Chapter 7:

Stream Applications & Algorithms (Clustering and Classification)

Lectures : Dr Eirini Ntoutsi, PD Dr Matthias Schubert
Tutorials: PD Dr Matthias Schubert

CluStream [AggEtAl03]

- The stream clustering process is separated into:
  - an online micro-cluster component, that summarizes the stream locally as new data arrive over time
    - Micro-clusters are stored in disk at snapshots in time that follow a pyramidal time frame.
  - an offline macro-cluster component, that clusters these summaries into global clusters
    - Clustering is performed upon summaries instead of raw data

CluStream: the micro-cluster summary structure

- Assume that the data stream consists of a set of multi-dimensional records $X_1,\ldots X_n,\ldots$, arriving at $T_1,\ldots,T_n,\ldots$: $X_i = (x_i^1,\ldots,x_i^d)$
- The micro-cluster summary for a set of d-dimensional points ($X_1, X_2, \ldots, X_n$) arriving at time points $T_1, T_2, \ldots, T_n$ is defined as:

$$CFT = (CF2^x, CF1^x, CF2^t, CF1^t, n)$$

$$CF2^x = \sum_{i=1}^{N} X_i^2 \qquad CF1^x = \sum_{i=1}^{N} X_i \qquad CF2^t = \sum_{i=1}^{N} T_i^2 \qquad CF1^t = \sum_{i=1}^{N} T_i$$

- Easy calculation of basic measures to characterize a cluster:
  - Center: $\frac{CF1^x}{n}$
  - Radius: $\sqrt{\frac{CF2^x}{n} - \left(\frac{CF1^x}{n}\right)^2}$
- Important properties of micro-clusters:
  - Incrementality: $CFT(C_1 \cup p) = CFT(C_1) + p$
  - Additivity: $CFT(C_1 \cup C_2) = CFT(C_1) + CFT(C_2)$
  - Subtractivity: $CFT(C_1 - C_2) = CFT(C_1) - CFT(C_2)$, $C_1 \supseteq C_2$

CluStream: overview

- A fixed number of *q* micro-clusters is maintained over time
- Initialize: apply *q*-Means over *initPoints*, built a summary for each cluster
- Online micro-cluster maintenance as a new point *p* arrives from the stream
  - Find the closest micro-cluster *clu* for the new point *p*
    - If *p* is within the max-boundary of *clu*, *p* is absorbed by *clu*
    - o.w., a new cluster is created with *p*
  - The number of micro-clusters should not exceed q
    - Delete most obsolete micro-cluster or merge the two closest ones
- Periodic storage of micro-clusters snapshots into disk
  - At different levels of granularity depending upon their recency
- Offline macro-clustering
  - Input: A user defined time horizon *h* and number of macro-clusters *k* to be detected
  - Locate the valid micro-clusters during *h*
  - Apply k-Means upon these micro-clusters → k macro-clusters

CluStream: Initialization step

- Initialization
  - Done using an offline process in the beginning
  - Wait for the first *InitNumber* points to arrive
  - Apply a standard *k*-Means algorithm to create *q* clusters
    - For each discovered cluster, assign it a unique ID and create its micro-cluster summary.

- Comments on the choice of *q*
  - much larger than the natural number of clusters
  - much smaller than the total number of points arrived

CluStream: Online step

- A fixed number of *q* micro-clusters is maintained over time
- Whenever a new point *p* arrives from the stream
  - Compute distance between *p* and each of the *q* maintained micro-cluster centroids
  - *clu* ← the closest micro-cluster to *p*
  - Find the max boundary of *clu*
    - It is defined as a factor of *t* of *clu* radius
  - If p falls within the maximum boundary of *clu*
    - *p* is absorbed by *clu*
    - Update clu statistics (incremental property)
  - Else, create a new micro-cluster with p, assign it a new ID, initialize its statistics
    - To keep the total number of micro-clusters fixed (i.e., *q*):
      - Delete the most obsolete micro-cluster or
        - If its safe based on its time statistics
      - Merge the two closest ones (Additivity property)
        - When two micro-clusters are merged, a list of ids is created. This way, we can identify the component micro-clusters that comprise a micro-cluster.

CluStream: Periodic micro-cluster storage

- Micro-clusters are stored as snapshots in time following the pyramidal pattern
  - They are stored at different levels of granularity based on their recency
- Snapshots are classified at different orders/levels i
  - For each order i, we store snapshots if the current timestamp t is dived by a^i, but not by a^{i+1}(to avoid redundancy)
  - At most a^b+1 snapshots are stored at each order; if a new snapshot arrives the oldest one is deleted.
- #orders: $\log_a(t)$
- #stored snapshots: $(a^b+1)\log_a(t)$

Level	Snapshots
0	59 57 55 53 51
1	58 54 50 36 42
2	60 52 44 36 28
3	56 40 24 8
4	48 16
5	32

Snapshots stored at t = 60, a=2, b=2

CluStream: Offline step

- The offline step is applied on demand
- User input: time horizon *h, #* macro-clusters k to be detected
- Step 1: Find the active micro-clusters during *h*:
  - We exploit the subtractivity property to find the active micro-clusters during *h*:
    - Suppose current time is t_c. Let S(t_c) be the set of micro-clusters at t_c.
    - Find the stored snapshot which occurs just before time t_c-*h.* We can always find such a snapshot *h'.* Let S(t_c–h') be the set of micro-clusters.
    - For each micro-cluster in the current set S(t_c), we find the list of its component micro-cluster ids. For each of the list of ids, find the corresponding micro-clusters in S(t_c–h').
    - Subtract the CF vectors for the corresponding micro-clusters in S(t_c–h')
    - This ensures that the micro-clusters created before the user-specified horizon do not dominate the result of clustering process
- Step 2: Apply k-Means over the active micro-clusters in *h* to derive the *k* macro-clusters
  - Initialization: centers are not picked up randomly, rather sampled with probability proportional to the number of points in a given micro-cluster
  - Distance is the centroid distance
  - New centers are defined as the weighted centroids of the micro-clusters in that partition

