Mining Big Data in Real Time

Albert Bifet

Dortmund, 24 October 2013
Albert Bifet

2004-2009
Ph. D. Degree
UPC-Barcelona Tech
Advisors: Ricard Gavaldà and José L. Balcázar.

2009-2012
Post-Doctoral Researcher
University of Waikato,
Hamilton, New Zealand

2011-2013
Researcher
Yahoo! Research Barcelona
Albert Bifet

2004-2009
Ph. D. Degree
UPC-Barcelona Tech
Advisors: Ricard Gavaldà and José L. Balcázar.

2009-2012
Post-Doctoral Researcher
University of Waikato, Hamilton, New Zealand

20011-2013
Researcher
Yahoo! Research Barcelona
Upcoming BIG DATA Mining Project! The goal of SAMOA is to provide a framework for mining data streams using a cluster/cloud environment. In particular, we are interested in using Storm (http://storm-project.net) and S4 (http://incubator.apache.org/s4/) as the underlying computational framework.
New Zealand
Hamilton
Outline

1. MOA: Massive Online Analysis
2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Mining
3. Summary and Future Work: SAMOA
Outline

1. MOA: Massive Online Analysis
2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Mining
3. Summary and Future Work: SAMOA
Data Streams

Big Data & Real Time

**Big data** refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze.
BIG Data

- Volume
- Variety
- Velocity

3 Vs
BIG Data

- Volume
- Variety
- Velocity

- Variability
- Value
- Veracity

6 Vs
Distributed systems
Methodology

Paolo Boldi

Facebook Four degrees of separation

Big Data does not need big machines, it needs big intelligence
Introduction: Data Streams

Data Streams
- Sequence is potentially infinite
- High amount of data: sublinear space
- High speed of arrival: sublinear time per example
- Once an element from a data stream has been processed it is discarded or archived

Example
Puzzle: Finding Missing Numbers
- Let $\pi$ be a permutation of $\{1, \ldots, n\}$.
- Let $\pi_{-1}$ be $\pi$ with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number
Introduction: Data Streams

Data Streams
- Sequence is potentially infinite
- High amount of data: sublinear space
- High speed of arrival: sublinear time per example
- Once an element from a data stream has been processed it is discarded or archived

Example

Puzzle: Finding Missing Numbers
- Let $\pi$ be a permutation of $\{1, \ldots, n\}$.
- Let $\pi_{-1}$ be $\pi$ with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Use a $n$-bit vector to memorize all the numbers ($O(n)$ space)
Introduction: Data Streams

Data Streams

- Sequence is potentially infinite
- High amount of data: sublinear space
- High speed of arrival: sublinear time per example
- Once an element from a data stream has been processed it is discarded or archived

Example

Puzzle: Finding Missing Numbers

- Let $\pi$ be a permutation of $\{1, \ldots, n\}$.
- Let $\pi_{-1}$ be $\pi$ with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Data Streams: $O(\log(n))$ space.
Introduction: Data Streams

Data Streams
- Sequence is potentially infinite
- High amount of data: sublinear space
- High speed of arrival: sublinear time per example
- Once an element from a data stream has been processed it is discarded or archived

Example
Puzzle: Finding Missing Numbers
- Let $\pi$ be a permutation of $\{1, \ldots, n\}$.
- Let $\pi_{-1}$ be $\pi$ with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Data Streams:
$O(\log(n))$ space.

Store
\[
\frac{n(n+1)}{2} - \sum_{j \leq i} \pi_{-1}[j].
\]
Data Streams

Sequence is potentially infinite
High amount of data: sublinear space
High speed of arrival: sublinear time per example
Once an element from a data stream has been processed it is discarded or archived

Tools:
 approximation
 randomization, sampling
 sketching
Data Streams

● Sequence is potentially infinite
● High amount of data: sublinear space
● High speed of arrival: sublinear time per example
● Once an element from a data stream has been processed it is discarded or archived

Approximation algorithms

● Small error rate with high probability
● An algorithm \((\varepsilon, \delta)\)–approximates \(F\) if it outputs \(\tilde{F}\) for which \(\Pr[|\tilde{F} - F| > \varepsilon F] < \delta\).
Data Streams Approximation Algorithms

1011000111 1010101

Sliding Window

We can maintain simple statistics over sliding windows, using $O\left(\frac{1}{\varepsilon} \log^2 N\right)$ space, where

