Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Lecture notes

Knowledge Discovery in Databases II

Winter Semester 2013/2014

Lecture 9: Stream classification

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

http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_II_(KDD_II)

Outline

- Motivation
- Data streams
- Data stream clustering
- Data stream clustering algorithms
- Data stream classification
- Data stream classifiers
- Data stream classifiers evaluation

Motivation

- So far: focus on batch learning using small datasets
- Batch learning:
  - Whole training data is available to the learning algorithm
  - Data instances are processed multiple times
  - e.g. k-Means clusterer (c.f., slide 4), ID3 classifier (c.f., slide 5)
- Assumption:
  - Instances are generated by some stationary probability distribution

DATABASE SYSTEMS GROUP

Motivation

LMU

- But, most interesting applications come from dynamic environments where data are collected over time
  - e.g., customer transactions, call records, customer click data.
- Batch learning is not sufficient anymore …
- Algorithms should be able to incorporate new data
  - There are algorithms that are incremental by nature, e.g. kNN classifiers, Naïve Bayes classifiers
  - But the majority of the algorithms need changes to make incremental induction, e.g., ID3, k-Means, DBSCAN.
- Algorithms should be able to deal with non-stationary data generation processes
  - Incorporate concept drift
  - Forget outdated data and adapt to the most recent state of the data

Motivation

- Cluster evolution

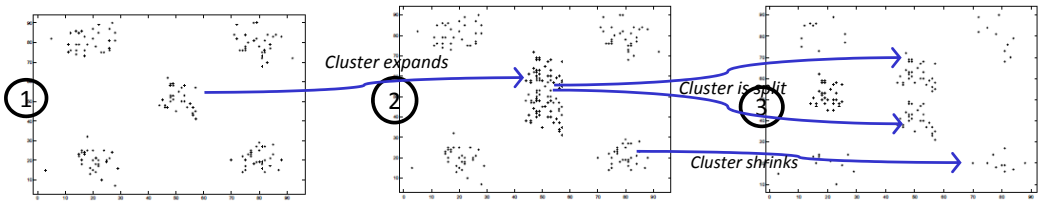


Figure: Data records at three consecutive time stamps, the clustering gradually changes
(from: *MONIC - Modeling and Monitoring Cluster Transitions*, Spiliopoulou et al, KDD 2006)

- Concept drift

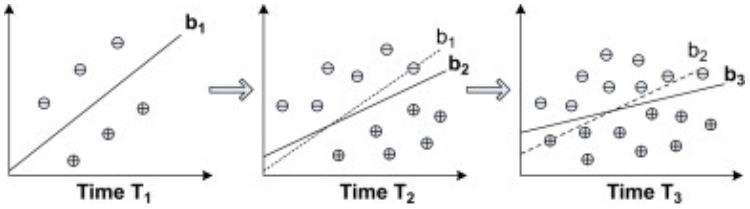
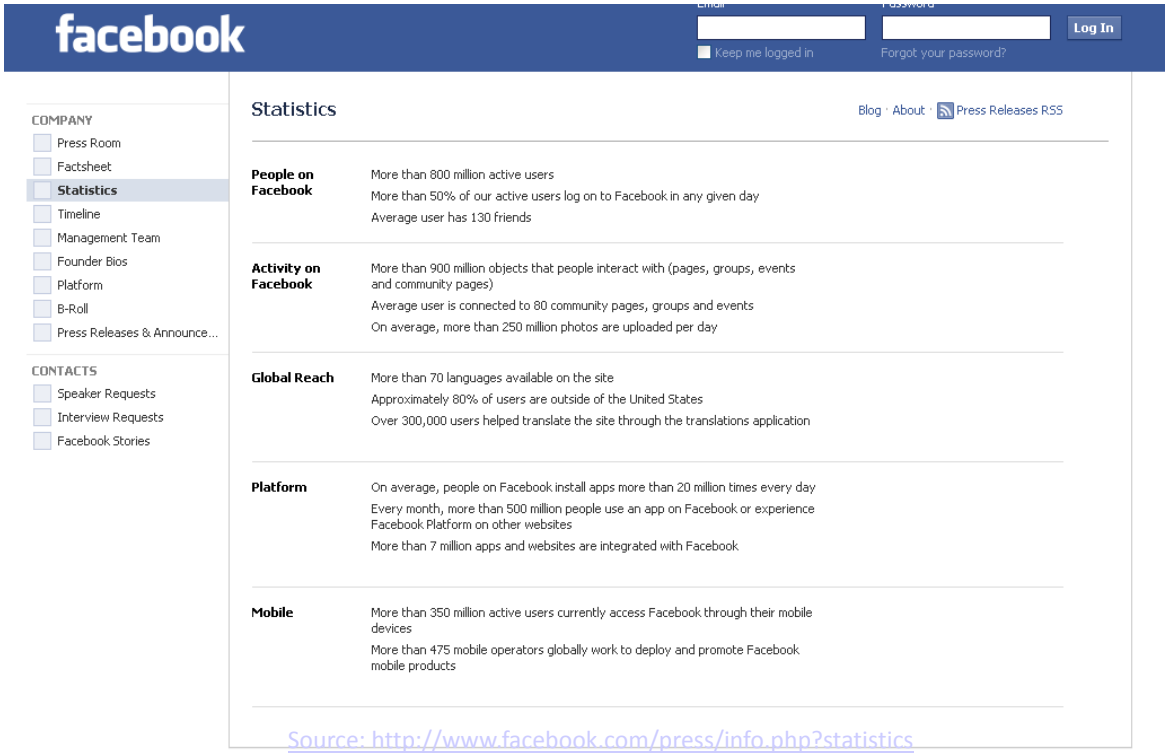


Fig. 1. An illustration of concept drifting in data streams. In the three consecutive time stamps T_1, T_2 and T_3, the classification boundary gradually drifts from b_1 to b_2 and finally to b_3.
(from: *A framework for application-driven classification of data streams*, Zhang et al, Journal Neurocomputing 2012)

