Predictive Learning on Data Streams

Haixun Wang[1] and Ying Yang[2]
[1]IBM T.J. Watson Research Center
[2]Monash University

Please find the latest version of this tutorial at:

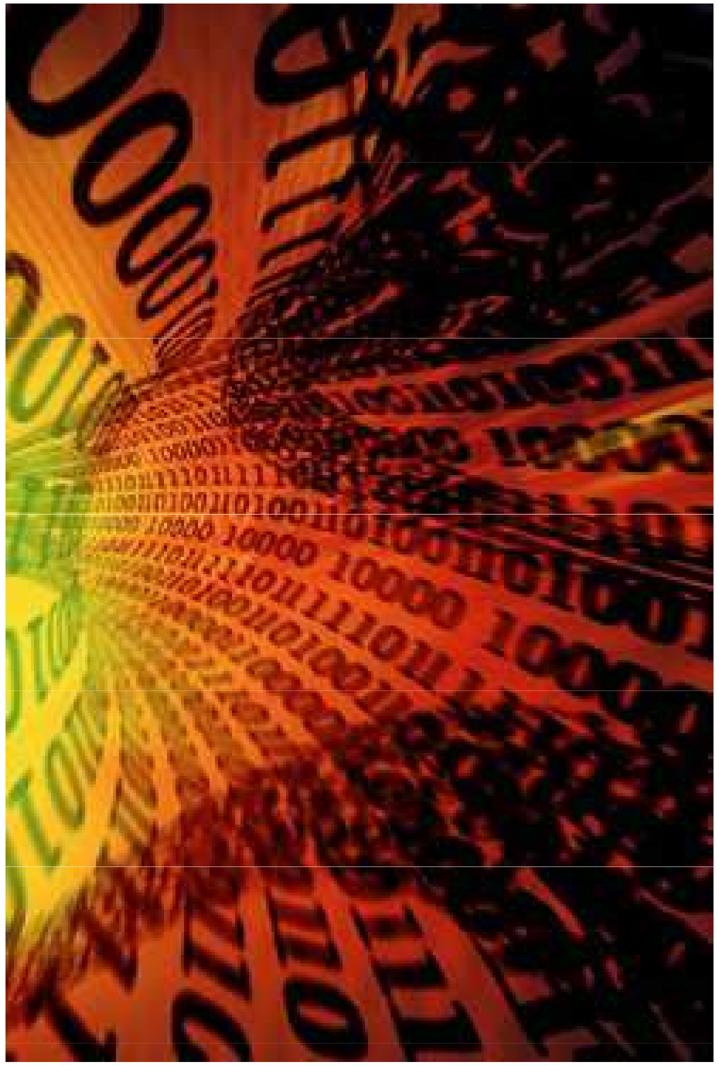
http://wis.cs.ucla.edu/~hxwang/tutorial.ppt

Outline

- Challenges
  - Volume, Speed, Time varying distribution
- Part I: Learning models from streams
  - Accuracy, Efficiency
  - Intelligent model reusing
- Part II: Applying models on streams
  - Efficiency, Accuracy
  - Intelligent load shedding

Background

- **Many real-life data are data streams in nature:**
  - data from large number of embedded sensors
  - high-speed real-time financial and retailer data
  - large-volume network traffic data
- **Resources are limited**
  - **CPU cycles, bandwidth, and memory**
- **Resource allocation is a very important issue.**



Challenges in Classifying Streams

- Cost of learning a model
  - impossible to mine the entire data at one time
  - can only afford **constant memory** per data sample
- Concept Drifts
  - previously learned models are invalid
  - model updates can be costly
- Cost of applying a model
  - classify data without seeing it?

PART I

Learning Classifiers from Streams

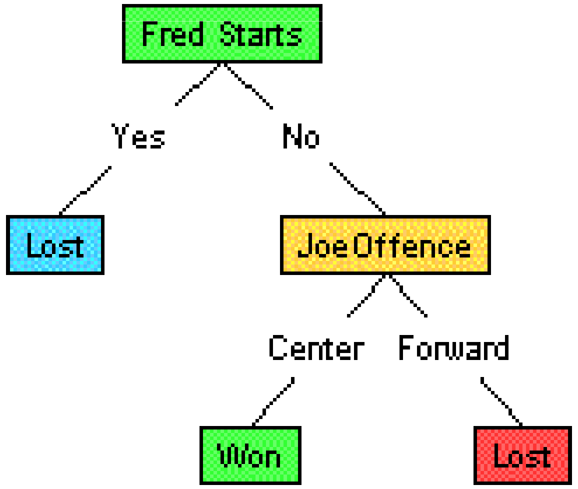
Issues: Accuracy, Cost, Model Reuse

Methods for Classifying Streams

- Decision Tree
- Hoeffding Trees
- VFDT and CVFDT
- Ensemble of Classifiers
- etc.

The Decision Tree Classifier

- Learning (Training) :
  - Input: a data set of (**a**, b), where **a** is a vector, b a class label
  - Output: a model (decision tree)
- Testing:
  - Input: a test sample (**x**, ?)
  - Output: a class prediction for **x**



The Decision Tree Classifier

- A divide-and-conquer approach
  - Simple algorithm, intuitive model
  - No ‘optimal’ model
- Compute information gain for data in each node
  - Super-linear complexity
- Typically a decision tree grows one level for each scan of data
  - Multiple scans are required
- The data structure is not ‘stable’
  - Subtle changes of data can cause global changes in the data structure

Idea of VFDT

- Task:
  - Given enough samples, can we build a tree in constant time that is ***nearly identical*** to the tree a batch learner (C4.5, Sprint, etc.) would build?
- Intuition:
  - With increasing # of samples, the # of possible decision trees becomes smaller
- Forget about concept drifts for now.

Hoeffding Bound

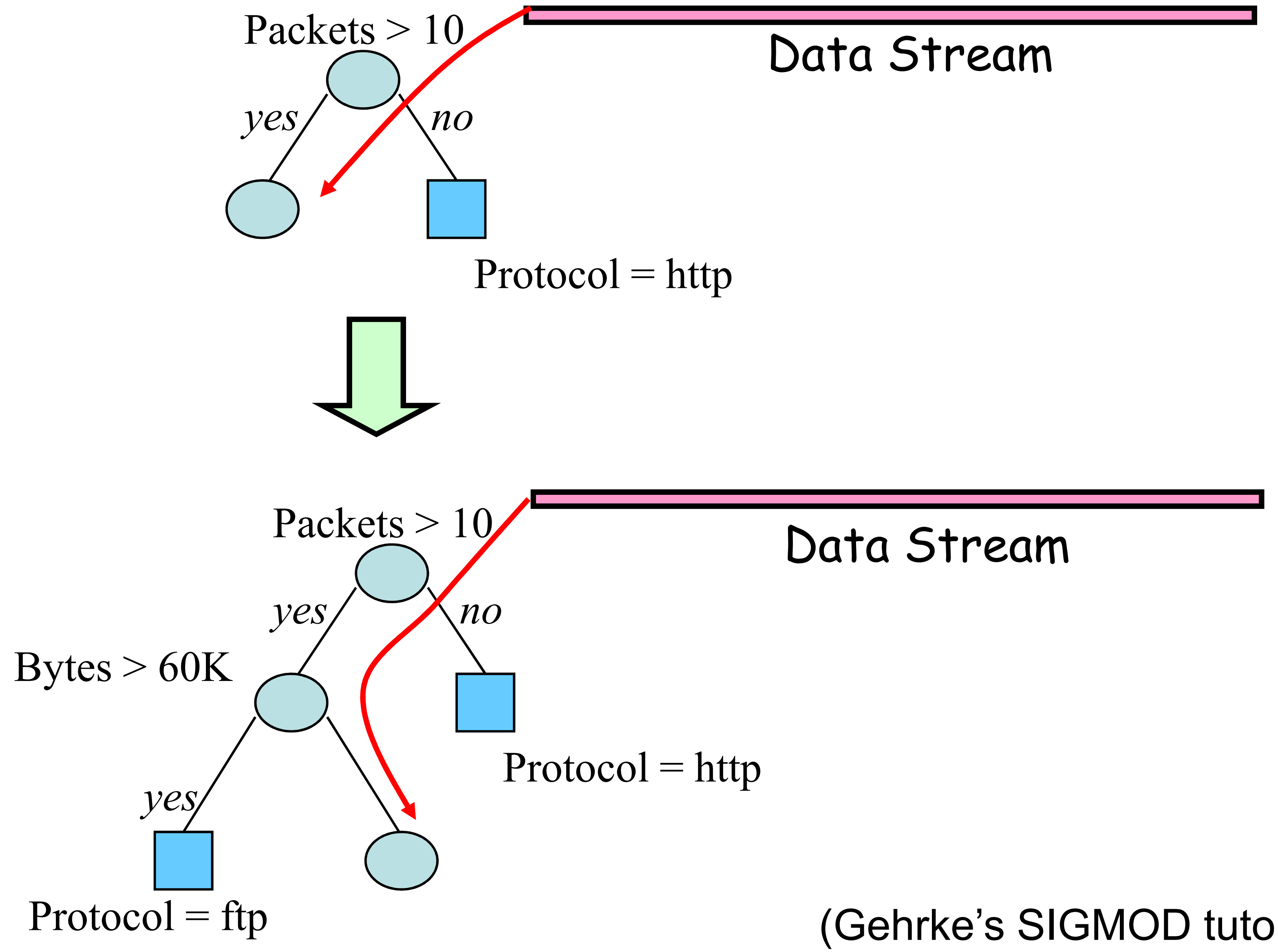
- Also known as additive Chernoff Bound
- Given
  - r : real valued random variable
  - n : # independent observations of r
  - R : range of r
- Mean of r is at least $r_{avg}-\varepsilon$, with probability $1-\delta$, or:
- $P(\mu_r \geq r_{avg} - \varepsilon) = 1-\delta$ and $\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$

Hoeffding Bound

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

- Properties:
  - Hoeffding bound is independent of data distribution
  - Error ε decreases when n (# of samples) increases
- At each node, we shall accumulate enough samples (n) before we make a split

Building a Hoeffding Tree



(Gehrke's SIGMOD tutorial)

Nearly Identical?

- Categorical attributes
  - with a high probability, the attribute we choose for split is the same attribute as would be chosen by a batch learner
  - identical decision tree
- Continuous attributes
  - discretize them into categorical ones

Building a Hoeffding Tree

- $G(X_i)$: the heuristic measure used to split a node (X_i is a discrete attribute)
- X_a, X_b : the attributes with the highest and second-highest G() after n examples
- $\Delta G = G(X_a) - G(X_b) \geq 0$

Building a Hoeffding Tree

- If $\Delta G > \varepsilon$, the Hoeffding bound states that :

$$P(\mu_{\Delta G} \geq \Delta G - \varepsilon > 0) = 1 - \delta$$

- $\mu_{\Delta G} > 0 \Rightarrow \mu_{G(X_a)} - \mu_{G(X_b)} > 0 \Rightarrow \mu_{G(X_a)} > \mu_{G(X_b)}$

- **Conclusion**: we have found a best attribute for split (X_a) with probability 1- δ

Hoeffding Tree: Pros and Cons

- Scales better than traditional DT algorithms
  - Incremental
  - Sub-linear with sampling
  - Small memory requirement
- Cons:
  - Only consider top 2 attributes
  - Tie breaking takes time
  - Grow a deep tree takes time
  - Discrete attribute only

VFDT

- Very Fast Decision Tree
  - Domingos, Hulten, 2000
- Various Improvement over Hoeffding Tree
  - Break near-ties more aggressively
  - G computed every n_{min} tuples (instead of for every tuple)
  - Deactivating unpromising leaf nodes
  - Dropping poor attributes
  - …
  - **Better time and memory performance**
- Still does not handle concept drifts

Concept Drifts

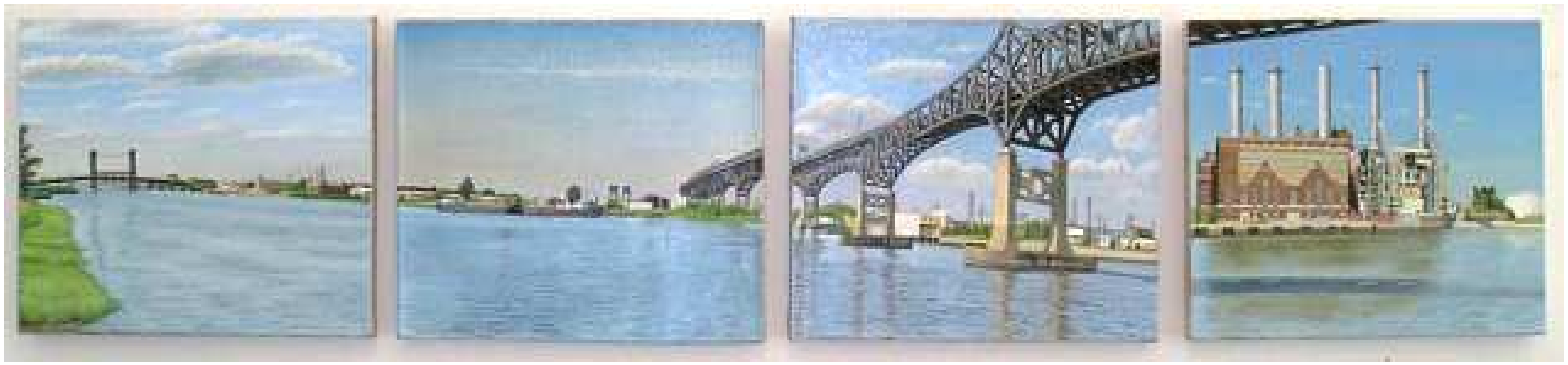
- Time-changing data streams
- Incorporate new samples and eliminate effect of old samples
- Naïve approach
  - Place a sliding window on the stream
  - Reapply C4.5 or VFDT whenever window moves
  - Time consuming!

CVFDT

- Concept-adapting VFDT
  - Hulten, Spencer, Domingos, 2001
- Goal
  - Classifying concept-drifting data streams
- Approach
  - Make use of Hoeffding bound
  - Incorporate “windowing”
  - Monitor changes of information gain for attributes.
  - If change reaches threshold, generate alternate subtree with new "best" attribute, but keep on background.
  - Replace if new subtree becomes more accurate.

Sliding Windows

- Mining data streams = Mining windows of static data ?
- Why we should be concerned whether <u>sample size</u> is large enough (even in the streaming environment).

