Random Patches achieve higher classification performance and lower computation cost using smaller trees (i.e., lower amount of nodes and shallower trees).

Prioritizing features should reduce the size of learners as opposed to randomly selecting features because it avoids using low-ranked features. Processing time, memory usage, and tree size should be reduced.

1.3 Contributions

The proposed contributions are as follows.

- A framework for simulation of variable feature spaces in data streams creating four scenarios: increment, removal, parabolic, and hyperbolic. In the increment scenario, the number of features increases as the stream progresses; in the removal scenario, the number of features decreases; in the parabolic scenario, the feature count initially increases and then decreases halfway through the stream; and in the hyperbolic scenario, the feature count initially decreases and then increases halfway through the stream;
- The proposal and evaluation of a modified version of SRP that uses a feature importance method to select features while new data arrive from the stream.

1.4 Financial Support

This project was funded by the Pontifical Catholic University of Paraná (PUCPR) with a monthly tuition waiver and the "ASSOCIAÇÃO BRASILEIRA DE EDUCAÇÃO E CULTURA - ABEC" via the "CONSULTORIA E FORMAÇÃO EM CIÊNCIA DE DADOS E APRENDIZAGEM DE MÁQUINA NOS COLÉGIOS MARISTAS" program. The financial support started in September 2022.

1.5 Overview

The project is divided in 7 chapters. Chapter 2 explains machine learning classifiers and data stream mining. Chapter 3 introduces Varying Feature Spaces in Data Stream mining and related works. Chapter 4 presents the proposed framework to simulate Varying Feature Spaces in Data Streams in four different settings (i.e. increment, removal, parabolic and hyperbolic). In Chapter 5, the Feature Importance Streaming Random Patches (FISPR) is presented. In Chapter 6, experiments on varying feature spaces with FISRP in data streams are reported. Chapter 7 contains the conclusions and future steps.

2

Classification in Data Streams

As the amount of data generated has increased in the last decade, the importance of analyzing information for strategic decision-making is unanimous and machine learning is used extensively for making classifications.

Classification in machine learning is described as pattern recognition in automatic discovery of regularities in data with the use of computer algorithms. The regularities can be used to classify the data into categories (BISHOP, 2007).

An example of a classification process is applied to handwritten digit recognition. Having a training set (a large set of images of digits and their labels) the machine learning algorithm can learn a function that is able to categorize never-before images on a test set (BISHOP, 2007).

Data streams are defined as "unbounded sequences of multidimensional, irregular, and intermittent observations made available along time" ((BAHRI, 2020), (BAHRI et al., 2021)).

Data stream classification, a supervised predictive task can be described as follows: $\mathcal{S}$ is a data stream providing instances $(\vec{x}^t, y^t)$, arriving at a timestamp t. $\vec{x}^t$ is d-dimensional vector and $y_t \in Y$ represents its corresponding class. The objective is to create and update a predictive model $f : \vec{x} \longrightarrow Y$ that is as accurate as possible while being restricted by memory usage and processing time (BIFET et al., 2010).

Prequential evaluation is a method used in data streams, in which the instance is used for testing followed by training the learner (GAMA; SEBASTIAO; RODRIGUES, 2013). The reason is that in data streams the instances arrive continuously and are not indefinitely stored in memory.

This work addresses supervised learning and prequential evaluation, which most of the related works in data stream classification used.

2.1 Dynamic Environments

The traditional classification in machine learning is called Offline learning or batch learning (BURLUTSKIY et al., 2016). In this setting, the entire dataset is available before the training phase begins. The model is able to train on each instance more than once, benefiting its general accuracy performance. Therefore, once the training is finished, the model will not be able to adapt to changes caused by new instances in the test dataset (GÉRON, 2022).

Classification in data streams is done using online learning, a more complex process that has no prior access to the complete dataset. An exception is mini-batches which process more than one instance at each iteration (LI et al., 2018).

Online learning algorithms process every new instance once. There is no need for storage or reprocessing. This means that Online learning algorithms benefit from not having to do multiple passes through the data set like batch algorithms (OZA; RUSSELL, 2001).

2.2 Concept Drift

In machine learning algorithms the task of learning from previous data to create a description to predict the label of a new instance is concept attainment (learning from examples). The learner uses a set of features from past instances and establishes a pattern (SCHLIMMER; GRANGER, 1986).

However, as new instances arrive this pattern may not accurately represent the data and the predictions will fail. This change in the concept or distribution of the dataset is called Concept Drift.

Concept Drift is the changing of the underlying data distribution or relationships between input attributes and class over time (HOENS; POLIKAR; CHAWLA, 2012)

Equation (2.1) represents Bayes' theorem to calculate the probability that the instance X_t belongs to class C_i:

$$p(C_i|X_{t+1}) = \frac{p(C_i)p(X_{t+1}|C_i)}{p(X_{t+1})} \tag{2.1}$$

where $p(C_i)$ is the prior probability of class C_i, $p(X_{t+1}|C_i)$ is the conditional probability of observing X_{t+1} given class C_i and $p(X_{t+1})$ is the probability of observing X_{t+1}. Concept drift can happen due to (HOENS; POLIKAR; CHAWLA, 2012):

- a change to the prior probability of observing each class $p(C_i)$;
- a change in the distribution of the classes;
- a change to the posterior distribution of class $p(C_i|X_{t+1})$.

Although the loss of predictive performance due to Concept Drift is present in batch learning, it is more challenging in data streams due to the complexity associated with the time dimension: transient and intermittent observations made available along time Mehta et al. (2017).

This work focuses on a variation of the concept drift called feature drift. Feature drift happens during a stream when a set of features becomes relevant or stops being relevant for classification (BARDDAL et al., 2017).

An example of this situation is a real-time score prediction of a soccer match. An unexpected situation such as players being switched can change the outcome of the match and all training made by the algorithm prior to this point may not be valid for making future predictions. Thus, the model may be discarded.

2.3 Base Classifiers

A base classifier (monolithic) is a single learner algorithm. This section presents two usual methods for data stream classification.

2.3.1 Naive Bayes

The algorithm assumes that all attributes are independent of each other and while that is not necessarily always true, it has a strong performance in real-world scenarios (BAHRI; MANIU; BIFET, 2018).

Naive Bayes classifier for data streams is based on the Bayes Theorem described in Equation (1).

The classifier works as follows: it learns from training data the conditional probability of each attribute belonging to a class. Classification is done by applying Bayes rules to calculate the probability of all classes for a given instance and selecting the class with the highest probability. (FRIEDMAN; GEIGER; GOLDSZMIDT, 1997)

2.3.2 Hoeffding Tree

A decision tree is a classification model that contains a tree-shaped structure. A tree consists of either a leaf (node without children) or a test node linked to at least two subtrees. Each test is based on one attribute and each outcome is a subtree.

Each instance starts its classification at the root (first node) and the traversal occurs until a leaf is reached. The label assigned to a leaf node represents the class prediction for the instance. (QUINLAN, 1996)

The Hoeffding Tree (HT) is an online learning version of a decision tree classifier. It is efficient in memory usage because it does not store any instances from the online stream, being able to learn by seeing every instance once and assuming that the distribution of data does not change over time (KUMAR; KAUR; SHARMA, 2015).

HT is based on the Hoeffding bound principle, which calculates the smallest number of instances at a given node to assess sufficient statistical information to select an optimal feature for splitting (LAVANYA et al., 2017). The Hoeffding bound is explained in Equation (2.2):

$$\varepsilon = \sqrt{\frac{R^2 \ln(\delta^{-1})}{2N}} \tag{2.2}$$

After N independent observations, with a probability $(1-\delta)$, the true mean of a random attribute of a range R does not differ more than ε from the estimated mean. Therefore the attribute selected after N instances is the same as if the selection were made using infinite instances (RUTKOWSKI et al., 2012).

Another metric used is Shannon's Entropy $H(k)$ (Equation 2.3), which is needed to calculate impurity in Hoeffding Trees, where $S(k)$ is the feature associated with split node k, $V(f_i)$ is the set of possible values for feature f_i, N is total number of instances, N^k is instances arriving at k and N_p^k is the subset of N_k in which value of $S(k)$ is equal to p:

$$H(k) = -\sum_{p \in V(S(k))} \frac{N_p^k}{N^k} \log_2 \frac{N_p^k}{N^k} \tag{2.3}$$

Equation 2.4 is used to calculate the decrease in impurity $DI(k)$ per split node k that divided feature f_i, where k_l and k_r is the number of instances passing respectively along the left and right children of k:

$$DI(k) = H(k) - N^{k_l} H(k_l) - N^{k_r} H(k_r) \tag{2.4}$$

The value of $DI(k)$ appears as $merit$ in Algorithm 1, which creates a list with the values for the decrease in impurity for every Feature in a given Feature Space. A larger value of $DI(k)$ indicates that the Feature more effectively separates the classes, leading to better classification.

Algorithm 1 Creation of bestSplitSuggestions of a Hoeffding Tree

```
    Symbols: BSS: bestSplitSuggestions
    Input: Learner
    Output: BSS Flag
1: for all F in Learner.FeatureSpace do
2:     bestSuggestion ← Add(AttributeClassObserver.getMerit(F))
3:     if bestSuggestion ≠ Null then
4:         BSS ← Add(bestSuggestion)
5:     end if
6: end for
7: Return BSS
```

Algorithm 2 demonstrates the decision of splitting a node. A split happens if only one valid suggestion exists. Also, if at least two suggestions exit and the difference between the two highest merits exceeds the Hoeffding bound or if the bound is below the tie threshold, the node is split on the best suggestion.

Algorithm 2 Split node of a Hoeffding Tree

```
Symbols: BSS: bestSplitSuggestions; HB: hoeffdingBound; TT: tieThreshold
Input: Learner, bestSplitSuggestions, hoeffdingBound, tieThreshold
Output: ShouldSplit Flag
1: ShouldSplit ← False
2: BSS.sortby(merit)
3: if SizeOf(BSS) < 2 ∧ SizeOf(BSS) > 0 then
4:     ShouldSplit ← True
5: else
6:     bestSuggestion ← BSS.get(SizeOf(BSS) − 1)
7:     secondBestSuggestion ← BSS.get(SizeOf(BSS) − 2)
8:     if (bestSuggestion.merit - secondBestSuggestion.merit) > HB ∨ HB < TT then
9:         ShouldSplit ← True
10:    end if
11: end if
12: Return ShouldSplit
```

The algorithm performs well after a few initial instances and given enough instances the model can be nearly identical to the batch decision trees (DOMINGOS; HULTEN, 2000). While using Hoeffding bound, the learner can achieve nearly identical output to a non-incremental learner with infinitely many examples (BIFET; GAVALDA, 2009).

2.3.3 Hoeffding Adaptive Trees

Hoeffding Adaptive Trees (HAT) is a variation of a HT that handles concept drift in data streams, partially restarting its tree.

HAT is able to reset nodes that are no longer relevant to the classification. By default, it uses ADWIN change detector which works as follows: checks if two distinct groups (an older one representing past data and a newer one with current data behavior) exist within the window of a node. If it does, a possible concept drift occurs and the oldest part of the window is discarded. The distinction is made using Hoeffding Bound (BIFET; GAVALDA, 2009).

Each node has its own ADWIN estimator; therefore different nodes can have data from different periods of the stream in their respective windows. Nodes at the root handle a broader range of data, while deeper nodes are more prone to instability due to the use of more specific subsets of data.

A node with a stable data subset will retain information from a longer window, hence making decisions based on long-term trends. Another node detecting frequent changes in its subset will have a shorter window, discarding older data earlier.

2.4 Ensemble Classifiers

Ensemble Classifiers combine the predictions of a group of base learners, being able to leverage weak learners achieving the performance of a strong learner if trained and combined strategically (GOMES et al., 2017a).

The advantage of having a group over a single learner is explained in the Law of Large Numbers (LLN) (GÉRON, 2022). LLN states that, as the number of experiments increases, the subset mean result gets increasingly close to the hypothetical mean (SEDOR, 2015). If a given classifier has a 51% chance of being correct, after 1000 classifiers the probability that the majority of them is correct is 75%. But this requires them to be independent from each other (GÉRON, 2022).

2.4.1 OzaBag

Bagging is an ensemble learning method that is known to be effective in improving generalization performance (BREIMAN, 1996).

It uses a method of sampling with reposition, allowing the classifiers to train on different variations of subsets of instances from the original data. These subsets are called bags and each one contains all attributes.

The limitation of Bagging is the need to know the size of the training data set beforehand. This information is necessary to determine the size N of the bags.

Online Bagging (OzaBag) (OZA; RUSSELL, 2001) is an online version that requires only one pass through the training data.

Since on the online version the size of the complete data set is not known the bagging works differently. For each learner a value K determines the number of times each instance is used for training.

The K number is determined using Poisson distribution with $\lambda = 1$, resulting in 37% of instances not being used for training, 37% of instances being used for training, and 26% of instances being used for training more than once (GOMES et al., 2017b).

Voting works by counting the prediction of each classifier. The class with the most votes is the prediction of the ensemble.

2.4.2 Adaptive Random Forest

Random Forest is a well-known ensemble classifier consisting of multiple decision trees. These trees are trained on resampled versions of the original data and random subsets of features are inspected at each node split of the trees (GÉRON, 2022).

Adaptive Random Forest (ARF) (GOMES et al., 2017b) is an algorithm that adapts Random Forest in the context of evolving data streams. It uses a resampling method based on Online Bagging and an adaptive strategy of warning detector and drift detector for each tree. The base learners are Hoeffding trees with each leaf split decision made within a random subset of features.

While Online Bagging uses a Poisson distribution with $\lambda = 1$, ARF uses $\lambda = 6$. This adjustment changes the weight each instance contributes during training: approximately 0.25% of instances are not used, 45% are used fewer than 6 times, 16% are used exactly 6 times, and 39% are used more than 6 times. A $\lambda > 1$ benefits the ensemble increasing the diversity of weights and changing the set of instances of all classifiers. The optimal value of λ depends on the data set (BIFET; HOLMES; PFAHRINGER, 2010).

Each tree contains a warning detector and a drift detector. The warning detector, with a low level of confidence, is a trigger for a new tree to be constructed in the background (while the current one is running). If the drift detector, with a higher level of confidence, is triggered the current tree is replaced by the new tree. (BIFET; GAVALDA, 2007)

The strategy for detection is using Adaptive Window (ADWIN) algorithm. ADWIN is a change detector and estimator that analyzes if a change occurred in the average value within a dynamic interval (window) of a stream. An older part of the window is dropped if its average value is different from the rest of the window. The only parameter of ADWIN is the confidence bound δ (BIFET; GAVALDA, 2007).

Prediction is made by counting votes of each base learner. These votes are weighted based on each learner's accuracy on the previous classification

(GOMES et al., 2017b).

2.4.3 Random Subspace Method

The Random Subspace Method (RSM) is an algorithm to construct and aggregate (combine) an ensemble of learners. Each learner is built by randomly selecting subsets of the feature space (HO, 1998).

After every selection is made, a subspace of features is created. All instances are inserted in this subspace and a learner is built using training instances. An instance with an unknown class is projected into the subspace and the learner can make a classification.

Since the RSM is a parallel learning algorithm, the learners are generated independently from each other and generalize their classifications differently.

One particularity of the method is that two distinguishable instances in the original feature space can become ambiguous in a given subspace (HO, 1998).

When a data set contains a relatively small number of instances compared to the number of attributes, building classifiers with different random subspaces may be better than using one classifier with the original feature space. Also, the number of training instances remains the same for both cases.

2.4.4 Streaming Random Patches

Streaming Random Patches (SRP) (GOMES; READ; BIFET, 2019) is an ensemble classifier that combines random subspaces with online bagging. Random Patches means a combination of random subsets of features and instances (LOUPPE; GEURTS, 2012).

Similarly to ARF, it uses warning and drift detectors, being able to create and train background trees once the warning is detected. These background models replace the current ones when the drift is detected. The detectors are also the same as in ARF: Adaptive Window (ADWIN) algorithm (GOMES; READ; BIFET, 2019).

SRP uses a global subspace randomization. The selection of the subspace of features is made globally once for the entire base learner, while in the ARF the selection is made locally for each leaf of the trees (GOMES; READ; BIFET, 2019). Therefore, the available features remain the same until the drift detection, when the tree is replaced. If the subspace contains features that are no longer relevant for classification (due to changes in the stream), the algorithm has to wait for the learner to decay its performance before creating a new subspace.

SRP is described in Algorithm 3. For each base model l a training instance (x, y) is used to evaluate its performance and estimate the weight of voting of each base learner (line 9). This process is the same as in ARF, reducing the influence of underperforming learners on the ensemble's prediction. Next, the learner is trained with the instance (x, y) and a background learner b is created with a new subspace of features if a warning is detected (line 12). This background learner b is trained until a drift is detected, when it replaces the current learner (line 15). This means that the ensemble will start with a model that is not entirely new.

The classification is done by selecting the class with the most votes considering the weight of each model.

Algorithm 3 Streaming Random Patches

Symbols: m: maximum features per subset; λ: Poisson distribution parameter; n: total number of models (n = |L|); δ_w: warning threshold; δ_d: drift threshold; S: Data stream; B: Set of background models; W(l): model l weight; P(.): model predictive performance estimation function; d(.): drift detection method

```
1: function TRAINSRP(m, n, $\delta_w, \delta_d$)
2:     L ← CreateBaseModels(n, m)
3:     W ← InitWeights(n)
4:     B ← {}
5:     while HasNext(S) do
6:         $(x, y)$ ← next(S)
7:         for all l in L do
8:             $\hat{y}$ ← predict(l, x)
9:             W(l) ← P(W(l), $\hat{y}$, y)
10:            Train(m, l, x, y)
11:            if d( $\delta_w$ , l, x, y) then                ▷ Warning detected?
12:                B(l) ← CreateBkgModel(m)
13:            end if
14:            if d( $\delta_d$ , l, x, y) then                ▷ Drift detected?
15:                l ← B(l)                   ▷ Replace l by background learner
16:            end if
17:        end for
18:        for all b in B do
19:            Train(m, b, x, y)
20:        end for
21:    end while
22: end function
```

The creation of the subspace of features is described in Algorithm 4. A random feature *RF* is selected (line 3). If *RF* does not belong in the Feature List *FL*, it is added. This process is repeated until 60% of all Features are present in *FL*. Features contained in *FL* are then used to create the subspace of features.

Algorithm 4 Subspace creation of SRP

Symbols: I: Instance; F: Feature; RF: Random Feature; TF: Total Features; FL: Feature List; SS: Subspace size

Input: All features

Output: Subspace of features

SS: Default Value = 60% of Total Features

```
1: for k ← 1 to SS do
2:     while True do
3:         RF ← SelectRandom(F)
4:         if RF ∉ FL then
5:             break
6:         end if
7:     end while
8:     FL ← Add(RF)
9: end for
```

SRP performs similarly to ARF while having a slightly higher average tree depth and processing time (GOMES; READ; BIFET, 2019). Paim & Enembreck (2025) concluded that the SRP achieves similar classification performance as the ARF, while inducing the development of deep trees, resulting in a significant increase in processing time and memory consumption. This will be confirmed in 6 and the proposed ensemble aims to handle the development of tree-depth.

2.5 Dynamic Feature Scoring in Data Streams

Scoring features based on their relevance regarding classification in supervised learning has been done in batch learning.

Gomes (GOMES et al., 2019) proposed Feature Scoring using Tree-Based Ensembles for Evolving Data Streams. The method is based on Mean Decrease in Impurity (MDI) and COVER, both used on Hoeffding Trees and its variations. MDI estimates the relevance score for a given feature in the total decrease in node impurity averaged across all split nodes in all trees. COVER calculates the relevance score based on the number of instances covered by each node (GOMES et al., 2019).