Example applications

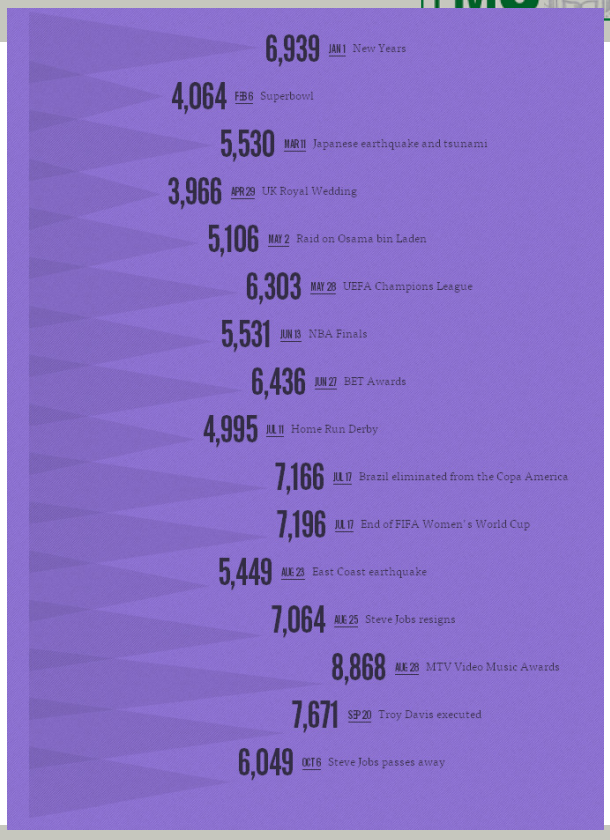
- Banks (credit card transactions, loan applications,…)
- Telecommunication (call records, sms, www usage,…)
- Health care systems (customer records in a hospital,…)
- Retail industry (transactions in a supermarket,…)
- WWW (content, interactions, TCP/IP traffic, customer click data,…)
- Science (experiment results,…)
- …

Example application: Facebook



Source: http://www.facebook.com/press/info.php?statistics

Example application: Twitter



Source: http://blog.kissmetrics.com/twitter-statistics/

Example application: CERN

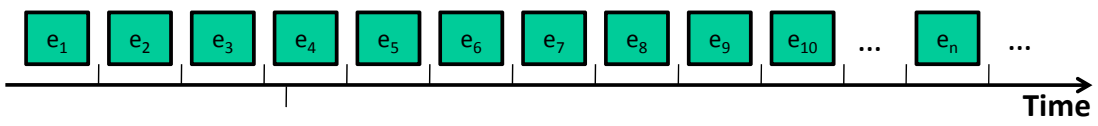
- Experiments at CERN are generating an entire petabyte (1PB=10^6 GB) of data every second as particles fired around the Large Hadron Collider (LHC) at velocities approaching the speed of light are smashed together
- "We don't store all the data as that would be impractical. Instead, from the collisions we run, we only keep the few pieces that are of interest, the rare events that occur, which our filters spot and send on over the network," he said.
- This still means CERN is storing 25PB of data every year – the same as 1,000 years' worth of DVD quality video – which can then be analyzed and interrogated by scientists looking for clues to the structure and make-up of the universe.

Source: http://public.web.cern.ch/public/en/LHC/Computing-en.html
Source: http://www.v3.co.uk/v3-uk/news/2081263/cern-experiments-generating-petabyte

Dynamic data/ Data streams

"Τα πάντα ῥεῖ καὶ οὐδὲν μένει"
("Τα panta rhei kai ouden menei")
"Everything flows, nothing stands still"
Heraclitus (535-475 BC)

- Data evolve over time as new data arrive and old data become obsolete
- We can distinguish between:
  - Dynamic data arriving at a low rate
    - Incremental methods might work for such cases
  - Data streams: possible infinite sequence of elements arriving at a rapid rate
    - new methods are required to deal with the amount and complexity of these data



An example dataset: Network traffic data stream

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.

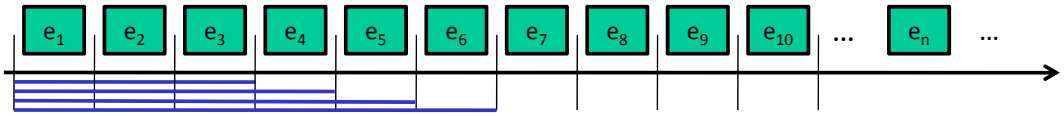
More on this dataset: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Challenges & requirements

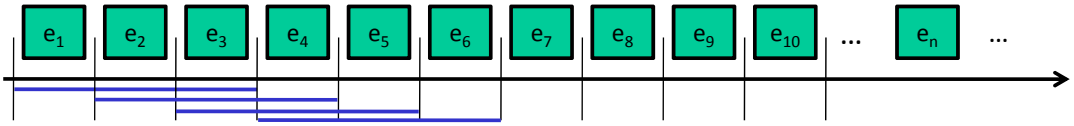
- Data Mining over stream data is even more challenging:
  - Huge amounts of data → only a small amount can be stored in memory
  - Arrival at a rapid rate → no much time for processing
  - The generative distribution of the stream might change over time rather than being stationary→ adapt and report on changes
- Requirements for stream mining algorithms:
  - Use limited computational resources:
    - Bounded memory
    - Small processing time
  - No random access to the data
    - Only 1 look at the data (upon their arrival)

Data ageing and window models I

- Usually we are not interested in the whole history of the stream but only in the recent history.
- There are different *ageing/weighting mechanisms or window models* that reflect which part of the stream history is important
  - Landmark window:
    - Nothing is forgotten.
    - All points have a weight w=1.