- $N$ is the length of the sliding window
- $\varepsilon$ is the accuracy parameter

We can maintain simple statistics over sliding windows, using $O\left(\frac{1}{\varepsilon} \log^2 N \right)$ space, where

- $N$ is the length of the sliding window
- $\varepsilon$ is the accuracy parameter

Data Streams Approximation Algorithms

101100011110 1010111

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon} \log^2 N)$ space, where

- $N$ is the length of the sliding window
- $\epsilon$ is the accuracy parameter

We can maintain simple statistics over sliding windows, using \( O\left(\frac{1}{\varepsilon} \log^2 N\right) \) space, where

- \( N \) is the length of the sliding window
- \( \varepsilon \) is the accuracy parameter

10110001111010 1011101

Sliding Window

We can maintain simple statistics over sliding windows, using $O\left(\frac{1}{\varepsilon} \log^2 N\right)$ space, where

- $N$ is the length of the sliding window
- $\varepsilon$ is the accuracy parameter

Data Streams Approximation Algorithms

101100011110101 0111010

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\varepsilon} \log^2 N)$ space, where

- $N$ is the length of the sliding window
- $\varepsilon$ is the accuracy parameter

Maintaining stream statistics over sliding windows. 2002
What is MOA?

Massive Online Analysis is a framework for online learning from data streams.

- It is closely related to WEKA
- It includes a collection of offline and online as well as tools for evaluation:
  - classification
  - clustering
- Easy to extend
- Easy to design and run experiments
History - timeline

WEKA

- 1993 - WEKA : project starts (Ian Witten)
- 1996 - First public release of WEKA in C
- Early 1997 - decision was made to rewrite WEKA in Java
- Mid 1999 - WEKA 3 (100% Java) released

MOA

- Nov 2007 - First public release of MOA: Richard Kirkby Thesis
- 2009 - MOA Concept Drift
- 2010 - MOA Clustering
- 2011 - MOA Graph Mining, Multi-label classification, Twitter Reader, Active Learning
- 2013 - MOA Outlier
*WEKA*

- **Waikato Environment for Knowledge Analysis**
- Collection of state-of-the-art machine learning algorithms and data processing tools implemented in Java
  - Released under the GPL
- Support for the whole process of experimental data mining
  - Preparation of input data
  - Statistical evaluation of learning schemes
  - Visualization of input data and the result of learning
- Used for education, research and applications
- Complements “Data Mining” by Witten & Frank & Hall
WEKA: Impact Downloads

Weka—Machine Learning Software in Java

Summary Files Reviews Support Wiki MediaWiki Code News

Date Range: 2000-11-05 to 2013-02-28

Downloads
3,770,004
In the selected date range

Top Country
United States
16% of downloaders

Top OS
Windows
63% of downloaders
WEKA: the bird
MOA: the bird

The Moa (another native NZ bird) is not only flightless, like the Weka, but also extinct.
The Moa (another native NZ bird) is not only flightless, like the Weka, but also extinct.
MOA: the bird

The Moa (another native NZ bird) is not only flightless, like the Weka, but also extinct.
Classification Experimental Setting

![Image of MOA Graphical User Interface]

- **Configure**
- **Evaluate Prequential**

<table>
<thead>
<tr>
<th>command</th>
<th>status</th>
<th>time elapsed</th>
<th>current activity</th>
<th>% complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate Prequential</td>
<td>running</td>
<td>54.64s</td>
<td>Evaluating learner...</td>
<td>25.69</td>
</tr>
<tr>
<td>Evaluate Prequential -l tree...</td>
<td>running</td>
<td>1m18s</td>
<td>Evaluating learner...</td>
<td>23.53</td>
</tr>
</tbody>
</table>

- **Preview (55.15s)**
- **Refresh**
- **Auto refresh:** every second

- **Export as .txt file...**

- **Evaluation**
  - **Values**
    - **Measure**
      - Accuracy: 74.40 - 73.58
      - Kappa: 46.52 - 44.32
      - Ram-Hours: 0.00 - 0.00
      - Time: 55.05 - 27.85
      - Memory: 0.00 - 0.00

- **Plot**
  - **Zoom in Y**
  - **Zoom out Y**
  - **Zoom in X**
  - **Zoom out X**
Classification Experimental Setting