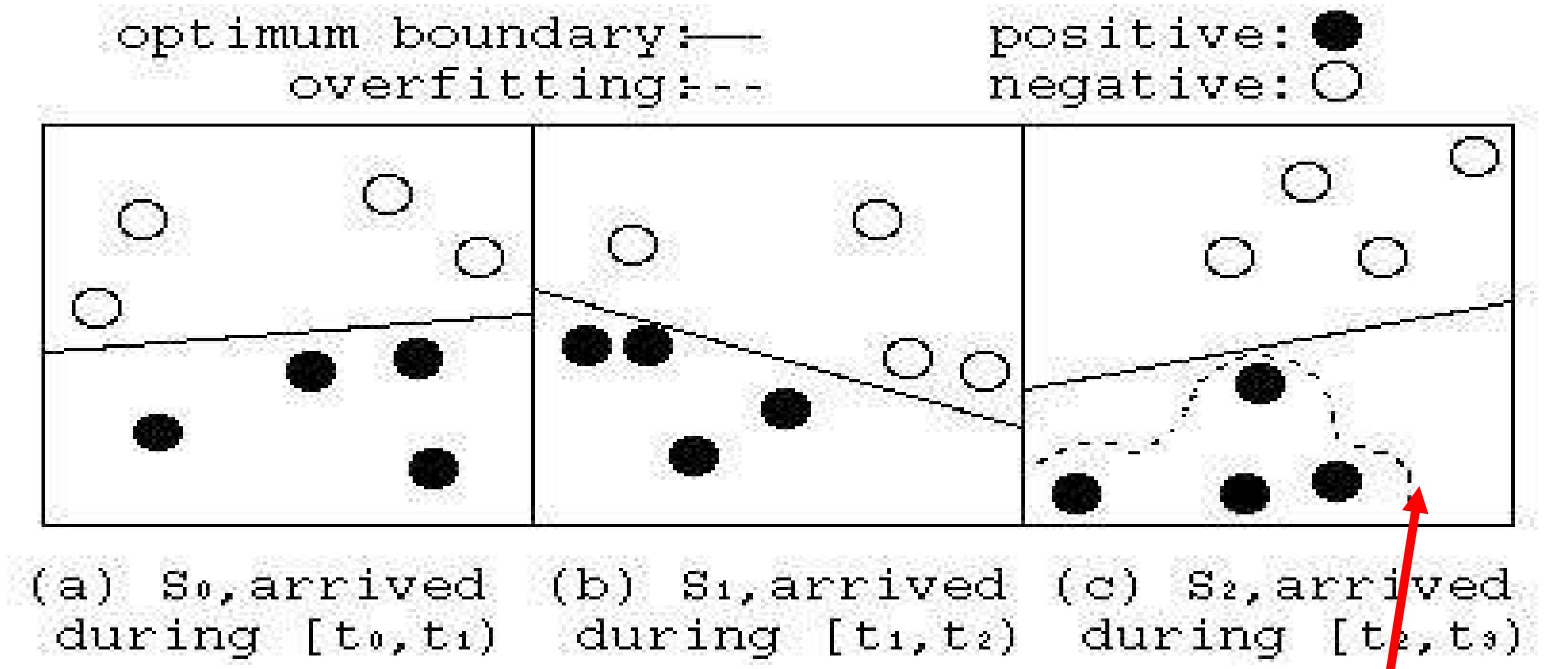


Robert Hendrickson
Hackensack River Panorama, oil on canvas

Pitfalls of Sliding Windows

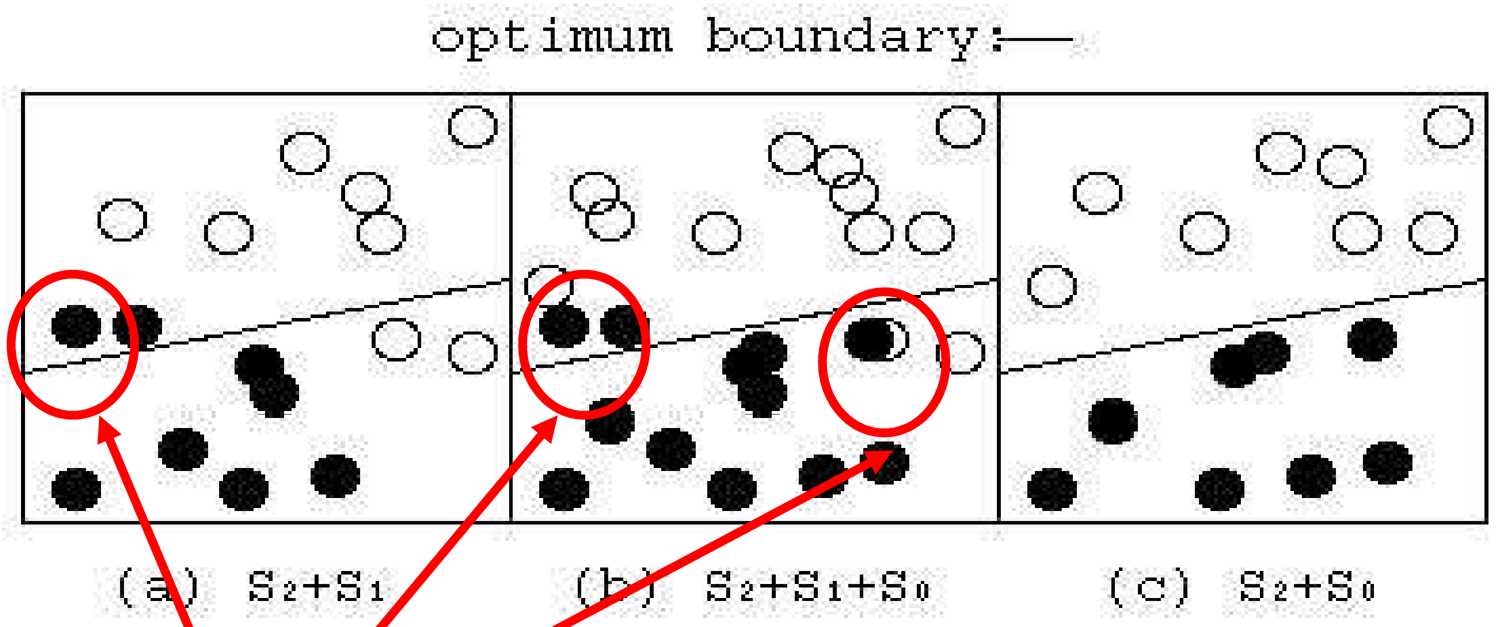
- Window-based incremental algorithm
  - incorporate new samples and eliminate effects of old samples
- One interpretation:
  - Old samples
    = samples outside the window
    = samples arrived *T* time units ago
  - How to decide *T*?

Data Distribution and Optimal Decision Boundaries


optimum boundary: —
overfitting: - -
positive: ●
negative: ○
(a) S_0, arrived during $[t_0, t_1)$
(b) S_1, arrived during $[t_1, t_2)$
(c) S_2, arrived during $[t_2, t_3)$

Overfitting!

Data Distribution and Optimal Decision Boundaries

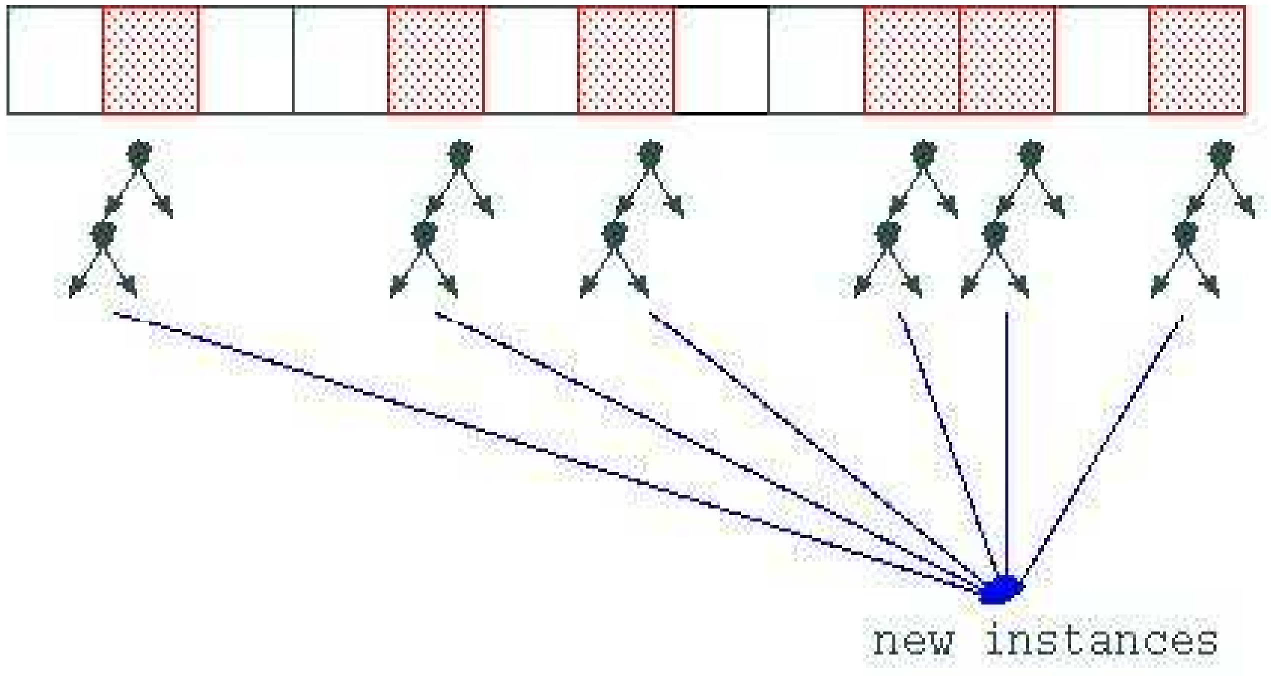


Conflicting Concepts!

Summary

- How to 'forget' old samples?
  - Discard instances after a fixed time period T
  - T is too large: conflicting concepts
  - T is too small: overfitting
- Other issues of a single model approach
  - Runtime performance
  - Ease of use
  - Parallelizability
  - …

Classifier Ensemble Method



▨ ▶ data chunks that have similar distribution as the new instances

Colorful Oil Painting Of A Musician Ensemble

Basic Idea

- Steam data is partitioned into sequential chunks
- Train a weighted classifier from each chunk
- The weight is based on the expected prediction accuracy on the current test examples
- Only top K classifiers are kept

Bias Variance Decomposition

- The expected added error of a classifier is expressed by:

$$Err = \frac{\delta_{\eta_c}^2}{s}$$

  - s is a constant independent of training model
  - $\delta_{\eta_c}^2$ denotes the variance

Accuracy Weighted Ensemble

$$\boxed{\begin{aligned} Err_i &= \frac{\delta^2_{\eta^i_c}}{s} \\ w_i &= \frac{c}{\delta^2_{\eta^i_c}} \end{aligned}}$$

Single Classifier

- Probability Output:

$$f_c^g(y) = p(c \mid y) + \eta_c^g(y)$$

- Assuming each partition is of the same size

$$\sigma^2_{\eta_c^g(y)} \geq \frac{1}{k^2} \sum_{i=n-k+1}^{n} \sigma^2_{\eta_c^i}$$

Ensemble Classifier

- Probability Output:

$$f_c^E(y) = p(c \mid y) + \eta_c^E(y)$$

- Naïve assumption: the variances of different classifiers are independent.

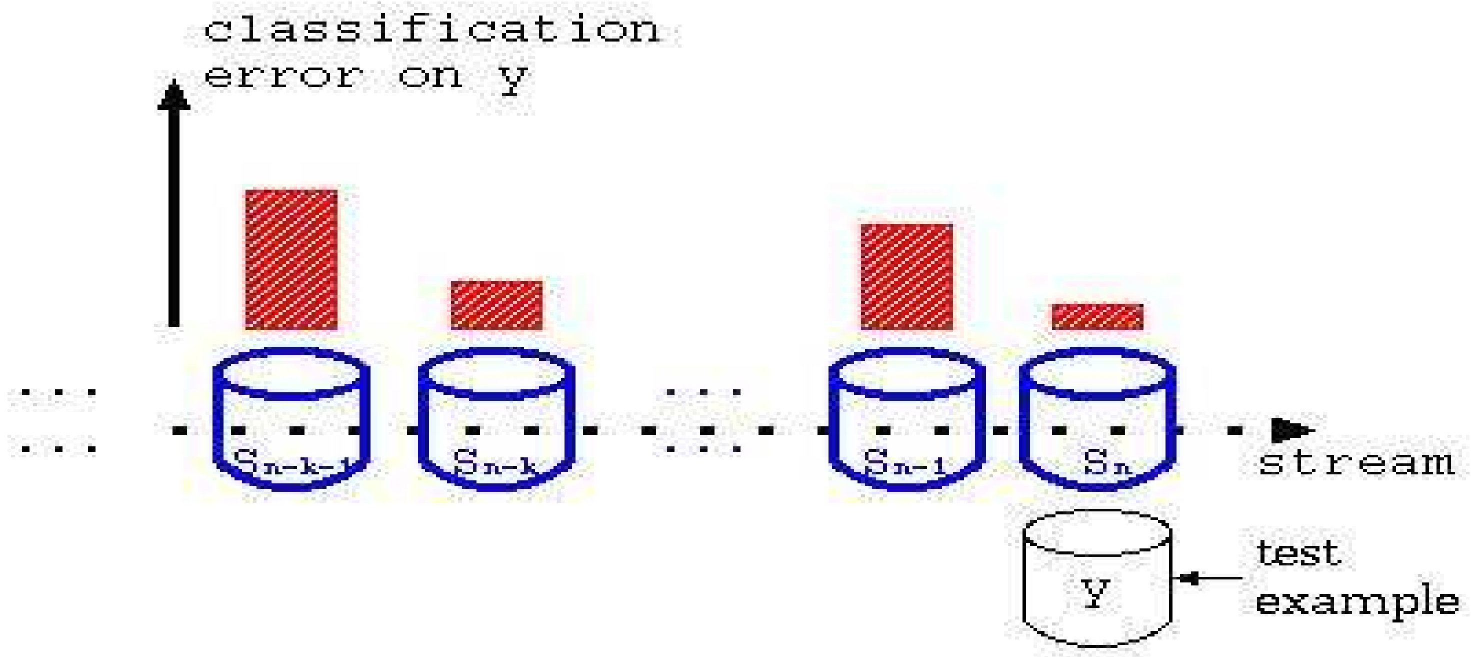
$$\sigma^2_{\eta_c^E(y)} = \sum_{i=n-k+1}^{n} w_i^2 \sigma^2_{\eta_c^i} \Bigg/ \left(\sum_{i=n-k+1}^{n} w_i \right)^2$$

- Conclusion: ensemble has smaller error

But in reality …

- We do not know
  - Error or variance or the function being learned
- Solution:
  - Use estimation!
  - Apply the i-th classifier on the current training data to estimate the error of the classifier

Weight Estimation



Problem I: Learning cost is still high

- Tree construction has super-linear complexity
  - Much effort has been made to build a decision tree faster (CLOUD, Rainforest, etc.)
- Approximate methods: Hoeffding bound
  - We still need to evaluate G()
  - Incremental updating a non-stable structure
- Learn/update a model is costly

Do we care about the shape of the tree?

