Addressing Feature Drift in Data Streams Using Iterative Subset Selection

Lanqin Yuan
University of Waikato
Hamilton, New Zealand
fyempathy@gmail.com

Bernhard Pfahringer
Deparment of
Computer Science
University of Waikato
Hamilton, New Zealand
bernhard@waikato.ac.nz

Jean Paul Barddal
Programa de Pós-Graduação
em Informática
Pontifícia Universidade
Católica do Paraná
Curitiba, Brazil
jean.barddal@ppgia.pucpr.br

ABSTRACT

Data streams are prone to various forms of concept drift over time including, for instance, changes to the relevance of features. This specific kind of drift is known as *feature drift* and requires techniques tailored not only to determine which features are the most important but also to take advantage of them. Feature selection has been studied and shown to improve classifier performance in standard batch data mining, yet it is mostly unexplored in data stream mining. This paper presents a novel method of feature subset selection specialized for dealing with the occurrence of feature drifts called Iterative Subset Selection (ISS), which splits the feature selection process into two stages by first ranking the features using some scoring function, and then iteratively selecting feature subsets using this ranking. This work further extends upon our prior work by exploring feeding information from the subset selection stage back into the ranking process. Applying our method to the Naïve Bayes and k-Nearest Neighbour classifier, we obtain compelling accuracy improvements when compared to existing works.

CCS Concepts

•Computing methodologies → Supervised learning by classification; Online learning settings; Feature selection; •General and reference → *Experimentation; Evaluation;* **•Information systems → Data streams;**

Keywords

Data Stream Mining; Feature Selection; Concept Drift; Embedded Feature Selection; Iterative Subset Selection; Backward Feature Elimination;

1. INTRODUCTION

Nowadays, many information systems are fed with potentially infinite and continuously generated sequential data. Examples of streaming data include posts on social media, data generated by wearable gadgets, and stock market trades. Motivated by these and several other applications, the data mining community has shifted its attention to streaming scenarios, where novel techniques are proposed every year. Many of the current developments tackle the transient characteristics of data streams, i.e., often the underlying function that maps instances to classes changes over time, thus giving rise to a phenomenon called *concept drift*.

In this paper, we focus on a specific kind of concept drift: feature drift. A feature drift occurs whenever a feature becomes, or ceases to be, relevant to class determination. Recent studies on this topic [6, 7, 4] have shown that the number of techniques that can dynamically determine which features are the most important over time, and that can also take advantage of this information, is rather small currently.

We introduce a novel method for feature selection called Iterative Subset Selection (ISS), which seeks to deal with such feature drifting scenarios. Our proposal is an embedded feature selection method, where the feature selection process is a part of the classification model construction process. We evaluate the effectiveness of ISS at addressing the effects of drift with two well known and widely used classifiers: k-Nearest Neighbours and Naïve Bayes. Additionally, we also evaluate our proposed algorithm against existing works on the topic, highlighting its effectiveness and efficiency.

This paper is an extended version of a work previously presented at the SAC 2018 conference in Pau, France [27]. We expand upon the previous work by investigating the algorithm's ability to correctly select the true number of relevant features. We also introduce and exploring a new method of feature ranking which utilizes information from the classification model.

This work is structured as follows: Section 2 introduces the task of data stream classification, while Section 3 details the type of drift we wish to tackle: feature drifts. Section 4 surveys related works on feature drift adaptation, which are later used during the empirical analysis. Section 5 introduces and details our proposed algorithm. Section 6 contains our extension to the previously presented work and discusses a new ranking method. Our assessment of the algorithm is presented in Section 7. Finally, Sections 8 and 9 provides directions for future work and the conclusions drawn, respectively.

2. DATA STREAM CLASSIFICATION

Learning from ephemeral sequences of data comprises all the problems of conventional batch learning including missing values, noise, class imbalance, and sparsity. On top of that, it introduces further difficulties such as single-pass processing, limited computational resources, and concept drifts [17].

In this paper we focus on the classification task, which regards learning and updating predictive models over time. Let S denote a data stream providing instances in the $i^t = (\vec{x}^t, y^t)$ form, where t is its arrival time stamp, $\vec{x}^t$ is a vector of features from the entire feature set X and $y^t \in Y$ its class. To denote the i^{th} feature of an instance $\vec{x}^t$, we will use the X_i notation, while $\vec{x}_i^t$ denotes the value of this feature for an instance $\vec{x}^t$, where t is dropped if unnecessary. Our ultimate goal in classification is to learn and update a model $h : X \rightarrow Y$ over time, upon the arrival of instances from S.

In streaming scenarios, instances are processed sequentially by the classifier as they arrive, and discarded after processing. Although there is no restriction against buffering and processing instances periodically, this must not jeopardize computational resources. This is important since main memory is finite and its usage must be optimized so that classifiers and their buffers fit into this limited space. Processing time per instance must also be limited. If not, arriving instances will be enqueued until the system crashes due to the lack of memory. Lastly, since data streams are inherently temporal, they are also expected to be ephemeral: the underlying function mapping instances to classes is likely to change, thus giving rise to a phenomenon named **concept drift** [23].

3. FEATURE DRIFT

As described in the seminal work of Widmer [23], data streams are susceptible to different types of drifts, including: (i) changes in the characteristic feature values, (ii) evolution of the value domains of features over time, (iii) *features that were once important and that now may become meaningless or the other way around*, and so on. In this paper, we tackle the above-emphasized type of drift, called **feature drift**. A feature drift occurs when a subset of features becomes, or ceases to be, relevant to the learning task [6].

To formalize feature drifts, we must first be able to discern between relevant and irrelevant features [29]. Given the entire feature set X, and subsets $S_i = X \setminus \{X_i\}$, then a feature X_i is **relevant** *iff* Definition 1 holds; otherwise, it is said to be **irrelevant**.

Definition 1. A feature X_i is relevant *iff* $\exists S_i' \subset S_i$, such that $P[Y|X_i, S_i'] > P[Y|S_i']$ holds.

Following the previous definition, if an arbitrary relevant feature is removed, then it will result in a reduction of overall prediction power, because (i) it alone is strongly correlated with the class, or (ii) it forms a feature subset with other features that together are correlated with the class (this concept is commonly referred as feature interaction [18]).

A feature drift occurs whenever a subset of features becomes, or ceases to be, relevant to class prediction. Given X at a timestamp t, we are able to select the ground-truth relevant features $X^* \subseteq X$ such that $\forall X_i \in X^*$ Definition 1 holds and $\forall X_j \in X \setminus X^*$ the same condition does not. A feature drift then occurs if, at a timespan between t_i and $t_j = t_i + \Delta$, X^* at t_i differs from X^* at t_j.

Like other types of drift, changes in the relevant subset of features X^* affect the ground-truth decision boundary. Ideally, we expect classification models to detect and adapt to changes in X^*, while changes may be (i) radical, where the classifier would forget its entire model and learn from scratch; or (ii) more slowly paced, where its model would be gradually updated given drifts [21].

4. RELATED WORK

Determining which features are relevant has been widely discussed in batch learning. However, the same cannot be said about learning in streaming environments. Recently, the surveys of [5, 6] highlighted that there are few works that explicitly tackle the problem of drifting features, and these basically consist of decision trees [13], rule learning [3] and ensembles [2, 21]. Furthermore, [6] provides a comprehensive evaluation of some stream classifiers, concluding that a single adaptive decision tree, namely the Hoeffding Adaptive Tree (HAT) [9], was the best performing classifier over all w.r.t. the trade-off between classification rate, processing time and memory usage. The HAT is an extension to the incremental Very Fast Decision Tree classifier [13], which uses the ADWIN drift detector [8] inside decision nodes to monitor the internal error rates of the tree. If a feature that is being used by a decision node becomes irrelevant, one would expect the error rate to increase, and ADWIN will flag a change. Consequently, the affected decision node will reset the entire sub-tree, replace the irrelevant split with a better one, potentially identifying and selecting a newly relevant attribute, and restart growing a new sub-tree. This allows the HAT classifier to detect and overcome feature drifts quickly and effectively.

More recently, the authors of [7] proposed a dynamic weighting scheme for the problem of classification over data streams. In that work, the estimation of the discriminative power of each feature is updated using a sliding window approach. Feature evaluation was performed using the Symmetrical Uncertainty scoring operator. The discriminative power of each feature was used as a weighting factor in the prediction process of both k-Nearest Neighbours and Naïve Bayes classifiers, which resulted in useful accuracy gains, at the expense of a reasonable amount of additional processing time and memory usage. The same authors proposed in [4] the use of the same incremental operators for performing actual feature selection with a method called DISCUSS. The results obtained show that DISCUSS is able to improve kNN and Naive Bayes learners in scenarios where the relationship between features and the class are mild, meaning that no higher-order interactions exist.