Gomes et al. (2019) defined the Equations for MDI and Cover. The MDI score of a feature f_i is defined in Equation 2.5, where T_n is the number of trees in the

Forest T and $DI(k)$ is the decrease in impurity (Equation 2.4):

$$MDI\left(f_i\right) = \frac{1}{T_n} \sum_{t \in T} \sum_{k \in t S(k) = f_i} DI(k) \tag{2.5}$$

Cover, which only analyzes the number of instances "covered" by each node, is defined in Equation 2.6:

$$\text{COVER}\left(f_i\right) = \frac{1}{T_n} \sum_{t \in T} \sum_{k \in t S(k) = f_i} N^k \tag{2.6}$$

These two metrics depend on the statistics of the split nodes and the number of instances that reach each split node. In batch learning, both MDI and COVER are calculated after the model is fully developed. Data streams are more complex because both MDI and COVER scores must be evaluated while the model tree is being updated. These metrics are calculated by counters at the leaves for each split node. Leaves contain strong information about the current statistics, as the tree only updates statistics at the leaves (GOMES et al., 2019).

It is important to mention that a given feature that was not used in any split node will have both score values as zero. Therefore, during the early training steps of a Hoeffding Tree, the scores of some features may not be available yet.

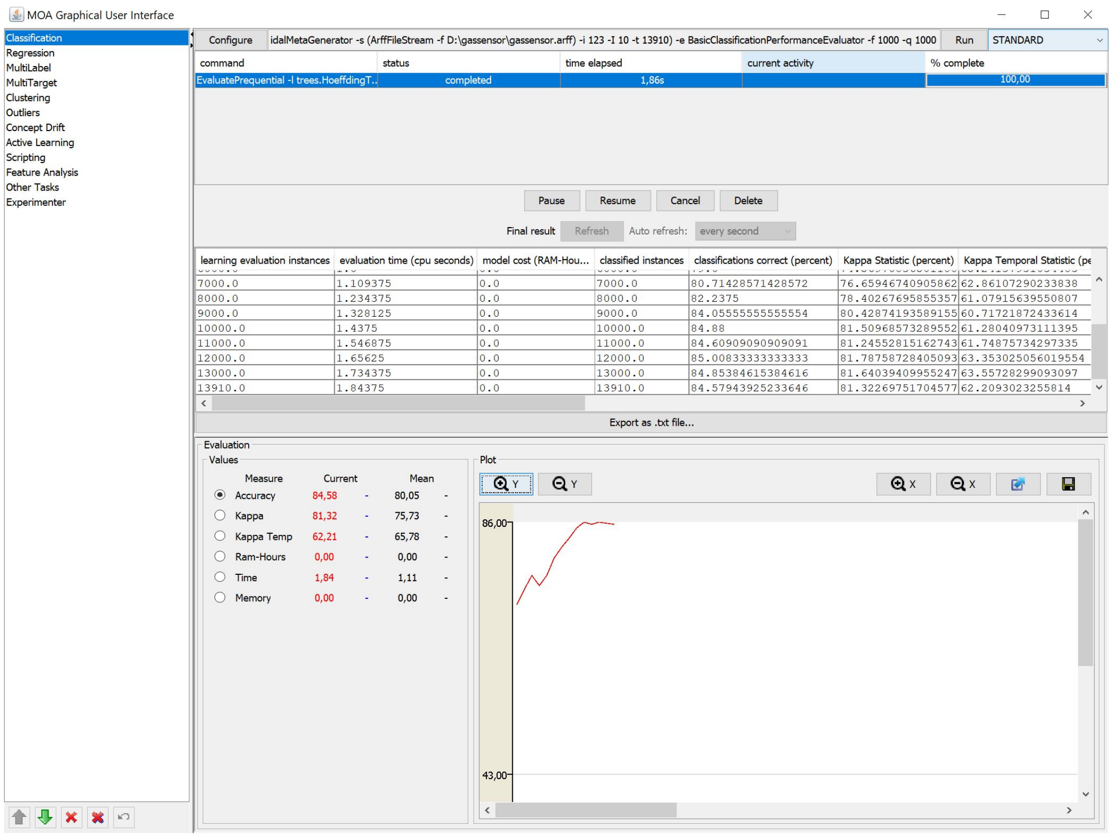
Also, the metrics may not correctly rank the features. The metrics only represent how well the features impact the tree and sometimes they are a consequence of unfortunate splits resulted by biased decisions of the tree algorithm. Removing a feature that was highly ranked by the metrics can improve classification performance (GOMES et al., 2019).

2.6 Massive Online Analysis - MOA

MOA (BIFET et al., 2010) is an open-source framework for data stream mining used in the experiments. It was developed by the University of Waikato to test state-of-the-art algorithms in their ability to make classification, clustering and evaluation. It can run both offline and online data sets.

It is also possible for the users to create and implement their own algorithms and compare them to the MOA's default ones (the current library is pretty extensive). Developed in Java, has compatibility with WEKA (also made by the University of Waikato). The Graphical User Interface for MOA is illustrated in

Figure 2.1.



```
idalMetaGenerator -s (ArffFileStream -f D:\gassensor\gassensor.arff) -i 123 -I 10 -t 13910) -e BasicClassificationPerformanceEvaluator -f 1000 -q 1000
```

command	status	time elapsed	current activity	% complete
EvaluatePrequential -l trees.HoeffdingT..	completed	1,86s		100,00

learning evaluation instances	evaluation time (cpu seconds)	model cost (RAM-Hou...	classified instances	classifications correct (percent)	Kappa Statistic (percent)	Kappa Temporal Statistic (pe
7000.0	1.109375	0.0	7000.0	80.71428571428572	76.65946740905862	62.86107290233838
8000.0	1.234375	0.0	8000.0	82.2375	78.40267695855357	61.07915639550807
9000.0	1.328125	0.0	9000.0	84.05555555555554	80.42874193589155	60.71721872433614
10000.0	1.4375	0.0	10000.0	84.88	81.50968573289552	61.28040973111395
11000.0	1.546875	0.0	11000.0	84.60909090909091	81.24552815162743	61.74875734297335
12000.0	1.65625	0.0	12000.0	85.00833333333333	81.78758728405093	63.353025056019554
13000.0	1.734375	0.0	13000.0	84.85384615384616	81.64039409955247	63.55728299093097
13910.0	1.84375	0.0	13910.0	84.57943925233646	81.32269751704577	62.2093023255814

	Measure	Current		Mean	
●	Accuracy	84,58	-	80,05	-
○	Kappa	81,32	-	75,73	-
○	Kappa Temp	62,21	-	65,78	-
○	Ram-Hours	0,00	-	0,00	-
○	Time	1,84	-	1,11	-
○	Memory	0,00	-	0,00	-

Figure 2.1: MOA Graphical User Interface (GUI).

2.7 Concluding Remarks

In this chapter, we present classification in data streams along with references that will be used later in this project. The algorithms are referenced in chapter 5. The SRP characteristic of deep trees, which is associated with increased processing time and memory consumption, is discussed in Chapters 6 and 7.

3

Varying Feature Spaces in Data Streams

This chapter presents the related works. As explained in Chapter 2, because online learning works without prior knowledge of incoming data, classifiers must be highly adaptable and effective in extracting information from the stream.

Furthermore, online learning carries the complexity of the constant changes in the stream's characteristics. As new instances arrive, they may include features that were not present in previous instances. The opposite scenario can happen: a feature no longer exists in a new instance.

3.1 Literature Review

The related works are divided into: Feature Selection in Streaming Data, Feature Evolution, Trapezoidal Data Streams and Varying Feature Spaces.

Feature Selection is related to this work because the proposed ensemble select features in its subspace and at the first split node (Chapters 5).

Varying feature spaces, a problem in which the dimensionality d increases or decreases during the processing of a data stream is also related. This problem is also referred to as *feature evolution* in the data stream literature (PENG et al., 2021) and it reflects situations where features may appear or disappear .

Trapezoidal Data Streams's concept inspired the creation of the Trapezoidal Meta Generator (Chapters 4).

3.1.1 Feature Selection in Streaming Data

Feature selection is the process of removing redundant and irrelevant features from a given complete feature space. This allows the model to select the "optimal feature subset", improving the prediction accuracy while reducing the computational cost. The focus is on selecting features from the group of existing features. It does not account for the arrival of new features over time or the dynamic nature of feature spaces, which is instead the focus of feature evolution (YOU et al., 2021).

Feature selection techniques are categorized into filter, wrapper, embedded and hybrid methods (WANG; TANG; LIU, 2017):

- Filter methods: select features based on their individual statistical relationship with the target variable, without involving any machine learning model. Features are ranked based on metrics like correlation, mutual information, or entropy;
- Wrapper methods: select features by evaluating how well a model performs with different subsets of features. The objective is to find the best set of features that gives the best model performance;
- Embedded methods: incorporate feature selection into the model training process itself (e.g., online decision trees trained on a set of features select the relevant ones);
- Hybrid methods: combine filter and wrapper.

Markov Blanket

A common filter-based technique used for Feature selection is the Markov Blanket, which leverages Bayesian networks and causal inference to select directly relevant features to predict a specific variable. Usually, the Markov Blanket of a target variable consists of:

- Parents (Direct Causes): features that directly influence the target variable;
- Children (Direct Effects): features that are influenced by the target variable;
- Spouses (Indirect Causes): features that share a child with the target variable, forming an indirect dependency.

All features outside the blanket are considered to provide no additional predictive value.

Online Feature Selection with Streaming Features (OSFSF), which are filter methods, such as in Alpha-investing (ZHOU et al., 2005) and SAOLA (YU et al., 2016), uses Markov Blanket to find the parent feature (direct cause) and child feature (direct effect) of a given target attribute (Figure 3.1). This subset of features is not sufficient for a robust prediction model because its Markov Blanket does not consider the spouse (YOU et al., 2021) and these methods are considered classical online learning: the feature space remains consistent while the instances arrive (WANG; YOU, 2020). Also, in Alpha-investing prior knowledge about the structure of the feature space is needed beforehand, making it difficult for usage in real-world data (WU et al., 2010).

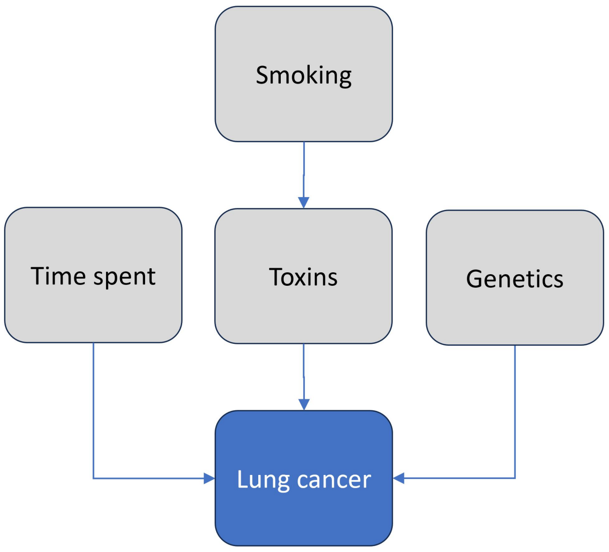


Figure 3.1: Example of Markov Blanket

Online Streaming Feature Selection (OSFS), another filter method, also uses Markov Blanket but filters redundant and irrelevant features (WU et al., 2010). But if used in very limited-size datasets, conditional independence tests are unreliable, affecting the functioning of OSFS. Also, if the stream contains features with low redundancy and high relevance, the accuracy of the model will be affected (WANG; YOU, 2020).

Processing time increases exponentially with the increase of a number of features and similarly to Alpha-investing, OSFS can benefit from having prior knowledge of the space of features. Having this information earlier in the stream, it is easier to find the strongly relevant features and eliminate the redundant features. This is not possible in real-world data and OSFS is not considered related to this project.

The Online Streaming Features Selection via Markov Blanket (OFSVMB) was proposed as an improvement in the feature selection methods that use traditional Markov Blanket. Compared to those, the number of conditional independence tests is reduced, the relevance and redundant analysis are made online and the parent, child and spouse of the target feature are found simultaneously. OFSVMB is a filter method, but it doesn't just consider individual features independently like the previous methods. Alternatively, it builds a causal structure with interactions between parents, children and spouses, creating model-aware structural reasoning (KHAN et al., 2022).

The OFSVMB improved mean prediction accuracy on real-world datasets compared to previous methods like OSFS, Alpha-investing and SAOLA. These algorithms only search the parent and child of the target feature. OFSVMB, however, has a slower running time because it searches for the spouse, parent and child while OSFS, Alpha-investing and SAOLA only search for the parent and child of the target feature. (KHAN et al., 2022).

Dynamic Feature Selection

Dynamic Feature Selection is the process of real-time selection of relevant features from a continuous stream of incoming data.

Dynamic Feature Selection is different from Online Feature Selection (Streaming Feature Selection). Instead of having a stream of features as input and selecting the subset of features from a high-dimensional space in a batch learning scenario, Dynamic Feature Selection receives input of instances, selecting features as new examples arrive (BARDDAL et al., 2019). The selection of features is based on currently available data, instead of using a fixed feature set (COVERT et al., 2023).

Iterative Subset Selection (ISS) is a supervised filter-wrapper method for dynamic feature selection. In each sliding window of instances, features are ranked based on their relevance to the classification task. The rank is done using metrics such as Information Gain, Average Euclidean Distance, or Symmetric Uncertainty. An initial subset of the top-f features is selected and the algorithm applies backward elimination—iteratively removing the least relevant feature and evaluating the model's accuracy using a decaying average. The subset that maximizes accuracy is selected. ISS adapts to both sudden and gradual changes in feature relevance by periodically re-ranking features. It is compatible with classifiers like k-Nearest Neighbors and Naïve Bayes, maintaining model accuracy in the presence of concept and feature drift (YUAN; PFAHRINGER; BARDDAL, 2018).

Adaptive Boosting for Feature Selection (ABFS) is an embedded method of dynamic feature selection combining chains of decision stumps and drift detectors (BARDDAL et al., 2019). Each decision stump, which is a one-level Hoeffding Tree, influences the creation of the next one by prioritizing instances that were hard to classify (GÉRON, 2022). Prioritization is done by adjusting the weight which influences the training of a candidate decision stump. Every instance is passed through the entire chain and if one decision stump is no longer maintaining classification performance, all subsequent ones will also be affected. For that reason, a drift detector was implemented to monitor the chain and to remove all decision stumps starting from the first one to misclassify onward. If no drift was detected, a candidate decision stump is trained on the instance and if this candidate selects a feature, this feature is added to a subset of features while the candidate is added to the chain. Finally, ABFS returns the instance filtering only the features from the subset of features. This is then passed to a classifier, since ABFS is not used for classification.

Online Fast Feature Selection (OFFESEL) is an unsupervised filter method for dynamic feature selection. In each sliding window of instances, the values of all features are first normalized to the range [0, 1], and outliers are removed using the Interquartile Range (IQR) method. Then, the average value of each feature is calculated. OFFESEL ranks the features based on how skewed their normalized values are — how far their average is from 0.5. Features with very high or very low averages are assumed to be more informative, as they are more likely to help separate the classes. The rank is used to select the features (HOCHMA; LAST, 2025).

3.1.2 Feature Evolution

In Feature Evolution, the feature space changes over time and the model adapts accordingly. This change is not abrupt and there is a transition period during which features from before and after the change exist simultaneously. While Feature Selection chooses the most relevant features at a given moment, Feature Evolution focuses on adapting to a changing feature space, deciding in real-time whether to keep, discard, or integrate new features.

Feature Evolvable Streaming Learning (FESL) was made to work with data streams in which old features vanish and new ones emerge. For example, in object recognition, many sensors will deteriorate and new ones will be deployed (HOU; ZHANG; ZHOU, 2017).

If there are limited samples described by the new features, and samples described by the old features are ignored, FESL assumes that there exists an overlapping period during which both old and new features can be used. Also, all old features vanish simultaneously.

Prediction with Unpredictable Feature Evolution (PUFE) is similar to FESL but has the assumption that old features do not vanish simultaneously. To tackle this problem, PUFE imputes missing values using matrix completion in which samples are observed without replacement. This allows the algorithm to establish a connection between the features that are vanishing in the old space and the emerging features in the new space (HOU; ZHANG; ZHOU, 2021).

The disadvantage of both FESL and PUFE is that it does not function in a scenario in which the period of old features is separated from the period of new features. It needs to exist an overlapping of both.

Dynamic Feature Space data streams assume that each instance can have different features and the feature spaces of the learner can also be different from the instances. Dynamic feature space using feature mapping (DCDF2M) was proposed as a general classifier and works by estimating the values for the unavailable features. This is done using feature mapping, a polynomial regression to analyze the relationship between pairs of features. For each pair of features, the co-occurrence of the features and the quality of the mappings are calculated. DCDF2M has a higher accuracy performance than OLVF but with a significantly higher processing time and memory usage (SAJEDI; RAZZAZI, 2024).

Generative Learning with Streaming Capricious data (GLSC) also estimates missing features, but is different from DCDF2M, because it assumes relationships between pairs of features to be not independent from each other. Also, while in DCDF2M all features have polynomial relations, in GLSC features have linear relationships with each other (HE et al., 2020).

3.1.3 Trapezoidal Data Streams

The situation of continuous learning from a data stream where both data volume and feature space size increase over time is called trapezoidal data streams. The doubly-streaming data nature of these streams is more complex than existing online learning problems (ZHANG et al., 2016).

A real-world example of trapezoidal streams, having simultaneously an increase in both data volume and feature dimension, is the graph node classification in the ego-network structure from a social network. Also, in text classification the number of documents and text increase over time (ZHANG et al., 2016).

Online Learning with Streaming Features (OL_{SF}) is an algorithm created to handle trapezoidal data streams that update its classifier's weight if the prediction is incorrect. New additional weights for the new features are learned.

OL_{SF} and its variations are not able to work with varying feature spaces when the feature space shrinks and new instances arrive with completely different sets of features (BEYAZIT; ALAGURAJAH; WU, 2019). Therefore, it is not related to this project.

3.1.4 Varying Feature Spaces

While in Trapezoidal Data Streams the feature space only grows over time, in varying feature spaces, the feature space can change arbitrarily — features may be added, removed, or even reappear. It is not possible to make assumptions about the feature space (BEYAZIT; ALAGURAJAH; WU, 2019).

The Online Learning from Data Streams with Varying Feature Spaces (OLVF) algorithm uses a projection of an existing classifier and current instance into the feature subspace. A prediction is made and a loss function is calculated. The classifier is updated based on an empirical risk minimization principle with adaptive constraints and the algorithms re-weigh the constraints with the classifier's confidence (BEYAZIT; ALAGURAJAH; WU, 2019).

The empirical risk minimization principle works by calculating the difference between the predicted label and the true label of a training instance: the algorithm updates the instance classifier by minimizing the empirical risk represented by the average loss over all training instances.

The projection confidence is a way to measure the loss of information after projecting a given instance onto a feature space. It is the probability that the given classifier makes a correct prediction (BEYAZIT; ALAGURAJAH; WU, 2019).

At the end of each update, sparsity is made to reduce classifier size and remove less relevant features.

Compared to OL_{SF}, the OLVF has better prediction accuracy while having learning classifiers with the same sparsity. This is valid for batch learning datasets and real-world Varying Feature Space (IMDB Movie Review dataset) (BEYAZIT; ALAGURAJAH; WU, 2019).

Although the OLVF uses a complex method that allows a classifier to make wrong predictions and adapt (changing its constraints), the selection of features to be included in each subspace of features is random. This project, on the other hand, selects the features to be included in the subspace based on importance ranking.

Nonetheless, this project uses that same method to simulate varying feature spaces as OLVF's removing ratio permutation.

3.2 Concluding Remarks

In this chapter, previous works related to Varying Feature Spaces in Data Streams were presented. It is important to emphasize that these works do not confirm or deny the hypotheses of this project. None of the works of this chapter uses a method of selecting features to be included in the subspace of features based on ranking of importance.

Furthermore, the utility of these methods can be questioned for use in ensembles, considering that state-of-the-art algorithms for classification in data streams incorporate embedded feature selection. Therefore, it is essential to consider the development of Varying Feature Spaces in Data Streams methods that are effective for algorithms producing classification ensembles in data streams.

