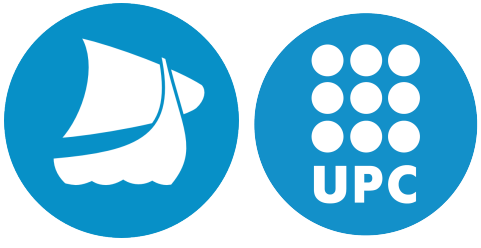
Improving Adaptive Bagging Methods for Evolving Data Streams

A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà

University of Waikato
Hamilton, New Zealand

Laboratory for Relational Algorithmics, Complexity and Learning **LARCA**
UPC-Barcelona Tech, Catalonia

Nanjing, 4 November 2009
1st Asian Conference on Machine Learning (ACML'09)



THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Motivation

MOA Software for Mining Data Streams

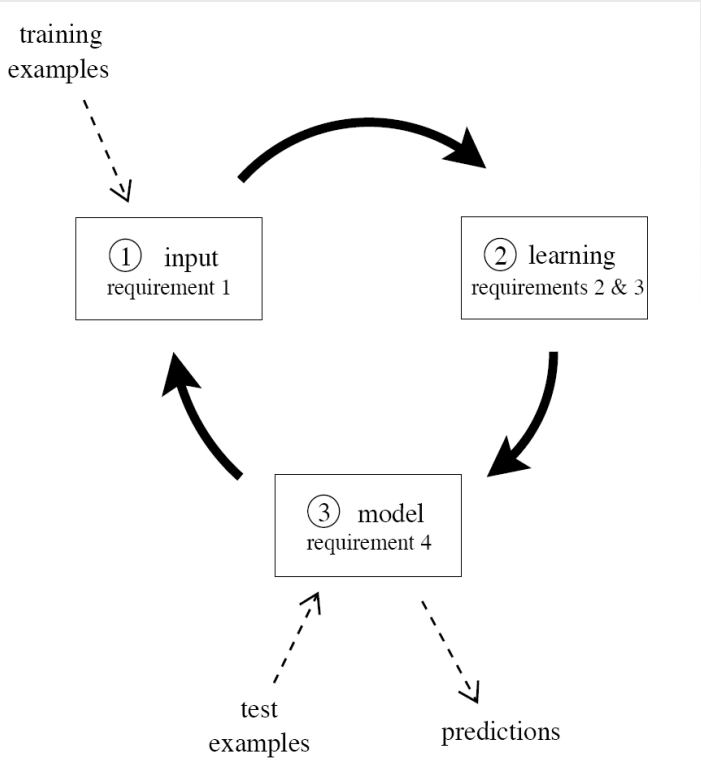
- Build a useful software mining for massive data sets

Bagging for data streams

- Improve accuracy on classification methods for data streams

Data stream classification cycle

1. Process an example at a time, and inspect it only once (at most)
2. Use a limited amount of memory
3. Work in a limited amount of time
4. Be ready to predict at any point



Realtime analytics: from Databases to Dataflows

Data streams

- Data streams are ordered datasets
- Not all datasets are data streams
- All dataset may be processed incrementally as a data stream

MOA: Massive Online Analysis

- Faster Mining Software using less resources

Instant mining: more for less

What is MOA?

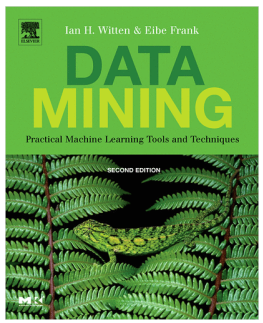
{M}assive {O}nline {A}nalysis is a framework for online learning from data streams.

- It is closely related to WEKA
- It includes a collection of offline and online as well as tools for evaluation:
  - boosting and bagging
  - Hoeffding Trees

  with and without Naïve Bayes classifiers at the leaves.

