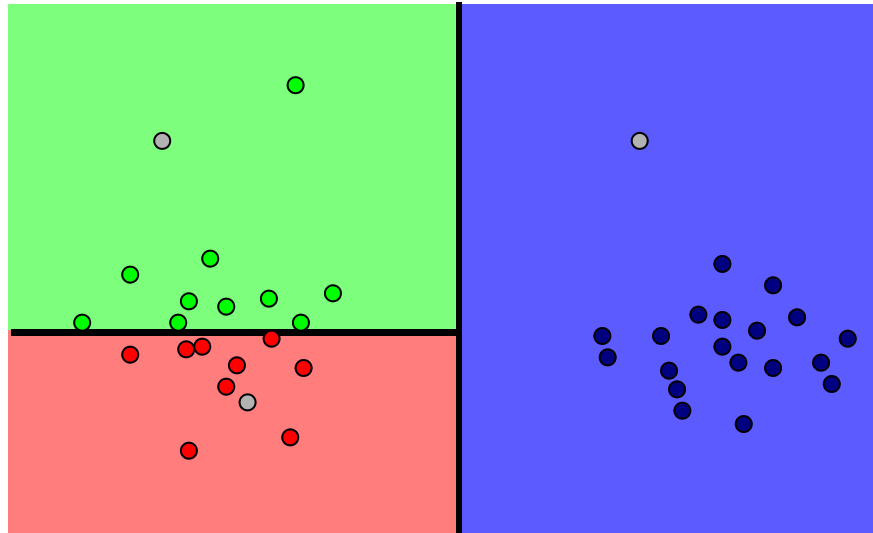- Motivation

- Data streams

- Data stream clustering

- Data stream classification
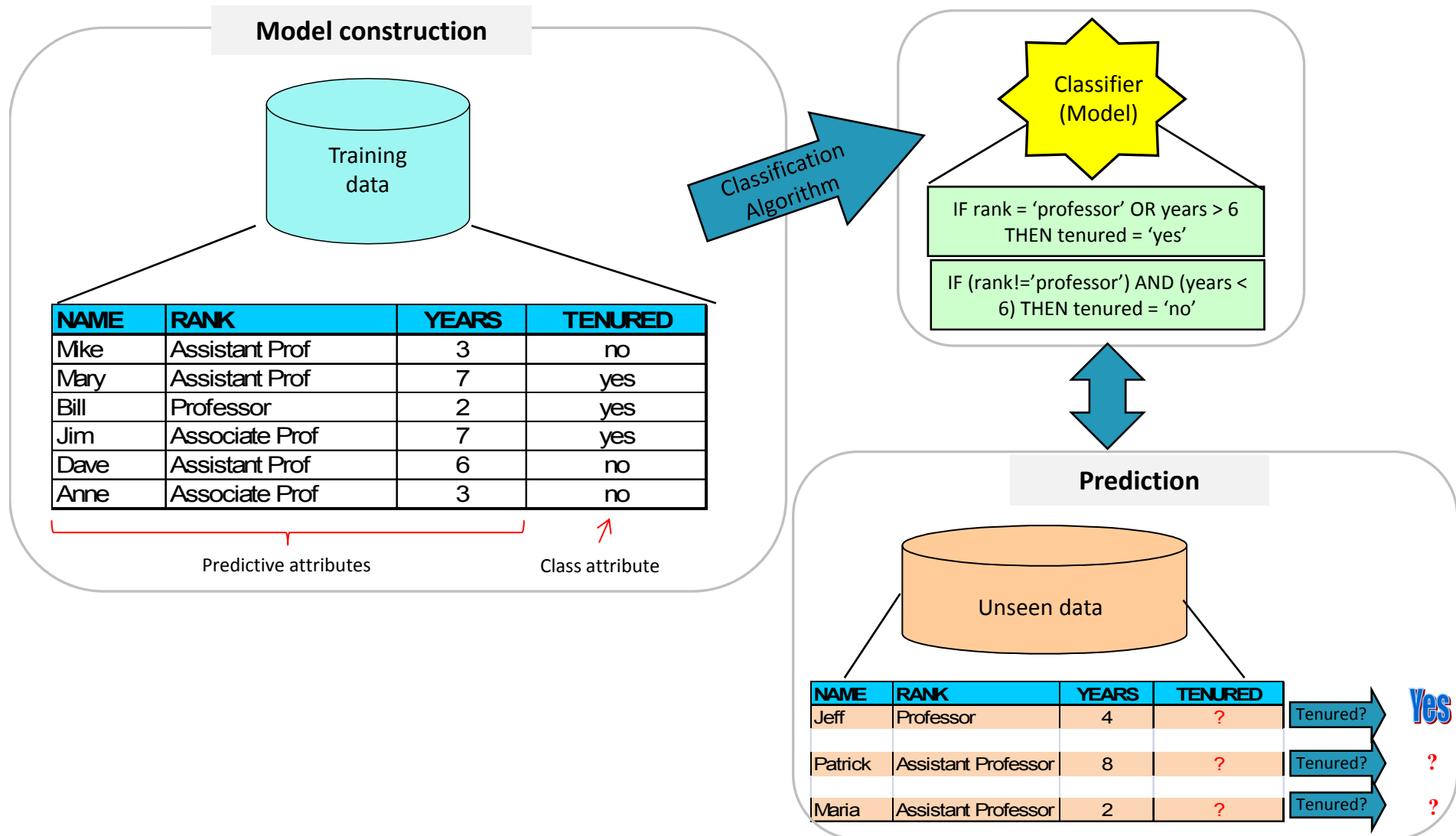
- Screw
- Nails
- Paper clips

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	?	Tenured?	**Yes**
Patrick	Assistant Professor	8	?	Tenured?	?
Maria	Assistant Professor	2	?	Tenured?	?

