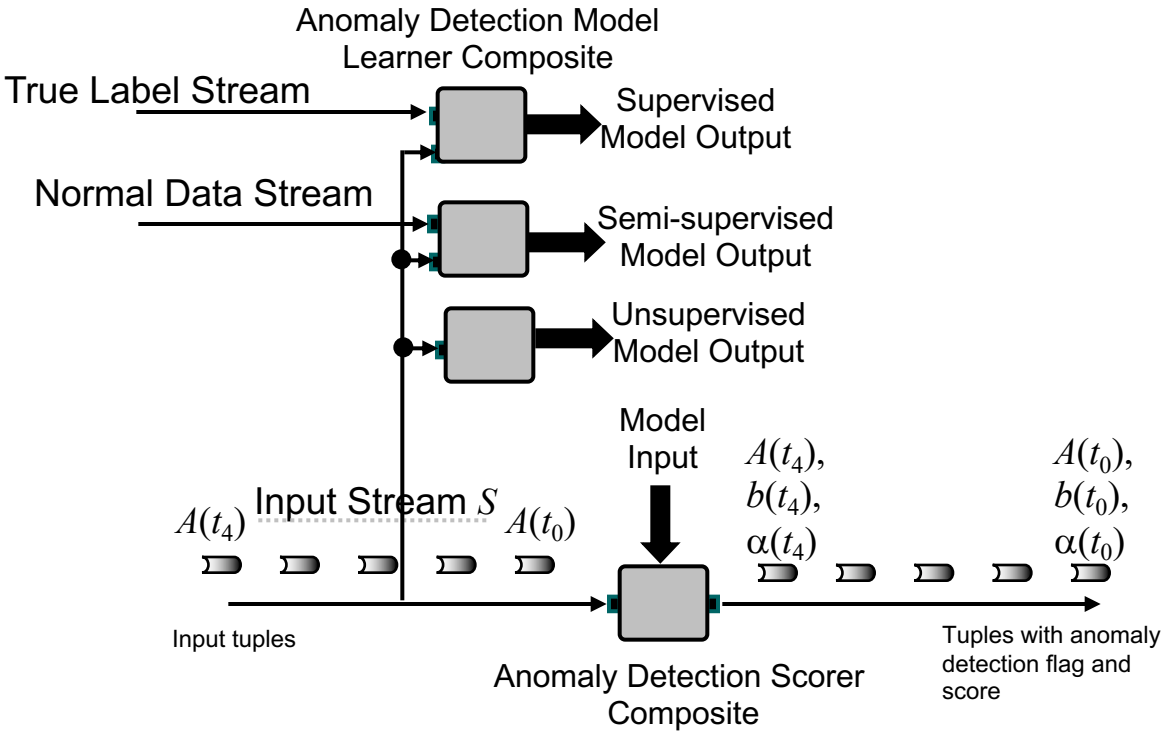
Streaming Algorithms III

Modeling and Evaluation

Outlier/Anomaly Detection

- Identifying patterns in data that do not conform to *expected* behavior
  - Anomaly, outlier, exception, aberration, surprise, peculiarity, or contaminant
- Can be performed on tuples with numeric as well as non-numeric attributes
- Types of Anomalies: Point Anomalies, Contextual Anomalies, Collective Anomalies
- Techniques for Anomaly Detection
  - Supervised
  - Semi supervised
  - Unsupervised

