D4.2 Basic Scalable Streaming Algorithms

Lead Author: Hamid Bouchachia
With contributions from: Waqas Jamil, Wenjuan Wang
Reviewer: [Expert chosen by the responsible for the deliverable]

<table>
<thead>
<tr>
<th>Deliverable nature:</th>
<th>Report (R) + Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level: (Confidentiality)</td>
<td>Public (PU)</td>
</tr>
<tr>
<td>Contractual delivery date:</td>
<td>November 30\textsuperscript{th} 2016</td>
</tr>
<tr>
<td>Actual delivery date:</td>
<td>November 30\textsuperscript{th} 2016</td>
</tr>
<tr>
<td>Version:</td>
<td>0.5</td>
</tr>
<tr>
<td>Total number of pages:</td>
<td>25</td>
</tr>
<tr>
<td>Keywords:</td>
<td>Basic online and streaming algorithms, preprocessing, Reservoir sampling, frequent directions, principal components analysis, singular value decomposition, random projection, moving average, aggregation algorithm.</td>
</tr>
</tbody>
</table>
Abstract

The present report describes a set of selected algorithms for basic processing of big data, in particular for data streams. They pertain to different classes of techniques: data sampling, feature reduction, compression and various statistical moments. The proposed algorithms are basic ones that can be used for various analytics purposes (classification, clustering, regression). They can be used online in real-time and can be implemented on a distributed platform to meet the scalability requirements.

Each class includes a number of algorithms. In particular, the report explains the purpose, the algorithmic steps and the distributed implementation of each algorithm.
Executive summary

This report describes the first version of SOLMA, the library of scalable streaming algorithms for predictive analytics and automatic knowledge discovery from big data. This version is expected to include basic stream sketches that enable to query the stream (statistic moments, heavy hitters, sampling, and feature reduction) anytime. The current state-of-the-art streaming algorithms for big data do not offer such diverse basic algorithms that will potentially represent routines/utilities in the library.

The report presents in particular a set of algorithms that can be categorized into the following:

- Moments: 7 basic as well as advanced routines are proposed: simple mean, simple variance, weighted mean, weighted variance, exponentially weighted mean and variance, moving average, aggregation algorithm.
- Sampling: 3 stream sampling algorithms are proposed. All of them are based on the popular reservoir sampling.
- Heavy hitters: one algorithm, the frequent directions algorithm, is implemented
- Feature reduction: 3 algorithms are presented: principal analysis, singular value decomposition and random projection

All algorithms are described in an accessible way providing details about:

- Purpose of the algorithm
- Algorithmic steps
- Distributed implementation

Currently we are still investigating matrix sketching, online SVD, random projection ensemble classification and random projection ensemble clustering for data streams. SOLMA will be even richer in terms of basic scalable streaming algorithms.
Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>Acronym</th>
<th>PROTEUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>687691</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Full Title</th>
<th>Scalable online machine learning for predictive analytics and real-time interactive visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project URL</td>
<td><a href="http://www.proteus-bigdata.com/">http://www.proteus-bigdata.com/</a></td>
</tr>
<tr>
<td>EU Project Officer</td>
<td>Martina EYDNER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4.2</td>
<td>Basic scalable streaming algorithms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work Package Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M012</td>
<td>M12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>version 1.0</th>
<th>final</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>report ☑demonstrator □ other ☑</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>public ☑ restricted □</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
<th>BU</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Responsible Author</th>
<th>Name</th>
<th>E-mail</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hamid Bouchachia</td>
<td><a href="mailto:abouchachia@bournemouth.ac.uk">abouchachia@bournemouth.ac.uk</a></td>
<td>+44 1202 96 24 01</td>
</tr>
</tbody>
</table>

Abstract (for dissemination)

The present report describes a set of selected algorithms for basic processing of big data, in particular for data streams. They pertain to different classes of techniques: data sampling, feature reduction, compression and various statistical moments. The proposed algorithms are basic ones that can be used for various analytics purposes (classification, clustering, regression). They can be used online in real-time and can be implemented on a distributed platform to meet the scalability requirements.

Each class includes a number of algorithms. In particular, the report explains the purpose, the algorithmic steps and the distributed implementation of each algorithm.

Keywords

Reservoir sampling, frequent directions, principal components analysis, singular value decomposition, random projection, moving average, aggregation algorithm.

Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev. No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 10th, 2016</td>
<td>V.0.0</td>
<td>H. Bouchachia</td>
<td>Structure of the document</td>
</tr>
<tr>
<td>October, 25th 2016</td>
<td>V.3</td>
<td>H Bouchachia, W. Jamil, Wenjuan Wang</td>
<td>Initial draft</td>
</tr>
<tr>
<td>Date: November, 14th. 2016</td>
<td>Version: V.5</td>
<td>Authors: H. Bouchachia, W. Jamil, Wenjuan Wang</td>
<td>Stage: First full draft</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Date: November, 24th. 2016</td>
<td>Version: V.6</td>
<td>Authors: Tao Cao</td>
<td>Stage: Review comments</td>
</tr>
<tr>
<td>Date: November, 25th. 2016</td>
<td>Version: V.7</td>
<td>Authors: H. Bouchachia</td>
<td>Stage: Final version</td>
</tr>
</tbody>
</table>
# Table of Contents

Executive summary ........................................................................................................ 4
Document Information .................................................................................................... 5
Table of Contents ........................................................................................................... 7
List of Algorithms ......................................................................................................... 8
List of Figures ................................................................................................................ 9
Abbreviations ................................................................................................................. 10