CluStream: overview

+ CluStream clusters large evolving data streams
+ Views the stream as a changing process over time, rather than clustering the whole stream at a time
+ Can characterize clusters over different time horizons in changing environment
+ Provides flexibility to an analyst in a real-time and changing environment

– Fixed number of micro-clusters maintained over time
– Sensitive to outliers/ noise

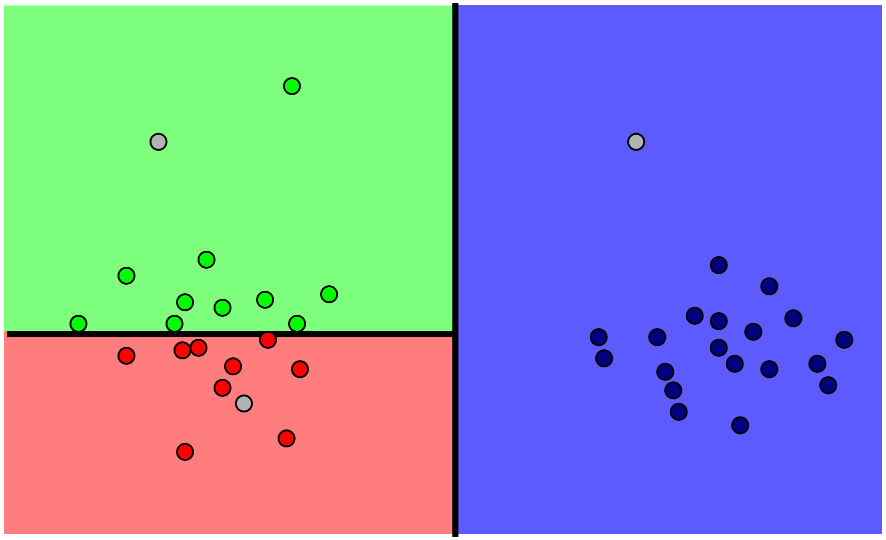
Summary: stream clustering

- A very important task given the availability of streams nowadays
- Stream clustering algorithm maintain a valid clustering of the evolving stream population over time
- Two generic approaches
  - Online maintenance of a final clustering model
  - Online summarization of the stream and offline clustering
    - Summaries!
- Different window models
- Evaluation is not straightforward
  - Internal measures of clustering quality (e.g., centroid's radius)
  - External measures of clustering quality (e.g., class labels)
- Specialized approaches for text streams, high-dimensional streams.

Resources

- C. Aggarwal, Data Streams: Models and Algorithms, Springer, 2007.
- [AggEtAl03] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu: A framework for clustering evolving data streams. VLDB, 2003.
- [EsterEtAl98] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, X. Xu, „Incremental Clustering for Mining in a Data Warehousing Environment", VLDB 1998.
- J. Gama, Knowledge Discovery from Data Streams, Chapman and Hall/CRC, 2010.
- [CaoEtAl06] F. Cao, M. Ester, W. Qian, A. Zhou: Density-Based Clustering over an Evolving Data Stream with Noise. SDM, 2006.
- [CheTu07] Y. Chen, L. Tu: Density-Based Clustering for Real-Time Stream Data. KDD, 2007.
- F. Farnstrom, J. Lewis, C. Elkan: Scalability for clustering algorithms revisited. ACM SIGKDD Explorations Newsletter 2(1):51-57, 2000.
- S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O' Callaghan: Clustering data streams: Theory and practice. IEEE TKDE 15(3):515–528, 2003.
- [OCaEtAl02] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, R. Motwani: Streaming-Data Algorithms for High-Quality Clustering. ICDE, 2002.
- www.utdallas.edu/~lkhan/Spring2008G/Charu03.ppt

Classification



- Screw
- Nails
- Paper clips

Training data

- New object

Task:

Learn from the already classified training data, the rules to classify new objects based on their characteristics.

The result attribute (class variable) is nominal (categorical)

The (batch) classification process

Model construction

Training data

NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Predictive attributes

Class attribute

Classification Algorithm

Classifier (Model)

IF rank = 'professor' OR years > 6 THEN tenured = 'yes'

IF (rank!='professor') AND (years < 6) THEN tenured = 'no'

Prediction

Unseen data

NAME	RANK	YEARS	TENURED
Jeff	Professor	4	?
Patrick	Assistant Professor	8	?
Maria	Assistant Professor	2	?

Tenured? Yes

Tenured? ?

Tenured? ?

Stream vs batch classification 1/2

- So far, classification as a batch/ static task
  - The whole training set is given as input to the algorithm for the generation of the classification model.
  - The classification model is static (does not change)
  - When the performance of the model drops, a new model is generated from scratch over a new training set.

