Aging and rejuvenating strategies for fading windows in multi-label classification on data streams

Martha Roseberry  
Virginia Commonwealth University  
Richmond, Virginia  
mroseberry@vcu.edu

Sašo Džeroski  
Jožef Stefan Institute  
Jožef Stefan International Postgraduate School  
Ljubljana, Slovenia  
saso.dzeroski@ijs.si

Albert Bifet  
AI Institute, University of Waikato  
Hamilton, New Zealand  
LTCI, Télécom Paris, IP Paris  
Paris, France  
abifet@waikato.ac.nz

Alberto Cano  
Virginia Commonwealth University  
Richmond, Virginia  
acano@vcu.edu

ABSTRACT
Combining the challenges of streaming data and multi-label learning, the task of mining a drifting, multi-label data stream requires methods that can accurately predict labelsets, adapt to various types of concept drift and run fast enough to process each data point before the next arrives. To achieve greater accuracy, many multi-label algorithms use computationally expensive techniques, such as multiple adaptive windows, with little concern for runtime and memory complexity. We present Aging and Rejuvenating kNN (ARkNN) which uses simple resources and efficient strategies to weight instances based on age, predictive performance, and similarity to the incoming data. We break down ARkNN into its component strategies to show the impact of each and experimentally compare ARkNN to seven state-of-the-art methods for learning from multi-label data streams. We demonstrate that it is possible to achieve competitive performance in multi-label classification on streams without sacrificing runtime and memory use, and without using complex and computationally expensive dual memory strategies.

CCS CONCEPTS
- Computing methodologies → Machine learning algorithms;

KEYWORDS
Machine Learning, Data Streams, Multi-label Learning

1 INTRODUCTION
As the amount of continuously generated data produced by everyday systems and devices increases, so does the need to accurately and efficiently mine data streams. Not only must models be able to make predictions for incoming data in real time, but they must also be able to incorporate new data and update in response to the constantly evolving nature of the stream [3]. Many methods have been proposed for responding to concept drift, including sliding windows, adaptive windows and evolving ensembles [9]. To improve accuracy, these methods often sacrifice simplicity and require substantial resources to run but resources are limited and as the data increases in complexity, their resource use becomes prohibitive. Multi-label streaming data [28] is a good example of complex data for which many proposed solutions require significant resources. It is common for multi-label classification methods to rely on multiple classifiers or multiple windows of short-term and long-term data. The complexity of the methods is dependent on the number of labels, which may be high [28].

In this paper, we present a series of resource efficient techniques to age and rejuvenate instances in a single data window. Each technique builds on the previous, with the final combined method, Aging and Rejuvenating kNN (ARkNN), weighting each data instance within the window based on its age, past performance, and similarity to the incoming data. Aging techniques allow us to decrease the importance of older and worse performing data, allowing ARkNN to respond to concept drift and prune data that contributes to error. Rejuvenation techniques allow us to keep beneficial instances without the complexity of multiple windows. We experimentally compare each added modification to demonstrate the impact of each technique on performance, runtime and memory use. The final ARkNN method is then experimentally compared to seven state-of-the-art multi-label streaming methods. We demonstrate that performance of ARkNN is better or comparable to that of state-of-the-art methods, while the average runtime of ARkNN is one fifth the next fastest model and its average memory use is 1/50th of the time of its nearest competitor.
The main contributions of this paper are:

- ARkNN: a fast and memory efficient algorithm for multi-label classification on data streams using instance aging and rejuvenation.
- A detailed analysis of the impact of six techniques to age, rejuvenate and prune data to respond to concept drift and improve performance with minimal resource cost.
- A thorough experimental study comparing the predictive performance, runtime, and memory use of ARkNN against seven state-of-the-art algorithms for multi-label classification on data streams across 37 datasets.

The rest of the paper is organized as follows. Background on data streams, multi-label learning and related windowing, aging and rejuvenation techniques is given in Section 2. A description of each of the six proposed techniques and how they build upon one another is detailed in Section 3. Section 4 describes the experimental setup, with experimental results and discussion presented in Section 5. Concluding remarks and future work are presented in Section 6.

2 BACKGROUND

Resource Efficient Data Mining

Most machine learning literature focuses on the accuracy and predictive power of machine learning algorithms. In many cases, however, computing resources are limited. As more data is collected and available to be mined, this problem is unlikely to diminish. In recent years, attention has been drawn to energy efficient data mining, particularly energy efficient deep learning. Others have suggested techniques for measuring energy and resources with the goal of making it easier for researchers to evaluate resource consumption when comparing methods. Mining streaming data is one scenario where resource efficiency is highly important, as the speed of the data can quickly make resources scant. However, less research works in this area focus on minimizing resource consumption [10].

Streaming Data

Streaming data refers to the situation where data is arriving as a potentially unbounded sequence. Formally, a data stream is an ordered sequence of data \( S = \{s_1, s_2, \ldots, s_t, \ldots\} \), where the data instance \( s_t \) arrives at time \( t \). Streaming data presents both the problem of never having all the data available and having a limited amount of time to process instance \( s_t \) before instance \( s_{t+1} \) arrives and creates a backlog. Further complicating data streams, it is generally assumed that the stream is not static.