1. Introduction ................................................................................................................ 11
   1.1. Document objectives ............................................................................................. 12
   1.2. Document structure .............................................................................................. 12

2. Reservoir sampling .................................................................................................... 13
   2.1. Reservoir Sampling .............................................................................................. 13
   2.2. Adaptive Reservoir Sampling ............................................................................. 14
   2.3. Weighted Reservoir Sampling ............................................................................. 14
   2.4. Distributed Reservoir Sampling .......................................................................... 15

3. Frequent directions .................................................................................................... 17

4. Moments ....................................................................................................................... 19
   4.1. Simple Mean ......................................................................................................... 19
   4.2. Simple Variance ................................................................................................... 19
   4.3. Weighted Mean .................................................................................................... 20
   4.4. Weighted Variance .............................................................................................. 20
   4.5. Exponentially Weighted Mean and Variance ...................................................... 21
   4.6. Moving Average .................................................................................................. 21
   4.7. Aggregation Algorithm ....................................................................................... 22

5. Feature Reduction ....................................................................................................... 24
   5.1. Online PCA ........................................................................................................... 24
   5.2. Singular Value Decomposition ............................................................................ 26
   5.3. Random Projection .............................................................................................. 27

6. Conclusions ................................................................................................................ 28

References ....................................................................................................................... 29
List of Algorithms

Algorithm 1: Reservoir sampling ................................................................. 13
Algorithm 2: Adaptive reservoir sampling .................................................. 14
Algorithm 3: Weighted random sampling (A-RES) ....................................... 15
Algorithm 4: Weighted random sampling (A-Chao) ...................................... 15
Algorithm 5: Frequent directions ............................................................... 17
Algorithm 6: Moving average (Innovation Algorithm) ................................. 21
Algorithm 7: Aggregation algorithm ........................................................... 23
Algorithm 8: Online PCA ......................................................................... 24
Algorithm 9: Combining eigenspace models .............................................. 25
List of Figures

Figure 1: 2-stage distributed reservoir sampling .......................................................... 16
Figure 2: 1-stage distributed reservoir sampling .......................................................... 16
Figure 3: Distributed FD .............................................................................................. 18
Figure 4: Distributed version of the aggregation algorithm .............................................. 23
Figure 5: Distributed version of OPCA ......................................................................... 25
Abbreviations

**AA:** Aggregation algorithm
**ARMA:** Autoregressive moving average  
**ARIMA:** Autoregressive integrated moving average  
**FD:** Frequent directions  
**MA:** Moving average  
**ML:** Machine Learning  
**PCA:** Principal components analysis  
**RP:** Random projection  
**RS:** Reservoir sampling  
**SOLMA:** Scalable Online Machine Learning and Data Mining Algorithms  
**SVD:** Single value decomposition
1. Introduction

Although, online learning algorithms are tightly related to primitives that operate in an incremental way accommodating data streams, there are not many machine libraries that offer such primitives.

Sketching technique is an appealing technique that allows producing summaries of the streaming data. Sketching is relevant to different tasks such as sampling, histograms, multi-resolution models (wavelets, transformations) and frequent items (itemsets, patterns). Transformations are often useful for other types of tasks such as feature reduction and reduction. Many known sketches are linear (based on some linear transformations) are: frequent items, norms, quantiles, histograms, random subset sums, different counting sketches, Bloom filters, etc.

Sketching is used to compute different types of frequency statistics [1, 11]. Such statistics are designed to provide inherent characteristics of the data. They may take the form of summaries that serve to approximate the information content of the data. The performance of some standard sketches algorithms using hashing has been reviewed in [31].

The main challenge for parallel computation is the size of the data, that is, when it is large and of the same order of magnitude as the time series, may lead the computation may be quadratic in the size of the series.

More advanced sketching techniques are those we encounter in typical machine learning algorithms and these are the ones we have considered and we will further investigate along the lifetime of PROTEUS. Sketches are useful for online machine learning algorithms as they allow computing the main elements of such algorithms in a recursive manner, thus avoiding storing and revisiting any data in the future.

Indeed, Online learning (OL) especially for data streams takes place over long periods of time, and is inherently open-ended. The aim is to ensure that the system remains amenable to refinement as long as data continue to arrive. It is interesting to note that online learning can also deal with applications starving of data (e.g., experiments that are expensive and slow to produce data as in some chemical and biological applications) as well as with applications that are data intensive (e.g., monitoring, information filtering, etc.).

OL faces the challenge of accurately estimating the statistical characteristics of data in the future. In non-stationary changing environments, the challenge becomes even more important, since the system’s behaviour may need to change drastically over time due to concept drift. The aim of OL is to ensure continuous adaptation, while storing only the learning model that will be used as basis in the future learning steps. As new data arrive, new memories may be created and existing ones may be modified allowing the system to evolve over time. For these reasons, sketches and summaries are quite appealing to consider as part of any online machine learning library like SOLMA.

In this document we report on a number of classes of techniques: sampling, moments, matrix sketching and feature reduction. Specifically, for the first class we will present discuss 3 algorithms based on the reservoir sampling technique. We also provide the implementation of 7 standard online moments. The third class includes one novel technique, called Frequent Directions (FD) a kind of heavy hitters. For the last class, we focus on online principal component analysis (OPCA) and singular value decomposition (SVD) and random projection. We show also how these basic algorithms can be implemented on a distributed platform. These algorithms are accessible from the project GitHub website: https://github.com/proteus-h2020/SOLMA. It is worthwhile to mention that other algorithms will be added during the execution of PROTEUS.
1.1. **Document objectives**

This document provides a brief description of basic scalable streaming algorithms that will be integrated in the SOLMA library. In particular we will describe and provide the generic steps of some of the selected algorithms: Online sampling, online FD, online moments, online PCA, offline SVD and online random projection.