- But, in a dynamic environment data change continuously
  - Batch model re-generation is not appropriate/sufficient anymore

Stream vs batch classification 2/2

- Need for new classification algorithms that
  - have the ability to *incorporate new data*
  - deal with non-stationary data generation processes (concept drift)
    - Ability to *remove obsolete data*
  - subject to:
    - resource constraints (processing time, memory)
    - single scan of the data (one look, no random access)

Non-stationary data distribution → Concept drift

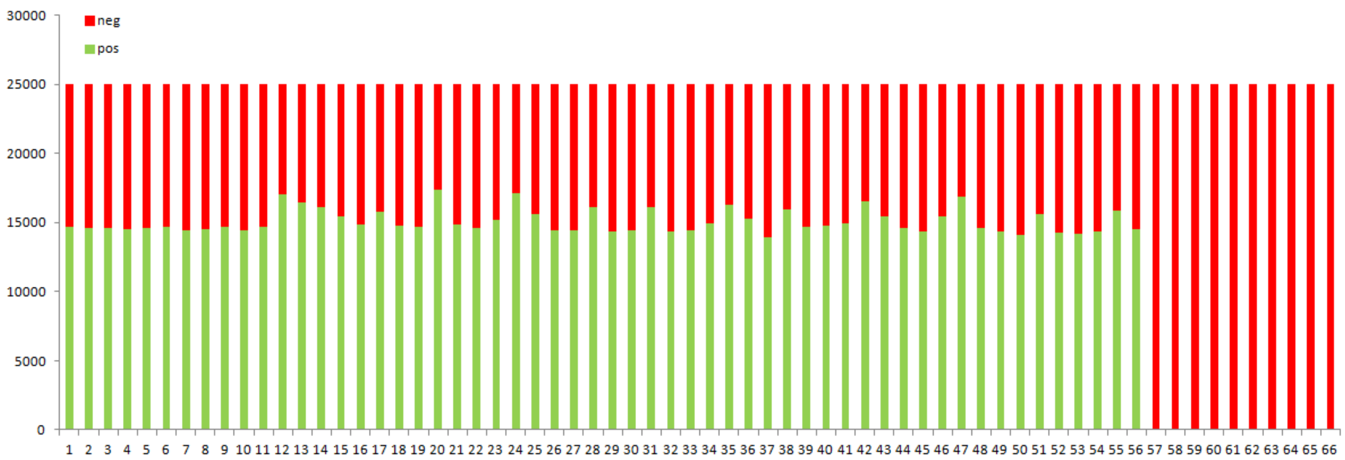
- In dynamically changing and non-stationary environments, the data distribution might change over time yielding the phenomenon of concept drift
- Different forms of change:
  - The input data characteristics might change over time
  - The relation between the input data and the target variable might change over time
- Concept drift between t_0 and t_1 can be defined as

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y).$$

  - P(X,y): the joint distribution between X and y
- According to the Bayesian Decision Theory: $p(y|X) = \frac{p(y)p(X|y)}{p(X)}$.
- So, changes in data can be characterized as changes in:
  - The prior probabilities of the classes p(y)
  - The class conditional probabilities p(X|y).
  - The posterior p(y|X) might change

Example: Evolving class priors

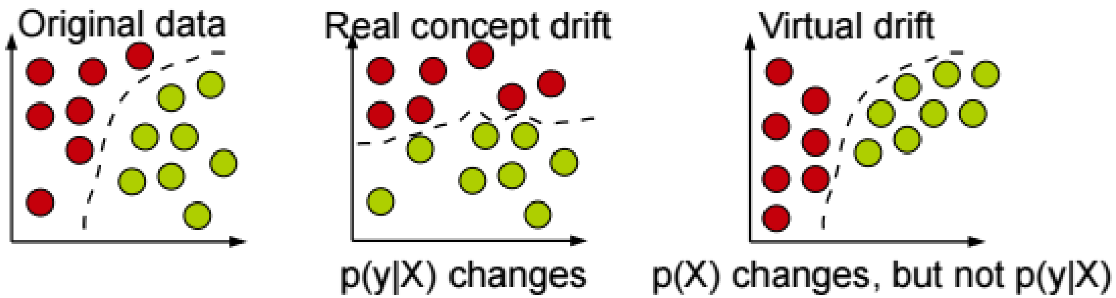
- E.g., evolving class distribution
  - The class distribution might change over time
  - Example: Twitter sentiment dataset
    - o 1.600.000 instances split in 67 chunks of 25.000 tweets per chunk
    - o Balanced dataset (800.000 positive, 800.000 negative tweets)
    - o The distribution of the classes changes over time
    - o Dataset online at: *https://sites.google.com/site/twittersentimenthelp/for-researchers*



Evolving class distribution [Sinelnikova12]

Real vs virtual drift

- Real concept drift
  - Refers to changes in p(y|X). Such changes can happen with or without change in p(X).
  - E.g., “I am not interested in tech posts anymore”
- Virtual concept drift
  - If the p(X) changes without affecting p(y|X)

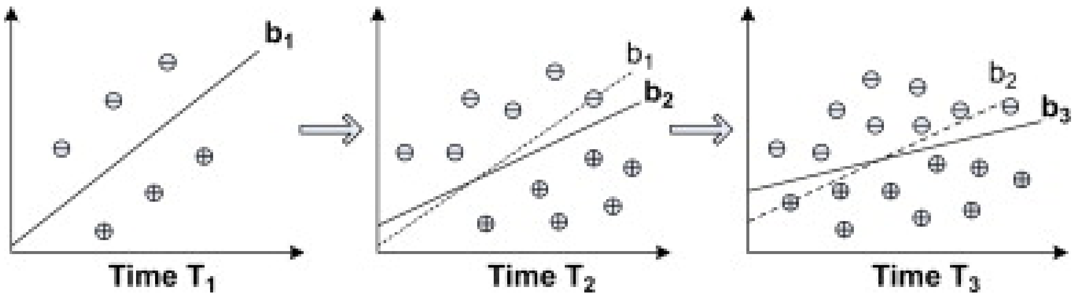


Source: [GamaETAl13]

  - Drifts (and shifts)
    - Drift more associated to gradual changes
    - Shift refers to abrupt changes

Model adaptation

- As data evolve over time, the classifier should be updated to “reflect” the evolving data
  - Update by incorporating new data
  - Update by forgetting obsolete data



The classification boundary gradually drifts from b_1 (at T_1) to b_2 (at T_2) and finally to b_3 (at T_3).
(Source: A framework for application-driven classification of data streams, Zhang et al, Journal Neurocomputing 2012)