Concept Drift

A concept drift is a change in the distribution of the incoming instances that may cause a change in the decision boundaries [12]. A wide variety of strategies for dealing with concept drift have been proposed, usually broken into two categories - continuously evolving methods and concept drift detectors [9]. Drift detectors use an outside method to monitor the performance of the learner. As the stream continues, if the detector encounters a sufficient loss in performance, it will trigger the learner to update or re-train based on newer data. A benefit of concept drift detectors is that the model only re-trains upon the detection of drift, saving resources when the model is stable. A disadvantage is that drift detectors are most effective for abrupt drift, where the data distribution changes suddenly.

However, concept drift may also appear as gradual or incremental drift. During gradual drift, instances from both the old concept and the new concept appear for some time together, with instances from the new concept slowly becoming dominant. In incremental drift, the old concept slowly transitions into the new concept. Both take place more slowly than abrupt drift, without a sharp divide between the two concepts. This makes gradual and incremental drifts difficult for drift detectors to detect. To handle slower drifts, many models incorporate a method for continuously evolving.

Sliding, Fading and Adaptive Windows

Continuous methods typically involve mechanisms to update the classification model as new information arrives and forget older concepts [9]. One of the most basic methods is to use a first-in, first-out sliding window. In its most basic form, a sliding window is a window of size \( m \) in which each data instance is stored. Once the window is full, as the newest instance arrives at time \( s_t \), the oldest instance in the window \( s_{t-m} \) is discarded. Using this mechanism, the model trained on the window slowly evolves with the stream, forgetting older concepts.

Sliding windows are an efficient method for forgetting older information, but while data is still in the window, all instances are valued equally. However, for a drifting stream, more recent information is often seen as more valuable. This leads to the definition of damped windows or fading windows [9]. In fading windows, a fading factor or decay function is used to weight older information, such that newer information has a higher weight and has a greater impact on the predictions. In this way, older instances in the window are not yet forgotten, but if the concept has shifted, the classifier will update more quickly because of the higher weight of new data. Fading windows are popular in pattern mining and finding frequent itemsets [7, 15, 26]. Fading or aging is also a common technique used in weighting ensembles of classifiers [11, 13] and fading windows can also be used to detect drift or evaluate classifiers [8, 20].

Forgetting mechanisms and drift detectors can also be used together. Adaptive size windows act as traditional sliding or fading windows, but monitor the incoming data to change the window size and abruptly forget information in the presence of abrupt concept drift [9]. Other methods use multiple windows. The popular adaptive windowing technique ADWIN monitors the data over two windows [5]. When a change is detected, the older window is abruptly dropped. Another mechanism uses two windows as short and long-term memories, summarizing past data into the long-term memory whenever the short-term memory is reduced due to a drift [14]. Multiple window methods can be very effective, but by requiring that multiple windows be maintained and updated, they often suffer from time and memory complexity.

Multi-Label Streaming Data

While complexity is a concern for all data stream mining, it is a particular challenge for multi-label classification of streaming data. Multi-label data refers to the situation where each data instance can be classified as belonging to multiple classes simultaneously. The classes are not mutually exclusive. In contrast to multi-class classification, each instance has the form \( s = (x, y) \) where \( y \) is the single relevant class, in multi-label classification \( s = (x, Y) \) where
Within the window, all instances are weighted equally. A k-nearest neighbors are found using the cosine distance, $D_C(s, n) = 1 - S_C(s, n)$ where $S_C$ is the cosine similarity of the instance $s$ and the neighbor $n$. For each label $l$ independently, the algorithms predicts label $l$ is relevant if a majority of the nearest neighbors labelsets contain $l$. Formally, the relative frequency ($rf$) of the label among the set of nearest neighbors $NN_k$ is defined as

$$rf = \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{1} (n_i \in NN_k, y_i = 1)$$

where $y_i$ is the $l$th label of the $i$th nearest neighbor. Each predicted label $z_l$ is defined as

$$z_l = \begin{cases} 1, & \text{if } rf \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

The full algorithms for SLI is shown in Algorithm 1.

### Algorithm 1: Sliding window model (SLI)

```algorithmm
Require: k, m

while stream.hasMoreInstances() do
    s ← stream.nextInstance()
    votes[|] ← {∅}^L.
    for n ∈ window.neighbors(s, k) do
        for label l ∈ L do
            votes[l].add(n.l)
        end for
    end for
    for label l ∈ L do
        z_l ← argmax(votes[l])
    end for
    window.add(s)
    if window.size() > m then
        window.removeOldest()
    end if
end while
```

As a lazy learner, kNN is a good method to test the effectiveness of the techniques we are applying to the stored data. An advantage of the mechanisms presented here is that they act upon the data, and can be applied in many scenarios using different classifiers, which makes them flexible and applicable to different situations.