- 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

- 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)

- 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

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

  where p(X,y) is the joint distribution between instance X and classes y

- According to the Bayesian Decision Theory: $\quad p(y|X) = \dfrac{p(y)p(X|y)}{p(X)}.$

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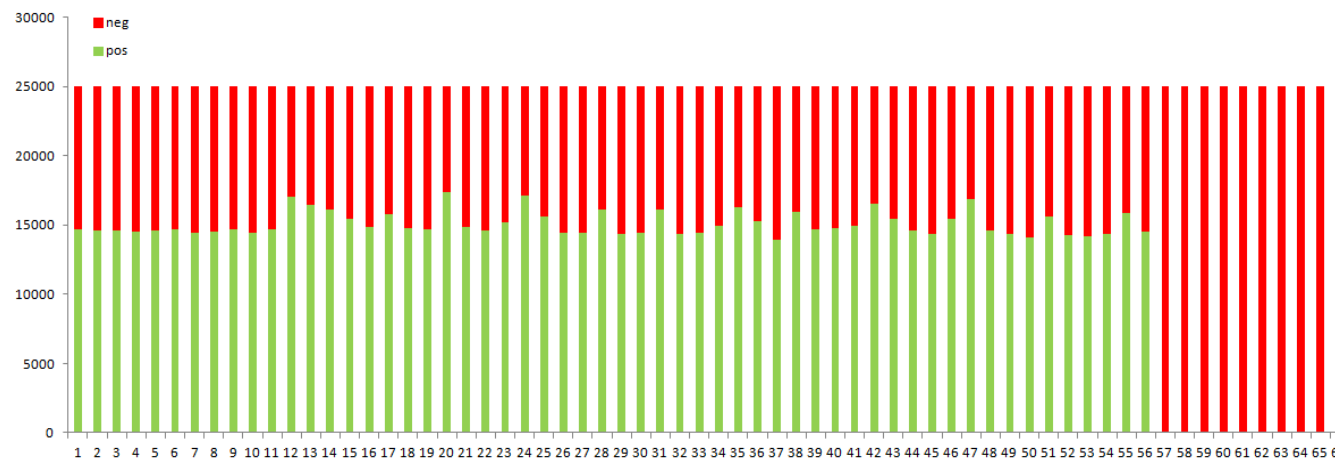

Evolving class distribution [Sinelnikova12]

- 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)
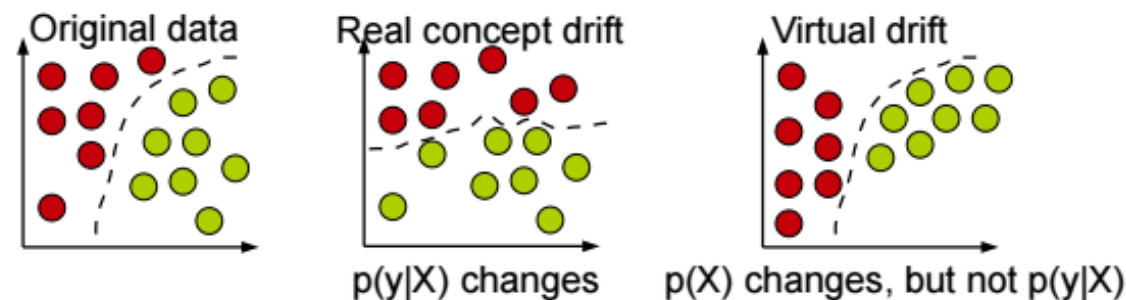
Source: [GamaETAl13]

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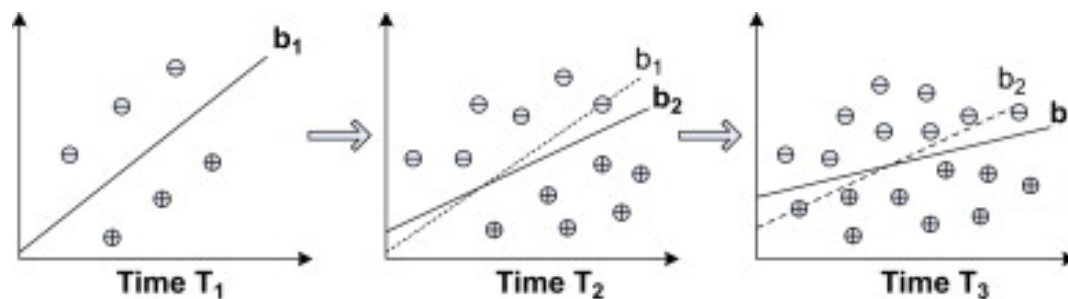
The classification boundary gradually drifts from b_1 (at T_1) to b_2 (at T_2) and finally to b_3 (at T_3).
(Source: A framework for application-driven classification of data streams, Zhang et al, Journal Neurocomputing 2012)

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
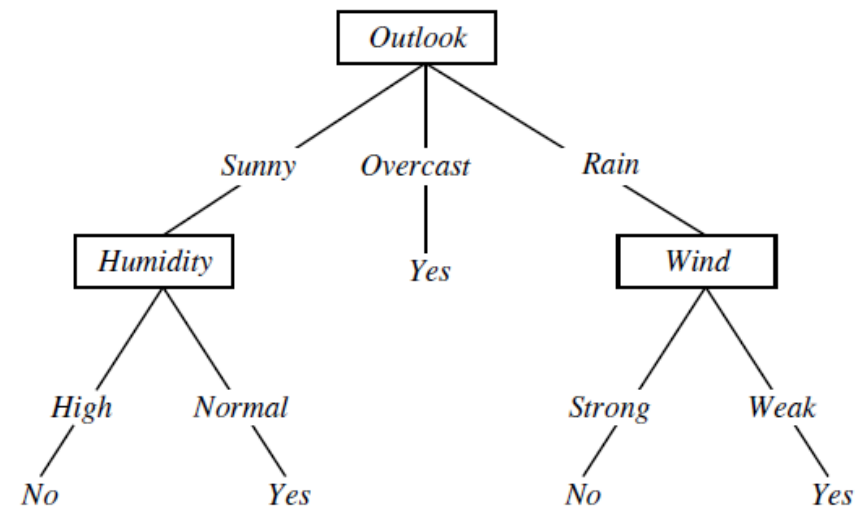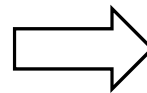 - …

Data stream classifiers, in this lecture

- Decision trees

- Naïve Bayes classifiers

(Batch) Decision Trees (DTs)