- The Hoeffding method tries to build decision trees “**nearly identical**” to trees that a batch classifier (ID3, C4.5, Sprint) would build
- But batch learner builds decision trees in a divide-and-conquer greedy manner (not optimal anyway)

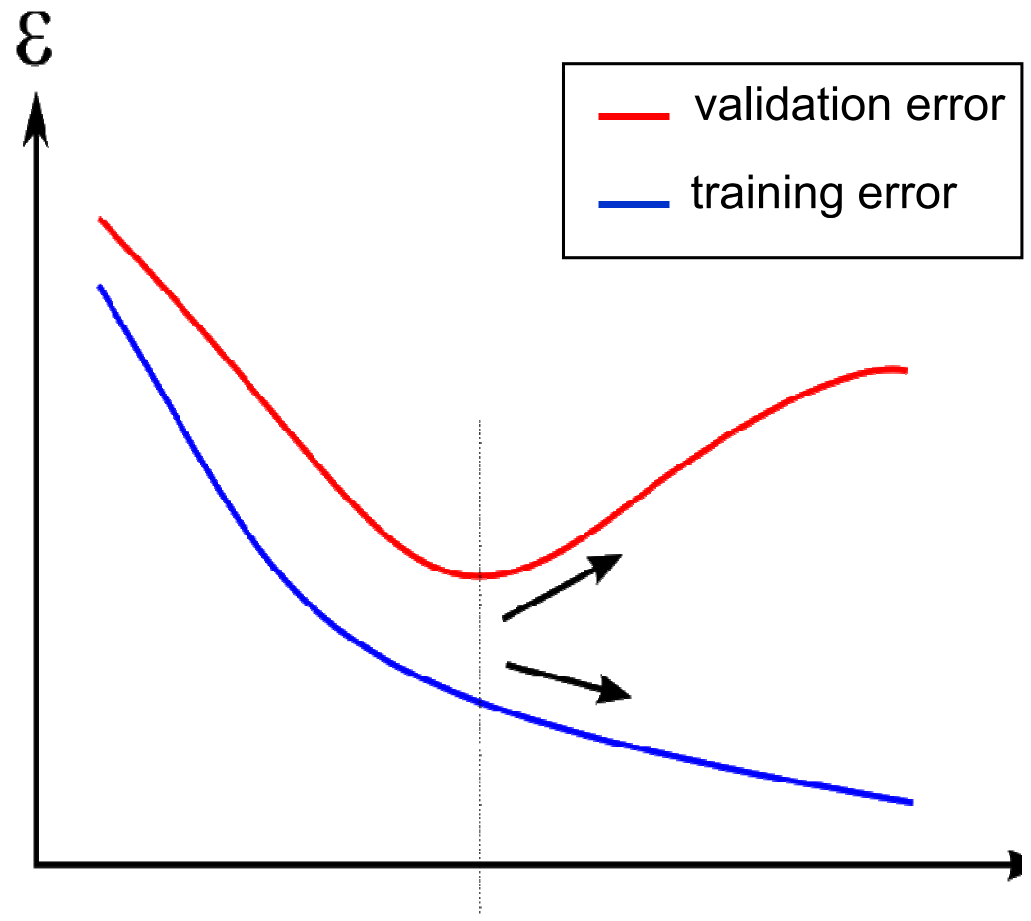
How about using Random Trees?

- Build an ensemble of N random trees
- At each node, randomly choose an attribute to split
- Throw samples into each tree
- Use class label distribution in the leaf nodes as probability output
- Constant cost of tree construction
- Maximize structural diversity in N trees

Problem II: dependence on the last window

- **Incrementally maintain a single classifier**
  - **tracking patterns in the recent window**
  - **the recent window contains the sole training data**
- **Maintain a set of classifiers (an ensemble)**
  - **each trained from historical data**
  - **Validate classifiers by the most recent data to find "good" classifiers.**
- **There is a flaw:**
  - too much dependence on the most recent data

Fundamental challenge: Overfitting



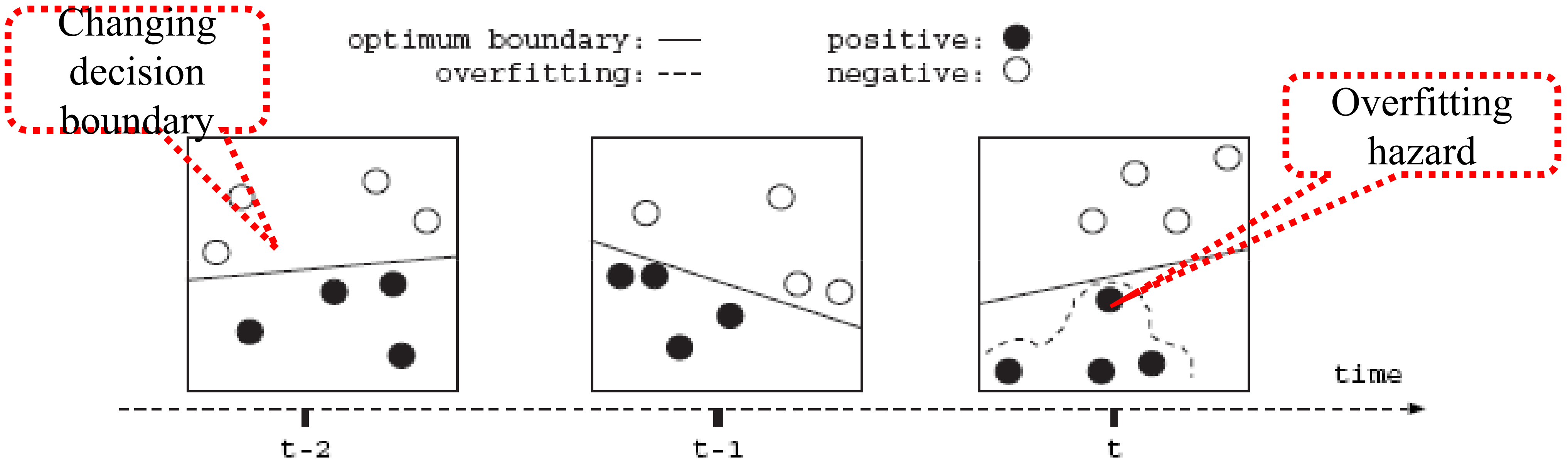
Overfitting in supervised learning (e.g. decision tree, neural network).

- Overfitting**: models are too specific, or too sensitive to the particulars of the training dataset.**
- Symptom: **the validation error increases while the training error steadily decreases.**
- Cause: **the training dataset is not large enough, or not representative enough**

Overfitting is prevalent in the streaming environment

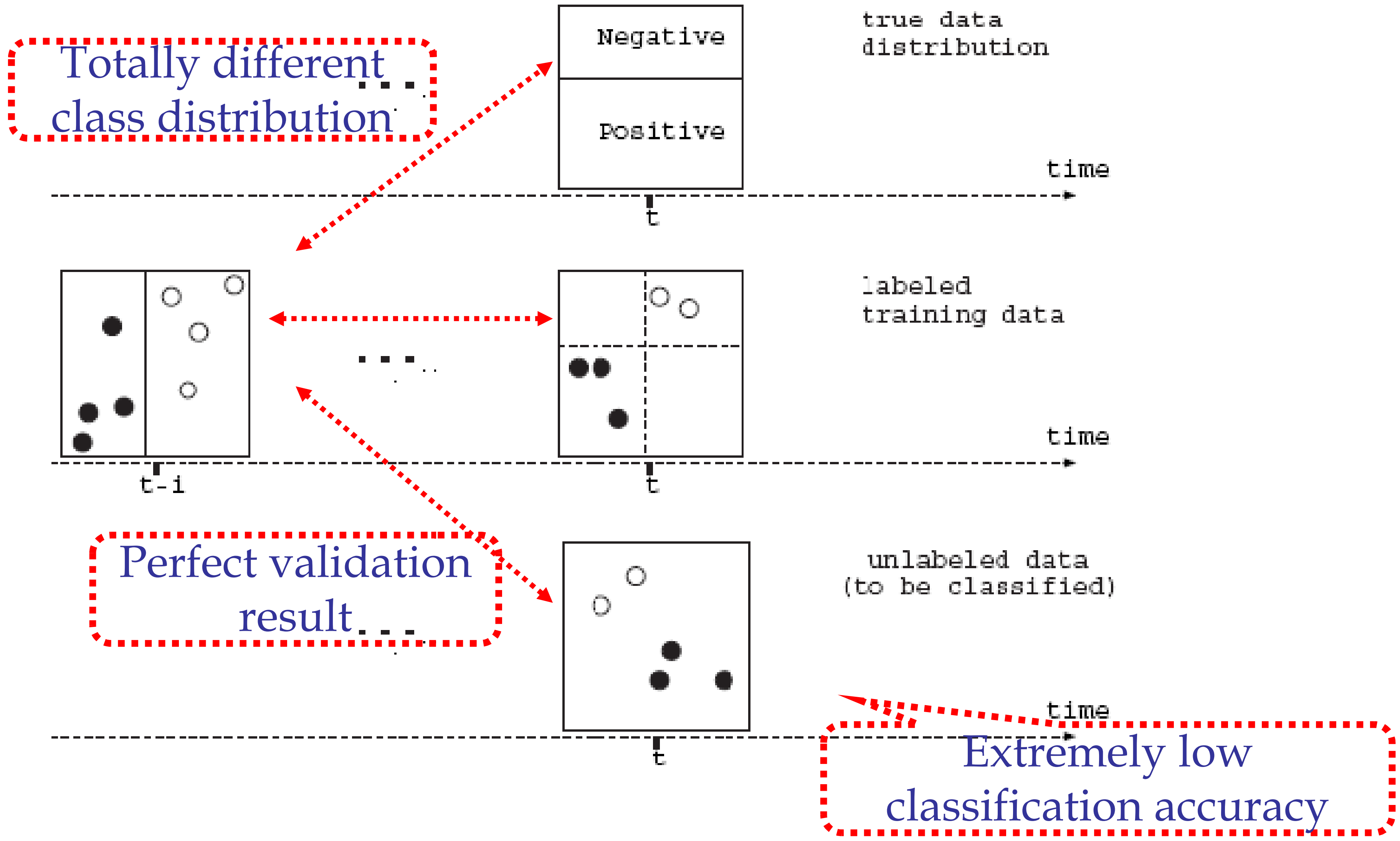
- **Inadequacy of training data.**
  - We must avoid conflicting concepts
  - Small windows are used
- **Unrepresentative training data.**
  - Stream data is bursty.
  - Samples may concentrate in a small region of the sample space.

Inadequacy of training data – single classifier



- Traditional solutions do not work in the streaming environment.
- Change detection
  - differences in data distributions ≠ concept drifts.
  - computational intensive
- Difficult to find the optimal rate to expire the old data.

Unrepresentative training data – ensemble classifier


Totally different class distribution
Negative
Positive
true data distribution
time
t
labeled training data
. . .
t-i
Perfect validation result
unlabeled data (to be classified)
Extremely low classification accuracy

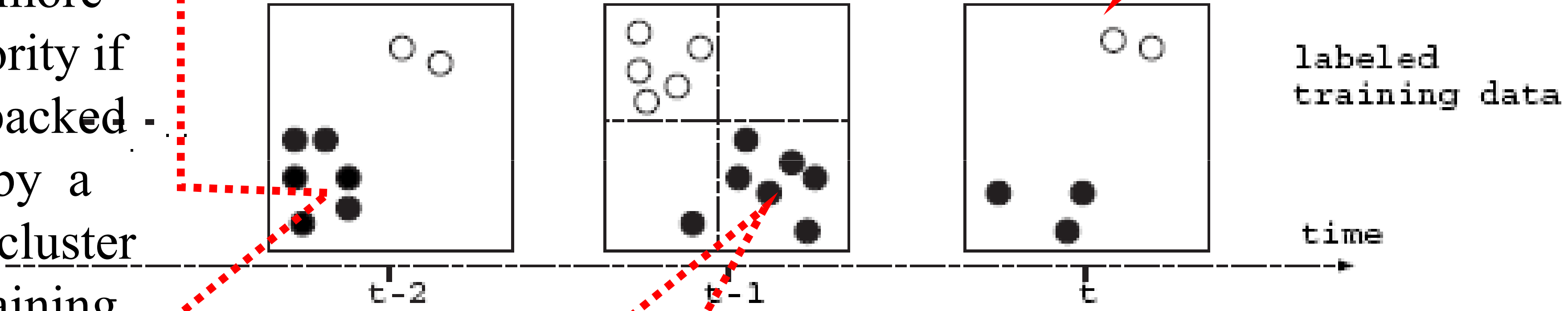
How to suppress overfitting in data streams?

- For static data, only sample size matters.
  - enlarging the training dataset to include more instances will help reducing overfitting.
- For stream data, we must consider 3 factors:
  - Size
  - Space
  - Time

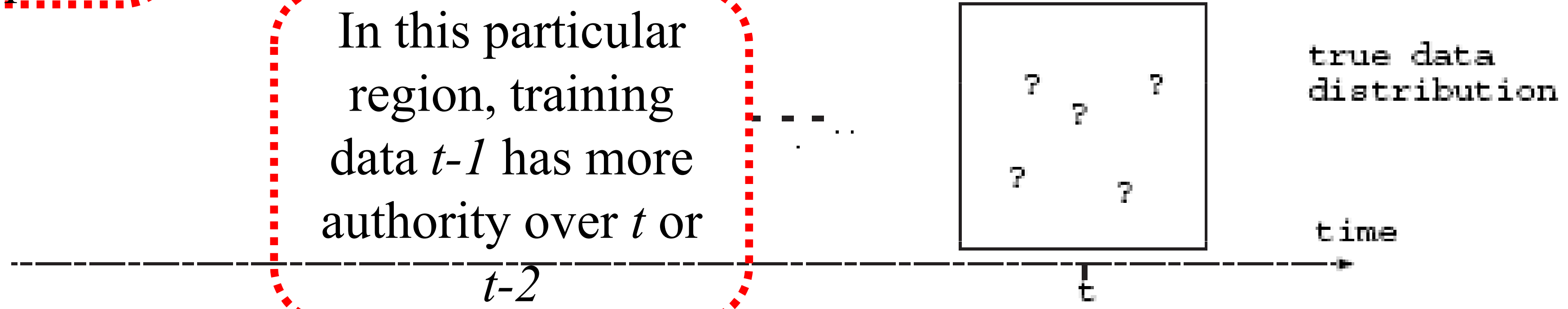
Time, Space, Size

Training data of time *t* has more authority than *t-1*, *t-2*, …

Classifier has more authority if it is backed up by a large cluster of training samples



In this particular region, training data *t-1* has more authority over *t* or *t-2*



What is the most likely real class distribution at time *t*?

Summary

- To find the *most likely* current class distribution, consider the time, the space, and the size factors in combining historical classifiers.
- How to quantify a historical classifier by time, space, and size?
- We formulate the problem from an optimization viewpoint.

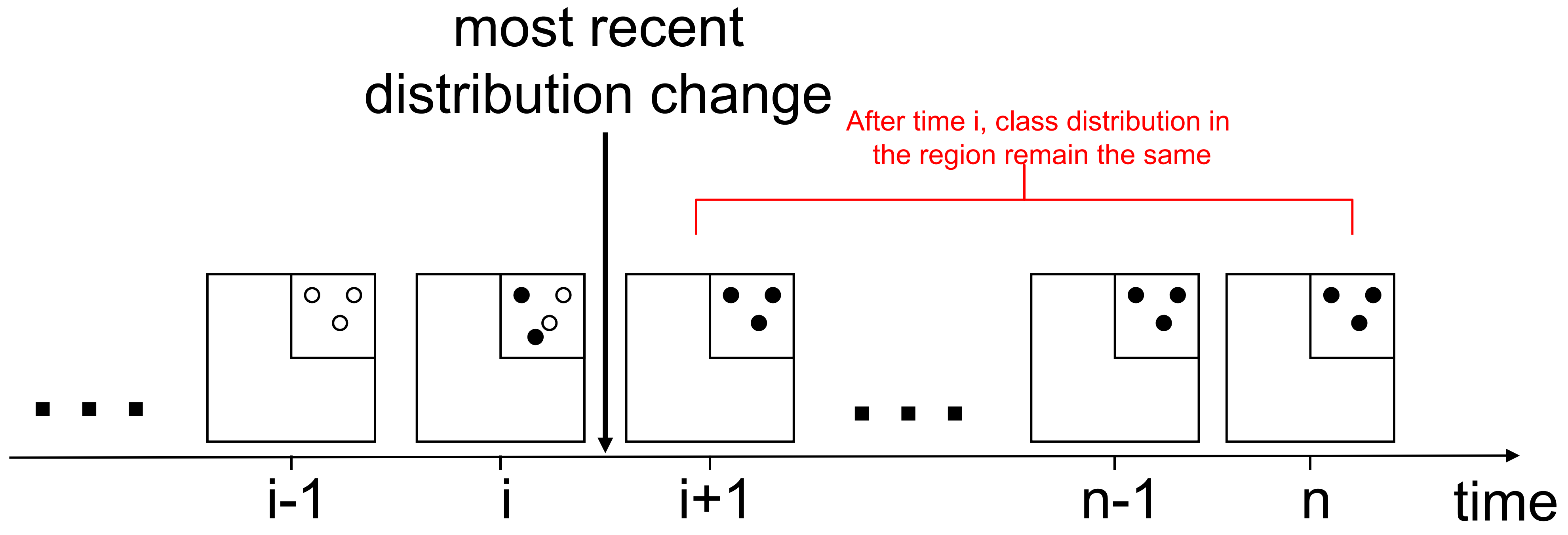
Modeling

- Distribution changes can occur abruptly
  - Previous work models smooth, continuous changes
- At any given time, the underlying data generating mechanism is in one of multiple states
  - Within each state, class distribution is stable
- State changes are modeled by a Markov model
  - The aggregated ingress rate is λ

Modeling

- We partition the stream into a sequence of time windows
  - Time-based windows make more sense than count-based windows for concept drifts
- We partition the feature space *V* into a set of non-overlapping regions.
  - Bursty arrivals may concentrate in one region.
- Weight classifiers by time and region.