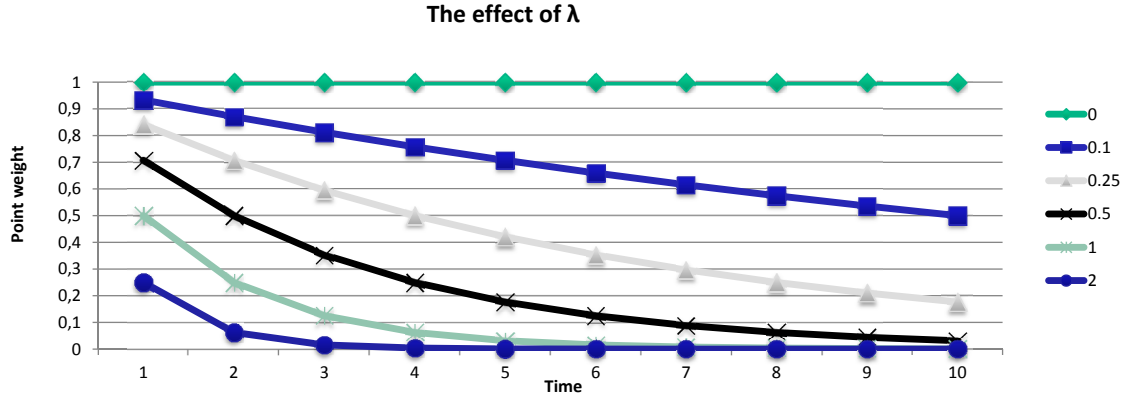


  - Sliding window:
    - Remember only the *n* more recent entries.
    - All points within the window have a weight w=1, for the rest: w=0.



Window models II

  - Damped window:
    - 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).
    - A widely used fading function in temporal applications is the exponential fading function: $f(t)=2^{-\lambda t}$, $\lambda>0$.
      - The decay rate λ determines the importance of historical data
      - The higher the value of λ, the lower the importance of old data



Mining over data streams

- We focus on:
  - Data stream clustering
  - Data stream classification

Clustering data streams

- One of the core tasks in DM
  - Used as either a standalone tool or as a preprocessing tool
- The (batch) clustering problem:
  - Given a set of measurements, observations, etc., the goal is to group the data into groups of similar data (clusters)

- The data stream clustering problem:
  - Maintain a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time.
- This implies:
  - Incremental clustering
  - Maintaining cluster structures that evolve over time
  - Working with summaries instead of raw data

Challenges & Requirements for data stream clustering

- Traditional clustering methods require access upon the whole dataset
  - Online maintenance of clustering
- The underlying population distribution might change: drifts/ shifts of concepts
  - One clustering model might not be adequate to capture the evolution
- The role of outliers and clusters are often exchanged in a stream
  - Clear and fast identification of outliers

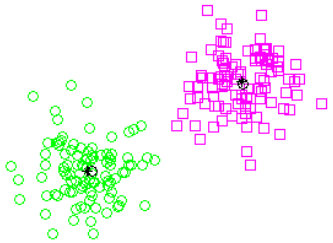
A taxonomy of stream clustering approaches *(& representative methods)*

	Static clustering	Dynamic/Stream clustering
Partitioning methods	• k-Means • k-Medoids	• Leader • Simple single pass k-Means • STREAM k-Means • CluStream
Density-based methods	• DBSCAN • OPTICS	• DenStream • incDBSCAN * • incOPTICS *
Grid-based methods	• STING	• DStream

(*) These methods require access to the raw data (this access might be limited though)

Partitioning methods

- Goal: Construct a partition of a set of objects into k clusters
  - e.g. k-Means, k-Medoids



- Two types of methods:
  - Adaptive methods:
    - Leader (Spath 1980)
    - Simple single pass k-Means (Farnstrom et al, 2000)
    - STREAM k-Means (O'Callaghan et al, 2002)
  - Online summarization - offline clustering methods:
    - CluStream (Aggarwal et al, 2003)

Leader (Spath 1980)

- The simplest single-pass partitioning algorithm
- Whenever a new instance p arrives from the stream
  - Find its closest cluster (leader), c_{clos}
  - Assign p to c_{clos} if their distance is below the threshold d_{thresh}
  - Otherwise, create a new cluster (leader) with p

\+ 1-pass and fast algorithm

\+ No prior information on the number of clusters

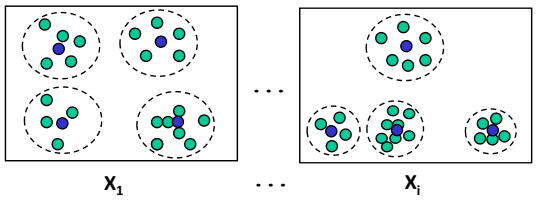
– Unstable algorithm

– It depends on the order of the examples

– It depends on a correct guess of d_{thresh}

STREAM k-Means
(O'Callaghan et al, 2002)

- An extension of k-Means for streams
  - The iterative process of static k-Means cannot be applied to streams
  - Use a buffer that fits in memory and apply k-Means locally in the buffer
- Stream is processed in chunks X_1, X_2…, each fitting in memory
  - For each chunk X_i
    - Apply *k*-Means locally on X_i (retain only the *k* centers)
    - $X' \leftarrow i*k$ weighted centers obtained from chunks X_1 … X_i
      - Each center is treated as a point, weighted with the number of points it compresses
    - Apply k-Means on X' output the k centers



CluStream
(Aggarwal et al, 2003)

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

CluStream:
the micro-cluster summary structure

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

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

$$\Sigma_{i=1}^{N} X_i^2 \qquad \Sigma_{i=1}^{N} X_i \qquad \Sigma_{i=1}^{N} T_i^2 \qquad \Sigma_{i=1}^{N} T_i$$

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

CluStream:
the algorithm

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

CluStream:
Initialization step

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

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

CluStream:
Online step

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

CluStream:
Periodic micro-cluster storage

- Micro-clusters snapshots are stored at particular moments
- If current time is t_c and user wishes to find clusters based on a history of length h
  - Then we use the subtractive property of micro-clusters at snapshots t_c and t_c-h
  - In order to find the macro-clusters in a history or time horizon of length h
- How many snapshots should be stored?
  - It will be too expensive to store snapshots at every time stamp
  - They are stored in a pyramidal time frame

- It is an effective trade-off between the storage requirements and the ability to recall summary statistics from different time horizons.