Evaluation procedures for Data Streams

- Holdout
- Interleaved Test-Then-Train or Prequential

![Diagram of data flow and model requirements]

1. Input requirement 1
2. Learning requirements 2 & 3
3. Model requirement 4
Classification Experimental Setting

Data Sources
- Random Tree Generator
- Random RBF Generator
- LED Generator
- Waveform Generator
- Hyperplane
- SEA Generator
- STAGGER Generator
Classification Experimental Setting

Classifiers
- Naive Bayes
- Decision stumps
- Hoeffding Tree
- Hoeffding Option Tree
- Bagging and Boosting
- ADWIN Bagging and Leveraging Bagging

Prediction strategies
- Majority class
- Naive Bayes Leaves
- Adaptive Hybrid
RAM-Hours

RAM-Hour
Every GB of RAM deployed for 1 hour

Cloud Computing Rental Cost Options
Clustering Experimental Setting
## Clustering Experimental Setting

<table>
<thead>
<tr>
<th><strong>Internal measures</strong></th>
<th><strong>External measures</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>Rand statistic</td>
</tr>
<tr>
<td>C Index</td>
<td>Jaccard coefficient</td>
</tr>
<tr>
<td>Point-Biserial</td>
<td>Folkes and Mallow Index</td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>Hubert $\Gamma$ statistics</td>
</tr>
<tr>
<td>Dunn’s Index</td>
<td>Minkowski score</td>
</tr>
<tr>
<td>Tau</td>
<td>Purity</td>
</tr>
<tr>
<td>Tau A</td>
<td>van Dongen criterion</td>
</tr>
<tr>
<td>Tau C</td>
<td>$V$-measure</td>
</tr>
<tr>
<td>Somer’s Gamma</td>
<td>Completeness</td>
</tr>
<tr>
<td>Ratio of Repetition</td>
<td>Homogeneity</td>
</tr>
<tr>
<td>Modified Ratio of Repetition</td>
<td>Variation of information</td>
</tr>
<tr>
<td>Adjusted Ratio of Clustering</td>
<td>Mutual information</td>
</tr>
<tr>
<td>Fagan’s Index</td>
<td>Class-based entropy</td>
</tr>
<tr>
<td>Deviation Index</td>
<td>Cluster-based entropy</td>
</tr>
<tr>
<td>Z-Score Index</td>
<td>Precision</td>
</tr>
<tr>
<td>$D$ Index</td>
<td>Recall</td>
</tr>
<tr>
<td>Silhouette coefficient</td>
<td>F-measure</td>
</tr>
</tbody>
</table>

Table: Internal and external clustering evaluation measures.
Clusterers
- StreamKM++
- CluStream
- ClusTree
- Den-Stream
- D-Stream
- CobWeb
http://www.moa.cms.waikato.ac.nz
Easy Design of a MOA classifier

- `void resetLearningImpl ()`
- `void trainOnInstanceImpl (Instance inst)`
- `double[] getVotesForInstance (Instance i)`
Easy Design of a MOA clusterer

- void resetLearningImpl ()
- void trainOnInstanceImpl (Instance inst)
- Clustering getClusteringResult ()
Extensions of MOA

- Multi-label Classification
- Active Learning
- Regression
- Closed Frequent Graph Mining
- Twitter Sentiment Analysis

Challenges for bigger data streams
Sampling and distributed systems (Map-Reduce, Hadoop, S4)
MOA - Massive Online Analysis

MOA: Impact Downloads

DOWNLOADS
26,696
In the selected date range

TOP COUNTRY
United States
18% of downloaders

TOP OS
Windows
72% of downloaders
Outline

1. MOA: Massive Online Analysis

2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Mining

3. Summary and Future Work: SAMOA
Outline

1. MOA: Massive Online Analysis

2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Pattern Mining

3. Summary and Future Work: SAMOA
Hoeffding Trees

Hoeffding Tree : VFDT

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. 2000

- With high probability, constructs an identical model that a traditional (greedy) method would learn
- With theoretical guarantees on the error rate

```
Contains “Money”

Time

Day
Yes
YES

Night
No
NO

YES

YES

YES
```

Hoeffding Naive Bayes Tree

Hoeffding Tree
Majority Class learner at leaves

Hoeffding Naive Bayes Tree


- monitors accuracy of a Majority Class learner
- monitors accuracy of a Naive Bayes learner
- predicts using the most accurate method
Bagging builds a set of $M$ base models, with a bootstrap sample created by drawing random samples with replacement.
Bagging

Figure: Poisson(1) Distribution.