### Fading window model - FAD

Our first mechanism for responding to concept drift is a fading window model, denoted FAD. Here, each instance is added to the window with weight $w = 1$. With the arrival of each new instance, the weight of each instance is multiplied by a constant fading factor $\delta \in [0, 1]$. Using this simple mechanism, when the prediction for instance $s_t$ is made, the oldest instance in the window, $s_{t-m}$, has the lowest importance and the newest, $s_{t-1}$, has the highest. As with the basic sliding window, when the window is full, the oldest instance is removed. Figure 2 illustrates the fading window, with instance weight depicted using shades of gray.
The prediction for FAD is performed by a \( k \)NN similar to that used with SLI, but here the weights of each instance are taken into consideration using weighted voting, such that

\[
rf = \frac{1}{k} \sum_{i=0}^{k} w_i \cdot 1 \quad (n_i \in NN, \ y_i = 1)
\]

where \( w_i \) is the weight of the \( i \)th nearest neighbor. The fading window mechanisms within ARkNN are shown in Algorithm 2, denoted with ABR.

**Query-based rejuvenation model - QBR**

The query-based rejuvenation model, QBR, implements the first mechanism to rejuvenate instances. For both the sliding window model and the fading window model, an instance will always be removed from the window once \( m \) more recent instances were added. Aging instances out in this manner promotes more recent information and allows a model to adapt to concept drift. However, it is not always the case that the instance that has been in the window longest is the least-valuable instance.

**Pruning instances model - PRU**

Lowering the weight of an instance reduces how much impact that instance has on future predictions. Despite the low weight, however, these instances are still in the window, using resources and potentially acting as nearest neighbors. The pruning instance model, PRU, adds a mechanism to prune the worst performing instances. As each instance arrives, the predicted labelset of the test instance, we conduct a posterior analysis of the contribution of the queried instances. When an instance \( n \) is queried as a nearest neighbor, its labelset \( Y_n \) is checked against the labelset \( Y_s \) of the arriving instance \( s \). For any label \( l \), the neighbor \( n \) is contributing true values if \( y_{n_l} \in Y_n \) equals \( y_{s_l} \in Y_s \), such that the total number of true labels is:

\[
true_n = \sum_{l=0}^{L} 1 \quad (y_{n_l} = y_{s_l}, \ y_{n_l} \in Y_n, \ y_{s_l} \in Y_s)
\]

The accuracy of the neighbor \( n \) is defined as \( \text{accuracy}_n = true_n / L \), where \( L \) is the number of possible labels.

The rejuvenation is computed using the accuracy of the neighbor \( n \) relative to the global accuracy of the window. The latter is computed prequentially as each prediction is made, comparing the predicted labelset to the true labelset. To rejuvenate \( n \), its weight is first updated as follows

\[
\text{weight}_{n^+} = \text{accuracy}_n - \text{accuracy}_{\text{window}}
\]

then set to zero if negative, i.e., \( \text{weight}_n = \max(0, \min(\text{weight}_n, 1)) \), so that \( \text{weight}_n \in [0, 1] \).

In this way, instances that are performing better than the window overall are rejuvenated. Their weight is increased and they will contribute more to future predictions and remain in the window longer. Conversely, instances that are preforming worse that the window overall will age. Their weight will decrease so they have less impact on future predictions. They will be removed from the window if they continue to negatively impact predictions, allowing the learner to respond to concept drift faster. The ABR sections of ARkNN are shown in Algorithm 2, denoted with ABR.
The distance weighting contributions to ARkNN are shown in Algorithm 2, denoted DWV.

**Algorithm 2: Aging and Rejuvenating kNN - ARkNN**

```
Require: k: number of neighbors, m: window size, δ: fading factor, f: fitness threshold
while stream.hasMoreInstances() do
  s ← stream.nextInstance()
  votes[0] ← {0}^L
  true[0] ← {0}^k
  σ[0] ← {0}^k
  for n ∈ window.neighbors(s, k) do
    σ[n] = 2 - distance(n, s)
  end
  for label l ∈ L do
    votes[l].add(n.l + n.w + σ[n])
    if n.l = s.l then true[n] = 1;
    relativeAcc = true[n]/L - windowAcc
    n.w = n.w + relativeAcc; 0 ≤ w ≤ 1
  end
  for label l ∈ L do
    z(l) ← argmax(votes[l])
  end
  windowAcc.update()
  for i ∈ window do
    i.w += δ
    if i.w < f then window.remove(i);
  end
  window.add(s)
  if window.size() > m then
    worst ← argmin(window.getWeight)
    window.remove(worst)
  end
```

Figure 4: Pruned Window.
resulting in fewer distance computations and comparisons when making predictions and training the model, improving the runtime and memory use of the model. Pruning contributions to ARkNN are shown in Algorithm 2, noted as PRU.

**Distance-weighted voting - DWV**

The last mechanism added to ARkNN is distance weighted voting, which is a well-known tactic. When predicting the labelset for each incoming instance s, each vote is weighted not only by the weight of the nearest neighbor, computed as in ABR, but also by the distance of that neighbor from s. Thus, the vote takes into consideration not just the age and performance of the neighbor, but also its similarity to s. The distances between the neighbor n and the incoming instance s is computed as the cosine difference, \( D_C(s, n) = 1 - S_C(s, n) \) where \( S_C \) is the cosine similarity and \( D_C \in [0,2] \). We want to give closer neighbors a higher weight, so the weight is adjusted by a factor of \( 2 - D_C(s, n) \), such that

\[ rf = \frac{1}{k} \sum_{i=0}^{k} w_{n_i} \ast (2 - D_C(s, n)) \ast \mathbb{1} | (n_i \in NN, y_{n_i} = 1) \]

The distance weighting contributions to ARkNN are shown in Algorithm 2, denoted DWV.