Outlier/Anomaly Detection



In unsupervised approaches no feedback is provided

Outlier/Anomaly Detection: Challenges

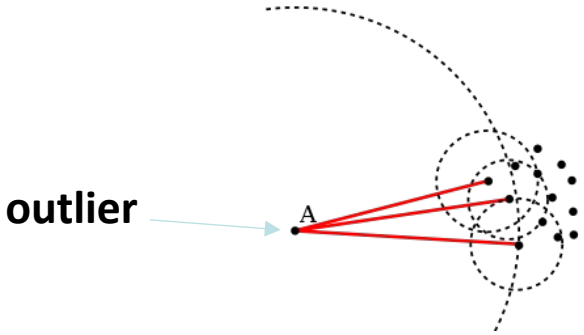
- The objective function or model for anomaly detection may often be subjective/heuristic
  - Should an outlier be distance-based, linear-model based or probabilistic?
  - Such assumptions can often be imperfect, and a specific algorithm being used may model the underlying generative process in a limited way.
- A model may work better on some parts of the data than other.
- Similarly, a given model may be better than another in a data-specific way, which is unknown a-priori.
- In the streaming setting: how can the algorithm adapt?

Outlier/Anomaly Detection Algorithms

- Classification based
  - Build classification model
- Clustering based
  - Use distance from closest centroid
- Nearest neighbor based
  - Use distance from nearest neighbor
- Statistical methods
  - Compute likelihood of being generated by model
- Information theoretic methods
  - Compute reduction in entropy (or other measures) by removing some data items
- Spectral methods
  - Use PCA-like transforms
- Time-series methods
  - Use filters with memory (Kalman, FMP) to find outliers

Example: Local Outlier Factor

- Intuition: a point is an outlier if its feature vector differs from those of its k nearest neighbors
- Unsupervised outlier detection, no labels in training data needed



- No predefined number of outliers needed
- Fast, robust at large scale, since only k nearest neighbors are taken into consideration

LOF

Let $\text{k-distance}(A)$ be the distance of the object A to the k-th nearest neighbor. Note that the set of the k nearest neighbors includes all objects at this distance, which can in the case of a "tie" be more than k objects. We denote the set of k nearest neighbors as $N_k(A)$.

This distance is used to define what is called *reachability distance*:

$$\text{reachability-distance}_k(A, B) = \max\{\text{k-distance}(B), d(A, B)\}$$

In words, the *reachability distance* of an object A *from* B is the true distance of the two objects, but at least the k-distance of B. Objects that belong to the k nearest neighbors of B (the "core" of B, see DBSCAN cluster analysis) are considered to be equally distant. The reason for this distance is to get more stable results. Note that this is not a distance in the mathematical definition, since it is not symmetric. (While it is a common mistake[3] to always use the k-distance, this yields a slightly different method, referred to as Simplified-LOF[3])

The *local reachability density* of an object A is defined by

$$\text{lrd}(A) := 1/\left(\frac{\sum_{B \in N_k(A)} \text{reachability-distance}_k(A, B)}{|N_k(A)|}\right)$$

which is the inverse of the average reachability distance of the object A *from* its neighbors. Note that it is not the average reachability of the neighbors from A (which by definition would be the $\text{k-distance}(A)$), but the distance at which A can be "reached" *from* its neighbors. With duplicate points, this value can become infinite.

The local reachability densities are then compared with those of the neighbors using

$$\text{LOF}_k(A) := \frac{\sum_{B \in N_k(A)} \frac{\text{lrd}(B)}{\text{lrd}(A)}}{|N_k(A)|} = \frac{\sum_{B \in N_k(A)} \text{lrd}(B)}{|N_k(A)|} / \text{lrd}(A)$$

which is the *average local reachability density of the neighbors* divided by the object's own local reachability density. A value of approximately 1 indicates that the object is comparable to its neighbors (and thus not an outlier). A value below 1 indicates a denser region (which would be an inlier), while values significantly larger than 1 indicate outliers.