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



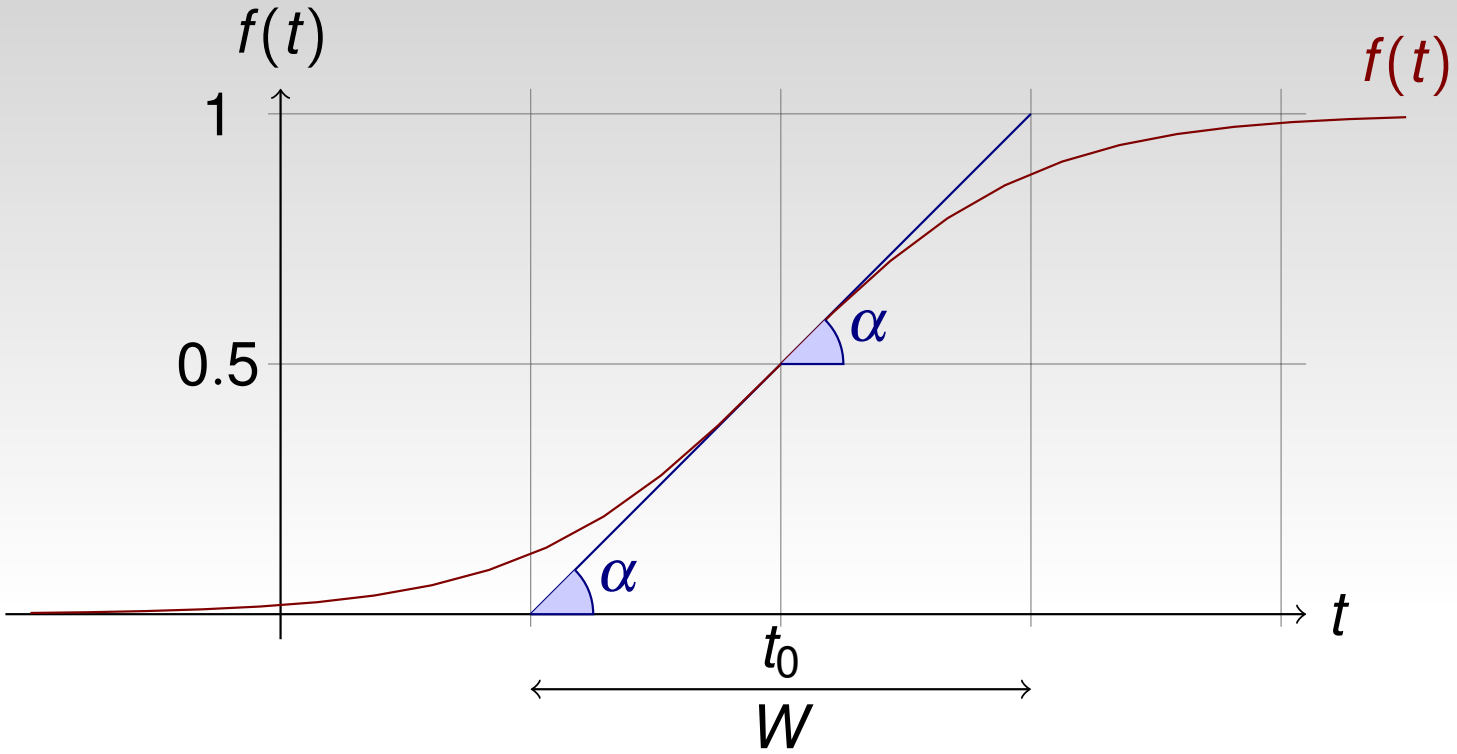
- Used for education, research and applications
- Complements “Data Mining” by Witten & Frank