**Aging and Rejuvenating kNN - ARkNN**

The final ARkNN algorithm is given in Algorithm 2. Taking mechanisms from each iteration presented in Section 3, ARkNN includes the following mechanisms:

- a fading window reducing the weight of each instance over time by a fading factor \( \delta \) as in FAD.
- accuracy-based rejuvenation of instances based on their accuracy relative to the window accuracy as in ABR.
- distance weighted voting as in DWV.
- instance pruning of all instances with a weight below a fitness threshold \( f \) as in PRU.
- removal of the instance with the lowest weight to maintain a window of size \( m \) as in QBR.

The time-complexity for computing the k-nearest neighbors is \( O(mkd) \) where \( m \) is the maximum size of the window, \( k \) is the number of neighbor used, and \( d \) is the dimensionality of the data. To make a multi-label prediction using the weighted votes from the \( k \) neighbors takes \( O(k|L|) \) time, where \( |L| \) is the number of labels. The combined complexity of ARkNN is \( O(mkd + k|L|) \).

This is the same complexity as the baseline sliding window model (SLM) shown in Algorithm 1. While ARkNN employs multiple mechanisms to improve prediction capabilities and the classifiers ability to react to concept drift, all of these strategies are simple strategies that act on a single window of stored instances and are very resource efficient. Additionally, since these strategies act on the window of instances, they are highly flexible and could easily be incorporated into models using different base classifiers other than kNN.

**4 EXPERIMENTAL SETUP**

This section introduces the experimental setup used to compare the proposed methods with the state of the art. The experiments are designed to answer the following research questions:

- **RQ1:** Does instance aging improve the predictions of the nearest neighbor classifier?
- **RQ2:** Do instance rejuvenation strategies allow us to retain relevant concepts in the stream?
- **RQ3:** Do instance pruning and distance-weighted voting improve the classifier’s predictions?
- **RQ4:** Are the proposed strategies competitive against other state of the art methods, particularly short and long-term memory based classifiers?

**Algorithms.** Table 1 enumerates the strategies proposed and the state of the art algorithms. The source code for all methods is available at https://github.com/canoalberto/ARkNN to facilitate the reproducibility of the experiments. All window-based methods are evaluated with a window size of 1,000 instances.

**Datasets.** Table 2 shows the 37 multi-label datasets evaluated and their properties. These include the number of instances, features, labels, cardinality, and density. The lower the density the more sparse positive labels are in the dataset.
Table 1: Algorithms compared in the experiments.

<table>
<thead>
<tr>
<th>Ref</th>
<th>Acronym</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>This paper</td>
<td>Sliding window</td>
<td>Sliding window</td>
</tr>
<tr>
<td>[27]</td>
<td>MLKNN</td>
<td>Multi-label kNN</td>
</tr>
<tr>
<td>[17]</td>
<td>MLSAMkNN</td>
<td>ML Self-Adjusting Memory kNN</td>
</tr>
<tr>
<td>[18]</td>
<td>MLSAkNN</td>
<td>ML Self-Adjusting kNN</td>
</tr>
<tr>
<td>[22]</td>
<td>OMK</td>
<td>Online Memory k-Means</td>
</tr>
<tr>
<td>[23]</td>
<td>ODM</td>
<td>Online Dual Memory</td>
</tr>
</tbody>
</table>