Part II

Contributions

4 Simulating Varying Feature Spaces in Data Stream

In Varying Feature Spaces a dataset has features that can appear or disappear as new instances arrive. An example is data regarding the educational sector during the pandemic period: schools were temporarily closed and some activities were interrupted. This means that there are missing data from that period.

There are no public datasets made specifically reproducing Varying Feature Spaces data streams. This creates a problem due to the lack of availability of these datasets.

Although experiments on Trapezoidal data stream using the IMDB Movie Review dataset were made in (BEYAZIT; ALAGURAJAH; WU, 2019), the dataset has simultaneously new features appearing and old ones disappearing. This project, on the other hand, experiments on Varying Feature Spaces scenarios that either the number of features increase, decrease or both happening asynchronously.

This means that there is never a moment when a feature is being included while another is being deleted. This is done to clearly isolate the scenarios. By modifying the feature space one process at a time, it is possible to assess how each change (e.g. Increment and Removal) affects the model performance. In contrast, making changes simultaneously would make it difficult to distinguish the impact of each individual scenario or an interaction between both.

A solution proposed by the author is to create a Varying Feature Spaces

Data streams generator. This generator treats non-existing features at any given moment as missing data. This process was done similarly to the varying feature space by Beyazit, Alagurajah & Wu (2019) explained in Chapter 3.

4.1 Trapezoidal Meta Generator

Trapezoidal Meta Generator is a framework created on MOA that takes a complete data stream and systematically removes some features in order to simulate the Varying Feature Spaces scenario. There are four scenarios: Increment, Removal, Parabolic and Hyperbolic.

These scenarios are described as follows.

4.1.1 Increment

Increment is similar to the Trapezoidal Data Streams proposed by Zhang et al. (2016). It is a stream in which both data volume and number of dimensions increase over time.

A set containing all features to be deleted FD is initialized (Algorithm 5). This set is used in all scenarios to represent features that are unavailable. As FD increases, the amount of features available for inclusion in the stream decreases.

Algorithm 5 Creating Total Features

```
    Symbols: F: Feature; S: Stream; FD: Features To Delete
1: for all F in S do
2:     FD ← Add(F)
3: end for
```

The description is on Algorithm 6. For every instance I of the Stream S, a number DSA of randomly selected features are removed from FD if the number of Instances Seen IS achieved the Space Change Period SCP (lines 2-6). Next, all Features from FD are removed from the I (lines 7-11). Therefore, as the stream progresses the size of FD reduces and the number of available features increases.

Algorithm 6 Trapezoidal Meta Generator: Increment

Symbols: `I`: Instance; `IS`: Instances Seen; `F`: Feature; `S`: Stream; `SCP`: Space Change Period; `DSA`: Delete Set Amount; `FD`: Features To Delete

Input: Stream with full features

Output: Stream with Varying Feature Spaces

```
1:  for all I in S do
2:      if (IS % SCP) = 0 then
3:          for k ← 1 to DSA do
4:              FD ← Remove(Random(F))
5:          end for
6:      end if
7:      if FD ≠ null then                    ▹ Remove features from instance
8:          for all F in FD do
9:              Instance ← Remove(F)
10:         end for
11:     end if
12:     Update IS
13: end for
```

An example of a stream being transformed into an Increment stream with 10 features is represented in Figure 4.1. For every instance all features contained in FD are deleted, allowing control of the null elements within the instance. Once the number of instances seen (IS) is divisible by Space Change Period (SCP), a quantity (DSA) of features is deleted from FD, increasing the number of non-null elements.

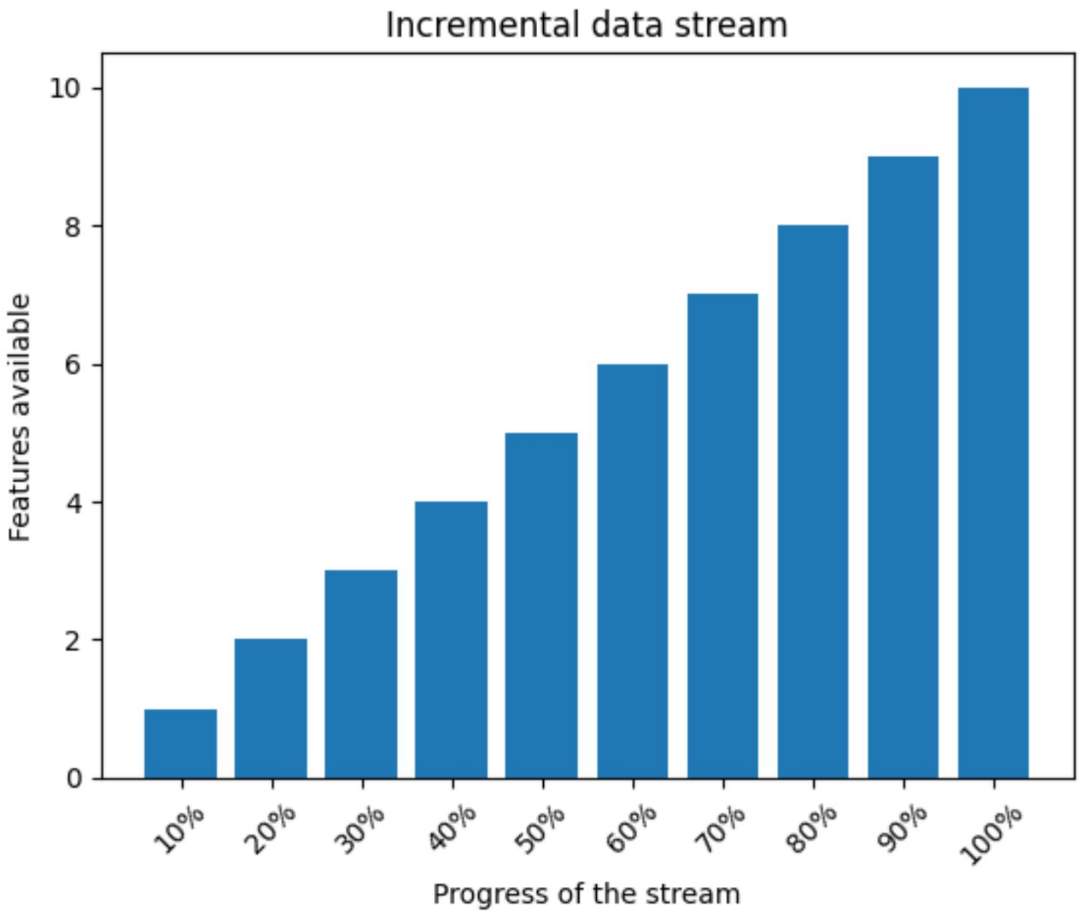


Figure 4.1: INCREMENT Data Stream.

4.1.2 Removal

Removal is the opposite of the Increment scenario: the number of available features decreases (Algorithm 7).

A set containing all features to be deleted FD is initialized (Algorithm 5).

For every instance I of the Stream S, a number DSA of randomly selected features are added to FD if the number of Instances Seen IS achieved the Space Change Period SCP (lines 2-12). Next, all Features from FD are removed from the I (lines 13-17). Therefore, as the stream progresses the size of FD increases and the number of available features decreases.

Algorithm 7 Trapezoidal Meta Generator: Removal

Symbols: `I`: Instance; `IS`: Instances Seen; `F`: Feature; `RF`: Random Feature; `S`: Stream; `SCP`: Space Change Period; `DSA`: Delete Set Amount; `FD`: Features To Delete

Input: Stream with full features

Output: Stream with Varying Feature Spaces

```
1: for all I in S do
2:     if (IS % SCP) = 0 then
3:         for k ← 1 to DSA do
4:             while True do
5:                 RF ← SelectRandom(F)
6:                 if RF ∉ FD then
7:                     FD ← Add(RF)
8:                     break
9:                 end if
10:            end while
11:        end for
12:    end if
13:    if FD ≠ null then                          ▷ Remove features from instance
14:        for all F in FD do
15:            Instance ← Remove(F)
16:        end for
17:    end if
18:    Update IS
19: end for
```

A representation of a Removal stream containing 10 features is in Figure 4.2.

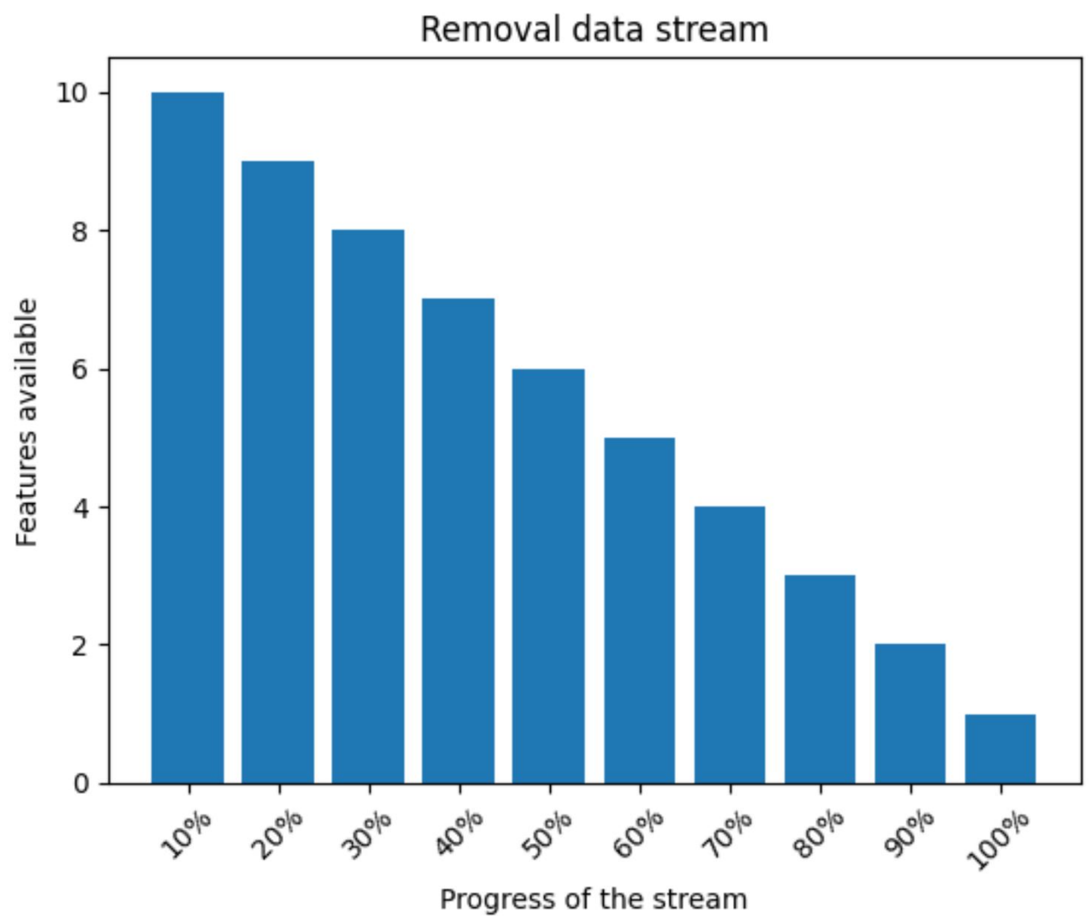


Figure 4.2: REMOVAL Data Stream.

4.1.3 Parabolic

Parabolic combines Increment and Removal. The first half of the stream has an increment of features, while in the second half the number of features decreases.

First, a set containing all features to be deleted FD is initialized (Algorithm 5).

Parabolic is described on Algorithms 8 and 9. For every instance I of the stream S, the number of Instances Seen IS is checked to determine if it has reached the Space Change Period SCP (line 3). Then, a tolerance margin around the midpoint of the instance stream is calculated based on half of the total number of instances HP and the percentage of the feature space change period $DESV$. This margin establishes an interval around the center where no changes occur to the feature set (line 4).

Next, if the stream is at the first half (IVT is set to False), a number DSA of randomly selected features is removed from FD (lines 5-10). Else, a number DSA of randomly selected features is added to FD (lines 11-22). If FD is not null, all Features from FD are removed from the Instance (lines 29-33).

Algorithm 8 Trapezoidal Meta Generator: Parabolic (part 1)

Symbols: I: Instance; IS: Instances Seen; TI: Total instances; F: Feature; TF: Total Features; RF: Random Feature; S: Stream; SS: Stream size; SCP: Space Change Period; DSA: Delete Set Amount; FD: Features To Delete; IVT: Inverter; HP: HalfWayPoint; DESV: Deviation

Input: Stream with full features

Output: Stream with Varying Feature Spaces

```
 1: IVT ← False
 2: for all I in S do
 3:     if (IS % SCP) = 0 then
 4:         if (IS < HP − DESV) ∨ (IS > HP + DESV) then
 5:             if IVT = False then                       ▷ Decrease number of features
 6:                 for k ← 1 to DSA do
 7:                     if SizeOf(FD) > 0 then
 8:                         FD ← Remove(Random(F))
 9:                     end if
10:                 end for
11:             else                                      ▷ Increase number of features
12:                 if SizeOf(FD) ≠ TF then
13:                     for k ← 1 to DSA do
14:                         while SizeOf(FD) ≠ TF do
15:                             RF ← SelectRandom(F)
16:                             if RF ∉ FD then
17:                                 FD ← Add(RF)
18:                                 Break
19:                             end if
20:                         end while
21:                     end for
22:                 end if
23:             end if
24:         end if
25:     end if
```

Algorithm 9 Trapezoidal Meta Generator: Parabolic (part 2)

```
26:     if IS $\geq$ HP $-$ 1 then
27:         IVT $\leftarrow$ True
28:     end if
29:     if FD $\neq$ null then                ▷ Remove features from instance
30:         for all F in FD do
31:             Instance $\leftarrow$ Remove(F)
32:         end for
33:     end if
34:     Update IS
35: end for
```

A 10 feature Parabolic stream is illustrated in Figure 4.3.

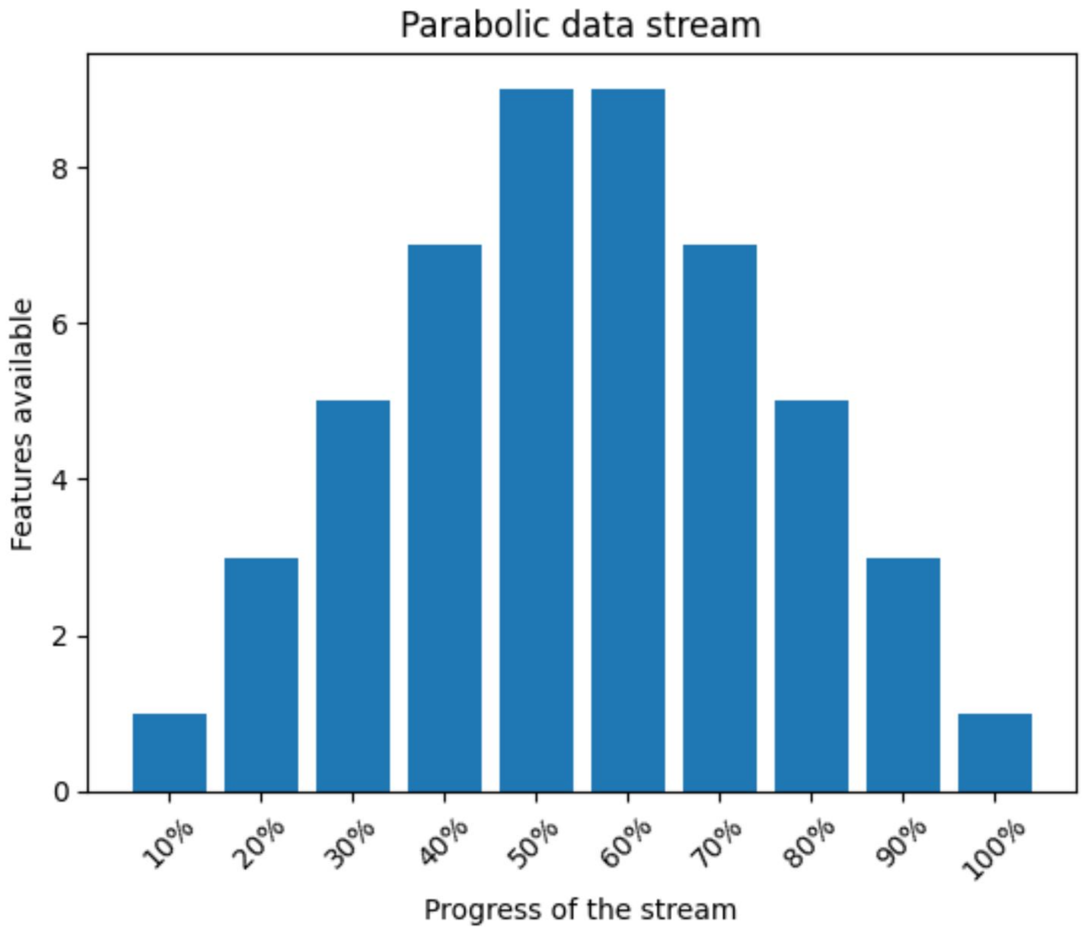


Figure 4.3: PARABOLIC Data Stream.

4.1.4 Hyperbolic

Hyperbolic also combines Increment and Removal. But having a decrease in the number of features at the first half of the stream and an increase in the rest. First, a set containing all features to be deleted FD is initialized (Algorithm 5).

Hyperbolic is described on Algorithms 10 and 11. For every instance I of the stream S, the number of Instances Seen IS is checked to determine if it has reached the Space Change Period SCP (line 3). A tolerance margin is used exactly as in Parabolic (line 4). If the stream is at the first half (IVT is set to False), a number DSA of randomly selected features is added to FD (lines 5-16). Else, a number DSA of randomly selected features is removed from FD (lines 17-23). Then, If FD is not null, all Features from FD are removed from the Instance (lines 29-33).

Algorithm 10 Trapezoidal Meta Generator: Hyperbolic (part 1)

Symbols: `I`: Instance; `IS`: Instances Seen; `TI`: Total instances; `F`: Feature; `TF`: Total Features; `RF`: Random Feature; `S`: Stream; `SS`: Stream size; `SCP`: Space Change Period; `DSA`: Delete Set Amount; `FD`: Features To Delete; `IVT`: Inverter; `HP`: HalfWayPoint; `DESV`: Deviation

Input: Stream with full features

Output: Stream with Varying Feature Spaces

```
1:  IVT ← False
2:  for all I in S do
3:      if (IS % SCP) = 0 then
4:          if (IS < HP − DESV) ∨ (IS > HP + DESV) then
5:              if IVT = False then                          ▹ Decrease number of features
6:                  if SizeOf(FD) ≠ TF then
7:                      for k ← 1 to DSA do
8:                          while SizeOf(FD) ≠ TF do
9:                              RF ← SelectRandom(F)
10:                             if RF ∉ FD then
11:                                 FD ← Add(RF)
12:                                 Break
13:                             end if
14:                         end while
15:                     end for
16:                 end if
17:             else                                         ▹ Increase number of features
18:                 for k ← 1 to DSA do
19:                     if SizeOf(FD) > 0 then
20:                         FD ← Remove(Random(F))
21:                     end if
22:                 end for
23:             end if
24:         end if
25:     end if
26:     if IS ≥ HP − 1 then
27:         IVT ← True
28:     end if
```

Algorithm 11 Trapezoidal Meta Generator: Hyperbolic (part 2)

```
29:     if FD ≠ null then                        ▷ Remove features from instance
30:         for all F in FD do
31:             Instance ← Remove(F)
32:         end for
33:     end if
34:     Update IS
35: end for
```

Figure 4.4 represents a 10 features Hyperbolic stream. The tolerance margin allows no changes to the feature subspace to occur in the middle of the stream.