WEKA: the bird

MOA: the bird

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

Concept Drift Framework

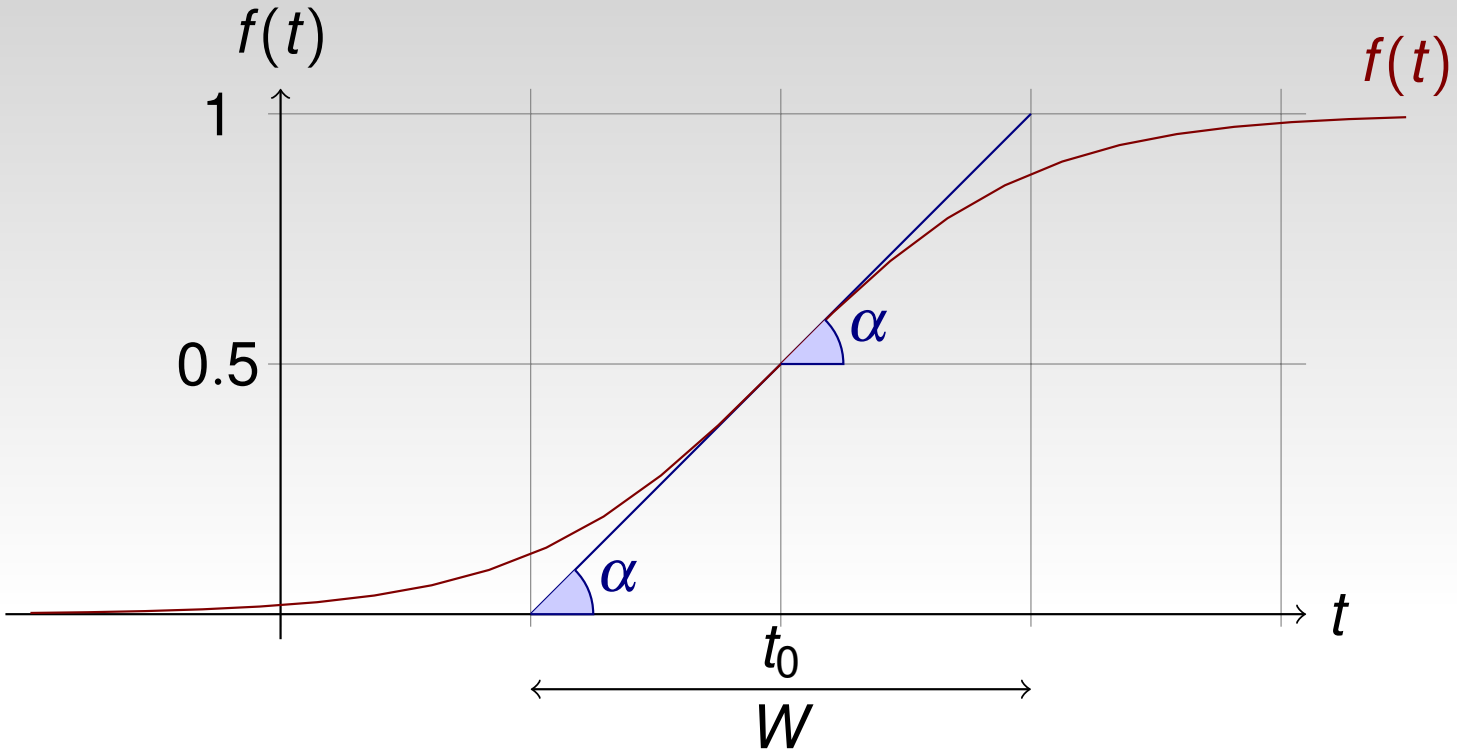


Definition

Given two data streams a, b, we define $c = a \oplus_{t_0}^{W} b$ as the data stream built joining the two data streams a and b

- $\Pr[c(t) = b(t)] = 1/(1 + e^{-4(t-t_0)/W})$.
- $\Pr[c(t) = a(t)] = 1 - \Pr[c(t) = b(t)]$

Concept Drift Framework



Example

- $(((a \oplus_{t_0}^{W_0} b) \oplus_{t_1}^{W_1} c) \oplus_{t_2}^{W_2} d) \ldots$
- $(((SEA_9 \oplus_{t_0}^{W} SEA_8) \oplus_{2t_0}^{W} SEA_7) \oplus_{3t_0}^{W} SEA_{9.5})$
- $\text{CovPokElec} = (\text{CoverType} \oplus_{581,012}^{5,000} \text{Poker}) \oplus_{1,000,000}^{5,000} \text{ELEC2}$