Table 2: Multi-label datasets and their properties.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Instances</th>
<th>Features</th>
<th>Labels</th>
<th>Cardinality</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>20NG</td>
<td>19,300</td>
<td>1,006</td>
<td>20</td>
<td>1.0289</td>
<td>0.0514</td>
</tr>
<tr>
<td>Bibtex</td>
<td>7,395</td>
<td>1,836</td>
<td>159</td>
<td>2.4019</td>
<td>0.0151</td>
</tr>
<tr>
<td>Birds</td>
<td>645</td>
<td>260</td>
<td>19</td>
<td>1.0140</td>
<td>0.0534</td>
</tr>
<tr>
<td>Bookmarks</td>
<td>87,856</td>
<td>2,150</td>
<td>208</td>
<td>2.0281</td>
<td>0.0098</td>
</tr>
<tr>
<td>CAL500</td>
<td>502</td>
<td>68</td>
<td>174</td>
<td>26.0438</td>
<td>0.1497</td>
</tr>
<tr>
<td>CHD49</td>
<td>555</td>
<td>49</td>
<td>6</td>
<td>2.5802</td>
<td>0.4300</td>
</tr>
<tr>
<td>Corel16k</td>
<td>13,766</td>
<td>500</td>
<td>153</td>
<td>2.8587</td>
<td>0.0187</td>
</tr>
<tr>
<td>Corel5k</td>
<td>5,000</td>
<td>499</td>
<td>374</td>
<td>3.5220</td>
<td>0.0094</td>
</tr>
<tr>
<td>Emotions</td>
<td>593</td>
<td>72</td>
<td>6</td>
<td>1.8685</td>
<td>0.3114</td>
</tr>
<tr>
<td>Enron</td>
<td>1,702</td>
<td>1,001</td>
<td>53</td>
<td>3.3784</td>
<td>0.0837</td>
</tr>
<tr>
<td>Eukaryote</td>
<td>7,766</td>
<td>440</td>
<td>22</td>
<td>1.1456</td>
<td>0.0521</td>
</tr>
<tr>
<td>Eurllex-sm</td>
<td>19,348</td>
<td>5,000</td>
<td>201</td>
<td>2.2133</td>
<td>0.0110</td>
</tr>
<tr>
<td>Flags</td>
<td>194</td>
<td>19</td>
<td>7</td>
<td>3.3918</td>
<td>0.4845</td>
</tr>
<tr>
<td>Genbase</td>
<td>662</td>
<td>1,186</td>
<td>27</td>
<td>1.2523</td>
<td>0.0464</td>
</tr>
<tr>
<td>GnegativePseAAC</td>
<td>1,392</td>
<td>440</td>
<td>8</td>
<td>1.0460</td>
<td>0.1307</td>
</tr>
<tr>
<td>HumanPseAAC</td>
<td>3,106</td>
<td>440</td>
<td>14</td>
<td>1.1851</td>
<td>0.0847</td>
</tr>
<tr>
<td>Hypercube</td>
<td>100,000</td>
<td>100</td>
<td>10</td>
<td>1.0002</td>
<td>0.1000</td>
</tr>
<tr>
<td>Hypersphere</td>
<td>100,000</td>
<td>100</td>
<td>10</td>
<td>2.3138</td>
<td>0.2314</td>
</tr>
<tr>
<td>Imdb</td>
<td>120,919</td>
<td>1,001</td>
<td>28</td>
<td>1.9997</td>
<td>0.0714</td>
</tr>
<tr>
<td>Langlog</td>
<td>1,460</td>
<td>1,004</td>
<td>75</td>
<td>1.1801</td>
<td>0.0157</td>
</tr>
<tr>
<td>Mediamill</td>
<td>43,907</td>
<td>120</td>
<td>101</td>
<td>4.3756</td>
<td>0.0433</td>
</tr>
<tr>
<td>Medical</td>
<td>978</td>
<td>1,449</td>
<td>45</td>
<td>1.2454</td>
<td>0.0277</td>
</tr>
<tr>
<td>Nuswide-Bow</td>
<td>269,648</td>
<td>500</td>
<td>81</td>
<td>1.8685</td>
<td>0.0231</td>
</tr>
<tr>
<td>Nuswide-cVLAD</td>
<td>269,648</td>
<td>128</td>
<td>81</td>
<td>1.8685</td>
<td>0.0231</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>13,929</td>
<td>1,002</td>
<td>23</td>
<td>1.6651</td>
<td>0.0723</td>
</tr>
<tr>
<td>PlantPseAAC</td>
<td>978</td>
<td>440</td>
<td>12</td>
<td>1.0787</td>
<td>0.0899</td>
</tr>
<tr>
<td>Reuters21578</td>
<td>6,000</td>
<td>500</td>
<td>103</td>
<td>1.4622</td>
<td>0.0142</td>
</tr>
<tr>
<td>Scene</td>
<td>2,407</td>
<td>294</td>
<td>6</td>
<td>1.0740</td>
<td>0.1790</td>
</tr>
<tr>
<td>Slashdot</td>
<td>3,782</td>
<td>1,079</td>
<td>22</td>
<td>1.1809</td>
<td>0.0537</td>
</tr>
<tr>
<td>Stacker-chess</td>
<td>1,675</td>
<td>585</td>
<td>227</td>
<td>2.4113</td>
<td>0.0106</td>
</tr>
<tr>
<td>Tmc2007</td>
<td>28,596</td>
<td>500</td>
<td>22</td>
<td>2.2196</td>
<td>0.1009</td>
</tr>
<tr>
<td>VirusGO</td>
<td>207</td>
<td>749</td>
<td>6</td>
<td>1.2174</td>
<td>0.2029</td>
</tr>
<tr>
<td>Water-Quality</td>
<td>1,060</td>
<td>16</td>
<td>14</td>
<td>5.0726</td>
<td>0.3623</td>
</tr>
<tr>
<td>Yahoo-Society</td>
<td>14,512</td>
<td>31,802</td>
<td>27</td>
<td>1.6704</td>
<td>0.0619</td>
</tr>
<tr>
<td>Yahoo-Computers</td>
<td>12,444</td>
<td>34,096</td>
<td>33</td>
<td>1.5072</td>
<td>0.0457</td>
</tr>
<tr>
<td>Yeast</td>
<td>2,417</td>
<td>103</td>
<td>14</td>
<td>4.2371</td>
<td>0.3026</td>
</tr>
<tr>
<td>Yelp</td>
<td>10,806</td>
<td>671</td>
<td>5</td>
<td>1.6383</td>
<td>0.3277</td>
</tr>
</tbody>
</table>

