Online Feature Screening for Data Streams With Concept Drift

Mingyuan Wang and Adrian Barbu, Senior Member, IEEE

Abstract—Screening feature selection methods are often used as a preprocessing step for reducing the number of variables before training a model. Traditional screening methods only focus on dealing with complete high dimensional datasets. However, modern datasets not only have higher dimensions and larger sample size, but also have properties such as streaming input, sparsity, and concept drift. Therefore a considerable number of online feature selection methods were introduced to handle these kinds of problems in recent years. Online screening methods are one of the categories of online feature selection methods. The methods that we propose in this paper are capable of handling all three situations mentioned above, in classification settings. Our experiments show that the proposed methods can generate the same feature importance as their offline versions with faster speed and less storage requirements. Furthermore, the results show that online screening methods with integrated model adaptation have a higher true feature detection rate than without model adaptation on data streams exhibiting concept drift. Among the three large real datasets that potentially have concept drift, online screening methods with model adaptation show advantages in either saving computation time and space, reducing model complexity, or improving prediction accuracy.

Index Terms—Concept drift, data stream mining, feature screening, feature selection, model adaptation

1 INTRODUCTION

With the explosive growth in the amount of data available in recent years, the efficiency of data processing remains a heated topic in both hardware and software fields. Various approaches have emerged to tackle this challenge, such as cloud computing, batch learning, online learning, more powerful hardware and so on. These approaches mainly focus on how to increase the loading capacity of the systems. On the other hand, the screening (filter) feature selection methods, which show impressive performance in reducing feature dimensions of large high dimensional data and improving overall model performance, especially with ad-hoc implementations, focus on how to scale down the data before considering the loading capacity of the learning system.

Screening feature selection methods are independent of the learning algorithms, which gives them a fast execution speed. They can be ideal add-ons when it comes to improving efficiency without losing performance. Building on the foundation of our previous survey on the performance of screening feature selection methods on real large datasets, in this article we introduce several novel online feature screening methods as extensions of existing offline screening methods to tackle datasets that come from a more modern environment.

In this article, five screening methods are selected from our previous survey [1] to be developed into their online versions. Among them, T-score [2] and Fisher Score [3] are mean-variance based methods, while Gini index [4], Chi-square score [5], and Mutual Information [6] are bin-count based methods.

The newly introduced methods are capable of working in one pass fashion using minibatches on extremely large data streams. They are compatible with sparse input and are robust when concept drift is present in the data stream. For mean-variance based methods, we used moving averages and altered the update frequency from at each input to only at non-zero inputs for sparse data. A fading factor was applied to each update to deal with concept drift issues. However, our main focus was on bin-count based methods where we adapted the underlying quantile summary mechanism from [7] to deal with streaming data, with sparsity, and with concept drift using a fading factor. To make it compatible with the discretization method we adapted and to our fading factor update mechanism under sparse inputs, we introduced an exact weight update procedure to replace the original one. We also introduced delayed-weight-update to cope with the sparse input data. Moreover, when applying the fading factor for model adaptation, in order to maintain the correct penalization order without compromising computing speed, we introduced a new updating order for the fading factor. In the case where the data stream has both sparsity and concept drift, we proposed the concept of time anchors and universal weight maps to further reduce the computing time of our algorithm while maintaining the weight-penalization accuracy.

Comparative evaluations were conducted between the proposed methods and their offline versions. The proposed
methods with model adaptation were also tested using synthetic data streams with concept drift. Finally, the performance of our methods was evaluated on large real datasets, with and without model adaptation. The methods were also compared with and usually outperformed some of the state-of-the-art feature screening methods that were built for concept drifting data. These real datasets that were used all have sparsity and potentially concept drift.

Two types of variables could exist in a data stream, based on the distribution they originate from: variables that are sampled from a continuous distribution and variables that come from a discrete distribution. For instance, in [8], the author used “continuous attributes” to refer to variables from continuous distributions. For simplicity, we will call these types of variables (or features) continuous features and discrete features respectively.

This study addresses the problem of online learning with continuous features with two approaches: mean-variance based feature screening methods based on moving averages, and quantile based methods based on the weighted quantile summary. The proposed algorithm also integrates ways to handle sparse streaming data as well as streaming data featuring concept drift, as adaptations to the needs of modern applications.

The contributions of this paper are the following:

- It shows how to adapt the mean-variance and bin-count based screening methods such as Mutual Information, Chi-squared score and Gini index for online screening. For that, it modifies the work from [7] to approximate continuous distributions with online histograms using quantile summaries. Theoretical guarantees of accuracy are also proved. The mean-variance based methods can be easily computed online using running averages, and are also included for completeness.
- It shows how to modify these online screening methods for handling sparse data and concept drift.
- Through extensive experiments, it shows that the proposed online screening methods compare favorably with their offline counterparts, in the sense that they can obtain the same feature ranking with less computational cost, while being able to handle streaming data. Simulations show that the proposed online methods are capable to handle concept drift with different drift speeds much better than the ones without model adaptation. Furthermore, experiments on three large real datasets show that the proposed methods usually outperform two state-of-the-art online screening methods from the literature.

1.1 Related Work

The study of feature selection on streaming data has a long history. Even before the big data boom, there already existed applications that generated and required processing streaming data under computation and memory constraints.

Feature selection methods can be divided into three categories based on their interaction with the learning algorithm: filters, wrappers and hybrids. Filters, also known as screening methods, compute a criterion for each feature and select features according to some selection scheme. They don’t involve a learner. Wrappers wrap feature selection with any learner. They use a feature importance measure calculated by the learner to decide what features to keep. Hybrids, also known as embedded methods, rely on a sparsity inducing built-in regularizer of the learner to drop irrelevant features. One such example is the Lasso[9].

Screening methods have the advantage of fast processing and easy integration with other systems. The methods introduced in this article also falls into this category. Considering the large number of streaming high dimensional applications, we feel its advantages make it an appropriate direction to pursue.

One property of large stream datasets is data sparsity. A sparse dataset is a dataset with a large proportion of its entries zero. Such a phenomenon appears in various domains, such as text processing, social network analysis, disease analysis, etc. In [10] was proposed an adaptive multimodal multi-task learning model that can co-regularize the modality agreement, temporal progression and discriminative capabilities of different modalities for chronic disease progression monitoring, which can be applied to sparse data. A comprehensive overview of multi-task learning, feature selection and handling of sparse data was shown in [11] through the theoretical explanation and real life applications to multiple social network learning problems.

Another property of continuously generated data stream is concept drift. The fundamental idea is that the relation between the prediction target and its features usually changes over time either gradually or drastically. Concept drift is also called model adaptation, feature drift, concept adaptation, depending on the focus of the dynamic connection between target and features.

Feature selection methods are expected to be robust in face of concept drift. One of the popular methods is the sliding window. Its idea is to only consider the most recent data, making it a natural adaptation to the drift problem. Using a month as a natural window, [12] introduced a two stage feature selection framework to first select feature candidates using symmetrical uncertainty, then a greedy stepwise search was performed using a generic learner to further refine the feature space. Although the search strategy involved a stopping criterion, it can still become time consuming when feature space is large. Also the symmetrical uncertainty used can only directly process discrete features. [13] also utilizes a fixed size moving window that stores the most recent data. It filters features based on the comparison of symmetrical uncertainty between a set of baseline features and non-baseline features dynamically. Again, the symmetrical uncertainty used can only directly process discrete features. This method requires a $p \times p$ conditional entropy matrix to be maintained constantly, where $p$ is the number of features. It will become a burden if the number of features grows ultra large. In [14], the authors used the cost-sensitive ReliefF method with a sliding window for dynamic feature selection.

Methods utilizing sliding windows rely heavily on the assumption that the decided sliding window fashion matches the actual shifting pattern of the data stream. One way to mitigate this disadvantage is to detect a valid drifting window first before performing feature selection. A feature selection framework for drifting data streams that
adopted this exact scheme was introduced in [15]. It first performed drift detection to identify the width of a drift window. Once a drifting point was detected, a feature selection procedure was carried on based on a relevancy-redundancy criteria calculated from an incrementally updated correlation coefficient matrix. [16] uses a generic drifting detector to tell which decision stumps in a stream-adaptive boosting were no longer compatible with the new data. The features were selected based on the decision stumps’ purity metric after updating the incompatible stumps.