Each base model’s training set contains each of the original training example $K$ times where $P(K = k)$ follows a binomial distribution.
Oza and Russell’s Online Bagging for $M$ models

1: Initialize base models $h_m$ for all $m \in \{1, 2, ..., M\}$
2: for all training examples do
3: for $m = 1, 2, ..., M$ do
4: Set $w = \text{Poisson}(1)$
5: Update $h_m$ with the current example with weight $w$

6: anytime output:
7: return hypothesis: $h_{\text{fin}}(x) = \arg \max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$
Data Mining Algorithms with Concept Drift

No Concept Drift

![Diagram showing no concept drift]

Concept Drift

![Diagram showing concept drift]

| Counter
| Counter\(_2\) 
| Counter\(_3\) 
| Counter\(_4\) 
| Counter\(_5\) 

DM Algorithm

Change Detect.

Static Model
Data Mining Algorithms with Concept Drift

No Concept Drift

DM Algorithm

input

Counter₅
Counter₄
Counter₃
Counter₂
Counter₁

output

Concept Drift

DM Algorithm

input

Estimator₅
Estimator₄
Estimator₃
Estimator₂
Estimator₁

output
Optimal Change Detector and Predictor

- High accuracy
- Low false positives and false negatives ratios
- Theoretical guarantees
- Fast detection of change
- Low computational cost: minimum space and time needed
- No parameters needed
Algorithm **ADaptive Sliding Window**

Example

\[ W = 101010110111111 \]

**ADWIN**: **ADAPTIVE WINDOWING ALGORITHM**

1. Initialize Window \( W \)
2. \textbf{for} each \( t > 0 \)
3. \hspace{1em} \textbf{do} \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. \hspace{1em} \textbf{repeat} Drop elements from the tail of \( W \)
5. \hspace{2em} \textbf{until} \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \) holds
6. \hspace{2em} for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. \hspace{1em} Output \( \hat{\mu}_W \)
Algorithm **ADaptive Sliding Window**

**Example**

\[ W = \begin{bmatrix} 1010101101111111 \end{bmatrix} \]

\[ W_0 = 1 \]

\[ W_1 = 0101011011111111 \]

**ADWIN**: **ADAPTIVE WINDOWING ALGORITHM**

1. Initialize Window \( W \)
2. for each \( t > 0 \)
   
   3. do \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
   
   4. repeat Drop elements from the tail of \( W \)
   
   5. until \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \) holds
   
   6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
   
   7. Output \( \hat{\mu}_W \)
Algorithm **ADaptive Sliding WINDOW**

**Example**

\[ W = 101010110111111 \]
\[ W_0 = 10 \quad W_1 = 1010110111111 \]

**ADWIN**: **ADAPTIVE WINDOWING ALGORITHM**

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. \[ W \leftarrow W \cup \{x_t\} \] (i.e., add \( x_t \) to the head of \( W \))
4. repeat \( W \) Drop elements from the tail of \( W \)
5. until \[ |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \] holds
6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. Output \( \hat{\mu}_W \)
Algorithm Adaptive Sliding Window

Example

\[ W = [101010110111111] \]
\[ W_0 = [101] \quad W_1 = [010110111111] \]

ADWIN: Adaptive Windowing Algorithm

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. \hspace{1em} do \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. \hspace{1em} repeat Drop elements from the tail of \( W \)
5. \hspace{1em} \hspace{1em} until \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_c \) holds
6. \hspace{1em} \hspace{1em} for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. \hspace{1em} \hspace{1em} Output \( \hat{\mu}_W \)
Algorithm Adaptive Sliding Window

Example

\[ \mathcal{W} = \overline{1010101101111111} \]
\[ \mathcal{W}_0 = \overline{1010}, \quad \mathcal{W}_1 = \overline{10110111111} \]