An algorithm called INTERACT, similar to our proposed method, has been proposed in [28]. The main differences between ISS compared to INTERACT are that ISS wraps around and embeds the feature selection process into the classifier to select feature subsets and that ISS is incremental.

Finally, it is worth mentioning that feature drifts have also been recently investigated in the context of regression, such as in the work of [14]. However, regression is outside the scope of the work presented here.

5. ITERATIVE SUBSET SELECTION

A naïve brute force approach to the problem of identifying irrelevant features in a stream would be: "for every new instance arriving in the stream, exhaustively search through every possible subset of features, evaluate each of these subsets using some goodness of fit criterion, and select the features of the best performing subset". Such an algorithm can implicitly deal with changes in feature relevance as the classification for each example is independent. In practice, however, such an algorithm would be unfeasible as it would be extremely inefficient and prohibitively expensive, regardless of working in a batch or in a streaming setting.

Our proposed method improves upon this naïve approach: features are first ranked using a scoring function, based on the existing batch feature selection method called ESFS [15, 25]. By doing this, only a limited linear number of subsets is evaluated. We embed the method into the classifier and use it directly as our goodness criterion. Sections 5.1 and 5.2 detail the ranking and subset selection steps, while Sections 5.3 and 5.4 show how these steps are embedded within kNN and Naïve Bayes, respectively.

5.1 Ranking

Algorithm 1 shows the pseudocode for the ranking of features. Initially, the relevance of each feature is determined by individually and independently evaluating the feature w.r.t. class prediction, using some scoring function. Features are sorted by their discriminating power using the score, allowing for irrelevant features to be filtered out; only the top f number of features (where f is a parameter) are kept for use in the next stage of subset selection, whereas features below the top f ranks are ignored. In general, it is recommended to set f to cover most, if not all features in the stream, so the method is able to adapt to drastic drifts. Smaller f values reduce the search space and can be used if there exists knowledge of the nature of features in the data stream before evaluation. Rankings are recomputed periodically every r instances to reduce computational load, using a sliding window of instances, resulting in each ranking being independent. Thus the method is able to implicitly deal with drift as only new instances in the window are used.

The use of prior knowledge was also explored in using information from the subset selection stage to scale the scores generated by the ranking function. This is discussed in more detail later in Section 6. The three ranking functions used in this work are the Average Euclidean Distance, Information Gain, and Symmetric Uncertainty [24].

Average Euclidean Distance (AED) is a simple ranking function based on the idea that after normalization, features which have the most influence on the decision of the class value will be on average further apart when plotted in Euclidean space. For example, for the problem of identifying the gender of adult Mallard ducks we would expect the colour of a duck's head hair to on average be far apart between male duck examples and female duck examples when plotted in euclidean space, as only male ducks grow green head hair baring genetic defects. For another feature, say the size of the duck, we would expect there to be on average less distance between adult males and females as a wider variety of factors can influence the size of the duck other than the gender, such as nutrition, environment, or genetics; whereas gender is the sole determinant for whether a adult Mallard duck grows green head hair. The calculation of AED is different depending on whether an attribute is nominal or numeric, while not requiring discretization as the other ranking functions. The numeric formula of AED is given in Equation 1, where X_i is a numeric attribute in a stream with L classes, $N_c(X_i)$ is the number of times that the class value c appeared in the window with a valid value for X, where MV is the mean value given by Equation 2, and X_c^n is the value of X for the n-th instance with a class value of c.

Algorithm 1 Ranking of features

Input:
W: Window of Stream instances
R: Scoring function
Output:
r: List of features sorted by best to worst score

```
1: function RankFeatures(W,R)
2:     S ← ∅
3:     for all Feature f in W do
4:         for all Instance w ∈ W do
5:             S_f ← S_f + R(w_f)
6:         end for
7:     end for
8:     Sort(S)
9:     return S
10: end function
```

$$\mathrm{AED}_{\mathrm{num}}(X_i) = \sqrt{\sum_{0 \le c < j < L} \left(\mathrm{MV}(X_c) - \mathrm{MV}(X_j)\right)^2} \quad (1)$$

$$\mathrm{MV}(X_c) = \frac{1}{N_c(X)} \sum_{n=0}^{N_c(X)} X_c^n \quad (2)$$

The AED formula for a nominal attribute is defined by Equation 3, where X_i is a nominal attribute with V categories in a stream with L number of known classes, X_{cv} is the number of instances with a c-th class label and variable value v, and $C_c(X_i)$ is the total number of instances with a label c with a valid (non-missing) value for X_i.

$$\mathrm{AED}_{\mathrm{nom}}(X_i) = \sum_{0 \le c < j < L} \left[\frac{1}{V} \sum_{v=0}^{V} \sqrt{\left(\frac{X_{cv}}{C_c(X_i)} - \frac{X_{jv}}{C_j(X_i)} \right)^2} \right] \quad (3)$$

Information Gain (IG), a synonym for the Kullback-Leibler divergence [19], is a widely used measure in machine learning and data mining. It measures the reduction of entropy by the introduction of a variable. As it is only possible to calculate Information Gain for nominal attributes,

discretization is required for numeric attributes. The discretization algorithm used in this work was PiD [16], a single pass discretization algorithm specialized for data streams. The formula used to calculate IG for an attribute X_i and a class attribute Y is defined as follows:

$$\mathrm{IG}(Y|X_i) = H(Y) - H(Y|X_i) \tag{4}$$

where $H(X_i)$ is the entropy for an attribute X_i with N distinct values is given by the following formula:

$$H(X_i) = \sum_{j=0}^{N} -P(X_i = x_j) \log P(X_i = x_j) \tag{5}$$

and $H(Y|X_i)$ is the conditional entropy, given by:

$$H(Y|X_i) = \sum_{j=0}^{N} p(x_j) H(Y|X_i = x_j) \tag{6}$$

Symmetric Uncertainty (SU) is a normalized version of Information Gain, and has been shown to give good performance for feature selection [26]. Its main advantage over Information Gain is that it overcomes the bias in the data (when there is significantly more or less of some class/classes than others). Again, Symmetric Uncertainty can only be calculated for nominal attributes, and thus, discretization is required for numeric attributes. The formula for Symmetric Uncertainty for an attribute X with the class attribute Y is is defined as follows:

$$\mathrm{SU}(X, Y) = \frac{IG(X|Y)}{H(X) + H(Y)} \tag{7}$$

5.2 Subset Selection

As streaming environments require algorithms to be fast, so to keep up with the stream, an exhaustive search of all combinations like the one mentioned in the aforementioned naïve method is out of the question. Therefore, we utilize the ranking to reduce the search space and Backward Feature Elimination [24] to select the best feature subset. As a consequence, we limit the search to a maximum of f iterations, meaning that the growth of the number of evaluations is linear in f rather than exponential, thus also avoiding a potential combinatorial explosion. Backward Elimination is selected over Forward Selection due to its potential for a slight optimisation for the k-Nearest Neighbour classifier while giving the same results. This is discussed in more detail in Section 5.3. Initially, a subset S that contains all f ranked features is evaluated and its prediction result recorded. The lowest ranked features are iteratively removed from S, and the new smaller S is re-evaluated until S is empty. The best subset is selected based on an estimate of the accuracy achieved by each of the subsets by maintaining counts of the number of times a subset correctly predicts the class when the classifier uses only features in the given subset. These counts are fuzzy counts as they are subject to a decay factor d to allow for implicit adaptation to change. Counts are tied to the size of the subset rather than the composition of the subset, similar to how counts are kept in the Space Saving Algorithm [20], thus reducing memory usage and complexity. The accuracy estimate for a subset S is computed using the simple formula of $\frac{\#\text{ of correct predictions with S}}{\text{total }\#\text{ of predictions with S}}$ and the final prediction of the classifier selected is based on the subset with the current best estimate. While only the final prediction is output, counts are updated for all subsets. The decay factor of the counts is a user-given parameter d and is applied at fix intervals i.

