Freie Universität Berlin

Lecture: Machine Learning for Data Science

Winter semester 2021/22

Lectures 22-23: Velocity (stream classification)

Prof. Dr. Eirini Ntoutsi

Outline

- Batch learning assumptions
- Motivation
- Data streams
- Data stream classification basics
- Data stream classifiers: Family of decision tree stream classifiers
- Data stream classifiers: evaluation aspects
- Things you should know from this lecture & reading material

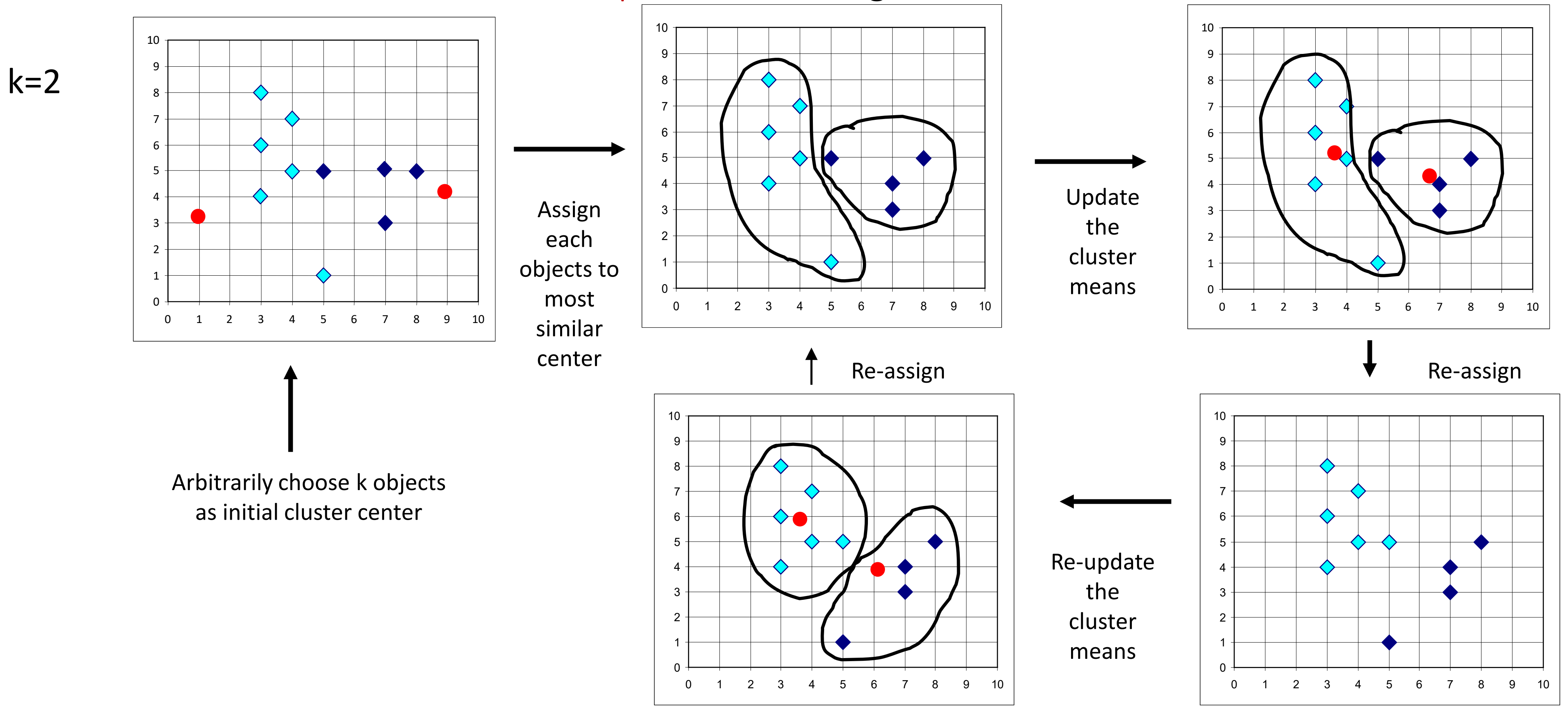
Traditionally, batch learning

- Most of the machine learning algorithms focus on batch learning.
- Batch learning:
  - The complete dataset is available to the learning algorithm
  - Data instances can be accessed multiple times

- This hold for many learners and for different learning task
  - e.g., for clustering: k-Means, DBSCAN, …
  - e.g., for classification: decision trees, Naïve Bayes, NNs…

Example: batch k-Means (see also lecture 10)

- The complete dataset is given as input to the algorithm
- The dataset is accessed multiple times during the iterations



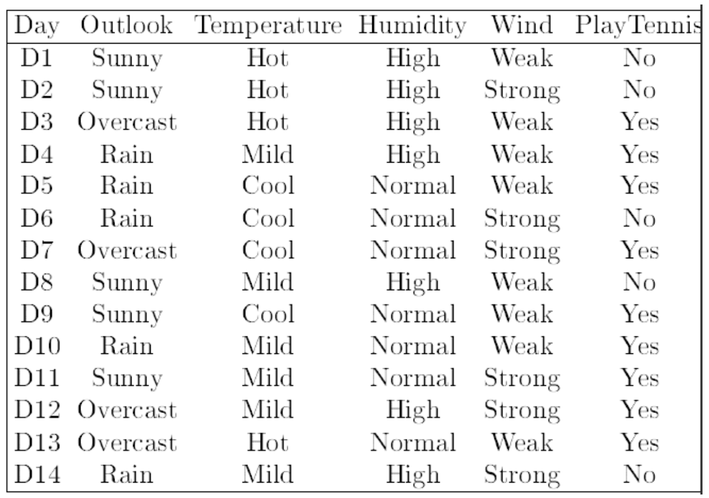
k=2
Assign each objects to most similar center
Update the cluster means
Re-assign
Re-update the cluster means
Re-assign
Arbitrarily choose k objects as initial cluster center

Example: batch ID3 (see also Lectures 3-4)

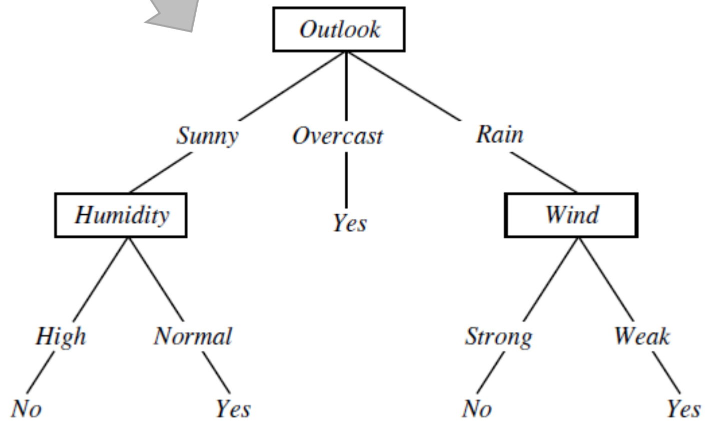
- The complete dataset is given as input to the algorithm
- The dataset is accessed multiple times during the iterations

- At start, all the training examples are at the root node.
- The best attribute is selected and used as the splitting attribute at the root
  - For each possible value of the test attribute, a descendant of the root node is created and *the* instances are mapped to the appropriate descendant node.
- Repeat the splitting attribute decision for each descendant node, so instances are partitioned recursively.
  - → we need access to the data again and again

Training set

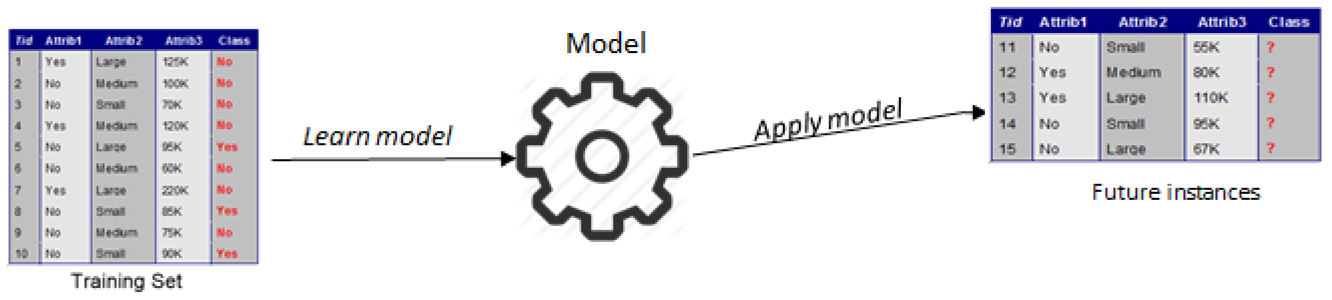


Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	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 learning assumes data stationarity

- Assumption: Data come from a stationary distribution
  - the distribution itself is unknown, but does not change
- This is reflected on i) how we train our models (batch learning mode)
  - The complete training set D is available all the time to the learning algorithm and can be accessed without any restrictions.



Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

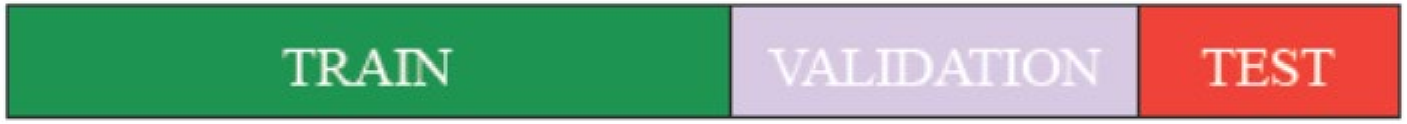
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Future instances

- This is reflected on ii) how we apply our models
  - The model (learned upon the fixed training set) is used for predicting future instances of the problem.

Batch learning assumes data stationarity

- This is reflected on iii) how we evaluate our models



  - Training set is a set of examples used for learning a model.
  - Validation set is a set of examples that can help to tune model parameters (e.g., selecting *k* in kNN).
  - Test set is used only to assess the performance of the final model and provide an estimation of the generalization error.
  - The assumption is that these sets come from the same distribution

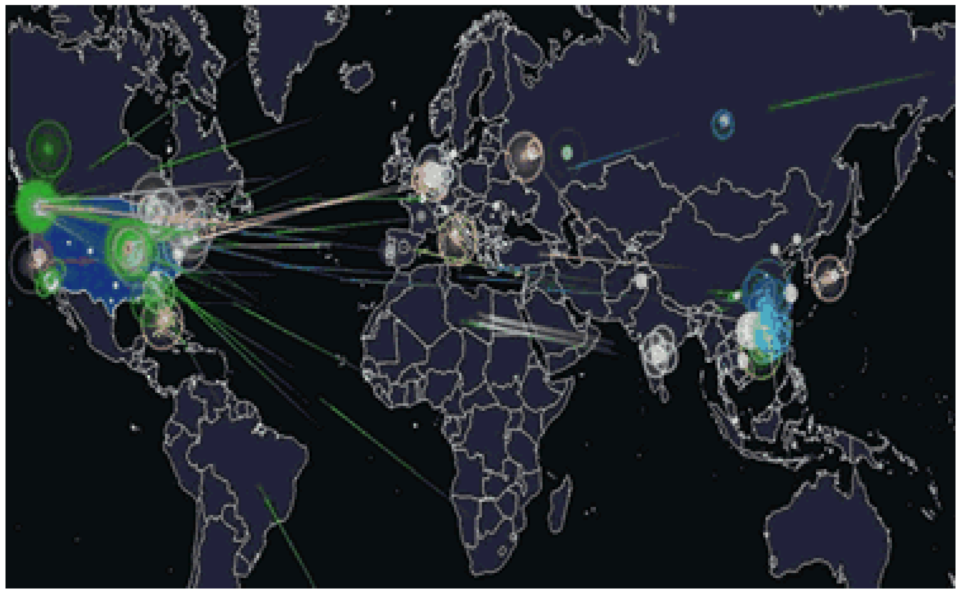
Outline

- Batch learning assumptions
- Motivation
- Data streams
- Data stream classification basics
- Data stream classifiers: Family of decision tree stream classifiers
- Data stream classifiers: evaluation aspects
- Things you should know from this lecture & reading material

Reality check: how realistic is the stationarity assumption?

- Most interesting applications nowadays come from dynamic environments where data are generated continuously over time (stream data)
- An example: Real-time network health monitoring
  - What are the profiles of normal connections?
  - What are the characteristics of attacks?

$\left.\begin{array}{l}\\ \\ \end{array}\right\}$ The characteristics of the (normal, attack) connections might change with time



Source: http://www.networkworld.com/article/2366962/microsoft-subnet/spellbound-by-maps-tracking-hack-attacks-and-cyber-threats-in-real-time.htmls