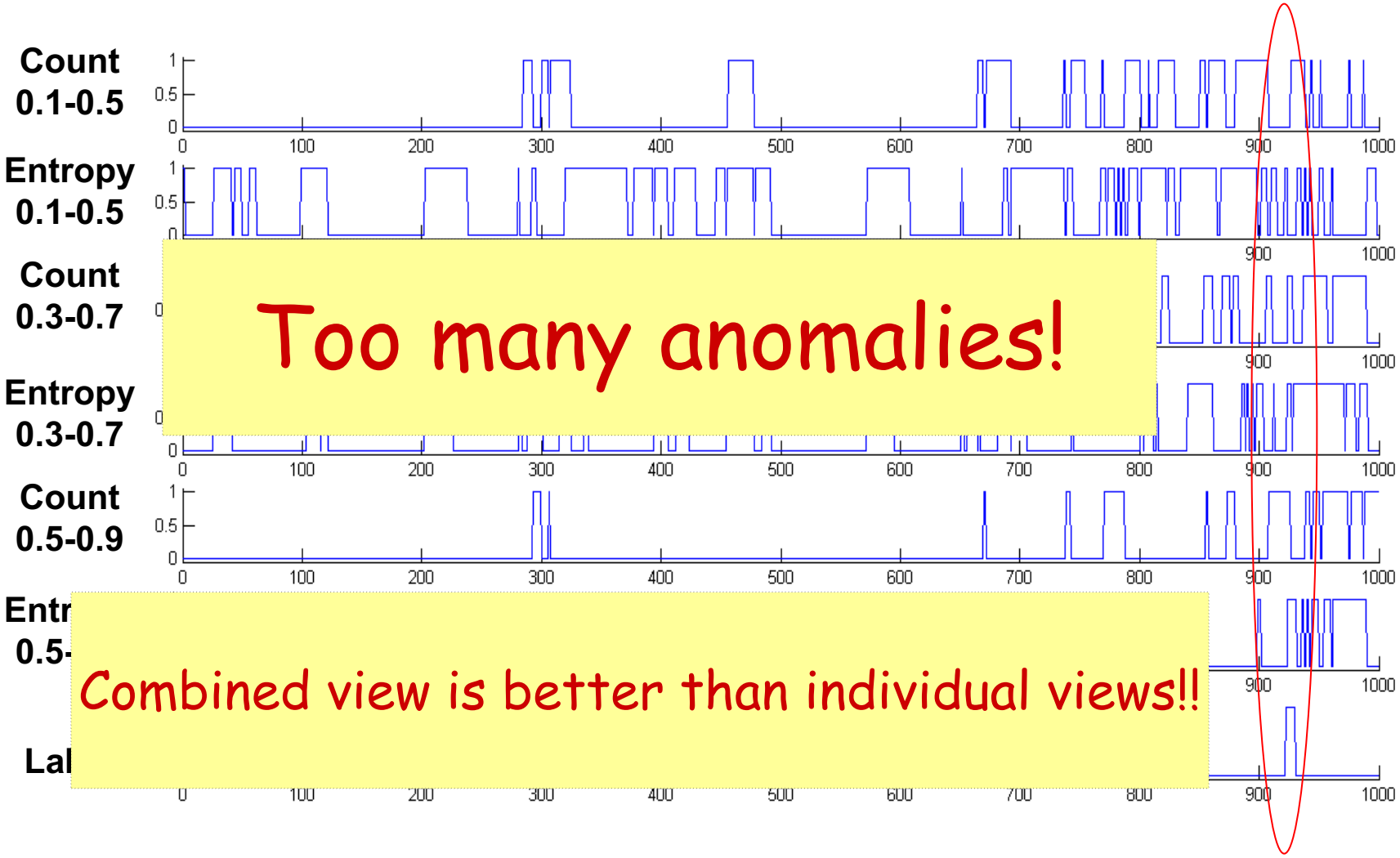
Ensembles for Outlier Detection

- Building ensembles is important
  - Atomic models usually generate many alarms
  - Users are overwhelmed by different rules, and they don't know which ones to trust
- is non-trivial
  - Can we find a consolidated solution without any knowledge of the true anomalies (**unsupervised**)
  - We don't know which atomic models are better and which are worse
  - There could be bad base detectors so that majority voting cannot work