1.2. **Document structure**

The document consists mainly of 4 sections. Each section describes a set of selected algorithms from one of the classes: sampling, moments, heavy hitters, and feature reduction.
2. Reservoir sampling

Sampling is an important technique for performing many approximation tasks such as answering queries or developing machine learning models from a finite set of data input. It aims to derive a sample that can represent the whole population [6]. Random sampling is a basic sampling scheme. The principle is to have a same possibility for each stream item to be selected into the sample. It reduces human bias potential and obtains a sample that can highly represent the population. Assuming we have a set of size $n$, random sampling is to select without replacement a sample of size $k$. Many algorithms have been developed to solve problems with a known total size $n$ [8, 26].

However, when it comes to data streams, the size $n$ is unknown beforehand. Thus, the sampling rate cannot be determined. Besides, sampling should be processed sequentially since the items arrive in stream. The most classical approach is reservoir sampling [19, 22, 27]. With this algorithm, the probability of each item selected into a fixed-size reservoir is equal. The algorithm maintains a random sample of size $s$ without replacement over a stream. It is initialized with the first $s$ elements; when the $i$-th element arrives for $i > s$, with probability $1/i$ the model adds the new element, and replaces replacing an element uniformly chosen from the current sample. There have been various extensions to the basic reservoir sampling algorithm.

It has been applied in many applications, for example, clustering [18, 12], spatial data management [25], etc. However, there are applications that need to adjust the reservoir size [2]. In this case, adaptive-size reservoir sampling can be applied. In some other applications, the stream items are assigned weights. Two weighted reservoir sampling algorithms [7, 5] are proposed for this situation.

In this project, reservoir sampling, adaptive reservoir sampling and two weighted reservoir sampling algorithms are implemented. In the following a short description of each algorithm is given.

2.1. Reservoir Sampling

The algorithm selects a random sample with a fixed size $k$ without replacement from a data stream of an unknown size $n$. Initially, it places the first $k$ items from the stream into the reservoir. Then, it iterates with each arriving item until the steam is exhausted. For the $i$-th item $S_i$, the algorithm generates a random number $j$ from 0 to $i$. If $j$ is less than or equal to $k$, the $j$-th item in the reservoir is replaced with the $S_i$. The probability of any item shown in the final reservoir is equal, i.e. $k/n$. The time complexity of reservoir sampling is $O(n)$. The algorithm is presented as follows:

Algorithm 1 Reservoir sampling

1: Input the reservoir size $k$
2: $i = 0$
3: for arriving stream item $S_i$ do
4: $i = i + 1$
5: if $i \leq k$ then
6: Add item $S_i$ into the reservoir
7: else
8: Generate a random number $j$ from 0 to $i$. if $j < k$, replace the $j$-th item in reservoir with $S_i$.
9: end if
10: end for
2.2. Adaptive Reservoir Sampling

With reservoir sampling, one obtains a fixed size sample. However, it is better to adjust the reservoir size in the middle of sampling in some applications; for instance, data collection over wireless sensor networks, approximate query processing, etc. [2] proposed an algorithm called adaptive reservoir sampling which maintains the reservoir sample after the size is adjusted. It is proven that, when the reservoir size decreases, the algorithm generates a sample in the reduced reservoir with a 100% uniformity confidence (UC), defined in [2]. This means each item in the reduced reservoir has an equal probability of being selected from the stream. In contrast, when the reservoir size is increased, the enlarged reservoir cannot be maintained with a 100% uniformity confidence.

The adaptive reservoir sampling algorithm is shown below. If the reservoir size does not change, reservoir sampling is used. If the reservoir size decreases by $\delta$, the algorithm discards $\delta$ items from the original reservoir and then continues. In contrast, if the reservoir size increases by $\delta$, the algorithm computes the minimum value of $m$ (defined as the number of incoming items used to fill the enlarged reservoir) that causes the uniformity confidence to exceed a given threshold $\zeta$. Afterwards, it flips a biased coin to decide on how many items $x$ are retained among the $k$ items in the original reservoir. $k$-$x$ items are randomly discarded from the original reservoir. The enlarged reservoir is refilled with $k+\delta$-$x$ items from the arriving $m$ items.

Algorithm 2: Adaptive reservoir sampling

1. Input $k$ the reservoir size, $i$ the number of stream items seen so far, $\zeta$ uniformity confidence threshold, $m$ an integer
2. while true do
3.   while reservoir size does not change do
4.     reservoir sampling (Algorithm 1)
5.   end while
6.   if reservoir size is decreased by $\delta$ then
7.     randomly discard $\delta$ items from the reservoir
8.   else
9.     find the minimum value of $m$ that causes the UC to exceed $\zeta$.
10.    flip a biased coin to decide on the number of items, $x$, to be retained among $k$ items already in the reservoir.
11.   randomly evict $k$-$x$ items from the reservoir.
12.   select $k+\delta$-$x$ items from the incoming $m$ items using reservoir sampling (Algorithm 1)
13. end if
14. end while

2.3. Weighted Reservoir Sampling

Weighted random sampling is used in cases where items are assigned with weights. The probability of each item being selected is determined by its weight. There are at least two ways to interpret naturally the item weights. The first interpretation is that the probability of being selected is determined by the weight of each item. The other one is that the probability of the item being in the final sample is determined by the relative weight of each item.