CluStream:
Offline step

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

CluStream: overview

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

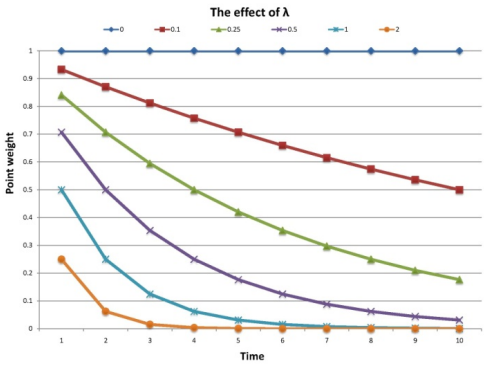
– Fixed number of micro-clusters maintained over time
– Sensitive to outliers/ noise

Density based methods

- Clusters as regions of high density surrounded by regions of low density (noise)
  - Density is measured locally, in the ε-neighborhood of each point
    - e.g. DBSCAN, OPTICS
- Very appealing for streams
  - No assumption on the number of clusters
  - Discovering clusters of arbitrary shapes
  - Ability to handle outliers and noise
- But, they miss a clustering model (or it is to complicated)
  - Clusters are represented by all their points
- Solution: Describe clusters as set of summaries
  - DenStream (Cao et al, 2006)

DenStream
(Cao et al, 2006)

- The online-offline rationale is followed:
  - Online summarization as new data arrive over time
    - Core, potential core and outlier micro–clusters
  - Offline clustering over the summaries to derive the final clusters
    - A modified version of DBSCAN over the summaries
- Data are subject to ageing according to the exponential ageing function (damped window model)
  - $f(t)=2^{-\lambda t}$, $\lambda>0$
  - λ the decay rate
  - higher λ, less important the historical data comparing to more recent data



DenStream:
summarizing the stream

- The micro-cluster summary at time t for a set of d-dimensional points (p_1, p_2, …, p_n) arriving at time points T_1, T_2, …, T_n is:

$$MC = (CF^1,\ CF^2,\ w)$$

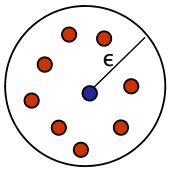
$$\sum_{i=1}^{n} f(t-T_i)p_i \qquad \sum_{i=1}^{n} f(t-T_i)p_i^2 \qquad \sum_{i=1}^{n} f(t-T_i)$$

- Easy computation of basic measures:
  - Center: $c = \frac{CF^1}{w}$
  - Radius: $r = \sqrt{\frac{CF^2}{w} - \left(\frac{CF^1}{w}\right)^2}$
- A micro-cluster summary c_p can be maintained incrementally
  - If a new point p is added to c_p: $c_p = (CF^2+p^2, CF^1+p, w+1)$
  - If no point is added to c_p for time interval δt:
    - $c_p = (2^{-\lambda\delta t}*CF^2, 2^{-\lambda\delta t}*CF^1, 2^{-\lambda\delta t}*w)$

DenStream:
core, potential core & outlier summaries

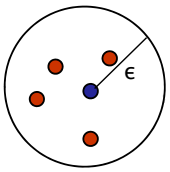
- Core (or dense) micro-clusters
  - $(w \geq \mu) \,\&\, (r \leq \varepsilon)$

ϵ: the radius threshold
μ: the weight threshold

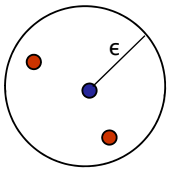


core-micro-cluster

- But, in an evolving stream, the role of clusters and outliers often interchange:
  - Should provide opportunity for the gradual growth of new clusters
  - Should promptly get rid of the outliers
- Potential core micro-clusters
  - $(w \geq \beta*\mu) \,\&\, (r \leq \varepsilon), 0 < \beta \leq 1$
- Outlier micro-clusters
  - $(w < \beta*\mu) \,\&\, (r \leq \varepsilon), 0 < \beta \leq 1$



p-core-micro-cluster



o-core-micro-cluster

DenStream:
the algorithm

- Two lists of p-micro-clusters and o-micro-clusters are maintained over time
- Initialize: apply DBSCAN over *initPoints* → p-micro-clusters
- Online step as a new point *p* arrives from the stream
  - Try to assign it to its closest p-micro-cluster
  - If this is not possible, try to assign it to its closest o-micro-cluster
    - Check if this o-micro-cluster can be upgraded
  - If both are not possible, create a new o-micro-cluster with p
- Periodic micro-cluster maintenance based on data ageing
- Offline macro-clustering
  - On demand, upon user request apply a modified version of DBSCAN over p-micro-clusters to derive the final clusters

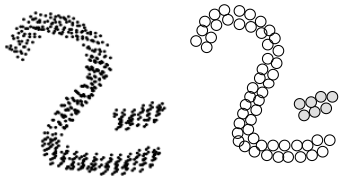
DenStream: online step

Two lists of p-micro-clusters and o-micro-clusters are maintained over time

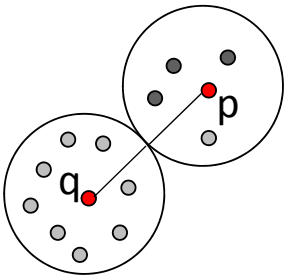
- When a new point *d* arrives
  - Find its closest p-micro-cluster *pclu*
    - If the updated radius of *pclu* ≤ ε, merge *d* to *pclu*
  - o.w. find its closest o-micro-cluster *oclu*
    - If the updated radius of *oclu* ≤ ε, merge *d* to *oclu*
    - Check if *oclu* can be upgraded to a p-micro-cluster (if now $w \geq \beta * \mu$)
  - o.w., create a new o-micro-cluster with *d*
- Periodic p- and o-micro-clusters update due to ageing
  - Delete a p-micro-cluster when $w < \beta * \mu$
  - Delete an o-micro-cluster when $w < \xi$ (expected weight based on its creation time)
    - The longer an o-micro-cluster exists, the higher its weight is expected to be
  - The number of p- and o-micro-clusters is bounded

DenStream: offline step

- Upon request, apply a variant of DBSCAN over the set of online maintained p-micro-clusters
  - Each p-micro-cluster c_p is treated as a virtual point located at the center of c_p with weight w.