**ADWIN:** Adaptive Windowing Algorithm

1. Initialize Window \( \mathcal{W} \)
2. for each \( t > 0 \)
3. do \( \mathcal{W} \leftarrow \mathcal{W} \cup \{ x_t \} \) (i.e., add \( x_t \) to the head of \( \mathcal{W} \))
4. repeat Drop elements from the tail of \( \mathcal{W} \)
5. until \( |\hat{\mu}_{\mathcal{W}_0} - \hat{\mu}_{\mathcal{W}_1}| \geq \varepsilon_c \) holds
6. for every split of \( \mathcal{W} \) into \( \mathcal{W} = \mathcal{W}_0 \cdot \mathcal{W}_1 \)
7. Output \( \hat{\mu}_{\mathcal{W}} \)
Algorithm **Adaptive Sliding Window**

Example

\[ W = 101010110111111 \]
\[ W_0 = 10101 \quad W_1 = 0110111111 \]

**ADWIN**: **ADAPTIVE WINDOWING ALGORITHM**

1. Initialize Window \( W \)
2. **for each** \( t > 0 \)
3. **do** \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. **repeat** Drop elements from the tail of \( W \)
5. **until** \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \) holds
6. **for every split of** \( W \) **into** \( W = W_0 \cdot W_1 \)
7. Output \( \hat{\mu}_W \)
Algorithm Adaptive Sliding Window

Example

\[ W = \overline{101010110111111} \]
\[ W_0 = \overline{101010} \quad W_1 = \overline{110111111} \]

**ADWIN: Adaptive Windowing Algorithm**

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. do \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. repeat Drop elements from the tail of \( W \)
5. until \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \) holds
6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. Output \( \hat{\mu}_W \)
Algorithm Adaptive Sliding Window

Example

\[ W = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]

\[ W_0 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \]

**ADWIN**: Adaptive Windowing Algorithm

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. do \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. repeat Drop elements from the tail of \( W \)
5. until \(|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c\) holds
6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. Output \( \hat{\mu}_W \)
Algorithm **ADaptive Sliding Window**

**Example**

\[ W = \overline{101010110111111} \]
\[ W_0 = \overline{10101011} \quad W_1 = \overline{0111111} \]

**ADWIN**: **ADAPTIVE WINDOWING ALGORITHM**

1. Initialize Window \( W \)
2. **for** each \( t > 0 \)
3. **do** \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. **repeat** Drop elements from the tail of \( W \)
5. **until** \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \) holds
6. **for** every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. **Output** \( \hat{\mu}_W \)
Algorithm Adaptive Sliding Window

Example

\[ W = \overline{101010110111111} \mid \hat{\mu}_W - \hat{\mu}_W \geq \varepsilon_c : \text{CHANGE DET.}! \]

\[ W_0 = \overline{101010110} \quad W_1 = \overline{111111} \]

\textbf{ADWIN: Adaptive Windowing Algorithm}

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. do \( W \leftarrow W \cup \{ x_t \} \) (i.e., add \( x_t \) to the head of \( W \))
4. repeat Drop elements from the tail of \( W \)
5. until \( |\hat{\mu}_W - \hat{\mu}_W| \geq \varepsilon_c \) holds
6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. Output \( \hat{\mu}_W \)
Algorithm **Adaptive Sliding Window**

**Example**

\[ W = \overline{1010101101111111} \]

Drop elements from the tail of \( W \)

\[ W_0 = \overline{101010110}, \quad W_1 = \overline{111111} \]

**ADWIN: Adaptive Windowing Algorithm**

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. do \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. repeat Drop elements from the tail of \( W \)
5. until \( |\mu_{W_0} - \mu_{W_1}| \geq \varepsilon_c \) holds
6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. Output \( \mu_W \)
Algorithm Adaptive Sliding Window

Example

\[ W = \overline{01010110111111} \quad \text{Drop elements from the tail of } W \]
\[ W_0 = \overline{101010110} \quad W_1 = \overline{111111} \]

ADWIN: Adaptive Windowing Algorithm

1. Initialize Window \( W \)
2. for each \( t > 0 \)
3. do \( W \leftarrow W \cup \{x_t\} \) (i.e., add \( x_t \) to the head of \( W \))
4. repeat Drop elements from the tail of \( W \)
5. until \( |\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c \) holds
6. for every split of \( W \) into \( W = W_0 \cdot W_1 \)
7. Output \( \hat{\mu}_W \)
Algorithm Adaptive Sliding Window

Theorem

At every time step we have:

1. (False positive rate bound). If $\mu_t$ remains constant within $W$, the probability that ADWIN shrinks the window at this step is at most $\delta$.