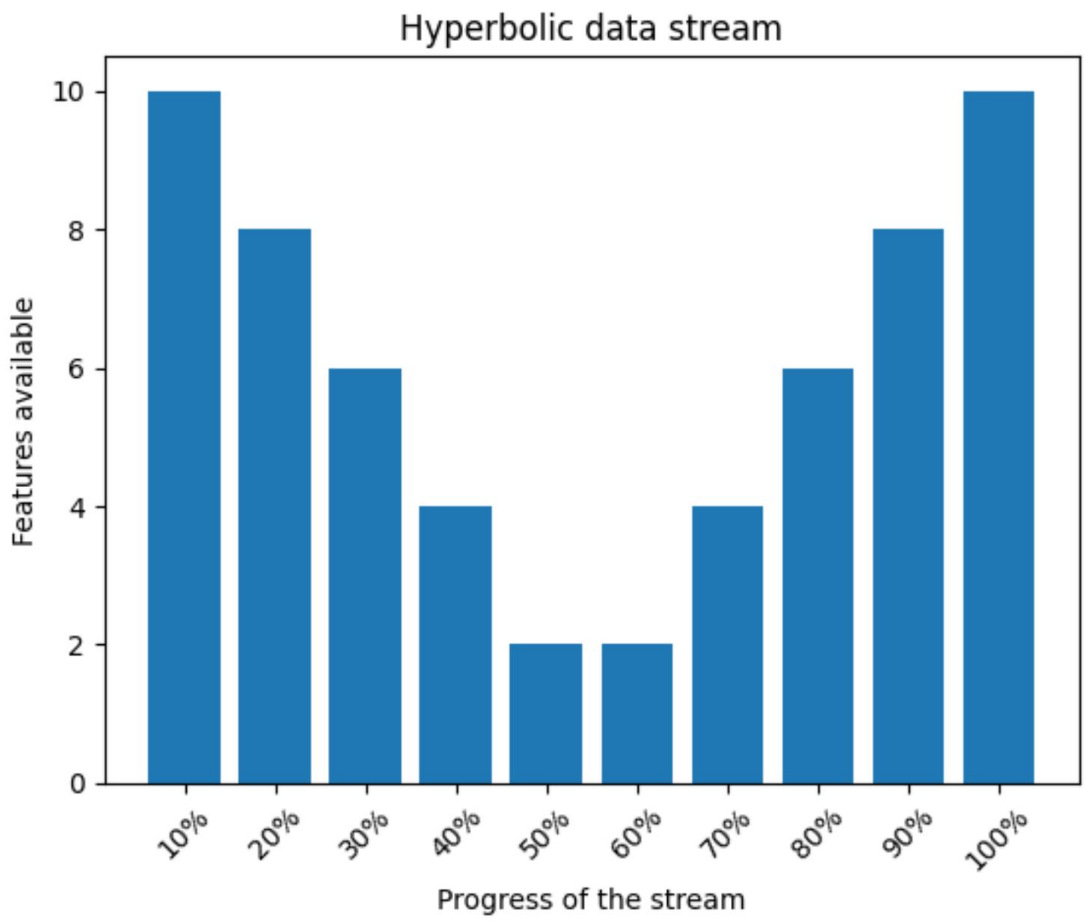


Figure 4.4: HYPERBOLIC Data Stream.

4.2 Concluding Remarks

In this chapter, we present the Trapezoidal Meta Generator. The generator and the 4 scenarios (Increment, Removal, Parabolic and Hyperbolic) are referenced in chapter 6 and 7.

5 Feature Importance Ensembles in Streaming Feature scenarios

This chapter presents a modified ensemble that combines the feature scoring with the Streaming Random Patches, called Feature Importance Streaming Random Patches (FISRP).

The proposed ensemble uses feature importance ranking to select features for their subspaces and an active strategy of replacing learners based on the features contained in their subspaces. This is different from Streaming Random Patches, which randomly selects features for the subspaces and employs a reactive strategy, having its learner's performance decay until a drift is detected and replacement occurs. This period between the learner's subspace having features that are no longer relevant for classification and the detection of drift could be explored through the active replacing with a learner containing selected features in its subspace.

5.1 Proposed method

FISRP combines the group of base learners from SRP with a method of selecting features to be present in their subspaces using ranking from Feature Scoring using Tree-Based Ensembles for Evolving Data Streams (GOMES et al., 2019). A feature ranking learner is used to create a list of the highest-ranking features. This learner is not used for classification and is trained in the same instances as the rest of the base learners.

While SRP uses a reactive strategy of creating background learners and replacing the current ones (using warning and drift detection respectively), FISRP integrates this with an active strategy: replacing learners in which their subspaces do not contain the features from the highest-ranking list. Also, each base learner is created forcing the split with a feature from the ranking.

FISRP is represented in Figure 5.1. First, the ensemble is initialized. All feature subspaces are created as described in Algorithm 17. For each instance, a verification is made (Algorithm 12), the feature ranking learner is trained, and its ranking is updated (Algorithm **??**). In each base learner, the instance is used for testing and training. Then, the warning verification is made: if this returns false, a second verification occurs (Algorithm 16). Finally, the drift verification is made.

This section is divided into: arrival of instance and creation of feature subspace and learner. There are references to the term "null". In MOA, "null" represents the absence of value, which is different from an empty string or zero.

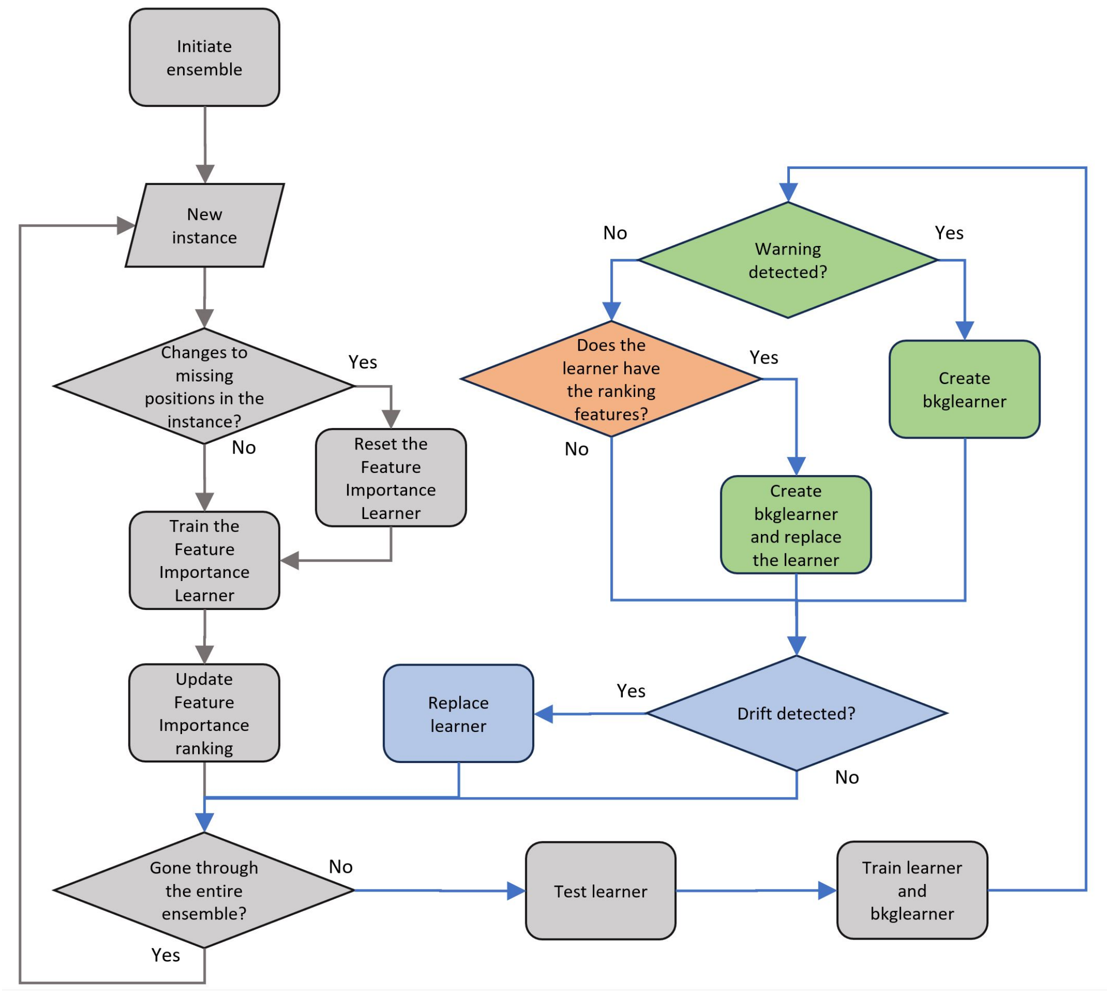

Initiate ensemble
New instance
Changes to missing positions in the instance?
Yes
No
Reset the Feature Importance Learner
Train the Feature Importance Learner
Update Feature Importance ranking
Gone through the entire ensemble?
No
Yes
Test learner
Train learner and bkglearner
Warning detected?
No
Yes
Does the learner have the ranking features?
Yes
No
Create bkglearner
Create bkglearner and replace the learner
Drift detected?
Yes
No
Replace learner

Figure 5.1: FISRP algorithm

5.1.1 Arrival of a new instance

Every instance is checked for changes in its missing positions (i.e. positions of elements whose value is null), as explained in Algorithm 12. This is a comparison between the previous instance and the current one. If such changes are detected, the feature ranking learner is reset, since it may no longer be relevant as the feature space may change.

Even a single missing position that has changed will cause the reset of the feature ranking learner. Regardless, the feature ranking learner is trained on the new instance (Algorithm 14, lines 1-7).

Algorithm 12 Changes to missing positions in the instance

```
Symbols: I: Instance; V: Value; F: Feature; CIA: Current Instance Array; PIA: Previous Instance Array
Input: Instance
Output: Changes to missing positions flag
1: ChangeDetected ← False
2: for all V in ($x^t$) do                                  ▷ Current Instance Array
3:     CIA ← Add(V)
4: end for
5: for all V in ($x^{t-1}$) do                              ▷ Previous Instance Array
6:     PIA ← Add(V)
7: end for
8: if PIA is not null then
9:     for k ← 0 to (SizeOf(PIA)-1) do
10:        usedToBeNull ← PIA.get(k).isNaN
11:        isNull ← CIA.get(k).isNaN
12:        if usedToBeNull ≠ isNull then        ▷ Changes to missing positions in the instance detected?
13:            ChangeDetected ← True
14:            break
15:        end if
16:    end for
17: end if                                                  ▷ Store Current Instance Array
18: Return ChangeDetected
```

The size of the highest-ranking features list is limited by featuresToPreserve (Algorithm 13), which is determined by maxRelevantFeaturesPercentage. The latter has a default value of 20%.

The reason for limiting the list is to ensure that randomly selected features remain in the subspace: if the size of the list reaches 40% of the total features, all subspaces would contain the same features (Algorithm 17, lines 4-20). It is important to still have random features to ensure diversity in the ensemble.

Algorithm 13 Features to preserve

```
Symbols: I: Instance
Input: Instance, maxRelevantFeaturesPercentage
Output: featuresToPreserve
1: featuresToPreserve ← numAttributes(I) × maxRelevantFeaturesPercentage
2: if featuresToPreserve ≥ numAttributes(I) − 1 then
3:     featuresToPreserve ← numAttributes(I) − 1
4: end if
```

Algorithm 14 Feature Importance ranking update (part 1)

Symbols: `I`: Instance; `V`: Value; `F`: Feature; `TF`: Total Features; `FIL`: Feature Importance Learner; `CFA`: Current Features Array; `IA`: Importance Array; `PFA`: Previous Features Array; `HRFL`: Highest-Ranking Feature List

Input: Instance, ChangeDetected, featuresToPreserve

Output: Features Ranking list

```
1: if ChangeDetected = True then
2:     Reset(FIL)
3: end if
4: if FIL = Null then
5:     Initialize(FIL)
6: end if
7: TrainOnInstance(FIL)
8: for i ← 0 to (TF-1) do
9:     CFA ← Add(FIL.topKFeatures(i))    ▷ Creation of Feature Ranking
10: end for
11: for j ← 0 to (TF-1) do
12:     IA ← Add(FIL.featureImportance(j))    ▷ Importance of each Feature Array
13: end for
14: if PFA ≠ Null then
15:     if PFA ≠ CFA then
16:         for k ← 0 to (SizeOf(CFA)-1) do
17:             if CFA.get(k) ∉ HRFL then
18:                 HRFL ← CFA.get(k)
19:             end if
20:         end for
21:         for l ← 0 to (SizeOf(IA)-1) do
22:             if IA.get(l) ≠ 0 then
23:                 if l ∈ HRFL then
24:                     Remove(valueOf(l) from HRFL)    ▷ Remove Features with zero relevance from HRFL
25:                 end if
26:             end if
27:         end for
```

Algorithm 15 Feature Importance ranking update (part 2)

```
28:         if SizeOf(HRFL) $\geq$ featuresToPreserve then
29:             HRFL.subList(featuresToPreserve, HRFL.size()).clear() ▷ Reduce HRFL size to featuresToPreserve
30:         end if
31:     end if
32: end if
```

All base learners in the ensemble train on the new instance. For each base learner, if a warning is detected, a background learner is created. If a drift is detected, the background learner replaces the main one.

The process is the same as SRP, the difference is if no warning is detected an additional verification is made (Algorithm 16). If the learner's subspace does not have the minimum amount of features from the highest-ranking features list, it is replaced. The amount is determined by featuresThresholdPercentage and its default value is 50%. This rationale creates an active strategy, replacing learners with potentially obsolete knowledge.

Algorithm 16 Verification if the learner contains ranking Feature

Symbols: F: Feature; CFA: Current Features Array; PFA: Previous Features Array; HRFL: Highest-Ranking Feature List

Input: Learner, instanceSeen, featuresThresholdPercentage, CFA, PFA, HRFL

Output: UpdateLearner Flag

```
1: UpdateLearner ← False
2: if CFA ≠ Null ∧ PFA ≠ Null ∧ PFA ≠ CFA ∧ instanceSeen ≥ Learner.CreatedOn then
3:     threshold ← featuresThresholdPercentage          ▷ Defined by user
4:     for all F in Learner.FeatureSpace do
5:         CurrentFeatures ← Add(F)
6:     end for
7:     Count ← 0
8:     for all F in CurrentFeatures do
9:         if F ∈ HRFL then
10:            Count ← Count + 1
11:        end if
12:    end for
13:    Percentage ← Count / SizeOf(HRFL)
14:    if Percentage < threshold then
15:        UpdateLearner ← True
16:    end if
17: end if
18: Return UpdateLearner
```

5.1.2 Creation of Feature Subspace and learner

While the original SRP selects each subspace of features randomly (described on Algorithm 4), FISRP creates its subspaces by combining feature importance ranking and random selection (similarly to SRP).

The creation of the Subspace of features of FISRP is described on Algorithm 17. First, all features are added to the Feature List FL (lines 1-3). If the highest-ranking features list $HRFL$ is null, a random feature is removed from FL (lines 5-7). Otherwise, the prioritization of feature is done: a random feature is selected until it is present in FL but not in $HRFL$. Then, it is removed from FL (lines

9-17). The process is repeated until the Subspace size SS is reached.

If the highest-ranking features list is empty, FISRP will create its subspace identically as SRP, making a random selection. This means that FISRP depends on the rate and quality of the Feature ranking learner: it must perform fast enough while selecting relevant features. The Feature Importance ranking Learner used to create the feature ranking list is a Hoeffding Adaptive Tree (HAT) because it is able to reset nodes.

Algorithm 17 Subspace creation of FISRP

Symbols: I: Instance; F: Feature; RF: Random Feature; TF: Total Features; FL: Feature List; HRFL: Highest-Ranking Feature List; SS: Subspace size

Input: All features

Output: Subspace of features

SS: Default Value = 60% of Total Features

HRFL: Features selected by highest-ranking features list

```
1:  for all F in TF do
2:      FL ← Add(F)
3:  end for
4:  for k ← 1 to (TF - SS) do
5:      if HRFL is null or SizeOf(HRFL) = 0 then
6:          RF ← SelectRandom(F)
7:          FL ← Remove(RF)
8:      else
9:          while True do
10:             if SizeOf(HRFL) = SizeOf(FL) then
11:                 break
12:             end if
13:             RF ← SelectRandom(F)
14:             if RF ∉ HRFL and RF ∈ FL then
15:                 FL ← Remove(RF)
16:                 break
17:             end if
18:         end while
19:     end if
20: end for
```

FISRP uses a variation of an HT, FISRP Hoeffding Tree (FISRP HT) as the base learner. This variation has a different process for the first node split.

The creation of bestSplitSuggestions (BSS) of an HT was explained in Algorithm 1 (chapter 2). BSS is a list containing the $decreaseinimpurity(DI(k))$ or $merit$ for all features in a feature space. Algorithm 18 represents the creation of BSS for a FISRP HT. The difference is that at the first level of Tree Depth, if a feature from the feature subspace is present in the highest-ranking features list its $merit$ is added to a second list, BSFBF. Else, it is added to the BSS list.

After going through the entire feature space, if BSFBF contains at least one element, BSS is reset and receives all elements from BSFBF. If there is no overlap between the subspace and the highest-ranking features list, all features from the subspace are copied to BSS.

After the first level of Tree Depth, the process of creating BSS is identical to the HT (Algorithm 1, chapter 2).

Algorithm 18 Creation of bestSplitSuggestions of a FISRP Hoeffding Tree

Symbols: BSS: bestSplitSuggestions; BSFBF: bestSuggestionsfromBestFeatures; HRFL: Highest-Ranking Feature List

Input: Learner; Highest-Ranking Feature List

Output: BSS Flag

```
1: if Learner.TreeDepth $> 0$ then
2:     for all F in Learner.FeatureSpace do
3:         bestSuggestion $\leftarrow$ Add(AttributeClassObserver.getMerit(F))
4:         if bestSuggestion $\neq$ Null then
5:             BSS $\leftarrow$ Add(bestSuggestion)
6:         end if
7:     end for
8: else
9:     for all F in Learner.FeatureSpace do
10:        bestSuggestion $\leftarrow$ Add(AttributeClassObserver.getMerit(F))
11:        if bestSuggestion $\neq$ Null then
12:            if HRFL $\neq$ Null $\wedge$ HRFL.contains(F) then
13:                BSFBF $\leftarrow$ Add(bestSuggestion)
14:            else
15:                BSS $\leftarrow$ Add(bestSuggestion)
16:            end if
17:        end if
18:    end for
19:    if SizeOf(BSFBF) $> 0$ then
20:        BSS $\leftarrow$ CopyOf(BSFBF)
21:    end if
22: end if
23: Return BSS
```

Although the creation of BSS in a FISRP HT is influenced by the highest-ranking feature list (only at the first level of Tree Depth), the selection of the feature for splitting is identical to that of a HT (Algorithm 2 of chapter 2): the one with the highest $merit$ value within BSS. This implementation is important because it allows the ensemble to initialize the trees with a strong feature while maintaining diversity for the subsequent nodes.

5.2 Concluding Remarks

In this chapter, the FISRP was proposed. In Chapter 6, both the algorithm's performance and its differences from SRP and ARF are presented.

The FISRP's protocol of prioritizing features in the learners' subspaces and at the first split node introduces additional processing overhead and it may not be beneficial in scenarios where dimensionality is not a problem. It is necessary to have an elevated amount of features.

In scenarios where the number of features only increases (such as Increment), it is possible for both SRP and ARF to process the entire stream with most of their learners without resetting. While this results in reduced processing time, the potential improvement in classification performance from using the newly available features is not explored. FISRP, on the other hand, being able to reset its learners frequently and prioritize features, is able to explore the newer features.

It is expected that the average tree size (nodes and depth) in FISRP is smaller than that of SRP.

6 Experimental Analysis

This chapter reports experiments with state-of-the-art classifiers and compares FISRP with SRP and ARF on Varying Feature Spaces scenarios.