Modeling


most recent
distribution change
After time i, class distribution in the region remain the same
i-1
i
i+1
n-1
n
time

The probability that the most recent state transition occurs between time window *i* and *i+1* is:

$$P(C_i) = e^{-\lambda(n-i)t} - e^{-\lambda(n-i+1)t}$$

Modeling

- f_i: observed positive class distribution
- x: *posterior* positive class distribution
- N_i : total # of instances we have observed
- The probability that we observe $N_i f_i$ positive instances out of the N_i total instances is

$$Y_i(x) = \binom{N_i}{N_i f_i} x^{N_i f_i} (1 - x)^{(N_i - N_i f_i)}$$

Modeling

- q_j : the event that a randomly drawn instance is positive
- $P(q_j)$: the positive class distribution
- $P(q_j|C_i)$: the positive class distribution given C_i (last concept drifts occurs at time i)
- the probability we observe what we have observed in all historical windows up to W_n is:

$$\begin{aligned} L_i &= \prod_{j=-\infty}^{n} Y_j(P(q_j|C_i)) \\ &= \prod_{j=-\infty}^{i} Y_j(P(q_j|C_i)) \times \prod_{j=i+1}^{n} Y_j(P(q_n|C_i)) \end{aligned}$$

Modeling

- We conclude that when

$$P(q_n|C_i) = \frac{\sum_{j=i+1}^{n} N_j f_j}{\sum_{j=i+1}^{n} N_j}$$

  L_i is maximized.

- **In other words, given the observations in each window W_i and the assumption that the most recent concept drift occurs between time i and i+1, above is the most likely current class distribution.**

Conclusion

- Given our observation of past windows, the most likely current class distribution is:

$$P(q_n) = \sum_i \left(\frac{\sum_{j=i}^{n} N_j f_j}{\sum_{j=i}^{n} N_j}\left(e^{-\lambda(n-i)t} - e^{-\lambda(n-i+1)t}\right)\right)$$

- Weight by time (exponentially), and size (linearly) in each region.

Summary

- To handle concept drifts
  - we keep on learning new models
  - we keep throwing away old models
  - the data we rely on is a tiny portion of the infinite stream

But history always repeats itself

- Assumption: the data generation mechanism works in different states
- In each state, the class distribution is stable
- State transition can occur anytime
- The total number of states are limited

Tasks

1. Learn from historical models, identify distinct concepts
2. Learn transition patterns among concepts
3. Predict next concept
4. Use models that correspond to the next concept to classify incoming data

RePro [KDD'05]

- Trigger
  - Using a current model, and monitor whether accumulated error exceeds a threshold
- Finding a concept
  - If error > threshold, learn a new concept c
  - Compare c with historical concepts x_i
  - If similarity between c and any x_i is less than a threshold, add c into historical concepts
- Transition matrix
  - A first order Markov chain of concepts

RePro: Concept Similarity

- Let x and y be two concepts.
- Let D be a dataset
- The similarity of x and y is proportional to the number of same predictions they make for items in D.

Unsolved Issue #1

- Concept similarity does not have transitivity.
  - A is similar to B and B is similar to C does not mean A is similar to C
  - Without pair-wise comparison, we cannot find concepts when change occurs slowly
  - Online pair-wise comparison is too time consuming to be practical for the stream environment

Unsolved Issue #2

- Concept similarity is hard to measure.
  - RePro relies on a dataset D to measure the similarity of two concepts
  - D’s data distribution and class distribution has significant impact on the similarity measure

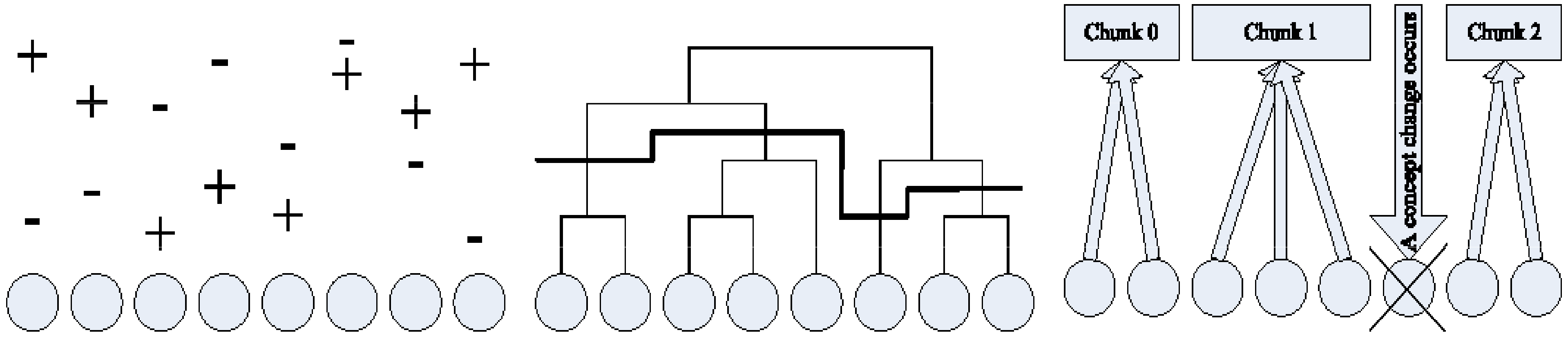
Unsolved Issue #3

- RePro is heavily dependent on parameters
  - The stable threshold
  - The trigger threshold
  - The concept similarity threshold

Cluster-based Model

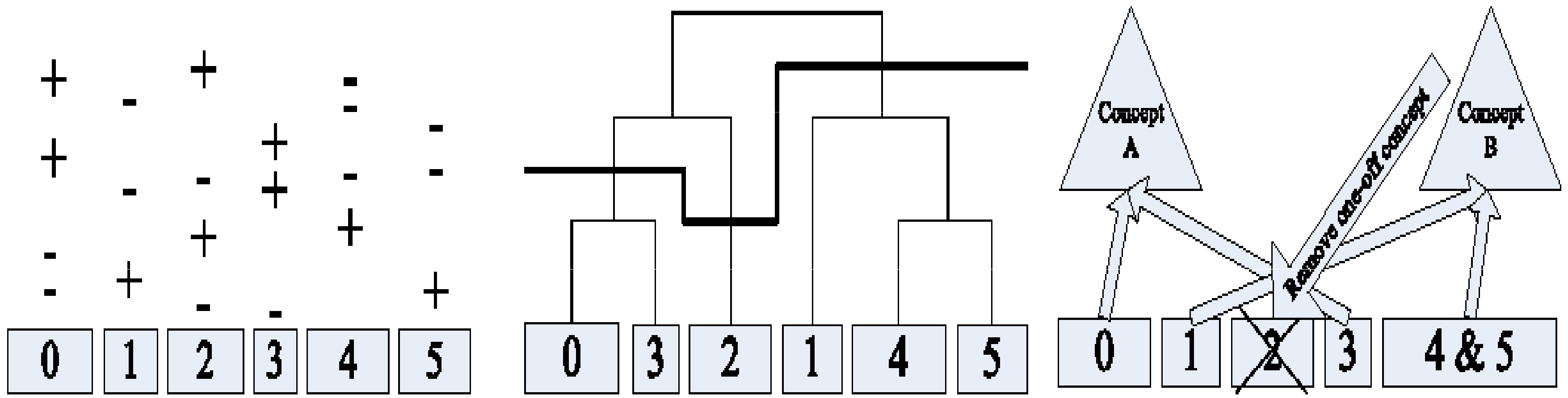
- Phase 1:
  - Build concept models offline
  - Using clustering to discover all underlying concepts
- Phase 2:
  - For incoming data, find the probability it belongs to each concept
  - Make weighted prediction

Clustering by concept



(a) Blocks of Data (b) Dendrogram and the Best Cut (c) Partition into Chunks

Clustering by concept



(a) Chunks of Data (b) Dendrogram and the Best Cut (c) Underlying Concepts

PART II

Applying Classifiers on Streams

Issues: Accuracy, Cost (CPU, bandwidth, …), Load Shedding

Background: Load Shedding

- Load shedding — **dropping *certain* data when the system is overloaded.**
- Current studies on load shedding focus on answering traditional (aggregation) queries, not on *mining* data streams.
- Cost is a major concern in classifying data streams
  - E.g. a large number of sensors simultaneously send data to central server for real time analysis
  - Central server cannot afford enough bandwidth and CPU resource to handle them

Optimization Problem

- **Resource allocation as optimization problems:**
  - minimizes resource usage subjective to a lower bound on precision
  - maximizes the precision subjective to an upper bound on resource

Possible Solutions

- Randomly shedding load
  - degradation of classification quality
- Using user-provided QoS metrics to shed load
  - QoS metric is often unavailable in dynamically changing stream environment

Load shedding makes a difference

(A motivating example)

- Two cameras are placed on two different spots on the highway
- Each camera takes a picture at each time unit and sends it to the central server
- The server can only process one of the two pictures at each time unit
- Goal: catch as many speeding cars as possible in real-time.

Load shedding makes a difference
(A motivating example)

- Assumptions:
  - At any time, highway A contains a speeding car with probability p_A; highway B contains a speeding car with probability p_B
  - p_A and p_B change with time
  - The classifier is always accurate

Load shedding makes a difference

(A motivating example)

- Scheme 1 (Naïve): at each time unit, choose A or B equally likely:

$$E_1 = \frac{p_A + p_B}{2}$$

- Scheme 2: depending on the most recent history of pictures from the two directions: if one contains a speeding car and the other does not, give it higher chance (*q*>0.5) to be chosen, otherwise, equally likely:

$$E_2 \geq E_1$$

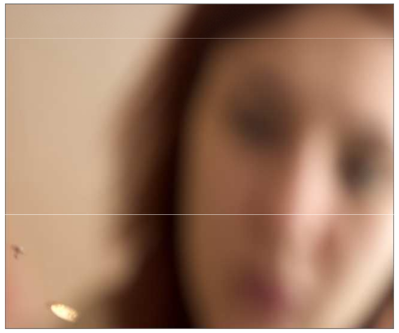
Load shedding makes a difference

(A motivating example)

- Scheme 2 automatically focuses on data with higher benefits;
- Scheme 2 does not depends on the parameters p_A and p_B, so if they drift with time, the scheme can adapt to the new parameters automatically.

Resource may not make a difference
(Another motivating example)

- You are to identify a person in a picture
- The picture quality is not good, and you haven't seen the real person for a long time, so you need time for a good look at the picture.
- But if you don't know the person to begin with, or if the person in the picture is beyond recognition, then everything is wasted …



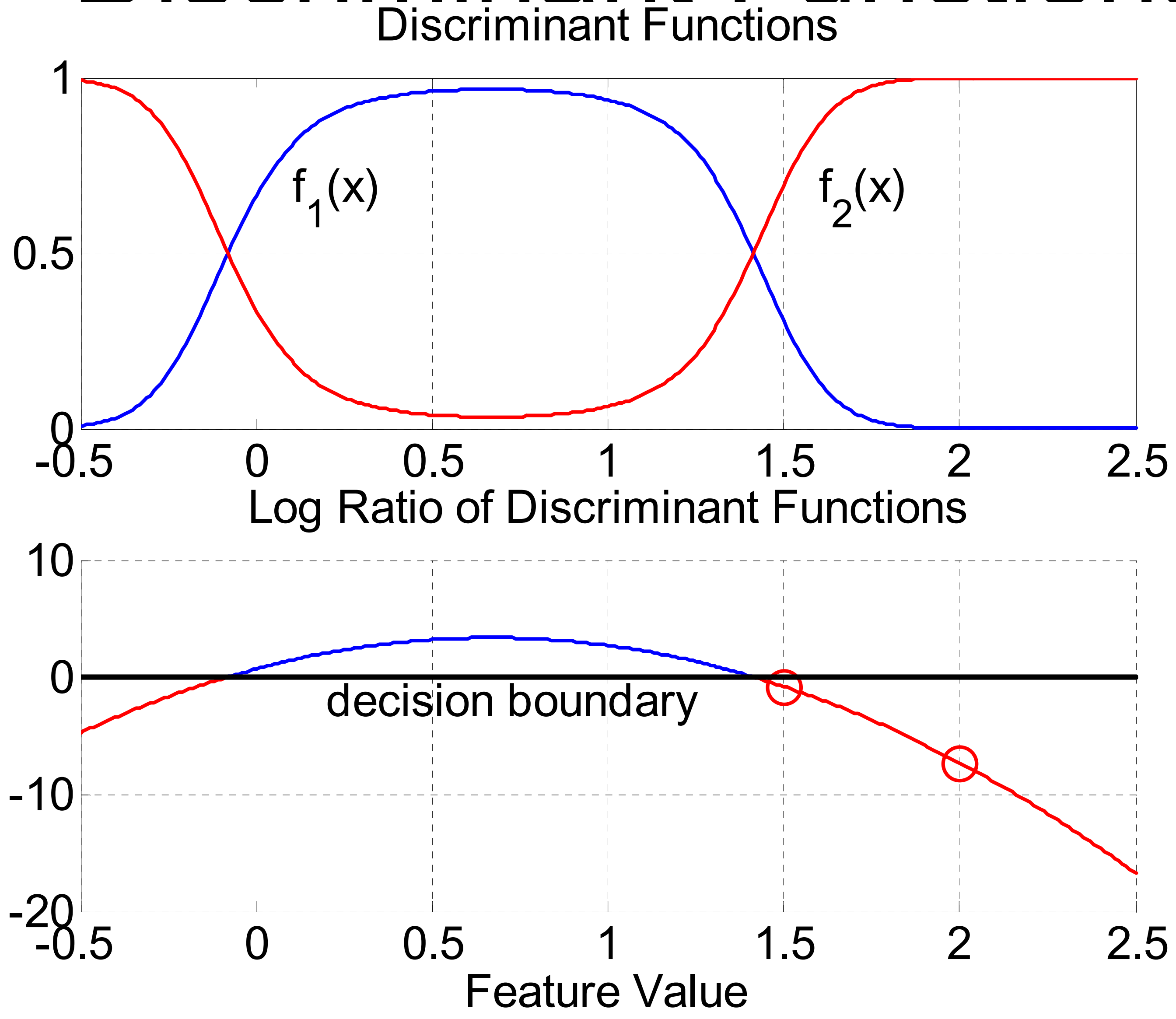
Summary

- We are concerned with the loss of classification quality due to lack of data observation, not the inherent “incompetence” of a classifier.
- Observations:
  – For certain instances, no matter how much time/bandwidth/observations you use, you cannot improve the quality of classification.
  – For certain instances, quality of classification can be improved greatly if more time/bandwidth/observations is used on these instances.
- Problem:
  – What are the instances that observations make a difference?
  – What kind of observations to make for those instances?