Algorithm 2 k-Nearest Neighbours Classifier Subset Selection

Input:
a: Array of accuracy estimates for each subset size
h: Size of Hill climbing window
b: Size of previous best subset
f: Upper bound of subset size
r: List of ranked features sorted descending from most relevant to least relevant
Output:
q: Final classification result after feature selection
b: Size of new best subset

```
1: function SelectSubsetKNN(a,h,b,f,r)
2:     l ← b − h; l ≥ 1            ▷ Lower bound of subset size
3:     u ← b + h; u ≤ f            ▷ Upper bound of subset size
4:     for i from 1 to u do
5:         S ← r[i]                ▷ fill S with ranked features
6:     end for
7:     for i from u to l do
8:         p ← kNN classification result using only features in S
9:         Update a[size(S)]
10:        S ← S − r[i]            ▷ Remove bottom ranked feature from S
11:        if i = b then
12:            q ← p
13:        end if
14:    end for
15:    b ← index of max(a)         ▷ Set new best subset size
16:    return q, b                 ▷ b is used for the next iteration of subset selection
17: end function
```

5.3 k-Nearest Neighbours Classifier

k-Nearest Neighbours (kNN) was one of the classifiers we explored. The implementation of the ISS subset selection for kNN is shown in Algorithm 2.

kNN classifies the target instance based on a majority vote by its k nearest neighbors' class values. Euclidean distance is often selected as the distance measure due to its simplicity. The cumulative nature of Squared Euclidean distance also allows for easy maintenance of distances for each instance in the window. The kNN classifier itself can utilize a slight optimisation from backward elimination using the knowledge that the Euclidean distance between the target instance and instances in the sliding window are normalized. Due to normalization, the absolute difference in squared Euclidean distance for each new feature added or taken away

is bounded between 0 and 1, meaning we can separate instances into 3 subsets: (i) set A where instances' distances are updated and the instances used in kNN evaluation, (ii) set B where instances' distances are kept up to date but the instances are ignored in kNN evaluation; and (iii) set C where instances are ignored entirely and their distances not kept up to date. Instances are able to move between A and B freely, and from B to C, with instances unable to leave C once they enter. Initially all instances begin in A. For a target instance s_p and a window S, we define d_i as the distance between s_p and some $s_i \in S$. d_k is the distance of the current k-th nearest neighbour. We calculate $n_i = \lceil d_i - d_k - 1 \rceil$ for $s_i \in S$, the minimum number of features that need to be removed for s_i to be closer to s_p than s_k, under the naïve assumption of the best case that d_i decreases by the maximum of 1, and that all other $d \in D$ remains the same. If $n_i > 0$, then we can safely ignore s_i for the next n_i iterations as it is impossible for s_i to be closer to s_p than s_k until at least n_i features are considered. s_i in this case moves from A to B and is unable to return to A the next n_i iterations of subset selection. Instances in B still need their distances kept up to date as after n_i iterations, s_i may rejoin A and n_i is recomputed again, with the algorithm repeating until all f subsets have been explored.

After $f/2$ total evaluations, it becomes possible to eliminate instances from further search as it becomes impossible for s_i to return to A from B if n_i is greater than the number of iterations left (l). In this case, s_i would move B to C if $n_i \geq l$, thus eliminating s_i from any further distance updating as it will no longer be evaluated. Forward selection would not allow for this optimisation to the same degree as all distances would initially start between 0 and 1 for the initial subset of only the top-ranked feature, meaning it is not possible to determine which instances can be placed into B initially.

Despite this and other optimisations, it was found that searching through every subset size for data streams is still generally computationally prohibitive, as streams are prone to dramatic drifts and may have a large number of features, which subsequently requires a large f parameter to be able to adapt to change. While the ranking is iterative and computationally inexpensive, the sets A and B are still potentially very large even after several iterations of subset selection. In the worst case, subset selection runs the kNN search algorithm f times for every new instance, which results in a complexity of $\mathcal{O}(kfw)$ where w is the size of the window containing the instances, and k is the number of nearest neighbours.

To alleviate this problem, the simple algorithm of Hill Climbing is utilized in a novel way to select the subset size and limit the number of subsets evaluated. Hill Climbing is a naïve greedy local search algorithm which incrementally and greedily selects the best solution from the search frontier. While Hill Climbing is known to be prone to becoming stuck in local optima, the occurrence of local optima is found to be mitigated by first initially sorting the features in the ranking stage, so long as the ranking function can sort relevant features above irrelevant features. Instead of searching across all subset sizes from 1 to f, the search frontier is limited by only evaluating subsets with a difference in a number of features of at most $\pm h$ around the selected subset size from the previous prediction, with h being the hill climb window which controls the number of subset sizes searched around the previously selected size. The complexity of the kNN search is therefore bounded by $\mathcal{O}(kw(2h+1))$, which allows experiments to complete in reasonable time. It was found that Hill Climbing caused a very negligible decrease (less than 1%) in overall accuracy compared to a static f comprising all features in the stream due to slightly worse initial accuracy, as the algorithm needs time to "climb".

5.4 Naïve Bayes Classifier

The Naïve Bayes classifier is a probabilistic classifier that determines its classification result using Bayes' theorem while making a strong (naïve) independence assumption between each of the features in a stream, given the class value. ISS works well with the Naïve Bayes classifier as Bayesian probability is cumulative when features are assumed to be independent. This means that the next smaller feature subset is easily and cheaply computed by removing the probabilities of the least desirable feature from the current total probabilities. As a result, the Naïve Bayes classifier does not face the same performance limitations that the kNN classifier does. Therefore, heuristic search, such as hill climbing, is not required to ensure reasonable run-times, even for large f. The implementation of ISS subset selection for the Naïve Bayes classifier is shown in Algorithm 3. A point to note is the necessity for the algorithm to maintain a separate sliding window for ranking as unlike the kNN classifier, the NB classifier does not use sliding windows by default.

6. UTILIZING ACCURACY DIFFERENCE BETWEEN SUBSETS

In the previously presented paper, a perceived improvement mentioned in the future works section was to utilize information from the subset selection stage in some manner. Currently, the ranking stage starts from scratch every time and does not utilize any information from the subset selection stage, or any prior knowledge for that matter. To explore ways to exploit this information, we looked at using the difference between the accuracy estimate of each subset to scale the ranking score. As each subset maintains its own accuracy estimate, we can easily obtain the change in the accuracy estimate caused by the iterative removal of a feature f from the subset by comparing the accuracy estimate of the subset containing f to the subset without f. We make the assumption that classification accuracy is expected to decrease for relevant features when removed from the subset, with the opposite happening for irrelevant features. As such, we attempt to penalize or reward features based on whether they decrease or increase the accuracy estimate when removed from the feature subset used for classification. We refer to this ranking as Accuracy Difference Scaled Ranking (ADSR). Algorithm 4 shows the pseudo-code of the ADSR ranking function. Given accuracy estimates of the subset sizes as features are iteratively removed A, and i the size of the subset for which has f as the lowest ranked feature, we define the accuracy difference d_f as $A_i - A_{(i-1)}$. We feed this accuracy difference back into the ranking function by multiplying the ranking score by a scalar determined by the accuracy difference. A simple scalar of $c * (1 - d_f) * R(f)$ is

Algorithm 3 Naïve Bayes Classifier Subset Selection

Input:
a: Array of accuracy estimates for each subset size
w: Ranking window
b: Size of previous best subset
f: Upper bound of subset size
r: List of ranked features sorted descending from most relevant to least relevant
Output:
q: Final classification result after feature selection
b: Size of new best subset

function SELECTSUBSETNB(a,b,w,f,r)
  for i from 1 to f **do**
    $S \leftarrow S + r[i]$ ▷ Fill S with ranked features
  end for
  for i from f to 1 **do**
    $p \leftarrow$ NB classification result using only features in S
    Update a[size(S)]
    $S \leftarrow S - r[i]$ ▷ Remove bottom ranked feature from S
    if $i = b$ **then**
      $q \leftarrow p$
    end if
  end for
  add instance to w
  while w's size $\geq$ w's max capacity **do**
    Remove oldest instance in w
  end while
  $b \leftarrow$ index of max(a) ▷ Set new best subset size
  return q, b ▷ b is used for the next iteration of subset selection
end function

used, where R is the ranking function, and c is a user defined scalar. As the accuracy difference is always bound between -1 and 1, the magnitude of the scaling is at most $2c$.

A user-given confidence threshold t is used to adjust to noise, with the scaling only occurring if $|d_f| > t$. We also explored only penalize features without rewarding features with a $d_f < 0$.

7. EVALUATION

In this section, we compare the average accuracy obtained by classifiers, their processing time, and the memory (RAM-Hours) required to process each of the tested data streams.