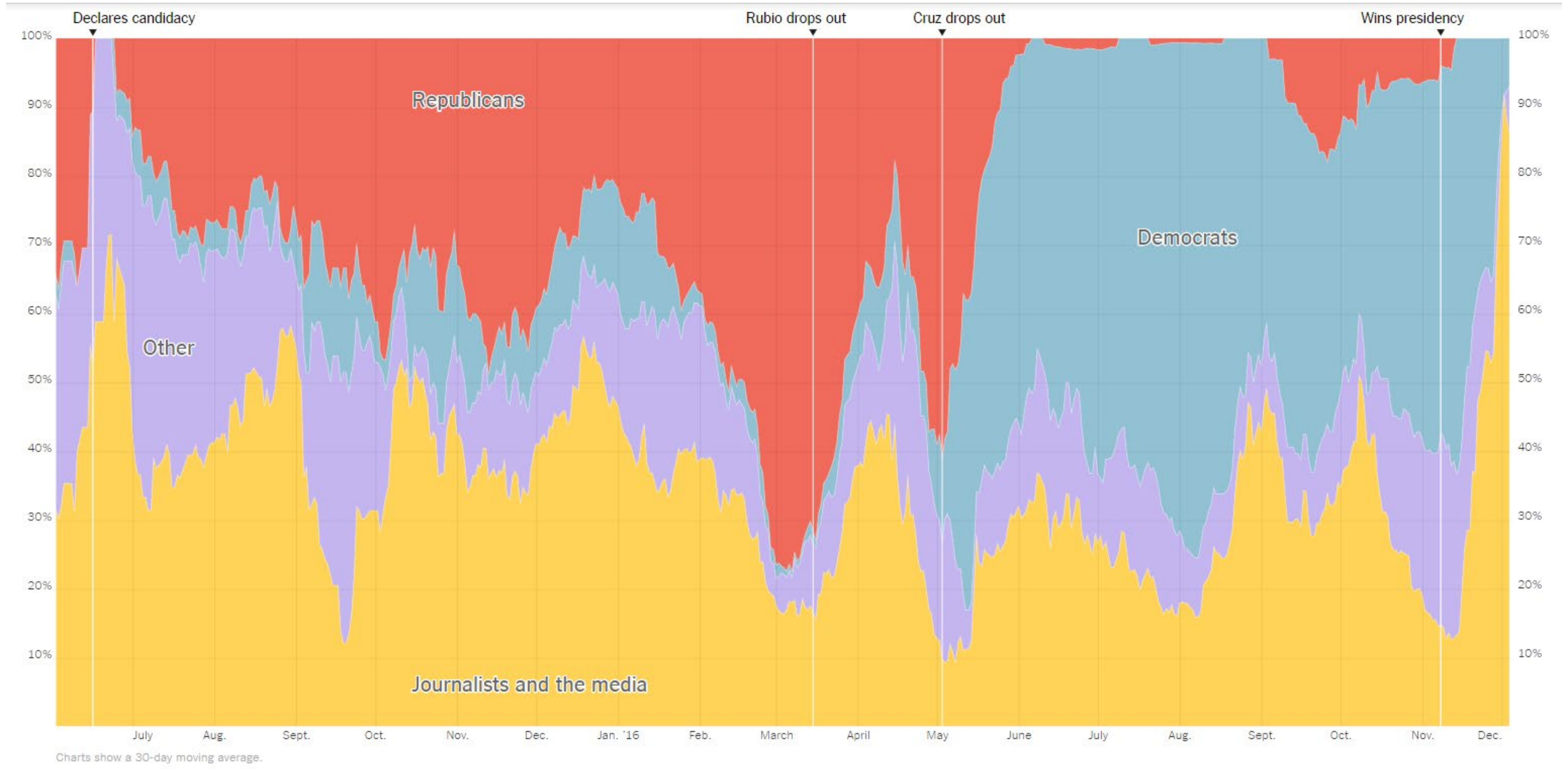
Example dataset: Network monitoring

time	duration	protocol_type	service	flag	src_bytes	dst_bytes	…	class
t_1	0	Tcp	http	SF	181	5450	…	normal
t_2	0	Tcp	http	SF	239	486	…	normal
…	…	…	…	…	…	…	…	…
t_{7838}	0	Icmp	ecr_i	SF	1032	0	…	smurf
t_{7839}	0	Icmp	ecr_i	SF	1032	0	…	smurf
…	…	…	…	…	…	…	…	…
t_{70531}	0	Tcp	private	S0	0	0	…	neptune
t_{70532}	0	tcp	private	S0	0	0	…	neptune
…	…	…	…	…	…	…	…	…
t_{492310}	0	tcp	http	SF	244	7161	…	normal
t_{492311}	0	tcp	http	SF	258	9517	…	normal
…	…	…	…	…	…	…	…	…

- The dataset consists of TCP connection records of LAN network traffic managed by Lincoln Labs.
- A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol.
- Connections are described in terms of 42 features like duration, protocol_type, service, flag, src_bytes, dst_bytes etc,.
- Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. There are 4 main categories of attacks: DOS, R2L, U2R, PROBING and are further classified into attack types, like buffer-overflow, guess-passwd, neptune etc.
- Most of the connections in this dataset are normal, but occasionally there could be a burst of attacks at certain times.
- More on this dataset: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Example: Social streams (Twitter)

- Example from Donald Trump stream in Twitter
  - Check this NY times article: “How to Know What Donald Trump Really Cares About: Look at What He’s Insulting”

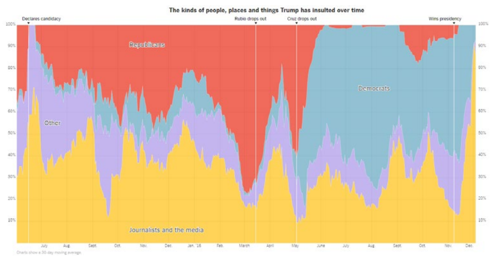


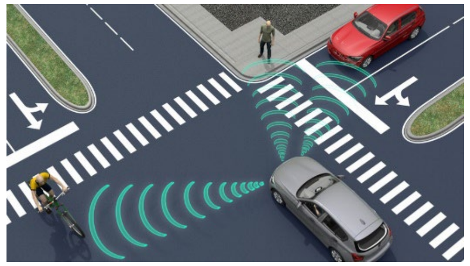
Source: http://www.nytimes.com/interactive/2016/12/06/upshot/how-to-know-what-donald-trump-really-cares-about-look-at-who-hes-insulting.html

Reality check: how realistic is the stationarity assumption?

- Plenty of applications
  - Social networks: topic identification, sentiment analysis, …
    - Topics might change
    - Sentiment towards certain topics might change
    - User preferences might change
    - Spammers adopt new attack strategies
  - Autonomous systems: navigation, obstacle avoidanc
    - The environment is dynamic (pedestrians, ca
    - The road network might change (extensions, p g, …)
  - Traffic management systems
    - Traffic changes as a result of events/constructions/weather …
    - People preferences change (electric scooters, …)

In all these cases, we deal with non-stationary/evolving data





Rethinking batch learning

- Most interesting applications nowadays come from dynamic environments where data are generated over time as a stream rather than in batch
- Batch learning is not sufficient anymore as
    - Data is never ending. What is the training set?
    - Multiple access to the data is not possible (e.g., telcos store call records for a certain period) or desirable (for efficiency issues, privacy issues …)
    - The data generation process is subject to changes over time
        - The patterns extracted upon such sort of data are also subject to change!
        - Machine Learning algorithms should be able to respond to underlying data changes

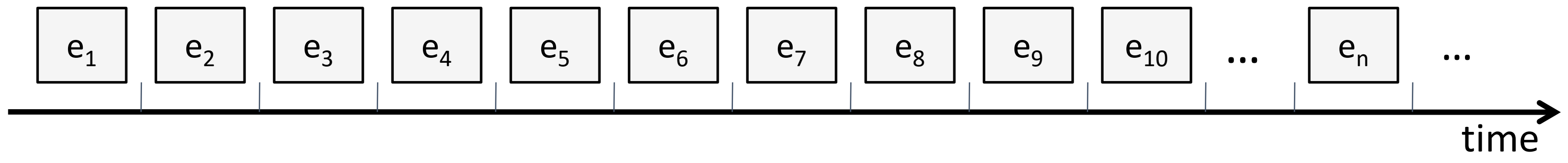
What do you think? What are your ideas w.r.t. these aspects?

Outline

- Batch learning assumptions
- Motivation
- Data streams
- Data stream classification basics
- Data stream classifiers: Family of decision tree stream classifiers
- Data stream classifiers: evaluation aspects
- Things you should know from this lecture & reading material

Data streams

- “A data stream is a potentially unbounded, ordered sequence of data items, which arrive continuously at high-speeds” Springer Encyclopedia of Machine Leaning, 2017



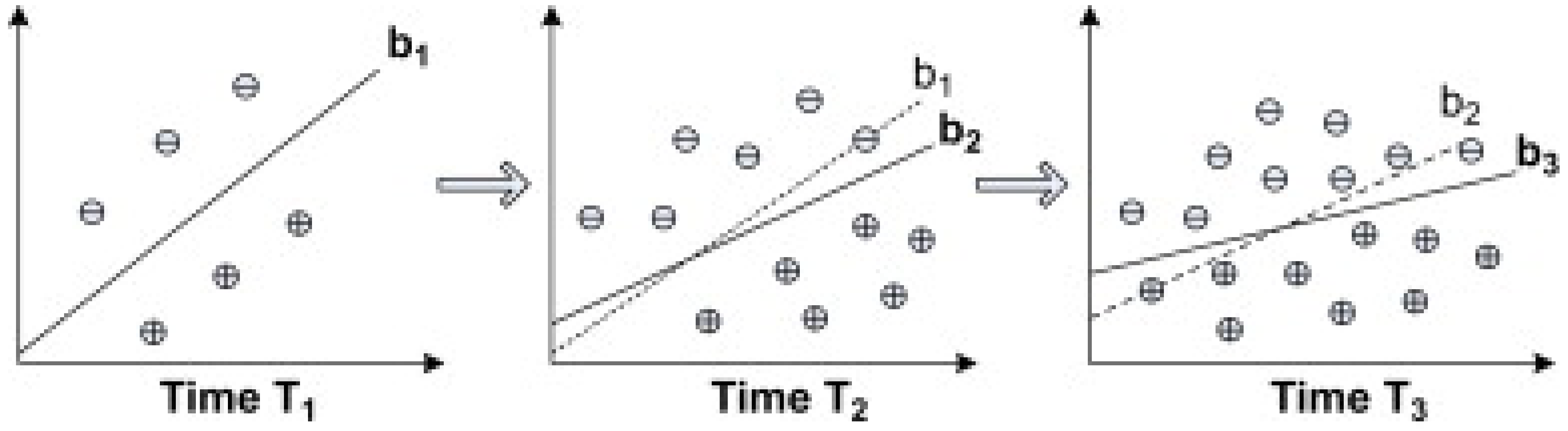
- Key characteristics
  - Huge volumes of continuous data, possibly infinite: Random access is expensive or un [illegible] e to e.g., privacy)
  - High arrival rate: response time matters
  - Non-stationary/ evolving data: Data evolve over time as new d [illegible] old data become obsolete/irrelevant

The fundamental for learning assumption that data are stationary is violated!!!

Why the stationarity violation is problematic?

Supervised learning case

- As data evolve with time, the classifier is becoming invalid/obsolete

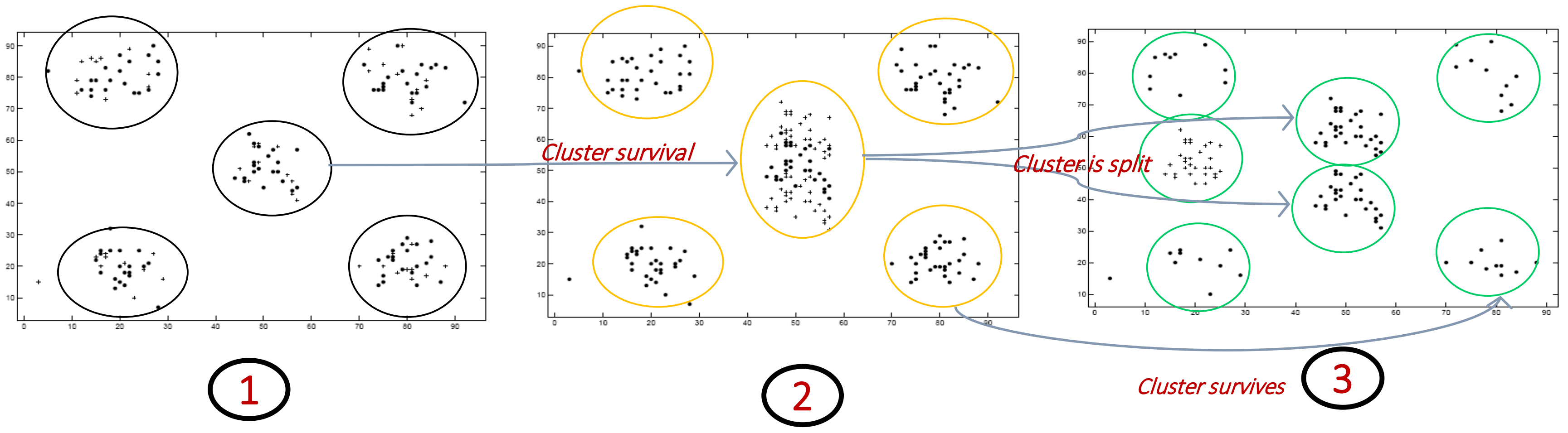


The classification boundary gradually drifts from b_1 (at T_1) to b_2 (at T_2) and finally to b_3 (at T_3).
(Source: A framework for application-driven classification of data streams, Zhang et al, Journal Neurocomputing 2012)

Why the stationarity violation is problematic?

Unsupervised learning case

- As data evolve with time, the clustering is becoming invalid/obsolete
- External changes: the relationship of a cluster to the other clusters might change, e.g., cluster survival, split, merge, appearance, disappearance
- Internal changes: the description of a cluster might change both externally (i.e., cluster members) and internally (cluster properties)



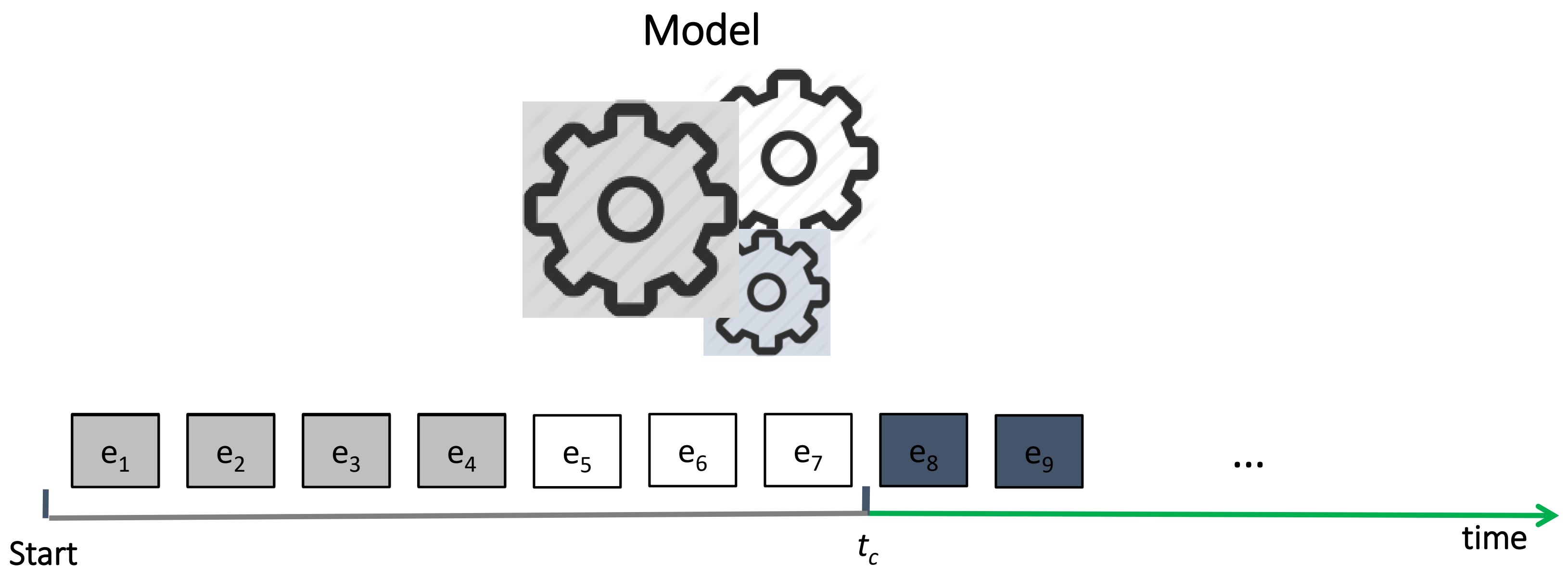
Source: The MONIC framework, Spiliopoulou et al, KDD06