Combining Atomic Detectors

	A_1	A_2	.....	A_{k-1}	A_k
Tuple 1	Y	N	.....	N	N
Tuple 2	N	Y	.....	Y	N
Tuple 3	Y	N	.....	N	N
Tuple 4	Y	Y	.....	N	Y
Tuple 5	N	N	.....	Y	Y
Tuple 6	N	N	.....	N	N
Tuple 7	N	N	.....	N	N

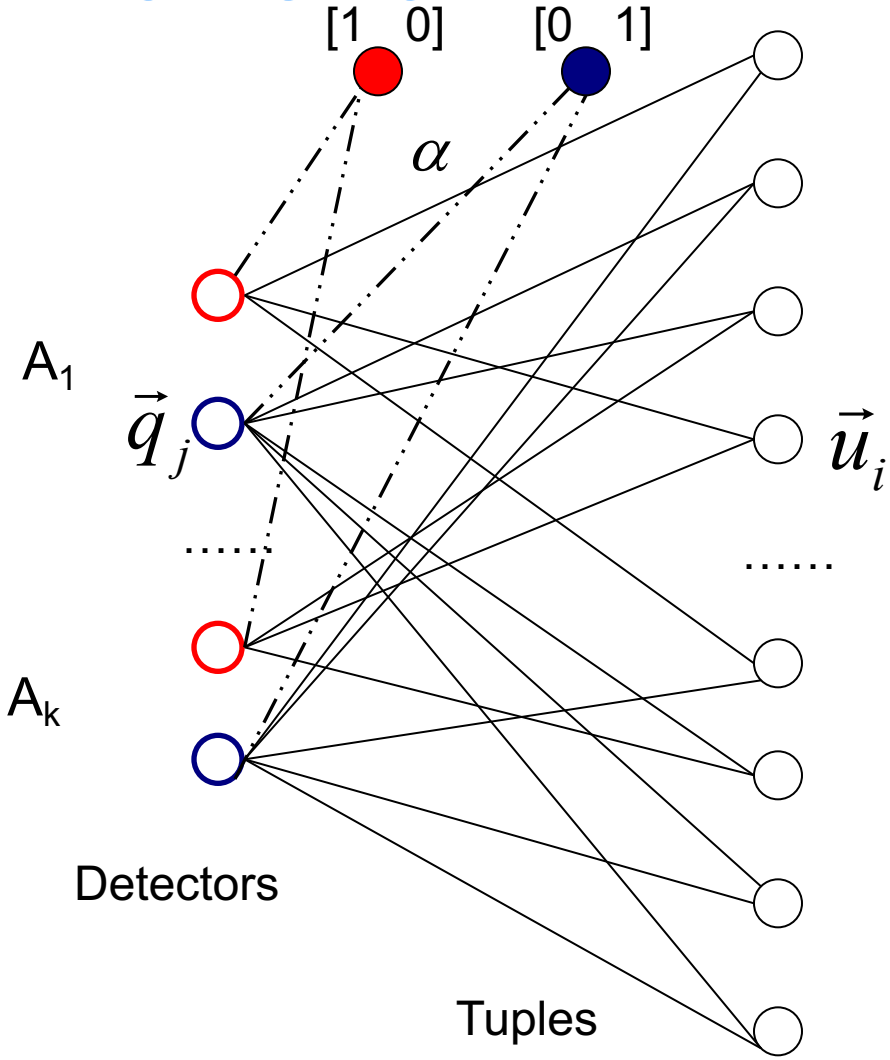
Need to Combine Detectors



Ensembles for Outlier Detection

- Principles
  - Consensus represents the best we can get from the atomic detectors
  - Atomic detectors should be weighted according to their detection performance
  - Tuples should be ranked according to their probability of being an anomaly
- Algorithm
  - Reach consensus among multiple atomic anomaly detectors in an unsupervised way
  - Automatically derive weights of atomic detectors and tuples