2. (False negative rate bound). Suppose that for some partition of $W$ in two parts $W_0 W_1$ (where $W_1$ contains the most recent items) we have $|\mu_{W_0} - \mu_{W_1}| > 2\epsilon_c$. Then with probability $1 - \delta$ ADWIN shrinks $W$ to $W_1$, or shorter.

ADWIN tunes itself to the data stream at hand, with no need for the user to hardwire or precompute parameters.
Algorithm **ADaptive Sliding Window**

**ADWIN** using a Data Stream Sliding Window Model,
- can provide the exact counts of 1’s in $O(1)$ time per point.
- tries $O(\log W)$ cutpoints
- uses $O\left(\frac{1}{\varepsilon} \log W\right)$ memory words
- the processing time per example is $O(\log W)$ (amortized and worst-case).

**Sliding Window Model**

<table>
<thead>
<tr>
<th>Content:</th>
<th>4</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity:</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Hoeffding Adaptive Tree:
- replace frequency statistics counters by estimators
  - don’t need a window to store examples, due to the fact that we maintain the statistics data needed with estimators
- change the way of checking the substitution of alternate subtrees, using a change detector with theoretical guarantees

Advantages over CVFDT:
1. Theoretical guarantees
2. No Parameters
**ADWIN Bagging (KDD’09)**

**ADWIN**
An adaptive sliding window whose size is recomputed online according to the rate of change observed.

**ADWIN has rigorous guarantees (theorems)**
- On ratio of false positives and negatives
- On the relation of the size of the current window and change rates

**ADWIN Bagging**
When a change is detected, the worst classifier is removed and a new classifier is added.
Leveraging Bagging for Evolving Data Streams

Randomization as a powerful tool to increase accuracy and diversity

There are three ways of using randomization:

- Manipulating the input data
- Manipulating the classifier algorithms
- Manipulating the output targets
Leveraging Bagging for Evolving Data Streams

Leveraging Bagging
- Using $\text{Poisson}(\lambda)$

Leveraging Bagging MC
- Using $\text{Poisson}(\lambda)$ and Random Output Codes

Fast Leveraging Bagging ME
- if an instance is misclassified: weight = 1
- if not: weight = $e_T/(1 - e_T)$,
Empirical evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>RAM-Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoeffding Tree</td>
<td>74.03%</td>
<td>0.01</td>
</tr>
<tr>
<td>Online Bagging</td>
<td>77.15%</td>
<td>2.98</td>
</tr>
<tr>
<td><strong>ADWIN Bagging</strong></td>
<td>79.24%</td>
<td>1.48</td>
</tr>
<tr>
<td>Leveraging Bagging</td>
<td>85.54%</td>
<td>20.17</td>
</tr>
<tr>
<td>Leveraging Bagging MC</td>
<td>85.37%</td>
<td>22.04</td>
</tr>
<tr>
<td>Leveraging Bagging ME</td>
<td>80.77%</td>
<td>0.87</td>
</tr>
</tbody>
</table>

**Leveraging Bagging**

- Leveraging Bagging
  - Using $\text{Poisson}(\lambda)$
- Leveraging Bagging MC
  - Using $\text{Poisson}(\lambda)$ and Random Output Codes
- Leveraging Bagging ME
  - Using weight 1 if misclassified, otherwise $e_T/(1 - e_T)$
Outline

1. MOA: Massive Online Analysis

2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Mining

3. Summary and Future Work: SAMOA
Active Learning

**ACTIVE LEARNING FRAMEWORK**

Input: labeling budget $B$ and strategy parameters

1. for each $X_t$ - incoming instance,
2. do if $\text{ACTIVE LEARNING STRATEGY}(X_t, B, \ldots) = \text{true}$
3. then request the true label $y_t$ of instance $X_t$
4. train classifier $L$ with $(X_t, y_t)$
5. if $L_n$ exists then train classifier $L_n$ with $(X_t, y_t)$
6. if change warning is signaled
7. then start a new classifier $L_n$
8. if change is detected
9. then replace classifier $L$ with $L_n$
Active Learning

<table>
<thead>
<tr>
<th></th>
<th>Controlling</th>
<th>Instance space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Budget</td>
<td>Coverage</td>
</tr>
<tr>
<td>Random</td>
<td>present</td>
<td>full</td>
</tr>
<tr>
<td>Fixed uncertainty</td>
<td>no</td>
<td>fragment</td>
</tr>
<tr>
<td>Variable uncertainty</td>
<td>handled</td>
<td>fragment</td>
</tr>
<tr>
<td>Randomized uncertainty</td>
<td>handled</td>
<td>full</td>
</tr>
</tbody>
</table>