Data stream classifiers

- The batch classification problem:
  - Given a finite training set D={(x,y)} , where y={y_1, y_2, …, y_k}, |D|=n, find a function y=f(x) that can predict the y value for an unseen instance x
- The data stream classification problem:
  - Given an infinite sequence of pairs of the form (x,y) where y={y_1, y_2, …, y_k}, find a function y=f(x) that can predict the y value for an unseen instance x
    - the label y of x is not available during the prediction time
    - but it is available shortly after for model update

Supervised scenario

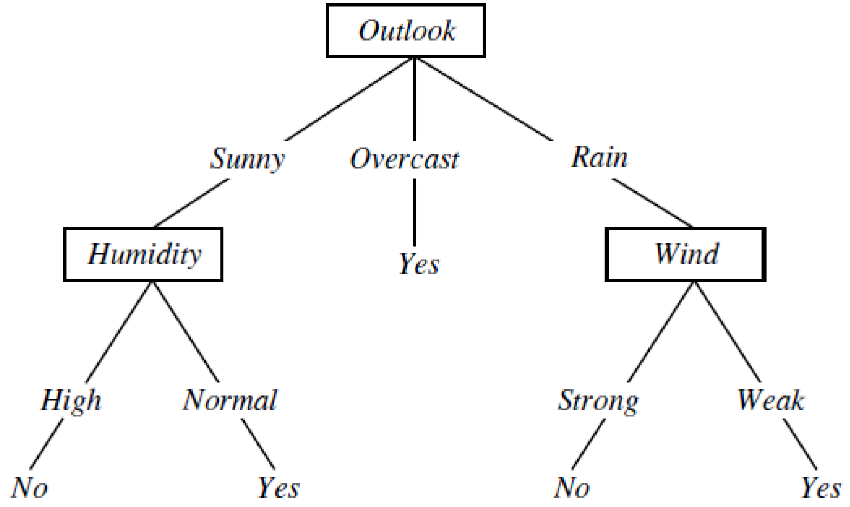
- Example applications:
  - Fraud detection in credit card transactions
  - Churn prediction in a telecommunication company
  - Sentiment classification in the Twitter stream
  - Topic classification in a news aggregation site, e.g. Google news
  - …

(Batch) Decision Trees (DTs)

- Training set: D = {(x,y)}
  - predictive attributes: $x=<x_1, x_2, \ldots, x_d>$
  - class attribute: $y=\{y_1, y_2, \ldots, y_k\}$
- Goal: find y=f(x)
- Decision tree model
  - nodes contain tests on the predictive attributes
  - leaves contain predictions on the class attribute

Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



(Batch) DTs: Selecting the splitting attribute

- Basic algorithm (ID3, Quinlan 1986)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root node

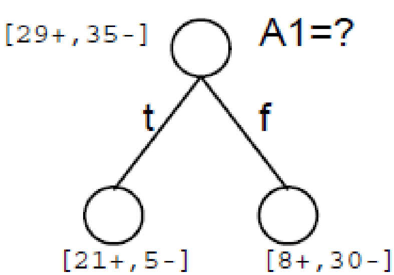
Main loop:

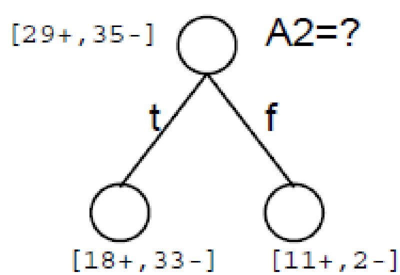
1. $A \leftarrow$ the "best" decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A, create new descendant of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

  - But, which attribute is the best?

Attribute selection measures:
- Information gain
- Gain ratio
- Gini index

(check Lecture 4, KDD I)





Goal: select the most "useful" attribute
- i.e., the one resulting in the purest partitioning

(Batch) DTs: Information gain

- Used in ID3
- It uses entropy, a measure of pureness of the data
- The information gain Gain(S,A) of an attribute A relative to a collection of examples S measures the gain reduction in S due to splitting on A:

$$Gain(S,A) = \boxed{Entropy(S)} - \boxed{\sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)}$$

Before splitting

After splitting on A

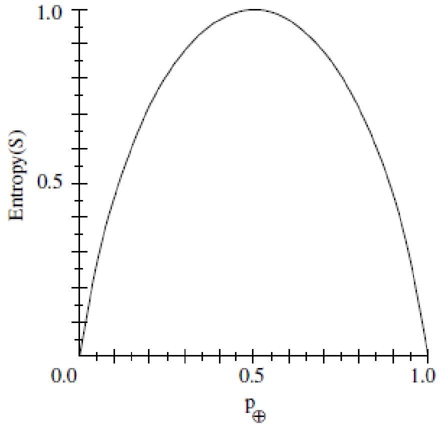
- Gain measures the expected reduction in entropy due to splitting on A

- The attribute with the higher entropy reduction is chosen

(Batch) DTs: Entropy

- Let S be a collection of positive and negative examples for a binary classification problem, C={+, -}.
- p_+: the percentage of positive examples in S
- p_-: the percentage of negative examples in S
- Entropy measures the impurity of S:

$$Entropy(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$



- Examples :
  - Let S: [9+,5-] $Entropy(S) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$
  - Let S: [7+,7-] $Entropy(S) = -\frac{7}{14}\log_2(\frac{7}{14}) - \frac{7}{14}\log_2(\frac{7}{14}) = 1$
  - Let S: [14+,0-] $Entropy(S) = -\frac{14}{14}\log_2(\frac{14}{14}) - \frac{0}{14}\log_2(\frac{0}{14}) = 0$