In the case of data streams, there are algorithms for both interpretations. Algorithm 3 proposed by in [7] applies the first interpretation. It is given as follows:
Algorithm 3: Weighted random sampling: Algorithm A-Res

1: Input $V$ a set of $n$ weighted items, $m$ the size of reservoir $R$, and $w_i$ the weight of stream item $S_i$
2: Keep the first $m$ items in the reservoir
3: For each item in the reservoir, calculate a key $k_i = u_i^{1/w_i}$, where $u_i = \text{random}(0, 1)$
4: for $i = m+1, m+2, \ldots, n$ do
5: \hspace{1em} put $T$ as the smallest key in $R$
6: \hspace{1em} for $S_i$, calculate a key $k_i = u_i^{1/w_i}$
7: \hspace{1em} if the key $k_i$ is larger than $T$ then
8: \hspace{2em} the minimum key item in $R$ is replaced by $S_i$
9: \hspace{1em} end if
10: end for

The key $k_i$ of stream item $S_i$ in the population is calculated as $k_i = u_i^{1/w_i}$ with a uniform random number $u_i = \text{random}(0,1)$. Firstly, the algorithm keeps the first $m$ items in the reservoir and calculates their key. If the key of the new arriving item is larger than the minimum key in the reservoir, the minimum key item is replaced by the new arriving one. This step is repeated until the data stream is exhausted.

For the second weight interpretation, Chao [5] proposed an algorithm called Algorithm A-Chao. Initially, it fills the reservoir with the first $m$ stream items. Then it calculates the relative weight of the new arriving item. This value is used to randomly decide if a uniformly selected item in the reservoir should be replaced by this new item. The algorithm is shown in Algorithm 4.

Algorithm 4: Weighted random sampling: Algorithm A-Chao

1: Input $V$ a set of $n$ weighted items, $m$ the size of reservoir $R$, and $w_i$ the weight of stream item $S_i$
2: Keep the first $m$ items in the reservoir
3: for $m < i < n$ do
4: \hspace{1em} calculate the probability $p_i = w_i/\left(\sum_{k=1}^{i} w_k\right)$ for the $S_i$
5: \hspace{1em} decide randomly if $S_i$ will be inserted into the reservoir
6: \hspace{1em} if true then
7: \hspace{2em} choose uniformly a random item from the reservoir and replace it with $S_i$
8: \hspace{1em} else
9: \hspace{2em} simply increase the total weight
10: \hspace{1em} end if
11: end for

2.4. Distributed Reservoir Sampling

To cope with high speed streams, a distributed approach needs to be taken. One natural way of implementing distributed stream sampling algorithms is a kind of stratification [2]. Sub-samples are computed on different distributed machines before they are combined at the level of a master
machine. This is the approach we are following in this research. The stream is processed window by window, where each window is sampled by a machine using a selected reservoir sampling algorithm. The outcome on each machine is sent to the master machine which applies reservoir sampling to produce a final sample. Another possibility consists of simply merging the output reservoirs, but this solution is not scalable. Figures 1 and 2 show both possibilities.

**Figure 1: 2-stage distributed reservoir sampling**

**Figure 2: 1-stage distributed reservoir sampling**
3. **Frequent directions**

Low rank approximations for large matrices are used in different data mining tasks such as Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), and k-means clustering [10]. There are very few techniques based on sketching to implement low rank approximation for streaming data (assuming that data is seen as a growing matrix). One new technique used for low-rank approximation is **Frequent Directions (FD)**.

The Frequent Directions algorithm is an extension of the Misra-Gries Frequent Items algorithm [23] for estimating counts of items in streaming data. To show the connection, we begin by briefly reviewing the Frequent Items algorithm before describing the Frequent Directions algorithm. Frequent Directions is a conceptually-simple, deterministic algorithm that is optimal with respect to sketch size and resulting accuracy (but not to run time). The algorithm is a deterministic algorithm (row/column update) which outperforms other available options in terms of space-error trade-off, for results see [10].

The goal of FD is to sketch a matrix $B$ that is significantly smaller than the original $A$ while this later is continuously updated with new data items. That is, given an arbitrary input matrix, $A \in \mathbb{R}^{n \times d}$, one row at a time; FD maintains a sketch matrix $B \in \mathbb{R}^{k \times d}$ such that $k < n$. A good sketch matrix $B$ is such that $A \approx B$ or equivalently $AA^T \approx BB^T$. Using such sketch, many operations on matrices can be efficiently computed. The FD algorithm achieves this goal by the guarantee: $||AA^T - BB^T|| \leq \epsilon ||A||_F^2$. The proof is unsurprisingly very similar to the frequent items proof. There are various implementations available of this algorithm; the one we implemented is as follows:

**Algorithm 5 Frequent Directions**

Input $l$, $A \in \mathbb{R}^{n \times m}$

$B \leftarrow \text{zero-matrix} \in \mathbb{R}^{l \times m}$, $A \in \mathbb{R}^{n \times m}$

For $t = 1, 2, \ldots n$ do

- Insert $A_i$ into a zero valued row of $B$ if $B$ has no zero valued rows then
  - $(U, \Sigma, V) \leftarrow \text{SVD}(B)$
  - $\delta \leftarrow \sigma_3^2$
  - $\Sigma \leftarrow \sqrt{\max(\Sigma^2 - \|I\delta, 0)}$
  - $B \leftarrow \Sigma V^T$

End if

Return: $B$

This algorithm has room for improvement, in terms of time and storage space. Most of the time is taken by the Singular-Value-Decomposition (SVD) which is calculated once every iteration and therefore the total running time is bounded by $O(ml)$. This gives an amortized update time of $O(ml)$ per row.