Framework



record i $\vec{u}_i = [u_{i0}, u_{i1}]$

detector j $\vec{q}_j = [q_{j0}, q_{j1}]$

probability of anomaly, normal

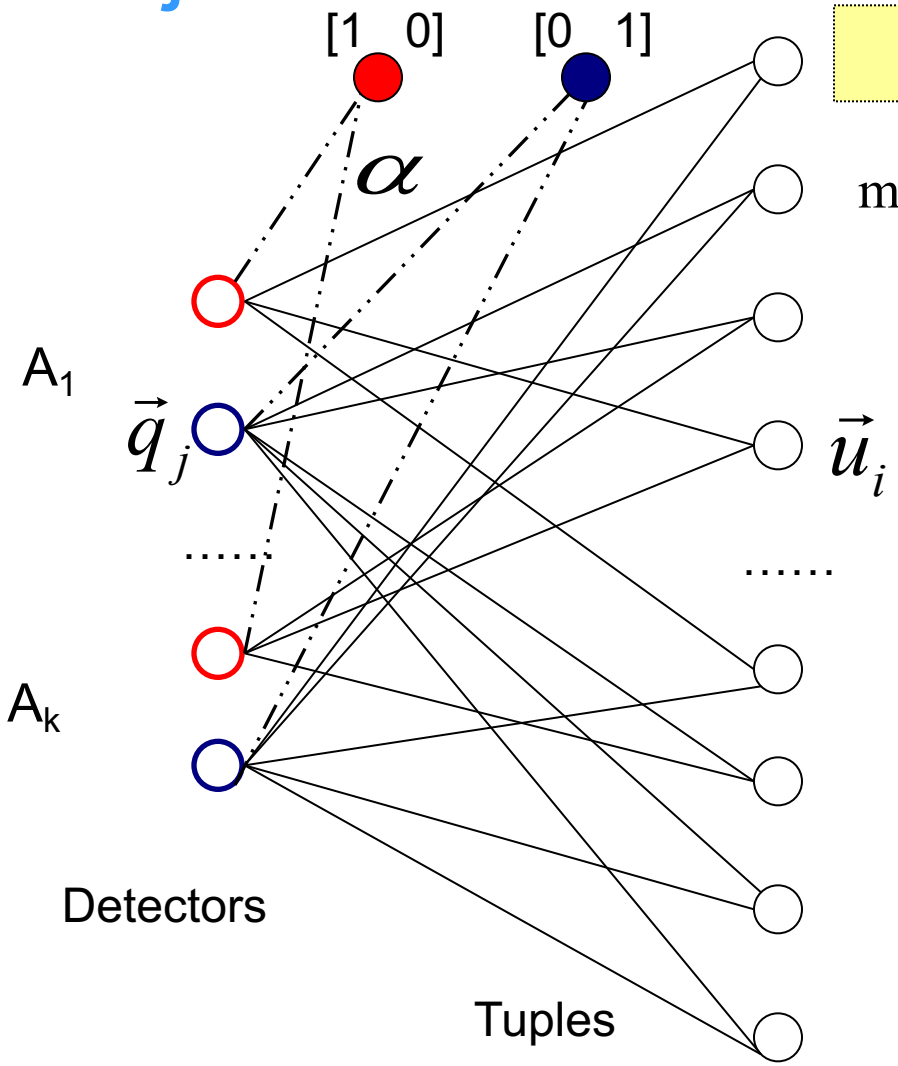
adjacency

$$a_{ij} = \begin{cases} 1 & u_i \leftrightarrow q_j \\ 0 & otherwise \end{cases}$$

initial probability

$$\vec{y}_j = \begin{cases} [1 \quad 0] & anomalous \\ [0 \quad 1] & normal \end{cases}$$