Requirements for stream learning

- Need for new learning algorithms that
  - have the ability to incorporate new data (incremental models)
  - deal with non-stationary data generation processes
    - Ability to discard obsolete data (or, obsolete (parts of the) model) (data ageing/ forgetting)
- subject to:
  - resource constraints (processing time, memory)
  - single scan of the data (one look, no random access)

Model adaptation in stream learning

- Model adaptation is a result of 2-directional learning
  - By incorporating new instances in the model → incremental models
  - By eliminating outdated information from the model → forgetting/re-learning/unlearning, …



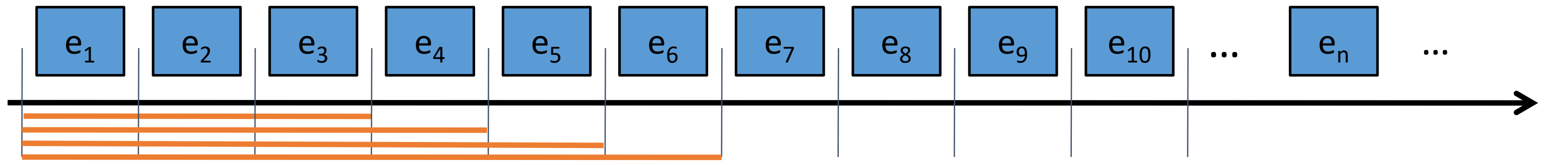
Forgetting is an important concept for streams

- Usually we are not interested in the whole history of the stream but only in the recent history
  - If I want to know my current customer profiles, I should probably look into my recent customer transactions
  - If I want to know the current topics in Twitter, I should probably focus on recent posts (last day, week,…)
- Different forgetting strategies
  - Forgetting data
    - Forgetting in windows, e.g., sliding window
    - Forgetting via ageing, e.g., exponential ageing
  - Forgetting (parts of a) model
    - re-learn locally vs globally

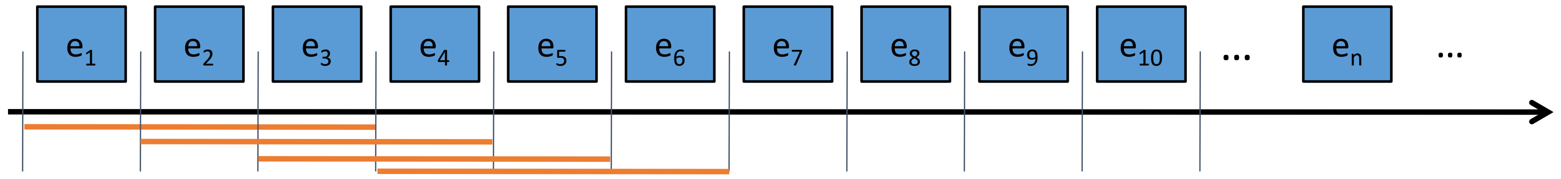
How can we “implement” forgetting? What are your ideas?

Forgetting data in windows

- Landmark window model:
  - Include all objects from a given landmark.
  - All points in the window have a weight w=1. Points outside the window have no effect (w=0).



- Sliding window model:
  - Remember only the n more recent entries, where n is the window size (e.g., number of instances, number of timestamps)
  - All points within the window have a weight w=1, for the rest: w=0.

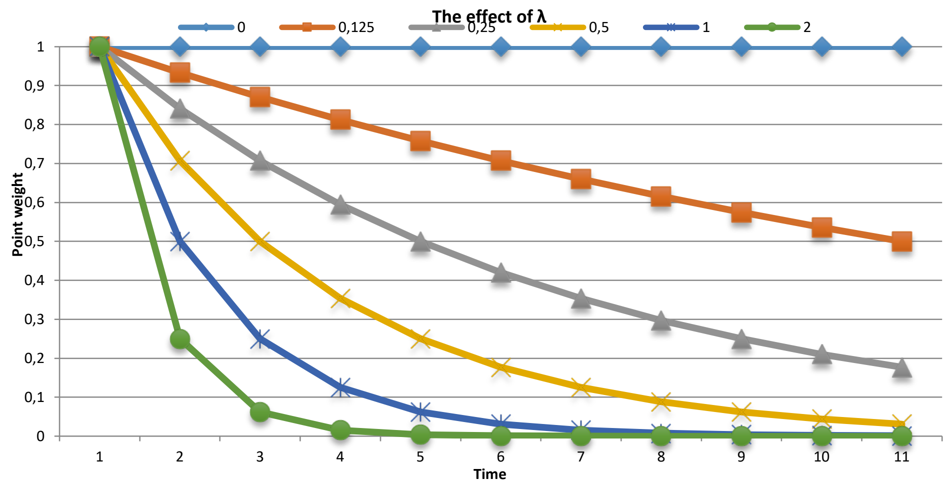


Forgetting data via ageing

- Damped window model:
  - Data are subject to ageing according to a fading function f(t), i.e., each point is assigned a weight that decreases with time t via f(t).
  - Past data is NOT completely discarded, but recent data are more influential (higher weights)
- A widely used fading function in temporal applications is the exponential fading function:

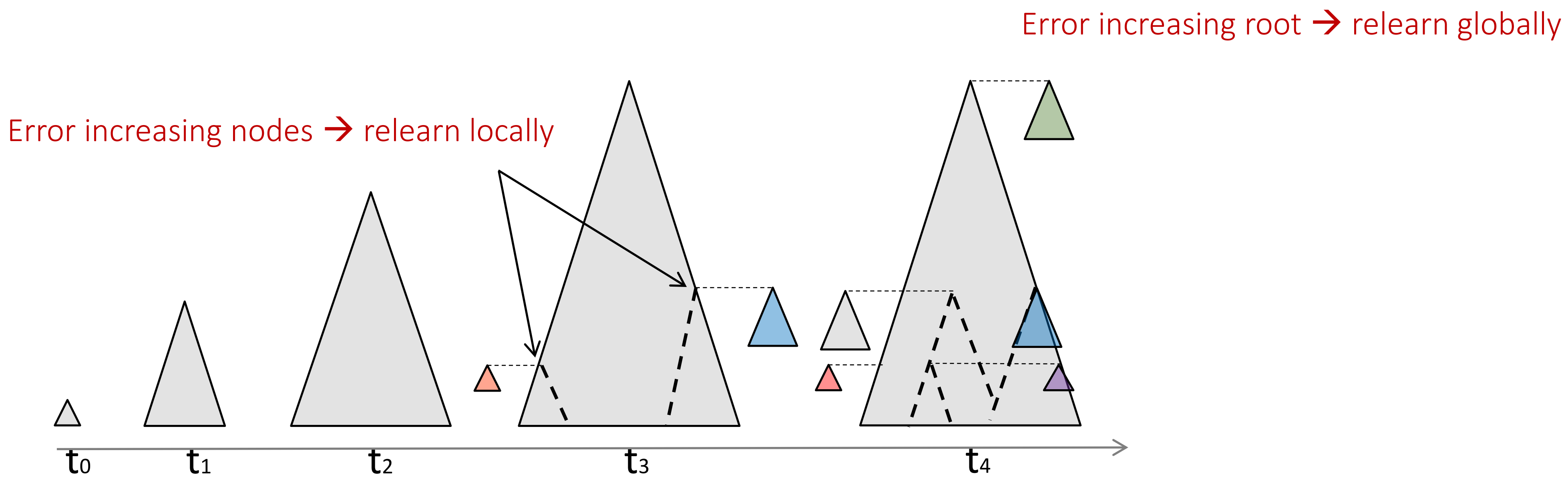
$$f(o,t) = e^{-\lambda(t-t_o)}$$

- *o*: is the instance/point
- *t*: is the current time
- t_o: is the arrival time of *o*
- λ (λ>0) is the decay rate that determines the importance of historical data
- t-t_o is the time elapsed since the occurrence of the point
- The higher λ, the lower the importance of old data



Forgetting (parts of) models

- Example of model-based forgetting for a decision tree model
  - Parts of the tree (subtrees) that do not perform well are re-learned (locally)
  - If the root is not performing well the whole tree is relearned (globally)


Error increasing root → relearn globally
Error increasing nodes → relearn locally
t0
t1
t2
t3
t4

Outline

- Batch learning assumptions
- Motivation
- Data streams
- Data stream classification basics
- Data stream classifiers: Family of decision tree stream classifiers
- Data stream classifiers: evaluation aspects
- Things you should know from this lecture & reading material

Concept drifts

- 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 (e.g., the Facebook population)
  - The relation between the input data and the target variable might change over time (e.g., when does a bank credit a loan?)
- Formally, concept drift: the joint distribution between the set of predictive variables X and target variable y, i.e., *P(X,y),* might change over the stream:

$$\exists X: P_t(X,y) \neq P_{t'}(X,y)$$

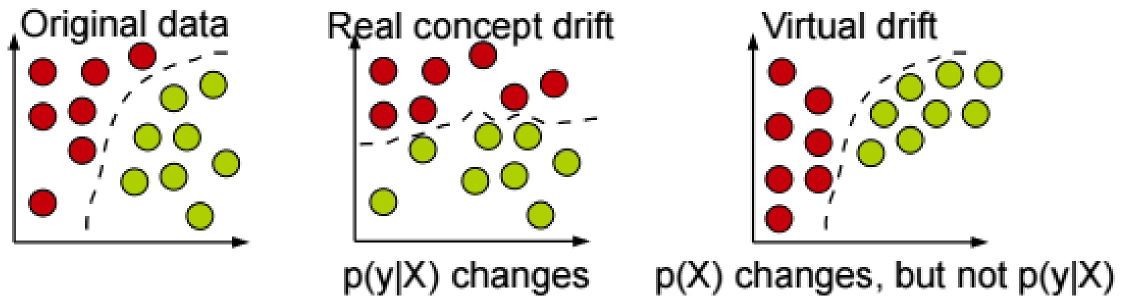
- According to the Bayesian Decision Theory a classification can be described as:

$$P_t(X,y) = P_t(X) * P_t(y|X)$$

- So, changes in data can be characterized as:
  - Changes in the marginal distribution P(X) $p(X) = \sum_{y=1}^{c} p(y)p(X|y)$
  - Changes in the posterior probabilities of classes P(y|X).

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

The stream classification task

- The batch classification problem:
  - Given a finite training set D={(x,y)} , where y={y1, y2, …, yk}, |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

(Fully) 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
  - …

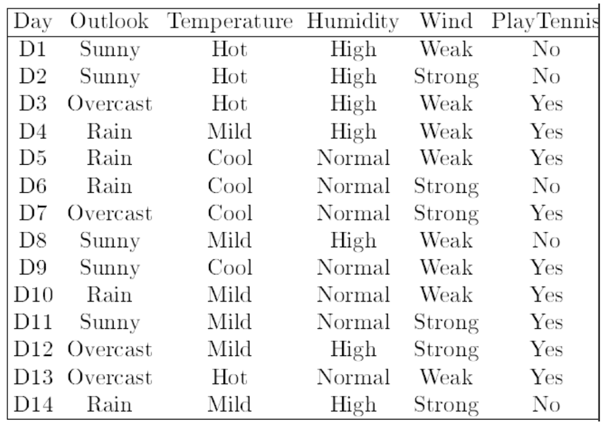
Outline

Data stream classifiers

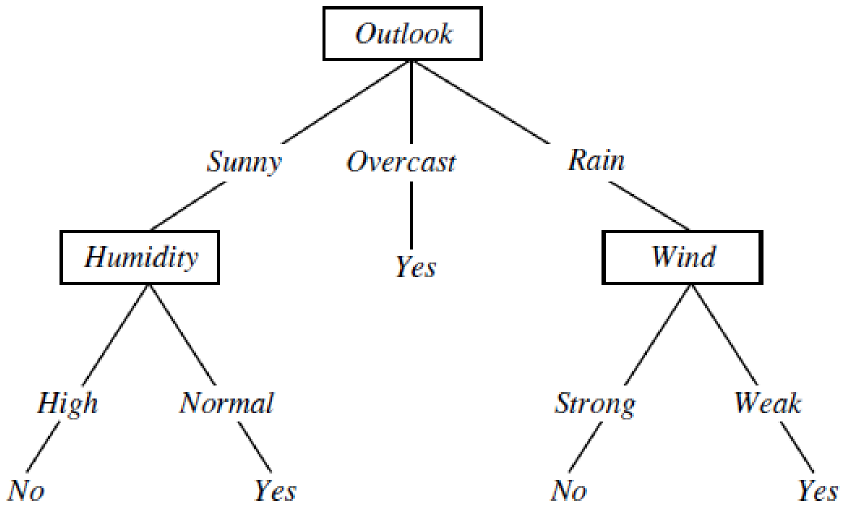
- Decision trees family of methods (this lecture)
- KNN family of methods (next lecture)
- Naïve Bayes family of methods
- SVM family of methods
- ANNs family of methods (next lecture)
- …

(Detour) (Batch) Decision Trees (DTs) – see also lectures 3-4

- 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 clas

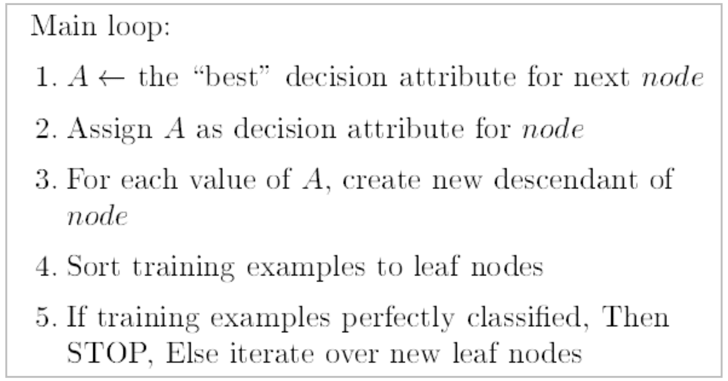


Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal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



(Detour) (Batch) DTs: Selecting the splitting attribute – see also lectures 3-4

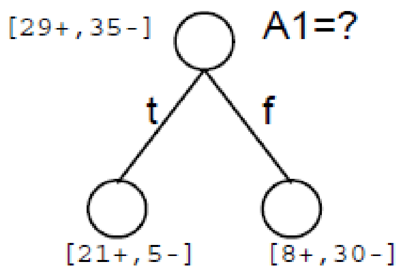
- Basic algorithm (ID3, Quinlan 1986)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root node

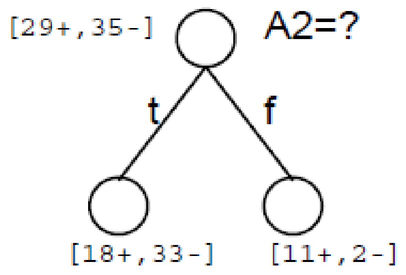


```
Main loop:
1. $A \leftarrow$ the "best" decision attribute for next node
2. Assign $A$ as decision attribute for node
3. For each value of $A$, create new descendant of
   node
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then
   STOP, Else iterate over new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

  - But, which attribute is the best?

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





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

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.
  - i.e., dataset in advance and multiply accesses to the dataset
- But, in a data stream environment
  - The stream is infinite
  - We cannot wait forever in a node → We will never take a splitting decision → No tree.

- Can we make a valid decision based on some data?
  - Hoeffding Tree (HT) 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, in any case, 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 with a range R
  - e.g., for a probability the range is 1
- Suppose we have n independent observations of this variable and we compute its mean r
- The Hoeffding bound states that with confidence $1-\delta$ the true mean of the variable, μ_r, is at least $r-\varepsilon$, i.e., $P(\mu_r \geq r-\varepsilon) = 1-\delta$
- 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 (R), number of observations (n) and desired confidence (δ).
  - 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

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

- Let *G() be the heuristic measure* for choosing the split attribute at a node
  - E.g., information gain
- 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()
- Let the difference between the 2 best attributes be: $\overline{\Delta G}=\overline{G(Xa)} - \overline{G(Xb)} \geq 0$
- ΔG is the random variable being estimated by the Hoeffding bound
- Given a desired confidence threshold *δ*, if $\overline{\Delta G} \geq \varepsilon$ after seeing *n* instances at the node
  - the Hoeffding bound guarantees that with probability 1-δ, $\Delta G \geq \overline{\Delta G}-\varepsilon>0$ or, in other words, $P(\Delta G \geq \overline{\Delta G}-\varepsilon) = 1-\delta$
  - 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
  - That is, we need to extend the sample by seeing more instances, until ε becomes smaller than $\overline{\Delta G}$

Hoeffding Tree algorithm pseudocode

```
Input: $\delta$ 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 $leaf(\#examples) > N_{min}$
        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 (referred to as gracePeriod).