In order to compare SRP, FISRP and ARF in a Varying Feature Space scenario, variations of these ensembles were created: DSSRP (Dynamic Subspace Streaming Random Patches), DSFISRP (Dynamic Subspace Feature Importance Streaming Random Patches), and DSARF (Dynamic Subspace Adaptive Random Forest). These variations create their feature subspace using only the features available at the moment. In contrast, the originals can select any feature for their subspace at any time, assuming the entire feature space is known and immutable. Therefore, SRP, FISRP, and ARF can benefit from having access to features before they appear on the stream. Such an unrealistic condition is not feasible in many real-world applications. This is also a limitation of MOA's implementation: it is not possible to control the availability of each feature while the stream is running.

The focus of this project is to highlight the difference between the proposed ensemble, the original version, and other ensembles. Therefore, a group of 10 datasets (real and synthetic) was selected. Experiments on DSSRP, DSFISRP, and DSARF are done on the connect4, covtypeNorm, gassensor, kdd99, nomao, Hyperplane_drift, RandomRBF, RandomTree, Text, and waveform datasets.

6.1 Experimental Protocol

In order to conduct robust, yet random experiments, a selection of 30 random seeds of streams was chosen. MOA's framework can generate streams by randomly selecting features to be added or removed based on seeds. The seeds are: 789, 53, 750, 765, 166, 834, 194, 296, 896, 883, 788, 619, 647, 453, 410, 911, 713, 140, 380, 123, 283, 946, 968, 31, 215, 299, 783, 996, 730 and 600.

The link for the algorithms and stream generator used in this project is as follows: https://github.com/Andre-Seiji/Feature-Importance-Streaming-Random-Patches.

For the comparison, each of the 4 scenarios of the trapezoidal meta generator (described in Chapter 4) had experiments with all 30 seeds.

6.2 Evaluation Protocol

For the experiments, the goal is to analyze prediction performance and computational cost. The metrics used, Kappa statistic, CPU seconds and RAM-Hours are default from MOA. Also, "BasicClassificationPerformanceEvaluator" is used as it calculates the metrics across the entire stream. Prequential evaluation (test-then-train) was used.

Since each experiment is done 30 times, for each metric, the average and standard deviation are calculated.

Kappa statistic

Kappa statistic is a popular metric for classification accuracy on unbalanced datasets. It is used in both offline learning and streaming mining of data streams.

The Kappa statistic **k** is represented in Equation 6.1:

$$k = \frac{p - p_r}{1 - p_r} \tag{6.1}$$

Where p is the accuracy of the classifier being analyzed and p_r is the expected accuracy considering the distribution of the sample. If the predictions coincide 100% with the correct ones, a value of $k = 1$ is given. Else, if the predictions coincide with the correct ones by chance, $k = 0$ (ŽLIOBAITĖ et al., 2015).

Time and memory metrics

"CPU seconds" and "RAM-Hours" are time metrics from MOA's framework used to estimate the computational resources usage.

Those metrics are directly influenced by the average tree depth of the ensemble as analyzed by Gomes, Read & Bifet (2019). A higher average tree depth is accompanied by a higher overall computational cost.

If two algorithms with similar prediction performance have different “CPU seconds” and “RAM-Hours”, it means that one is more efficient or compact.

Tree metrics

"AVG Tree Nodes" calculates the average node counts (internal and leaf) across all trees from the ensemble and "AVG Tree Depth" is the average path from the root to any leaf.

Nemenyi test

The Nemenyi test is a non-parametric post-hoc method used to compare multiple models across the same datasets without assuming a normal distribution. It is applied after a significant Friedman or Kruskal–Wallis test to assess which specific pairs differ. The test evaluates differences in average ranks and computes a critical difference (CD):

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{6.2}$$

where q_α is the critical value from the studentized range distribution, k is the number of models, and N is the number of blocks (e.g., datasets) (NEMENYI, 1963). Any pair of models whose mean-rank difference exceeds the CD value is significantly different and models not connected by a line are considered significantly different. The lowest (best) ranks are to the right. (DEMŠAR, 2006).

6.3 Datasets

Datasets from applications of real-world data and synthetically generated on MOA (BIFET et al., 2010).

Table 6.1 summarizes the datasets used in this chapter: 5 from real-world data and 5 from synthetic. The real datasets were chosen following a minimum requirement of 40 features and 10,000 instances. CovtypeNorm and kdd99 have a large number of instances and Nomao and gassensor have more than 100 features.

The synthetics were generated on MOA with 10 drifts and at least 50 features and 100,000 instances. These datasets are briefly described below.

Table 6.1: Datasets

	Attributes	Instances	Type
NOMAO	118	34,465	Real
GASSENSOR	128	13,910	Real
Connect-4	42	67,557	Real
CovtypeNorm	54	581,012	Real
kdd99	41	4,898,431	Real
Hyperplane_drift	100	100,000	Synthetic
RandomRBF	50	100,000	Synthetic
RandomTree	100	100,000	Synthetic
Text	100	100,000	Synthetic
Waveform	100	500,000	Synthetic

NOMAO

NOMAO (CANDILLIER; LEMAIRE, 2012) is a dataset created as a challenge. It was created based on *Nomao* search engine that contains data collected from multiple sources from the web. The classification determines whether two places are duplicates by comparing their name, phone, address and GPS data. It has 34,465 instances, 118 attributes and two classes.

GASSENSOR

GASSENSOR Vergara et al. (2012) and Rodriguez-Lujan et al. (2014) is a dataset regarding chemical sensors exposed to six gases changing over time.

The data was collected in a gas delivery platform over a period of 36 months. It contains 13,910 instances, 128 attributes and six classes.

Connect-4

Connect-4 is a dataset about a board game. 42 positions going from left to right, top to bottom. The prediction is the outcome class for the first player being (2: win, 1: loss and 0:draw). It contains 67,557 instances, 42 attributes and 3 classes.

CovtypeNorm

Normalized dataset of forest cover type classification (BLACKARD, 1998). It contains 54 features and 581,012 instances.

kdd99

With 41 features and 4.9 million instances, it is a dataset of intrusion detection systems and each connection is classified as normal or attack (TAVALLAEE et al., 2009).

Hyperplane_drift

A classification of a rotation Hyperplane with 100 features and 100,000 instances. The dataset was generated with 10 drifts equally distributed (BIFET et al., 2010).

RandomRBF

RandomRBF is generated by randomly selecting class centers. These centers are used for sampling using Gaussian distribution to create new instances. The dataset has 50 features, 50 clusters, 100,000 instances and 10 drifts equally distributed (BIFET et al., 2010).

RandomTree

RandomTree is a dataset created by randomly splitting features and generating labels at its leaves. It has 100 features, 100,000 instances and 10 equally distributed drifts (BIFET et al., 2010).

Text

Text simulates sentiment analysis on tweets and was created with 100 features, 100,000 instances and 10 drifts equally distributed (BIFET et al., 2010).

Waveform

A dataset of identification of three types of waveform with 100 features, 500,000 instances and 10 drifts equally distributed (BIFET et al., 2010).

6.4 Results

This section presents the results of the experiment: DSARF, DSSRP and DSFISRP.

DSARF, DSSRP and DSFISRP

In Tables 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7, the DSARF, DSSRP and DSFISRP are compared across all datasets and four scenarios. In the Kappa comparison, the higher average value is highlighted in bold; for the rest of the tests, the lower average value is presented in bold. The reason is that Kappa is the only test that having a higher value is better. Lastly, the number of wins for each model and the results of the Nemenyi tests are shown (the ensemble with the lowest ranking is better).

Despite all Nemenyi kappa tests (Figures 6.1, 6.2, 6.3 and 6.4) showing that there is no significant difference between the three models, DSARF achieved higher kappa values, followed by DSFISRP. However, in experiments characterized by high dimensionality such as Text and WAVEFORM both on Increment, Parabolic and Hyperbolic scenarios, DSFISRP is superior (Table 6.2).

In Text on Increment, the values of kappa were: 48 (DSARF), 24 (DSSRP) and 53 (DSFISRP) (Table 6.2). AVG Tree Depth values were: 112 (DSARF), 836 (DSSRP) and 31 (DSFISRP) (Table 6.6). Evaluation time values were: 750 (DSARF), 1823 (DSSRP) and 1057 (DSFISRP) (Table 6.3).

In this example, DSFISRP achieved kappa values that were more than twice those of DSSRP while having a slightly higher evaluation time than DSARF. This highlights a deficiency of DSSRP in the Text dataset, as it continues to increase the size of the trees without improving classification performance (as shown in

Figure 6.5), resulting in significantly higher evaluation times. Text on Parabolic had similar results.

In WAVEFORM on Increment, the values of kappa were: 56 (DSARF), 45 (DSSRP) and 63 (DSFISRP) (Table 6.2). AVG Tree Nodes values were: 10036 (DSARF), 12062 (DSSRP) and 1876 (DSFISRP) (Table 6.5). This behavior of having higher kappa values with lower AVG Tree Nodes and AVG Tree Depth was also present on Parabolic and Hyperbolic scenarios. DSFISRP on Hyperbolic highlights the ensemble's ability to recover after the middle of the stream, when the number of features available increases (Figure 6.6).

However, on all scenarios of WAVEFORM, DSFISRP's evaluation time and RAM-hours were higher because of the extra computational cost of running the ranking learner and the high frequency of resetting its base learners.

In these examples, DSFISRP's strategy of using smaller and newer trees (with feature prioritization) results in better classification performance.

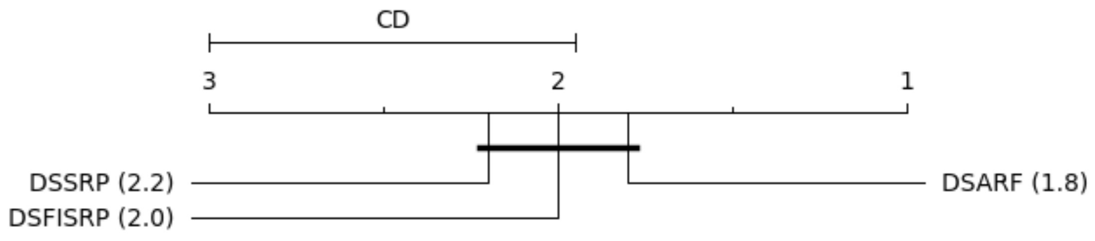


Figure 6.1: Nemenyi Kappa Increment

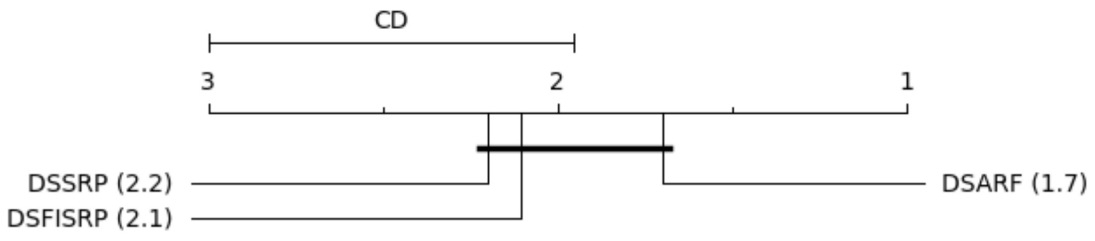


Figure 6.2: Nemenyi Kappa Removal

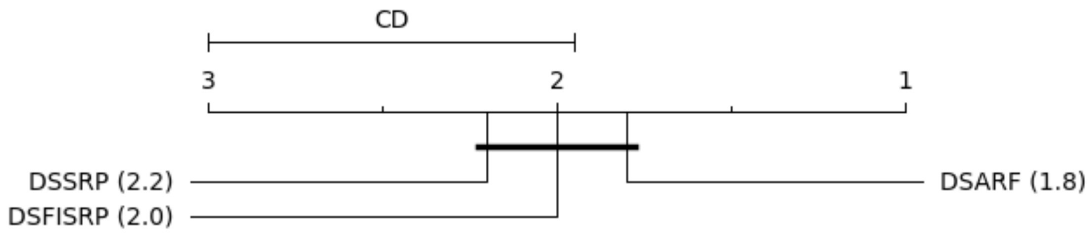


Figure 6.3: Nemenyi Kappa Parabolic

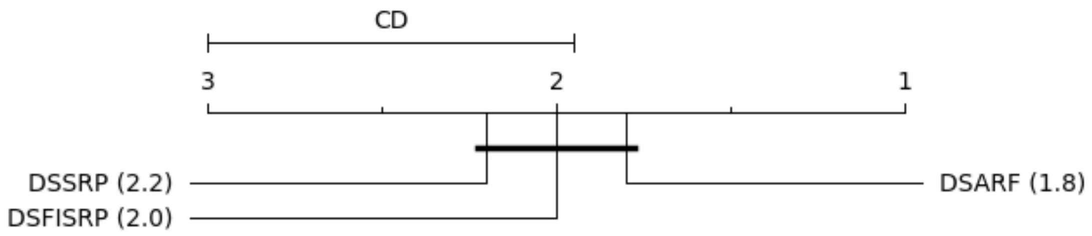


Figure 6.4: Nemenyi Kappa Hyperbolic

Table 6.2: Kappa Statistic

	Increment			Removal			Parabolic			Hyperbolic		
	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP
connect4 (R)	**42.61**	42.32	42.10	**38.02**	36.76	37.00	**40.65**	39.48	39.43	**40.77**	40.21	40.35
	± 1.44	± 1.34	± 1.28	± 1.31	± 1.29	± 1.35	± 0.74	± 0.72	± 0.77	± 1.43	± 1.30	± 1.32
covtypeNorm (R)	**85.74**	85.65	85.52	**81.78**	80.70	80.85	**83.95**	83.58	83.58	**86.95**	86.31	86.36
	± 3.19	± 3.15	± 3.36	± 2.87	± 3.08	± 3.01	± 3.40	± 3.31	± 3.31	± 2.90	± 3.51	± 3.48
gassensor (R)	91.05	**94.67**	93.30	87.03	92.28	**92.67**	90.04	**93.70**	92.78	87.51	**93.21**	92.78
	± 0.63	± 0.39	± 0.51	± 0.96	± 0.40	± 0.55	± 0.94	± 0.55	± 0.71	± 0.67	± 0.29	± 0.35
kdd99 (R)	99.94	**99.96**	99.95	99.92	**99.93**	99.92	99.94	**99.96**	99.95	99.95	**99.97**	99.96
	± 0.01	± 0.01	± 0.01	± 0.01	± 0.00	± 0.00	± 0.01	± 0.00	± 0.01	± 0.00	± 0.00	± 0.00
nomao (R)	93.05	**93.65**	93.30	91.67	**92.00**	91.90	91.80	**92.27**	92.15	92.78	**93.14**	92.70
	± 0.13	± 0.12	± 0.20	± 0.72	± 0.66	± 0.73	± 0.78	± 0.66	± 0.62	± 0.34	± 0.34	± 0.37
Hyperplane (S)	**23.16**	15.97	21.93	**32.74**	27.36	20.78	**24.71**	15.96	20.36	**30.16**	24.80	22.05
	± 1.39	± 1.84	± 0.76	± 1.28	± 1.14	± 0.87	± 1.07	± 1.87	± 0.73	± 1.65	± 1.78	± 1.05
RandomRBF (S)	**91.51**	83.97	88.37	93.55	**93.71**	92.23	**90.32**	83.98	86.41	**94.61**	94.50	93.83
	± 0.36	± 1.71	± 0.74	± 0.42	± 0.39	± 0.51	± 0.48	± 1.71	± 0.87	± 0.30	± 0.34	± 0.35
RandomTree (S)	**7.18**	3.61	6.35	**5.38**	3.83	4.75	**5.55**	3.53	5.07	5.79	4.60	**6.22**
	± 3.80	± 0.66	± 2.04	± 2.53	± 0.76	± 1.77	± 1.83	± 0.64	± 1.38	± 3.51	± 2.13	± 1.64
Text (S)	48.19	23.85	**52.68**	**45.88**	38.86	45.14	47.82	23.85	**50.65**	45.65	29.71	**47.45**
	± 10.10	± 7.97	± 8.49	± 8.35	± 9.22	± 8.66	± 5.08	± 7.97	± 4.33	± 5.98	± 7.18	± 4.85
WAVEFORM (S)	55.85	45.13	**63.19**	**65.36**	64.09	64.21	56.94	44.10	**61.13**	63.81	61.69	**66.80**
	± 4.77	± 5.35	± 3.52	± 3.44	± 3.49	± 3.41	± 4.12	± 5.32	± 3.59	± 3.98	± 4.89	± 3.25
Wins	**5**	3	2	**6**	3	1	**5**	3	2	**4**	3	3

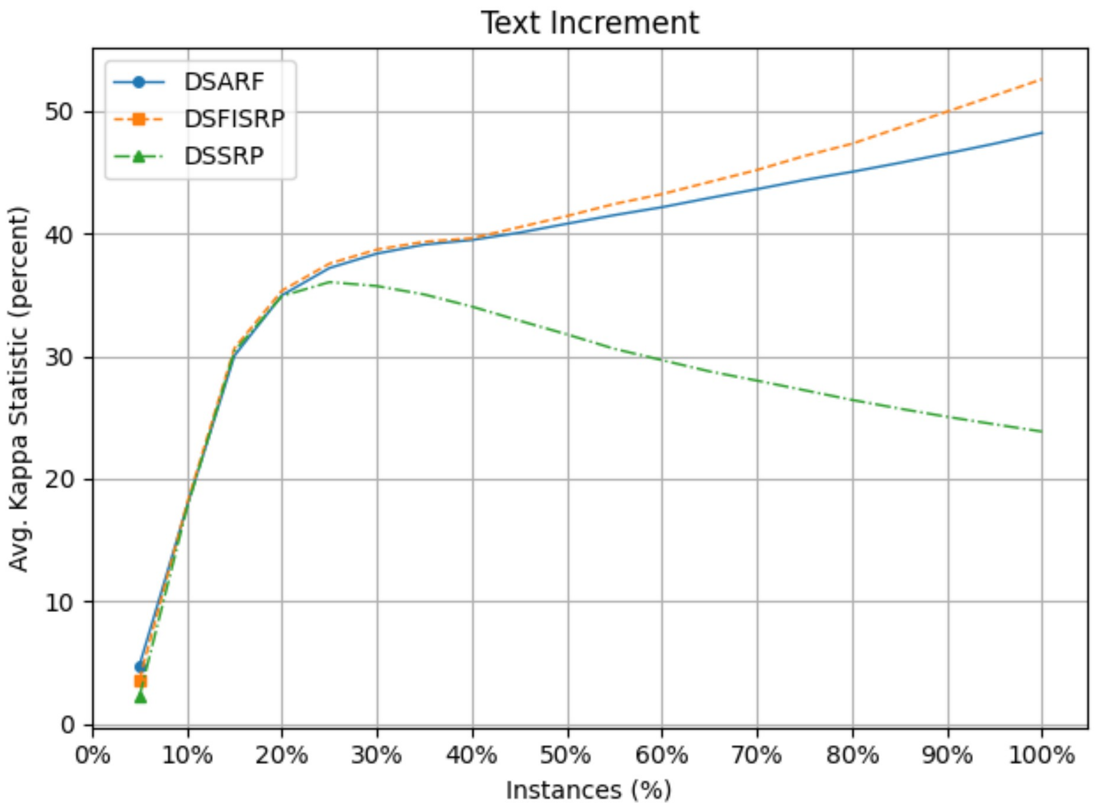


Figure 6.5: Evolution of the Kappa Statistic throughout the Stream - Text Increment

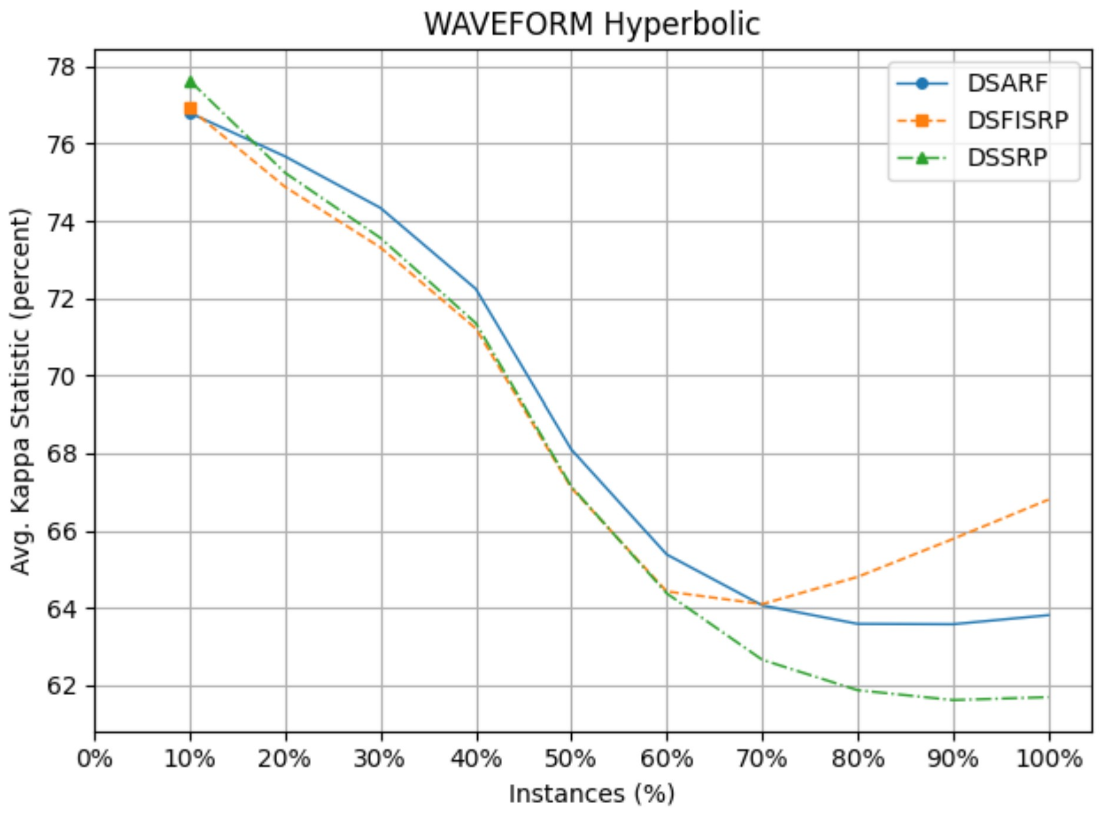


Figure 6.6: Evolution of the Kappa Statistic throughout the Stream - WAVEFORM Hyperbolic

Nemenyi tests demonstrate that there is a significant difference regarding evaluation time across all ensembles. DSARF had lower evaluation time in all scenarios (Figures 6.7, 6.8, 6.9 and 6.10).