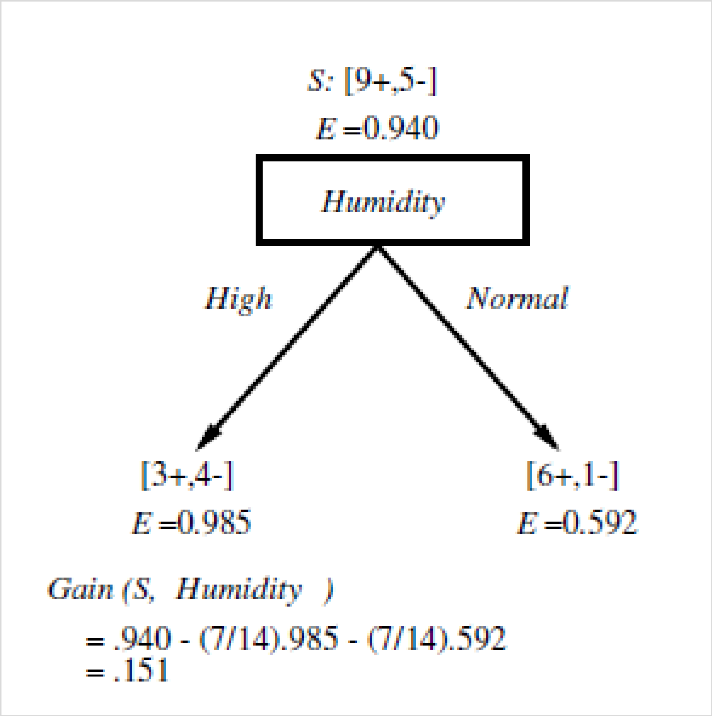
in the general case
(*k-classification problem)*

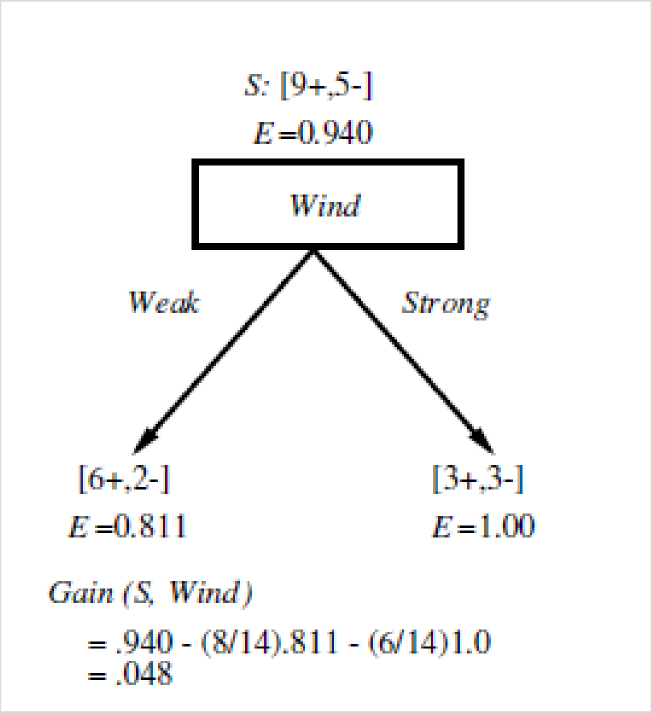
$$Entropy(S) = \sum_{i=1}^{k} -p_i \log_2(p_i)$$

- Entropy = 0, when all members belong to the same class
- Entropy = 1, when there is an equal number of positive and negative examples

(Batch) DTs: Information gain example

- Which attribute to choose next?





From batch to stream DT induction

- Thus far, in order to decide on which attribute to use for splitting in a node (essential operation for building a DT), we need to have all the training set instances resulting in this node.
- But, in a data stream environment
  - The stream is infinite
  - We cant wait for ever in a node
- Can we make a valid decision based on some data?
  - Hoeffding Tree or Very Fast Decision Tree (VFDT) [DomingosHulten00]

Hoeffding Tree [DomingosHulten00]

- Idea: In order to pick the best split attribute for a node, it may be sufficient to consider only a small subset of the training examples that pass through that node.
  - No need to look at the whole dataset
  - (which is infinite in case of streams)

- Problem: How many instances are necessary?
  - Use the Hoeffding bound!

The Hoeffding bound

- Consider a real-valued random variable r whose range is R
  - e.g., for a probability the range is 1
  - for information gain the range is $\log_2(c)$, where c is the number of classes
- Suppose we have n independent observations of r and we compute its mean $\bar{r}$
- The Hoeffding bound states that with confidence $1\text{-}\delta$ the true mean of the variable, μ_r, is at least $\bar{r}-\varepsilon$, i.e., $P(\mu_r \geq \bar{r}-\varepsilon) = 1-\delta$
- The ε is given by:

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$$

- This bound holds true regardless of the distribution generating the values, and depends only on the range of values, number of observations and desired confidence.
  - A disadvantage of being so general is that it is more conservative than a distribution-dependent bound

Using the Hoeffding bound to select the best split at a node

- Let *G()* be the heuristic measure for choosing the split attribute at a node
- After seeing *n* instances at this node, let
  - X_a : be the attribute with the highest observed *G()*
  - X_b : be the attribute with the second-highest observed *G()*
- $\overline{\Delta G} = \overline{G}(X_a) - \overline{G}(X_b) \geq 0$ the difference between the 2 best attributes
- $\overline{\Delta G}$ is the random variable being estimated by the Hoeffding bound
- Given a desired δ, if $\overline{\Delta G} > \varepsilon$ after seeing n instances at the node
  - the Hoeffding bound guarantees that with probability $1-\delta$, $\Delta G \geq \overline{\Delta G} - \varepsilon > 0$.
  - Therefore we can confidently choose X_a for splitting at this node
- Otherwise, i.e., if $\overline{\Delta G} < \varepsilon$, the sample size is not enough for a stable decision.
  - With R and δ fixed, the only variable left to change ε is n
  - We need to extend the sample by seeing more instances, until ε becomes smaller than $\overline{\Delta G}$