Once a splitting decision is made, is not revised.
No backtracking!
See also lectures 3-4

Hoeffding tree algorithm more details

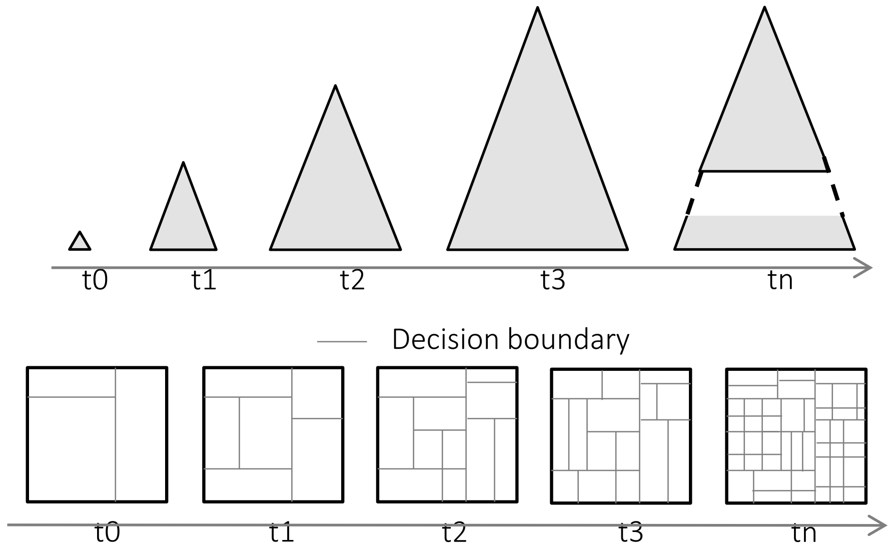
- Breaking ties
  - When ≥2 attributes have very similar G() values, potentially many examples will be required to decide between them with high confidence.
  - This is presumably wasteful, as it makes little difference which attribute is chosen.
  - Idea: Break it by splitting on current best if $\overline{\Delta G} < \varepsilon < \tau$, where τ is a user-specified tie threshold

- Grace period (N_{min})
  - Recomputing G() after each instance is too expensive.
  - A user can specify the number of instances in a node that must be observed before attempting a new split
  - Some tools like MOA call this grace period

MOA: the most popular open source framework for data stream mining (https://moa.cms.waikato.ac.nz/)

Hoeffding Tree overview

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

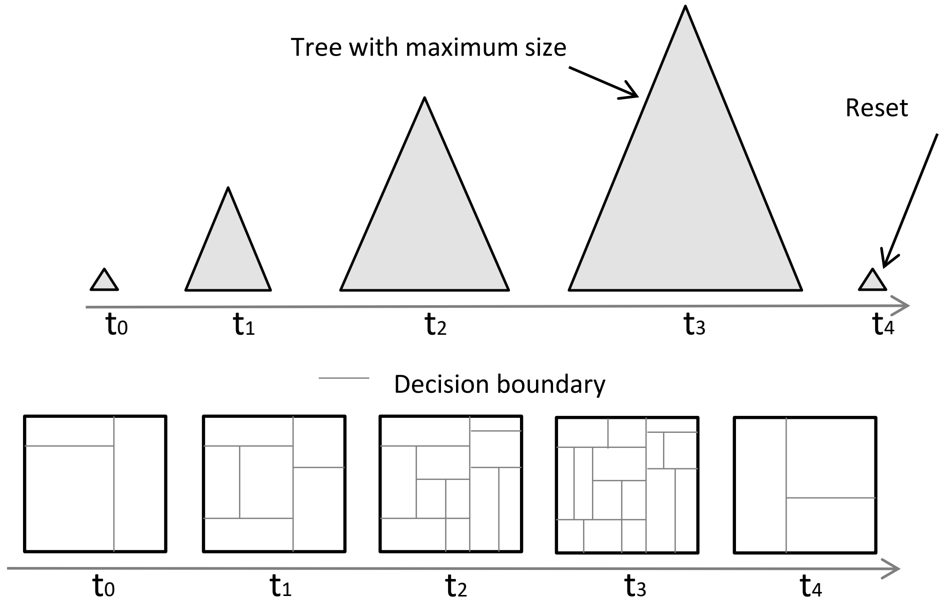


HT over time [Mahmud15]

How can we "implement" forgetting? What are your ideas?

Adaptive Size Hoeffding Tree (ASHT) [BifetEtAl09]

- Also known as restricted Hoeffding Tree
- Introduces a maximum size bound: size = #splitting nodes
- When the limit is reached, the tree is reset
  - Test for the limit, after a node is split. If violated, reset the tree.



ASHT over time [Mahmud15]

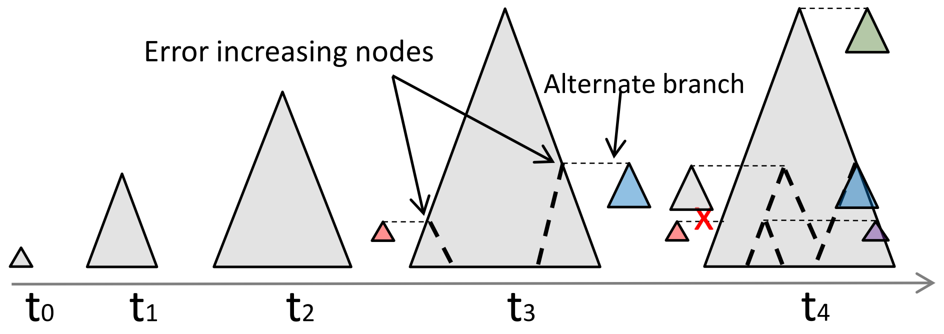
So, the forgetting is global.

How can we "implement" local forgetting? What are your ideas?

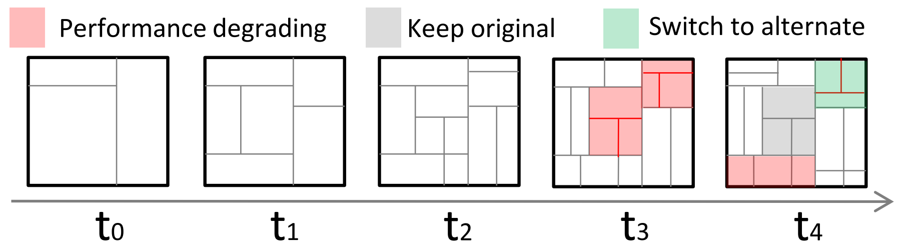
- The tree forgets
  - but, due to the reset, it looses all information learned thus far (and usually it takes some time to build this information)

Concept-Adapting Hoeffding Tree [HultenEtAl01]

- Starts maintaining an alternate sub-tree when the performance of a node drops
- 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



So, the forgetting is local.



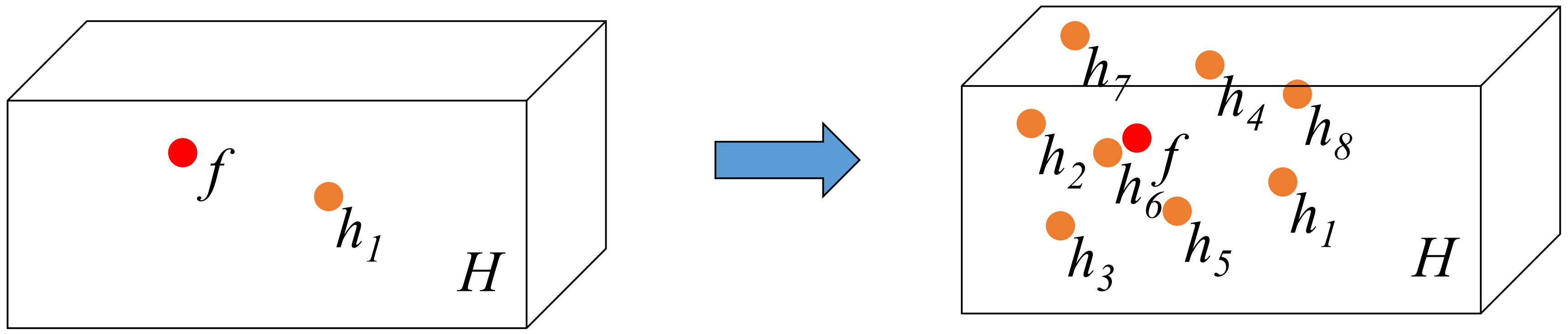
AdaHT over time [Mahmud15]

Concept-Adapting Hoeffding Tree [HultenEtAl01]

- Goal: keep the tree model consistent with a sliding window of examples W
- Upon the arrival of a new example/instance x from the stream
  - Adding to the window: (x,y) is added into the window & the oldest window example (x',y') needs to be forgotten
  - Model update (addition): (x,y) is incorporated into the tree model (ending up in leaf node l)
  - The necessary statistics along the path from the root to l are updated
  - The tree might be able to grow (since the number of instances in l is increased)
  - Model update (forgetting): since (x',y') is not part of the window anymore the corresponding statistics in the model need to be updated
  - Due to the these changed in the statistics, old splits might not be best anymore. Splits are periodically checked and if invalidated, i..e, $\overline{\Delta G} < \varepsilon$, this indicates that either the original split was wrong or that the data generation process has changed.
    - In such a case, an alternate subtree is grown from this node

From single decision tree models to ensembles of decision tree models

- Ensemble learning: Combine multiple models (base learners) to learn an overall better model



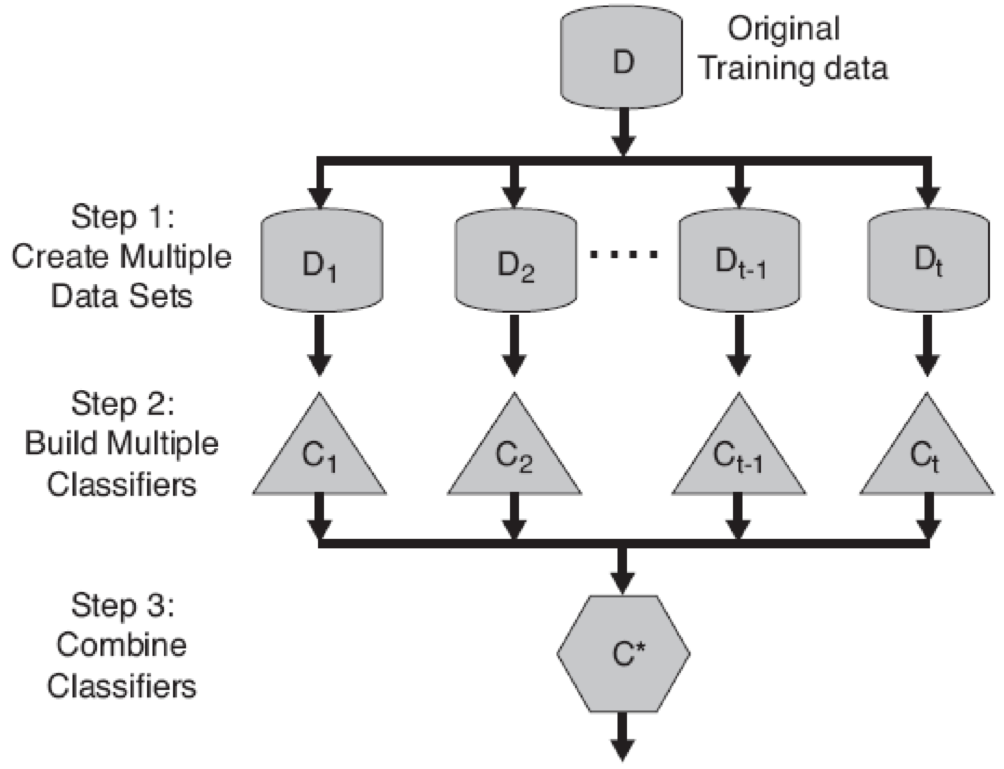
- Sounds wonderful, but for an improved performance necessary conditions need to hold:
  - Base classifiers are sufficiently “accurate”
    - better than random predictions
  - There are no correlations between the predictions of the base-learners
    - Base learners should make different errors (be diverse)

Ensemble learning is (another) important topic which we will not cover in this lecture.

I list a couple of resources at the end of the slides if you are interested.