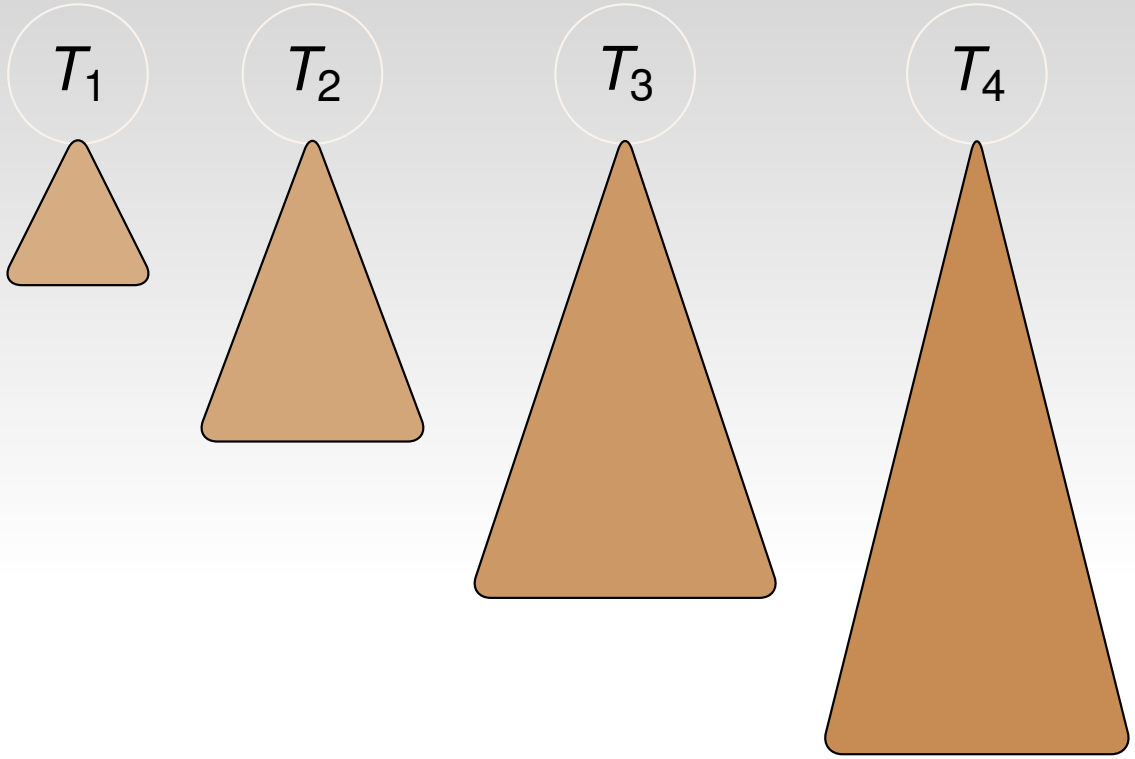
New Ensemble Methods For Evolving Data Streams

New Ensemble Methods For Evolving Streams (KDD'09)

- a new experimental data stream framework for studying concept drift
- two new variants of Bagging:
  - `ADWIN` Bagging
  - Adaptive-Size Hoeffding Tree (ASHT) Bagging.
- an evaluation study on synthetic and real-world datasets