LoadStar [SDM 2004]

- *A Quality of Decision* (QoD) measure
  - If QoD is high, then we don’t want to spend more time on the input: the current classification is already good.
  - If QoD is low, then we should give the classifier more resource so that it can make a better decision
- QoD is based on the *prediction of feature value* at the next time unit

Quality of Decision— Discriminant Functions



Quality of Decision Based on Log Ratio

- Assume that we can derive a distribution of the feature value in the next time unit:

$$\vec{X} \sim p(\vec{x})$$

- We can compute the expected value of a discriminant function:

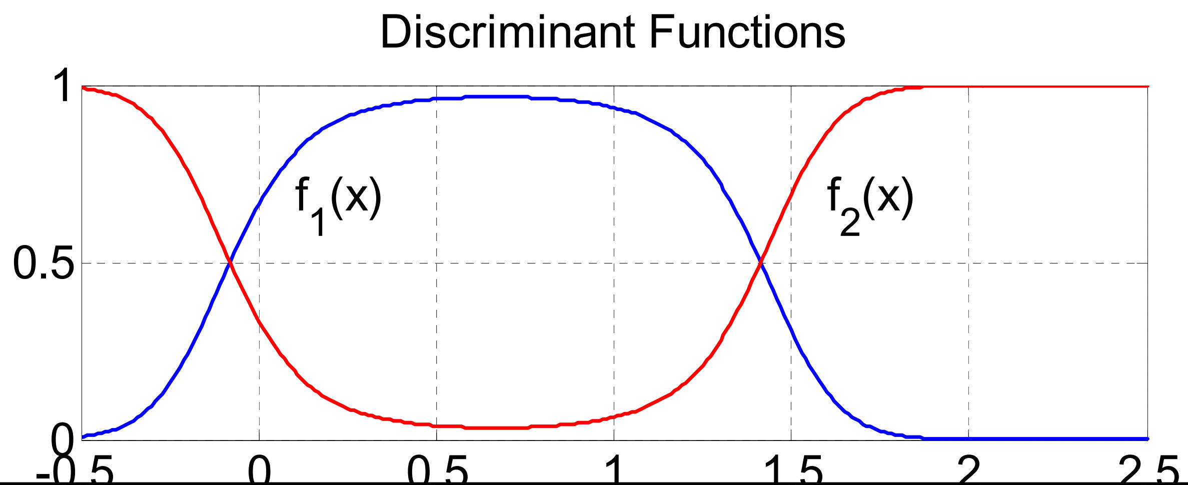
$$E_{\vec{X}}[\log f_i(\vec{x})] = \int_{\vec{x}} [\log f_i(\vec{x})] p(\vec{x}) d\vec{x}$$

- The decision: $\delta_1 : \quad k = \arg\max_i E_{\vec{X}}[\log f_i(\vec{x})]$
- The Quality of Decision (QoD):

$$Q_1 = E_{\vec{X}}\left[\log\left(\frac{f_k(\vec{x})}{f_{\tilde{k}}(\vec{x})}\right)\right] = E_{\vec{X}}[\log f_k(\vec{x})] - E_{\vec{X}}[\log f_{\tilde{k}}(\vec{x})]$$

Quality of Decision Based on Log Ratio

- $Q_1 \geq 0$, the higher the Q_1, the more confident we are.
- We hope more resources can improve the confidence of tasks with low Q_1
- Problem: what if best decision and the 2nd best decision are equally bad?



Quality of Decision Based on Overall Risk

- At a point *x* in the feature space, if we decide the class is c_i, then the conditional risk of our decision is:

$$R(c_i \mid \vec{x}) = \sum_{j=1}^{K} \sigma(c_i \mid c_j) P(c_j \mid \vec{x})$$

- We assume zero-one loss:

$$\sigma(c_i \mid c_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

- The expected risk is:

$$E_{\vec{x}}\left[R(c_i \mid \vec{x})\right] = \int_{\vec{x}} R(c_i \mid \vec{x})\, p(\vec{x})\, d\vec{x}$$

Quality of Decision Based on Overall Risk

- The decision based on the expected risk:

$$\delta_2: \quad k = \operatorname{argmin}_i E_{\vec{X}}[R(c_i \mid \vec{x})]$$

- The Bayesian risk:

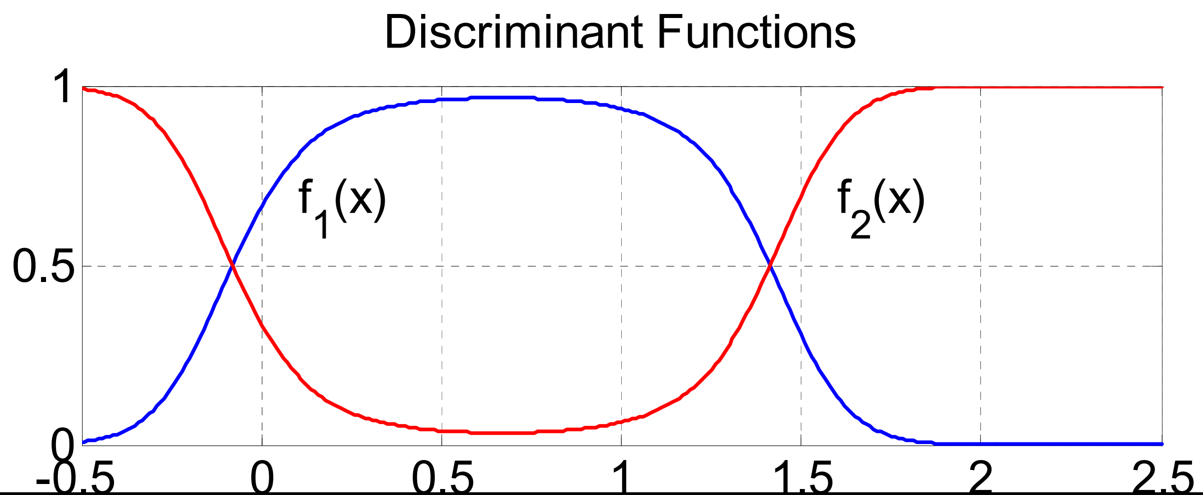
$$E_{\vec{x}}\left[R(c^* \mid \vec{x})\right] = \int_{\vec{x}} R(c^* \mid \vec{x}) p(\vec{x}) d\vec{x}$$

- The Quality of Decision (QoD) is defined as the difference of expected risk:

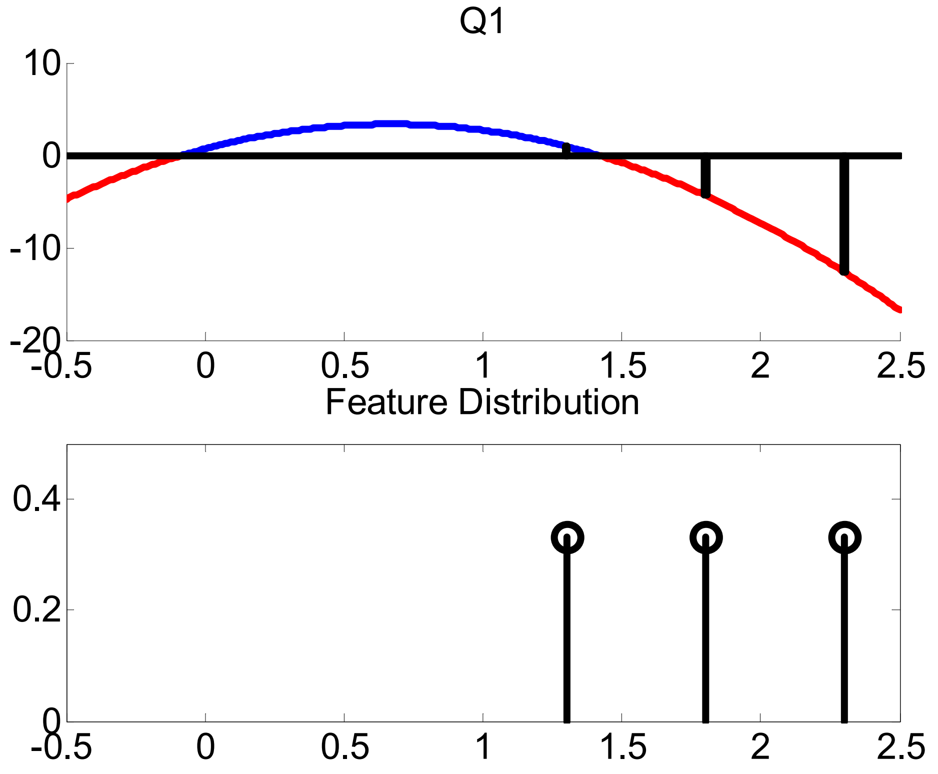
$$\begin{aligned} Q_2 &= 1 - \left(E_{\vec{x}}[R(c_k \mid \vec{x})] - E_{\vec{x}}\left[R(c^* \mid \vec{x})\right]\right) \\ &= 1 - \int_{\vec{x}} \left[P(c^* \mid \vec{x}) - P(c_k \mid \vec{x})\right] p(\vec{x}) d\vec{x} \end{aligned}$$

Quality of Decision—Based on Overall Risk

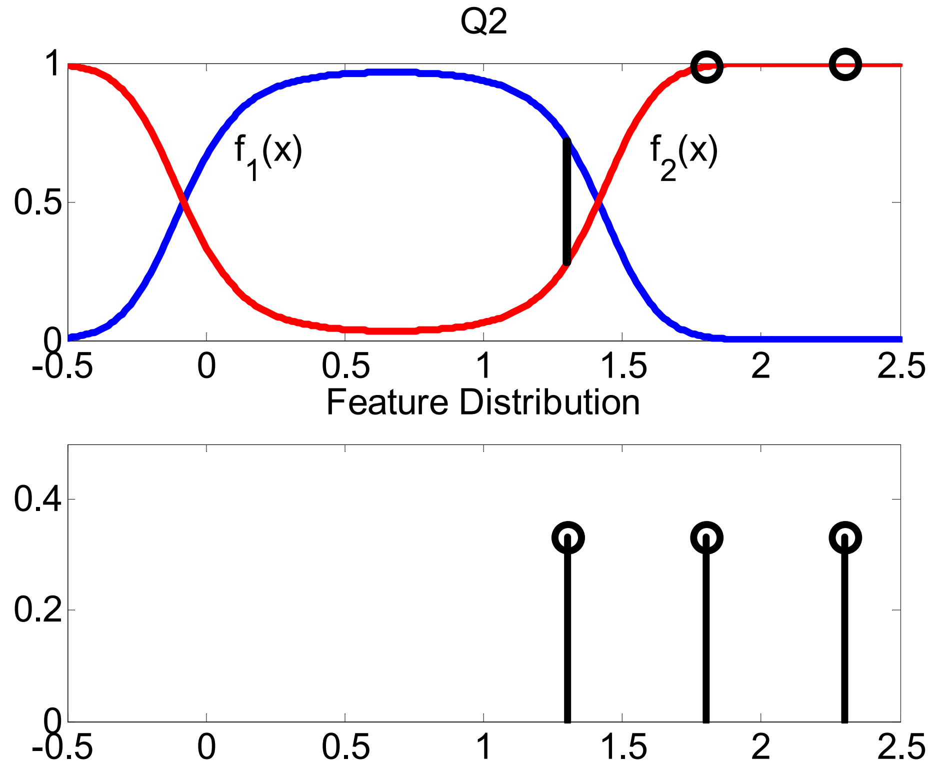
- $0 \le Q_2 \le 1$, the higher the Q_2, the more confident we are.
- $Q_2=1$ if and only if c_k is the minimum-risk decision at all region of the feature space with non-zero probability.
- We did not use the expected risk itself as the quality of decision because we are judging the decision, not the classifier.



Q1 vs. Q2



Q1
10
0
-10
-20
-0.5
0
0.5
1
1.5
2
2.5
Feature Distribution
0.4
0.2
0



Q2
1
0.5
0
$f_1(x)$
$f_2(x)$
-0.5
0
0.5
1
1.5
2
2.5
Feature Distribution
0.4
0.2
0

Quality of Decision—Naïve Bayesian Classifier

- By “naïve”, we assume that the distributions of the feature values are conditionally independent given the class labels;
- By “Bayesian”, we restrict our discriminant functions to be the posterior distributions of each class.
- It has been shown that the performance of naïve Bayesian classifiers are competitive with other sophisticated classifiers.
- The computation is simplified with the assumption of conditional independence.

Quality of Decision—Naïve Bayesian Classifier

- For Q1, instead of computing the expectation over the joint distribution, we can compute the expectation over each feature separately, and then take the sum. Therefore, the computation becomes simpler:

$$E_{\vec{X}}\left[\log f(\vec{x})\right] \;\Rightarrow\; \sum_j E_{X_j} \log P(x_j \mid c_i) + \log P(c_i)$$

- For Q2, the Monte Carlo method becomes easier, because when sampling, we can draw samples for each feature following its own marginal distribution, independent of other features, and then put them together.

Q1 under Naïve Bayesian Classifier

$$E_{\vec{X}}[\log f_i(\vec{x})] = E_{\vec{X}}[\log P(c_i \mid \vec{x})] = E_{\vec{X}}\left[\log \frac{P(\vec{x} \mid c_i)P(c_i)}{\sum_j P(\vec{x} \mid c_j)P(c_j)}\right]$$

$$\begin{aligned}
&E_{\vec{X}}\left[\log\left(P(\vec{x} \mid c_i)P(c_i)\right)\right] \\
&= E_{\vec{X}}\left[\log P(\vec{x} \mid c_i)\right] + E_{\vec{X}}\left[\log P(c_i)\right] \\
&= E_{\vec{X}} \sum_j \log P(x_j \mid c_i) + \log P(c_i) \\
&= \sum_j E_{\vec{X}} \log P(x_j \mid c_i) + \log P(c_i) \\
&= \sum_j E_{X_j} \log P(x_j \mid c_i) + \log P(c_i)
\end{aligned}$$

Q2 under Naïve Bayesian Classifier

$$Q_2 = 1 - \int_{\vec{x}} \left[\frac{P(\vec{x} \mid c^*)P(c^*) - P(\vec{x} \mid c_k)P(c_k)}{\sum_j P(\vec{x} \mid c_j)P(c_j)} \right] p(\vec{x})d\vec{x}$$

$$= 1 - \int_{\vec{x}} \left[\frac{\prod_i P(x_i \mid c^*)P(c^*) - \prod_i P(x_i \mid c_k)P(c_k)}{\sum_j \prod_i P(x_i \mid c_j)P(c_j)} \right] p(\vec{x})d\vec{x}$$

- The Monte Carlo method becomes easier, because when sampling, we can draw samples for each feature following its own marginal distribution, independent of other features, and then put them together.

Feature Prediction—Challenges

- The feature distribution in the next time unit: $\vec{x} \sim p(\vec{x})$
- We can simply use the prior distribution.
- However, we can do better—taking advantage of temporal locality:
  - sensors that measure temperature or river water level
  - images of consecutive snapshots from satellite
- In addition, the data characteristics may drift with time—time-varying models.