- Training set: D = {(x,y)}
  - predictive attributes: x=<x_1, x_2, ..., x_d>
  - class attribute: y={y_1, y_2, ..., 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

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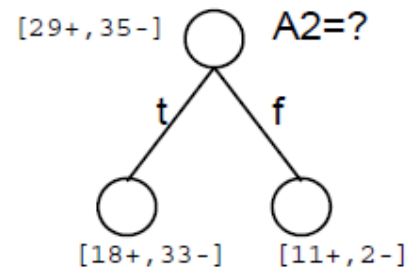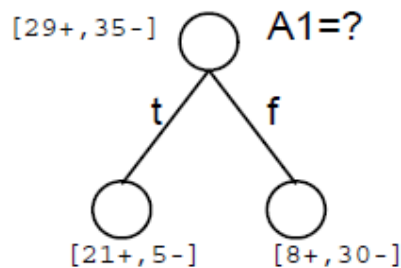
Goal: select the most "useful" attribute
- i.e., the one resulting in the purest partitioning

[29+,35-]   A1=?

t   f

[21+,5-]   [8+,30-]

[29+,35-]   A2=?

t   f

[18+,33-]   [11+,2-]

(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:

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

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:

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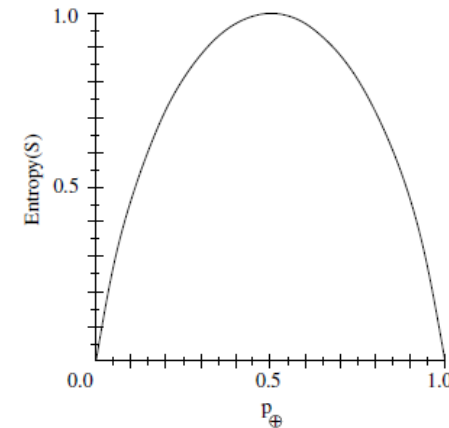
- Examples :

  - Let S: [9+,5-] $Entropy(S) = -\dfrac{9}{14}\log_2(\dfrac{9}{14}) - \dfrac{5}{14}\log_2(\dfrac{5}{14}) = 0.940$

  - Let S: [7+,7-] $Entropy(S) = -\dfrac{7}{14}\log_2(\dfrac{7}{14}) - \dfrac{7}{14}\log_2(\dfrac{7}{14}) = 1$

  - Let S: [14+,0-] $Entropy(S) = -\dfrac{14}{14}\log_2(\dfrac{14}{14}) - \dfrac{0}{14}\log_2(\dfrac{0}{14}) = 0$
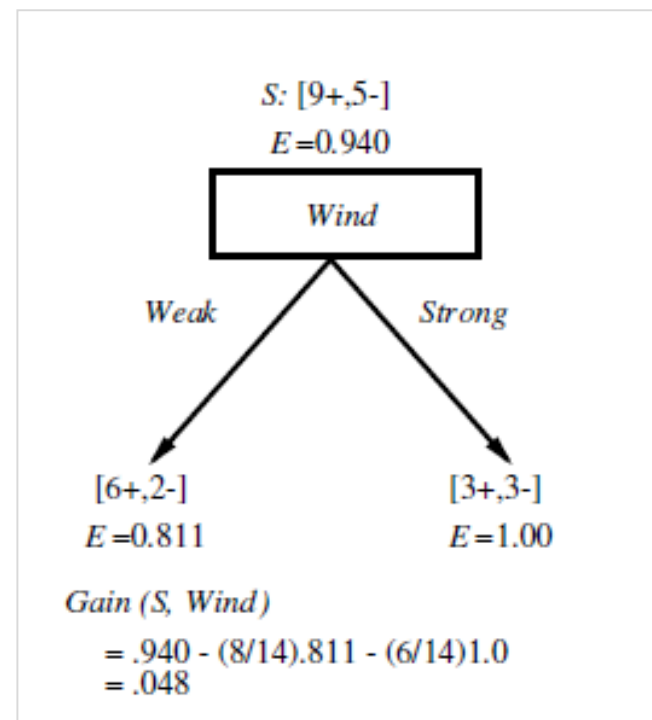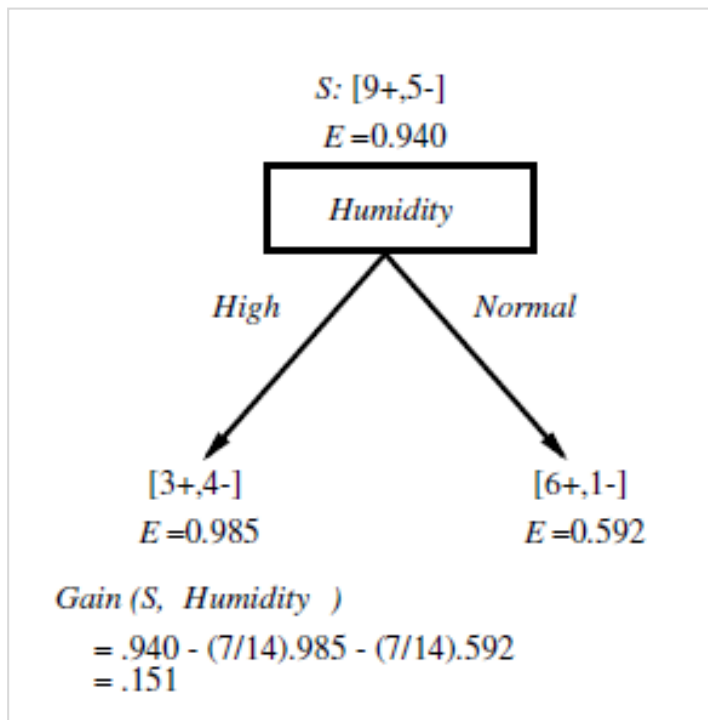
in the general case
(*k-classification problem*)

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

- 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?

S: [9+,5-]
E = 0.940

Humidity

High   Normal

[3+,4-]   [6+,1-]
E = 0.985   E = 0.592

Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

S: [9+,5-]
E = 0.940

Wind

Weak   Strong

[6+,2-]   [3+,3-]
E = 0.811   E = 1.00

Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048

- 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]

- 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-δ* the true mean of the variable, μ_r, is at least $\bar{r}$-ε, i.e., $P(\mu_r \geq \bar{r}\text{-}ε) = 1\text{-}δ$
- 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

- 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-δ*, $\Delta G \geq \overline{\Delta G} - \varepsilon > 0$.
  - 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 $\overline{\Delta G}$

- ■ **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.