In the connect4 and covtypeNorm datasets, DSFISRP had a lower Evaluation time than DSSRP (Table 6.3). This is also true for RAM-hours (Table 6.4), AVG Tree Depth (Table 6.6), and AVG Age (Table 6.7). This is consistent with the idea that frequently resetting the base learners results in reduced processing time.

In the Increment and Parabolic scenarios of Hyperplane, RandomRBF, RandomTree, and WAVEFORM, in comparison with DSFISRP, DSSRP achieved lower evaluation time (Table 6.3) and RAM-hours (Table 6.4). The reason is that the base learners from DSSRP created at the beginning of the stream are mostly the same at the end: fewer background learners are created and the amount of base learners replaced is smaller (compared to DSFISRP) resulting in significantly lower computational cost. Even though the trees are bigger, the Evaluation time is smaller.

For these examples, the DSFISRP's benefit of having smaller trees does not overcome the extra computational cost of running a ranking learner and constantly resetting all classification learners.

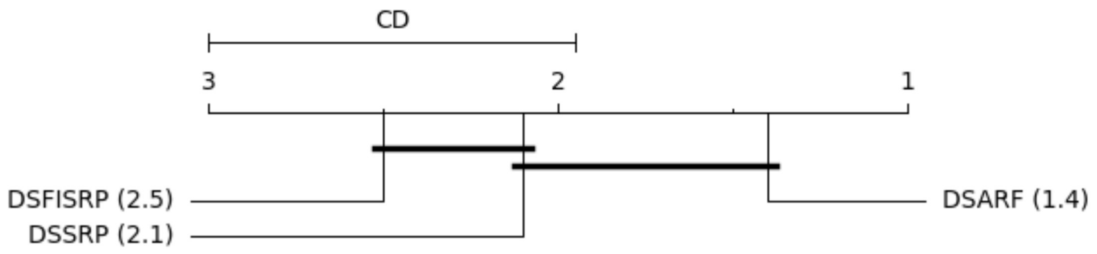


Figure 6.7: Nemenyi Evaluation time Increment

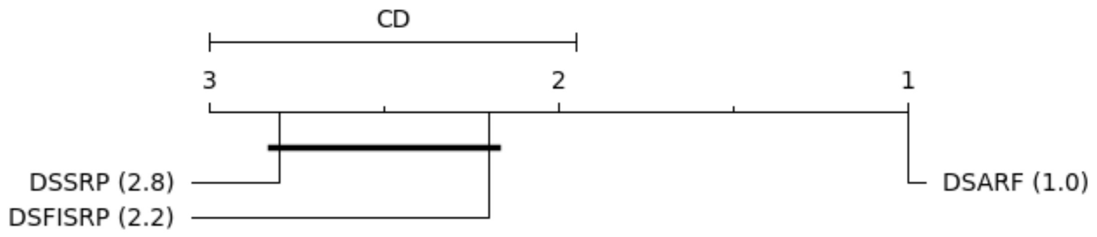


Figure 6.8: Nemenyi Evaluation time Removal

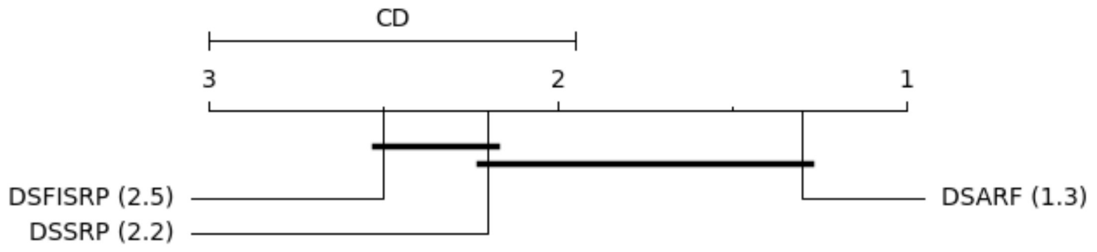


Figure 6.9: Nemenyi Evaluation time Parabolic

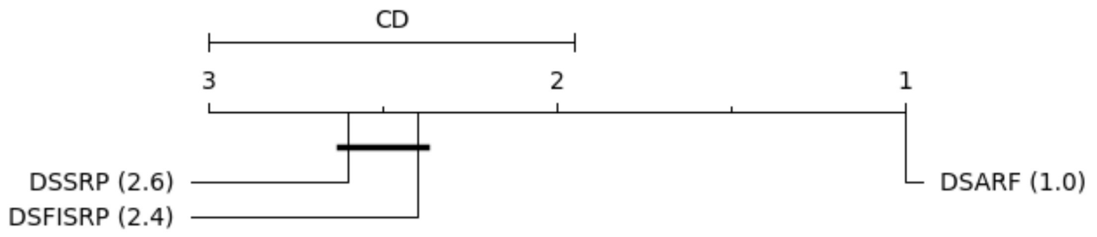


Figure 6.10: Nemenyi Evaluation time Hyperbolic

Table 6.3: Evaluation time

	Increment			Removal			Parabolic			Hyperbolic		
	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP
connect4 (R)	**323.83**	1065.78	853.42	**239.41**	744.45	665.51	**284.28**	872.70	726.21	**287.83**	927.40	794.57
	± 6.10	± 34.93	± 36.59	± 5.29	± 38.52	± 25.85	± 8.16	± 28.77	± 27.48	± 6.34	± 31.40	± 26.47
covtypeNorm (R)	**2768.02**	6809.89	6075.74	**2461.04**	6250.81	5829.52	**2690.16**	6519.19	5789.52	**2809.64**	7091.21	6513.47
	± 134.44	± 434.94	± 380.91	± 111.42	± 447.25	± 443.65	± 147.22	± 458.89	± 408.73	± 166.71	± 396.38	± 461.62
gassensor (R)	**148.91**	221.39	255.41	**144.03**	297.99	296.68	**133.82**	198.68	241.90	**157.56**	264.23	300.68
	± 3.85	± 6.22	± 5.00	± 3.31	± 3.25	± 4.47	± 3.79	± 8.08	± 6.94	± 4.38	± 8.14	± 4.72
kdd99 (R)	**4343.40**	10744.95	10167.56	**4797.95**	10794.30	11364.51	**4897.01**	9790.81	9364.94	**5293.54**	11286.09	12309.71
	± 654.77	± 1673.81	± 1315.19	± 469.30	± 893.15	± 738.97	± 758.95	± 1389.00	± 999.24	± 581.04	± 391.08	± 552.87
nomao (R)	**259.96**	574.50	529.78	**175.35**	393.99	391.75	**192.03**	414.07	411.49	**226.82**	515.15	488.40
	± 5.39	± 10.40	± 10.76	± 4.80	± 9.33	± 10.30	± 3.84	± 9.54	± 8.25	± 3.51	± 10.44	± 11.38
Hyperplane (S)	1639.96	**720.30**	3041.67	**1807.89**	5354.10	3026.95	1661.39	**728.50**	2796.65	**1804.84**	3877.17	3248.01
	± 47.60	± 76.48	± 57.28	± 40.23	± 83.21	± 51.91	± 37.32	± 83.17	± 39.53	± 40.39	± 56.72	± 47.75
RandomRBF (S)	358.19	**301.60**	906.13	**594.79**	1274.71	1251.76	462.49	**300.21**	959.61	**504.43**	974.53	1219.76
	± 15.99	± 8.09	± 28.38	± 16.81	± 24.57	± 27.35	± 13.86	± 7.82	± 17.87	± 12.99	± 21.87	± 32.26
RandomTree (S)	2292.15	**2073.68**	3833.44	**2122.98**	5904.84	4883.55	**2110.89**	2320.36	3944.68	**2253.59**	4848.42	4600.61
	± 104.47	± 695.67	± 76.01	± 94.54	± 145.86	± 178.41	± 60.69	± 761.78	± 126.53	± 91.03	± 135.94	± 147.04
Text (S)	**749.54**	1822.70	1056.99	**598.11**	1983.36	1713.98	**684.36**	1823.02	1218.03	**748.17**	1670.94	1644.93
	± 77.63	± 422.19	± 110.54	± 35.16	± 75.61	± 102.34	± 42.59	± 418.93	± 93.80	± 35.54	± 97.29	± 98.95
WAVEFORM (S)	2401.67	**1649.88**	4112.03	**2276.31**	7192.37	7518.63	2258.12	**1566.06**	4972.47	**2324.74**	5794.14	6629.94
	± 197.42	± 332.04	± 99.63	± 33.05	± 128.44	± 432.64	± 108.49	± 321.80	± 296.93	± 101.00	± 162.92	± 528.40
Wins	**6**	4	0	**10**	0	0	**7**	3	0	**10**	0	0

Nemenyi tests demonstrate that there is a significant difference regarding RAM-hours across all ensembles. DSARF had lower values across all scenarios (Figures 6.11, 6.12, 6.13 and 6.14). DSFISRP achieved lower values than DSSRP in Removal and Hyperbolic. In these scenarios, DSSRP does not have the benefit of having most of its base learners remaining the same throughout the entire stream: the number of base learners replaced and the number of background learners created are significantly higher than in Increment and Parabolic scenarios. DSFISRP's strategy of using smaller, newer trees results in lower computational costs than DSSRP. (Table 6.4)

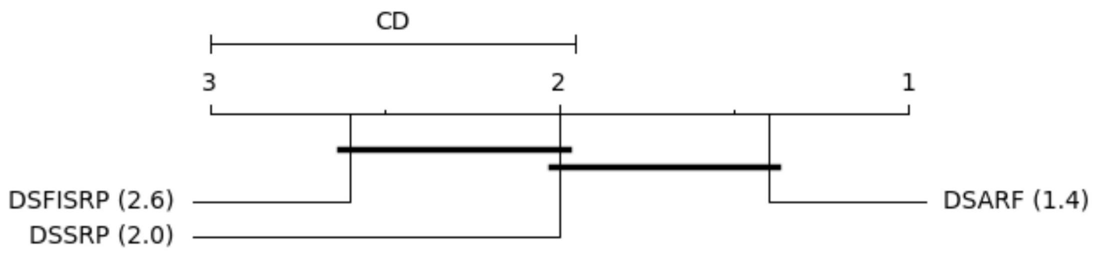


Figure 6.11: Nemenyi RAM-Hours Increment

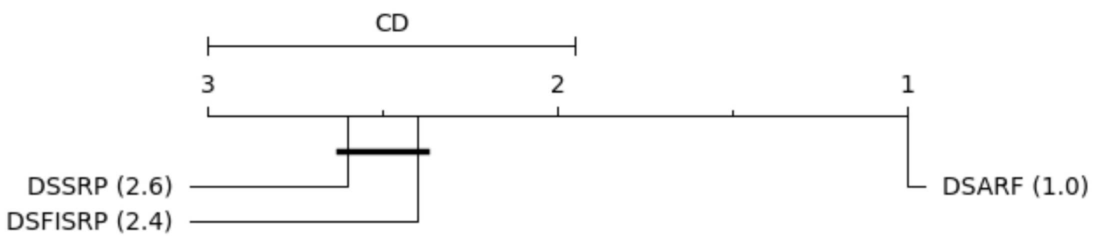


Figure 6.12: Nemenyi RAM-Hours Removal

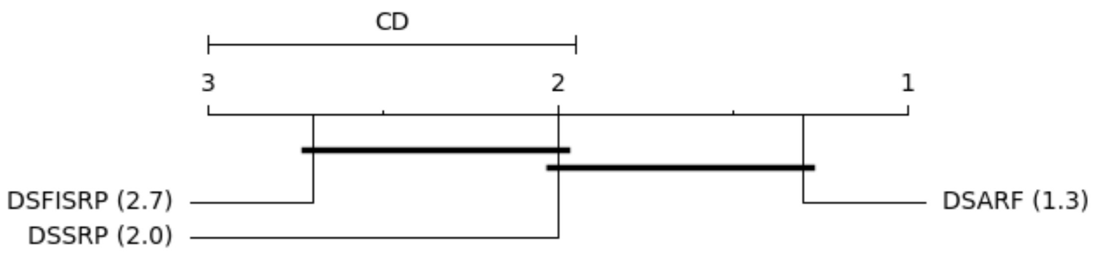


Figure 6.13: Nemenyi RAM-Hours Parabolic

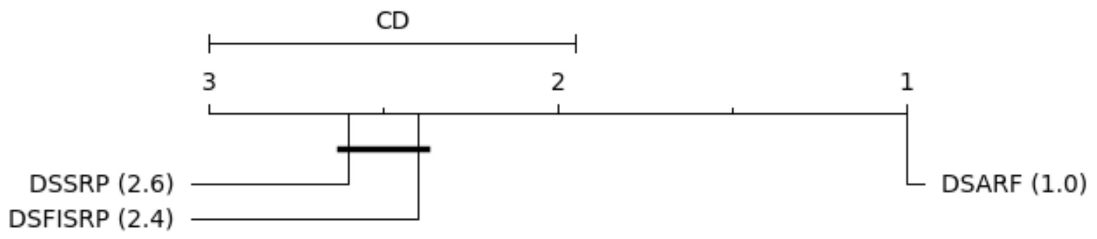


Figure 6.14: Nemenyi RAM-Hours Hyperbolic

Table 6.4: RAM-Hours

	Increment			Removal			Parabolic			Hyperbolic		
	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP
connect4 (R)	**6.65e-05**	2.27e-04	1.80e-04	**4.90e-05**	1.53e-04	1.39e-04	**5.83e-05**	1.80e-04	1.51e-04	**5.88e-05**	1.93e-04	1.68e-04
	± 1.45e-06	± 7.61e-06	± 8.30e-06	± 1.34e-06	± 8.26e-06	± 6.15e-06	± 1.88e-06	± 6.57e-06	± 6.25e-06	± 1.40e-06	± 6.45e-06	± 6.95e-06
covtypeNorm (R)	**7.57e-04**	1.69e-03	1.54e-03	**6.62e-04**	1.54e-03	1.49e-03	**7.14e-04**	1.57e-03	1.46e-03	**7.64e-04**	1.75e-03	1.66e-03
	± 3.16e-05	± 9.95e-05	± 8.48e-05	± 2.83e-05	± 1.07e-04	± 1.12e-04	± 3.35e-05	± 1.04e-04	± 9.00e-05	± 3.35e-05	± 8.26e-05	± 1.03e-04
gassensor (R)	**3.33e-05**	5.18e-05	5.85e-05	**2.96e-05**	6.49e-05	6.82e-05	**2.89e-05**	4.47e-05	5.60e-05	**3.35e-05**	5.86e-05	6.81e-05
	± 1.33e-06	± 1.51e-06	± 1.89e-06	± 1.12e-06	± 9.81e-07	± 1.39e-06	± 1.28e-06	± 1.79e-06	± 2.13e-06	± 1.41e-06	± 1.83e-06	± 1.22e-06
kdd99 (R)	**1.54e-03**	3.76e-03	3.78e-03	**1.65e-03**	3.71e-03	4.07e-03	**1.74e-03**	3.36e-03	3.43e-03	**1.83e-03**	3.91e-03	4.52e-03
	± 2.51e-04	± 5.66e-04	± 5.19e-04	± 1.60e-04	± 3.01e-04	± 2.86e-04	± 2.84e-04	± 4.76e-04	± 3.87e-04	± 2.08e-04	± 1.39e-04	± 2.40e-04
nomao (R)	**5.22e-05**	1.25e-04	1.17e-04	**3.42e-05**	8.14e-05	8.53e-05	**3.79e-05**	8.61e-05	8.93e-05	**4.52e-05**	1.11e-04	1.07e-04
	± 1.25e-06	± 2.34e-06	± 3.32e-06	± 1.12e-06	± 2.03e-06	± 2.56e-06	± 9.48e-07	± 1.91e-06	± 1.74e-06	± 8.13e-07	± 2.21e-06	± 3.80e-06
Hyperplane (S)	4.19e-04	**1.72e-04**	5.66e-04	**4.12e-04**	1.17e-03	5.71e-04	3.97e-04	**1.73e-04**	5.24e-04	**4.38e-04**	8.66e-04	6.05e-04
	± 1.82e-05	± 1.96e-05	± 1.43e-05	± 1.04e-05	± 1.81e-05	± 1.34e-05	± 1.16e-05	± 2.08e-05	± 1.19e-05	± 1.21e-05	± 1.28e-05	± 1.38e-05
RandomRBF (S)	7.91e-05	**6.69e-05**	1.87e-04	**1.20e-04**	2.69e-04	2.67e-04	9.65e-05	**6.66e-05**	2.02e-04	**1.05e-04**	2.09e-04	2.62e-04
	± 3.62e-06	± 1.90e-06	± 5.23e-06	± 3.30e-06	± 5.01e-06	± 7.36e-06	± 3.05e-06	± 1.81e-06	± 4.05e-06	± 2.76e-06	± 4.81e-06	± 7.68e-06
RandomTree (S)	6.02e-04	**4.96e-04**	8.65e-04	**4.73e-04**	1.29e-03	1.10e-03	**5.13e-04**	5.40e-04	8.97e-04	**5.47e-04**	1.09e-03	1.05e-03
	± 3.27e-05	± 1.64e-04	± 1.82e-05	± 2.21e-05	± 3.14e-05	± 4.53e-05	± 1.67e-05	± 1.71e-04	± 3.08e-05	± 2.51e-05	± 3.09e-05	± 3.81e-05
Text (S)	**1.79e-04**	4.46e-04	2.35e-04	**1.27e-04**	4.49e-04	3.90e-04	**1.53e-04**	4.46e-04	2.71e-04	**1.62e-04**	3.84e-04	3.72e-04
	± 2.42e-05	± 1.05e-04	± 2.32e-05	± 7.06e-06	± 1.60e-05	± 2.24e-05	± 1.31e-05	± 1.05e-04	± 2.14e-05	± 8.25e-06	± 2.06e-05	± 2.28e-05
WAVEFORM (S)	6.91e-04	**4.49e-04**	1.02e-03	**5.68e-04**	1.76e-03	1.88e-03	6.11e-04	**4.26e-04**	1.26e-03	**6.15e-04**	1.43e-03	1.68e-03
	± 8.65e-05	± 9.82e-05	± 2.72e-05	± 1.12e-05	± 3.19e-05	± 1.14e-04	± 4.66e-05	± 9.42e-05	± 7.67e-05	± 4.03e-05	± 4.12e-05	± 1.28e-04
Wins	**6**	4	0	**10**	0	0	**7**	3	0	**10**	0	0

Although the Nemenyi tests indicate that there is no significant difference regarding AVG Tree Nodes between DSFISRP and DSARF, DSFISRP was superior in the Increment and DSARF was better in the other scenarios (Figures 6.15, 6.16, 6.17 and 6.18). DSFISRP does have fewer nodes in all synthetic datasets in Increment (Table 6.5).