Movement prediction

- To make intelligent load shedding decisions at time t, we need to know the distribution of a point’s position at time t+1.
- Our assumption:
  - a point’s location in the feature space at time t+1 solely depends on its location at time t.
  - features are independent to each other with regard to points’ movement
- We build a Markov model for each feature

Markov Model

- Let X be a feature that has M distinct values.
- We learn a state transition matrix K of size M£M, where entry K_{ij} is the probability that X will take value j at time t+1 given X=i at time t.
- We derive K through MLE (maximum likelihood estimation):

  $$K_{ij} = n_{ij} / \sum_k n_{ik}$$

  the fraction of transitions from i to j among all transitions from i to k, for all possible k.

Feature Prediction—Finite-Memory Markov-Chains

- A discrete-time Markov-chain is defined over a set of M states, $s_1, \ldots, s_M$, and an $M \times M$ state transition probability matrix P, where P_{ij} is the probability of moving to state s_j in the next time unit given the current state being s_i ;
- Can handle both categorical and numerical features;
- If a data stream gets load at time t_0, then $p_0(x)=e_i$;
- If a data stream does not get load at time t_j, then $p_j(x)=p_{j-1}(x)P$;

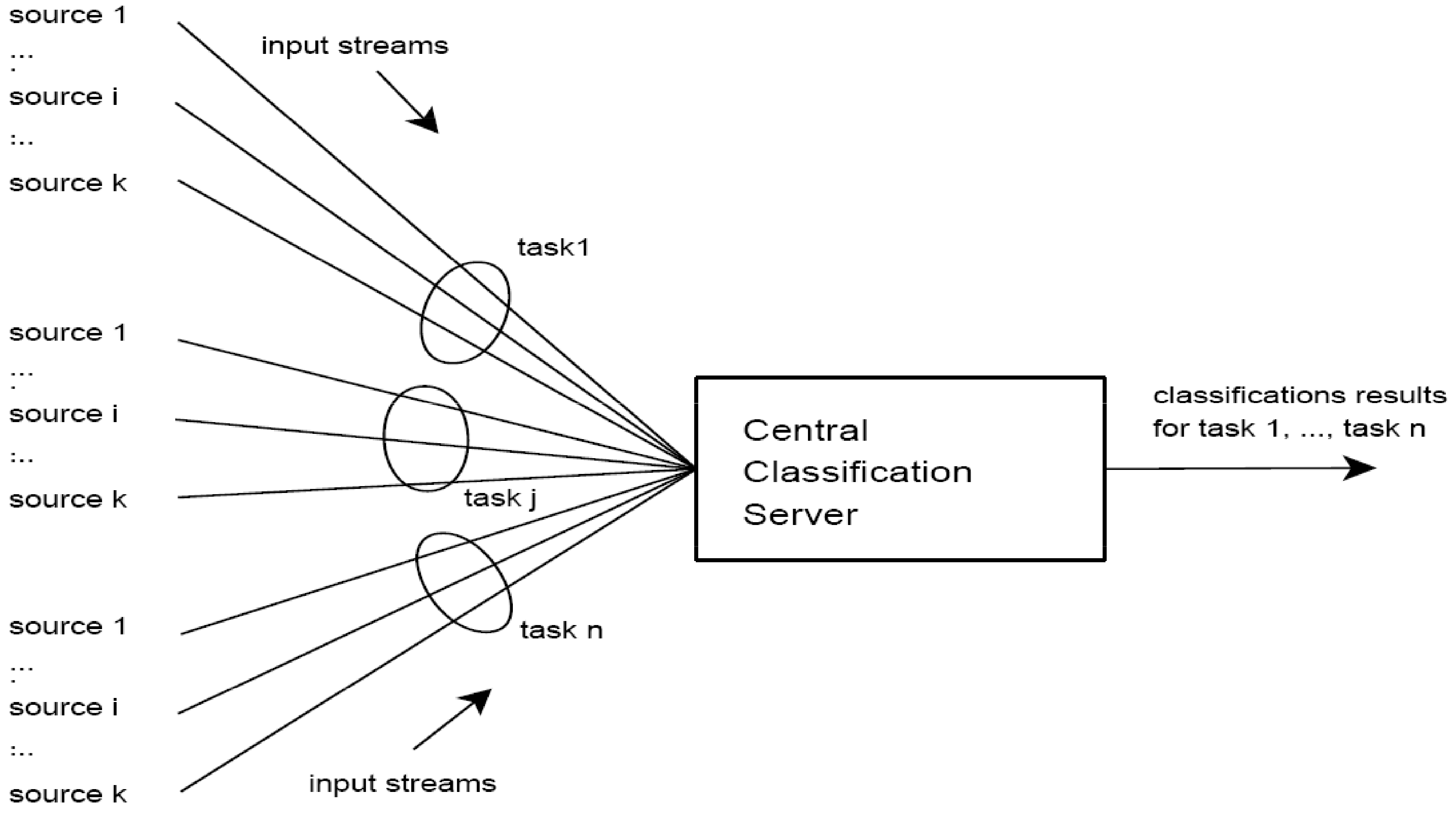
Feature Prediction—Learning Parameters for the MCs

- The parameters of the Markov-chains are updated using the most recent W historic transactions to adapt to drifts.
- Without load shedding, we can have the maximum-likelihood estimation:

$$\hat{P}_{ij} = \frac{n_{ij}}{\sum_k n_{ik}}$$

- With load shedding, some observations are missing, so EM algorithms are needed, which are time consuming.
- An approximate algorithms: when a data stream gets loads, it always gets a pair of consecutive loads (whenever possible), then we still use the most recent *W* transitions.

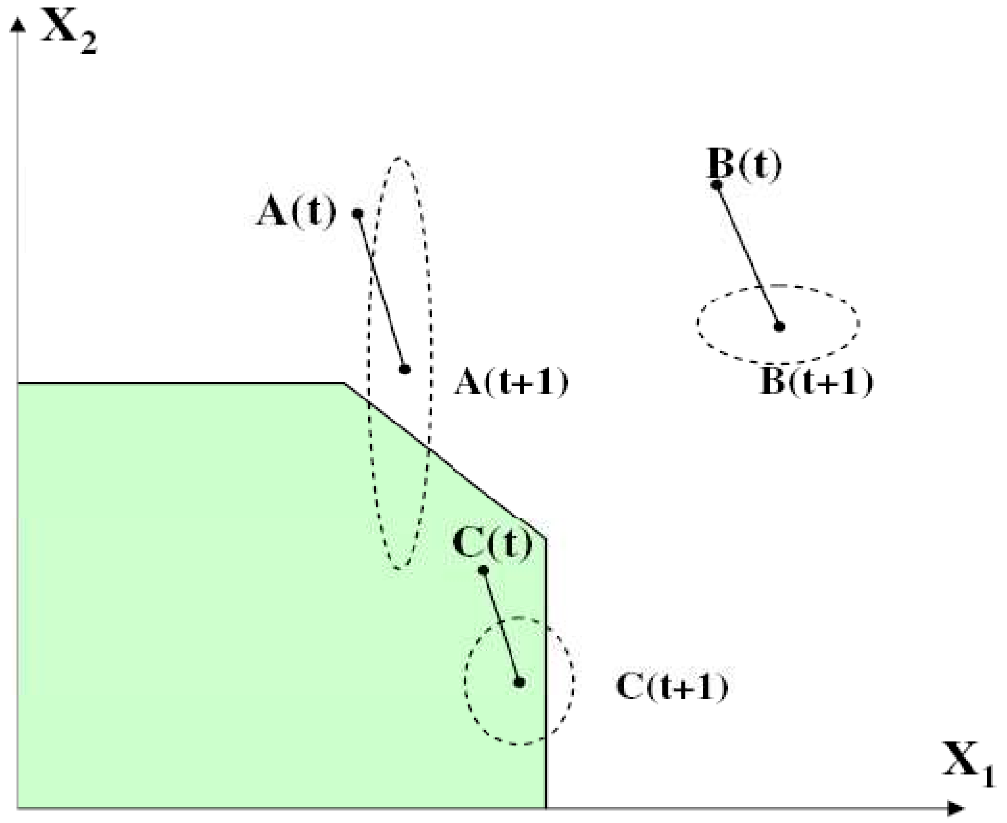
Multi-attribute/ Multi-stream



Problem

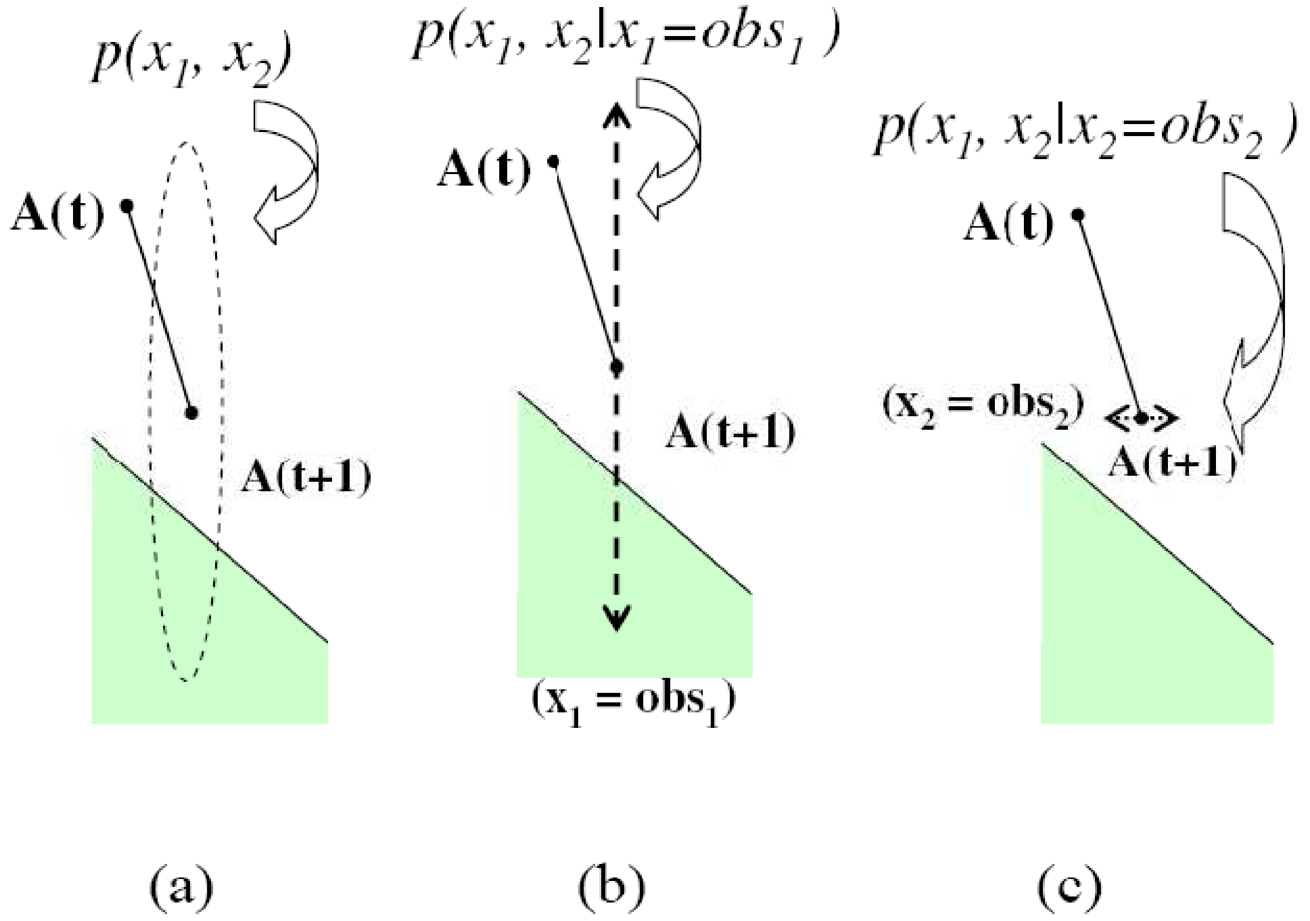
- A central classifier
  - monitors **n×k** streams from **n** tasks,
  - has capacity to process **m** out of the **n×k** input streams.
- which of the input streams should be inspected so that the classification quality is least affected?

Task movement in the feature space

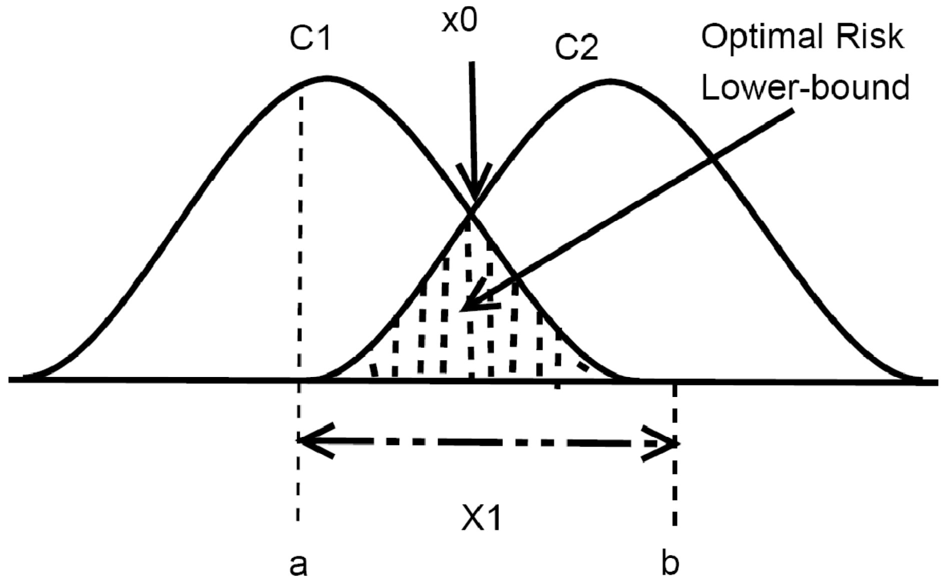


- For A, it is more beneficial to observe X_2 than X_1;
- For C, X_1 than X_2;
- For B, neither observation is critical for classification.