Another popular strategy to handle the drift problem is the fading factor. In [17], both sliding window and fading factor are used. In the fading factor variation, feature measures such as the Gini index and entropy are calculated incrementally over the entire data stream. However the criterion they used also works with discrete features only.

There are methods handling the drifting feature selection problem by adding a follow-up right after the feature selection stage. The authors in [18] introduced a feature fusion method to suppress the negative effect caused by concept drifting after the feature selection step.

One genre of feature selection on drifting data is to maintain a dataset throughout the processing of the data stream. One such method is [19], where the algorithm maintains a set of micro-clusters based on the arriving new instances. Once a drift is detected, all features that relate to the drift will have their mutual information updated using data in all the clusters. Eventually feature selection was conducted based on mutual information. The computing time of this method goes up if the newly arrived instances keep failing to meet the criteria for removing outdated micro-clusters. In [20], the authors tackled the drift issue by maintaining a fixed size pool of the newest set of training data and refined the feature size such that the data in the pool is always compatible with the trend of the newly arrived data. However the method didn’t cover the issue where an irrelevant feature later becomes relevant again, a case that can be handled by our approach.

As an interesting new approach that is different from the others, [21] assumed the coefficients of a linear model to be a set of distributions. The algorithm then incrementally updates the parameters of these distributions. The feature importances were computed based on these distributions on request. Another recent interesting method, [22], mixed a set of features selected by Information Gain with several sets of random features. Feature sets with above average performance were passed to a genetic algorithm to cross-breed the final selected feature set.

Despite various situations in which the aforementioned methods can be applied, to the best our knowledge, none of the existing methods were capable of using metrics that was designed for discrete features (such as mutual information) for screening continuous features in a one pass fashion. In order to calculate an appearance based criterion such as mutual information and chi-squared score from continuous features, one needs to be able to discretize the continuous features in an online fashion. Much effort has been put into this area, since good discretizations are based on quantiles. It has been proven in [23] that \( O(N) \) space is required for an algorithm to compute the exact quantiles in a single pass of streaming data. In order to describe a one pass data stream in a reduced space, approximate quantile calculation methods are needed. Some earlier works are [24] and [25]. Their algorithms are used to calculate uniform quantiles in a single pass. In [8] a two-stage framework was proposed to obtain discrete binned features from continuous features. The suggested method first constructs many equal width intervals to capture and update the partition of incoming sample points. At query time, it aggregates the collected intervals to generate equal-width histograms or equal-frequency histograms. As intuitive as it is, this is a non-deterministic method, in the sense that there are no deterministic guarantees on the estimation error.

Manku et al., [26] introduced a single pass algorithm to compute a deterministic \( \epsilon \)-approximate uniform quantile summary. It requires prior knowledge of the sample size \( N \) and has a space complexity of \( O(\log^2 \epsilon N) \). Another algorithm that does not require prior knowledge of \( N \) was also proposed by Manku et al., in [27]. The space complexity for this algorithm is \( O(\sqrt{\log \frac{\sqrt{N}}{\epsilon}} + \log^3 \frac{1}{\epsilon}) \) with a failure probability of \( \delta \). A more recent approach (the GK algorithm) was introduced by Greenwald et al., in [28] to compute a deterministic \( \epsilon \)-approximate quantile summary on a single pass of streaming data without the prior knowledge of \( N \). This method imposed a tree structure and it is an improvement of Manku’s algorithm, with a space bound of \( O(\log \epsilon N) \). In [29], an improvement was made on the GK algorithm to significantly reduce the computational cost. The computational cost of this multi-level quantile summary algorithm is \( O(N \log \frac{\sqrt{\epsilon}}{\epsilon}) \). It is shown in their experiments that it can achieve about 200-300 times speedup over the GK algorithm. However, its storage requirement of \( O(\log^2 \epsilon N) \) is higher than the GK algorithm.

Recent work in Xgboost[7] extended [28] and [29] with a focus on processing weighted data. This saves space and improves summary accuracy when dealing with data streams containing duplicate values. The algorithm introduced in their article has the same guarantee as the GK algorithm. It can be plugged into all GK frameworks and its extensions.

There also exist works that focus on other aspects of calculating quantile summaries. In [30] was introduced an algorithm to compute uniform quantiles over sliding windows. Cormode et al., [31] proposed an algorithm to handle the biased quantile problem. It has a storage bound of \( O(\log^U \log \epsilon N) \) and time complexity of \( O(\log^U \log \epsilon) \) where \( N \) is the sample stream size and \( U \) is the size of the domain from which the sample points are drawn. Efforts were also made to compute approximate quantiles for distributed streams and sensor networks. Greenwald et al., [32] proposed an algorithm for calculating \( \epsilon \)-approximate quantiles for sensor network applications. In [33] an algorithm with a space complexity of \( O(U \log \epsilon) \) was proposed to compute medians and other quantiles in sensor networks.

## 2 Proposed Methods

Two categories of online feature screening methods are introduced in this section: mean-variance based methods and bin count based methods. For mean-variance based methods, we will give an overview and introduce the modifications that are integrated to tackle data streams with
sparsity and concept drift. For bin count based methods, we will briefly go over the concept of quantile summary, which is the foundation of our proposed methods. Our modification that helps provide exact observation count will also be introduced during the overview. Lastly, we will introduce extensions for the bin count based methods to handle data streams with sparsity and concept drift.

2.1 Mean-Variance Based Methods

The mean-variance based methods obtain the scoring from mean and variance estimates of the feature values corresponding to each class. The means and variances are known to be able to computed online using moving averages as shown below.

2.1.1 Running Averages

Without losing generality, we assume that a sample \( x_i \) arrives at each time step \( i = 1, 2, 3, \ldots, n \). The \( j \)-th feature value at the \( i \)-th time is \( x_{ij} \). At time \( n \), the average value \( \mu_{nj} \) of the \( j \)-th feature is:

\[
\mu_{nj} = \frac{1}{n} \sum_{i=1}^{n} x_{ij}. \tag{1}
\]

It can be updated incrementally as follows:

\[
\mu_{nj} = \frac{n-1}{n} \mu_{(n-1)j} + \frac{1}{n} x_{nj}. \tag{2}
\]

For the variance, we will use the running mean of squares \( MS_{nj} = \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 \), which can also updated incrementally:

\[
MS_{nj} = \frac{n-1}{n} MS_{(n-1)j} + \frac{1}{n} x_{nj}^2. \tag{3}
\]

Consequently the variance of \( j \)-th feature at time \( n \) can be computed as:

\[
\sigma_{nj}^2 = MS_{nj} - \mu_{nj}^2 \tag{4}
\]

Thus each feature’s mean and variance can be computed exactly and updated incrementally as new samples arrive.

2.1.2 Criteria of Interest

It is well known that T-score and Fisher Score are calculated from mean, variance and sample count.

The running means can be used to compute for a feature \( x \) and each class \( c \in \{1, \ldots, C\} \) the mean \( \mu_{cj} \), sample count \( n_{cj} \), and standard deviation \( \sigma_{cj} \) of the values of all observations belonging to class \( c \).

**T-Score.** For a feature \( x_j \), its T-score is calculated as:

\[
T_j = \frac{|\mu_{j(1)} - \mu_{j(2)}|}{\sqrt{\frac{\sigma_{j(1)}^2}{n_{j(1)}} + \frac{\sigma_{j(2)}^2}{n_{j(2)}}}}. \tag{5}
\]

The higher the score, the more relevant the feature is to the target variable.

**Fisher Score.** Similarly, the Fisher Score of feature \( x_j \) is defined as:

\[
Fisher_j = \frac{\sum_{c=1}^{C} n_{cj}^{(c)} (\mu_{cj}^{(c)} - \mu_j)^2}{\sum_{c=1}^{C} n_{cj}^{(c)} (\sigma_{cj}^{(c)})^2} \tag{6}
\]

where \( \mu \) is feature \( x_j \)’s mean and \( \mu_{cj}^{(c)} \) and \( \sigma_{cj}^{(c)} \) have been defined above. It is clear that (5) and (6) are calculated from basic components such as means and variances, which can be updated incrementally as described above.

2.1.3 Sparse Input

In the sparse input scenario, since the zero values are not shown in the data stream, one only needs to update the running averages with the values that are shown, and keep a record of the sample count.