From single decision tree models to ensembles of decision tree models

- The different base learners can be generated in different ways
  - By varying the training sets (Methods: bagging, boosting)
    - Bagging: sample from a uniform distribution
    - Boosting: sample from a weighted distribution
  - By manipulating the input features (Methods: learn on varying subspaces, use multi-view data)
  - By manipulating the class labels (Methods: various methods for mapping multi-class to 2-class problems)
  - By manipulating the learning algorithms (Methods: introduce randomness, employ varying initial models)

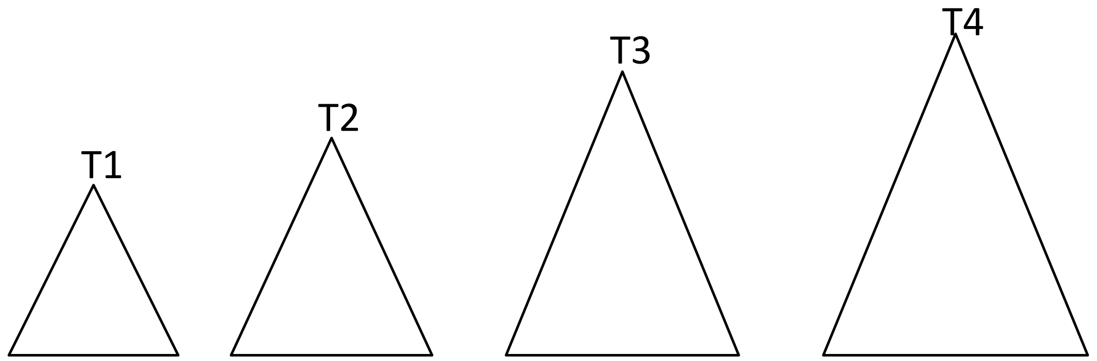


D
Original Training data
Step 1: Create Multiple Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2: Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3: Combine Classifiers
C^*

Figure 5.31. A logical view of the ensemble learning method.

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

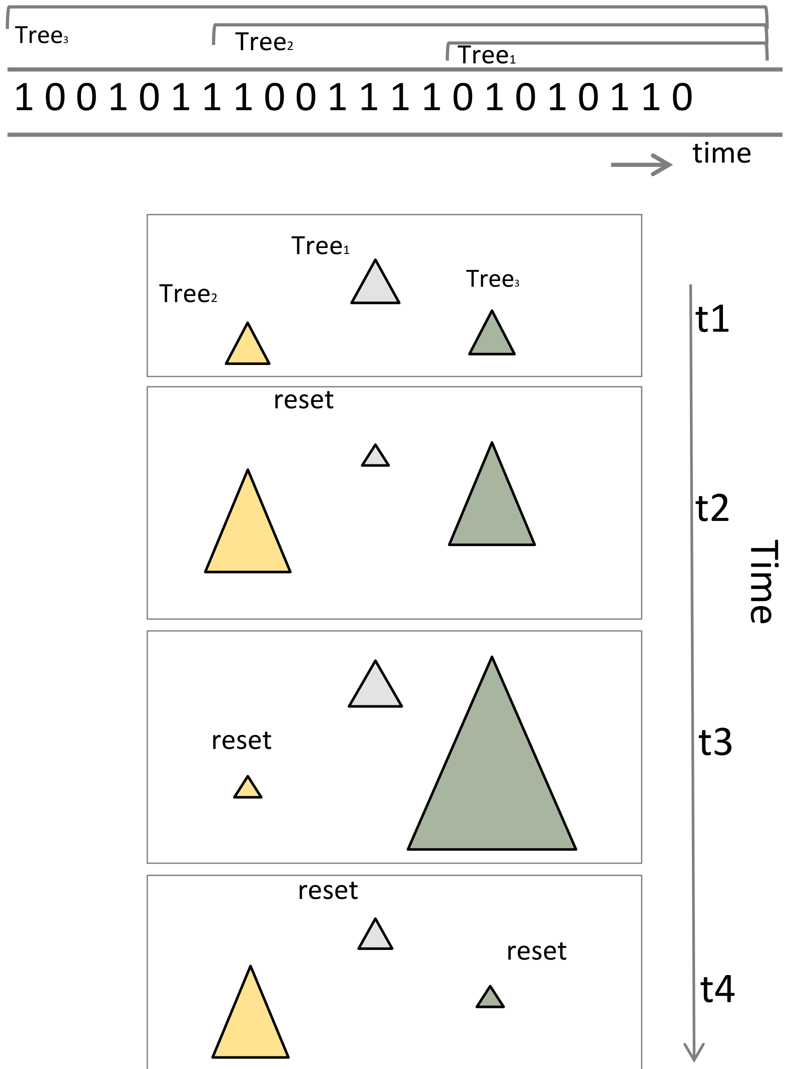
- Introduces an ensemble (bagging) of ASHTs of different sizes
- The different size ASHTs increase tree diversity (and therefore, bagging performance) as follows:



  - Smaller trees adapt more quickly to changes
  - Larger trees perform better during periods with no or little change

- How the tree sizes are set?
  - 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

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



ASHT over time [Mahmud15]

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 the user defined limit
- AdaHT forgets my replacing sub-trees with new ones
- Bagging ASHT uses varying size trees that respond differently to change

Outline

Evaluation of data stream classifiers

- Evaluating the quality of a classifier is a critical task
- Traditional evaluation that assumes a fixed training-test set is not adequate.
- The evaluation should also take into account the evolving nature of the data.

(batch) Classifier evaluation – see also lecture 6

- The quality of a classifier is evaluated over a separate test set of labeled instances
- For each test instance, its true class label is compared to its predicted class label (by some classifier) and the disagreement is computed
- Confusion matrix: A useful tool for analyzing how well a classifier performs.
    - For an m-class problem, the matrix is of size m x m
    - An example of a matrix for a 2-class problem:

Predicted class

Actual class

	C_1	**C_2**	**totals**
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P’	N’	

- Terminology
    - Positive tuples: tuples of the main class of interest
    - Negative tuples: all other tuples

(batch) Classifier evaluation measures – see also lecture 6

	C_1	C_2	totals
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P'	N'	

- Standard evaluation measures like accuracy, ER, BER etc
- Accuracy/ Recognition rate:
  - % of test set instances correctly classified

$$accuracy(M) = \frac{TP + TN}{P + N}$$

classes	buy_computer = yes	buy_computer = no	total	recognition(%)
buy_computer = yes	6954	46	7000	99.34
buy_computer = no	412	2588	3000	86.27
total	7366	2634	10000	95.42

- Error rate/ Missclassification rate:
  - % of test set instances incorrectly classified
  - error_rate(M)=1-accuracy(M)

$$error\_rate(M) = \frac{FP + FN}{P + N}$$

(batch) Classifier evaluation methods – see also lecture 6

- Standard evaluation setup: typically k-fold cross-validation

- Holdout method
  - Given data is randomly partitioned into two independent sets
    - Training set (~2/3) for model construction, Test set (~1/3) for evaluation
- Cross-validation (k-fold cross validation, usually k = 10)
  - Randomly partition the data into k mutually exclusive subsets D1, …, Dk each approximately equal size.
  - Training and testing is performed k times
    - At the i-th iteration, use Di as test set and the rest k-1 partitions as training set
  - Accuracy is the avg accuracy over all iterations

Stream classifiers evaluation: Evaluation setup

- Two main approaches:
  - Holdout evaluation: two separate datasets one for training one for testing
  - Prequential evaluation (or, interleaved test-then-train evaluation): one dataset for training and testing

Stream classifiers evaluation: Holdout evaluation

- Holdout evaluation
  - 2 separate datasets: the train dataset and an independent test dataset
  - Training: The classifier is trained online using the train dataset
    - Dynamic train dataset
  - Testing: The classifier is evaluated over the test dataset
    - The test set is realized
      - at the beginning of the data stream (so it is static) or,
      - regularly (so, it is dynamic)
  - Evaluation measures reported
    - Over the test set

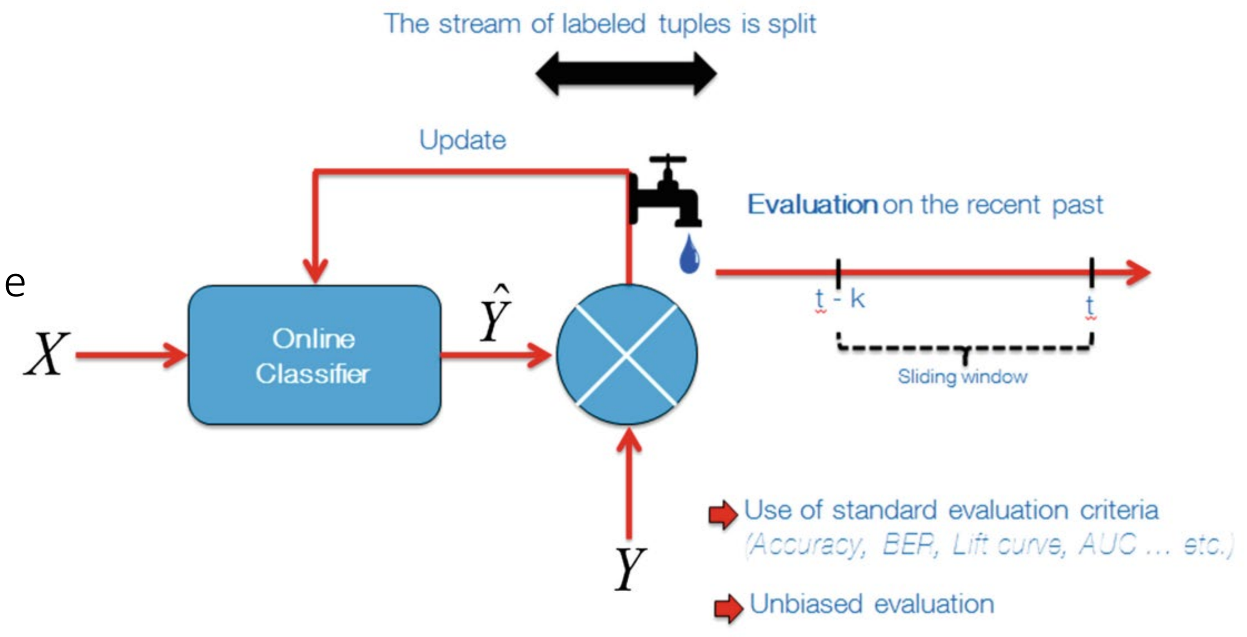
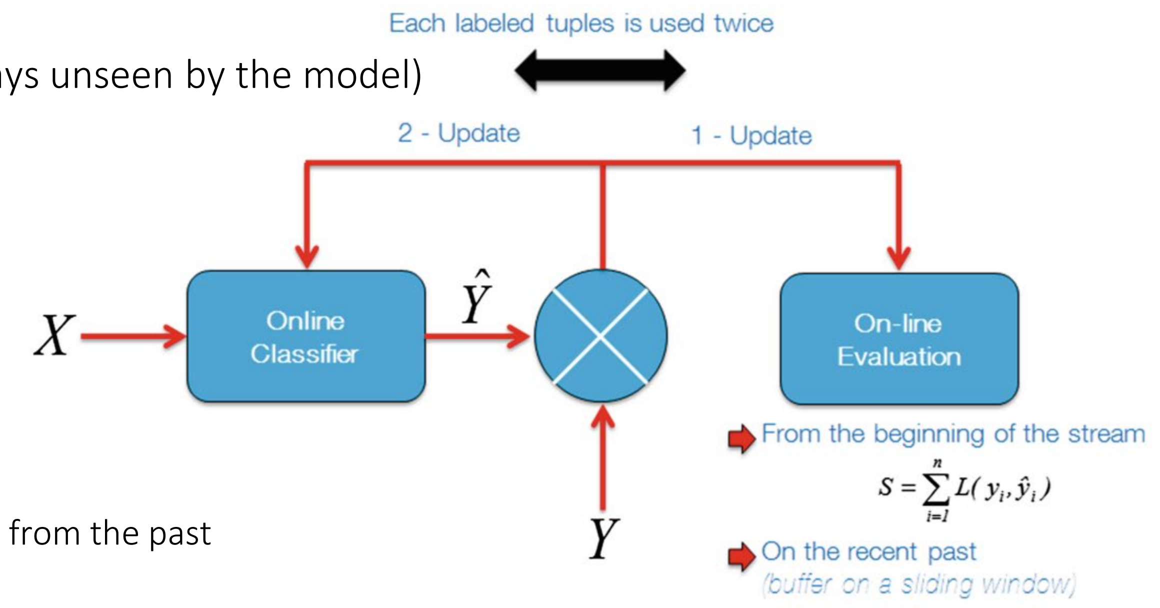

The stream of labeled tuples is split
Update
Evaluation on the recent past
X
Online Classifier
$\hat{Y}$
t - k
t
Sliding window
Y
Use of standard evaluation criteria (Accuracy, BEP, Lift curve, AUC ... etc.)
Unbiased evaluation

Fig. 4. Holdout Evaluation

Source: Lemaire et al

Stream classifiers evaluation: Prequential evaluation

- Prequential or, interleaved test-then-train evaluation
  - One dataset for training and testing
  - Model is first tested then trained on each instance
    - Test set is also dynamic!
    - Not holdout set is needed (test examples are always unseen by the model)
    - All available data are fully used
  - Evaluation measures reported
    - From the beginning of the stream
    - Over the recent part, e.g.,
      - sliding window
      - buffer
      - fading factor to decrease the weights of prediction from the past

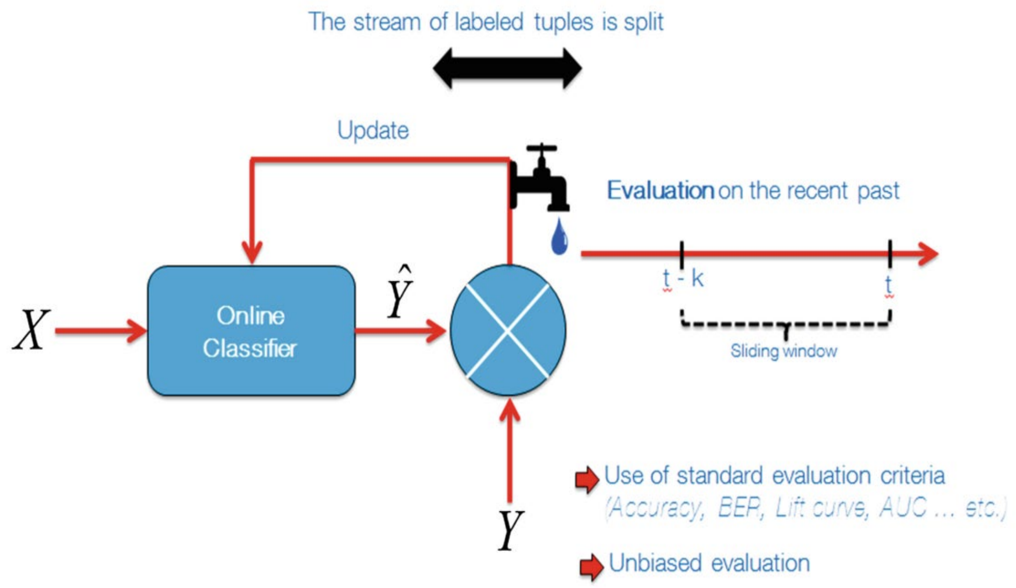


Each labeled tuples is used twice
2 - Update
1 - Update
X
Online Classifier
$\hat{Y}$
On-line Evaluation
Y
From the beginning of the stream
$S = \sum_{i=1}^{n} L(y_i, \hat{y}_i)$
On the recent past
(buffer on a sliding window)