Hoeffding Tree algorithm

- **Input:** δ desired probability level.
- **Output:** $\mathcal{T}$ A decision Tree
- **Init:** $\mathcal{T} \leftarrow$ Empty Leaf (Root)
- While (TRUE)
  - Read next Example
  - Propagate Example through the Tree from the Root till a leaf
  - Update Sufficient Statistics at leaf
  - If *leaf(#examples)mod* N_{min}*=0*
    - Evaluate the merit of each attribute
    - Let A_1 the best attribute and A_2 the second best
    - Let $\epsilon = \sqrt{R^2 ln(1/\delta)/(2n)}$
    - If $G(A_1) - G(A_2) > \epsilon$
    - Install a splitting test based on A_1
    - Expand the tree with two descendant leaves

Those needed by the heuristic evaluation function G()

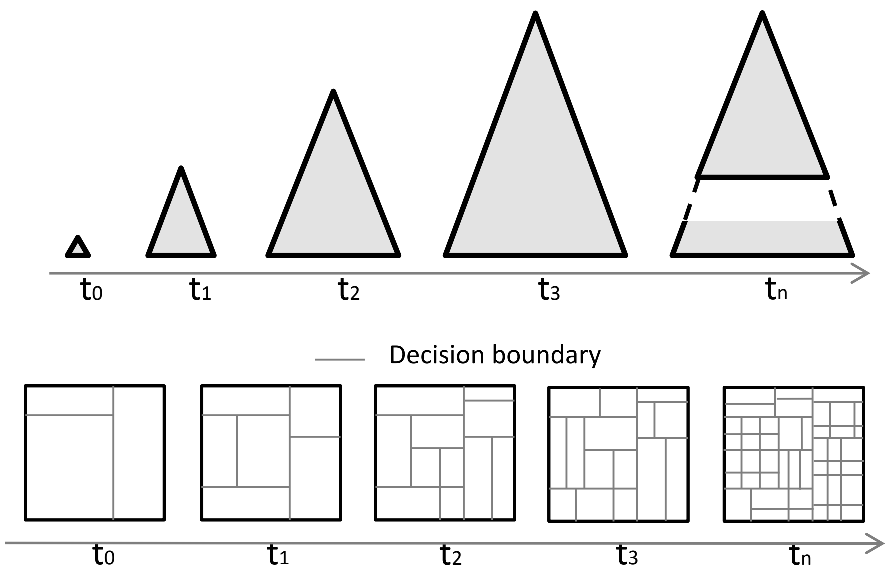
The evaluation of G() after each instance is very expensive.
→ Evaluate G() only after N_{min} instances have been observed since the last evaluation.

Hoeffding tree algorithm more details

- Breaking ties
  - When ≥2 attributes have very similar G's, potentially many examples will be required to decide between them with high confidence.
  - This is presumably wasteful, as it makes little difference which is chosen.
  - Break it by splitting on current best if $\Delta G<\varepsilon<\tau$, τ a user-specified threshold
- Grace period (MOA's term)
  - Recomputing G() after each instance is to expensive.
  - A user can specify # instances in a node that must be observed before attempting a new split

Hoeffding Tree overview

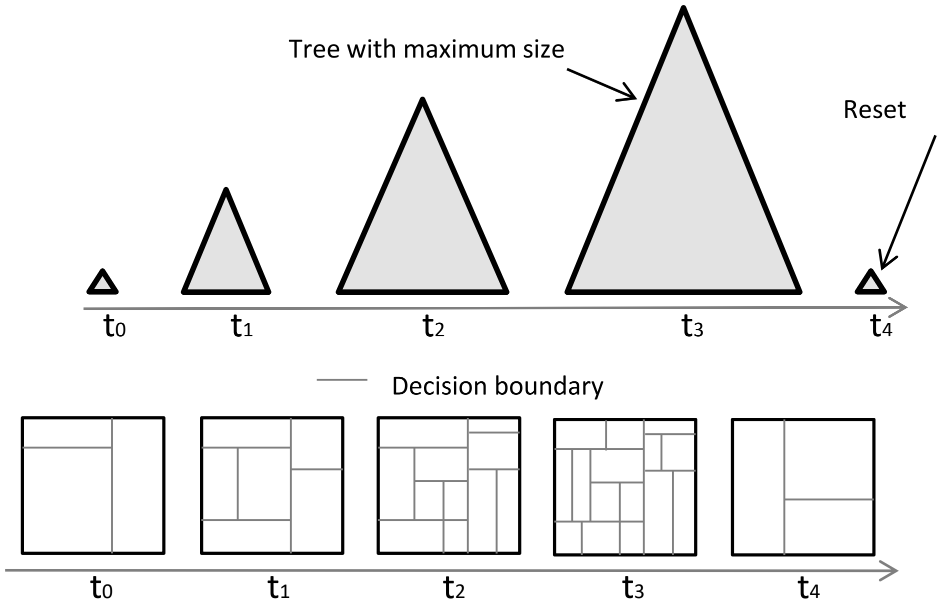
- The HT accommodates new instances from the stream
- But, doesn’t delete anything (doesn’t forget!)
- With time
  - The tree becomes more complex (overfitting is possible)
  - The historical data dominate its decisions (difficult to adapt to changes)



HT over time [Mahmud15]

Adaptive Size Hoeffding Tree (ASHT) [BifetEtAl09]

- Introduces a maximum size (#splitting nodes) bound
- When the limit is reached, the tree is reset
  - Test for the limit, after node's split