2.1.4 Model Adaptation

In order to handle concept drift, a fading factor strategy is used to gradually decrease the weight of past samples in the running averages:

\[
\mu_{nj} = \alpha \mu_{(n-1)j} + x_{nj} \]

\[
MS_{nj} = \alpha MS_{(n-1)j} + x_{nj}^2 \tag{7}
\]

where \( \alpha \in (0, 1) \) is a fading factor that is set by the user. This way the last sample has weight 1, previous one has weight \( \alpha \), and the sample at time \( n-t \) has weight \( \alpha^{t} \) at time \( n \).

**Sparse Input.** When adaptation is required with sparse input, a time anchor is employed for each feature in every class to record the last appearance of non-zero values \( n_{last} \). Accumulated statistics are then updated the next time a non-zero value appears:

\[
\mu_{nj} = \alpha^{(n-n_{last})} \mu_{last,j} + x_{nj} \]

\[
MS_{nj} = \alpha^{(n-n_{last})} MS_{last,j} + x_{nj}^2 \]

where \( \mu_{last,j} \) and \( MS_{last,j} \) is the penalized running average and running mean of squares at the last appearance of a non-zero value. Observe that \( \mu_{nj} \) and \( MS_{nj} \) can be computed at any time, keeping in mind that \( x_{nj} \) is zero if not present.

2.2 Bin Count Based Methods

These feature screening methods assume that each feature is discrete, taking a number \( B \) of values (bins). In this work we will focus on mutual information, chi-squared score and Gini index, which will be described below.

One approach to compute these criteria for continuous features is to approximate them using online histograms through a proper online discretization method. For that, online quantiles must be computed and online bin counts for discrete bins based on the online quantiles must be updated at the same time. In Section 2.2.2, we walk through the quantile summary used in [7]. We will introduce in Section 2.2.3 an alternate procedure with a maximum 3\( e \) error, which was modified in a way to be compatible with the discretization method we adapted in Section 2.2.4. Moreover the procedure was also made compatible with the handling of sparse input and fading factor updates that will be introduced in Sections 2.2.5 and 2.2.6 respectively. The introduced procedures will generate on demand bin counts from
data stream features whose values are sampled from continuous distributions.

### 2.2.1 Criteria of Interest

Mutual Information, Chi-squared score and Gini Index are all calculated directly or indirectly (through a probability estimation) from bin counts of the data belonging to the different classes.

**Mutual Information.** For a feature \( x_j \), its mutual information can be calculated as:

\[
I(x_j, y) = \sum_{b=1}^{B} \sum_{c=1}^{C} P(x_j \in bin_b, y = c) \log \frac{P(x_j \in bin_b, y = c)}{P(x_j \in bin_b)P(y = c)}
\]

where \( P(x_j \in bin_b, y = c) \) is the joint probability of having feature values fall into \( bin_b \) and label value equal to \( c \), while \( P(x_j \in bin_b) \) and \( P(y = c) \) are the marginal probabilities.

In the case of samples with discrete features, the probability can be expressed as:

\[
P(x_j \in bin_b, y = c) = \frac{n_{x_j \in bin_b, y = c}}{n},
\]

\[
P(x_j \in bin_b) = \frac{n_{x_j \in bin_b}}{n},
\]

\[
P(y = c) = \frac{n_{y=c}}{n},
\]

where \( n, n_{x_j \in bin_b, y = c}, n_{x_j \in bin_b}, n_{y=c} \) denote the sample counts that fall into the respective value groups.

**Chi-Squared Score.** With a similar definition of \( n \)’s, the chi-squared score of a feature \( x_j \) can be defined as:

\[
\chi^2_j = \sum_{b=1}^{B} \sum_{c=1}^{C} \left( \frac{n_{x_j \in bin_b, y = c} - \hat{n}_{x_j \in bin_b, y = c}}{\hat{n}_{x_j \in bin_b, y = c}} \right)^2,
\]

where:

\[
\hat{n}_{x_j \in bin_b, y = c} = \frac{n_{x_j \in bin_b}n_{y=c}}{n}
\]

**Gini Index.** For a given feature \( x_j \), let \( A_h = \{i, x_{ij} \leq h\} \) denote the number of samples whose values of the \( j \)-th feature is smaller than or equal to \( h \) and \( B_h = \{i, x_{ij} > h\} \). The feature \( j \)’s Gini Index can be expressed as:

\[
\text{Gini}_j = P(A_h)(1 - \sum_{c=1}^{C} P(C_c|A_h)^2) + P(B_h)(1 - \sum_{c=1}^{C} P(C_c|B_h)^2),
\]

where \( P(A_h) \) is the number of samples in subset \( A_h \) divided by the total number of samples, and \( P(C_c|A_h) \) is the conditional probability of samples having label \( c \) given that they are in subset \( A_h \). Let \( n_{x_j \in A_h, y = c} \) denote the number of samples in \( A_h \) with label \( c \) and let \( n_{x_j \in A_h} \) denote the number of samples in \( A_h \). Then \( P(C_c|A_h) = n_{x_j \in A_h, y = c} / n_{x_j \in A_h} \) remains the same for \( P(B_h) \) and \( P(C_c|B_h) \). The value of \( h \) for each feature is chosen to give the minimum Gini Index for that feature.

These criteria can be calculated on demand if the bin counts can be updated online in a streaming fashion. However, many features are continuous features and need to be discretized into a small number of bins for these criteria to be effective. Most effective discretizations are based on quantiles of the data, so the quantiles need to be computed online, at the same time with the discretization into the bins obtained from the quantiles, and with the bin counts based on the discretization. Quantile summaries will be used for this purpose.

### 2.2.2 Quantile Summary Overview

For a data stream with known size \( N \), the basic idea is to use several container like sub-summaries \( s \) to store approximate ranking information on parts of the data stream. In turn, an aggregated summary \( S \) of the sub-summaries can describe the entire data stream. Various operations are conducted continuously to maintain the estimations in these containers such that at any time \( n \), the summary \( S(n) \) can answer any \( r \)-quantile query with \( \epsilon \) precision.

Let \( D \) be a small input stream

\[
D = \{(x_1, w_1), (x_2, w_2), \ldots, (x_n, w_n)\},
\]

where \((x_i, w_i)\) is a data point with its value \( x_i \) and its weight \( w_i \). Usually the data weights are set to 1. For each value \( v \), two rank functions and a weight function are defined,

\[
\begin{align*}
& r^-(v) = \sum_{(x, w) \in D, x < v} w, \\
& r^+(v) = \sum_{(x, w) \in D, x \leq v} w, \\
& w(v) = r^+(v) - r^-(v) = \sum_{(x, w) \in D, x = v} w.
\end{align*}
\]

The weight of \( D \) is defined as:

\[
w(D) = \sum_{(x, w) \in D} w
\]

Without loss of generality, in the rest of this article, weight will be used to refer to the calculated weight value \( w(v) \), rank will be used to refer to the calculated rank value of \( r^+(v) \) or \( r^-(v) \).

A summary \( S \) or sub-summary \( s \) consists of several tuples \( S = \{T_1, T_2, \ldots, T_h\} \). Each tuple has the form \( T=(\bar{v}, \bar{r}^-, \bar{r}^+, \bar{w}) \) and describes a number of similar data points from the data stream. Here \( \bar{v} \) denotes the data value estimation this tuple represents, \( \bar{r}^- \) and \( \bar{r}^+ \) are the lowest and highest estimated ranks of this value in the current sub-summary, and \( \bar{w} \) is the accumulated weight value which represents how many data points this tuple covers. The sub-summary \( s \) entries \( T_1, T_2, \ldots, T_h \) are sorted by the values of their corresponding \( v \)’s.

A multi-level summary structure \( S = \{s_1, s_2, \ldots, s_L\} \) consists of several sub-summaries, shown in Fig. 1, when it is not queried. It is used to maintain the desired precision as well as to speed-up the calculation. \( L \) is the total number of levels, \( s_l \) denotes the sub-summary at level \( l \in \{0, 1, \ldots, L\} \).
The whole data stream is divided into consecutive segments of size \( b = \lceil \frac{L}{l} \rceil \), where \( L \) is the largest integer that makes \( b2^{l-1} \leq N \), and \( N \) is the size of the entire data stream.