Table: Summary of strategies.
Outline

1 MOA: Massive Online Analysis

2 Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Pattern Mining

3 Summary and Future Work: SAMOA
Multi-label classification

- **Binary Classification:** e.g. is this a beach? \( \in \{ \text{No, Yes} \} \)
- **Multi-class Classification:** e.g. what is this? \( \in \{ \text{Beach, Forest, City, People} \} \)
- **Multi-label Classification:** e.g. which of these? \( \subseteq \{ \text{Beach, Forest, City, People} \} \)

- **Binary Relevance method (BR)**
  - One binary classifier for each label:
    - simple; flexible; fast but does not explicitly model label dependencies

- **Label Powerset method (LP)**
  - One multi-class classifier; one class for each labelset
Data Streams Multi-label Classification

- **Adaptive Ensembles of Classifier Chains (ECC)**
  - Hoeffding trees as base-classifiers reset classifiers based on current performance / concept drift

- **Multi-label Hoeffding Tree**
  - Label Powerset method (LP) at the leaves an ensemble strategy to deal with concept drift

- **MOA Multi-label Setting**
  - generating synthetic multi-label data streams
  - setting a benchmark on real-world and synthetic data
Outline

1. MOA: Massive Online Analysis
2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Mining
3. Summary and Future Work: SAMOA
Classification using patterns: mapping from patterns to vectors of attributes
## Itemset Mining

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
<th>Frequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>6</td>
<td>c</td>
</tr>
<tr>
<td>d2</td>
<td>5</td>
<td>e, ce</td>
</tr>
<tr>
<td>d3</td>
<td>4</td>
<td>a, ac, ae, ace</td>
</tr>
<tr>
<td>d4</td>
<td>4</td>
<td>b, bc</td>
</tr>
<tr>
<td>d5</td>
<td>4</td>
<td>d, cd</td>
</tr>
<tr>
<td>d6</td>
<td>3</td>
<td>ab, abc, abe, be, bce, abce</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>de, cde</td>
</tr>
<tr>
<td>d1</td>
<td>abce</td>
<td>Support</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>d2</td>
<td>cde</td>
<td>6</td>
</tr>
<tr>
<td>d3</td>
<td>abce</td>
<td>5</td>
</tr>
<tr>
<td>d4</td>
<td>acde</td>
<td>4</td>
</tr>
<tr>
<td>d5</td>
<td>abcde</td>
<td>4</td>
</tr>
<tr>
<td>d6</td>
<td>bcd</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Support</th>
<th>Frequent</th>
<th>Gen</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ab,abc,abe</td>
<td>ab</td>
<td></td>
</tr>
<tr>
<td></td>
<td>be,bce,abce</td>
<td>be</td>
<td>abce</td>
</tr>
<tr>
<td>3</td>
<td>de,cde</td>
<td>de</td>
<td>cde</td>
</tr>
</tbody>
</table>
### Itemset Mining

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
<th>Frequent</th>
<th>Gen</th>
<th>Closed</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1 abce</td>
<td>6</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d2 cde</td>
<td>5</td>
<td>e,ce</td>
<td>e</td>
<td>ce</td>
<td></td>
</tr>
<tr>
<td>d3 abce</td>
<td>4</td>
<td>a,ac,ae,ace</td>
<td>a</td>
<td>ace</td>
<td></td>
</tr>
<tr>
<td>d4 acde</td>
<td>4</td>
<td>b,bc</td>
<td>b</td>
<td>bc</td>
<td></td>
</tr>
<tr>
<td>d5 abcde</td>
<td>3</td>
<td>d,cd</td>
<td>d</td>
<td>cd</td>
<td>abce</td>
</tr>
<tr>
<td>d6 bcd</td>
<td>3</td>
<td>ab,abc,abe</td>
<td>ab</td>
<td>abce</td>
<td>abce</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be,bce,abce</td>
<td>be</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>de,cde</td>
<td>de</td>
<td>cde</td>
<td></td>
</tr>
</tbody>
</table>
Coreset of a set $P$ with respect to some problem
Small subset that approximates the original set $P$.
  • Solving the problem for the coreset provides an approximate solution for the problem on $P$.