- Core-micro-clusters (redefined)
- Directly density reachable (redefined)
  - c_p is directly density reachable from c_q if:
    - c_q is a c-micro-cluster and
    - $dist(c_p,c_q) \leq 2\varepsilon$ (i.e. they are tangent or intersecting)



- Density reachable (redefined)
  - A p-micro-cluster c_p is density reachable from a p-micro-cluster c_q if there is a chain of p-micro-clusters $c_{p1}=c_q$, cp_2, …, $c_{pn}=c_p$.
- Density connected (redefined)

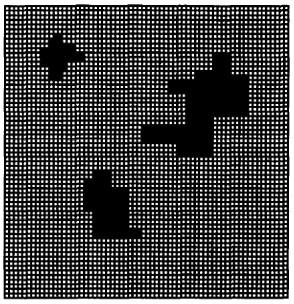
DenStream
overview

+ DenStream clusters large evolving data stream
+ Discover clusters of arbitrary shapes, following the density-based paradigm
+ No assumption on the number of clusters
+ Noise/ outlier handling

– The choice of the parameters ε, β, μ
– Constant parameters over time, what about clusters with different density

Grid based methods

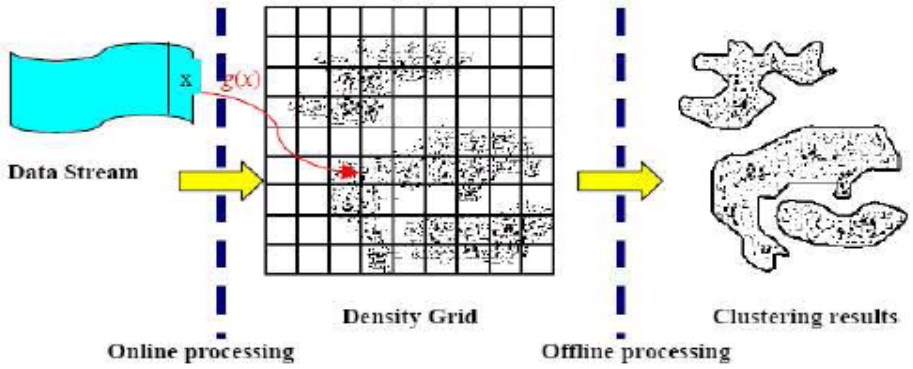
- A grid structure is used to capture the density of the dataset.
  - A cluster is a set of connected dense cells
  - e.g. STING
- Appealing features
  - No assumption on the number of clusters
  - Discovering clusters of arbitrary shapes
  - Ability to handle outliers



- In case of streams
  - The grid cells “constitute” the summary structure
  - Update the grid structure as the stream proceeds
  - DStream (Chen & Tu, 2007)

Dstream
(Chen & Tu, 2007)

- The online-offline rationale is followed:
  - Online mapping of the new data into the grid
  - Offline computation of grid density and clustering of dense cells



Dstream:
Summarizing the stream into the grid

- Data ageing (damped window model):
  - D(x,t) =λ^{t-tc}, t_c is the arrival time for point x, t is the current timepoint
  - λ in (0,1) is the *decay factor*
- The density of a grid cell *g* at time *t*: $D(g,t) = \sum_{x \in E(g,t)} D(x,t)$
- The characteristic vector of a grid cell g is defined as:

(t_g, t_m, D, label, status)

Last update time | last time g was removed from grid_list | Last density update | the class label of the grid | {sporadic, normal}

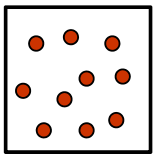
- The grid density can be updated incrementally

$D(g,t_n) = \lambda^{t_n - t_l} D(g,t_l) + 1$ t_n: the new record arrival time; t_l: the last record arrival

Dstream:
Dense, Sparse and Transitional grid cells

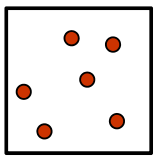
- The density of a grid is constantly changing over time.
- Dense grid cells

$$D(g,t) \geq \frac{C_m}{N(1-\lambda)} = D_m \qquad C_m > 1$$



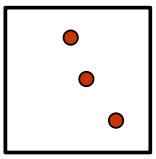
- Transitional grid cells

$$D(g,t) \leq \frac{C_l}{N(1-\lambda)} = D_l \quad 0 < C_l < 1$$



- Sparse grid cells

$$\frac{C_l}{N(1-\lambda)} \leq D(g,t) \leq \frac{C_m}{N(1-\lambda)}$$



N: #cells in the grid

DStream:
the algorithm


```
1.  procedure D-Stream
2.      t_c = 0;
3.      initialize an empty hash table grid_list;
4.      while data stream is active do
5.          read record x = (x_1, x_2, ..., x_d);
6.          determine the density grid g that contains x;
7.          if(g not in grid_list) insert g to grid_list;
8.          update the characteristic vector of g;
9.          if t_c == gap then
10.             call initial_clustering(grid_list);
11.         end if
12.         if t_c mod gap == 0 then
13.             detect and remove sporadic grids from grid_list;
14.             call adjust_clustering(grid_list);
15.         end if
16.         t_c = t_c + 1;
17.     end while
18. end_procedure
```