Fig. 5. Prequential Evaluation

Source: Lemaire et al

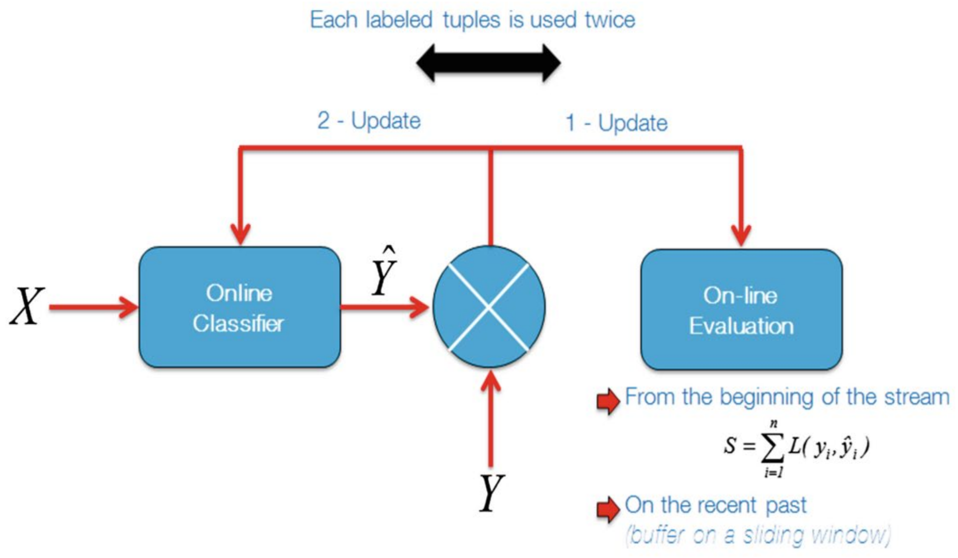
Stream classifiers evaluation: Holdout vs Prequential evaluation



The stream of labeled tuples is split
Update
Evaluation on the recent past
t - k
t
Sliding window
X
Online Classifier
Ŷ
Y
Use of standard evaluation criteria (Accuracy, BEP, Lift curve, AUC ... etc.)
Unbiased evaluation

Fig. 4. Holdout Evaluation

Source: Lemaire et al



Each labeled tuples is used twice
2 - Update
1 - Update
X
Online Classifier
Ŷ
Y
On-line Evaluation
From the beginning of the stream
$S = \sum_{i=1}^{n} L(y_i, \hat{y}_i)$
On the recent past (buffer on a sliding window)

Fig. 5. Prequential Evaluation

Source: Lemaire et al

Stream classifiers evaluation: evaluation measures

Cohen's kappa: More information

Kappa value	Classifier's performance
0%-20%	bad
21%-40%	fair
41%60%	moderate
61%-80%	substantial
81%-100%	(almost) perfect

- Accuracy, over a sliding window
  - % of test set instances correctly classified
- Cohen's kappa measure, over a sliding window
  - normalizes the accuracy of a classifier p_o by that of a chance predictor p_c

$$k = \frac{p_0 - p_c}{1 - p_c}$$

  - po is the observed agreement $p_o = \sum_{i=1}^{k} P(\hat{y} = i | y = i)$
    - In our case: correct predictions by the model
  - pc is the chance/random agreement using the observed data to calculate the probabilities of each predictor randomly seeing each class: $p_c = \sum_{i=1}^{k} P(\hat{y} = i) P(y = i)$

Stream classifiers evaluation: evaluation measures

- An example on Cohen’s kappa
- Let the confusion matrix over the test set (sliding window)

classes	Yes	no	total
Yes	15	5	20
no	10	70	80
total	25	75	100

$$p_o = \sum_{i=1}^{k} P(\hat{y} = i | y = i)$$

$$p_c = \sum_{i=1}^{k} P(\hat{y} = i) P(y = i)$$

$$k = \frac{p_0 - p_c}{1 - p_c}$$

- **What is kappa? Is this a good classifier? Your turn …**
- The observed agreement in the predictions p_o is:
  p_o=(15+70)/100=0.85
- The agreement by chance p_c is:
  - Agreement by chance in Yes plus agreement by chance in No
  - p_c=(20/100)*(25/100)+ (80/100)*(75/100)=0.65
- So, k=(0.85-0.65)/(1-0.65)=0.57

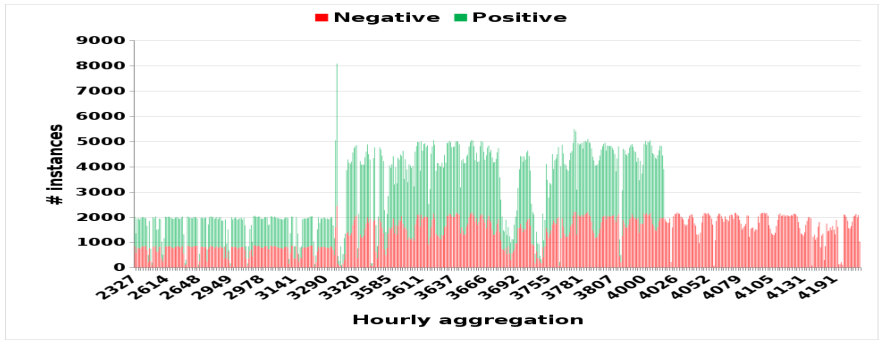
Kappa value	Classifier‘s performance
0%-20%	bad
21%-40%	fair
41%60%	moderate
61%-80%	substantial
81%-100%	(almost) perfect

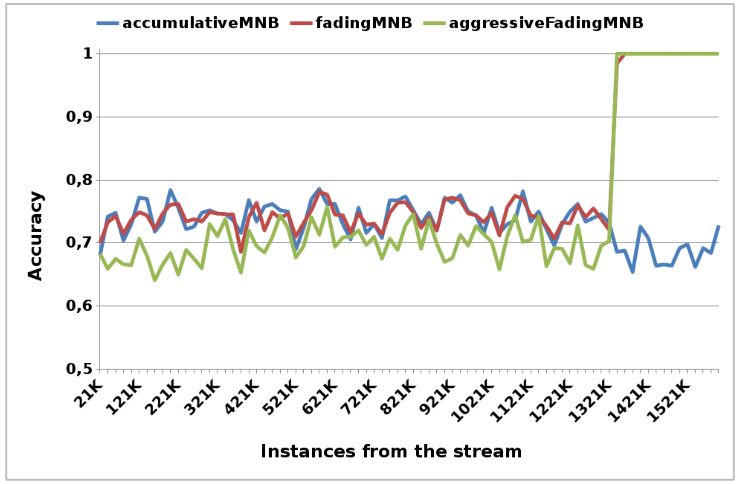
Stream classifiers evaluation: evaluation measures

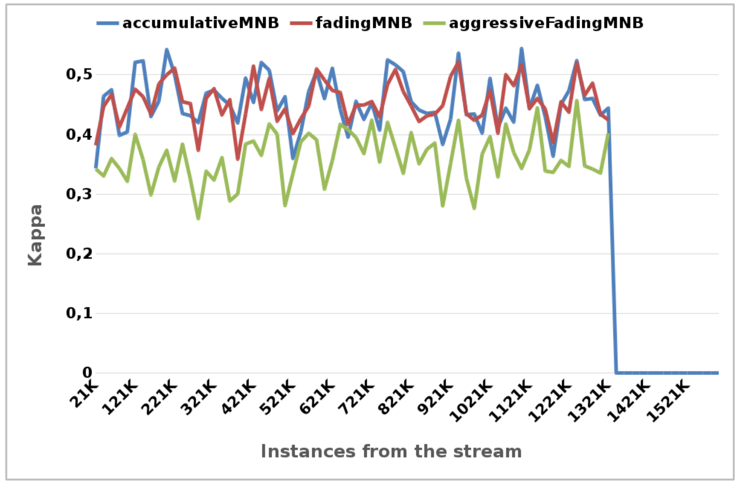
- A modification of Cohen’s kappa statistic, is the Kappa Plus Statistic (K+) which replaces the random classifier p_c with the so-called persistent classifier, which predicts the label of the latest example received.
- Similarly to the discussion for the batch case, evaluation of stream classifiers is a topic on its own
  - A good reference on this topic:
    - Evaluation methods and decision theory for classification of streaming data with temporal dependence

Accuracy vs Kappa

- Typically reporting over time/progress of the stream to demonstrate how the learner reacts to underlying stream changes
- Prequential evaluation, hourly-aggregated stream [WagnerEtAl15]





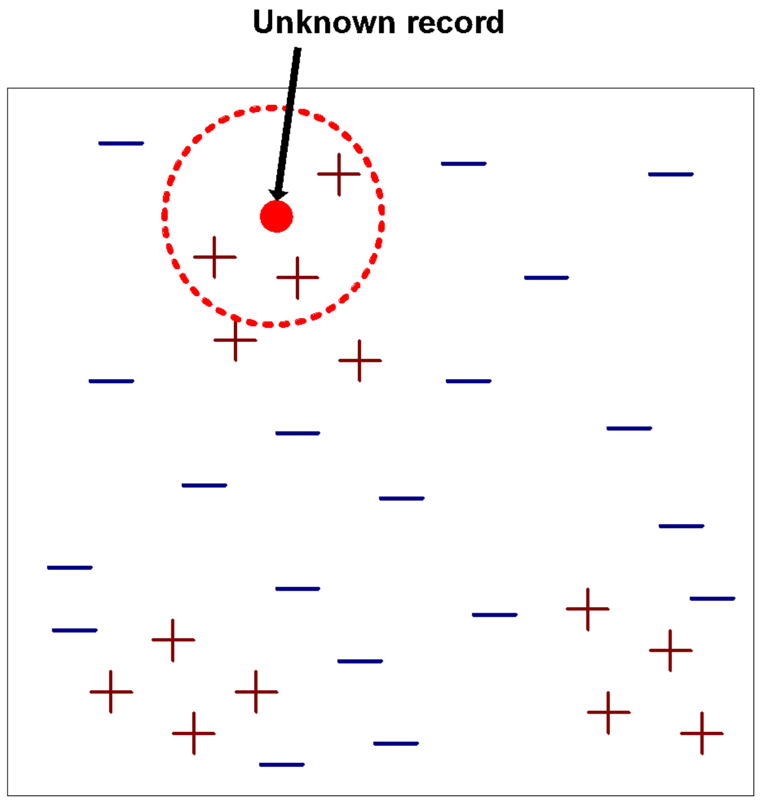


Outline

(batch) k-Nearest Neighbor (kNN) classifier – see also lecture 4

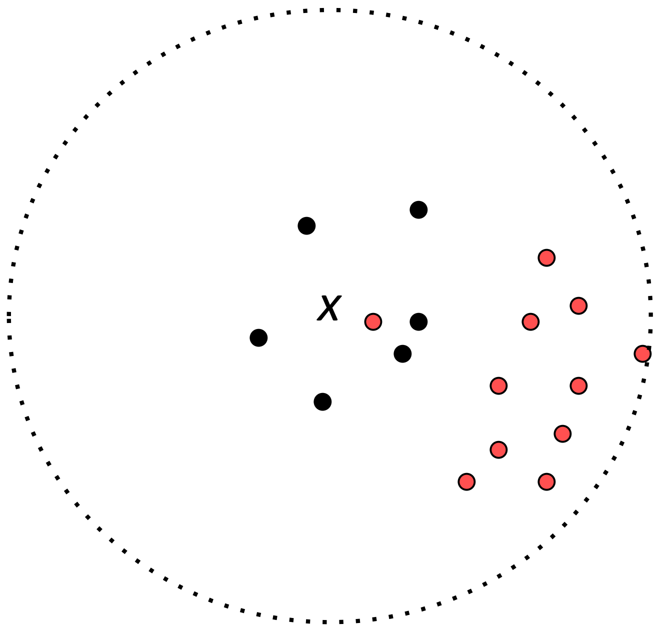
- Input:
  - A training set *D* (with known class labels)
  - A distance measure to compute the distance between two instances
  - The number of neighbors *k*

- Classification: Given a new unknown instance *X*
  - Compute distance to the training records, i.e., $dist(X, d), d \in D$
  - Identify the *k* nearest neighbors
  - Use the class labels of the *k* nearest neighbors to determine the class label of *X* (e.g., by taking majority vote)



(batch) k-Nearest Neighbor (kNN) classifier – see also lecture 4

- The class of an unknown instance *X* is determined from its neighbors
- If *k*=1, the class is that of the closest instance
- In the general case (*k*>1) → voting in the neighborhood of the instance
  - Majority voting: take the majority vote of class labels among the neighbors
    - All neighbors contribute equally to the classification
    - The algorithm is very sensitive to the choice of *k*
  - Weighted voting: each neighbor d contributes with a weight proportional to its distance from the unknown instance X
    - e.g., a possible weight factor: $w=1/dist(X,d)^2$

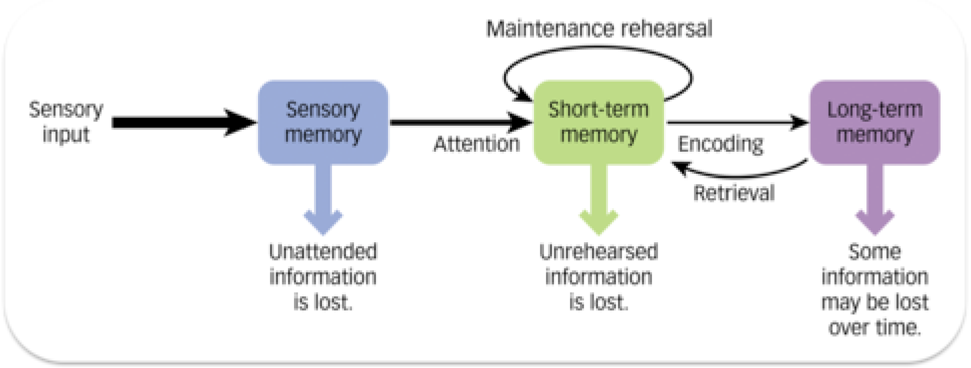


From batch kNN to stream kNN

- What are your ideas: How can we apply kNN over a (never) ending stream of (non-stationary) instances?
- Concrete questions to be answered:
  - How can we add new information (incremental update)?
  - How can we remove outdated information (forgetting)?
  - How can we find outdated information?
  - kNN is a lazy learner (it stores instances, not a model) but in a stream not all instances can be stored (bounded memory constraint). How can we deal with the efficiency aspects?