At the lowest level, \( s_0 \) is defined to hold all recently arrived points until it reaches size \( b \). The tuples in \( s_0 \) are constructed with

\[
\begin{align*}
v &= x_i, \quad r^{-}(x_i) = r^{-}(x_i), \\
r^{+}(x_i) &= r^{+}(x_i), \quad \bar{w}(x_i) = w(x_i),
\end{align*}
\]

therefore \( s_0 \) is a \( \epsilon_0 \)-approximate summary, where \( \epsilon_0 = 0 \). It can answer all query questions exactly.

**Algorithm 1. General Procedure for Quantile Summary**

**Input:** \( x_i, i = 1, l \)
1: push \( x_i \) into \( s_0 \)
2: if \( \text{size}(s_0) < b \) then
3: go back to line 1
4: else
5: \( s_{\text{temp}} = \text{PRUNE}(s_0, \frac{\text{size}(s_0)}{2}) \)
6: clear \( s_0 \)
7: for \( i = 1, \ldots, L \) do
8: if \( \text{size}(s_i) = 0 \) then
9: \( s_i = s_{\text{temp}} \)
10: clear \( s_{\text{temp}} \)
11: else
12: \( s_{\text{temp}} = \text{MERGE}(s_{\text{temp}}, s_i) \)
13: \( s_{\text{temp}} = \text{PRUNE}(s_{\text{temp}}, \frac{\text{size}(s_{\text{temp}})}{2}) \)
14: if \( \text{size}(s_{\text{temp}}) < b \) then
15: \( s_i = s_{\text{temp}} \); clear \( s_{\text{temp}} \); break
16: else
17: clear \( s_i \)
18: end if
19: end if
20: end for
21: end if
22: Output: \( s = \text{MERGE}(s_0, s_1, \ldots, s_L) \)

Algorithm 1 shows the basic procedures of summarizing when a new data point \( x_i \) arrives in the data stream. In the algorithm, \( s_{\text{temp}} \) denotes a temporary sub-summary, \( s_j \) denotes the \( j \)-th level in the multi-level summary structure mentioned above.

PRUNE\((s, k)\) is an operation that converts a sub-summary \( s \) with size \( k \) and precision \( \epsilon' \) into a sub-summary with size \( k+1 \) and precision \( \epsilon' + \frac{1}{k} \). It is shown in [29] that each level in the summary can maintain an error less than \( \epsilon \). The PRUNE procedure is detailed below.

**Algorithm 2. Prune Operation PRUNE\((s, b)\)**

**Input:** Sub-summary \( s \), maximum size \( g \) of output sub-summary
1: \( i = 1 \), \( s_{\text{out}} = \emptyset \)
2: while \( d < b + 1 \) do
3: \( d = \lceil \frac{i-1}{b} \rceil w(s) \)
4: \( i_{\text{err}} = Q(s, d) \)
5: \( s_{\text{out}} \leftarrow s_{\text{out}} \cup \{ (x_{i_{\text{err}}}, r^{-}_{i_{\text{err}}}, r^{+}_{i_{\text{err}}}, w_{i_{\text{err}}}) \} \)
6: \( i + + \)
7: end while
8: Output: \( s_{\text{out}} \)

In the PRUNE operation, maximum \( b \) tuples are chosen from the input sub-summary according to the position indicator \( d = \frac{i-1}{b} w(s) \), \( i = 1, 2, \ldots, b + 1 \). Algorithm 3 below shows how to use the query operation \( Q(s, d) \) to choose tuples to form the new sub-summary. The \( r^{-}(x_i), r^{+}(x_i), \bar{w}(x_i) \) of the selected tuples are copied from the original sub-summary.

**Algorithm 3. Query Function \( Q(s, d) \)**

**Input:** Sub-summary \( s \) with tuple values \( v_i = x_i, i = 1, 2, \ldots, k \), position indicator \( d \) with \( 0 \leq d \leq w(s) \)
1: if \( d < \frac{1}{b} [r^{-}_i(x_i) + r^{+}_i(x_i)] \) then
2: return 1
3: end if
4: if \( d \geq \frac{1}{b} [r^{-}_i(x_k) + r^{+}_i(x_k)] \) then
5: return \( k \)
6: end if
7: Find \( i \) such that
8: \( \frac{1}{b} [r^{-}_i(x_i) + r^{+}_i(x_i)] \leq d < \frac{1}{b} [r^{-}_i(x_{i+1}) + r^{+}_i(x_{i+1})] \)
9: if \( d < r^{-}_i(x_k) + \bar{w}_i(x_i) + r^{+}_i(x_{i+1}) - \bar{w}_i(x_{i+1}) \) then
10: return \( i \)
11: else
12: return \( i + 1 \)
13: end if

Another operation is MERGE\((s_1, s_2)\), which combines two sub-summaries/summaries into one sub-summary/summary. Tuples from sub-summaries/summaries \( s_1 \) and \( s_2 \) are sorted together by their value \( v \). For each unique value \( v \) from \( s_1 \) and \( s_2 \), the quantities \( r^{-}(x_i), r^{+}(x_i), \bar{w}(x_i) \) are updated as follows:

\[
\begin{align*}
r^{-}(v) &= r^{-}_{s_1}(v) + r^{-}_{s_2}(v) \\
r^{+}(v) &= r^{+}_{s_1}(v) + r^{+}_{s_2}(v) \\
\bar{w}(v) &= \bar{w}_{s_1}(v) + \bar{w}_{s_2}(v)
\end{align*}
\]

Let the precisions of the two sub-summaries/summaries be \( \epsilon_a \) and \( \epsilon_b \). The precision of the merged sub-summary/summary \( s_{\text{temp}} \) is \( \max(\epsilon_a, \epsilon_b) \) [29]. The detailed MERGE algorithm is in Appendix A. Algorithm 5, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2022.3232752. Such MERGE operations are one-way mergeable based on the definition in [34]. The MERGE operation can also be applied to more than two sub-summaries/summaries. As one could see in the last line of Algorithm 1, the summary \( S \) is the outcome of a MERGE operation over all sub-summaries within the multi-level summary structure \( S \). Therefore, \( S \) also consists of several tuples.

It is shown in [29] that the outcome summary \( S \) is an \( \epsilon \)-approximate summary of the entire stream.

According to Algorithm 1, the decision whether to perform a PRUNE or a MERGE operation is based on the segment size \( b \) which is determined by the stream size \( N \). In order to obtain a summary from a data stream with unknown stream size \( N \), the input data stream is divided into pieces of disjoint sub-streams \( B_i, i = 0, 1, \ldots, m \). Sub-stream \( B_i \) has size \( \frac{N}{m} \) and covers data arriving in the time interval \( \left[ \frac{i-1}{m}, \frac{i}{m} \right) \). With fixed sub-stream sizes, aggregated
summaries can be obtained. Due to the fact that two summaries \( S \) and \( s \) have the same structure, \( S \) can perform PRUNE and MERGE operations with \( s \). Therefore after obtaining summaries \( S_i \) for each sub-stream \( B_t \), a multi-level summary structure \( [S] \) can now be constructed from the summaries \( S_i \) of the sub-streams \( B_t \). The procedure is described below:

1) The summary \( S_t \) of current sub-stream \( B_t \) is updated and maintained until the last data point in \( B_t \) has arrived. \( \epsilon' = \frac{\epsilon}{2} \) is used to set the size limit.

2) A \( \frac{\epsilon}{2} \)-approximate summary is obtained as an output of \( S_t \). The output is then set to PRUNE with the desired size of \( \frac{\epsilon}{2} \) and assigned to \( S_t \).

3) A set of summaries of all sub-streams \( \tilde{S} = \{S_0, S_1, \ldots, S_n\} \) is computed. \([S]\) is obtained by applying MERGE to the summaries from \( \tilde{S} \).

### 2.2.3 Exact Weight Update

When used as a query algorithm, only the rank \( r \) and value \( v \) of the tuple will be used to answer the question. According to the original PRUNE operation in Section 2.2.2, the selected tuples along with their stored elements are directly moved to the resulting summary with the unselected tuples removed. This guarantees the \( \epsilon \) maximum error, as shown in [7], [29]. However, for the purpose of providing accurate bin counts and correctly applying the fading factor later, especially when the input stream is sparse, the preservation of the complete weight values at the correct timestamps is required.