Grid update

Initialization

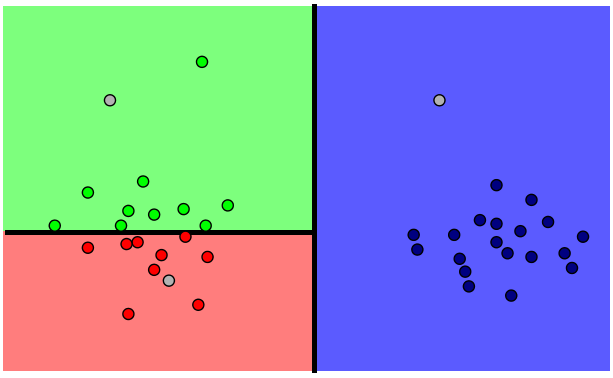
Clustering

DStream
overview

+ DStream clusters large evolving data stream
+ It can discover clusters of arbitrary shapes
+ No assumption on the number of clusters
+ Distinguishes noise and outliers
+ The grid provides a level of abstraction over the data

– The choice of the grid parameters
– Fixed grid parameters

Classification



- Screw
- Nails
- Paper clips

} Training data

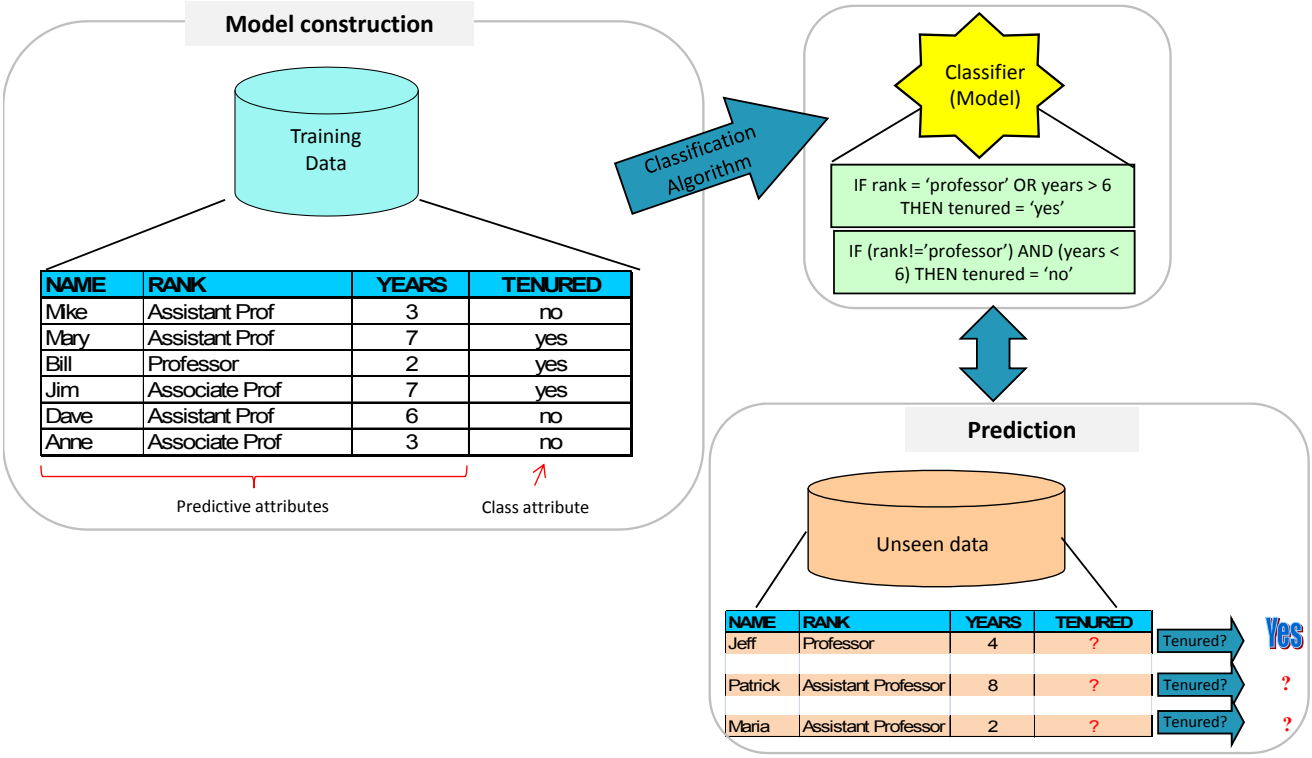
- New object

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

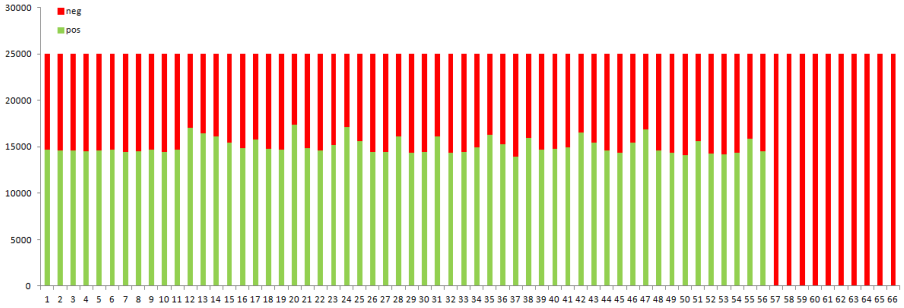


Classification over dynamic data I

- 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
  - 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
  - update existing models by incorporating new data
  - deal with non-stationary data generation processes (concept drift)
  - subject to:
    - Resource constraints (processing time, memory)
    - Single scan of the data (one look, no random access)

Classification over dynamic data II

- Non-stationary data (evolving distribution)



Evolving data distribution (from the BA of Alina's Sinelnikova , "*Sentiment analysis in the Twitter stream*", LMU 2012.)