Experiments were conducted using the Massive Online Analysis (MOA) framework [10]. A large combination of ISS' parameter configurations were explored for both the kNN and NB classifiers. The parameters explored were: the interval at which decay occurs for the accuracy counts i (500, 1000, and 1500 were tested), the decay factor of the accuracy counts d (0.05, 0.1, 0.15, 0.2, and 0.25 were tested), the window size used by ranking functions w (500, 1000, and 1500 were tested), and the interval at which ranking occurs r (500, 1000, and 1500 were tested). The number of ranked features f was set to encompass all features in the stream. ADSR was not utilized to scale ranking scores except for

Algorithm 4 Accuracy difference scaled ranking function

Input:
W: Window of Stream instances
R: Scoring function
D: Array of accuracy estimate difference from removing each feature to the subset
c: Scalar
o: Penalize only indicator
t: Confidence threshold
Output:
r: List of ranked features sorted descending from most relevant to least relevant

function RANKACCURACYDIFFERENCE(W,D,R,c,o,t)
  $S \leftarrow RankFeatures(W, R)$
  for all Feature $f \in W$ **do**
    if $|D_f| > t$ **then**
      if o =false **or** (o = true **and** $D_f > 0$) **then**
        $S_f \leftarrow S_f * (1 - D_f) * c$
      else
      end if
    end if
  end for
  $Sort(S)$
  return S
end function

the experiments mentioned in Section 7.4. The kNN with ISS specific hill climb window size h parameter, which controls the number of subsets searched around the current best size, was also explored with sizes of 2, 4 and 6. The source files of the experiments conducted can be found online at https://github.com/EMPaThy789/ISS-MOA.

Since accessibility to real-world data streams was limited, artificially generated data streams were used in their steed. We employed the following stream generator which feature gradual feature drift: AGRAWAL [1], Assets [7], BG [5], LED [10] and SEA [6, 22]. Sudden feature drift was tested using a custom made generator for this work called the Conditional Generator [27]; it is available in the project repository.

The Conditional generator generates classes with certain attributes associated with each class. Each attribute associated with a class has either some valid range or some set of possible values specified, depending on whether the attribute is numeric or nominal. Each instance is generated by first generating a class, and then randomly re-setting the associated attributes to valid ranges for that class. Drift are introduced by re-associating attributes and their valid ranges. The streams generated using this generator were: Conditional, FY A, FY B, FY C, and FY D, with each stream increasing in complexity. The number of features relevant to the classification problem out of all features for each stream is shown in Table 1. The first three columns of the table show the number of feature relevant to the classification problem for each stream concept, and the last column shows the total number of features in each stream. FY A and FY B both feature no drift and were repeated as separate streams to encompass feature drifts, and are referred to as FY A drift and FY B drift.

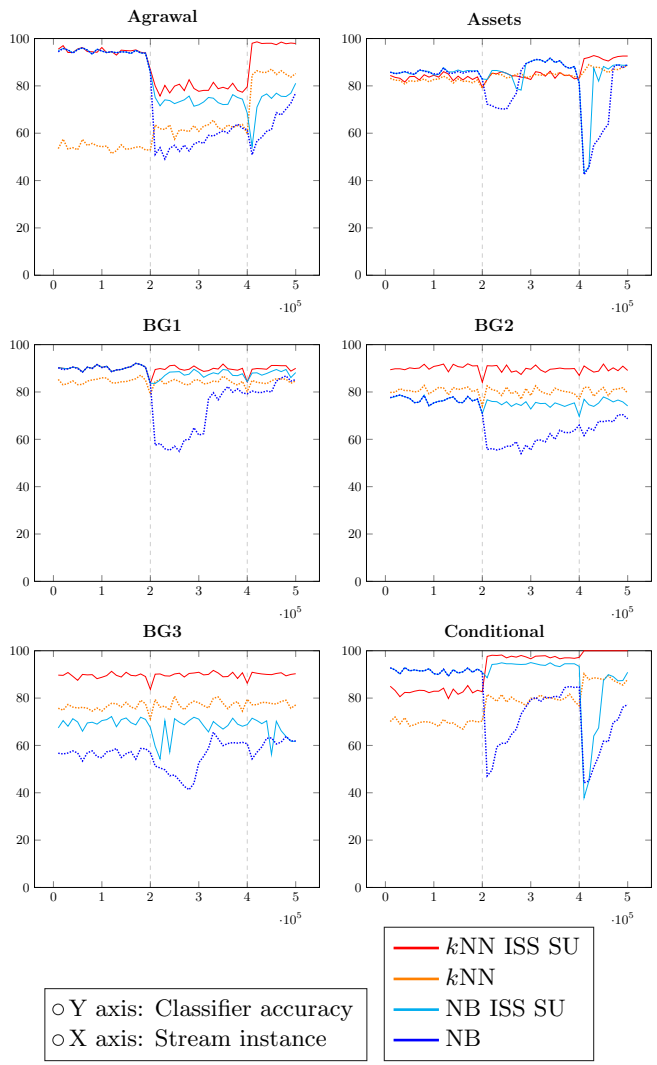


Figure 1: Time series plots for experiments' accuracy (%) against stream instances.

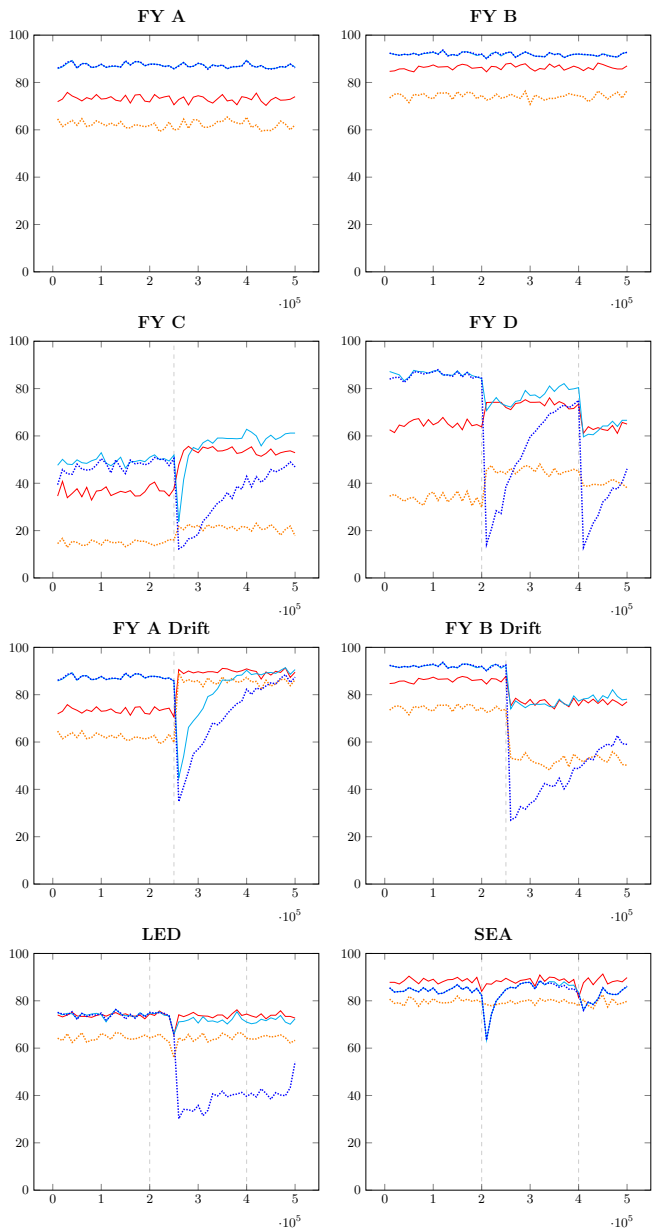


Figure 2: Time series plots for experiments' accuracy (%) against stream instances continued. Refer to 1 for legend.

Each experiment consisted of running the classifier on a generated data stream capped at 500,000 instances of data. Points for feature drift were set at every 200,000 instances for all feature drifting streams, except the LED, FY A drift, FY B drift, and FY C, where only a single drift occurs after 250,000 instances.

A large number of classifier parameters were tested and are broadly discussed in this section. We focus our analysis on a default parameter configuration featuring a window size w of 1000, a ranking interval r of 500, a decay rate d of 0.1, a decay interval i of 1000. This particular configuration was selected based on the trade-off between accuracy and computational resources. A hill climbing window h of 4 was selected as the default for experiments using the kNN classifier.