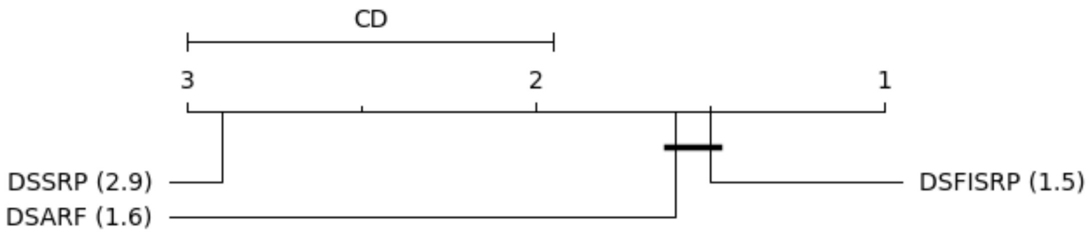


Figure 6.15: Nemenyi AVG Tree Nodes Increment

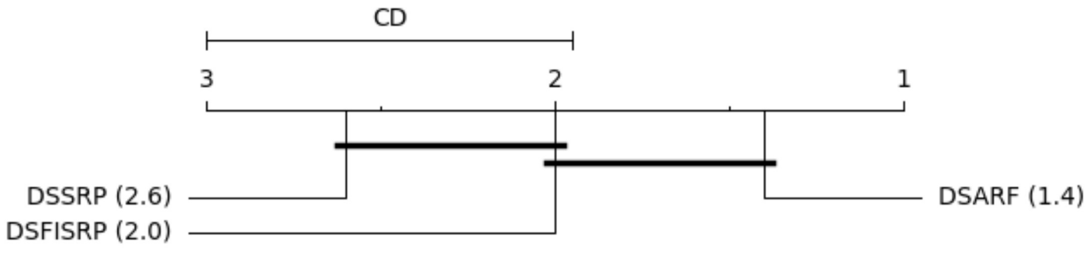


Figure 6.16: Nemenyi AVG Tree Nodes Removal

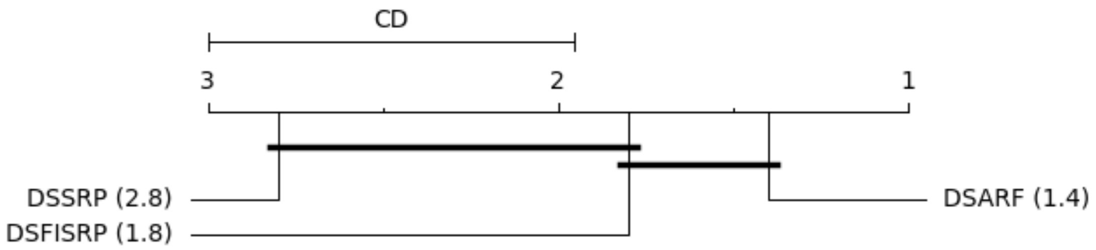


Figure 6.17: Nemenyi AVG Tree Nodes Parabolic

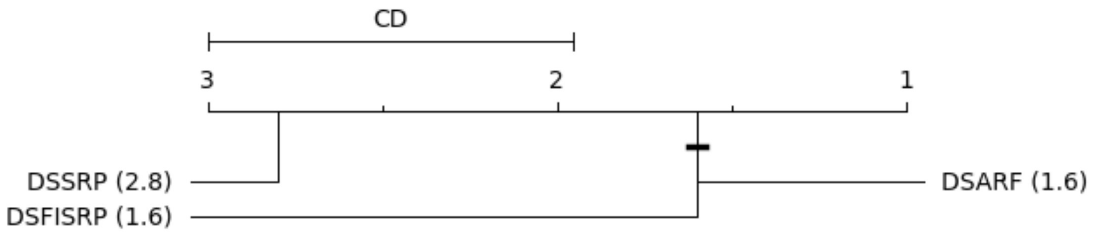


Figure 6.18: Nemenyi AVG Tree Nodes Hyperbolic

Table 6.5: AVG Tree Nodes

	Increment			Removal			Parabolic			Hyperbolic		
	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP
connect4 (R)	**57.34** ± 7.46	513.32 ± 56.43	87.72 ± 17.29	**1.00** ± 0.00	1.00 ± 0.00	1.06 ± 0.15	**6.41** ± 3.77	25.58 ± 11.45	12.32 ± 8.71	**56.02** ± 8.12	372.96 ± 50.26	174.66 ± 17.55
covtypeNorm (R)	**29.43** ± 1.88	119.99 ± 6.73	51.98 ± 4.72	**1.00** ± 0.00	1.00 ± 0.00	1.00 ± 0.00	**24.57** ± 17.63	73.77 ± 57.68	83.28 ± 96.83	**28.86** ± 1.81	120.53 ± 5.96	52.43 ± 4.74
gassensor (R)	**3.84** ± 0.53	24.52 ± 4.57	5.27 ± 0.41	**6.52** ± 2.73	15.65 ± 2.32	8.95 ± 2.20	**6.48** ± 2.27	17.85 ± 3.04	8.25 ± 1.79	**3.36** ± 0.49	24.31 ± 4.63	5.20 ± 0.59
kdd99 (R)	**58.38** ± 0.46	78.39 ± 8.11	69.32 ± 2.98	**1.00** ± 0.00	1.00 ± 0.00	1.05 ± 0.25	16.98 ± 28.72	13.58 ± 17.58	**12.55** ± 21.03	**57.57** ± 0.87	79.27 ± 7.98	115.76 ± 7.58
nomao (R)	**22.21** ± 1.93	92.38 ± 4.09	39.30 ± 4.15	**53.44** ± 12.10	179.90 ± 52.20	109.01 ± 30.25	**52.24** ± 11.17	179.02 ± 43.88	88.14 ± 21.99	**21.90** ± 2.15	93.39 ± 5.36	37.57 ± 4.42
Hyperplane (S)	3623.14 ± 27.44	4020.38 ± 190.30	**9.37** ± 1.36	345.13 ± 13.99	1190.58 ± 12.54	**31.04** ± 27.32	353.06 ± 11.76	3797.50 ± 156.22	**31.31** ± 27.40	2046.33 ± 42.61	6133.91 ± 74.18	**9.38** ± 1.56
RandomRBF (S)	1929.65 ± 66.54	3262.79 ± 105.60	**175.05** ± 17.00	337.61 ± 22.52	785.77 ± 46.88	**228.20** ± 80.04	338.05 ± 22.44	3262.41 ± 105.55	**214.62** ± 80.04	768.27 ± 64.94	1726.45 ± 143.88	**158.77** ± 14.62
RandomTree (S)	4729.41 ± 1087.87	10694.92 ± 3314.35	**744.34** ± 148.66	**762.96** ± 237.58	1351.30 ± 348.46	968.35 ± 285.40	**803.75** ± 257.39	5994.80 ± 1412.93	1019.08 ± 406.20	3431.28 ± 379.07	7755.08 ± 1973.14	**598.78** ± 114.44
Text (S)	2490.54 ± 244.67	2361.11 ± 105.87	**395.84** ± 90.99	**132.84** ± 45.84	307.34 ± 39.27	173.09 ± 47.07	**138.08** ± 25.91	2361.11 ± 105.87	188.29 ± 50.77	1435.07 ± 131.96	1250.63 ± 178.11	**270.58** ± 81.17
WAVEFORM (S)	10035.60 ± 906.44	12062.08 ± 2083.88	**1876.18** ± 630.05	**705.15** ± 131.22	3993.10 ± 304.58	2124.53 ± 1294.33	**705.74** ± 136.41	10764.09 ± 1587.00	1722.11 ± 998.01	5312.31 ± 521.14	14508.68 ± 1176.58	**1194.70** ± 334.60
Wins	**5**	0	5	**8**	0	2	**7**	0	3	**5**	0	5

According to all Nemenyi tests, there is no significant difference between DSFISRP and DSARF in AVG Tree Depth (Figures 6.19, 6.20, 6.21 and 6.22). DSFISRP had the same Nemenyi rank as DSARF in Increment, Parabolic and Hyperbolic. However, DSARF had a lower ranking in the Removal because in this scenario the base learners reset more frequently (Table 6.6). This is shown in (Table 6.7).

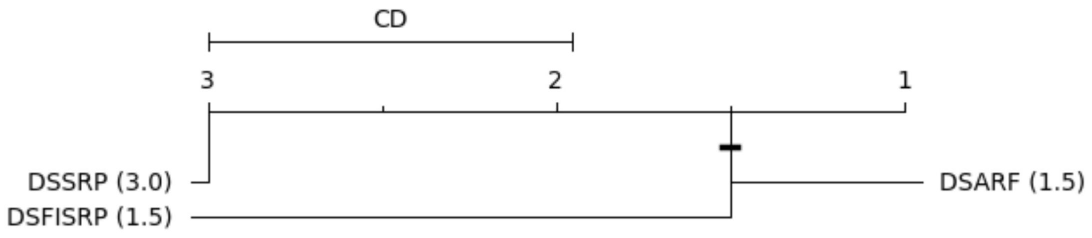


Figure 6.19: Nemenyi AVG Tree Depth Increment

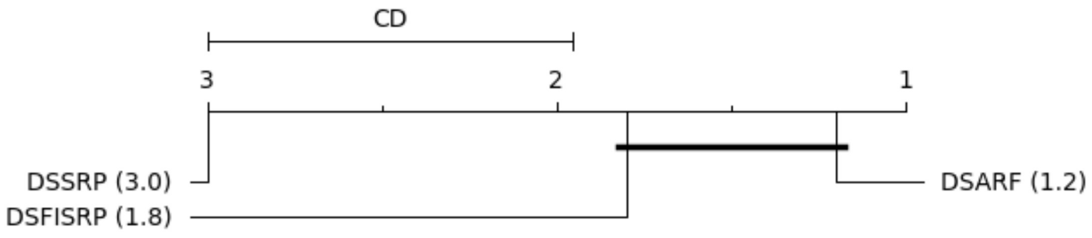


Figure 6.20: Nemenyi AVG Tree Depth Removal

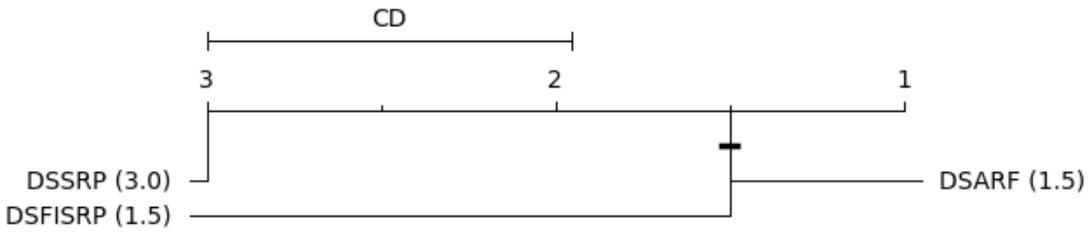


Figure 6.21: Nemenyi AVG Tree Depth Parabolic

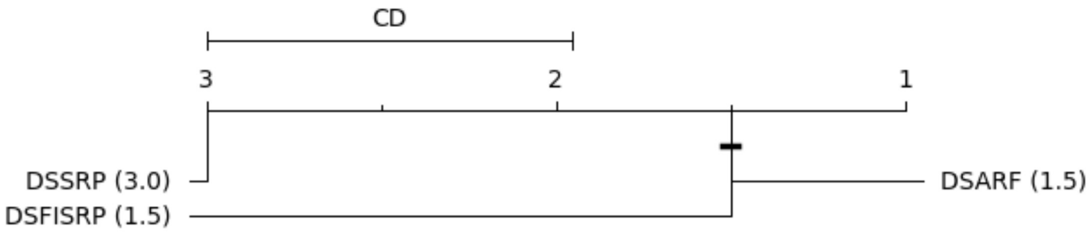


Figure 6.22: Nemenyi AVG Tree Depth Hyperbolic

Table 6.6: AVG Tree Depth

	Increment			Removal			Parabolic			Hyperbolic		
	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP
connect4 (R)	**4.72** ± 0.29	8.75 ± 0.52	6.14 ± 0.38	**4.06** ± 0.26	6.04 ± 0.23	5.09 ± 0.26	**4.42** ± 0.29	7.83 ± 0.58	5.49 ± 0.46	**4.59** ± 0.28	7.86 ± 0.48	6.27 ± 0.39
covtypeNorm (R)	**8.81** ± 0.81	16.19 ± 1.68	12.19 ± 1.93	**7.47** ± 0.75	12.37 ± 0.96	10.71 ± 0.81	**8.19** ± 1.08	15.74 ± 1.85	12.48 ± 1.99	**9.44** ± 0.70	14.96 ± 1.08	12.41 ± 0.99
gassensor (R)	**1.78** ± 0.16	4.61 ± 0.21	1.96 ± 0.17	**0.86** ± 0.17	2.23 ± 0.09	1.80 ± 0.09	**1.53** ± 0.15	3.66 ± 0.23	1.97 ± 0.10	**1.03** ± 0.14	3.39 ± 0.16	1.73 ± 0.08
kdd99 (R)	**1.82** ± 0.32	3.17 ± 0.40	2.22 ± 0.29	**1.02** ± 0.24	1.65 ± 0.26	1.36 ± 0.27	**1.81** ± 0.29	2.86 ± 0.21	2.17 ± 0.28	**1.17** ± 0.15	2.21 ± 0.13	1.65 ± 0.19
nomao (R)	**4.94** ± 0.26	7.20 ± 0.21	5.17 ± 0.48	**4.08** ± 0.19	5.55 ± 0.24	4.68 ± 0.22	**4.83** ± 0.29	6.69 ± 0.26	5.18 ± 0.50	**4.15** ± 0.19	6.21 ± 0.24	4.53 ± 0.32
Hyperplane (S)	15.66 ± 0.16	16.20 ± 0.16	**4.63** ± 0.44	10.25 ± 0.13	12.93 ± 0.05	**4.89** ± 0.47	12.90 ± 0.10	16.12 ± 0.20	**4.71** ± 0.46	12.40 ± 0.15	15.26 ± 0.07	**4.53** ± 0.43
RandomRBF (S)	17.20 ± 0.40	19.22 ± 0.33	**10.40** ± 0.27	11.38 ± 0.16	13.20 ± 0.11	**10.37** ± 0.18	14.32 ± 0.27	19.22 ± 0.32	**10.16** ± 0.21	14.09 ± 0.31	15.93 ± 0.19	**10.83** ± 0.17
RandomTree (S)	7.86 ± 1.43	11.04 ± 2.09	**5.68** ± 0.20	**5.22** ± 0.18	6.69 ± 0.32	5.77 ± 0.23	6.63 ± 0.82	10.60 ± 2.32	**5.86** ± 0.27	6.15 ± 0.25	7.68 ± 0.28	**5.72** ± 0.14
Text (S)	112.08 ± 35.32	835.59 ± 143.23	**30.52** ± 4.79	**19.48** ± 2.30	37.71 ± 4.72	28.57 ± 4.29	49.19 ± 11.30	835.59 ± 143.23	**30.37** ± 3.66	62.09 ± 15.97	170.95 ± 42.35	**36.22** ± 8.70
WAVEFORM (S)	21.01 ± 0.61	21.07 ± 0.50	**17.23** ± 0.60	**16.15** ± 0.22	19.58 ± 0.20	18.36 ± 0.43	18.15 ± 0.45	20.90 ± 0.43	**17.33** ± 0.66	18.29 ± 0.49	21.04 ± 0.25	**17.88** ± 0.37
Wins	**5**	0	5	**8**	0	2	**5**	0	5	**5**	0	5

All Nemenyi tests show that there is no significant difference between DSFISRP and DSARF in AVG Age (Figures 6.23, 6.24, 6.25 and 6.26). In Increment and Parabolic, DSFISRP's trees are reset more frequently than DSARF's (Table 6.7).