For this purpose, in the PRUNE operation, after each selection, instead of carrying elements directly from the original tuples to the tuples in the new summary, a different approach is taken. Let the value of each tuple associated with original summary be \( v_k, k \in \{1, 2, \ldots, h, \ldots, b\} \), where \( i \) is the index of the last selected tuple and \( j \) is the index of the currently selected tuple. Let the value of each tuple associated with output summary be \( u_h, h \in \{1, 2, \ldots, q, \ldots, b/2\} \), where the tuple containing \( u_h \) corresponds to the tuple containing \( v_j \). The following adjustments are made:

\[
\begin{align*}
    u_h &= v_j \\
    \tilde{r}^-(u_h) &= \tilde{r}^-(u_{i+1}) \\
    \tilde{r}^+(u_h) &= \tilde{r}^+(v_j) \\
    \tilde{w}(u_h) &= \sum_{k=i+1}^{j} \tilde{w}(v_k)
\end{align*}
\]  

For an input \( \epsilon' \)-approximate summary, the adjusted PRUNE operation will generate a \( (\frac{\epsilon}{2} + \frac{\epsilon}{2}) \)-approximate summary.

**Lemma 2.1.** Let \( U \) be the summary with \( h \) tuples pruned from an \( \epsilon' \)-approximate quantile summary \( V \). Then \( U \) is a \( (\frac{\epsilon}{2} + \frac{\epsilon}{2}) \)-approximate summary.

As it will be shown next, this new PRUNE procedure causes the error at each level of the multi-level summary to increase to a maximum \( 3\epsilon \). With other procedures remaining the same, the entire summary has a maximum error of \( 3\epsilon \).

**Proposition 2.1.** Each level the multi-level summary handled by the new PRUNE procedure maintains an error less than \( 3\epsilon \).

### 2.2.4 Quantile Binning

If the weight of each data point is 1, the weight element in a tuple can represent the number of data points this tuple covers. Using the procedure described in the last two sections, a final summary \([S]\) consisting of \( m \) tuples can be generated. \([S]\) is then further aggregated into a smaller number of bins. For the purpose of comparability with our offline method survey [1], procedures that mimic the discretization in [35] are introduced. Details are shown in Algorithm 4, where \( d_{inter} \) denotes the interval length when data points are equally divided into \( K \) segments, \( Bin_k \) denotes the bin count in the \( k \)-th bin, \( p_i \) is the position index of \( i \)-th point, and \( T_j,w \) represents the weight of the \( j \)-th tuple.

The score criteria defined in Section 2.2.1 can then be calculated using Bin\(_k\).

**Algorithm 4.** Bin Aggregation Procedure

**Input:** Summaries \([S] = T_1, T_2, \ldots, T_n\), number \( N \) of data points, user defined number of final bins \( K \).
1: \( d_{inter} = \frac{|S|}{K}, h = 0 \)
2: for \( i = 1, \ldots, K - 1 \) do
3: \( p_i = i * d_{inter} \)
4: end for
5: for \( k = 1, \ldots, K \) do
6: \( Bin_k = 0 \)
7: end for
8: for \( j = 1, \ldots, m \) do
9: \( Bin_k = Bin_k + T_j.w \)
10: if \(|Bin_k| \geq p_h \) then
11: \( h_{temp} = h \)
12: \( h = h + 1 \)
13: while \(|Bin_{h_{temp}}| \geq p_h \) do
14: \( h = h + 1 \)
15: end while
16: else
17: continue
18: end if
19: end for
20: **Output:** \( Bin_k, k = 1, K \)

### 2.2.5 Sparse Input

If the input data is sparse, one must take into account the fact that the zero values should be assigned weight one, therefore in the scenario of sparse input, the zero values need to be processed through the summary. However, processing zero values will be computationally inefficient. We take the advantage of the fact that sparse data has an enormous number of zero values and the quantile summary directly stacks the weights of identical data values together. The total number of zero values is recorded for each feature. When the algorithm is called to provide feature importance score, a single data point \((x, w)\), with \( x = 0 \) and \( w \) equal to the recorded number of zero values, is pushed through the algorithm before aggregating the summary.

Although the incoming order of the zero values are changed, the algorithm still builds the multi-level summary using PRUNE and MERGE operations. Therefore the summary under sparse input condition is still an \( 3\epsilon \)-approximate quantile summary.
2.2.6 Model Adaptation

In the situation of concept drift, a fading factor strategy is still adopted. Let $\alpha$ denote the fading factor, $w_i$ be the weight value of data point at time $i$ and $W_i$ be the weight values in the multi-level quantile summary at time $i$. Then the fading equation is:

$$W_i = \alpha W_{i-1} + w_i \quad (16)$$

Notice in (16), the weight in $W_{i-1}$ needs to be updated every time a new data point arrives. This includes all the tuples on all the levels in the summary. Such high frequency repeated calculation can be very time consuming. Therefore, we split the update process into two parts. The first part, we update only on recently established tuples in $s_0$ (shown in Fig. 1). The second part, based on the number of data points covered in $s_0$, we only penalize the weights in the multi-level structure whenever a PRUNE and MERGE operation initialized by $s_0$ happens. Let the number of data points in $s_0$ be $k$ and the weights in multi-level summary be $W_i$. The update for part two becomes:

$$W_i = \alpha^k W_{i-1}. \quad (17)$$

Sparse Input. When sparse inputs are received in a model adaptation setting, since the weight of each data point is penalized according to the order it arrives, injecting the weights of all zero value data points at the end does not provide the correct penalized weight. In order to keep the weight accumulation match the data point order for the zero value data points, time anchors and universal weight maps are employed. The universal weight map is a vector that stores the accumulated penalized weight at each timestamp for each class. These maps are used across all features. The time anchors are designed to record the timestamp of the last non-zero data point.

When the incoming data stream has at least two classes, simply recording the class label and recovering the penalized weight for zero values by repeatedly penalizing the weights in the summary obtained so far can be extremely time consuming. In contrast, the universal weight map and time anchor can achieve the same purpose with very small extra storage space and faster computing speed. Given the universal weight map recorded so far for each class, the weight for all zero values between last non-zero data point and the current non-zero data point can be calculated. Let the time index for last non-zero data point be $a$ and current non-zero data point be $b$.

$$w_c = M_{b,c} - M_{a,c} \times \alpha^{(b-a)} \quad (18)$$

$w_c$ denotes the recovered weight for zero values, $M_{b,c}$ indicates the recorded weight value for class $c$ at time index $b$. When a non-zero value arrives, $w_c$ is calculated and added to summary before any other procedures.

The handling of a sparse stream here also only changes the passing order of the zero values without changing the PRUNE and MERGE operation. Therefore the multi-level summary in this situation is still $3\epsilon$-approximate.

2.3 Time Complexity

Comparing to [29], our quantile summary calculation when in sparse-input-without-drifting mode has a time complexity of $O(M_s N_s \log \left( \frac{1}{\epsilon \log \epsilon N_s} \right))$ to account for all $M$ features. Here, $M_s$ is the average column density, and $N_s$ is the average number of nonzero values plus one across all features. In this mode, the 0 level block size is still calculated from the sub-stream size $N'$. Therefore the time spent on each summary of the sub-stream given the sub-stream size $N'$ is still $N' \log \left( \frac{1}{\epsilon \log \epsilon N'} \right) + \sum_{i=2}^{s} \log \left( \frac{1}{\epsilon \log \epsilon N'} \right)$. However, the total number of actual instances becomes $N_s$ instead of $N$, which causes the number of sub-streams to decrease. Moreover for each actual instance, the operations are only operated on part of the total features. Without loss of generality, $M_s$ is used to represent the activated features in each iteration. In total, it gives a time complexity of $O(M_s N_s \log \left( \frac{1}{\epsilon \log \epsilon N_s} \right))$. To put it into perspective, for the real datasets we used in Section 3.3, the $M_s$ values are 242, 115 and 169, and the $N_s$ are 4.97, 80.24 and 37.

When our method operates under sparse-input-with-drifting mode, it has a time complexity of $O(M_s N_s \log \left( \frac{1}{\epsilon \log \epsilon N_s} \right))$ where $N_s$ is interpreted differently than the non-drifting situation. Here $N_s$ is twice the number of non-zero values. At 0 level, the block size is still calculated from the sub-stream size $N'$. However, to maintain the correct penalization, each time when a PRUNE operation happens at level 0, an extra scanning of every tuple in the current multi-level summary structure needs to be performed. Therefore at the 0 level, instead of spending $\sum_{i=1}^{N} (\log b + \frac{b}{2})$ time for the zero value data points, time anchors and universal weight maps are employed. The universal weight map is a vector that stores the accumulated penalized weight at each timestamp for each class. These maps are used across all features. The time anchors are designed to record the timestamp of the last non-zero data point.