Outline

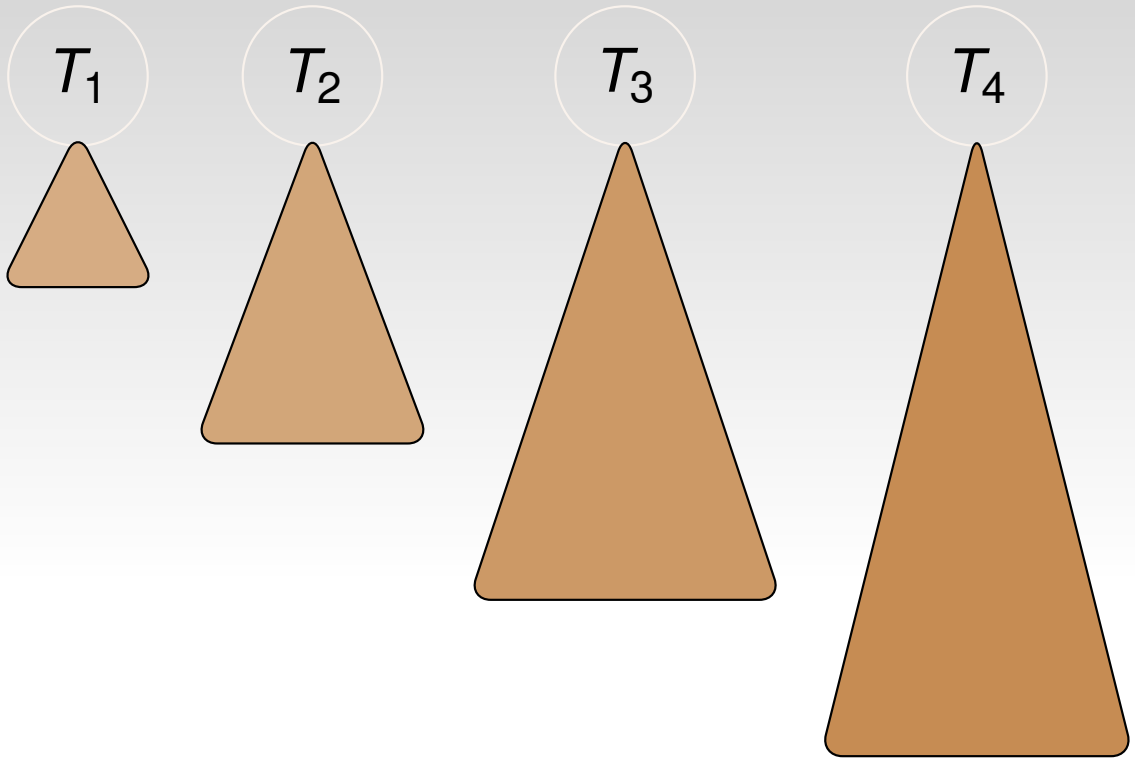
Adaptive-Size Hoeffding Tree



Ensemble of trees of different size

- each tree has a maximum size
- after one node splits, it deletes some nodes to reduce its size if the size of the tree is higher than the maximum value

Adaptive-Size Hoeffding Tree



Ensemble of trees of different size

- smaller trees adapt more quickly to changes,
- larger trees do better during periods with little change
- diversity

Adaptive-Size Hoeffding Tree

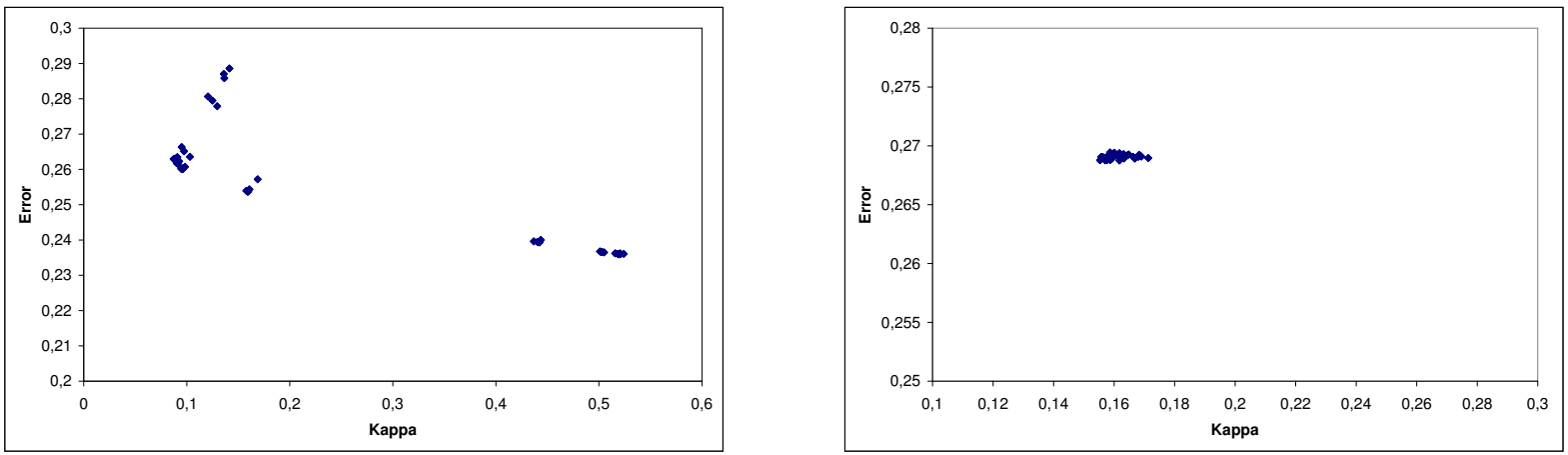


Figure: Kappa-Error diagrams for ASHT bagging (left) and bagging (right) on dataset RandomRBF with drift, plotting 90 pairs of classifiers.

Improvement for ASHT Bagging Method

Improvement for ASHT Bagging ensemble method

- Bagging using trees of different size
  - add a change detector for each tree in the ensemble
    - DDM: Gama et al.
    - EDDM: Baena, del Campo, Fidalgo et al.

Outline

ADWIN Bagging

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.