- **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)
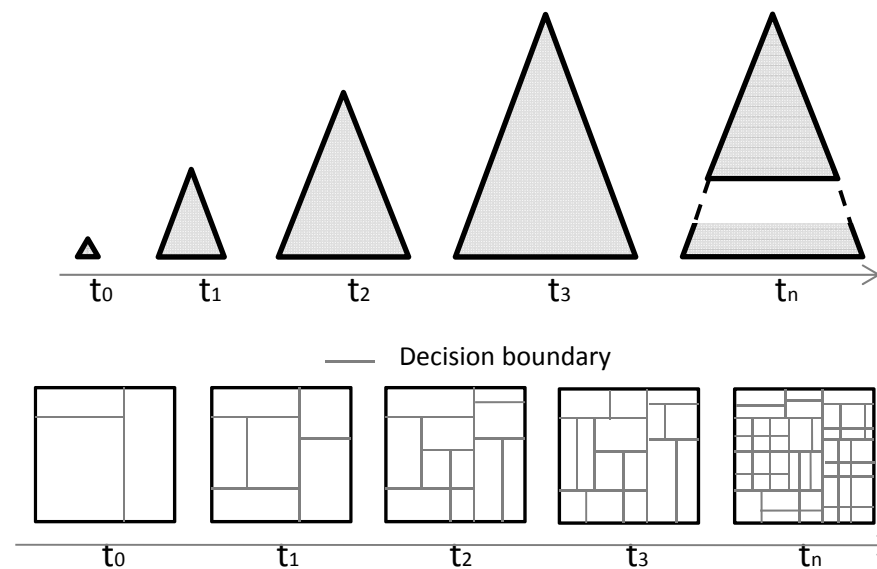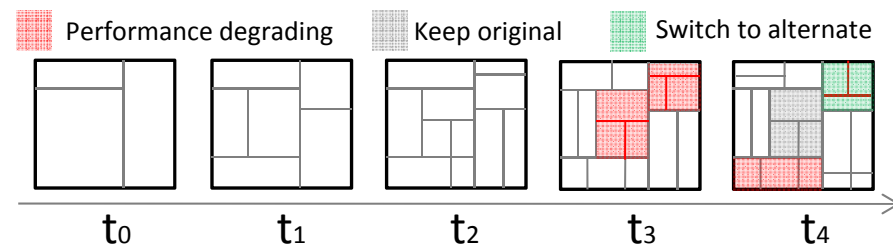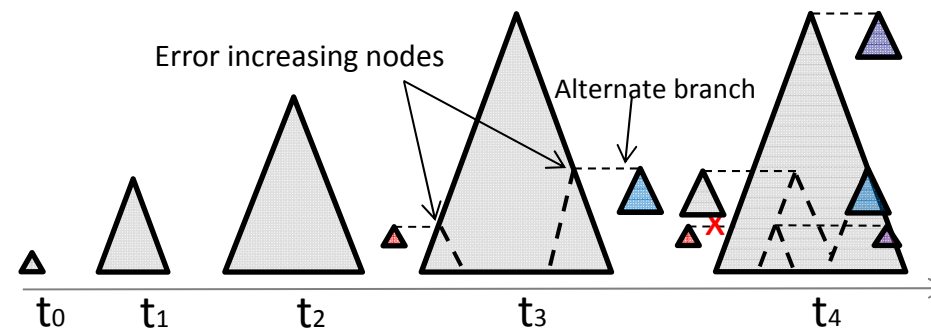

HT over time [Mahmud15]

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

Tree with maximum size

Reset

t_0 t_1 t_2 t_3 t_4

Decision boundary

t_0 t_1 t_2 t_3 t_4

ASHT over time [Mahmud15]
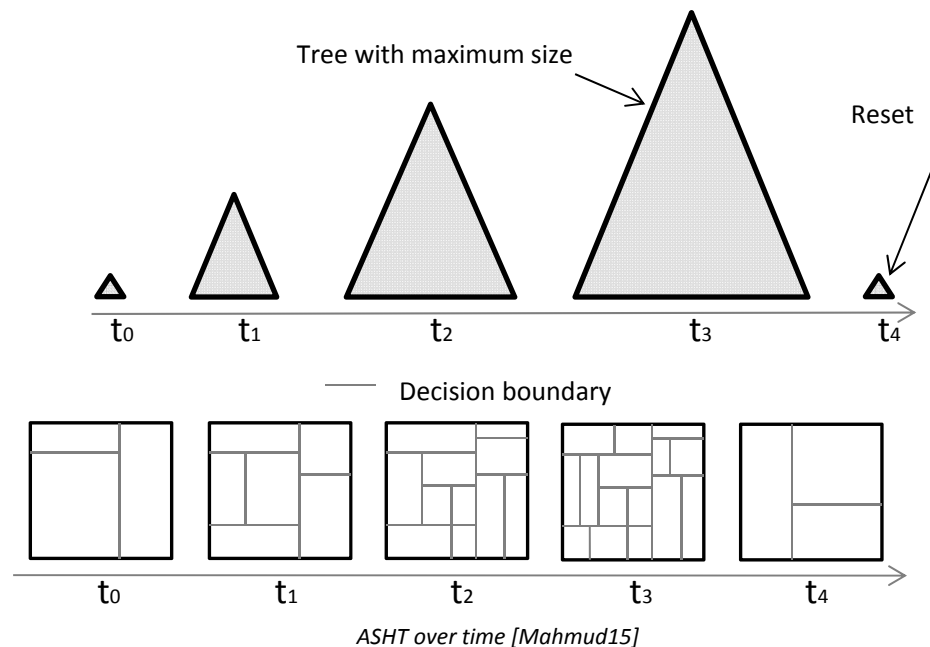
- The tree forgets
  - but, due to the reset, it looses all information learned thus far

- 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]