$\delta$-tolerance Closed Graph
A graph $g$ is $\delta$-tolerance closed if none of its proper frequent supergraphs has a weighted support $\leq (1 - \delta) \cdot \text{support}(g)$.
  • Maximal graph: 1-tolerance closed graph
  • Closed graph: 0-tolerance closed graph.
Relative support of a closed graph
Support of a graph minus the relative support of its closed supergraphs.
- The sum of the closed supergraphs’ relative supports of a graph is equal to its own support.

\((s, \delta)\)-coreset for the problem of computing closed graphs
Weighted multiset of frequent \(\delta\)-tolerance closed graphs with minimum support \(s\) using their relative support as a weight.
# Graph Dataset

<table>
<thead>
<tr>
<th>Transaction Id</th>
<th>Graph</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{O}$ . $\text{C} \cdot \text{C} \cdot \text{S} \cdot \text{N}$ . $\text{O}$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$\text{O}$ . $\text{C} \cdot \text{C} \cdot \text{S} \cdot \text{N}$ . $\text{C}$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$\text{O}$ . $\text{C} \cdot \text{S} \cdot \text{N}$ . $\text{C}$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$\text{N}$ . $\text{C} \cdot \text{C} \cdot \text{S} \cdot \text{N}$</td>
<td>1</td>
</tr>
</tbody>
</table>
## Graph Coresets

<table>
<thead>
<tr>
<th>Graph</th>
<th>Relative Support</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>C - C - S - N</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C - S - N</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C - S</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table:** Example of a coreset with minimum support 50% and $\delta = 1$
Graph Coresets

Figure: Number of graphs in (40%, δ) for NCI.
ChemDB dataset

![Memory ChemDB Dataset](image)
ChemDB dataset
Outline

1. MOA: Massive Online Analysis
2. Adaptive Size Sliding Window Learning
   - Classification
   - Active Learning
   - Multi-label Classification
   - Frequent Pattern Mining
3. Summary and Future Work: SAMOA
New Techniques: Distributed Systems

Hadoop, S4 and Storm
Hadoop
Hadoop architecture
Apache Mahout

Mahout: open source framework
Apache S4
Apache S4

A keyless event (EV) arrives at PE1 with quote: "I meant what I said and I said what I meant.", Dr. Seuss

**QuoteSplitterPE** (PE1) counts unique words in Quote and emits events for each word.

**WordEvent**
- **EV**: WordEvent
- **KEY**: word="I"
- **VAL**: count=4

**WordCountPE** (PE2–4) keeps total counts for each word across all quotes. Emits an event any time a count is updated.

**UpdateCountEv**
- **EV**: UpdateCountEv
- **KEY**: sortID=2
- **VAL**: word=said count=9

**SortPE** (PE5–7) continuously sorts partial lists. Emits lists at periodic intervals.

**PartialTopKEv**
- **EV**: PartialTopKEv
- **KEY**: topk=1234
- **VAL**: words=[w:cnt]

**MergePE** (PE8) combines partial TopK lists and outputs final TopK list.

**PE ID** | **PE Name** | **Key Tuple**
---|---|---
PE1 | QuoteSplitterPE | null
PE2 | WordCountPE | word="said"
PE4 | WordCountPE | word="I"
PE5 | SortPE | sortID=2
PE7 | SortPE | sortID=9
PE8 | MergePE | topK=1234
Storm from Twitter
Storm

Stream, Spout, Bolt, Topology
Runaway complexity in Big Data
Nathan Marz, 2012
SAMOA: MOA + S4/Storm
Vertical Hoeffding Tree
Summary

Massive Online Analysis is a framework for online learning from data streams.

http://moa.cs.waikato.ac.nz

- It is closely related to WEKA
- It includes a collection of offline and online as well as tools for evaluation:
  - classification
  - clustering
  - frequent pattern mining
- MOA deals with evolving data streams
- MOA is easy to use and extend
SAMOA: MOA + S4/Storm
Thanks!

SAMOA Scalable Advanced Massive Online Analysis

WELCOME TO SAMOA!

Upcoming BIG DATA Mining Project! The goal of SAMOA is to provide a framework for mining data streams using a cluster/cloud environment. In particular, we are interested in using Storm (http://storm-project.net) and S4 (http://incubator.apache.org/s4/) as the underlying computational framework.