For the summary of each sub-stream, it becomes $N' \log b + \frac{3N'}{2} + N' \log \frac{N'}{b}$, which still has $O(N' \log \frac{1}{\epsilon \log \epsilon N'})$. At the level of the entire algorithm, the total number of actual instances becomes $N_s$ instead of $N$, which causes the number of sub-streams to decrease. Similar to the sparse-input without drifting mode, it gives a time complexity of $O(M_s N_s \log \left( \frac{1}{\epsilon \log \epsilon N_s} \right))$. To put it into perspective, for the real datasets we used in Section 3.3, the $M_s$ are 242, 115 and 169, and the $N_s$ are 7.94, 158.48 and 72.

2.4 Minibatch Processing

We have noticed during our experiments that for non-sparse inputs, the summaries of all features are required to be visited for each new observation. This behavior incurs an increased computation time. Therefore within a reasonable storage budget, batch data handling can be integrated in the algorithm as it gives a considerable acceleration by reducing the visit frequency. This batch procedure is denoted by minibatch in the rest of the text.

3 Experimental Evaluation

Experiments are conducted first on small real datasets to evaluate the difference between the proposed online screening
methods and their offline counterparts. Then simulations are conducted to evaluate concept drift, and finally experiments on large real datasets are conducted and compared with other online screening methods from the literature.

### 3.1 Comparison Between Online and Offline Methods

To perform the experiments, each dataset is processed in a one-pass fashion by both online and offline version of screening methods respectively. Each dataset is passed through each algorithm and parameter setting once and the weight scores for all features are calculated. For quantile summary based methods, we use \( K = 5 \) quantile bins throughout all our experiments. We test the computation time by fixing either the minibatch size or the precision parameter \( \epsilon \) and varying the other parameter. When testing the approximation accuracy, we only fix the minibatch size. The fixed values are 250 for the minibatch and 0.001 for \( \epsilon \). The varying ranges are \( \epsilon = \frac{1}{f} \), where \( f \in \{5, 50, 100, 500, 1000, 1500, 2000\} \) and minibatch\( = 2^k \), where \( k \in \{0, 1, 2, \ldots, 11\} \). The feature rank is calculated by sorting the feature weights monotonically according to the importance score obtained by the corresponding screening method. Features that have low rank values are more important (i.e., a feature with rank value 1 is the most important). For Gini index, the feature that has smaller weight has lower (better) rank value. On the contrary, for the other methods, the feature that has larger weight has lower rank value. Feature weights and feature ranks were used to construct different kinds of tables to evaluate the performance of the online methods and compare them with offline methods.

#### 3.1.1 Construction of Comparison Tables

Five types of tables are constructed according to minibatch size, feature weights and feature rank.

1) **The influence of minibatch size on computation time.** This table shows the computation time (in milliseconds) of the online quantile based methods. Here we only evaluate the time using Matlab 2018b [42]. For the offline screening methods we used the same Matlab 2018b implementations as those in [1]. The online screening methods were implemented by ourselves.

#### 3.1.2 Data Sets

The datasets used for online-offline comparison are shown in Table 1.

Gisette, Dexter, and Madelon are part of the NIPS 2003 Feature selection challenge [36] and are also available on the UCI Machine Learning Repository. Dorothea is part of the NIPS 2003 Feature selection challenge and is also available on the UCI Machine Learning Repository. We combined the training and validation sets in order to get a larger sample body. The Url dataset contains a total of 121 data files, one for each monitored day. We only used data from Day0 in our experiments. The Kdd12 dataset originates from the second track of the KDD Cup 2012. The raw version can be found on kaggle.com, made available by the organizers and Tencent Inc. The data we use comes from LIBSVM [41]: a library for support vector machines. Only the first 16000 samples were used in our experiments.

#### 3.1.3 Results

The following results are based on the output generated using Matlab 2018b [42]. For the offline screening methods we used the same Matlab 2018b implementations as those in [1]. The online screening methods were implemented by ourselves.

Comparison of the Moving Average Based Methods. It is proved in Section 2.1 that the moving average based online screening methods can achieve exactly the same result as their offline version. Therefore we don’t provide any real data analysis here.

Comparison of the Online Quantile Based Methods. First we conducted a study of the computation time of the online quantile based methods. Here we only evaluate the time from when the data was input to when the sample point

### Table 1

The Classification Datasets Used for Evaluating the Online/Offline Screening Methods

| Dataset       | Feature type | Number of features | Number of observations |
|---------------|--------------|--------------------|------------------------|
| Gisette [36]  | Continuous   | 5000               | 7000                   |
| Dexter [36]   | Continuous   | 20000              | 600                    |
| Madelon [36]  | Continuous   | 500                | 2600                   |
| SMK_CAN_187   | Continuous   | 19993              | 187                    |
| GLI_85 [38]   | Continuous   | 22283              | 85                     |
| Dorothea [36] | Continuous   | 100,000            | 1,150                  |
| Url [39]      | Continuous + Binary | 74,110 | 16,000                |
| Kdd12 [40]    | Binary       | 48,957             | 16,000                 |

5) The influence of \( \epsilon \) on the rank accuracy among top 10% most important features. This table shows the average unmatched ranking ratio of online methods with respect to corresponding offline methods when varying the value of \( \epsilon \). Only the top 10% of features ranked by offline scores are involved. The average mis-rank ratio is calculated as

\[
\text{DR} = \frac{1}{p} \sum_{j=1}^{p} \frac{|w_{on}^j - w_{off}^j|}{\max(W_{off}) - \min(W_{off})},
\]

where \( p \) is the total number of features. \( W_{on} \) and \( W_{off} \) are the score vectors of all features generated from online and offline methods, \( w_{on}^j \) and \( w_{off}^j \) are the scores of the \( j \)-th feature from online and offline methods.
is the target feature vector, the relation is assumed to be
Furthermore, in some cases, the results are reported as sample count differences per bin per feature. Although different datasets have different schedules, all differences were annihilated when \( \epsilon \) decreased below 0.001. The online quantile method can in practice achieve a speed advantage with little to no error in bin count estimation.

Table 5 shows the accuracy measure from Eq. (19) of the scores calculated by three online methods compared to their offline counterparts when the \( \epsilon \) value was varied. The scores were computed using the Chi-square score, Gini index and Mutual information respectively. Among the three methods

\[ y_i = x_i^T \beta_i + c + e_i, \]

where \( \beta_i \) is a coefficient vector with its elements corresponding to the elements in \( x_i \), \( \epsilon \) is a constant, \( e_i \) is a random noise \( e_i \sim N(0,1) \). The feature vector \( x_i \) is generated from a
uniformly correlated model:

\[ x_i = n z_i + e_i, \]  

(21)

\[ z_i \sim N(0,1), \]  

where \( z_i \sim N(0,1), \) \( e_i \) is a vector of the same length as \( x_i \), with all entries equal to 1, \( v \) is a parameter to control the correlation between features, and \( e_{ij} \) is a noise vector with its \( j \)-th element \( e_{ij} \sim N(0,1). \) In this setup, the correlation between any two features is \( n = (1 + v)^{-1}. \) In our experiment \( v \) is set to 0.5.

Let \( j \) denote the element index in the coefficient vector \( \beta_i \) and \( k \) denote the number of non-zero coefficients. The indices of non-zero coefficients are shifted every \( l \) observations. Let \( b \) denote the non-zero signal value. Then \( \beta_i \) is constructed as:

\[ \beta_{ij} = \begin{cases} b & \text{if } j \in (|J| : |J| + k) \\ 0 & \text{otherwise} \end{cases} \]  

(22)

In our experiments, we choose the feature space to be 1000 dimensional, thus \( x_i \in \mathbb{R}^{1000}. \) The number of true features \( k \) is set to 100. The coefficient signal value is set to \( b = 1. \) A total of 100,000 samples are generated. We change the value of \( l \) (number of samples until a feature shift occurs) to control the concept drift level.