- 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
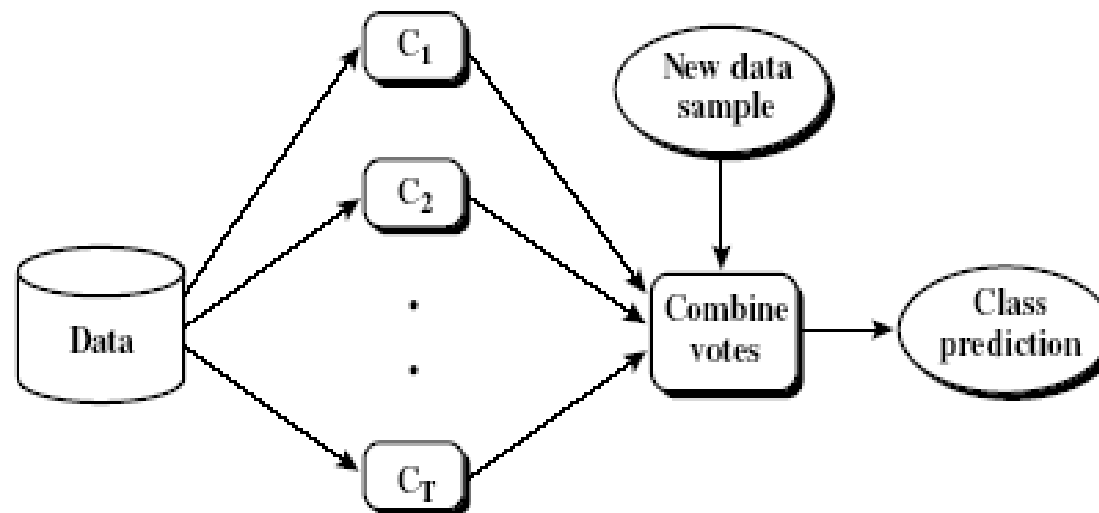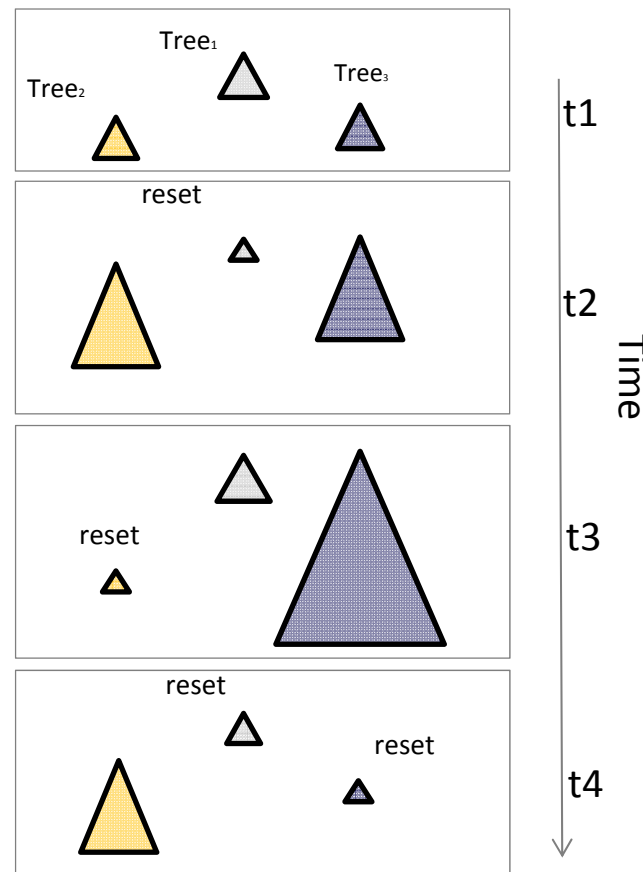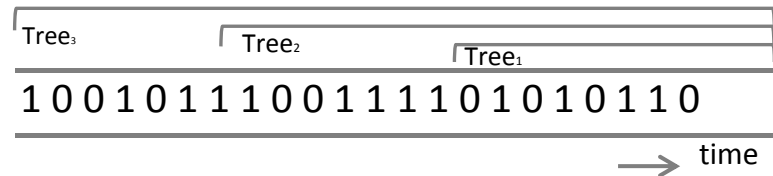
- Bagging

  – 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

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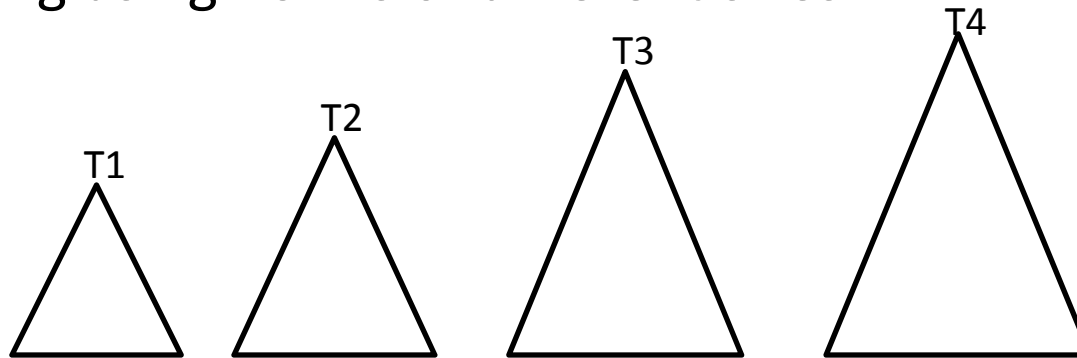
  - Smaller trees adapt more quickly to changes
  - Larger trees perform better during periods with no or little change
  - The max allowed size for the nth ASHT tree is twice the max allowed size for the $(n-1)^{th}$ tree.
  - Each tree has a weight proportional to the inverse of the square of its error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

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 classifiers, in this lecture

- Decision trees

- Naïve Bayes classifiers

- Given an instance X with attributes $(A_1 A_2 \ldots A_n)$
  - Goal is to predict class label c in C
  - Specifically, we want to find the value c of C that maximizes P(c|X)

$$c = \arg\max_{c \in C} P(X \mid c) P(c)$$

max a posteriori = the most likely class

likelihood

prior

- How can we estimate c?