**Metrics.** Dozens of metrics are used to evaluate the performance of multi-label classifiers [6]. The most representative metrics are subset accuracy, accuracy, and F-Measure. Given $n$ instances and $L$ labels, a true labelset $Y_i = \{y_{i1}, \ldots, y_{iL}\}$ and a predicted labelset $Z_i = \{z_{i1}, \ldots, z_{iL}\}$, the example-based metrics are defined as:

\[
\text{Subset accuracy} = \frac{1}{n} \sum_{i=0}^{n} \mathbb{1} \left( |Y_i = Z_i| \right)
\]

\[
\text{Accuracy} = \frac{1}{n} \sum_{i=0}^{n} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}
\]

\[
\text{F-Measure} = \frac{1}{n} \sum_{i=0}^{n} \frac{2 \times |Y_i \cap Z_i|}{|Y_i| + |Z_i|}
\]

Moreover, data stream algorithms are expected to be of low computational and memory complexity. Therefore, we must jointly assess the classification metrics, the runtime (seconds), and the memory consumption (RAM-Hours). All methods are run on an AMD 5950X 16-core CPU with 64 GB RAM and Ubuntu 22.04.

5 RESULTS

Comparison of the proposed techniques

First, we evaluate each of the proposed techniques to analyze their effectiveness, starting with instance aging using a fading window. Table 3 shows the average and the rank for the subset accuracy, accuracy, F-measure, runtime and RAM-hours for each of the six techniques detailed in Section 3 across all 37 evaluated datasets.

To evaluate the impact of instance aging using a fading window, we compare the fading window model (FAD) with the baseline sliding window model (SLI). Experiments show that weighting using a fading window does increase all subset accuracy, accuracy, and F-measure. It has worse runtime and RAM-hours, but the impact here is very minimal. This demonstrates that although the fading mechanism is very simple, it does result in improved predictions at very low cost.

Comparing the results for the query-based rejuvenation model (QBR) to those for FAD, we actually see a decrease in performance. QBR performs worse than FAD, and actually worse than SLI, across all metrics. It is clear that this naive method for rejuvenation is not beneficial and that instances should not be rejuvenated solely for being queried as a nearest neighbor. This reflects the importance of the adaptation to concept drift as the most similar instances in the window do no longer necessarily provide meaningful information to the classifier. Looking at the accuracy-based rejuvenation model (ABR), however, we see that rejuvenation of instances improves the classification performance. Here, instances are rejuvenated based on the accuracy of their contributions to predictions. Across subset accuracy, accuracy and F-measure, ABR consistently performs better than FAD. Therefore, instances and concepts strengthened by the accuracy-based rejuvenation strategy are relevant and beneficial to the classification model.

Looking at subset accuracy, accuracy and F-measure for the pruning model (PRU) and the distance-weighted voting model (DWV) we also see incremental improvements in predicting. Pruning all instances below a fitness threshold provides slight gains in these metrics and distance-weighted voting still more. Interestingly, the average runtime and RAM-hours of PRU are the worst of the six models, we don’t see improvements in resource use until we add in the distance-weighted voting. This demonstrates the inter-connectivity of the strategies. The aging, rejuvenation and distance-weighting strategies improve the accuracy of the classification
Table 3: Comparison of the proposed aging and rejuvenating strategies over the average performance on 37 multi-label datasets. The last row averages the ranks across all metrics.

<table>
<thead>
<tr>
<th>Avg. Perf.</th>
<th>SLI</th>
<th>FAD</th>
<th>QBR</th>
<th>ABR</th>
<th>PRU</th>
<th>DWV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset acc.</td>
<td>0.2790</td>
<td>0.2943</td>
<td>0.2642</td>
<td>0.3221</td>
<td>0.3345</td>
<td><strong>0.3353</strong></td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.3904</td>
<td>0.4138</td>
<td>0.3724</td>
<td>0.4364</td>
<td>0.4477</td>
<td><strong>0.4486</strong></td>
</tr>
<tr>
<td>F-Measure</td>
<td>0.4817</td>
<td>0.4963</td>
<td>0.4624</td>
<td>0.5173</td>
<td>0.5275</td>
<td><strong>0.5285</strong></td>
</tr>
<tr>
<td>Runtime</td>
<td><strong>0.0740</strong></td>
<td>0.0745</td>
<td>0.0755</td>
<td>0.0760</td>
<td>0.0798</td>
<td>0.0757</td>
</tr>
<tr>
<td>RAM-Hours</td>
<td>6.13E-3</td>
<td>6.51E-3</td>
<td>7.17E-3</td>
<td>7.79E-3</td>
<td>8.55E-3</td>
<td>6.87E-3</td>
</tr>
</tbody>
</table>

Table 4: Comparison with other Multi-Label kNN classifiers

Looking at just the predictive performance (subset accuracy, accuracy, and F-measure), ARkNN does not achieve the best results. AESAKNNS has the highest average for all three metrics and the highest rank for two of the three. This is mainly because AESAKNNS is an ensemble method. However, ARkNN is a close contender. When looking at the subset accuracy, accuracy and F-measure, only AESAKNNS and ODM consistently outperform ARkNN and the differences are small. Looking at the runtime and memory use, ARkNN is clearly superior. The average runtime for ARkNN is 1/5th of the average runtime of AESAKNNS, the next fastest competitor. In addition, the ranks show that AESAKNNS is on average the second slowest algorithm, whereas ARkNN is consistently fast, on average ranking as the fastest. Similarly, the average RAM-hours for ARkNN are 1/50th of the RAM-hours for OMK, and ARkNN also achieves that best rank for RAM-hours. Looking at the overall average rank across all five metrics, ARkNN clearly out-performs the other methods. The Nemenyi critical value is 1.7261 for \( \alpha = 0.05 \) indicating statistical differences between ARkNN and all methods expect MLSAkNN and ODM for the average rank.