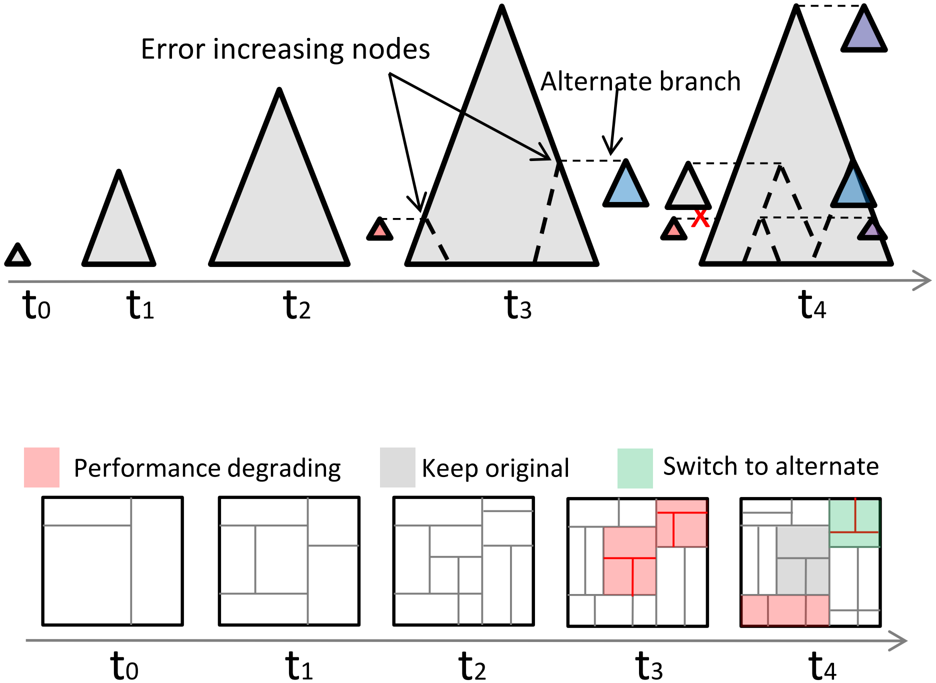


ASHT over time [Mahmud15]

- The tree forgets
  - but, due to the reset, it looses all information learned thus far

Concept-Adapting Hoeffding Tree [HultenEtAl01]

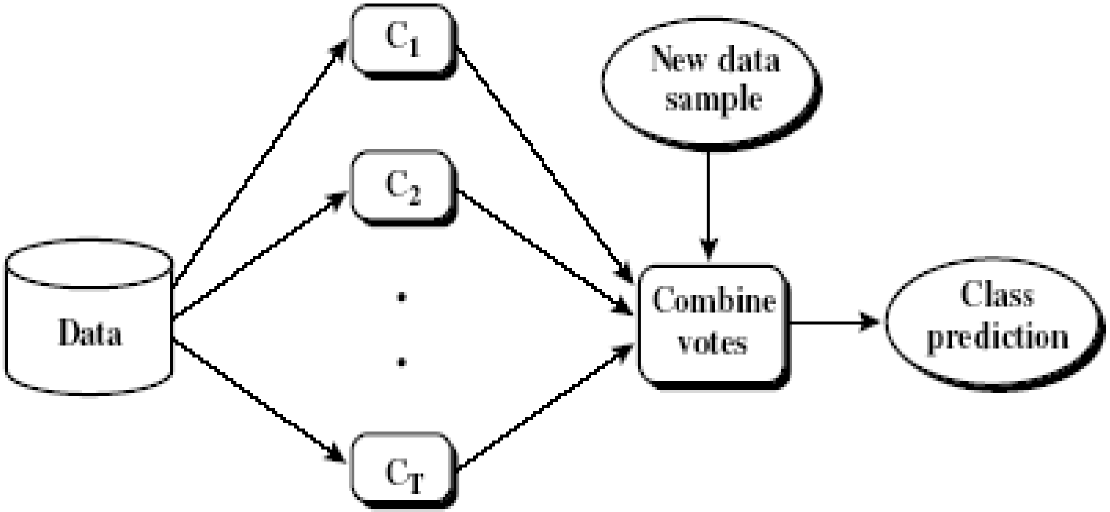
- Starts maintaining an alternate sub-tree when the performance of a node decays
- When the new sub-tree starts performing better, it replaces the original one
- If original sub-tree keeps performing better, the alternate sub-tree is deleted and the original one is kept



AdaHT over time [Mahmud15]

Ensemble of classifiers

- Idea:
  - Instead of a single model, use a combination of models to increase accuracy
  - Combine a series of T learned models, M_1, M_2, …, M_T, with the aim of creating an improved model M*
  - To predict the class of previously unseen records, aggregate the predictions of the ensemble

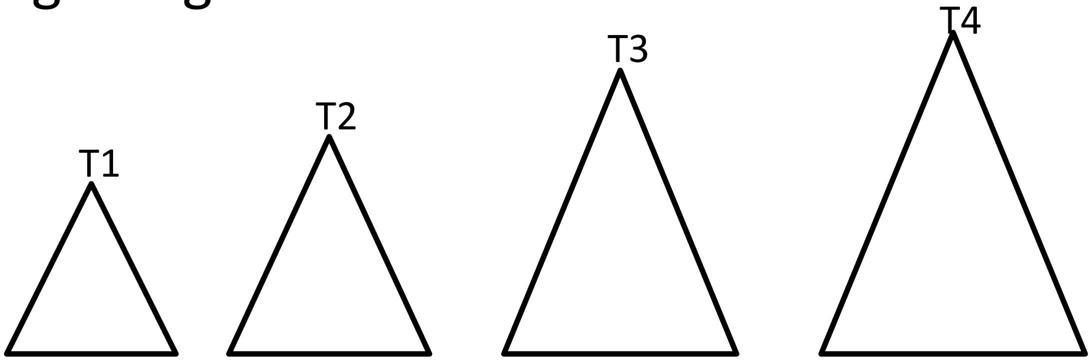


Many methods

- Bagging
  - Generate training samples by sampling with replacement (bootstrap)
  - Learn one model at each sample
- Boosting
  - At each round, increase the weights of misclassified examples
- Stacking
  - Apply multiple base learners
  - Meta learner input = base learner predictions