  - Class prior P(c): How often c occurs?
    - Just count the relative frequencies in the training set
  - Instance likelihood P(X|c): What is the probability of an instance X given the class c?
    - $P(X|c) = P(A_1 A_2 \ldots A_n |c)$
    - i.e., the probability of an instance given the class is equal to the probability of a set of features given the class

- How to estimate $P(A_1 A_2 \ldots A_n | c)$?

- Assume independence among attributes A_i when class is given:

  - $P(A_1 A_2 \ldots A_n | C_j) = \prod P(A_i | c) = P(A_1 | c) P(A_2 | c) \ldots P(A_n | c)$

  Strong conditional
  independence assumption!!!

  - Can estimate $P(A_i | c)$ for all A_i and c in C based on training set

  - New point is classified to:

$$c = \arg\max_{c \in C} P(c) \prod P(A_i \mid c)$$

Naive Bayes classifier (batch): Example

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

Test instance X

Outlook	Temperature	Humidity	Wind	Play
Sunny	Cool	High	Strong	?

Observations

$$P(yes \mid X) = \frac{P(X \mid yes)P(yes)}{P(X)} = \frac{P(O="sunny" \mid yes)P(T="cool" \mid yes)P(H="high" \mid yes)P(W="strong" \mid yes)P(yes)}{P(X)}$$

$$P(O="sunny" \mid yes) = \frac{2}{9} \qquad P(T="cool" \mid yes) = \frac{3}{9} \qquad P(H="high" \mid yes) = \frac{3}{9} \qquad P(W="strong" \mid yes) = \frac{3}{9}$$
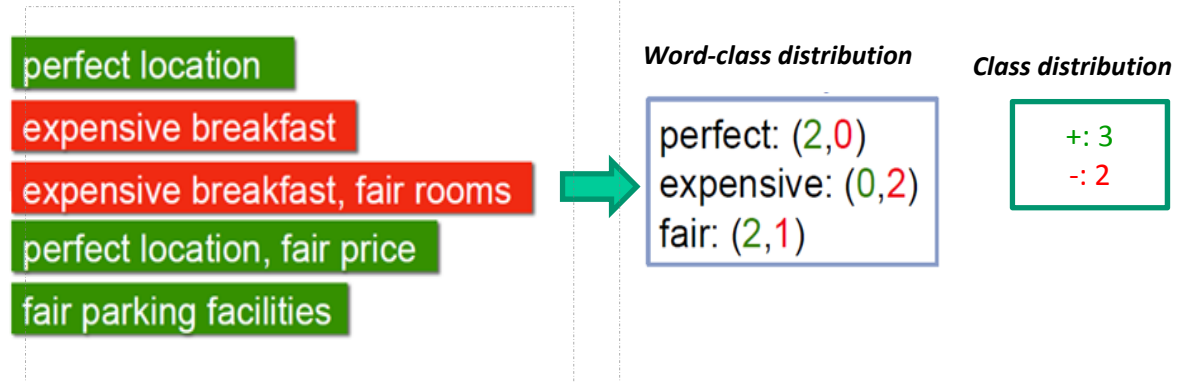
$$P(yes) = \frac{9}{14}$$

- How can we maintain the model estimates over time based on the stream?

- How can we include new instances in the model?

- How can we forget obsolete instances?

- In what follows, we assume a stream of documents (text data). The solutions though are not limited to text

Naive Bayes classifier (batch)

(fixed) **Training set D**

perfect location
expensive breakfast
expensive breakfast, fair rooms
perfect location, fair price
fair parking facilities

(fixed) **MNB Model**

Word-class distribution

perfect: (2,0)
expensive: (0,2)
fair: (2,1)

Class distribution
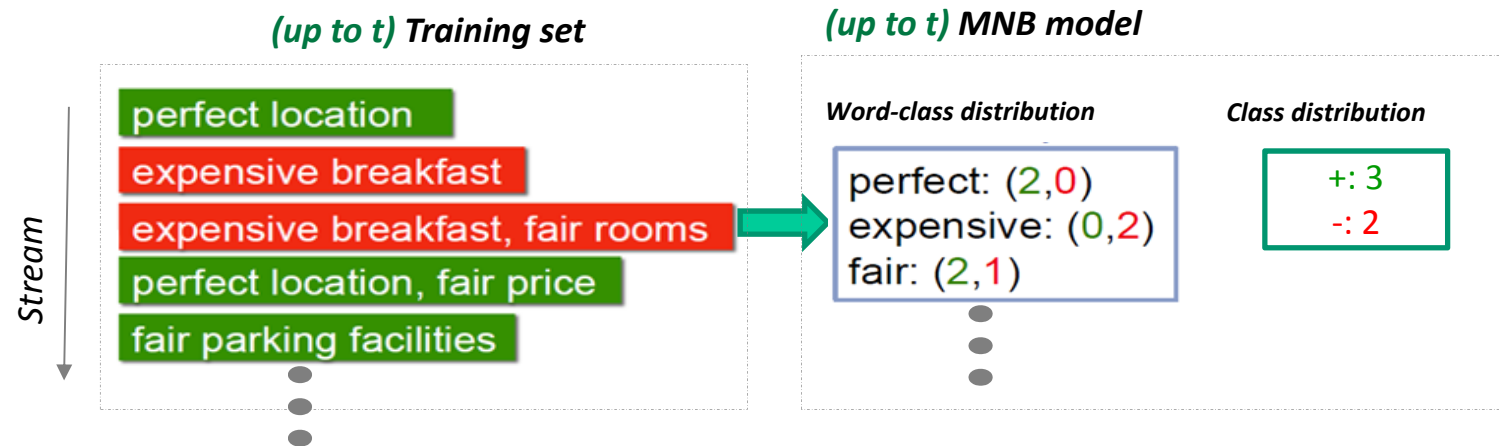
+: 3
-: 2

- Prediction, for a new document d:

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

Class-prior estimation

Word class conditional estimation

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

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

Fixed counts from D
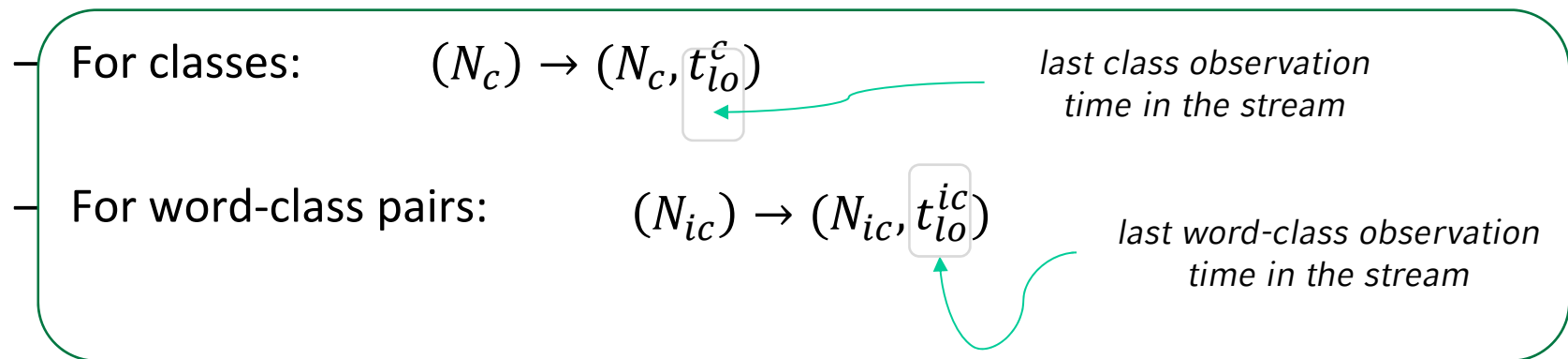
Naive Bayes classifier on stream – accumulative approach

(up to t) Training set

perfect location

expensive breakfast

expensive breakfast, fair rooms

perfect location, fair price

fair parking facilities

Stream

(up to t) MNB model

Word-class distribution

perfect: $(2,0)$
expensive: $(0,2)$
fair: $(2,1)$

Class distribution

+: 3
-: 2

- Prediction: based on model counts up to *t*

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

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

Accumulated counts from the beginning of the stream

- Model update: add *d* information to affected, N_c, N_{ic} in the model

- Long memory problem
  - Nothing is forgotten, new instances are always accumulated
    - ➔ difficult to adapt in times of change

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

  - For classes: $(N_c) \rightarrow (N_c, t_{lo}^c)$ *last class observation time in the stream*

  - For word-class pairs: $(N_{ic}) \rightarrow (N_{ic}, t_{lo}^{ic})$ *last word-class observation time in the stream*

  - Timestamp propagation: from documents $\rightarrow$ 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

- Gradual ageing – exponential ageing 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* timeunits

- Updated temporal probability estimates

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

What exactly is stored in the model?

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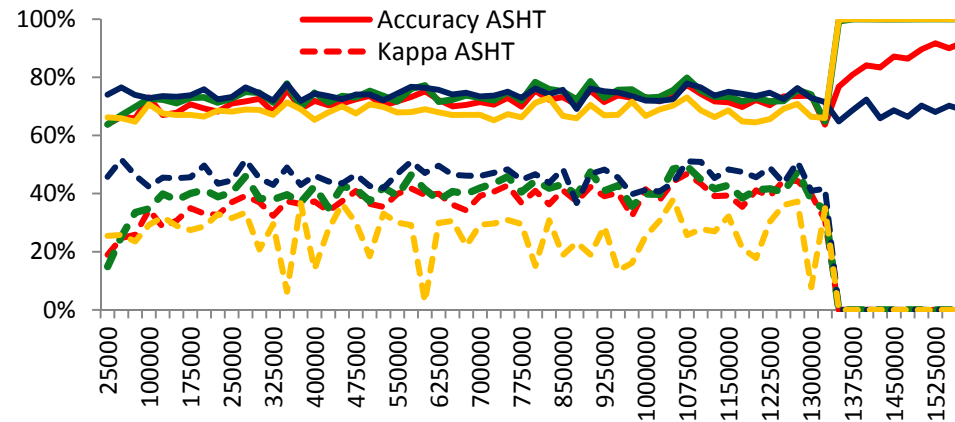
ageing effect

fadingMNB

- N_c and N_{ic} accumulated over the stream
- a-posteriori ageing over the accumulated counts

  − Gaps in observations are penalized
  − But, as soon as an observation re-appears, all its previous weight is revived.

aggressiveFadingMNB

- the faded counts are stored in the model
- ageing over the faded counts

  − More drastic ageing
  − Gaps in observations are penalized, even if we make the same observation again later
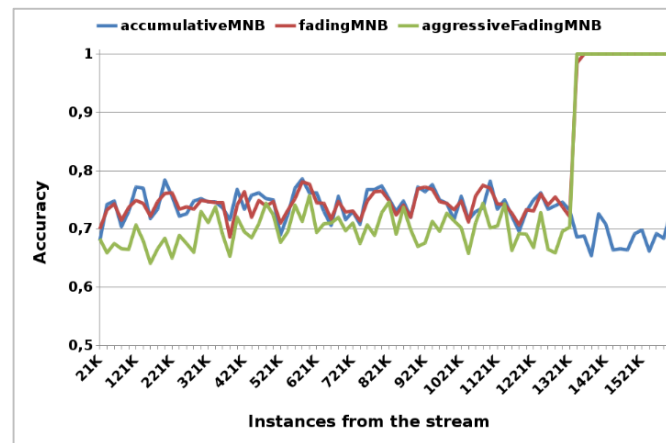
- Easy model maintenance when adding a new document *d* at *t*:
  – Update the model counts based on *d*
  – Set the last observation time (*lo*) in the affected entries to *t*

The ageing effect

- No-ageing (accumulativeMNB)

Source: [Sinelnikova12]

- Effect of ageing (ageing-based MNB)

Source: [WagnerEtAl15]

- Naïve Bayes classifiers are ideal choices for streams

  – Popular, simple, powerful

  – allows for the seamless adaptation of the model based on new instances

  – deals with dynamic feature spaces

- AccumulativeMNB counts for new instances but does not forget

  – Difficult to adapt to changes

- Ageing-based MNBs provide a temporal model that allows for ageing of the model based on the recency of the observations

- 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

- 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:

<div align="center">

Predicted 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'	

</div>

Actual class

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

(batch) Classifier evaluation measures

- ## Accuracy/ Recognition rate:

  - % of test set instances correctly classified

	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'	

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

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)

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

- Holdout method
  - Given data is randomly partitioned into two independent sets
    - o Training set (~2/3) for model construction, Test set (~1/3) for evaluation
    - (-) You have less data for training, problematic for small datasets.