In order to implement FD on a distributed platform, as described in [10], the input can be distributed among several machines, where each machine produces a summary. The FD outcome of all machines can be then combined in a straightforward way. For an input $A = [A_1; A_2; \ldots; A_p]$, where $A_j$ is a sequence of input (batch) and without loss of generality let $B_j$ be the FD outcome of
Aj. Then thanks to the property of mergeable summary [Agarwal et al., 2013], the output is simply the combination \( B = \{ B_1; B_2; \ldots; B_p \} \).

Figure 3: Distributed FD
4. Moments

Basic moments for streaming are provided. These can be used when developing online algorithms or simply showing basic statistics of the data flow. The code is available at: https://github.com/proteus-h2020/proteus-backend/blob/master/proteus-examples/src/main/java/com/treelogic/proteus/examples/AverageExample.java. This will be however enriched with further moments during the course of the project.

4.1. Simple Mean

The mean of \( n \) data points is given as:

\[
\mu_n = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{n} \left( x_n + \sum_{i=1}^{n-1} x_i \right)
\]  

(1)

By simple manipulation we can compute the mean recursively to obtain:

\[
\mu_n = \frac{n-1}{n} \mu_{n-1} + \frac{1}{n} x_n
\]  

(2)

To compute the mean in parallel, given two datasets A and B whose means are computed on two machines, the following formula can be used to compute the overall mean:

\[
\mu = \frac{n_A}{n_A+n_B} \mu_A + \frac{n_B}{n_A+n_B} \mu_B
\]  

(3)

4.2. Simple Variance

The variance of \( n \) data points is given by

\[
\sigma_n^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_n)^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i^2 - 2x_i \mu_n + \mu_n^2)
\]  

(4)

Using basic algebraic manipulation we get:

\[
\sigma_n^2 = \left( \frac{1}{n} \sum_{i=1}^{n} x_i^2 \right) - \left( \frac{1}{n} \sum_{i=1}^{n} x_i \right)^2
\]  

(5)

It is easy to show that

\[
\sigma_n^2 = \frac{n-1}{n} \sigma_{n-1}^2 + \frac{1}{n-1} (x_n - \mu_n)^2
\]  

(6)

To implement this in parallel given two datasets A and B, we use the quantity \( M_A = \sum_{x_i \in A} (x_i - \mu A)^2 \) s.t. \( \sigma n^2 = 1n - 1MA \), then:

\[
M_{AUB} = M_A + M_B + (\mu_A - \mu_B)^2 n_A n_B / n
\]  

(7)

Similarly, the covariance can be obtained as follows:

\[
C_{n}^{x,y} = \frac{n-1}{n} C_{n-1}^{x,y} + \frac{1}{n-1} (x_n - \mu_n^x) (y_n - \mu_n^y)
\]  

(8)
To compute the covariance in a distributed way, we use the quantities $C_A^{xy} = \sum_{i=1}^{n_A} (x_i - \mu_A)(y_i - \mu_Y)$ and $C_B^{xy} = \sum_{i=1}^{n_B} (x_i - \mu_X)(y_i - \mu_Y)$ the co-moments matrices of two datasets A and B computed possibly on two machines, the combination is given by the following formula:

$$C_{AUB}^{xy} = C_A^{xy} + C_B^{xy} + (\mu_A^x - \mu_B^x)(\mu_A^y - \mu_B^y)n_A n_B / n$$  \hspace{1cm} (9)

The unbiased estimator of the covariance is obtained as $\text{Cov}(x,y) = \frac{1}{n-1} C_{AUB}^{xy}$

### 4.3. Weighted Mean

Let the weighted mean for $n$ samples defined as follows:

$$\mu_n = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$  \hspace{1cm} (10)

It is equivalent to the simple mean when all the weights are equal, however when the weights are not equal, weights can be thought of sample frequencies, or they can be used to calculate probabilities. Each weight can be normalised, that is divided by the sum of weights ($W_n$). By doing some basic manipulation we can write the weighted mean as:

$$\mu_n = W_{n-1} \mu_{n-1} + \frac{w_n}{W_n} x_n$$  \hspace{1cm} (11)

Like in Eq. 3, the distributed computation of two weighted means is given as:

$$\mu = \frac{W_A}{W_A + W_B} \mu_A + \frac{W_B}{W_A + W_B} \mu_B$$

where $W_A = \sum_{x_i \in A} w_i$ and $W_B = \sum_{x_i \in B} w_i$

### 4.4. Weighted Variance

We follow similar arguments used in the simple variance case with a slight modification, this time:

$$\sigma_n^2 = \frac{1}{W_n} \sum_{i=1}^{n} w_i (x_i - \mu)^2 = \frac{1}{W_n} \sum_{i=1}^{n} w_i x_i^2 - \mu^2$$  \hspace{1cm} (13)

Let $S_n = W_n \sigma_n^2$, where $W_n = \sum_{i=1}^{n} w_i$. Then we can obtain the following recursive formula:

$$S_n = S_{n-1} + w_n(x_n - \mu_{n-1})(x_n - \mu_n)$$  \hspace{1cm} (14)

We get on-line equation for variance:

$$\sigma_n = \sqrt{\frac{S_n}{W_n}}$$  \hspace{1cm} (15)

The distributed version can be computed in a similar way as in Eq. 9.
4.5. Exponentially Weighted Mean and Variance