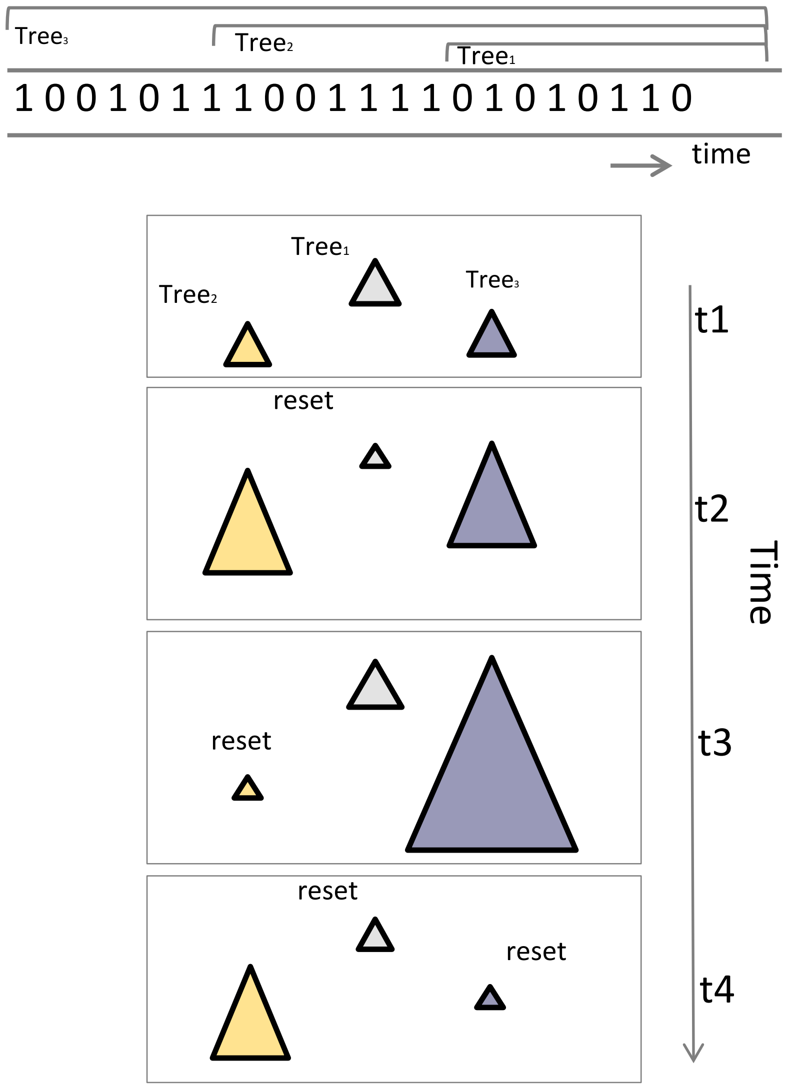
Ensemble of Adaptive Size Hoeffding Trees (ASHT) [BifetEtAl09] 1/2

- Bagging using ASHTs of different sizes



  - Smaller trees adapt more quickly to changes
  - Larger trees perform better during periods with no or little change
  - The max allowed size for the nth ASHT tree is twice the max allowed size for the (n-1)th tree.
  - Each tree has a weight proportional to the inverse of the square of its error
  - The goal is to increase bagging performance by tree diversity

Ensemble of Adaptive Size Hoeffding Trees (ASHT) [BifetEtAl09] 2/2



Hoeffding Tree family overview

- All HT, AdaHT, ASHT accommodate new instances from the stream
- HT does not forget
- ASHT forgets by resetting the tree once its size reaches its limit
- AdaHT forgets my replacing sub-trees with new ones
- Bagging ASHT uses varying size trees that respond differently to change

Data stream classification: overview

- Extending traditional classification methods for data streams implies that
  - They should accommodate new instances
  - They should forget obsolete instances
- Typically, all methods incorporate new instances from the model
- They differ mainly on how do they forget
  - No forgetting, sliding window forgetting, damped window forgetting,…
- and which part of the model is affected
  - Complete model reset, partial reset, …
- So far, we focused on fully-supervised learning and we assumed availability of class labels for all stream instances
  - Semi-supervised learning
  - Active learning
- Dealing with class imbalances, rare-classes
- Dealing with dynamic feature spaces

Resources

- C. Aggarwal, Data Streams: Models and Algorithms, Springer, 2007.
- J. Gama, Knowledge Discovery from Data Streams, Chapman and Hall/CRC, 2010.
- [GameEtAl13] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia, A Survey on Concept Drift Adaptation, *ACM Computing Surveys* 46(4), 2014.
- [DomingosHulten00] Pedro Domingos and Geoff Hulten, Mining high-speed data streams, KDD, 2000.
- [BifetEtAl09] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Ricard Gavaldà, Improving Adaptive Bagging Methods for Evolving Data Streams, AML, 2009.
- [HultenEtAl01] Geoff Hulten, Laurie Spencer, Pedro Domingos, Mining time-changing data streams, KDD, 2001.
- [WagnerEtAl15] S. Wagner, M. Zimmermann, E. Ntoutsi, M. Spiliopoulou, Ageing-based Multinomial Naive Bayes Classifiers over Opinionated Data Streams. ECML PKDD, 2015.