- Concept drift

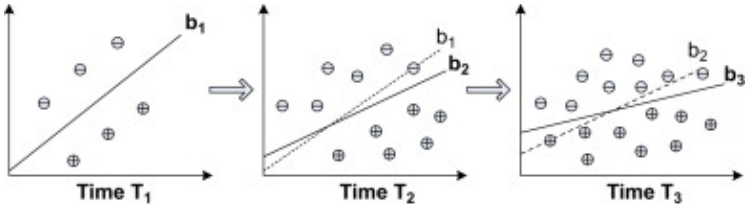


Fig. 1. An illustration of concept drifting in data streams. In the three consecutive time stamps T_1, T_2 and T_3, the classification boundary gradually drifts from b_1 to b_2 and finally to b_3.
(from: *A framework for application-driven classification of data streams*, Zhang et al, Journal Neurocomputing 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
- 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

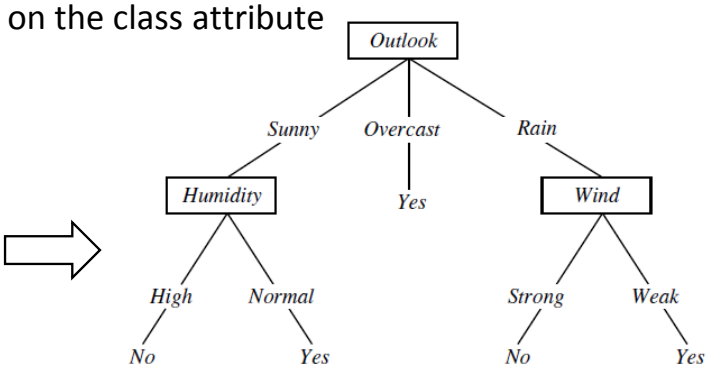
- Decision trees
- Ensemble methods

(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



(Batch) DTs: Splitting attribute

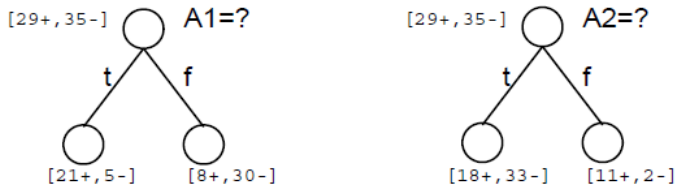
- Basic algorithm (ID3, Quinlan 1986)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root node

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

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

(check Lecture 4, KDD I)

  - But, which attribute is the best?



Goal: select the most useful attribute for classifying examples.
• useful → the resulting partitioning is as pure as possible
• pure partition: all its instances belong to the same class.

(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

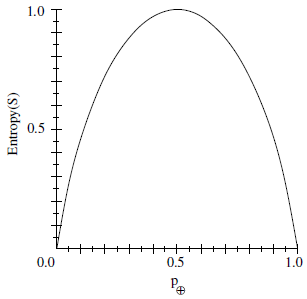
- Gain measures the expected reduction in entropy due to splitting on A

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

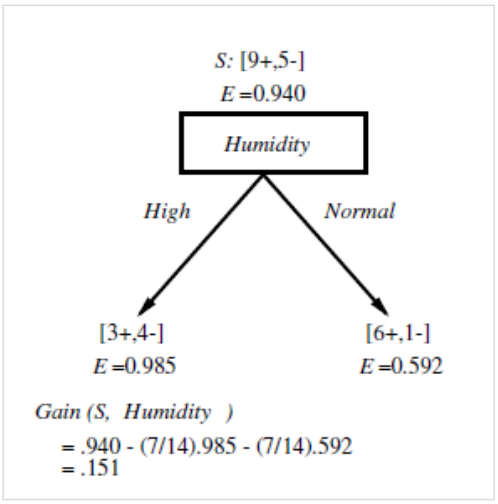
in the general case (*k-classification problem)*

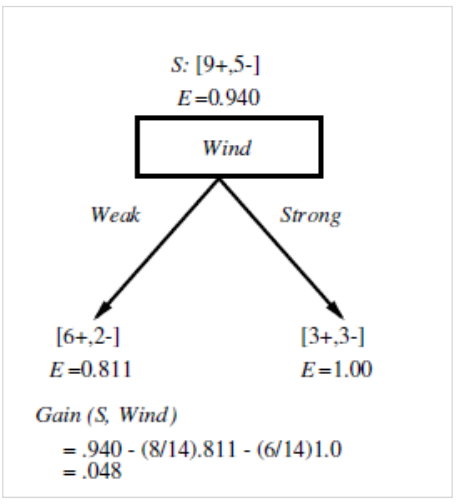
$$Entropy(S) = \sum_{i=1}^{k} - p_i \log_2(p_i)$$

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

(Batch) DTs: Information gain example

- Which attribute to choose next?





Hoeffding trees

- 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)
  - E.g., use the first few examples to choose the split at the root
- Problem: How many instances are necessary?
  - Hoeffding bound!
- Hoeffding tree variations
  - Very Fast DTs (VFDT), Domingos and Hulten, KDD 2000.
  - VFDTc, Gamma et al, KDD 2003
  - …

Hoeffding bound I

- Consider a real-valued random variable r whose range is R
  - e.g., for a probability the range is one,
  - for an 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 probability 1-δ the true mean of the variable μ_r will not differ by more than ε from the estimated mean after n independent observations, i.e., $P(\mu_r \geq \bar{r}-\varepsilon) = 1-\delta$, where:

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

- 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

Hoeffding bound II

- 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, the Hoeffding bound guarantees that with probability 1-δ , X_a is the correct choice for this node, if n instances have been seen so far in this node and $\overline{\Delta G} > \varepsilon$.
  - In this case, the sample size is enough to decide on X_a
- Otherwise, i.e., if $\overline{\Delta G} < \varepsilon$, the sample size is not enough for a stable decision.
  - With R and δ fixed, the only variable left to change ε is n
    - We need to extend the sample by seeing more instances, until ε becomes smaller than $\overline{\Delta G}$

Hoeffding tree induction algorithm

- **Input:** δ desired probability level.
- **Output:** $\mathcal{T}$ A decision Tree
- **Init:** $\mathcal{T} \leftarrow$ Empty Leaf (Root)
- While (TRUE)
  - Read next Example
  - Propagate Example through the Tree from the Root till a leaf
  - Update Sufficient Statistics at leaf
  - If $leaf\,(\#examples) > N_{min}$
    - 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.