This is more clearly shown in Figure 5. The height of each arc indicates the average rank across all datasets achieved by each algorithm. A greater height indicates a higher rank and better performance. Looking at the central circle of wedges indicating runtime, it clear that MLkNN, with its complex computation of prior and posterior probabilities, has the worst runtime and that ARkNN, with its simple instance-based strategies, is the fastest. Combined with the pink arc indicating memory use, it is clear that ARkNN is substantially more resource efficient than its competitors resulting in the best combined performance.

The complex strategies used by state-of-the-art algorithms to improve predictions and respond to concept drift regularly come at a steep cost in terms of resource use. MLSAMkNN, OMK and ODM all use dual short- and long-term memories. MLSAMPkNN and MLSAkNN use a resource-heavy self-adjusting window. AESAKNNS is an ensemble method that uses a collection of ADWIN detectors. These are all state-of-the-art techniques and AESAKNNS and ODM in particular have excellent predictive power. When classifying streaming multi-label data, however, resource use is important.

Table 4: Comparison of the Aging and Rejuvenating kNN (ARkNN) with other multi-label kNN-based classifiers over the average performance on 37 multi-label datasets. The last row averages the ranks across all metrics.

<table>
<thead>
<tr>
<th>Avg. Perf.</th>
<th>MLkNN</th>
<th>MLSAMkNN</th>
<th>MLSAMPkNN</th>
<th>MLSAkNN</th>
<th>AESAKNNS</th>
<th>OMK</th>
<th>ODM</th>
<th>ARkNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset acc.</td>
<td>0.2171</td>
<td>0.2860</td>
<td>0.3074</td>
<td>0.3159</td>
<td><strong>0.3404</strong></td>
<td>0.2501</td>
<td>0.3477</td>
<td>0.3353</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.3023</td>
<td>0.3850</td>
<td>0.4106</td>
<td>0.4400</td>
<td><strong>0.4618</strong></td>
<td>0.3609</td>
<td>0.4565</td>
<td>0.4486</td>
</tr>
<tr>
<td>F-Measure</td>
<td>0.3824</td>
<td>0.4687</td>
<td>0.4920</td>
<td>0.5307</td>
<td><strong>0.5539</strong></td>
<td>0.4418</td>
<td>0.5351</td>
<td>0.5285</td>
</tr>
<tr>
<td>Runtime</td>
<td>3.2822</td>
<td>5.6077</td>
<td>0.5213</td>
<td>0.4533</td>
<td>0.3503</td>
<td>0.5167</td>
<td>0.4086</td>
<td><strong>0.0757</strong></td>
</tr>
<tr>
<td>RAM-Hours</td>
<td>9.67E-1</td>
<td>3.69E+0</td>
<td>4.09E+0</td>
<td>1.26E+0</td>
<td>6.14E-1</td>
<td>3.47E-1</td>
<td>3.52E-1</td>
<td><strong>6.87E-3</strong></td>
</tr>
</tbody>
</table>

Looking at the subset accuracy, accuracy, F-measure, and the corresponding ranks for the subset accuracy, accuracy, F-measure, runtime, and RAM-hours for ARkNN and each of the seven compared algorithms across all 37 datasets.

Table 4: Comparison of the Aging and Rejuvenating kNN (ARkNN) with other multi-label kNN-based classifiers over the average performance on 37 multi-label datasets. The last row averages the ranks across all metrics.

<table>
<thead>
<tr>
<th>Avg. Perf.</th>
<th>SLI</th>
<th>FAD</th>
<th>QBR</th>
<th>ABR</th>
<th>PRU</th>
<th>DWV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset acc.</td>
<td>6.7162</td>
<td>5.2568</td>
<td>4.4730</td>
<td>4.0676</td>
<td>3.6486</td>
<td>5.1486</td>
</tr>
<tr>
<td>Runtime</td>
<td>7.5946</td>
<td>5.0000</td>
<td>4.5405</td>
<td>4.6351</td>
<td>5.5135</td>
<td>4.0135</td>
</tr>
<tr>
<td>RAM-Hours</td>
<td>6.8000</td>
<td>4.9429</td>
<td>5.1143</td>
<td>5.0571</td>
<td>6.6571</td>
<td>3.5714</td>
</tr>
</tbody>
</table>

As shown in Table 4, ARkNN is clearly superior. The average runtime for ARkNN is 1/5th of the average runtime of AESAKNNS, the next fastest competitor. In addition, the ranks show that AESAKNNS is on average the second slowest algorithm, whereas ARkNN is consistently fast, on average ranking as the fastest. Similarly, the average RAM-hours for ARkNN are 1/50th of the RAM-hours for OMK, and ARkNN also achieves that best rank for RAM-hours. Looking at the overall average rank across all five metrics, ARkNN clearly out-performs the other methods. The Nemenyi critical value is 1.7261 for \( \alpha = 0.05 \) indicating statistical differences between ARkNN and all methods expect MLSAkNN and ODM for the average rank.