Objective



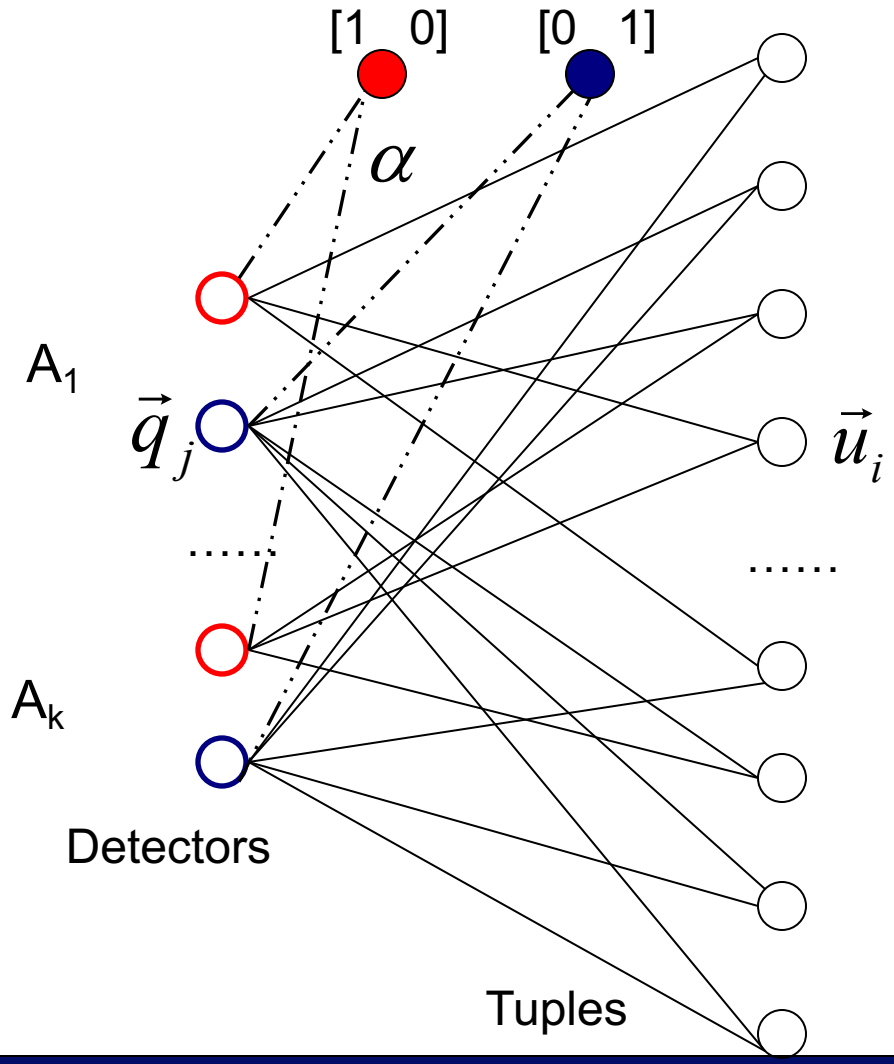
minimize disagreement

$$\min_{Q,U}\left(\sum_{i=1}^{n}\sum_{j=1}^{v} a_{ij} \| \vec{u}_i - \vec{q}_j \|^2 + \alpha \sum_{j=1}^{v} \| \vec{q}_j - \vec{y}_j \|^2\right)$$

Similar probability of being an anomaly if the record is connected to the detector