Here we state a more useful scenario for data streams, we state few equations to calculate exponentially weighted mean and variance. The standard formula for exponentially weighted moving average is:

\[ \mu_n = \alpha \mu_{n-1} + (1 - \alpha)x_n \] (16)

where \( 0 < \alpha < 1 \), and we use the lower bound of 0 rather than 1 for convenience. We have on-line version as:

\[ \mu = \alpha^n x_0 + \sum_{i=1}^{n} (1 - \alpha) x_i \] (17)

We can write down the weights directly, since they’re independent of \( n \) and by summing geometric series we have the following:

\[ W_n = w_{n,0} + \sum_{i=1}^{n} w_{n,i} = \alpha^n + (1 - \alpha^n) = 1 \] (18)

Similarly for variance we have \( S_n = n\sigma_n^2 \), then we can derive:

\[ S_n = (1 - \alpha)S_{n-1} + \alpha(x_n - \mu_n)(x_n - \mu_{n-1}) \] (19)

and variance is:

\[ \sigma_n^2 = \frac{S_n}{W_n} = S_n = (1 - \alpha)(S_{n-1} + \alpha(x_n - \mu_{n-1})^2) \] (20)

4.6. Moving Average

Moving average is a process where the observation at step \( t \) linearly depends on some observations of a white noise sequence. Formally, this can be expressed:

\[ X_t = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q} \] (21)

where \( Z_t \) is white noise with zero mean and \( \sigma^2 \) as variance and \( \theta_1, ..., \theta_q \neq 0 \) are constants.

Often to approximate exponentially weighted average, for instance in the area of financial time series [32], Kalman filtering is used. Moreover, Kalman filter is the only equivalent to exponentially moving average for the case of random walk with noise [14]. Hence, when dealing with time series, kalman filters can be of extreme use.

Interestingly enough, we only require to focus on the innovation step of Kalman filter, as the problem in hand is to fit a moving average model to the observation \( x_1, ..., x_n \) with parameter \( q \) such that the mean squared distance between the set of observations is minimum. Note that the innovation in Kalman filter is defined as the difference between the observation and its prediction. We adopt the algorithm proposed in [34] shown in Algorithm 6 below to implement moving average.
Algorithm 6 Moving average (Innovation Algorithm)

1: **Input:** q
2: $k(i, j) = E(X_i, X_j)$
3: $v_0 = k(1, 1)$
4: $\hat{x}_1 = 0$
5: for $n = 1, 2, \ldots$ do
6:     for $k = 0, \ldots, n - 1$ do
7:         $\theta_{n, n-k} = v_{k-1}^{-1} \left( k(n+1, k+1) - \sum_{j=0}^{k-1} \theta_{k, k-j} \theta_{n, n-j} v_j \right)$
8:     end for
9:     $\hat{x}_{n+1} = \sum_{j=1}^{n} \theta_{n,j} \left( x_{n+1-j} - \hat{x}_{n+1-j} \right)$
10: $v_n = k(n+1, n+1) - \sum_{j=0}^{n-1} \theta_{n,n-j} v_j$
11: end for

Such algorithm is a typical example of how to have a recursive prediction, but it does not qualify as competitive on-line statistics algorithm\(^1\), there is no (mention on) guarantee of the bounds. Recently there have been two advances in on-line learning of ARMA [3] and ARIMA [21]. These two algorithms will be implemented and integrated into SOLMA as well.

### 4.7. Aggregation Algorithm

Aggregation algorithm (AA) [28] is a typical online learning algorithm that operates as an ensemble. AA is used mainly for competitive online prediction, where the goal is merging predictions of a number of experts. On-line learning consists of learning a sequentially presented set of training data upon arrival, without re-examining data that has been processed so far. In general on-line learning is practical for applications where the data set is large and cannot be processed at once due to memory constraints. Practically an on-line learner receives a new data instance, along with current hypothesis, checks if the data instance is covered by the current hypothesis and updates the hypothesis accordingly. The protocol of on-line learning can be summarized as follows: the learner receives an observation; the learner makes a decision; the learner receives the ground truth; learner incurs the loss and updates its hypothesis. The learning process is based on the minimisation of the loss (regret) which corresponds to the discrepancy between the loss and the loss of the best expert in hindsight.

The AA algorithm stands as a generalisation of the popular Weighted Majority algorithm [20]. It provides a weighted average that has bounds in the case of mixable game. In order to see the algorithm applied on brier game or for time series, please refer to [29] and [16] respectively.

In this section we provide the algorithmic details of AA and we show how it can be implemented in a distributed fashion for handling data streams. Aggregation algorithm is a typical example, which uses concept of weighted average and the exponential weighted average. However it goes one step beyond, that it provides an average that has bounds, in the case of mixable game.

Let $\Omega$ be an outcome space, $\Gamma$ be a prediction space and $\Theta$ be a (possibly infinite) set of experts. The learning process of AA can be seen as a game between a learner, experts and nature:

For any input at time $t$

---

\(^1\) An online algorithm is competitive, if the ratio between that algorithm and its optimal batch learning counterpart is bounded.
- Every expert $\theta \in \Theta$ makes a prediction $\gamma^\theta_t \in \Gamma$
- Learner L observes all predictions $\gamma^\theta_t$
- Learner L outputs a prediction $\gamma_t \in \Gamma$
- Nature outputs $\omega_t \in \Omega$
- Learner suffers a loss $\lambda(\gamma_t, \omega_t)$

The loss of AA cannot be much larger than the best expert for a mixable finite experts game while uniformly initialising the prior weights of the experts:

$$\text{Loss}(AA) = \text{Loss}_{\text{best}}(\theta) + \frac{\log N}{\eta}$$

(22)

where $\theta \in \Theta$, $\eta$ is the learning rate, and $N$ is the number of experts. This bound (22) is shown [30] to be optimal in a very strong sense, meaning that it cannot be improved by any other prediction algorithm. The pseudo-code is as follows [30]:

Algorithm 7 Aggregation Algorithm

 initialise weights $w^0_n = q_n$, $n = 1, 2, \ldots N$
 for $t = 1, 2, \ldots$ do
 notice experts prediction $\gamma^n_t$, $n = 1, 2, \ldots, N$
 weights normalisation $p^n_t = \frac{w^n_{t-1}}{\sum_{i=1}^N w^n_{t-1}}$
 solve the system $(\omega \in \Omega) \lambda(\gamma_t, \omega) \leq -\frac{C(n)}{\eta} \log \sum_{n=1}^N p^n_{t-1} e^{-\eta \lambda(\gamma^n_t, \omega_t)}$ w.r.t $\gamma$
 and output a solution $\gamma_t$
 notice $\omega_t$
 experts weights update $w^n_t = w^n_{t-1} e^{-\eta \lambda(\gamma^n_t, \omega_t)}$, $n = 1, 2, \ldots, N$
 end for

AA can be applied to achieve desired objectives such as weighted average. AA is quite appealing when mixing different methods but also for its easy implementation in distributed fashion.

Figure 4: Distributed version of the aggregation algorithm
5. Feature Reduction

5.1. Online PCA

Principal component Analysis (PCA) is a popular approach for dimensionality reduction. Suppose we have a random vector \( X = (X_1, \ldots, X_i)^T \), with a population variance-covariance matrix \( \Sigma \), then we can consider the following linear equation:

\[
Y_i = e_{i1}X_1 + e_{i2}X_2 + \ldots + e_{ii}X_i
\]

We can plug in values of \( i \) and obtain different equations which can be thought of linear regression, predicting \( Y_i \) from \( X_1, \ldots, X_i \) with no intercept. \( e_{i1}, \ldots, e_{ii} \) can be thought of as regression coefficients. We select these coefficients that maximise:

\[
\text{var}(Y_i) = \sum_{k=1}^{p} \sum_{l=1}^{p} e_{ik} e_{il} \sigma_{kl}
\]

where \( \sigma_{kl} \) denotes the \( k \)-th row and \( l \)-th column in \( \Sigma \). The main constraints added are that the sum of squared of coefficients adds to 1 and that the new component will be uncorrelated with all previously defined components. Hence:

\[
\text{cov}(Y_{i-1}, Y_i) = \sum_{k=1}^{p} \sum_{l=1}^{p} e_{i-1,k} e_{il} \sigma_{kl} = 0
\]

Formally the problem can be defined as given \( X \in \mathbb{R}^{d \times n} \), minimise over \( Y \in \mathbb{R}^{k \times d} \) where \( k < d \):

\[
\min_{\phi} \| X - \phi Y \|_F^2 \quad \text{or} \quad \min_{\phi} \| X - \phi Y \|_2^2
\]

In batch learning by just considering the top left singular vectors of the covariance matrix and projecting them gives the optimal solution for both norms. More formally if \( U_k \) is the span of the top \( k \) left singular vectors of \( X \), then \( Y = U_k^T X \) and \( \phi = U_k \) represents the optimal solution.

The few attempts that have been made to solve this problem in on-line setting do not provide the same solution for both norms. For instance, [4] provides bounds for Frobenius norm, while [17] provides spectral bounds. In [4] two algorithms are presented. The first algorithm requires Frobenius norm of \( X \) as input which makes it unrealistic for on-line setting. The second algorithm uses Frequent Directions and does not impose the Frobenius norm of \( X \) as input.

In [17], two algorithms are discussed. The first algorithm is space efficient, while the second one is time efficient. Both algorithms seem comparatively more practical. In this deliverable, we have considered the space-efficient version, as it is conceptually easier to understand and serves as basis for the time-efficient one. Unfortunately none of the papers gives empirical evidence for any of these algorithms. Thus, this report provides the first attempt to implement it.
Algorithm 8 Online PCA

Input: X, Δ
U ← all zero matrix (zero rows and zero columns)
B ← covariance sketch with precision ρ
for x_t ∈ X do
    append x_t to the sketch B
    while \(|(I - UU^T)|^2 \geq \Delta\) do
        Add top left singular vector of \((I - UU^T)B\) to U
    end while
    yield y_t = U^T x_t
end for
Return: Y

The algorithm starts with an empty projection matrix \(U\) and then adds singular vectors until some pre-specified value of \(\Delta\) is achieved. The second matrix used by the algorithm is \(B\) which is initialised using some sketching technique like Frequent Directions.

In order to implement online PCA in a distributed way, we may rely on two possibilities:

a- Merging the eigenspace models: the models can be merged using the approach developed in [Hall et al.] which shows how eigenspace models can be combined. For the sake of illustration, we consider two models computed by two different machines in parallel: \(\Omega = (\mu, U_{np}, \Lambda_{pp}, N)\) and \(\Psi = (\nu, V_{nq}, \Delta_{qq}, M)\) where \(\mu\) and \(\nu\) indicate the mean of the datasets, \(U_{np}\) and \(V_{nq}\) are the eigenvectors, \(\Lambda_{pp}\) and \(\Delta_{qq}\) are the eigenvalues and \(N\) and \(M\) are the size of the datasets of the two models. The combination results in a new model: \(\Phi = (z, W_{nrtr}, \Pi_{rrt}, P)\). The merge is done using Algorithm 9 below.

b- A more efficient alternative to implement OPCA in distributed fashion is to distribute data sample by sample on the existing machine. Each machine will run the optimization problem in parallel to compute \(U_i\) and \(B_i\). Then the top left singular vector, \(T_i\), is returned. These vectors are then concatenated to provide \(U\) which will sent to all machine to project the original input to produce the low-dimensional input \(y_i\). Figure 5 illustrates the process.