Intuition

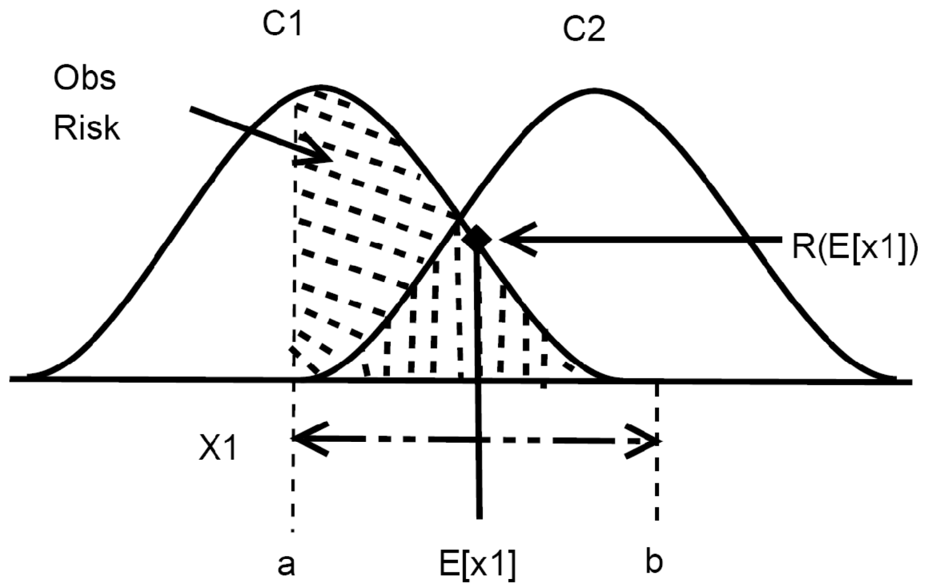

$p(x_1, x_2)$
A(t)
A(t+1)
$p(x_1, x_2 | x_1 = obs_1)$
A(t)
A(t+1)
$(x_1 = obs_1)$
$p(x_1, x_2 | x_2 = obs_2)$
A(t)
$(x_2 = obs_2)$
A(t+1)
(a)
(b)
(c)

Dissecting the risk



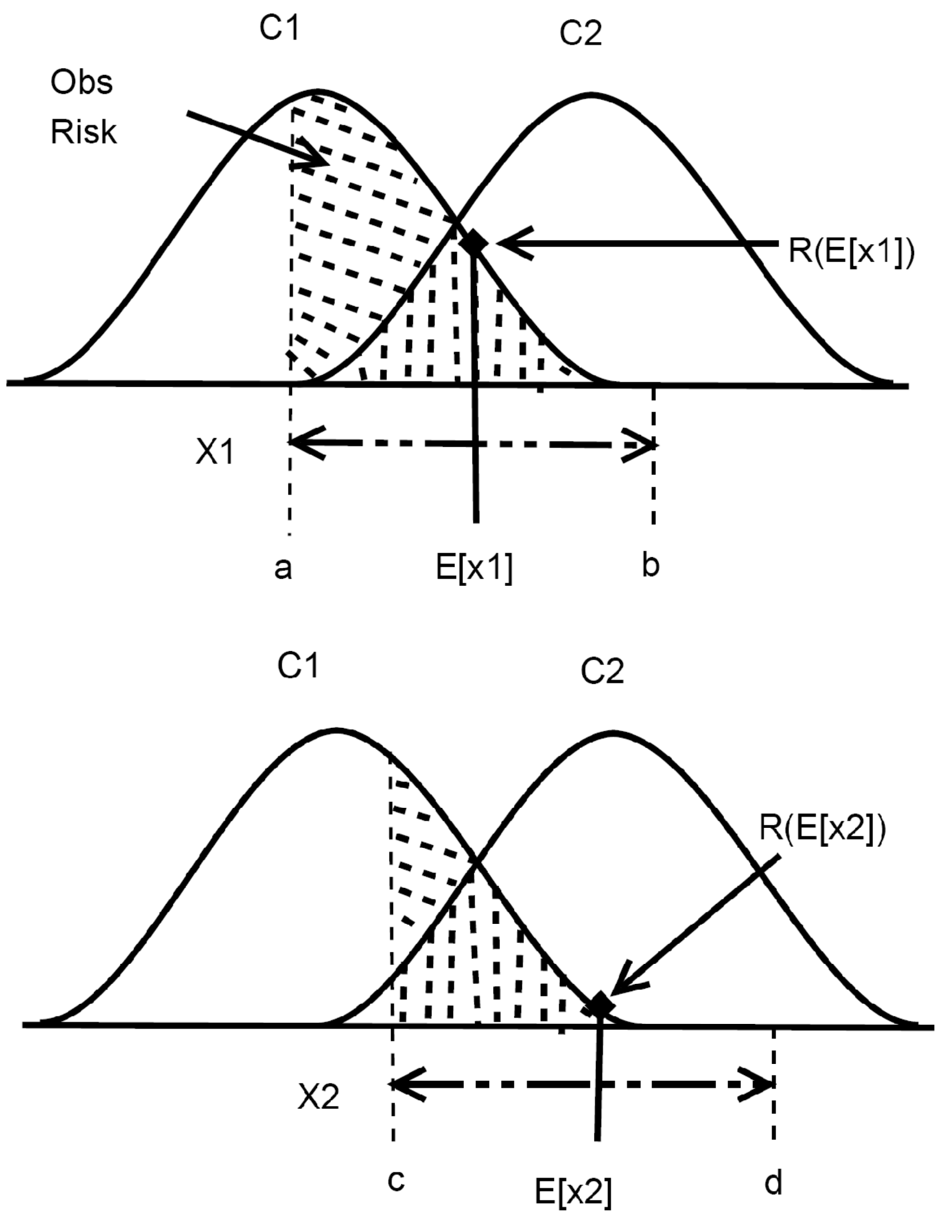
- Let $p(C_1|x)$ and $p(C_2|x)$ be the posterior distributions of two classes C_1 and C_2.
- If $X_1=x$ at time t+1, we predict C_1 if $p(C_1|x) > p(C_2|x)$
- If x distributes uniformly in range [a, b], the unavoidable, lowest expected risk is shown as the shaded area.

Dissecting the risk



- If we don't know x at time t+1, we still need to make a prediction.
- Assume we predict C_2, then the risk is the shaded area
- We call the enlarged area "***observational risk***", because it is caused by our failure to observe the value of X_1

Dissecting the risk



- Which attribute X_1 or X_2 to observe?
- $E[X_2]$ has much lower risk.
- However, observing X_1 reduces a much larger amount of observational risk

Analysis

- The optimal decision at location $\mathbf{x}$ is

$$c^* = \operatorname{argmin}_i R(c_i | \mathbf{x})$$

- Risk decomposition:

Lowest possible, unavoidable risk

$$\begin{aligned} R_{before}(c_k) &= E_{\mathbf{x}}[R(c_k|\mathbf{x})] = \int_{\mathbf{x}} R(c_k|\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ &= \boxed{\overbrace{\int_{\mathbf{x}} R(c^*|\mathbf{x})p(\mathbf{x})d\mathbf{x}}^{\text{Optimal Risk Lower-bound}}} \\ &+ \boxed{\underbrace{\int_{\mathbf{x}} [R(c_k|\mathbf{x}) - R(c^*|\mathbf{x})]p(\mathbf{x})d\mathbf{x}}_{\text{Expected Observational Risk}}} \end{aligned}$$

Risk introduced by lack of knowledge of true data

Analysis

- Risk to minimize:

$$R^{obs}_{before}(c_k) = \int_{\mathbf{x}} [R(c_k|\mathbf{x}) - R(c^*|\mathbf{x})]p(\mathbf{x})d\mathbf{x}$$

- Risk after observing feature x_j:

$$R^{obs}_{after}(c^0_k|obs_j) = \int_{(\mathbf{x}|x_j = obs_j)} [R(c^0_k|\mathbf{x}) - R(c^*|\mathbf{x})]p(\mathbf{x}|obs_j)d\mathbf{x}$$

- Our metric (we use expected value of x_j to replace x_j)

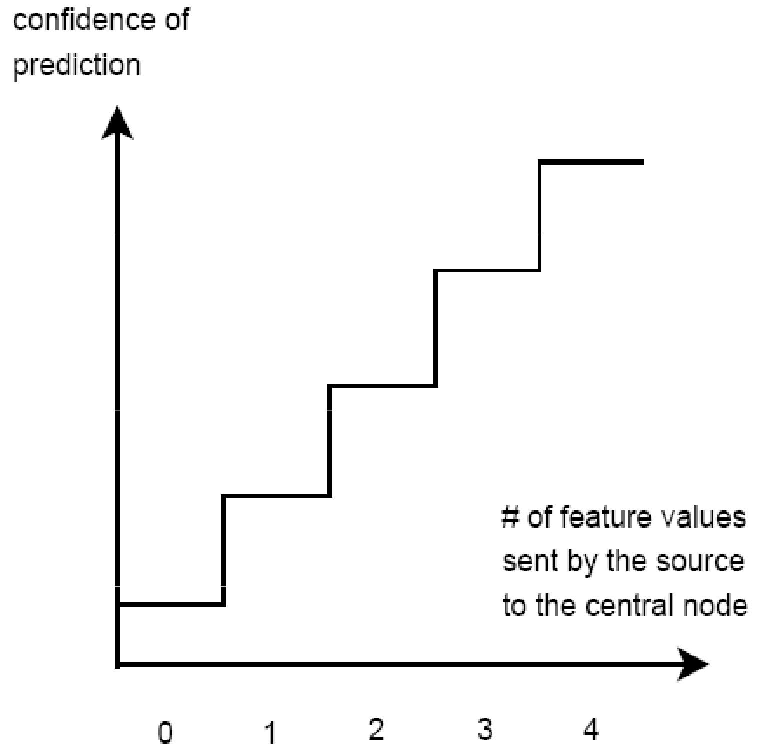
$$Q_{Obs}(X_j) = R^{obs}_{before}(c_k) - R^{obs}_{after}(c^0_k|E[x_j])$$

Load Shedding through Data Transform

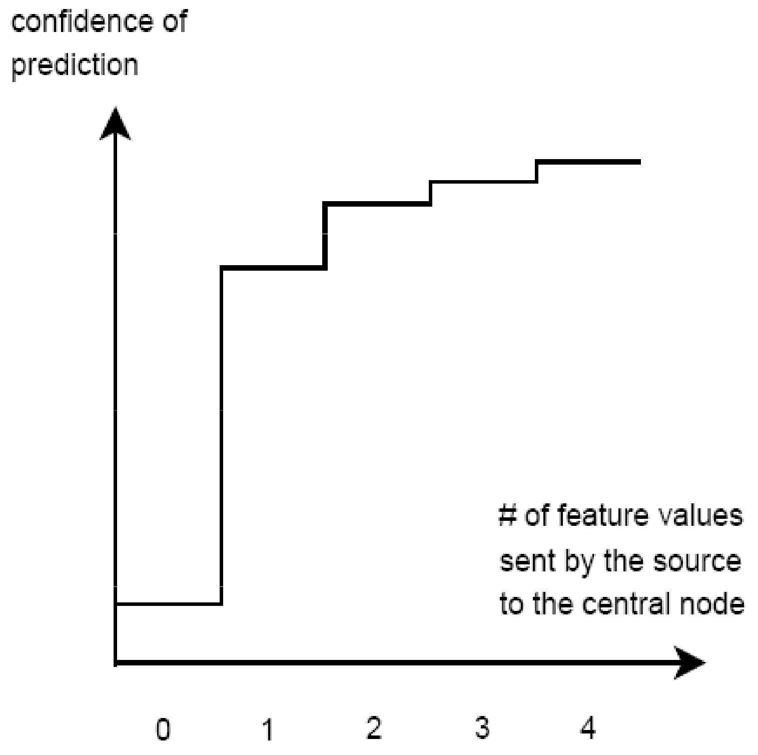
- Progressive classifier $\{C_{I_1}, C_{I_1 \cup I_2}, ...\}$
  - $\{I_1, I_2, \ldots\}$ is partial information of *x*
  - C_{I_1} predicts *x*'s class based on I_1
  - $C_{I_1 \cup I_2}$ predicts *x*'s class based on I_1 and I_2
  - …
- The central node takes the following steps to classify *x*:
  - make prediction based on partial information received from *x*
  - Decide what additional partial information I_k can best improve the confidence of prediction
  - based on how much confidence I_k can improve, decide if the source node should send I_k.
- Partial information I_k can be the value of *x* on the *k*-th dimension.

Progressive classifier

- A progressive classifier based on such partial information may not be optimal.



a



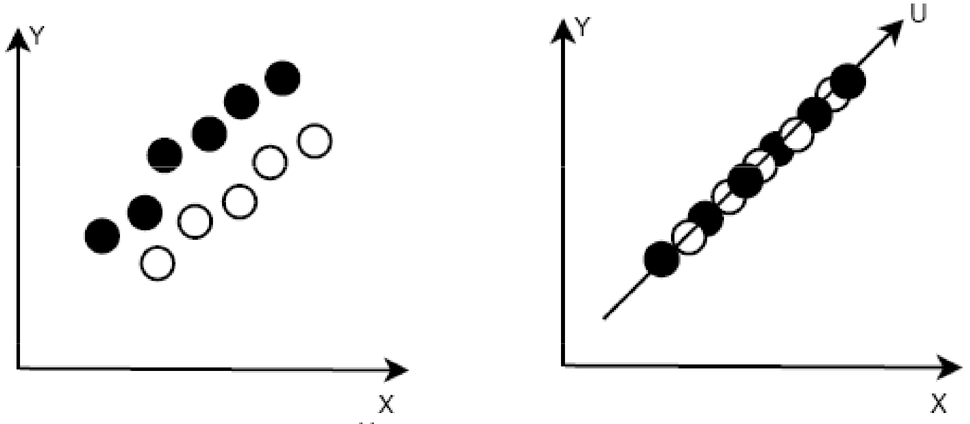
b

Progressive classifier

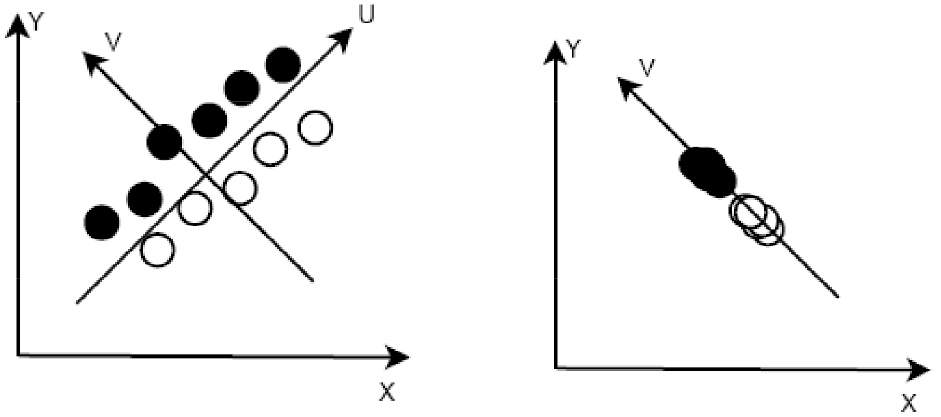
- We want to design a progressive classifier which has the following properties:
  - The “class information” concentrates on the first few features
  - The features are independent of each other
    - Prevents a classifier from inspecting a feature whose “class information” is covered by features already inspected

Transform

- To compress class information into a few feature values, a transform is required.
- However, many traditional transforms are not suitable for supervised learning.



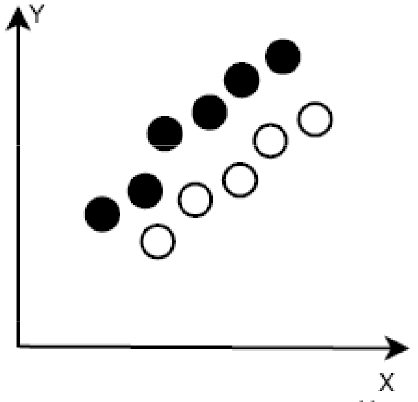
Transform to minimize mean square error

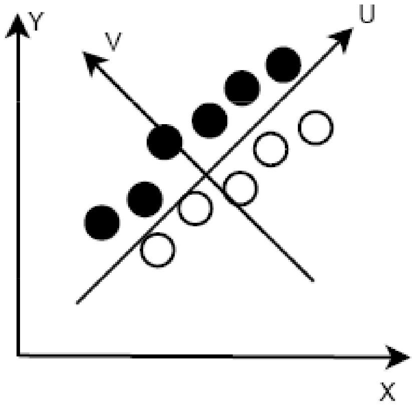


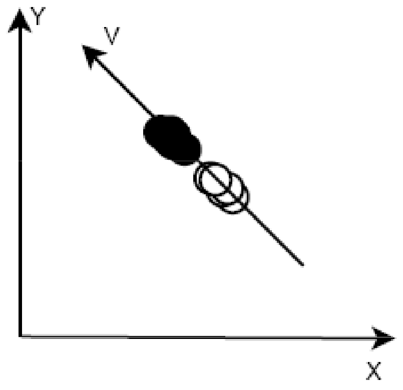
Transform to maintain class separability

Transform

- To compress class information into a few feature values, a transform is required.
- However, many traditional transforms are not suitable for supervised learning.