Optimal Change Detector and Predictor

ADWIN

- High accuracy
- Fast detection of change
- Low false positives and false negatives ratios
- Low computational cost: minimum space and time needed
- Theoretical guarantees
- No parameters needed
- Estimator with Memory and Change Detector

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 101010110111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 101010110111111

W_0= 1 W_1 = 01010110111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 101010110111111

W_0= 10 W_1 = 1010110111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

$W =$ 101010110111111

$W_0 =$ 101 $W_1 =$ 010110111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

$W =$ 101010110111111

$W_0 =$ 1010 $W_1 =$ 10110111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 1010101101111111

W_0= 10101 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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 1010101101111111

W_0= 101010 W_1 = 1101111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 101010110111111

W_0= 1010101 W_1 = 10111111

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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 1010101101111111

W_0= 10101011 W_1 = 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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 101010110111111 $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_c$: CHANGE DET.!

W_0= 101010110 W_1 = 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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 1 01010110111111 Drop elements from the tail of W

W_0= 101010110 W_1 = 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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Example

W= 01010110111111 Drop elements from the tail of W

W_0= 101010110 W_1 = 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** $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \varepsilon_c$ holds
6 for every split of W into $W = W_0 \cdot W_1$
7 Output $\hat{\mu}_W$

Algorithm ADaptive Sliding WINdow

ADWIN

Theorem

At every time step we have:

1. (False positive rate bound). *If μ_t remains constant within W, the probability that* ADWIN *shrinks the window at this step is at most δ.*
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 $|\mu_{W_0} - \mu_{W_1}| > 2\varepsilon_c$. 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.

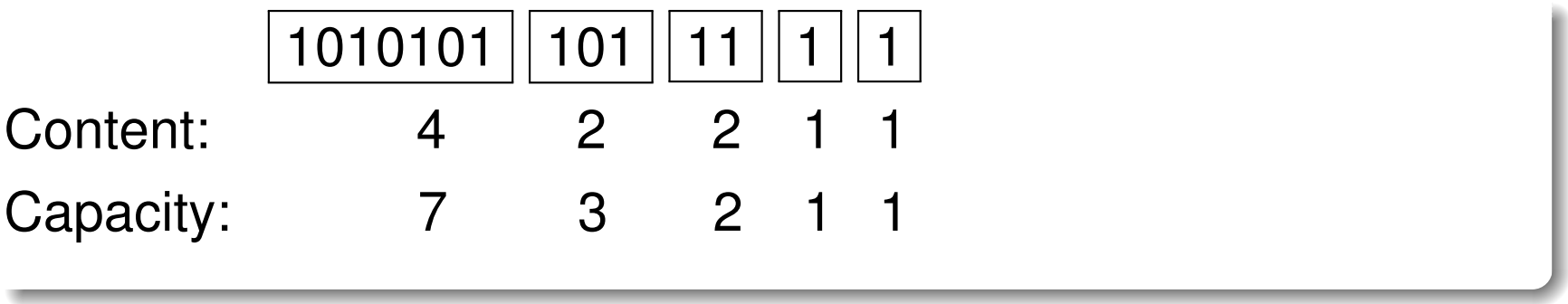
Algorithm ADaptive Sliding WINdow

ADWIN

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 $O(\frac{1}{\varepsilon} \log W)$ memory words
- the processing time per example is $O(\log W)$ (amortized and worst-case).

Sliding Window Model



ADWIN bagging using Hoeffding Adaptive Trees

Decision Trees: Hoeffding Adaptive Tree

CVFDT: Hulten, Spencer and Domingos

- No theoretical guarantees on the error rate of CVFDT
- Parameters needed : size of window, number of examples,...

Hoeffding Adaptive Tree:

- replace frequency statistics counters by estimators
  - don't need a window to store examples
- use a change detector with theoretical guarantees to substitute trees

Advantages:

1. Theoretical guarantees
2. No Parameters

Outline

Empirical evaluation

Dataset	Most Accurate Method
Hyperplane Drift 0.0001	Bag10 ASHT W+R
Hyperplane Drift 0.001	DDM Bag10 ASHT W
SEA W = 50	Bag`ADWIN` 10 HAT
SEA W = 50000	Bag`ADWIN` 10 HAT
RandomRBF No Drift 50 centers	Bag 10 HT
RandomRBF Drift .0001 50 centers	Bag`ADWIN` 10 HAT
RandomRBF Drift .001 50 centers	DDM Bag10 ASHT W
RandomRBF Drift .001 10 centers	Bag`ADWIN` 10 HAT
Cover Type	DDM Bag10 ASHT W
Poker	Bag`ADWIN` 10 HAT
Electricity	DDM Bag10 ASHT W
CovPokElec	Bag`ADWIN` 10 HAT