### Results

In this section, we use detection rate @\( k \) to measure how well the algorithm performed in recovering the true features. Given a set \( FS \) of \( |FS| = k \) selected feature indices and a set \( TU \) of true feature indices, the detection rate @\( k \) is defined as:

\[ \text{DetRate}_{@k} = \frac{|FS \cap TU|}{|TU|} \]  

(23)

All \( k \)-s in this section are set to 500.

**Performance Plots.** Fig. 2 visualizes the performance of the online screening methods in handling data with concept
drift. Fig. 2a shows the overall performance of different screening methods as the detection rate @500 versus sample size $N$. The shifting rate $l$ is fixed to 2000, and the fading factor $\alpha$ is fixed to $\alpha = 0.9$. The five screening methods show very similar performance. However compared to the others, the Gini Index requires more samples to establish a good performance at the beginning. Fig. 2b shows how the shift rate $l$ affects the variable detection rate. The fading factor $\alpha$ is still fixed to $\alpha = 0.9$. It is obvious that a fast shifting rate has a negative impact on the performance of the algorithm. From Fig. 2c it can be observed that by adjusting the fading factor $\alpha$, the detection rate can be improved even for data with a faster shifting rate. The heatmap from Fig. 2d shows the impact of the fading factor on detection rate in detail.

In Fig. 3, it is shown in detail how the detection rate changes when different numbers of features are selected by the online screening methods as well as whether model adaptation helps improve the performance of the screening methods. Across all four shift rates that were tested, the screening methods with model adaptation outperform those without model adaptation. Generally, the slower the shift the fewer features need to be selected to provide a full detection. Even in the fastest shift rate that is tested (shift every 250 samples), the screening methods with model adaptation still manage to detect all true features with fewer selected features and consistently detect more true features than the methods without model adaptation.

### 3.3 Real Data Evaluation

This section evaluates the performance of the introduced feature screening methods on large real world datasets. The misclassification error rate is used as the evaluation metric. The time unit used for evaluation is seconds. Our methods and the sparse FSA learner were implemented in C++ and compiled in MATLAB 2018b. The rest of the comparison methods, learners and the other main experiments were conducted using Python 3.7.

**Screening With Adaptation versus Non-Adaptation.** We first compare the performance of the introduced screening methods with adaptation versus their non-adaptation versions. All screening methods will be used to select the same number of features. Three learners were trained on the data with the selected features. For learners, Sparse FSA, SGD with log loss, and SGD with perceptron loss (linear loss) are chosen. The SGD methods were from the Python scikit-learn package.

**Introduced Methods versus Others.** In the second part, we compare the performance of our methods with other state-of-the-art feature screening methods with drifting adaptation functionality. FIRES [21] and MC-NN [19] were selected not only due to their state-of-the-art performance shown in their own papers but also because they met the minimum computing time requirement for processing our largest dataset. For FIRES, we used the Python implementation from the author’s GitHub repository. The implementation of MC-NN was obtained from pystreamfs Python package. Both methods were modified by us to handle sparse matrices more efficiently and to reduce computing time.

For each experiment setting, we conduct 10 repeated runs with a fixed set of different random seeds. This is due to the fact that the order of the instances can generate result differences from one run to another. All values in our tables are the average of these 10 runs. The only exception are all the experiments with URL data using FSA as the learner.
due to its long training time. The model parameters in our experiments were also not fine tuned to show optimal results. This is to prevent over tuning. The goal is to compare different feature screening methods under the same settings.

### 3.3.1 Data Sets

The three datasets used are presented in Table 7. There, the number of instances represents the size of the training sets. The number of features is the count of all features in the training sets that don’t have all-zero values.

The Url dataset has already been introduced in Section 3.1. Here Url data from day 0 to 99 is used to train the model. Data from day 100 is used as test set. Because this data has huge number of instances and features, only the data from day 98 and day 99 was processed by the FIRES and the MC-NN methods. The training part of this dataset has a 0.99996 sparsity.

The 20NewsGroups is an email content data that is generated according to \[44\]. The data originates from the UCI repository [45]. We also extracted the timestamp for each email and sorted the data in time order. The emails before “1993-04-22 00:00:00” are used as training data and the rest as test data. The training part of this dataset has a 0.99975 sparsity.

The Spam Corpus was developed in [46]. We used the version from spamassassin.apache.org. The data remains its original order. 80% of the data was used for training. The training part of this dataset has a 0.99975 sparsity.

### 3.3.2 Results

The detailed comparison of adaptive versus non-adaptive methods as well as training with versus without feature screening are presented in Appendix C, available in the online supplemental material.

Tables 8 and 9 show results from the best of the introduced adaptive feature screening methods comparing to state of art methods. For the moving-average based and quantile based methods, the best of each category was selected.

The three sub-tables in Table 8 show the misclassification error rate for three different learners respectively. The best value of each dataset is shown in bold. We also conducted two sample T-test between each pair of compared methods. If a method showed no significant difference ($p \geq 0.05$) from the best performed method, it is also shown in bold.

Using Sparse FSA as the learner, the quantile based methods had the best result on two datasets. Because we didn’t perform hypothesis test on URL data, we can’t say that best performance of the quantile based method is significant. The moving average based methods had the best result on one dataset while the result of FIRES was not significantly different than that of the moving average based method’s. Using L-SGD, the quantile based methods performed the best on all three datasets, while for URL data, the moving average based methods and MC-NN weren’t shown to be significantly different than the quantile based methods. For results from the P-SGD learner, the quantile based methods performed the best in two datasets. The moving average based methods performed best in one dataset. For 20NewsGroup data, the moving average didn’t show a significant difference from the best performing method. For the Spam Corpus, FIRES didn’t show a significant difference from the best performing method. For URL, all other three methods didn’t show significant difference from the best performing method.

Table 9 compares the computing time of the compared methods. It should be noted that our methods were implemented in C++ while the two compared methods FIRES and MC-NN were implemented in Python. For URL, FIRES and MC-NN only processed two days’ data. Therefore their computing time for the full data will be roughly 50 (total 100 days) times the time showed in the table. Our experiments using the aforementioned implementations indicate that our methods seem to be faster than the two methods compared, while usually doing a better job at screening.

### 4 Conclusion

In our studies of online screening methods, the moving average based online screening methods are proved to have the same performance as their offline version and have the advantage of a faster speed and lower storage requirements. The experiments in Section 3.1 show that the bin-count based online screening methods can also achieve the same results as their offline versions given the right $\epsilon$. Moreover, they can obtain faster or about the same computation speed compared to their offline versions with the right combination of $\epsilon$ and minibatch size.

The results in Section 3.2 give empirical evidence that adding model adaptation capability to screening methods...
can help improve their performance when data exhibits concept drift. It is shown that to some degree, adjusting the fading factor can assist screening methods to tackle datasets with a high concept drift rate.

The real data analysis in Section 3.3 further demonstrates the capability of online screening methods in dealing with large real datasets with sparsity and possible concept drift. The results in the tables of Appendix C, available in the online supplemental material, show that for the 20news-groups data, in 3 out of 3 scenarios, our best performing online screening methods provide significant improvement to underlying learning algorithms. For URL data, the table shows that none of the online screening methods provides significant improvement to the underlying learning algorithms. The table of the Spam Corpus data shows that in 1 out of 3 scenarios, our best performing online screening method provides significant improvement to underlying learning algorithms.

Comparing the online screening methods with and without adaptation, among all 4 scenarios where using feature screening yielded significantly better results than not using them, 3 of these best results are generated by using the adaptive version of the screening methods. All of these differences are significant, and all of them are quantile/bin-count based methods.

For all experiments, the training time after the feature screening procedure was significantly reduced, which is expected. It is worth noting the time saving benefit is more obvious when the underlying learner is supposed to take a lot of time, such as in the case of the URL data using Sparse FSA. Two exceptions are when Sparse FSA was used on the 20NewsGroup and Spam Corpus datasets. There is no improvement in time because the Sparse FSA computation depends on the sparsity of the remaining features, and possibly the most sparse features were eliminated in the first few epochs of Sparse FSA.

When comparing with other state-of-the-art methods, quantile/bin-count based online screening methods with adaptation perform the best. Even when taking statistical significance into consideration, they still score the best performance 6 times (excluding Url data with FSA) compared to FIRES which was best 2 times (ties to moving-average based methods). Our introduced methods (C++) also used lesser processing time comparing to the other two methods (Python), especially for datasets with large number of observations and features such as URL.