Adaptive Size Hoeffding Tree (ASHT)

- Hoeffding tree
  - Leaf nodes are transformed into decision nodes by splitting on an attribute
  - Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute
    - Hoeffding bound

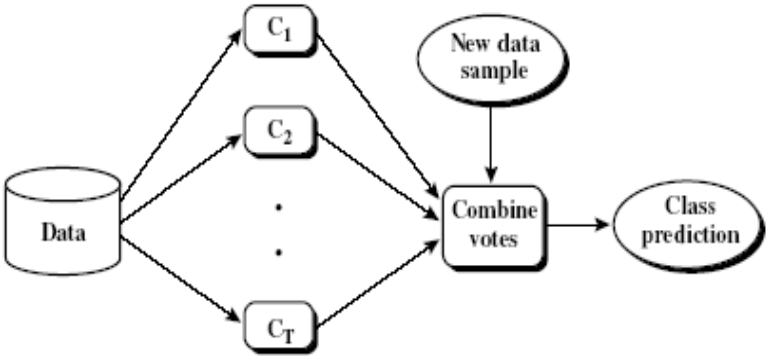
- Adaptive size Hoeffding tree (ASHT)
  - The tree has a maximum size (# of splitting nodes)
  - After one node splits, if the number of split nodes of the ASHT tree is higher than the maximum value, then it deletes some nodes to reduce its size

Data stream classifiers

- Decision trees
- Ensemble methods

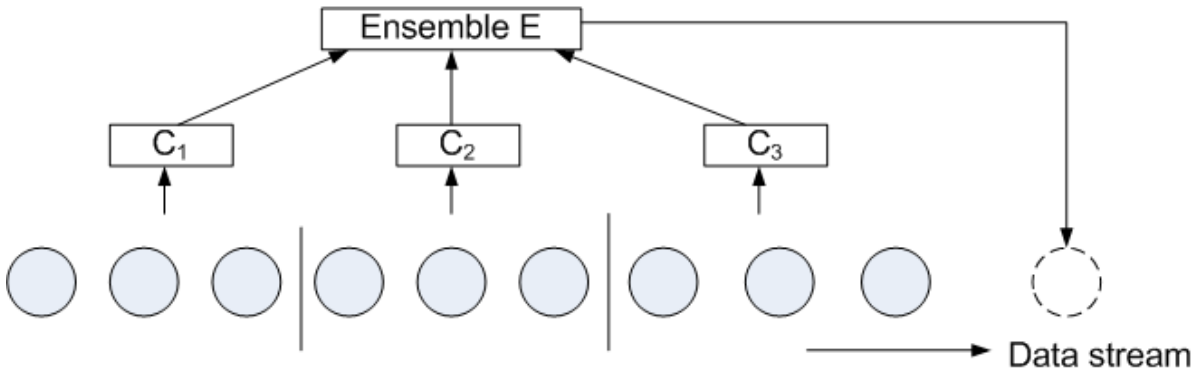
Ensemble of classifiers

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



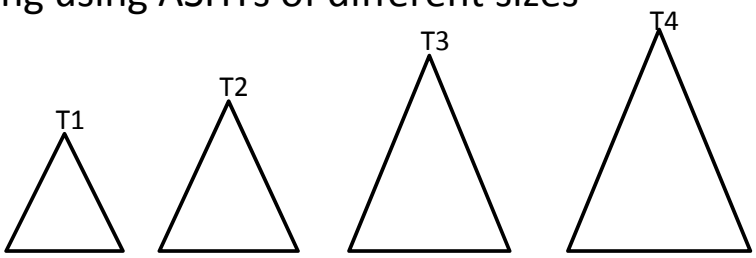
Ensemble Models in streams

- Idea: Use a **divide-and-conquer** manner to build models from continuous data streams



Ensemble of Adaptive Size Hoeffding Trees

- 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 n^{th} 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

(batch) Classifier evaluation

- The quality of a classifier is evaluated over a *test set*, different from the training set
- For each instance in the test set, we know its true class label
- Compare the predicted class (by some classifier) with the true class of the test instances
- Terminology
  - Positive tuples: tuples of the main class of interest
  - Negative tuples: all other tuples
- A useful tool for analyzing how well a classifier performs is the *confusion matrix*
- 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'	

(batch) Classifier evaluation measures

	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/ 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	
buy_computer = no	412	2588	3000	
total	7366	2634	10000	95.42

- Error rate/ Missclassification rate: error_rate(M)=1-accuracy(M)

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

- More effective when the class distribution is relatively balanced
  - Check Lecture 4, KDD I for more evaluation measures also if classes are imbalanced!

(batch) Classifier evaluation methods

- 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, k = 10 usually)
  - Randomly partition the data into k mutually exclusive subsets D_1, …, D_k each approximately equal size
  - Training and testing is performed k times
    - At the i-th iteration, use D_i as test set and others as training set
  - Accuracy is the avg accuracy over all iterations
- Bootstrap: Samples the given training data uniformly with replacement
  - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Check Lecture 4, KDD I for more evaluation methods, their pros and cons!

Evaluation methods for streams

- Holdout evaluation
  - 2 separate datasets for training (~70% - 80% of the dataset) and testing (~20%-30% of the dataset)
  - Train model on training set
  - Test model on test set
    - Static test set!!!
- Prequential evaluation (Interleaved test-then-train)
  - One dataset for training and testing
  - Models are first tested then trained in each instance
    - Test set is dynamic!!!

Evaluation measures for streams

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

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

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

- Both measures are computed based on the most recent samples through some
  - sliding window
  - fading function

Resources

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