Empirical evaluation

	SEA W= 50000		
	Time	Acc.	Mem.
BagADWIN 10 HAT	154.91	**88.88** ± 0.05	2.35
DDM Bag10 ASHT W	44.02	88.72 ± 0.05	0.65
NaiveBayes	5.52	84.60 ± 0.03	0.00
NBADWIN	12.40	87.83 ± 0.07	0.02
HT	7.20	85.02 ± 0.11	0.33
HT DDM	7.88	88.17 ± 0.18	0.16
HAT	20.96	88.40 ± 0.07	0.18
BagADWIN 10 HT	53.15	88.58 ± 0.10	0.88
Bag10 HT	30.88	85.38 ± 0.06	3.36
Bag10 ASHT W+R	33.56	88.51 ± 0.06	0.84

Empirical evaluation

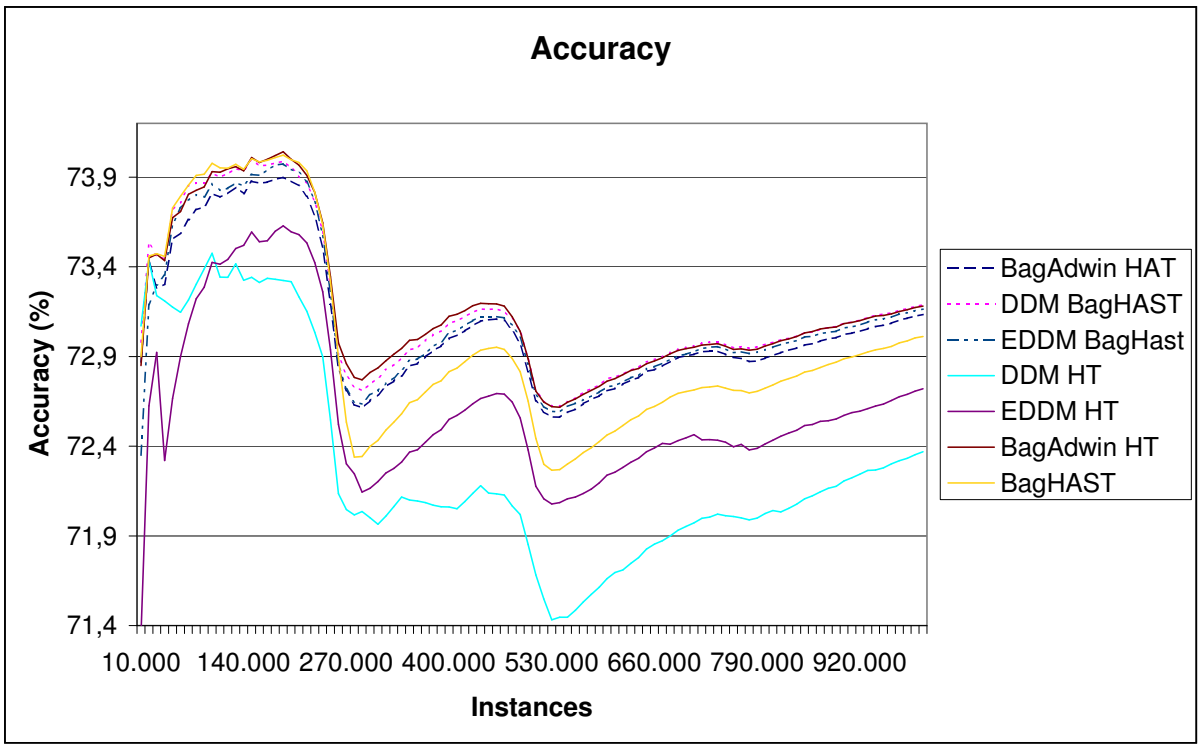


Figure: Accuracy on dataset LED with three concept drifts.

Summary

http://www.cs.waikato.ac.nz/~abifet/MOA/

Conclusions

- New improvements for ensemble bagging methods:
  - Adaptive-Size Hoeffding Tree bagging using change detection methods
  - ADWIN bagging using Hoeffding Adaptive Trees
- MOA is easy to use and extend

Future Work

- Extend MOA to more data mining and learning methods.