Finally, statistical significance across methods is verified using Wilcoxon's test, or a combination of Friedman and Nemenyi's tests depending on the number of methods compared [12]. All tests assume a confidence level of 95%.

7.1 k Nearest Neighbour Classifier

Across all parameter configurations for the kNN classifier, it was found that the three ranking functions (AED, IG, and SU) performed relatively similarly in most cases in terms of classification accuracy, with the IG and SU ranking functions tending to have slightly better results. An exception is the AGRAWAL scenario, for which the AED ranking function performed noticeably worse than the other two ranking functions, though still better than kNN standalone. For runtime and memory usage, it was found that the AED ranking

Table 1: Number of features relevant to the classification problem for each concept across all data streams used in this work.

	Concept 1	Concept 2	Concept 3	Total features
AGRAWAL	1	3	3	9
ASSETS	4	3	4	5
BG	3	3	3	10
BG 2	3	3	3	15
BG 3	3	3	3	15
CONDITIONAL	18	7	15	40
FY A	19	-	-	40
FY B	13	-	-	120
FY C	6	6		106
FY D	20	18	17	120
FY A Drift	19	16	-	40
FY B Drift	13	15	-	120
LED	7	7	-	24
SEA	2	2	2	13

function tended to take longer and use more memory than the SU and IG ranking functions. The IG and SU ranking functions gave very similar results for both accuracy, runtime, and memory usage, with SU tending to have a slight edge. Table 2 compares the accuracy results for the different ranking functions with the default parameter configuration which supports this general trend. Due to these aforementioned advantages, we focused on the SU ranking function in this work and use it as the default ranking function for experiments.

We compare the default configuration's results to the baseline kNN classifier's results, which used a k of 10 and a window size of 1000. Full results of experiments conducted for this work can be found and reproduced on the GitHub repository mentioned in the previous section.

A comparison between the default configuration and the kNN standalone classifier is shown in Table 3. The results for kNN with ISS are displayed in black with the increase/decrease in performance from kNN without ISS and kNN with ISS displayed in the number in brackets, with blue indicating better results for ISS, and red indicating worse. We observe that the kNN classifier benefits from ISS feature selection in classification accuracy in all experiments for the default configuration. These improvements are supported by Wilcoxon's test which indicates statistically significant results. Figures 1 and 2 show time series graphs of the classification accuracy plotted against the number of instances stream for each experiment conducted in this work, with the occurrence of feature drifts marked by gray the vertical lines. We observe improvements for kNN with ISS, which plots a line which mirrors the shape of standalone kNN classifier plot and is almost always of a higher accuracy at all stages of the stream across all scenarios.

Runtime was generally slower, except for the streams generated by the Conditional generator. There no significant differences were found. Memory usage was also lower for ISS for the BG, LED, and SEA data streams: an improvement in RAM-Hours was observed despite slower runtime results. The difference again is significant according to a Wilcoxon's paired test.

Overall, all configurations of kNN experiments conducted in this work produced a higher average accuracy than the

Table 2: ISS experiment accuracy results (%) for different ranking functions using the default configuration with the kNN classifier.

Stream	AED kNN-ISS	IG kNN-ISS	SU kNN-ISS
AGRAWAL	77.15	**88.97**	**88.97**
ASSETS	85.42	**85.61**	85.53
BG	89.78	**89.79**	**89.79**
BG2	**89.77**	**89.77**	89.76
BG3	**89.59**	**89.59**	**89.59**
CONDITIONAL	91.14	91.84	**91.86**
FY A	71.29	**72.87**	**72.87**
FY B	84.98	**85.35**	**85.33**
FY C	41.08	**44.99**	**44.99**
FY D	**67.98**	67.96	67.90
FY A Drift	81.24	**81.27**	**81.27**
FY B Drift	80.52	80.52	**80.53**
LED	73.60	**73.64**	73.62
SEA	88.28	88.28	**88.29**

Table 3: ISS experiment results using default configuration for the kNN classifier using the SU ranking function.

Stream	Avg Accuracy (%)	CPU Time (s)	RAM-Hours (kb-hour)
AGRAWAL	88.97 (+25.34)	353.31 (+153.63)	63.58 (+28.04)
ASSETS	85.53 (+1.41)	304.95 (+153.00)	41.13 (+20.79)
BG	89.79 (+5.45)	456.48 (+236.69)	86.43 (-53.03)
BG 2	89.76 (+9.72)	517.89 (+53.49)	152.55 (+16.24)
BG 3	89.59 (+12.62)	565.14 (+91.24)	166.47 (+125.05)
CONDITIONAL	91.86 (+14.61)	735.15 (-559.53)	341.12 (-252.37)
FY A	72.87 (+10.69)	770.34 (-788.02)	443.87 (-440.08)
FY B	85.33 (+11.11)	1621.89 (-4640.54)	2713.85 (-7602.82)
FY C	44.99 (+26.96)	745.03 (-4758.63)	1149.98 (-7077.83)
FY D	67.90 (+28.35)	1384.43 (-4978.44)	2356.87 (-8125.98)
FY A Drift	81.27 (+7.40)	1000.18 (-408.01)	580.64 (-224.66)
FY B Drift	80.53 (+17.21)	1576.40 (-3964.14)	2644.58 (-6506.91)
LED	73.62 (+9.29)	824.93 (+69.26)	326.19 (-29.44)
SEA	88.29 (+8.75)	426.26 (+151.83)	98.32 (-35.83)

kNN baseline classifier. The impact of parameters on overall average classification accuracy is varying depending on the streams. Most experiments had only slight differences ($<$ 2%) in overall classifier accuracies. In 4 streams (AGRAWAL, BG2, SEA, and LED), parameters had little impact on the overall mean classifier accuracy, with only around a 1% difference in overall mean accuracy between the best performing ISS experiment and the worst performing ISS experiment tested in this work.

While it was found that accuracy tended to be largely similar across all configurations, runtime and RAM-hours performance was found to be much more dependent on the parameters set for ISS. It was found that the size of the w had the most impact on computation time and RAM-Hours, as a larger w meant more instances to search through for each kNN evaluation. The size of the hill climb window h also had a noticeable impact on computation time and RAM-Hours, as a larger window hill climbing window means that more subsets were searched at every iteration; however, this was found to be secondary to the size of w.

7.2 Naïve Bayes Classifier

As with the kNN classifier, we again focus on a default configuration of ISS for comparison. Other parameter configurations were also explored but were found to have similar performance for accuracy, runtime, and memory usage. Using hill climbing to reduce search space was also explored

but results indicated that the gains in runtime and memory usage were minimal, at a slight cost of prediction accuracy. Due to this, hill climbing was not used for experiments using the Naïve Bayes classifier.

We examined results for all three ranking functions (AED, IG, and SU) for the NB classifier, the results shown in Table 4. We found all three ranking functions to be very similar in their overall performance, with differences between accuracy being less than 1%, run time less than 1 second, and RAM-hours less than 0.8 kb-hour for all ranking functions on all streams.

The accuracy results for the default configuration are presented in Table 5. As with the kNN tables, the results for NB with ISS are displayed in black with the increase/decrease in performance from NB without ISS and NB with ISS displayed in the number in brackets, with blue indicating better results for ISS, and red indicating worse. Concerning accuracy, the default configuration of NB with ISS outperformed NB standalone by a noticeable amount in most of the streams tested. The exceptions to this were the FY A, and FY B scenarios, which featured no feature drift, where NB performed within 0.1% accuracy to NB with ISS. Despite these minor accuracy losses, Wilcoxon's test showed significant improvements when ISS is used with NB classification.

Again referring to Figures 1 and2 which show the plotted classification accuracy against the stream for each experiment, we notice that for the FY A and FY B scenarios, the NB and NB ISS lines overlap across the entire plot. We also observe that in all scenarios except the BG3 scenario, NB with ISS performs very closely to NB standalone before the occurrence of any feature drift; the two configurations diverge however upon the occurrence of drift with NB with ISS adapting significantly faster than standalone NB across all scenarios.

Regarding computational resources, we notice that NB with ISS was always slower than NB by itself, and also always used more memory. The differences in run time can be attributed to the high speed of the NB classifier which means that the reduction in computation time gained from feature selection is still not as large as the overhead runtime cost incurred from ISS. Despite these increases being statistically significant, the runtime and memory usage for NB with ISS is still within reasonable limits and the classifier still has much significantly better runtime and memory usage compared to the kNN classifier.