In summary, our introduced online feature screening methods show advantages when it comes to processing time. This advantage expands further, the more computationally intensive the underlying learning task is. Our methods were also shown to improve the learning tasks in 4 out of 8 cases (excluding Url data with FSA) and tie on the performance in 4 out of 8 cases (again excluding Url data with FSA). Among the improved cases, the adaptive version of our methods also showed better performances (3 out of 4) than non-adaptive when data was expected to have a drifting effect. All 3 of them are quantile/bin-count based adaptive screening methods, which were our focus. Our methods, especially quantile/bin-count based, also showed better performance in both misclassification error rate and computing time than other state-of-the-art methods in tasks of tackling large and even ultra large sparse datasets with the potential of concept drift.

5 Future Work

Some criteria used by screening methods such as mutual information and Gini index are also used in some learning algorithms such as decision trees as a component to measure impurity. It will be an interesting topic to research how to integrate our introduced procedures with these learning algorithms, especially to cope with training on sparse, streaming, and concept drifting datasets.

Moreover, throughout the years some works showed that decision trees can be mapped into certain neural networks. Several mapping methods were mentioned in [47], [48], [49] and [50]. A new framework named Neural Rule Ensembles (NRE) [51], introduced in the recent literature, also focused on such a mapping strategy. It shows that any decision tree can be mapped into a set of neural rules, and the ensemble of neural rules can subsequently be trained using backpropagation. Therefore the expected improvement in training speed on decision trees using such online screening measures can hopefully also reflect on neural network training.

References

[1] M. Wang and A. Barbu, “Are screening methods useful in feature selection? An empirical study,” PLoS One, vol. 14, no. 9, 2019, Art. no. e0220842.
[2] J. C. Davis and R. J. Sampson, Statistics and Data Analysis in Geology. Hoboken, NJ, USA: Wiley, 1986.
[3] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification. Hoboken, NJ, USA: Wiley, 2012.
[4] C. Gini, “Variability and mutability, contribution to the study of statistical distribution and relations,” Studi Economico-Giuridici della R, 1912.
[5] H. Liu and R. Setiono, “Chi2: Feature selection and discretization of numeric attributes,” in Proc. IEEE Int. Conf. Tools Artif. Intell., 1995, pp. 388–391.
[6] D. D. Lewis, “Feature selection and feature extraction for text categorization,” in Proc. Workshop Speech Natural Lang., 1992, pp. 212–217.
[7] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2016, pp. 785–794.
[8] R. Agrawal and A. N. Swami, “A one-pass space-efficient algorithm for finding quantiles,” in Proc. Int. Conf. Manage. Data, 1995.
[9] R. Tibshirani, “Regression shrinkage and selection via the lasso,” J. Roy. Statist. Soc. Ser. Methodol., vol. 58, pp. 267–288, 1996.
[10] L. Nie, L. Zhang, Y. Yang, M. Wang, R. Hong, and T.-S. Chua, “Beyond doctors: Future health prediction from multimedia and multimodal observations,” in Proc. ACM Int. Conf. Multimedia, 2013, pp. 591–600.
[11] L. Nie, X. Song, and T.-S. Chua, Learning From Multiple Social Networks. Berlin, Germany: Springer, 2016, pp. 1–118.
[12] F. A. M. Zaki and T. S. Chin, “FWFS: Selecting robust features for evolving data streams with concept drift and class imbalance,” Comput. Intell. Neurosci., vol. 2021, 2021, Art. no. 8813806.
[13] J. C. Chamby-Diaz, M. Recamande-Mendoza, and A. L. Bazzan, “Dynamic correlation-based feature selection for feature drifts in data streams,” in Proc. IEEE 5th Braz. Conf. Intell. Syst., 2019, pp. 198–203.
[16] J. P. Barddal, F. Enembreck, H. M. Gomes, A. Biêt, and B. Pfahringer, “Boosting decision stumps for dynamic feature selection on data streams,” *Inf. Syst.*, vol. 83, pp. 13–29, 2019.

[17] B. Sovdat, “Updating formulas and algorithms for computing entropy and Gini index from time-changing data streams,” *Proc. ACM SIGKDD Exp. Neww., Seis.*, vol. 14, no. 3, pp. 634–643, 2018.

[18] C. Deng et al., “Enhancing the discrimination ability of a gas sensor array based on a novel feature selection and fusion framework,” *Sensors*, vol. 18, no. 6, 2018, Art. no. 1909.

[19] M. S. Hammad, F. Stahl, and A. Badii, “Real-time feature selection technique with concept drift detection using adaptive micro-clusters for data stream mining,” *Knowl.-Based Syst.*, vol. 161, pp. 205–239, 2018.

[20] G. A. Magpa, G. Mosciano, E. Martinelli, and C. Di Natale, “Unsupervised on-line selection of training features for a robust classification with drifting and faulty gas sensors,” *Sensors Actuators B, Chem.*, vol. 258, pp. 1242–1251, 2018.

[21] J. Haug, M. Pawelczyk, K. Broelemann, and G. Kasneci, “Leveraging model inherent variable importance for stable online feature selection,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 1478–1480.

[22] D. W. Fernando and N. Komninos, “FeSA: Feature selection architecture for ransomware detection under concept drift,” *Comput. Secur.*, vol. 116, 2022, Art. no. 102659.

[23] J. I. Munro and M. S. Paterson, “Selection and sorting with limited storage,” *Theor. Comput. Sci.*, vol. 12, no. 3, pp. 315–323, 1980.

[24] R. Jain and I. Chlamtac, “The F2 algorithm for dynamic calculation of quantiles and histograms with arbitrary sampling observations,” *Commun. ACM*, vol. 28, no. 10, pp. 1076–1085, 1985.

[25] J. Gama and C. Pinto, “Discretization from data streams: Applications to histograms and data mining,” in *Proc. ACM Symp. Appl. Comput.*, 2006, pp. 662–667.

[26] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, “Approximate medians and other quantiles in one pass and with limited memory,” *ACM SIGMOD Rec.*, vol. 27, no. 2, pp. 426–435, 1998.

[27] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, “Random sampling techniques for space efficient online computation of order statistics of large datasets,” *ACM SIGMOD Rec.*, vol. 28, no. 2, pp. 251–262, 1999.

[28] M. Greenwald et al., “Space-efficient online computation of quantile summaries,” *ACM Sigmod Rec.*, vol. 30, no. 2, pp. 58–66, 2001.

[29] Q. Zhang and W. Wang, “A fast algorithm for approximate quantiles in high speed data streams,” in *Proc. IEEE Int. Conf. Sci. Stat. Database Manage.*, 2007, pp. 29–29.

[30] X. Lin, H. Lu, J. Xu, and J. X. Yu, “Continuously maintaining quantile summaries of the most recent N elements over a data stream,” in *Proc. IEEE Int. Conf. Data Eng.*, 2004, pp. 362–373.

[31] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, “Space-and-time-efficient deterministic algorithms for biased quantiles over data streams,” in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Proc. Database Syst.*, 2006, pp. 263–272.

[32] M. B. Greenwald and S. Khaanna, “Power-conserving computation of order-statistics over sensor networks,” in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Proc. Database Syst.*, 2004, pp. 275–285.

[33] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, “Medians and beyond: New aggregation techniques for sensor networks,” in *Proc. Int. Conf. Embedded Netw. Sensor Syst.*, 2004, pp. 239–249.

[34] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, “Mergeable summaries,” *ACM Trans. Database Syst.*, vol. 38, no. 4, pp. 1–28, 2013.

[35] X. V. Nguyen, “Information theoretic feature selection, version 5.0,” Updated 07 Jul. 2014, 2014. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/47129-information-theoretic-feature-selection

[36] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror, “Result analysis of the NIPS 2003 feature selection challenge,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2005, pp. 545–552.

[37] A. Spira et al., “Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer,” *Nature Med.*, vol. 3, no. 3, pp. 203–207, 1997.

[38] W. A. Freije et al., “Gene expression profiling of gliomas strongly predicts survival,” *Cancer Res.*, vol. 64, no. 18, pp. 6503–6510, 2004.

[39] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Identifying suspicious URLs: An application of large-scale online learning,” in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 681–688.