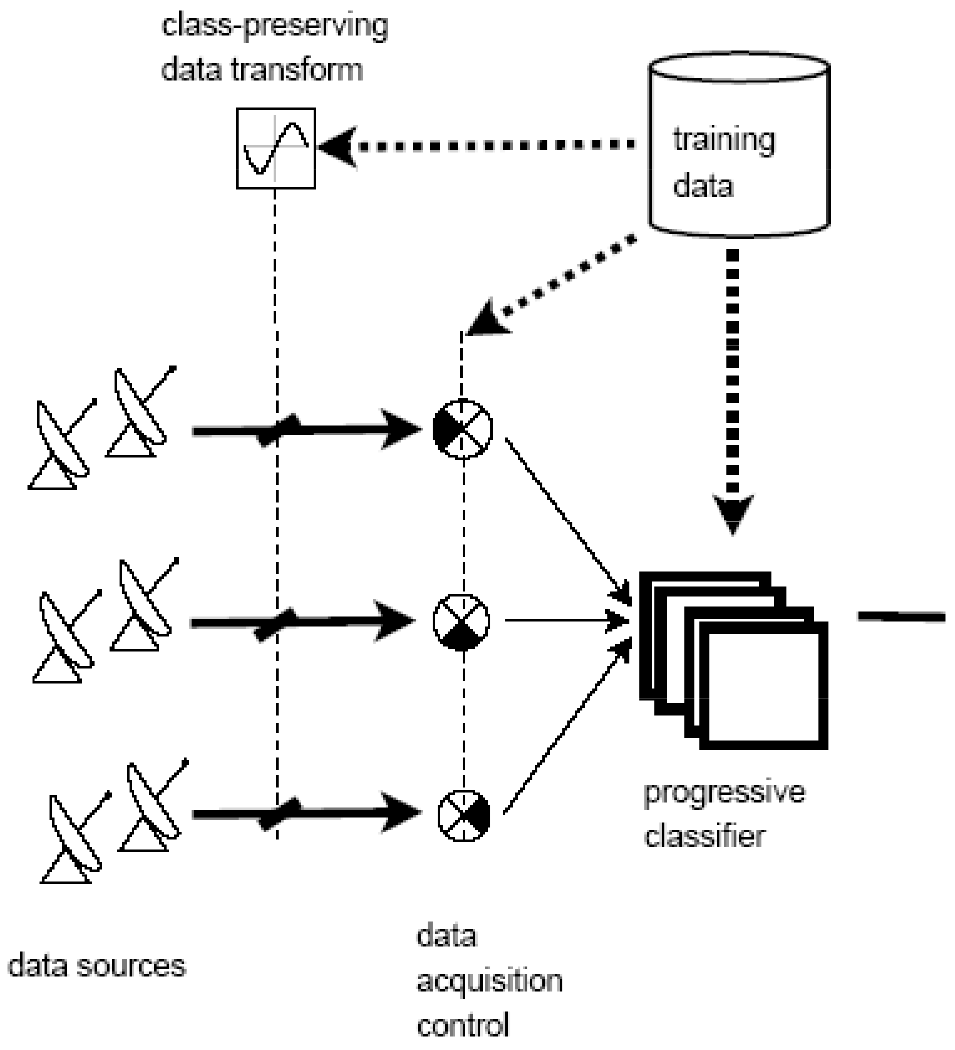
Transform to maintain class separability

This transform is desired

Temporal locality

- In many stream applications, data exhibits **temporal locality**.
  - temperatures of a region usually do not change dramatically over a short period of time
  - temporal locality allows us to focus on changes in the data rather than the data itself.
- We hope that the transform can preserve temporal locality.

Architecture



algorithm

- Overview
- Class Preserving Transform
- Data Acquisition Scheme
- Data Acquisition Scheme

Overview of LOCI

- Two phases:
- Training phase:
  - learn a data transform matrix *U*
  - learn a data acquisition scheme
  - learn the progressive classifier
- Testing phase:
  - Incoming record is transformed with U
  - Using the learned data acquisition scheme, the central server queries source nodes for partial information of the data
  - classifier classifies the data using the partial information

Class Preserving Transform

- principal component analysis (PCA)
  - *x* is the vector in *d*-dimensional space *X*
  - $\bar{x}$ is the mean of *x*
  - Covariance matrix is $\Sigma_x = E[(x-\bar{x})(x-\bar{x})']$
  - Let $\lambda_1, \cdots, \lambda_d$ be the eigenvalues of Σ_x and satisfy
    - $\lambda_1 \geq \cdots \geq \lambda_d$
  - $U=(u_1,u_2,\ldots,u_d)$ is the transform matrix of PCA, where u_i is the eigenvector of λ_i
  - for each vector *x*, the new vector after transform

    $y = U'x$
  - But PCA doesn’t consider the class information

Class Preserving Transform

- KL-3
  - assume μ is the mean of all examples and μ_i is the mean of examples in class C_i
  - Use $S_w = \sum_{i=1}^{K} p_i \overline{\Sigma}_i$ to replace Σ_x
    - Where $\overline{\Sigma}_i = E[(x-\mu_i)(x-\mu_i)']$
  - A new transform matrix $U=(u_1,u_2,\ldots,u_d)$ is obtained from S_w
  - The transformed space is $Y=(Y_1,Y_2,\ldots,Y_d)=U'X$
  - $(Y_1,Y_2,\ldots,Y_d)$ is ranked according to $J(Y_j) = \frac{u_j S_b u_j}{\lambda_j}$
    - Where $S_b = \sum_{i=1}^{K} p_i (\mu_i - \mu)(\mu_i - \mu)'$

Class Preserving Transform

- KL-3 satisfies all the requirements:
  - independence
    - $\sum_y = E[(y-\bar{y})(y-\bar{y})'] = U' S_w U = diag(\lambda_1, ..., \lambda_d)$
  - temporal locality preserving
    - with Parseval's theorem, we get $\sum_{i=1}^{d} |x_i|^2 = \sum_{i=1}^{d} |y_i|^2$
    - hence we get $\|x - x'\|^2 = \|y - y'\|^2$
  - class information preserving
    - the new criterion *J(Y$_i$)* aims at maximizing intra-class similarity and minimizing inter-class similarity

Data Acquisition Scheme

- Temporal locality
  - In many cases, for a feature, the values at adjacent time unit is similar
  - If y_t is the same as y_{t-1}, source node only needs tell central node no change happens
- First source nodes send boolean flag
  - 0 means no change happens
  - 1 means change happens
- Then, if necessary, source nodes send real values
  - If 0, <u>no need</u>
  - If 1
    - If $y_t \neq y_{t-1}$ has similar class information with y_t, <u>no need</u>
    - If $y_t \neq y_{t-1}$ has much less class information with y_t, <u>need</u>

Data Acquisition Scheme

- We consider three cases:

temporal locality	class information	flag sent	information obtained	bandwidth used
$y_t = y_{t-1}$	$=$	0	y_t	1
$y_t \neq y_{t-1}$	$\approx$	1	$\neg y_{t-1}$	1
$y_t \neq y_{t-1}$	$<$	1	y_t	$1 + s$

  - $\neg y_{t-1}$ means $y_t \neq y_{t-1}$,called **negative knowledge**
  - at the first two cases, *1* bit is needed.
  - At the third case, *1+s* bits are needed.
- Our goal is to design a data acquisition schema that mostly operates in the first two cases.

Reference Vector and Boolean Vector

- Source nodes compare the new vector with that in reference vector to decide whether change happens
  - Each source node S_i has a reference vector $V_i = (v_{i1}, v_{i2}, ..., v_{id})$
  - Each V_i has a mirror vector V_i' in central node
- Source nodes use boolean vector to tell central node which values change
  - To facilitate measuring communication cost, boolean vector has the same size with real value
  - Boolean vector is composed of *L* ‘0’ or ‘1’.
  - *L* is the size of real value
  - At each time unit, each source node generates $\lceil d / L \rceil$ boolean vectors

Example

- At time *t*, S_1 generates a new 8-dimensional KL-3 vector {1,2,3,4,4,5,3,1}
- Assume *L*=4
- $V_1 = V_1'$={1,4,3,3,4,2,3,1}
- Then S_1 has 2 boolean vectors: {0,1,0,1} and {0,1,0,0}
- When S_1 sends {0,1,0,1} to central node, central node obtains partial information as {1,¬4,3, ¬3}.
- When S_1 sends {0,1,0,0}, central node has {1,¬4,3, ¬3,4, ¬2,3,1}
- With 2 4-bit transmissions, central node obtain 5 specific values plus some negative knowledge.
- When S_1 sends 2nd value, central nodes has {1,2,3, ¬3,4, ¬2,3,1}. V_1 and V_1' change to {1,4,3,3,4,2,3,1}.

More definitions

Let (y_1,y_2,…,y_d) be the new KL-3 vector at S_i and (v_{i1},v_{i2},…,v_{id}) be the reference vector at S_i

- **next boolean vector**

  If Si already sends *k* boolean vector, *(k+1)*-th boolean vector is called as *next boolean vector*, denoted as $bv_n(i)$

- **changed value**

  if $y_k \neq v_{ik}$, y_k is called a *changed value*. The set of them is denoted as *cv(i)*

  if a changed value is already sent to central node, it is called as *sent changed value*. The set of them is denoted as $cv_s(i)$

  otherwise it is called as *un-sent changed value*. The set of them is denoted as $cv_u(i)$

More definitions

- **Current partial knowledge**
  - Assume S_i already sent k boolean vectors, central node's current partial knowledge of S_i is denoted as $dc(i)=\{d_1,d_2,\ldots,d_{kL}\}$

$$d_j = \begin{cases} v_{ij} & : \quad y_j \text{ is not a changed value} \\ y_j & : \quad y_j \text{ is a sent changed value} \\ \neg vij & : \quad y_j \text{ is an un-sent changed value} \end{cases}$$

- continue the above example
  - After receiving {0,1,0,1}, $bv_n(1)$={0,1,0,0}, $cv(1)$={2,4}, $cv_s(1)=\emptyset$, $cv_u(1)$={2,4}, $dc(i)$={1,¬4,3,¬3}
  - If at next step, S_i sends 2nd value, $bv_n(1)$={0,1,0,0}, $cv(1)$={2,4}, $cv_s(1)$={2}, $cv_u(1)$={4}, $dc(i)$={1,2,3,¬3}

Data Acquisition Scheme

- At each time unit, it contains multiple steps

1 each source node sends the first boolean vector

2 central node selects a source node to query one more value

3 after deciding source node, central node selects whether real value or boolean vector is sent, and which one

4 this process continues until central node receives *C* values

Select source node

- Central node queries a source node for more value if
  - Its current classification is of low quality
  - More information form it can improve the quality
- Classification quality $\alpha(\mathrm{i}) = \mathrm{p}(\mathrm{c} \mid \mathrm{d_c}(\mathrm{i})) - \mathrm{p}(\tilde{\mathrm{c}} \mid \mathrm{d_c}(\mathrm{i}))$
  - Where *c* is the class whose posterior probability is highest and $\tilde{\mathrm{c}}$ is the second best class.
- Consumed bandwidth $\beta(\mathrm{i}) = \log(\mathrm{n_{all}} - \mathrm{n_{sent}} + 1)$
  - Where $\mathrm{n_{all}} = \mathrm{d} + \lceil \mathrm{d}/\mathrm{L} \rceil$ and $\mathrm{n_{sent}} = |\mathrm{d_c}(\mathrm{i})|/\mathrm{L} + |\mathrm{cv_s}(\mathrm{i})|$
- The criterion of selecting source node is

$$\mathrm{Q}(\mathrm{i}) = \frac{\alpha(\mathrm{i})}{\beta(\mathrm{i})}$$

Select value

- There are $|cv_u(i)|$ unsent changed values and $(\lceil d/L \rceil - k)$ unsent boolean vectors. Central node will decide whether boolean vector or changed value is sent and which one.
- The most beneficial boolean vector
  - Since KL feature is sorted in descending order of class separability, next unsent boolean vector is most beneficial
  - We assume the selected boolean vector is {0,…,0}. The benefit of it is estimated as:

$$\gamma_b(i) = p(c \mid d_c(i), 0) - p(\tilde{c} \mid d_c(i), 0)$$

Select value

- The most beneficial boolean vector
  - select changed value is more difficult
    - Because central node already obtains negative knowledge of it
    - Although certain value has more class information, if its negative knowledge contains similar class information as it, its benefit is still small
  - We select the value which can bring most additional class information given that its negative knowledge is known

$$H_1(j) = p(\neg v)\sum_c - p(c \mid \neg v)\log p(c \mid \neg v)$$
$$H_2(j) = \sum_{k \neq v} - p(k)\sum_c p(c \mid k)\log p(c \mid k)$$
$$H(j) = H_1(j) - H_2(j)$$

  - *H(j)* is used to measure decrease of uncertainty after knowing y_j. The feature with biggest *H* is selected as next changed value.
  - we define its benefit as

$$\gamma_c(i) = \sum_{k \neq v} p(k)(p(c_k \mid d_c(i), k) - p(\tilde{c}_k \mid d_c(i), k))$$

overview

- At each step, central node computes Q for each source node, and selects one with smallest Q
- Assume selected source node is S_i, central node computes $\gamma_b(i)$ and $\gamma_c(i)$, then compare them.
  - If $\gamma_b(i) \geq \gamma_c(i)$, S_i sends next boolean vector
  - If $\gamma_b(i) < \gamma_c(i)$, S_i sends next changed value
- This process repeats until C is reached

Progressive Classification

- The progressive classifier is $\{C_{I_1}, C_{I_1 \cup I_2}, ...\}$
  - Then what is I_j?
    - We define each I_j contains certain number of KL feature values
      - $I_1=\{y_1,y_2,\ldots,y_L\}$
      - $I_2=\{_{yL+1},y_{L+2},\ldots,y_{2L}\}$
    - y_j is either positive information or negative knowledge
  - C_{I_1} uses $\{y_1,y_2,\ldots,y_L\}$ to make prediction
  - $C_{I_1 \cup I_2}$ uses $\{y_1,y_2,\ldots,y_{2L}\}$ to make prediction
  - …

Progressive Classification

- Each sub-classifier is a naïve Bayesian classifier
- It assigns y to class c_i if
  - $p(c_i \mid y) \geq p(c_j \mid y)$ for $1 \leq j \leq K, j \neq k$
- According to Bayes theorem
  - $p(c_i \mid y) = \dfrac{p(y \mid c_i) p(c_i)}{p(y)}$
  - Where $p(y \mid c_i) = \prod_{k=1}^{d} p(y_k \mid c_i)$ and $p(y) = \prod_{k=1}^{d} p(y_k)$
- For negative knowledge ¬v
  - $p(\neg v \mid c_i) = 1 - p(v \mid c_i)$ and $p(\neg v) = 1 - p(v)$

Conclusion

- Classification is a most important task on stream data analysis
- Data volume and change of data distribution pose challenges to learning models and using models
- Intelligent model reusing and intelligent load shedding are good solutions

Fair Use Agreement