7.3 Feature selection accuracy

This section discusses ISS' performance in selecting the right number of features. As we used synthesized data for all experiments conducted in this work, the true number of features relevant to the classification problem of each stream is known; with this knowledge in hand, we examine ISS' ability in selecting the correct number of features by comparing the size of the subset of relevant features selected to the true number of relevant features. Taking the weak assumption that the ranking stage is able to order relevant features above irrelevant features in at least some manner, we assume that a subset size closer to the true number of relevant features would indicate good performance for the algorithm. We compare only the size and not the composition as ISS' feature subset selection searches for a feature subset size rather than a feature subset composition.

Table 4: ISS experiment accuracy results (%) for different ranking functions using the default configuration with the NB classifier.

Stream	AED NB-ISS	IG NB-ISS	SU NB-ISS
AGRAWAL	81.51	**81.80**	81.78
ASSETS	85.29	84.51	**84.85**
BG	88.55	**88.57**	88.55
BG2	**75.93**	75.88	75.89
BG3	**68.15**	68.13	68.06
CONDITIONAL	89.05	**89.25**	89.10
FY A	**87.15**	87.12	87.13
FY B	**91.88**	91.84	91.84
FY C	**52.81**	52.6	53.01
FY D	**78.09**	77.86	77.92
FY A Drift	**84.98**	84.53	84.47
FY B Drift	84.57	84.57	**84.62**
LED	72.82	**72.79**	**72.79**
SEA	83.79	**83.90**	**83.90**

Table 5: ISS experiment results using default configuration for the NB classifier using the SU ranking function.

Stream	Avg Accuracy (%)	CPU Time (s)	RAM-Hours (kb-hour)
AGRAWAL	81.78 (+8.46)	3.00 (+1.60)	0.54 (+0.54)
ASSETS	84.85 (+3.86)	17.66 (+1.04)	2.39 (+2.38)
BG	88.55 (+8.76)	2.32 (+1.48)	0.44 (+0.44)
BG 2	75.89 (+8.94)	3.64 (+2.07)	1.08 (+1.08)
BG 3	68.06 (+11.68)	3.55 (+2.37)	1.05 (+1.05)
CONDITIONAL	89.10 (+11.13)	8.43 (+5.66)	3.93 (+3.92)
FY A	87.13 (0.00)	12.08 (+6.82)	7.03 (+6.99)
FY B	91.84 (-0.04)	41.75 (+25.49)	70.57 (+70.22)
FY C	53.01 (+12.30)	58.21 (+22.91)	92.59 (+90.76)
FY D	77.92 (+15.88)	66.83 (+26.86)	117.44 (+115.09)
FY A Drift	84.47 (+4.97)	12.07 (+6.60)	7.01 (+6.98)
FY B Drift	84.62 (+15.61)	41.60 (+25.98)	70.25 (+69.92)
LED	72.79 (+16.35)	5.50 (+3.18)	2.19 (+2.18)
SEA	83.90 (+0.14)	4.18 (+2.51)	0.97 (+0.97)

Tables 6 and 7 show the results of the kNN and NB classifiers using the standard ISS configuration. The columns of the tables are split into each stream concept in the first row. The second row of the tables split the results into the percentage of predictions that the algorithm made which correctly predicted the true number of features, and the percentage of predictions predictions within two to the true number of relevant features for each stream concept. We observed the number of classifications made where the subset was: the same size as the true number of relevant features, and within ± 2 of the true number of features.

We found that for most cases, ISS was not always able to correctly identify the true number of relevant features for either of the classifiers. The kNN classifier in general had better performance than the NB classifier in this regard, with the the kNN classifier making 90%+ of predictions using a subset size the same as the true number of relevant features for AGRAWAL, BG, and SEA scenarios, while the NB classifier only achieves this in the BG scenario.

kNN with ISS was also able to achieve a 90%+ number of predictions within two of the true number of relevant fea-

Table 6: Table showing the feature count accuracy (%) for the kNN classifier using the standard ISS parameter configuration.

	Concept 1		Concept 2		Concept 3	
	Correct	Within 2	Correct	Within 2	Correct	Within 2
AGRAWAL	98.36	99.96	38.33	100.00	100.00	100.00
Assets	0.02	46.95	3.65	100.00	0.00	100.00
BG	98.83	99.25	98.09	100.00	100.00	100.00
BG 2	1.12	99.20	0.14	99.99	17.15	100.00
BG 3	15.74	97.41	0.00	97.88	2.55	100.00
Conditional	0.02	0.16	0.00	1.16	0.00	0.00
FY A	0.00	0.00	-	-	-	-
FY B	0.05	1.15	-	-	-	-
FY C	0.04	0.10	0.00	4.82	-	-
FY D	0.00	0.00	0.11	25.24	4.76	9.49
FY A Drift	0.00	0.00	0.38	2.44	-	-
FY B Drift	0.07	0.94	0.07	0.49	-	-
LED	43.55	98.44	48.33	99.23	-	-
SEA	97.94	97.95	99.55	99.96	93.77	99.98

Table 7: Table showing the feature count accuracy (%) for the Naïve Bayes classifier using the standard ISS parameter configuration.

	Concept 1		Concept 2		Concept 3	
	Correct	Within 2	Correct	Within 2	Correct	Within 2
AGRAWAL	3.00	36.46	0.00	99.55	50.46	100.00
Assets	10.99	100.00	0.82	100.00	18.07	90.72
BG	99.85	100.00	14.02	100.00	0.00	100.00
BG 2	1.47	1.47	0.00	85.55	0.00	99.48
BG 3	0.00	99.33	0.00	100.00	45.36	69.21
Conditional	0.00	0.00	0.00	0.30	0.00	0.00
FY A	0.40	12.41	-	-	-	-
FY B	0.89	16.23	-	-	-	-
FY C	0.87	11.63	0.00	2.88	-	-
FY D	3.85	17.08	10.77	36.55	0.81	3.30
FY A Drift	0.11	12.37	11.08	38.68	-	-
FY B Drift	1.76	9.62	0.27	3.34	-	-
LED	6.96	32.52	39.01	99.46	-	-
SEA	21.75	22.27	0.00	33.38	0.00	2.86

tures for 19/20 of the concepts not generated by the Conditional generator, while NB did the same for only 13/20 of the concepts not generated by the Conditional generator.

For stream datasets generated by the Conditional generator (Conditional, FY A, FY A Drift, FY B, FY B Drift, FY C, and FY D) for which the concepts were more complex, ISS struggled to produce a subset size close to the true number of relevant features for both the kNN and NB classifiers, with the classifier making less than 50% of predictions using a subset size within two to the true number of relevant features for either classifier across all concepts. An interesting observation to make is the generally larger performance improvements observed for the kNN classifier when utilizing ISS for these subsets compared to those not generated by the Conditional generator. Despite ISS being unable to perfectly identify the true number of relevant features, the algorithm is still selecting useful subsets able to boost classification accuracy and to reduce resource consumption.

7.4 Accuracy difference scaling ranking

Experiments with accuracy difference scaling enabled for ranking (ADSR) were conducted to explore its effect on classification accuracy. Parameter configurations differed in confidence thresholds t (from 0 to 0.2), scalar values c (1 to 10), and whether or not to only penalize the ranking score o. Experiment used the same configuration as those mentioned in Section 7. We found that accuracy difference scaling had marginal impact on the overall classification accuracy and in some cases gave worse accuracy results. A specific parameter configuration using a confidence threshhold t of 0.05, a scaling weight c of 4, and only penalization was selected as the default configuration. This configuration was selected based on its performance across all data sets tested.

Table 8: Table comparing the classification accuracy of kNN and NB with ISS using accuracy difference ranking compared to without. Results have been rounded to 2DP

	kNN-ISS	kNN-ISS AGR	kNN Diff	NB-ISS	NB-ISS AGR	NB Diff
AGRAWAL	88.97	89.18	0.21	81.78	81.78	0.00
Assets	85.53	85.63	0.10	84.85	84.73	-0.12
BG	89.79	89.87	0.08	88.05	88.48	0.43
BG 2	89.76	89.86	0.10	75.89	75.65	-0.24
BG 3	89.59	89.66	0.07	68.06	67.79	-0.29
Conditional	91.86	92.07	0.21	89.13	89.10	-0.03
FY A	72.87	73.12	0.25	87.13	87.13	0.00
FY B	85.33	86.29	0.96	91.84	91.85	0.01
FY C	44.98	44.98	0.00	53.01	53.00	-0.01
FY D	67.90	66.88	-1.02	77.92	77.93	0.01
FY A Drift	81.27	81.57	0.30	84.47	84.47	0.00
FY B Drift	80.53	86.29	5.76	84.62	91.85	7.23
LED	73.62	73.72	0.10	72.79	72.82	0.03
SEA	88.29	88.35	0.06	83.90	83.90	0.00

Table 8 shows the classification accuracy results for kNN and NB with ISS in the default parameter configuration using ADSR compared to without ADSR. The fourth and seventh columns show the difference between ADSR enabled and disabled, with red indicating worse results and blue indicating better results for ADSR. Overall, we notice that utilizing ADSR in the ranking function does not improve accuracy in a significant way in most cases tested, with only one stream (FY B Drift had improvement over 1% in overall accuracy. The kNN classifier tended to perform slightly better with ADSR, however ADSR often tends to produce indifferent or slightly worse results when used with the NB classifier.