SAMKNN - kNN Classifier with Self Adjusting Memory [LosingEtAl2016]

- SAMKNN is inspired by biological memory models and their coordination
  - The dual-store memory model consists of a short-term (STM) and a long-term (LTM) memory models

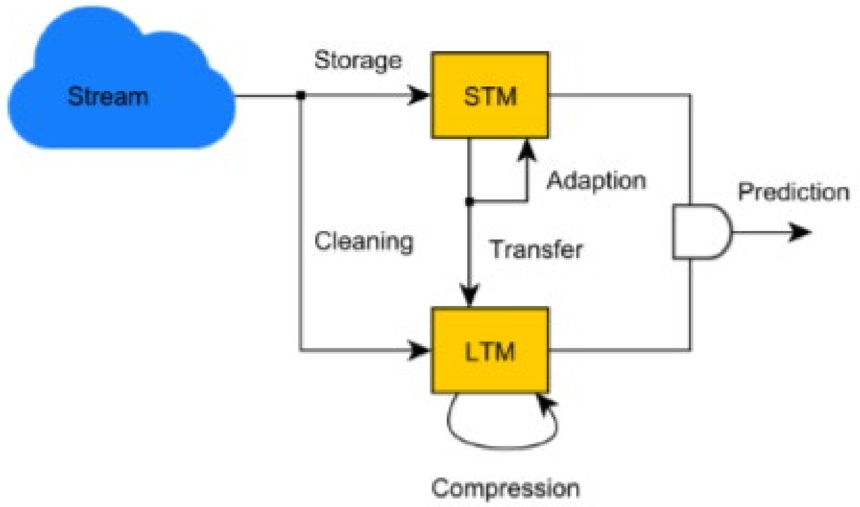


  - Sensory information arrives at the STM and is joined by context relevant knowledge from the LTM
  - The capacity of the STM is quite limited and information is kept up to one minute, whereas the LTM is able to preserve it for years
  - Immediate processing e.g. remembering the beginning of a read sentence, largely uses the STM.
  - Knowledge recalled from the past either explicitly requires the LTM e.g. consciously remembering events in life, or in an implicit way e.g. how to ride a bike.

Basic idea of SAMKNN

- SAMKNN is inspired by the dual-store memory model
  - Explicit separation of current and past knowledge stored in dedicated memories
  - The different memories have different conservation spans
  - (filtered) Information is transferred from the STM to the LTM
    - The goal of the filtering is to remove any outdated information from the LTM
      - This is evaluated by finding that information that contradicts STM, so solving inconsistencies
- The basic model is a kNN classifier

SAMKNN Architecture



- Incoming examples are stored within the STM.
- The cleaning process keeps the LTM all-time consistent with the STM.
- Information is transferred from STM into the LTM
- LTM is compressed each time the available space is exhausted.
- Both models are considered during prediction, depending on their past performances.

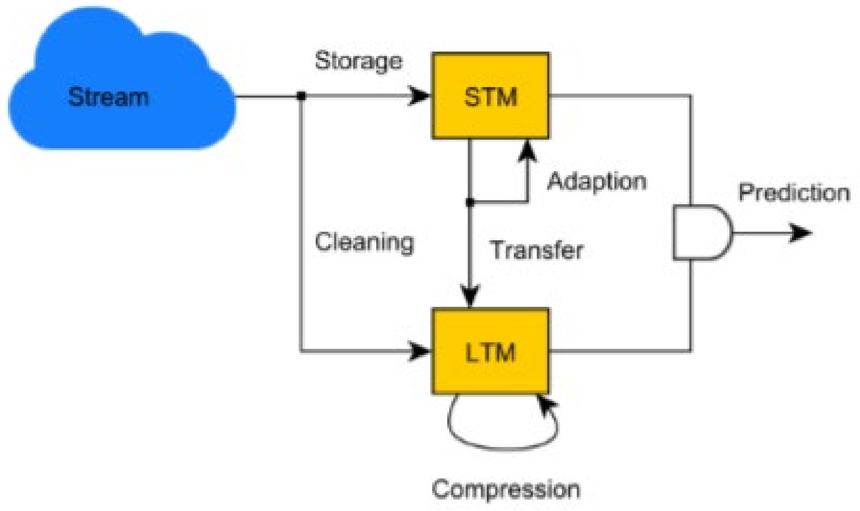
Model definition

- Memories are represented by sets M_{ST}, M_{LT}, M_C
- M_{ST} represents the current knowledge (I will also use the term STM for simplicity of notation)
  - it is implemented as a sliding window of the most recent *m* examples (I will also use the term LTM for simplicity of notation)
- M_{LT} represents in a compressed way the past knowledge which does not contradict the M_{ST}.
  - M_{LT} consists of *p* compressed "points" (cluster centers, see next slides)
- The combined memory (C) is a union of M_{LT} and M_{ST} with size m+p
- Every set induces a distance-weighted kNN model

$$M_{ST} \longrightarrow \text{kNN}_{M_{ST}}$$
$$M_{LT} \longrightarrow \text{kNN}_{M_{LT}}$$
$$M_{C} \longrightarrow \text{kNN}_{M_{C}}$$

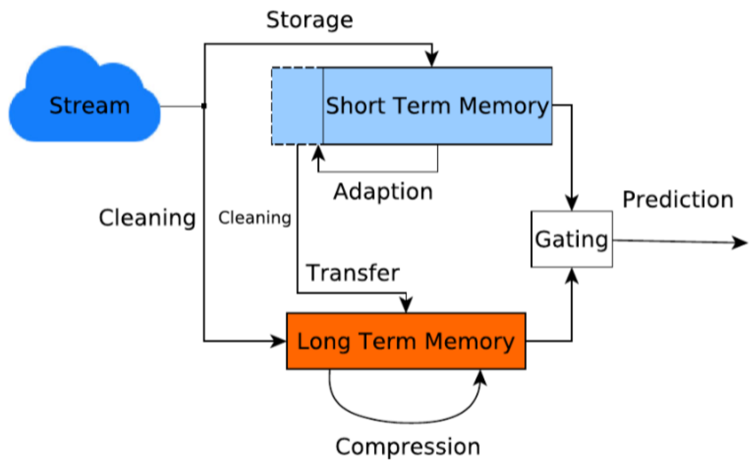
Key operations in SAMKNN

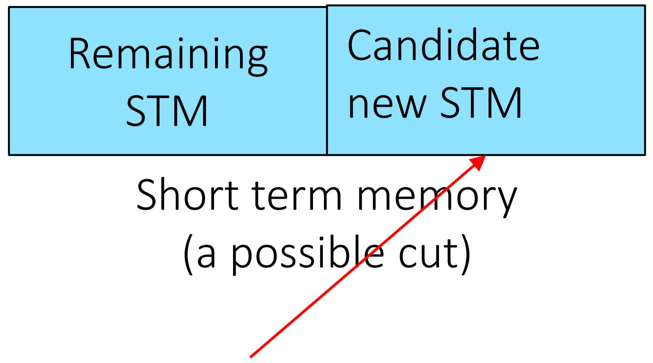
- STM adaptation
- Cleaning and transferring from STM to LTM
- Compression of the LTM
- Prediction



Key operations in SAMKNN: STM adaptation

- STM is a dynamic sliding window and grows continuously with each new instance
  - Recal: STM's goal is to focus on current information!
  - Therefore, its size has to be reduced, whenever the concept changes such that examples of the former concept are dropped
- When does a concept change?
- SAM-kNN does not explicitly detect a concept change → size is adjusted such that the error of the remaining STM is minimized
- To this end, differently sized STMs are evaluated and the one with the minimum error is chosen
  - It tries out only a logarithmic number of windows (discussion omitted)
- Whenever the STM is shrunk, the sorted out data (Remaining STM in the figure) are not simply discarded as they might contain important information (e.g., for reoccurring concepts).
- The idea instead is to transfer them to the LTM
  - The transferred instances however should be consistent with the LTM





Short term memory
(a possible cut)

Evaluate the Interleaved test-then-train error on right (most recent part of the memory)

Key operations in SAMKNN: Cleaning and transferring

- Remaining STM information (previous slide) should be transferred into the LTM (principle: no information should be lost)
- LTM data that are not consistent with (new) STM should be removed from the LTM
- So, we need a cleaning operation that resolves such inconsistencies between remaining STM and LTM
- Cleaning: For each example (x_i,y_i) to be transferred to the LTM
  - Determine the kNN of x_i in the remaining STM memory which belong to the same class y_i
  - Find the distance to the k^{th} nearest neighbor $\rightarrow \theta$
  - Find the θ-neighborhood of x_i in the LTM
    - Remove those instances from LTM that have a different class than y_i $\rightarrow$ local cleaning/forgetting.

Key operations in SAMKNN: Compression of the LTM

- Whenever the size limit is reached, i.e., $m+p > L_{max}$, the LTM data are compressed
- Compression is done using clustering and in particular the k-Means++ algorithm
  - k-Means++: a variation of k-Means that uses carefully choses the initial seeds of the clusters
- Clustering is performed within instances of the same class
  - The number of clusters k is selected so that the number of instances is reduced by half
- So, in the general case, LTM consists of a mix of real points and compressed points (cluster centers)

Key operations in SAMKNN: Prediction

- Each model contributes to the final prediction with a weight
- Weights represent the accuracy of the corresponding model on the recent data
- The model with the highest weight gives the final prediction

$$x \mapsto \begin{cases} \mathrm{kNN}_{M_{ST}}(x) & \text{if } w_{ST} \geq \max(w_{LT}, w_C) \\ \mathrm{kNN}_{M_{LT}}(x) & \text{if } w_{LT} \geq \max(w_{ST}, w_C) \\ \mathrm{kNN}_{M_C}(x) & \text{if } w_C \geq \max(w_{ST}, w_{LT}) \end{cases}$$

- In case of ties, there is a prioritization from left to right:

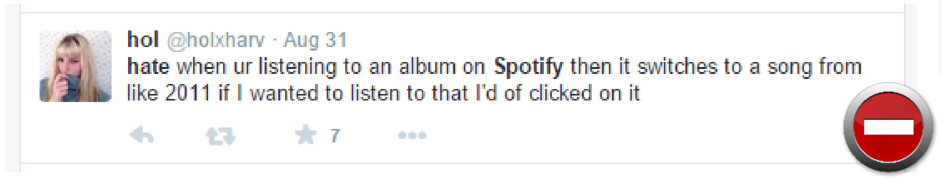
$$\mathrm{kNN}_{M_{ST}}, \mathrm{kNN}_{M_{LT}}, \mathrm{kNN}_{M_C}$$

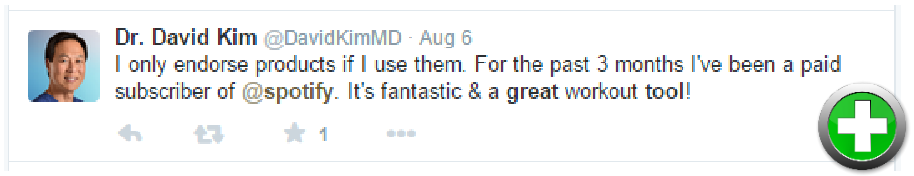
Outline

Problem specific motivation: Sentiment analysis over textual streams

- Goal: Given an opinionated stream, automatically determine whether an opinion is positive or negative.

…


hol @holxharv · Aug 31
hate when ur listening to an album on Spotify then it switches to a song from like 2011 if I wanted to listen to that I'd of clicked on it

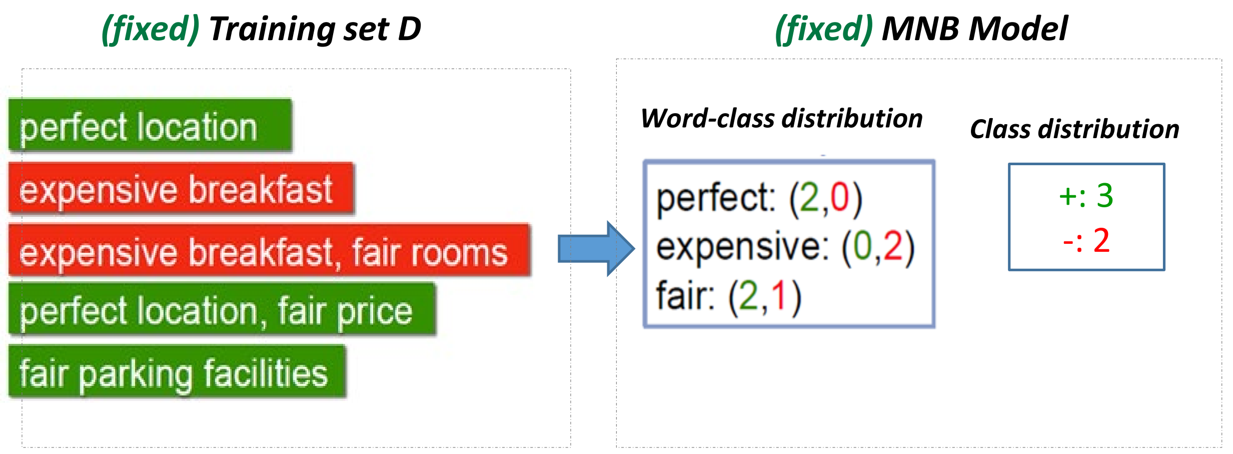

Dr. David Kim @DavidKimMD · Aug 6
I only endorse products if I use them. For the past 3 months I've been a paid subscriber of @spotify. It's fantastic & a great workout tool!