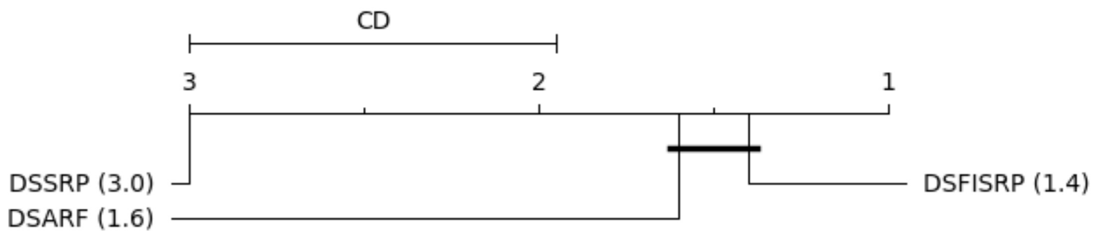


Figure 6.23: Nemenyi AVG Age Increment

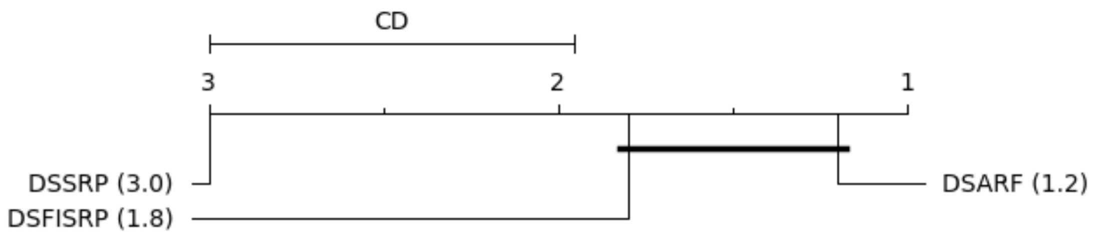


Figure 6.24: Nemenyi AVG Age Removal

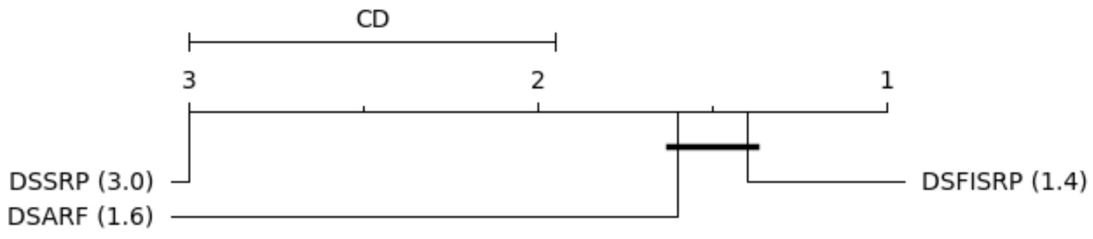


Figure 6.25: Nemenyi AVG Age Parabolic

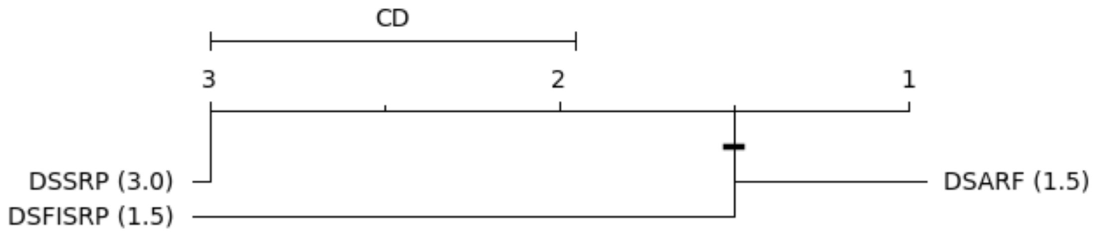


Figure 6.26: Nemenyi AVG Age Hyperbolic

Table 6.7: AVG Age

	Increment			Removal			Parabolic			Hyperbolic		
	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP	DSARF	DSSRP	DSFISRP
connect4 (R)	**1377.77**	3913.31	1850.88	**1072.87**	2418.39	1644.85	**1259.45**	2960.60	1658.90	**1199.67**	3078.64	1877.15
	± 101.28	± 169.23	± 180.78	± 62.05	± 126.46	± 90.75	± 98.04	± 181.45	± 169.25	± 68.62	± 150.53	± 118.40
covtypeNorm (R)	**5131.27**	18826.84	11037.98	**4850.90**	18400.62	12925.09	**4186.84**	14951.89	10400.21	**5902.26**	21008.20	14154.94
	± 785.18	± 2777.13	± 2277.13	± 567.00	± 2096.49	± 1327.33	± 805.17	± 2591.89	± 2405.22	± 787.55	± 2699.94	± 2136.54
gassensor (R)	1112.15	4512.16	**954.21**	**628.87**	1548.56	882.53	913.63	2737.27	**875.35**	**828.90**	3228.89	888.77
	± 96.71	± 301.76	± 117.73	± 69.29	± 64.97	± 48.19	± 75.03	± 252.06	± 68.77	± 80.86	± 254.34	± 78.92
kdd99 (R)	**458171.18**	940499.05	698410.51	**365274.68**	764729.09	705007.57	**391732.43**	756194.05	666584.05	**391723.07**	852883.15	686053.08
	± 5036.28	± 13493.99	± 111829.72	± 75734.34	± 68091.45	± 133062.57	± 55510.54	± 69790.03	± 116018.63	± 65222.63	± 58793.37	± 118989.81
nomao (R)	**2718.96**	6423.71	3428.12	**2174.78**	4301.95	3194.39	**2521.62**	5550.10	3323.50	**2315.69**	5140.08	3145.54
	± 76.68	± 149.41	± 286.37	± 78.28	± 174.63	± 151.25	± 74.04	± 134.49	± 255.59	± 93.98	± 208.77	± 238.26
Hyperplane (S)	51294.76	53669.29	**749.72**	7459.82	14037.41	**836.51**	21699.69	52325.13	**782.13**	20923.09	43442.47	**733.41**
	± 437.87	± 990.12	± 120.99	± 488.66	± 246.13	± 158.21	± 215.31	± 1040.08	± 145.55	± 338.49	± 406.88	± 137.00
RandomRBF (S)	52370.16	52403.67	**6013.77**	7516.84	14397.93	**6335.06**	22417.51	52386.31	**5531.32**	22399.54	44013.52	**7470.69**
	± 0.29	± 132.48	± 519.85	± 88.19	± 233.10	± 273.81	± 3.77	± 64.77	± 275.61	± 2.44	± 5.86	± 481.01
RandomTree (S)	39402.11	67667.28	**7758.84**	**7734.83**	13577.19	9275.40	18314.00	52503.80	**8718.24**	18296.24	36217.92	**10222.31**
	± 4928.68	± 7008.48	± 553.04	± 447.55	± 248.11	± 597.89	± 1607.94	± 7717.20	± 514.80	± 1058.53	± 3304.43	± 1178.37
Text (S)	46910.23	93139.71	**10836.49**	**10549.37**	25264.98	13536.69	19128.04	93139.71	**10151.92**	22310.41	44320.26	**10994.85**
	± 841.45	± 3639.59	± 2465.89	± 1220.47	± 3006.29	± 2356.18	± 497.46	± 3639.59	± 1063.07	± 487.40	± 2249.67	± 2323.27
WAVEFORM (S)	269062.69	295809.82	**73130.48**	**48269.01**	90295.95	75755.19	120995.57	282817.75	**68146.95**	118759.53	197819.47	**80961.35**
	± 3907.98	± 11961.27	± 9289.65	± 752.96	± 1238.23	± 6027.40	± 2717.89	± 10447.18	± 6768.63	± 5578.07	± 5914.25	± 8931.96
Wins	4	0	**6**	**8**	0	2	4	0	**6**	**5**	0	5

For example, in Text in Increment, DSSRP and DSFISRP achieved values of 835 and 30, respectively; similarly, in the Text in Parabolic, the results were 835 and 30, respectively. This highlights a deficiency of DSSRP in the Text dataset, as it continues to increase the size of the trees without improving classification performance, resulting in significantly higher evaluation times.

In these examples, DSFISRP achieved kappa values that were more than twice those of DSSRP (52.68 vs 23.85 and 50.65 vs 23.85, respectively) (Table 6.2). Furthermore, DSFISRP's evaluation times were significantly lower than those of DSSRP (1056.99 vs 1822.70 and 1218.03 vs 1823.02, respectively) (Table 6.3).

In datasets where DSSRP generates high levels of tree depth, DSFISRP creates smaller trees and achieves better classification performance.

6.5 Concluding Remarks

In this chapter, we present the experiments on Varying Feature Spaces comparing DSARF, DSSRP, and DSFISRP.

Overall, all three ensembles had similar kappa values, but DSARF had lower evaluation time and RAM-hours. DSARF was comparable to DSFISRP in AVG Tree Nodes, AVG Tree Depth, and AVG Age.

In experiments with high dimensionality (Text and WAVEFORM on Increment, Parabolic, and Hyperbolic), DSFISRP achieved a higher classification performance while using smaller trees, while both DSARF and DSSRP suffered from excessively growing trees.

Part III

Conclusions

7 Conclusions

As stated Paim & Enembreck (2025), the SRP achieves similar classification performance as the ARF, while inducing the development of deep trees, resulting in a significant increase in processing time and memory consumption. This project proposed a variation of SRP using smaller trees.

The hypothesis of this project is:

- In Varying Feature Spaces in Data Streams scenario, where the SRP's trees grow extensively, the proposed Feature Importance Streaming Random Patches achieve higher classification performance and lower computation cost using smaller trees (i.e., lower amount of nodes and shallower trees).

Experiments demonstrate that the method proposed can mitigate the problems found in SRP. In the Text dataset on Increment, the AVG Tree Depth of DSSRP was 836, indicating an elevated tree growth. DSARF's AVG Tree Depth was 112, and DSFISRP had a significantly lower value of 30, highlighting the use of smaller trees. AVG Tree Nodes were, respectively, 2361, 2491, and 396. DSFISRP's evaluation times and RAM-Hours were also significantly lower than DSSRP's.

In the Text dataset on Parabolic, the AVG Tree Depth of DSSRP was 836. DSARF's AVG Tree Depth was 49, and DSFISRP's was 30. AVG Tree Nodes were, respectively, 2361, 138, and 188. DSFISRP's evaluation times and RAM-Hours were significantly lower than DSSRP's.

In these examples, the kappa values of DSFISRP were greater than twice that of DSSRP's and higher than DSARF's. Therefore, DSFISRP outperformed DSSRP and DSARF in situations where trees grow extensively and dimensionality is a problem. Thus, the hypothesis mentioned before was validated.

In the WAVEFORM on Increment, Parabolic, and Hyperbolic scenarios, DSFISRP achieved significantly higher kappa values than DSSRP and DSARF while having lower AVG Tree Nodes and AVG Tree Depth. However, DSFISRP's evaluation time and RAM-hours were higher due to the extra computational cost of running the ranking learner and the high frequency of resetting its base learners.

DSFISRP has an advantage in scenarios in which both DSARF and DSSRP have trees grow extensively.

Overall, DSARF had similar kappa values compared to DSFISRP and DSSRP, but with lower evaluation time and memory usage. This is a consequence of SRP's deep trees and DSFISRP's protocol increasing processing time and memory consumption. DSFISRP achieved similar values as DSARF on AVG Tree Nodes, AVG Tree Depth and AVG Age indicating the efficiency of the protocol of using smaller and newer trees.

7.1 Future steps

The proposed ensemble has limitations regarding the use of the ranking learner. This learner carries extra processing and increases memory use and evaluation time. Furthermore, as explained in Algorithm 12 in chapter 5, this learner is reset even with only one missing position changed between the previous and current instance. This may cause the learner to be sensitive.

The monitoring of the importance of each feature could be done using the ensemble without the need for an extra learner. This could reduce the computational cost.

The author's suggestions for next steps:

- Changes to the reset protocol of the ranking learner (as described on Algorithm 12), making it less sensible to changes in the missing positions between instances;
- Implement the ranking of feature importance through the ensemble without the need to use a ranking learner. This can be achieved by implementing the MDI score (described in Equation 2.5, in Chapter 2) natively in the ensemble and should reduce evaluation time and memory use;
- Identify and experiment on scenarios characterized by high dimensionality that lead to trees with substantial depth. This is motivated by the results of the Text dataset experiments and it should highlight the difference between DSFISRP and DSSRP.

References

BAHRI, M. *Improving IoT data stream analytics using summarization techniques*. Tese (Doutorado) — Institut Polytechnique de Paris, 2020.

BAHRI, M.; BIFET, A.; GAMA, J.; GOMES, H. M.; MANIU, S. Data stream analysis: Foundations, major tasks and tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley Online Library, v. 11, n. 3, p. e1405, 2021.

BAHRI, M.; MANIU, S.; BIFET, A. A sketch-based naive bayes algorithms for evolving data streams. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 604–613.

BARDDAL, J. P.; ENEMBRECK, F.; GOMES, H. M.; BIFET, A.; PFAHRINGER, B. Boosting decision stumps for dynamic feature selection on data streams. *Information Systems*, Elsevier, v. 83, p. 13–29, 2019.

BARDDAL, J. P.; GOMES, H. M.; ENEMBRECK, F.; PFAHRINGER, B. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, Elsevier, v. 127, p. 278–294, 2017.

BEYAZIT, E.; ALAGURAJAH, J.; WU, X. Online learning from data streams with varying feature spaces. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2019. v. 33, n. 01, p. 3232–3239.

BIFET, A.; GAVALDA, R. Learning from time-changing data with adaptive windowing. In: SIAM. *Proceedings of the 2007 SIAM international conference on data mining*. [S.l.], 2007. p. 443–448.

BIFET, A.; GAVALDA, R. Adaptive learning from evolving data streams. In: SPRINGER. *Advances in Intelligent Data Analysis VIII: 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31-September 2, 2009. Proceedings 8*. [S.l.], 2009. p. 249–260.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: SPRINGER. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I 21*. [S.l.], 2010. p. 135–150.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KRANEN, P.; KREMER, H.; JANSEN, T.; SEIDL, T. Moa: Massive online analysis, a framework for stream classification and clustering. In: PMLR. *Proceedings of the first workshop on applications of pattern analysis*. [S.l.], 2010. p. 44–50.

BISHOP, C. M. *Pattern recognition and machine learning (information science and statistics)*. [S.l.]: Springer New York, 2007.

BLACKARD, J. *Covertype*. 1998. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50K5N.

BREIMAN, L. Bagging predictors. *Machine learning*, Springer, v. 24, p. 123–140, 1996.

BURLUTSKIY, N.; PETRIDIS, M.; FISH, A.; CHERNOV, A.; ALI, N. An investigation on online versus batch learning in predicting user behaviour. In: SPRINGER. *Research and Development in Intelligent Systems XXXIII: Incorporating Applications and Innovations in Intelligent Systems XXIV 33*. [S.l.], 2016. p. 135–149.

CANDILLIER, L.; LEMAIRE, V. Design and analysis of the nomao challenge active learning in the real-world. In: CITESEER. *Proceedings of the ALRA: active learning in real-world applications, workshop ECML-PKDD*. [S.l.], 2012. p. 1–15.

COVERT, I. C.; QIU, W.; LU, M.; KIM, N. Y.; WHITE, N. J.; LEE, S.-I. Learning to maximize mutual information for dynamic feature selection. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2023. p. 6424–6447.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, v. 7, n. Jan, p. 1–30, 2006.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2000. p. 71–80.

FRIEDMAN, N.; GEIGER, D.; GOLDSZMIDT, M. Bayesian network classifiers. *Machine learning*, Springer, v. 29, p. 131–163, 1997.

GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. *Machine learning*, Springer, v. 90, p. 317–346, 2013.

GÉRON, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. [S.l.]: " O'Reilly Media, Inc.", 2022.

GOMES, H. M.; BARDDAL, J. P.; ENEMBRECK, F.; BIFET, A. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 50, n. 2, p. 1–36, 2017.

GOMES, H. M.; BIFET, A.; READ, J.; BARDDAL, J. P.; ENEMBRECK, F.; PFHARINGER, B.; HOLMES, G.; ABDESSALEM, T. Adaptive random forests for evolving data stream classification. *Machine Learning*, Springer, v. 106, p. 1469–1495, 2017.

GOMES, H. M.; MELLO, R. F. de; PFAHRINGER, B.; BIFET, A. Feature scoring using tree-based ensembles for evolving data streams. In: IEEE. *2019 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2019. p. 761–769.

GOMES, H. M.; READ, J.; BIFET, A. Streaming random patches for evolving data stream classification. In: IEEE. *2019 IEEE international conference on data mining (ICDM)*. [S.l.], 2019. p. 240–249.

HE, Y.; WU, B.; WU, D.; BEYAZIT, E.; CHEN, S.; WU, X. Toward mining capricious data streams: A generative approach. *IEEE transactions on neural networks and learning systems*, IEEE, v. 32, n. 3, p. 1228–1240, 2020.

HO, T. K. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, Ieee, v. 20, n. 8, p. 832–844, 1998.

HOCHMA, Y.; LAST, M. Fast online feature selection in streaming data. *Machine Learning*, Springer, v. 114, n. 1, p. 1, 2025.

HOENS, T. R.; POLIKAR, R.; CHAWLA, N. V. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, Springer, v. 1, p. 89–101, 2012.

HOU, B.-J.; ZHANG, L.; ZHOU, Z.-H. Learning with feature evolvable streams. *Advances in Neural Information Processing Systems*, v. 30, 2017.

HOU, B.-J.; ZHANG, L.; ZHOU, Z.-H. Prediction with unpredictable feature evolution. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, v. 33, n. 10, p. 5706–5715, 2021.

KHAN, W.; KONG, L.; BREKHNA, B.; WANG, L.; YAN, H. Online streaming features selection via markov blanket. *Symmetry*, MDPI, v. 14, n. 1, p. 149, 2022.

KUMAR, A.; KAUR, P.; SHARMA, P. A survey on hoeffding tree stream data classification algorithms. *CPUH-Res. J*, v. 1, n. 2, p. 28–32, 2015.

LAVANYA, K.; BAJAJ, S.; TANK, P.; JAIN, S. Handwritten digit recognition using hoeffding tree, decision tree and random forests—a comparative approach. In: IEEE. *2017 international conference on computational intelligence in data science (ICCIDS)*. [S.l.], 2017. p. 1–6.

LI, C.; ZHOU, P.; XIONG, L.; WANG, Q.; WANG, T. Differentially private distributed online learning. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 30, n. 8, p. 1440–1453, 2018.

LOUPPE, G.; GEURTS, P. Ensembles on random patches. In: SPRINGER. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I 23*. [S.l.], 2012. p. 346–361.

MEHTA, S. et al. Concept drift in streaming data classification: algorithms, platforms and issues. *Procedia computer science*, Elsevier, v. 122, p. 804–811, 2017.

NEMENYI, P. B. *Distribution-free multiple comparisons*. [S.l.]: Princeton University, 1963.

OZA, N. C.; RUSSELL, S. J. Online bagging and boosting. In: PMLR. *International Workshop on Artificial Intelligence and Statistics*. [S.l.], 2001. p. 229–236.

PAIM, A. M.; ENEMBRECK, F. Adaptive random tree ensemble for evolving data stream classification. *Knowledge-Based Systems*, Elsevier, v. 309, p. 112830, 2025.

PENG, J.; GUO, J.; YANG, Q.; LU, J.; SHAO, J. A general framework for mining concept-drifting data streams with evolvable features. In: IEEE. *2021 IEEE International Conference on Data Mining (ICDM)*. [S.l.], 2021. p. 1276–1281.

QUINLAN, J. R. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 28, n. 1, p. 71–72, 1996.

RODRIGUEZ-LUJAN, I.; FONOLLOSA, J.; VERGARA, A.; HOMER, M.; HUERTA, R. On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemometrics and Intelligent Laboratory Systems*, Elsevier, v. 130, p. 123–134, 2014.

RUTKOWSKI, L.; PIETRUCZUK, L.; DUDA, P.; JAWORSKI, M. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 25, n. 6, p. 1272–1279, 2012.

SAJEDI, R.; RAZZAZI, M. Data stream classification in dynamic feature space using feature mapping. *The Journal of Supercomputing*, Springer, p. 1–19, 2024.

SCHLIMMER, J. C.; GRANGER, R. H. Incremental learning from noisy data. *Machine learning*, Springer, v. 1, p. 317–354, 1986.

SEDOR, K. The law of large numbers and its applications. *Lakehead University: Thunder Bay, ON, Canada*, 2015.

TAVALLAEE, M.; BAGHERI, E.; LU, W.; GHORBANI, A. A. A detailed analysis of the kdd cup 99 data set. In: IEEE. *2009 IEEE symposium on computational intelligence for security and defense applications*. [S.l.], 2009. p. 1–6.

VERGARA, A.; VEMBU, S.; AYHAN, T.; RYAN, M. A.; HOMER, M. L.; HUERTA, R. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, Elsevier, v. 166, p. 320–329, 2012.

WANG, H.; YOU, D. Online streaming feature selection via multi-conditional independence and mutual information entropy. *International Journal of Computational Intelligence Systems*, Atlantis Press, v. 13, n. 1, p. 479–487, 2020.

WANG, S.; TANG, J.; LIU, H. *Feature Selection*. 2017.

WU, X.; YU, K.; WANG, H.; DING, W. Online streaming feature selection. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 1159–1166.

YOU, D.; LI, R.; LIANG, S.; SUN, M.; OU, X.; YUAN, F.; SHEN, L.; WU, X. Online causal feature selection for streaming features. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, 2021.

YU, K.; WU, X.; DING, W.; PEI, J. Scalable and accurate online feature selection for big data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, ACM New York, NY, USA, v. 11, n. 2, p. 1–39, 2016.

YUAN, L.; PFAHRINGER, B.; BARDDAL, J. P. Iterative subset selection for feature drifting data streams. In: *Proceedings of the 33rd annual ACM symposium on applied computing*. [S.l.: s.n.], 2018. p. 510–517.

ZHANG, Q.; ZHANG, P.; LONG, G.; DING, W.; ZHANG, C.; WU, X. Online learning from trapezoidal data streams. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 28, n. 10, p. 2709–2723, 2016.

ZHOU, J.; FOSTER, D.; STINE, R.; UNGAR, L. Streaming feature selection using alpha-investing. In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. [S.l.: s.n.], 2005. p. 384–393.

ŽLIOBAITĖ, I.; BIFET, A.; READ, J.; PFAHRINGER, B.; HOLMES, G. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, Springer, v. 98, p. 455–482, 2015.