Tables 9 and 10 show the feature selection accuracy results for kNN and NB with ISS and ADSR enabled. The columns are split into each stream concept in the first row. The second row splits the results into the percentage that the algorithm correctly predict the true number of features, and predictions within two to the true number of relevant features for each stream concept. Difference between ADSR and without ADSR are highlighted, with red indicating worse results for ADSR and blue indicating better. As with the classification accuracy results, ADSR's impact on feature selection accuracy were overall mixed. For the kNN classifier, ADSR tended to produce worse or slightly worse results while producing mostly similar results for the NB classifier.

As the results for both the classification accuracy and feature selection accuracy were mixed and any improvements are not significant, we did not utilize it further in our experiments.

7.5 ISS against DFW and HAT

We compared ISS against kNN/NB with DFW [7], an algorithm which attempts to address feature drifts via feature weighing, and HAT, a classifier which has been shown to give good performance in feature drifting streams [6]. Comparison results between ISS, DFW, and HAT are shown in Tables 11, 12, and 13.

Overall, it was observed that HAT produced the highest

Table 9: Table showing the feature count accuracy (%) for the kNN classifier using the standard ISS parameter configuration with ADSR.

	Concept 1		Concept 2		Concept 3	
	Correct	Within 2	Correct	Within 2	Correct	Within 2
AGRAWAL	97.97 -0.38	99.96	64.93 +26.6	100.00	100.00	100.00
Assets	0.02	46.95	3.65	100.00	0.00	99.92
BG	98.83	99.25	98.09	100.00	100.00	100.00
BG 2	0.00 -1.12	98.75 -0.45	0.14	100.00 +0.01	17.41 +0.26	99.99 -0.01
BG 3	15.63 -0.11	97.41	0.00	99.97 +2.09	2.55	100.00
Conditional	0.01 -0.01	0.04 -0.12	0.00	0.03 -1.13	0.00	0.00
FY A	0.00	0.00	-	-	-	-
FY B	0.00 -0.05	2.79 +1.64	-	-	-	-
FY C	0.04	0.10	0.00	4.82	99.99	-
FY D	0.01 +0.01	0.01 +0.01	0.01 -0.1	8.58 -16.66	0.01 -4.75	2.89 -6.6
FY A Drift	0.00	0.00	2.49 +2.11	2.95 +0.51	17.97	-
FY B Drift	0.01 -0.06	3.83 +2.89	0.30 +0.23	0.93 +0.44	-	-
LED	36.21 -7.34	96.99	53.04 +4.71	97.4 -1.83	-	-
SEA	98.18 +0.24	98.19 +0.24	99.48	99.94 -0.02	93.77	99.98

Table 10: Table showing the feature count accuracy (%) for the Naïve Bayes classifier using the standard ISS parameter configuration with ADSR.

	Concept 1		Concept 2		Concept 3	
	Correct	Within 2	Correct	Within 2	Correct	Within 2
AGRAWAL	0.00 -3.00	36.46	0.00	99.55	51.14 +0.68	100.00
Assets	10.99	100.00	0.82	100.00	12.68 -5.39	81.36 -9.36
BG	99.85	100.00	14.77 +0.75	100.00	100.00 +100	100.00
BG 2	1.47	1.47	0.00	85.55	0.00	99.48
BG 3	0.00	99.33	0.00	100.00	45.36	69.21
Conditional	0.00	0.00	0.00	0.30	0.00	0.00
FY A	0.40	12.41	-	-	-	-
FY B	0.89	16.23	-	-	-	-
FY C	0.87	11.63	0.00	2.88	-	-
FY D	3.85	17.08	10.77	36.55	0.81	3.30
FY A Drift	0.11	12.37	11.09 +0.01	38.69 +0.01	-	-
FY B Drift	1.76	9.62	0.27	0.62 -2.72	-	-
LED	6.96	32.52	39.56 +0.55	99.46	-	-
SEA	21.75	22.27	0.00	33.36	0.00	2.86

mean accuracy for most experiments, and was generally competitive with NB-ISS/kNN-ISS for the stream scenarios where it was not the highest. DFW was observed to be competitive with ISS for the kNN classifier for mean accuracy; however, a much more pronounced difference in mean accuracy was observed for NB where ISS achieved better results. The results obtained by the statistical tests are shown in Figure 3: they corroborate that HAT is still the best performing classifier, yet, the results obtained are not statistically superior when compared to both NB-ISS and KNN-ISS.

Computation time (Table 12) and RAM-hours (Table 13) is where ISS excelled, with NB-ISS being the fastest classifier for all of the streams tested by a large margin. Both the ISS versions of kNN and NB having much better computation time and RAM-hours results than their DFW counterparts for all streams. kNN specifically had significantly better computation time and RAM-hours performance for ISS than DFW. NB-ISS generally used half as much CPU time compared to the HAT, and achieved better RAM-hours results for all but the BG2, BG3, and LED scenarios. In Figures 4 and 5 we report the results of the statistical tests performed

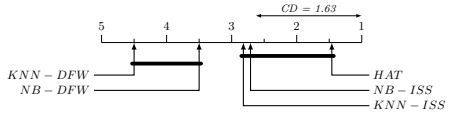


Figure 3: Critical difference chart for accuracy rates.

Table 11: Accuracy rates (%) obtained by HAT, ISS using the SU ranking function and DFW for both NB and kNN classifiers. Best result highlighted.

Stream	NB-ISS	NB-DFW	kNN-ISS	kNN-DFW	HAT
AGRAWAL	81.78	68.97	88.97	88.99	**89.46**
ASSETS	84.85	74.00	85.53	91.56	**94.13**
BG	88.55	83.94	89.79	89.78	**89.86**
BG 2	75.89	75.67	89.76	89.50	**89.77**
BG 3	68.06	56.11	**89.59**	89.44	**89.59**
CONDITIONAL	89.10	78.12	91.86	92.56	**96.17**
FY A	**87.13**	62.98	72.87	66.99	86.56
FY B	**91.84**	72.88	85.33	80.69	91.81
FY C	**53.01**	46.70	44.98	27.98	46.01
FY D	**77.92**	68.06	67.90	50.43	75.82
FY A Drift	84.47	68.58	81.27	77.92	**89.14**
FY B Drift	84.62	73.74	80.53	73.77	**89.18**
LED	72.79	72.28	73.62	71.76	**73.69**
SEA	83.90	73.61	88.29	84.80	**88.87**

Table 12: CPU Time (s) obtained by HAT, ISS using the SU ranking function and DFW for both NB and kNN classifiers. Best result highlighted.

Stream	NB-ISS	NB-DFW	kNN-ISS	kNN-DFW	HAT
AGRAWAL	**3.00**	14.48	353.31	1077.56	7.42
ASSETS	**17.66**	33.63	304.95	500.09	35.10
BG	**2.32**	8.81	456.48	450.95	6.14
BG 2	**3.64**	16.60	517.89	691.29	7.11
BG 3	**3.55**	20.07	565.14	778.33	7.88
CONDITIONAL	**8.43**	106.61	735.15	2980.35	14.32
FY A	**12.08**	262.93	770.34	4064.95	40.81
FY B	**41.75**	2284.51	1621.89	14990.08	105.56
FY C	**58.21**	1719.44	745.03	12047.13	233.72
FY D	**66.83**	2816.18	1384.43	19907.43	281.96
FY A Drift	**12.07**	305.55	1000.18	4262.95	38.05
FY B Drift	**41.60**	2317.48	1576.40	15277.31	117.59
LED	**4.18**	59.67	824.93	931.52	14.81
SEA	**5.50**	30.54	426.26	1813.26	11.43

Table 13: RAM-Hours (kb-hour) obtained by HAT, ISS using the SU ranking function and DFW for both NB and kNN classifiers. Best result highlighted.