This is more clearly shown in Figure 5. The height of each arc indicates the average rank across all datasets achieved by each algorithm. A greater height indicates a higher rank and better performance. Looking at the central circle of wedges indicating runtime, it clear that MLkNN, with its complex computation of prior and posterior probabilities, has the worst runtime and that ARkNN, with its simple instance-based strategies, is the fastest. Combined with the pink arc indicating memory use, it is clear that ARkNN is substantially more resource efficient than its competitors resulting in the best combined performance.

The complex strategies used by state-of-the-art algorithms to improve predictions and respond to concept drift regularly come at a steep cost in terms of resource use. MLSAMkNN, OMK and ODM all use dual short- and long-term memories. MLSAMPkNN and MLSAkNN use a resource-heavy self-adjusting window. AESAKNNS is an ensemble method that uses a collection of ADWIN detectors. These are all state-of-the-art techniques and AESAKNNS and ODM in particular have excellent predictive power. When classifying streaming multi-label data, however, resource use is important.

As shown in Table 4, ARkNN is clearly superior. The average runtime for ARkNN is 1/5th of the average runtime of AESAKNNS, the next fastest competitor. In addition, the ranks show that AESAKNNS is on average the second slowest algorithm, whereas ARkNN is consistently fast, on average ranking as the fastest. Similarly, the average RAM-hours for ARkNN are 1/50th of the RAM-hours for OMK, and ARkNN also achieves that best rank for RAM-hours. Looking at the overall average rank across all five metrics, ARkNN clearly out-performs the other methods. The Nemenyi critical value is 1.7261 for \( \alpha = 0.05 \) indicating statistical differences between ARkNN and all methods expect MLSAkNN and ODM for the average rank.

As shown in Table 4, ARkNN is clearly superior. The average runtime for ARkNN is 1/5th of the average runtime of AESAKNNS, the next fastest competitor. In addition, the ranks show that AESAKNNS is on average the second slowest algorithm, whereas ARkNN is consistently fast, on average ranking as the fastest. Similarly, the average RAM-hours for ARkNN are 1/50th of the RAM-hours for OMK, and ARkNN also achieves that best rank for RAM-hours. Looking at the overall average rank across all five metrics, ARkNN clearly out-performs the other methods. The Nemenyi critical value is 1.7261 for \( \alpha = 0.05 \) indicating statistical differences between ARkNN and all methods expect MLSAkNN and ODM for the average rank.

As shown in Table 4, ARkNN is clearly superior. The average runtime for ARkNN is 1/5th of the average runtime of AESAKNNS, the next fastest competitor. In addition, the ranks show that AESAKNNS is on average the second slowest algorithm, whereas ARkNN is consistently fast, on average ranking as the fastest. Similarly, the average RAM-hours for ARkNN are 1/50th of the RAM-hours for OMK, and ARkNN also achieves that best rank for RAM-hours. Looking at the overall average rank across all five metrics, ARkNN clearly out-performs the other methods. The Nemenyi critical value is 1.7261 for \( \alpha = 0.05 \) indicating statistical differences between ARkNN and all methods expect MLSAkNN and ODM for the average rank.
a real-world scenario it is unrealistic to assume that computing resources are infinite, and the streaming nature of the data requires that models can make predictions and update as quickly as the data is arriving. Our experiments demonstrate that ARkNN, using a collection of simple and highly resource-efficient techniques, can achieve predictive power competitive with state-of-the-art multi-label streaming algorithms using much fewer resources.

6 CONCLUSIONS

We have proposed ARkNN, a method using a combination of simple, resource efficient techniques to weight instances within a single updating window to classify streaming, multi-label data. ARkNN gives greater importance to neighbors that are closer, newer and better performing, allowing it to respond to concept drift and quickly devalue and remove older concepts and noisy data, while retaining older information that consistently benefits the model. We have shown experimentally that each of the mechanisms used by ARkNN improves the predictive performance of the algorithm, without increasing its runtime or memory use. In a comprehensive study using real world data, we have shown that ARkNN has predictive capabilities that exceed many state-of-the-art methods and are closely comparable even to the most highly accurate algorithms. ARkNN achieves this predictive performance while using fewer resources than any of the compared methods. When evaluating runtime and memory use, as well as predictive performance, ARkNN outperforms all compared algorithms, making it an excellent choice for mining multi-label data streams, where resources are limited. ARkNN shows that it is possible to achieve excellent predictive power using methods that are resource efficient. Future works will also investigate the dynamic relevance of attributes (e.g. Relief family) in combination with dynamic instance weighting.

REFERENCES