Figure 5: Distributed version of OPCA
5.2. Singular Value Decomposition

One of the most important aspect of stream processing is the time complexity of the algorithms. SVD is used everywhere, we provide a faster SVD algorithm. A lot of Machine Learning textbooks focusses on the Mahalonbis Distance, but in practice it is better to use penalised version. The reason behind this warning is to avoid the calculation of SVD, because the inverse entail a division by the covariance matrix singular values. When the input features are correlated you will get some singular values close to 0. So when computing the inverse of the covariance matrix you will divide by a very small number. This will make some of the newly derived features very large. This is unwanted since those features have the least use for machine learning purposes.

The un-centred covariance is calculated by using $X^T X$, if one need the centred version then we want to accomplish $(X^T X + \lambda I)^{-1}$, let $s_1, s_2, ...$ be our singular values of $X$, by replacing $X$ with its SVD($U \Sigma V^T$) and and applying Woodbury idenity [17], we get:

$$(X^T X + \lambda I)^{-1} V \text{diag} \left( \frac{s_i^2}{s_i^2 + \lambda} \right) V^T$$

The formula avoids division by a small number, furthermore, important features are shrunk less in comparison to other features. The whole process can be summarised as follows:

**Algorithm 9** Merging of two eigenspace models

**Input:** $\Omega = (\mu, U, \Lambda, N), \Psi = (\nu, V, \Delta, M)$

**Output:** $\Phi = (z, W, \Pi, P)$

1. $P = N + M$
2. $z = \frac{1}{P} (N \mu + M \nu)$
3. $\beta = \mu - \nu$
4. $G = U^T V$
5. $H = V - U G$

for each column vector of $H$ do

   Discard this column if it is of small magnitude

end for

$g = U^T \beta$

$h = \beta - U g$

$\nu =$ orthonormalbasis for $[H, h]$

$\gamma = \nu^T \beta$

$\Gamma = \nu^T V$

$p =$ size of $\Lambda$

$q =$ size of $\Delta$

$t =$ number of basis vectors in $\nu$

$A =$LHS of Equation 31 (see [4])

$\Pi =$ eigenvalues of $A$

$R =$ eigenvectors of $A$

$W = [U \nu] R$

discard small eigensolutions as appropriate

**Return:** $z, W, \Pi, P$
1. Compute $X^TX$
2. Compute \( \text{SVD}(X^TX) = VS^2V^T = VDV^T (D = S^2) \). Step 1 and 2 are in the case you don’t have a solver for SVD of large matrices.
3. Take the first top \( k \) singular values. Those are \( d_1 = s_1^2, \ldots, d_k = s_k^2 \).
4. Compute the transformed features: \( X_{\text{trans}} = V \text{diag} \left[ \sqrt{\frac{d_i}{d_i + \lambda}} \right] \).
5. Compute the Euclidean distance using the transformed features.

### 5.3. Random Projection

Random projection (RP) [9] is a technique that has found substantial use in the area of algorithm design (especially approximation algorithms), by allowing one to substantially reduce dimensionality of a problem while still retaining a significant degree of problem structure. In particular, given \( N \) points in \( n \)-dimensional Euclidean space, we can project these points down to a random \( p \)-dimensional subspace for \( p \leq n \).

Let \( X = \{x_i \in \mathbb{R}^n | i = 1 \ldots N \} \) be the input vectors in an \( n \)-dimensional space. RP embeds these vectors into a lower dimensional space \( \mathbb{R}^p \) where \( p \ll n \): \( x_i \in \mathbb{R}^n \rightarrow y_i \in \mathbb{R}^p \). The set \( Y = \{y_i \in \mathbb{R}^p | i = 1, N \} \) are called the embedding vectors.

To do this, a set of random vectors are generated \( M = \{r_k \in \mathbb{R}^p | k = 1, n \}. \) \( r_k \)'s are either generated uniformly over the \( p \)-dimensional unit space or chosen from a Bernoulli +1/-1 distribution and the vectors are normalized so that \( ||r_k||_2, k = 1, n \). The obtained matrix \( M \in \mathbb{R}^{p \times n} \) is used to compute the embedding \( y_i \) of \( x_i \) as follows: \( y_i = Mx_i \).

The distributed version of RP is straightforward. All needed is to replicate the random matrix over the machines that compute the projected data.

Note this work is currently being developed for a more ambitious setting namely random projection ensemble classification and random projection ensemble clustering for data streams.
6. Conclusions

The present document describes a set of basic streaming algorithms. We do not make any distinction between “online” and “streaming” as they fit both purposes. For each algorithm, we provided few details that allow the reader to understand the: purpose, the algorithmic steps, and the distributed implementation. The proposed algorithms were selected in a way to reflect on the different aspects related to big data, both data-at-rest and data-in-motion. We, in particular, focused on: sampling (4 algorithms), feature reduction (3 algorithms), compression (1 algorithm), and moments (5 simple ones and 2 algorithms). It is important to note that other basic algorithm will be included in SOLMA as we move into advanced algorithms. All algorithms are available on Github (https://github.com/proteus-h2020/SOLMA).

Currently we are still investigating matrix sketching, online SVD, random projection ensemble classification and random projection ensemble clustering for data streams. SOLMA will even richer in terms of basic scalable streaming algorithms.
References