Stream	NB-ISS	NB-DFW	kNN-ISS	kNN-DFW	HAT
AGRAWAL	**0.54**	2.65	63.58	49454.93	1.28
ASSETS	**2.39**	4.67	41.13	190.83	6.61
BG	**0.44**	1.70	86.43	222.97	0.47
BG 2	1.08	4.96	152.55	626.55	**0.69**
BG 3	1.05	5.99	166.47	547.08	**0.82**
CONDITIONAL	**3.93**	49.64	341.12	352504.03	7.69
FY A	**7.03**	151.83	443.87	625934.49	54.21
FY B	**70.57**	3811.55	2713.85	539595.71	261.76
FY C	**92.59**	2659.90	1149.98	1980181.04	5495.08
FY D	**117.44**	4803.79	2356.87	1327763.88	3916.08
FY A Drift	**7.01**	176.45	580.64	657448.93	38.27
FY B Drift	**70.25**	3866.58	2644.58	562537.78	386.37
LED	0.97	23.99	326.19	1056.71	**0.93**
SEA	**2.19**	7.12	98.32	165520.00	2.35

for CPU time and RAM-Hours measurements. We see that indeed NB-ISS is the fastest algorithm, followed by the HAT and NB-DFW, and that these three are statistically faster when compared to kNN methods. Regarding RAM-hours, NB-ISS is again highlighted as the most light-weighted approach, followed by HAT.

8. FUTURE WORKS

The proposed algorithm is somewhat specialized for only kNN and NB, and thus, a more generalized algorithm is

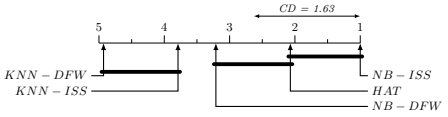


Figure 4: Critical difference chart for CPU time rates.

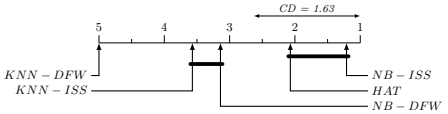


Figure 5: Critical difference chart for RAM-Hours rates.

something worth looking into. However, such a generalization of the algorithm is something which is not without difficulty, as the algorithm relies on the cumulative nature of the kNN and NB classifiers to keep computational costs at a reasonable rate. Consequently, any other classifier to be used with ISS will most likely also need to be able to take advantage of information from the $(n-1)$-th subset for the computation of the (n)-th subset.

Another part of the algorithm which should be investigated more thoroughly, is the heuristic used to determine the subset size for kNN. In this work, hill climbing was selected for its simplicity and more importantly its speed, yet, other heuristics may have better performance. One such direction would be to explore utilization of late acceptance hill climbing [11], a simple meta-heuristic which delays the hill climbing comparison, causing new solutions to be compared to a past solution several iterations ago rather than the immediately previous solution.

Finally, applications for the algorithm outside of classification problems, such as regression, should also be explored in future work.

9. CONCLUSION

In this paper, we introduced a novel embedded feature selection method aimed at addressing feature drift in fast-flowing data streams. Our method divided feature selection into two stages, by first ranking features and creating a subset, then iteratively evaluating and removing the lowest ranked feature from the subset until the subset is empty. Experimental results indicate that our method is successful at improving mean classification accuracy in feature drifting streams for both k-Nearest Neighbours and Naïve Bayes classifiers. In addition to the accuracy improvements, noticeable gains in runtime and RAM hours were achieved for the kNN classifier in scenarios with a large number of irrelevant features.

10. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993. Special issue on Learning and Discovery in Knowledge-Based Databases.

[2] G. H. Albert Bifet, Eibe Frank and B. Pfahringer, editors. *Accurate Ensembles for Data Streams: Combining Restricted Hoeffding Trees using Stacking*, volume 13 of *JMLR Proceedings*. JMLR.org, 2010.

[3] E. Almeida, C. A. Ferreira, and J. Gama. Adaptive model rules from data streams. In *ECML/PKDD (1)*, volume 8188 of *Lecture Notes in Computer Science*, pages 480–492. Springer, 2013.

[4] J. P. Barddal, F. Enembreck, H. M. Gomes, A. Bifet, and B. Pfahringer. Merit-guided dynamic feature selection filter for data streams. *Expert Systems with Applications*, 116:227 – 242, 2019.

[5] J. P. Barddal, H. M. Gomes, and F. Enembreck. A survey on feature drift adaptation. In *Proceedings of the International Conference on Tools with Artificial Intelligence*. IEEE, November 2015.

[6] J. P. Barddal, H. M. Gomes, F. Enembreck, and B. Pfahringer. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, 127(Supplement C):278–294, 2017.

[7] J. P. Barddal, H. M. Gomes, F. Enembreck, B. Pfahringer, and A. Bifet. On dynamic feature weighting for feature drifting data streams. In *ECML/PKDD'16*, Lecture Notes in Computer Science. Springer, 2016.

[8] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*, 2007.

[9] A. Bifet and R. Gavaldà. *Adaptive Learning from Evolving Data Streams*, pages 249–260. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[10] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, Aug. 2010.

[11] E. K. Burke and Y. Bykov. A late acceptance strategy in hillclimbing for exam timetabling problems. 2008.

[12] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, Dec. 2006.

[13] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[14] J. Duarte and J. Gama. Feature ranking in hoeffding algorithms for regression. In *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*, pages 836–841, 2017.

[15] H. Fu, Z. Xiao, E. Dellandréa, W. Dou, and L. Chen. *Image Categorization Using ESFS: A New Embedded Feature Selection Method Based on SFS*, pages 288–299. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[16] J. Gama and C. Pinto. Discretization from data streams: Applications to histograms and data mining. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 662–667, New York, NY, USA, 2006. ACM.

[17] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, Mar. 2014.

[18] I. Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[19] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[20] A. Metwally, D. Agrawal, and A. El Abbadi. *Efficient Computation of Frequent and Top-k Elements in Data Streams*, pages 398–412. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[21] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, and L. Wan. Heterogeneous ensemble for feature drifts in data streams. In *Advances in Knowledge Discovery and Data Mining*, volume 7302 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2012.

[22] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

[23] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, 23(1):69–101, Apr. 1996.

[24] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[25] Z. Xiao, E. Dellandrea, W. Dou, and L. Chen. ESFS: A new embedded feature selection method based on SFS. Technical Report RR-LIRIS-2008-018, LIRIS UMR 5205 CNRS/INSA de Lyon/Universite Claude Bernard Lyon 1/Universite Lumiere Lyon 2/Ecole Centrale de Lyon, Sept. 2008.

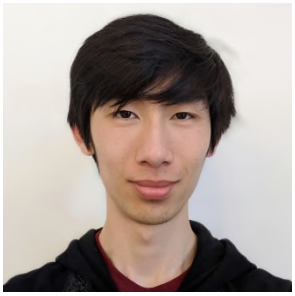
[26] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 856–863, 2003.

[27] L. Yuan, B. Pfahringer, and J. P. Barddal. Iterative subset selection for feature drifting data streams. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 510–517, New York, NY, USA, 2018. ACM.

[28] Z. Zhao and H. Liu. Searching for interacting features. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1156–1161, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[29] Z. Zhao, F. Morstatter, S. Sharma, S. Alelyani, A. Anand, and H. Liu. Advancing feature selection research. *ASU feature selection repository*, pages 1–28, 2010.

ABOUT THE AUTHORS:



Lanqin Yuan received his Bachelor of Computer and Mathematical Science with First class Honours in 2017 from the University of Waikato, New Zealand. He is currently working as a software engineer. His areas of interests are Machine learning, Reinforcement learning, Data mining, and Data streams.

Bernhard Pfahringer received his PhD degree from the University of Technology in Vienna, Austria, in 1995. He is a Professor with the Department of Computer Science at the University of Waikato. His interests span a range of data mining and machine learning sub-fields, with a focus on streaming, randomization, and complex data.

Jean Paul Barddal received his B.Sc. at the Pontifical Catholic University of Paraná in 2013, where he was awarded with the best student prize. He also received his M.Sc. and Ph.D. degrees from the same institution in 2015 and 2018, respectively. He has experience in Computer Science with emphasis on Data Science, Machine Learning, and Knowledge Discovery in both theoretical and practical terms. His works mainly focus on classification, regression, clustering, and feature selection on streaming scenarios.