- Cross-validation (*k*-fold cross validation, usually *k* = 10)
  - Randomly partition the data into *k* mutually exclusive subsets D_1, …, D_k each approximately equal size.
  - Training and testing is performed *k* times
    - o At the *i*-th iteration, use D_i as test set and the rest *k*-1 partitions as training set
  - Accuracy is the avg accuracy over all iterations
  - (+) All data are exploited for training and testing

- Check Lecture 4, KDD I for more evaluation methods, their pros and cons!

- ## Holdout evaluation

  - 2 separate datasets for training (~70% - 80% of the dataset) and testing (~20%-30% of the dataset)

  - Online model training on the training set

  - Online model testing on test set

    - o *Static* test set!!!

- ## Prequential evaluation (or, interleaved test-then-train evaluation)

  - One dataset for training and testing

  - Models are first tested then trained in each instance

    - o Test set is *dynamic*!!!

    - o But it assumes the direct availability of the labels of the arriving instances for testing.

- Accuracy, over a sliding window

  - % of test set instances correctly classified

- Cohen's kappa measure, over a sliding window   *Agreement by chance*

  - normalizes the accuracy of a classifier p_0 by that of a chance predictor p_c

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

Observed agreement

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

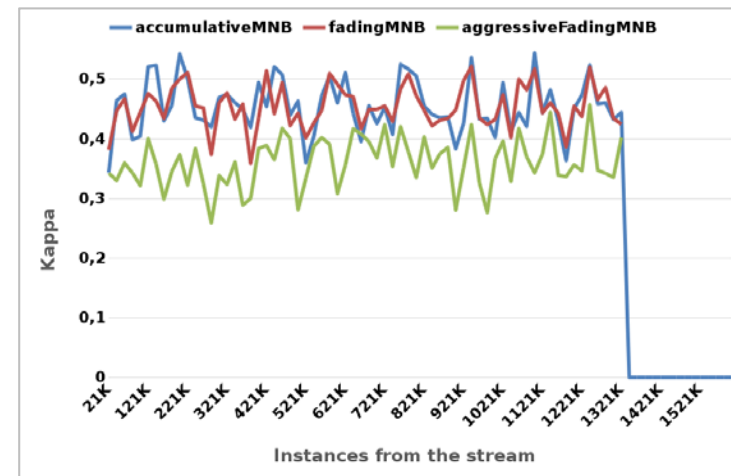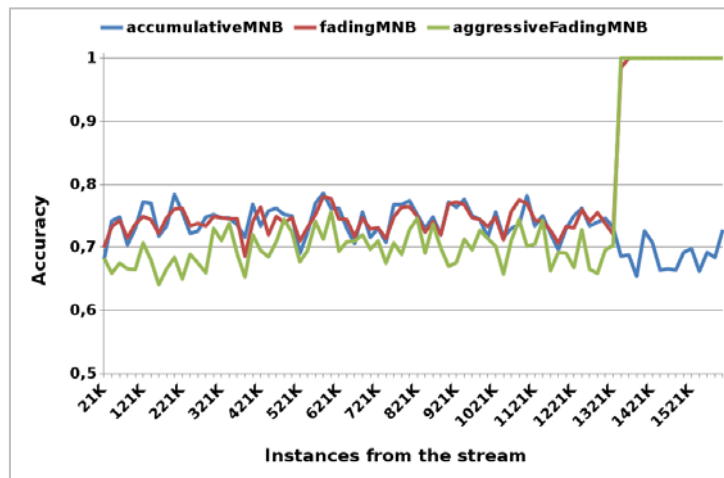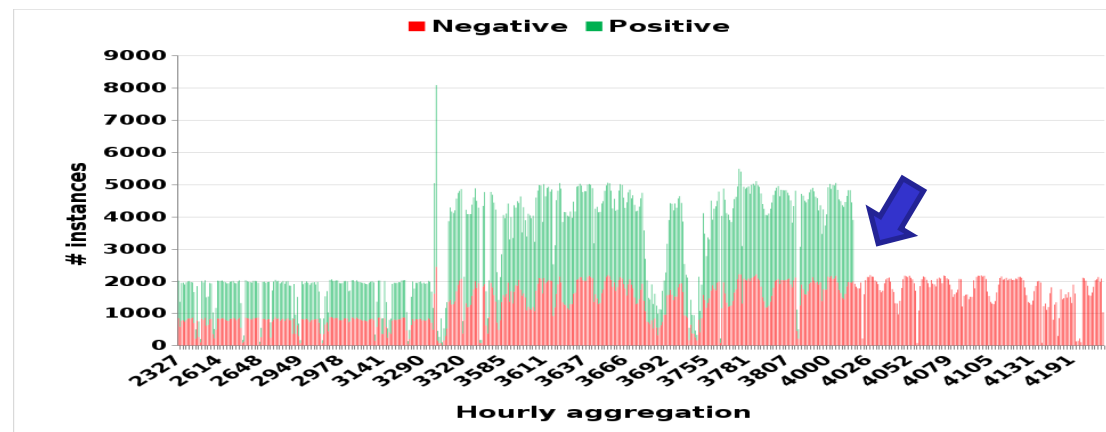| Kappa value	Classifier's performance
0%-20%	bad
21%-40%	fair
41%60%	moderate
61%-80%	substantial
81%-100%	(almost) perfect

$p_o = (15+70)/100 = 0.85$
$p_c = (20/100)*(25/100) + (80/100)*(75/100) = 0.65$

agreement by chance in "yes"   *agreement by chance in "no"*

$k = (0.85-0.65)(1-0.65) = 0.57$

Accuracy vs Kappa

- Prequential evaluation, hourly-aggregated stream [WagnerEtAl15]

- 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

- 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.