High dimensional &dynamic feature space

- Causes of concept drifts in an opinionated stream
  - New vocabulary/ features: New topics of interest arise over time, e.g., “#Brexit”, #refugeecrisis” … and old topics fade out (e.g., Eurovision 2017)
  - Change of sentiment: Even for the same topic, the sentiment of the crowd might change, e.g., “#brexit”, “same-sex marriage”.
  - Different context: “heavy” negative for a camera, positive for a solid wood furniture

A simple yet powerful classifier for text

- Traditional Multinomial Naïve Bayes (MNB) is a batch learner



- Prediction, for a new document *d*:

$$P(c|d) = \frac{P(d|c)}{P(d)} \xrightarrow{\text{Independence assumption}} P(c|d) = \frac{P(c)\prod_{i=1}^{|d|} P(wi|c)^{f_i^d}}{P(d)}$$

Class-prior estimation

$$\hat{P}(c) = \frac{N_c}{|\mathcal{D}|}$$

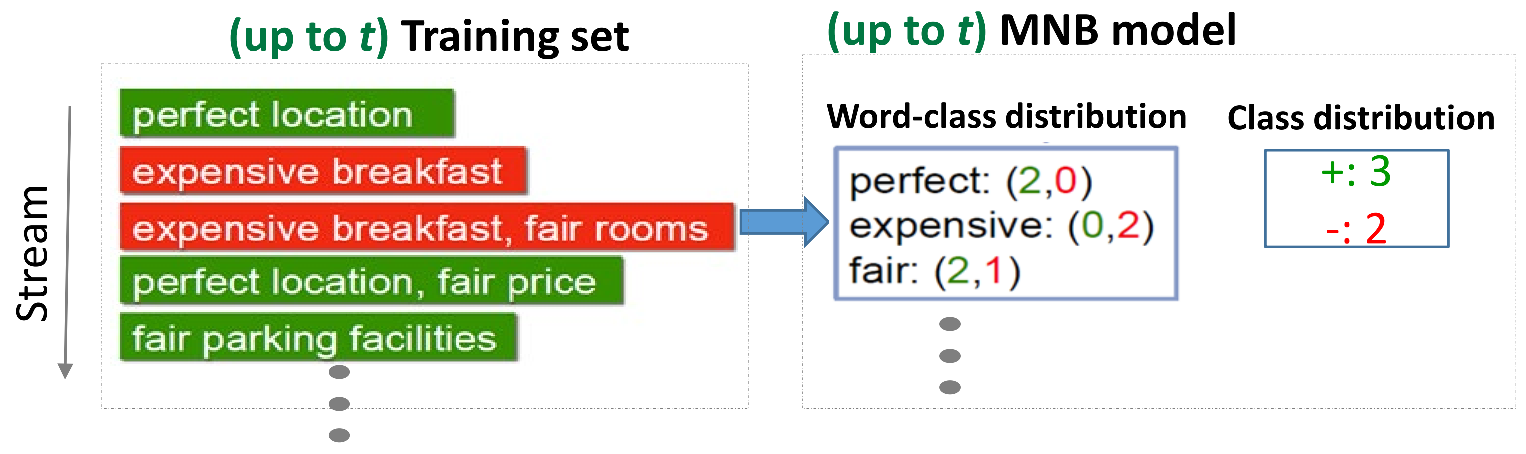
Word class conditional estimation

$$\hat{P}(w_i|c) = \frac{N_{ic}}{\sum_{j=1}^{|V|} N_{jc}}$$

$\boxed{\text{Fixed counts from D}}$

MNB is an incremental model

- Incremental Multinomial Naïve Bayes (MNB) for streams (accumulativeMNB)



- Prediction for a new document *d* at *t*: based on model counts up to *t*:

$$\hat{P}^t(c|d) = \hat{P}^t(c)\prod_{i=1}^{|d|}\hat{P}^t(w_i|c)^{f_i^d}$$

$$\hat{P}^t(c) = \frac{N_c{}^t}{|\mathcal{D}|^t} \qquad \hat{P}^t(w_i|c) = \frac{N_{ic}{}^t}{\sum_{j=1}^{|V|^t} N_{jc}{}^t}$$

Accumulated counts from the beginning of the stream

- Model adaptation
  - New instances are incorporated into the model
  - Nothing is forgotten! Long memory problem!

AN MNB that forgets: modeling data recency [WagnerEtAl2015]

- A temporal model that keeps track of the last time that an observation is made in the stream
  - For classes:

$$(N_c) \rightarrow (N_c, \boxed{t_{lo}^{c}})$$

last class observation time in the stream

  - For word-class pairs:

$$(N_{ic}) \rightarrow (N_{ic}, \boxed{t_{lo}^{ic}})$$

last word-class observation time in the stream

  - Timestamp propagation: from documents → classes, word-class pairs
  - Temporal de-coupling of words from documents
    - Observation updates might come from different documents
  - Allows differentiation of the observations based on their recency

Adaptation 1/2: by adding new instances

Ageing-based MNB

- Updated temporal probability estimates

$$\hat{P}^t(c) = \frac{N_c^t * \boxed{e^{-\lambda\cdot(t-t_{lo}^{c})}}}{|\mathcal{S}^t|}$$

What exactly is stored in the model?

$$\hat{P}^t(w_i|c) = \frac{N_{ic}^t * \boxed{e^{-\lambda\cdot(t-t_{lo}^{(w_i,c)})}}}{\sum_{j=1}^{|V^t|} N_{jc}^t * \boxed{e^{-\lambda\cdot(t-t_{lo}^{(w_j,c)})}}}$$

ageing effect

Adaptation 2/2: by forgetting (downgrading estimates)

- Forgetting strategy: Gradual ageing (exponential fading function)

$$age(o,t) = e^{-\lambda(t-t_o)}$$

t: current time
t_o: object's arrival time
λ: the decay rate

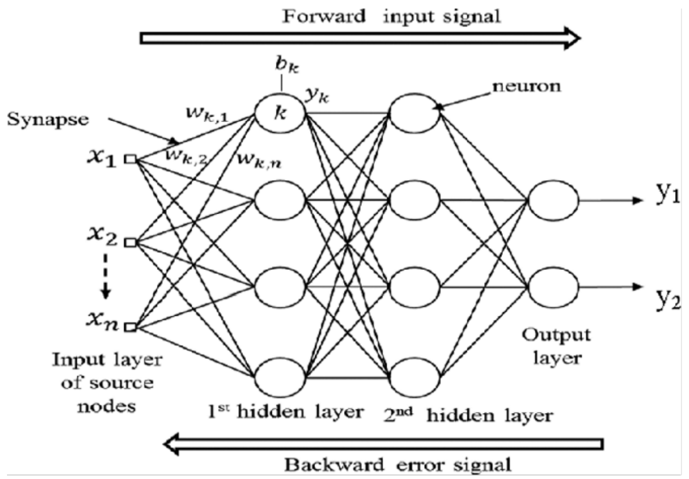
  - higher λ, less important the historical data
  - Points are halved every *1/λ* time units

Outline

- Batch learning assumptions
- Motivation
- Data streams
- Data stream classification basics
- Data stream classifiers: Family of decision tree stream classifiers
- Data stream classifiers: evaluation aspects
- Data stream classifiers: SAMKNN
- Data stream classifiers: Neural network models
- Things you should know from this lecture & reading material

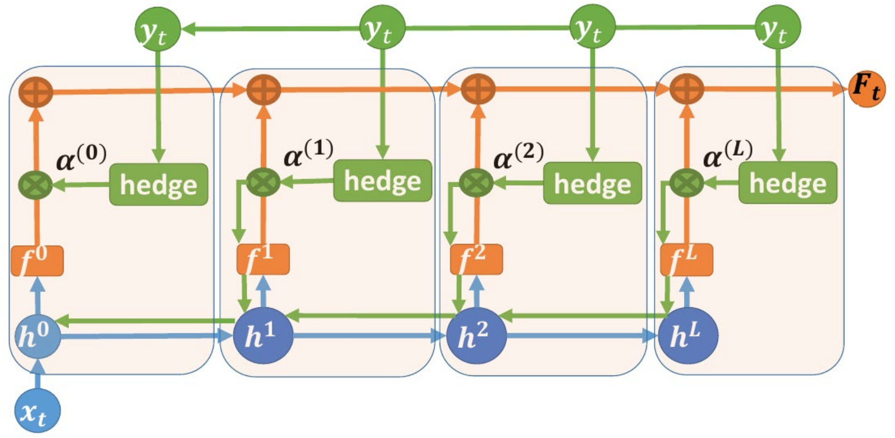
Online Deep Learning: Learning Deep Neural Networks on the Fly [SahooEtAl2018]

- Problems with traditional (deep) neural networks for streaming data
  - The network structure is typically fixed
  - Vanishing gradient problem in NNs
    - "In machine learning, the **vanishing gradient problem** is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value" from Wikipedia
  - ...



Online Deep Learning: Learning Deep Neural Networks on the Fly [SahooEtAl2018]

- Use an *"overcomplete network"* and adapt its depth online and automatically per demand. Each layer possesses an individual classifier, which is separately parametrized.
- More concretely
  - The DNN consists of a *fixed* number of *L* layers
  - Each layer possesses a *separate classifier*
    - It's separately parametrized by $\Theta^{(l)}$ and predicts using the parameters $W^{(l)}$ of the hidden layers so far
    - Hidden layer parameters are still shared between subsequent layers
  - Each layer thus makes a prediction of the input at different depths
- With increasing depth, the predictions make use of more previous (shared, latent) information
- Finally the individual prediction results are aggregated and weighted by $a^{(l)}$ to make the final decision
- The alpha weights control the contribution of each classifier



Online Deep Learning: Learning Deep Neural Networks on the Fly [SahooEtAl2018]

- The alpha weights are adapted using Hedge backpropagation
  - This allows the network to automatically find the optimal depth classifiers and dynamically vary the DNN capacity
- It's more robust to the vanishing gradient problem since gradients are also backpropagated from the individual classifiers
- It can be seen as an ensemble of networks of different depths that compete with another to increase their performance
- Enables a form of life-long learning (which focus mainly on information retention)
  - No clear forgetting strategy!

Outline

- Batch learning assumptions
- Motivation
- Data streams
- Data stream classification basics
- Data stream classifiers: Family of decision tree stream classifiers
- Data stream classifiers: evaluation aspects
- Data stream classifiers: SAMKNN
- Data stream classifiers: MNB
- Data stream classifiers: Neural network models
- Things you should know from this lecture & reading material

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 (full availability of class labels for all instances)
  - Semi-supervised learning, Active learning
- Many interesting variations of the problem: Dealing with class imbalances, rare-classes, new classes
- Dealing with dynamic feature spaces (e.g., text)

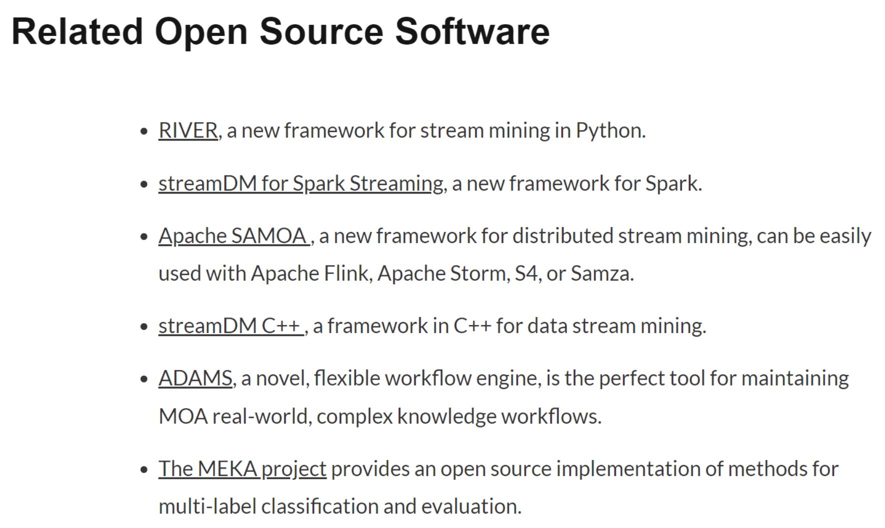
Things you should know from this lecture

- Velocity challenges
- Stream characteristics
- Why batch learning is not sufficient
- Stream classification basics
- Hoeffding-tree and the rest of the HT family stream classifiers
- Evaluation of classifiers: evaluation setups and evaluation measures

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 (full availability of class labels for all instances)
  - Semi-supervised learning, Active learning
- Dealing with class imbalances, rare-classes
- Dealing with dynamic feature spaces

Hands on experience

- Familiarize yourself with popular frameworks for stream learning
  - MOA: the most popular open source framework for data stream mining https://moa.cms.waikato.ac.nz/ (in Java)
    - "Machine Learning for Data Streams with Practical Examples in MOA" book
  - Scikit-multiflow "A machine learning package for streaming data in Python"
    - https://scikit-multiflow.github.io/
- 

Related Open Source Software

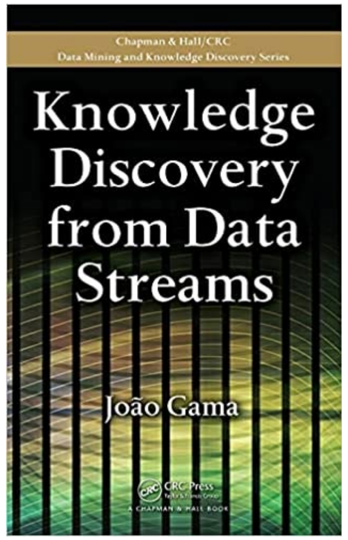
- RIVER, a new framework for stream mining in Python.
- streamDM for Spark Streaming, a new framework for Spark.
- Apache SAMOA , a new framework for distributed stream mining, can be easily used with Apache Flink, Apache Storm, S4, or Samza.
- streamDM C++ , a framework in C++ for data stream mining.
- ADAMS, a novel, flexible workflow engine, is the perfect tool for maintaining MOA real-world, complex knowledge workflows.
- The MEKA project provides an open source implementation of methods for multi-label classification and evaluation.

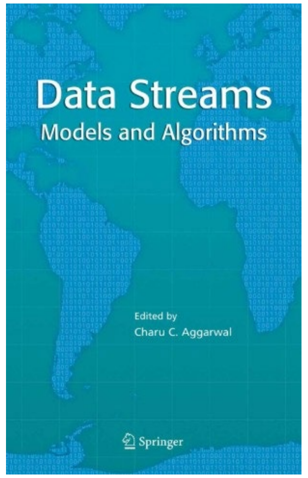
Source: https://moa.cms.waikato.ac.nz/

Thank you

Questions/Feedback/Wishes?

Reading material


Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series
Knowledge Discovery from Data Streams
João Gama
CRC Press



- Book: Knowledge discovery from data streams, J. Gamma
- Book: Data streams – Models and Algorithms, C. Aggrawal
- A survey on concept drift adaptation, Gamma et al, 2014
- [DomingosHulten00] The Hoeffding Tree paper
- [BifetEtAl09] New ensemble methods for evolving data streams
- [HultenEtAl01] Mining Time-Changing Data Streams
- [WagnerEtAl15] Ageing-Based Multinomial Naive Bayes Classifiers Over Opinionated Data Streams
- [LosingEtAl19] KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift
- [SahooEtAl2018] Online Deep Learning: Learning Deep Neural Networks on the Fly

- Resources on ensemble learning
  - Dietterich: Ensemble Methods in Machine Learning, http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf
  - Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, Chapman and Hall/CRC, 2012.
  - Also for streams:
    - Ensemble learning for data stream analysis: A survey

- Resources on feature-evolving data streams (someone asked in the previous lecture)
  - xStream: Outlier Detection in Feature-Evolving Data Streams
  - *Learning under Feature Drifts in Textual Streams*
  - Learning with Feature Evolvable Streams

Acknowledgements

- The slides are based on
  - *DM2 lecture@LUH(@Eirini Ntoutsi), KDD2/ SS16 lecture@LMU Munich (@Eirini Ntoutsi, Matthias Schubert, Arthur Zimek)*
  - Hossain Mahmud, MSc, "*Ensemble Learning in Data Streams*", TUM Munich, 2016.
  - For the SAMKNN part, MSc Amir Abolfazli helped with the slides
  - For the online deep learning part, MSc Philip Naumann helped with the slides