Do not deviate much from the initial probability

Methodology



Iterate until convergence

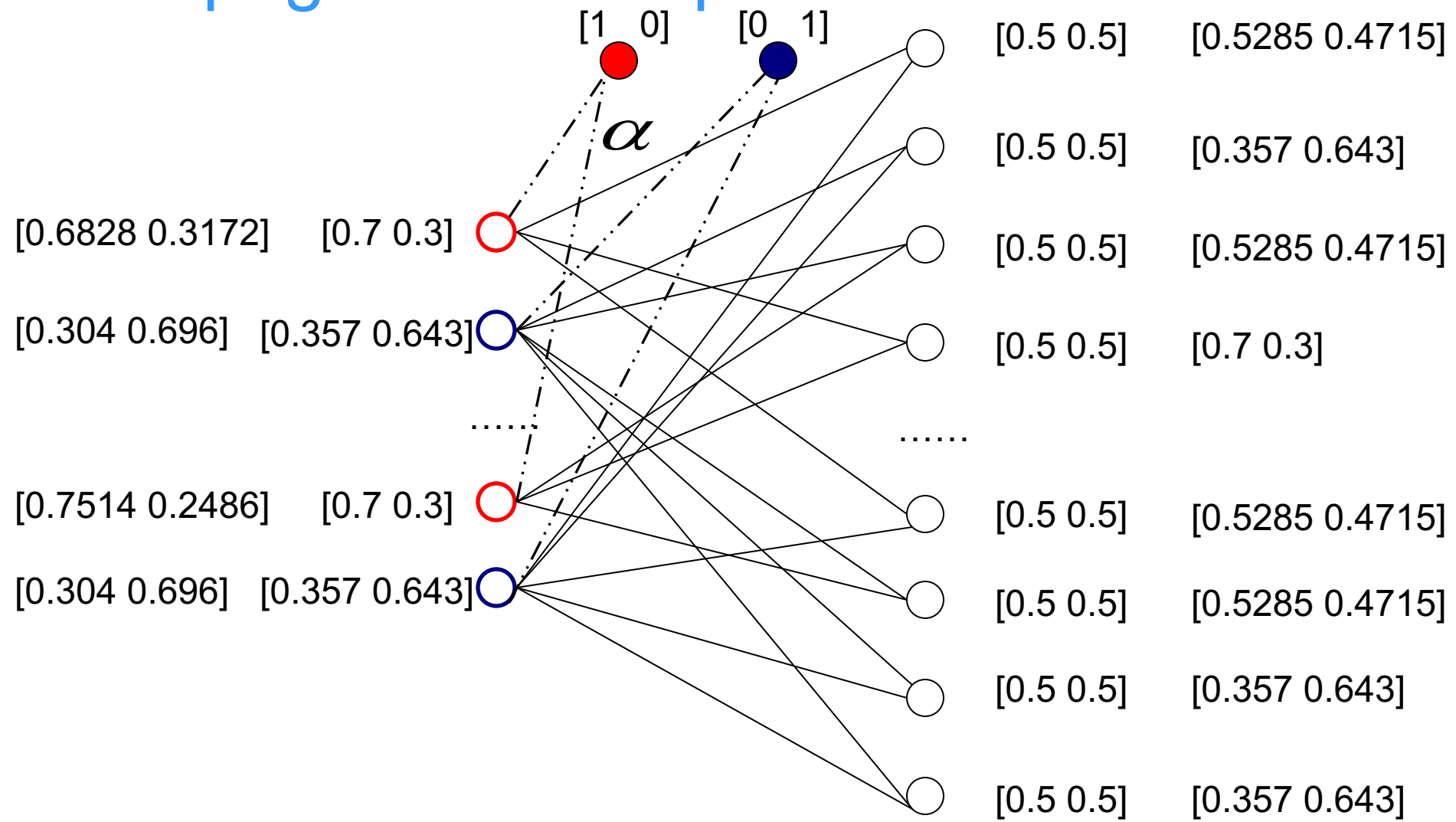
Update detector probability

$$\vec{q}_j = \frac{\sum_{i=1}^{n} a_{ij}\vec{u}_i + \alpha\vec{y}_j}{\sum_{i=1}^{n} a_{ij} + \alpha}$$

Update record probability

$$\vec{u}_i = \frac{\sum_{j=1}^{v} a_{ij}\vec{q}_j}{\sum_{j=1}^{v} a_{ij}}$$

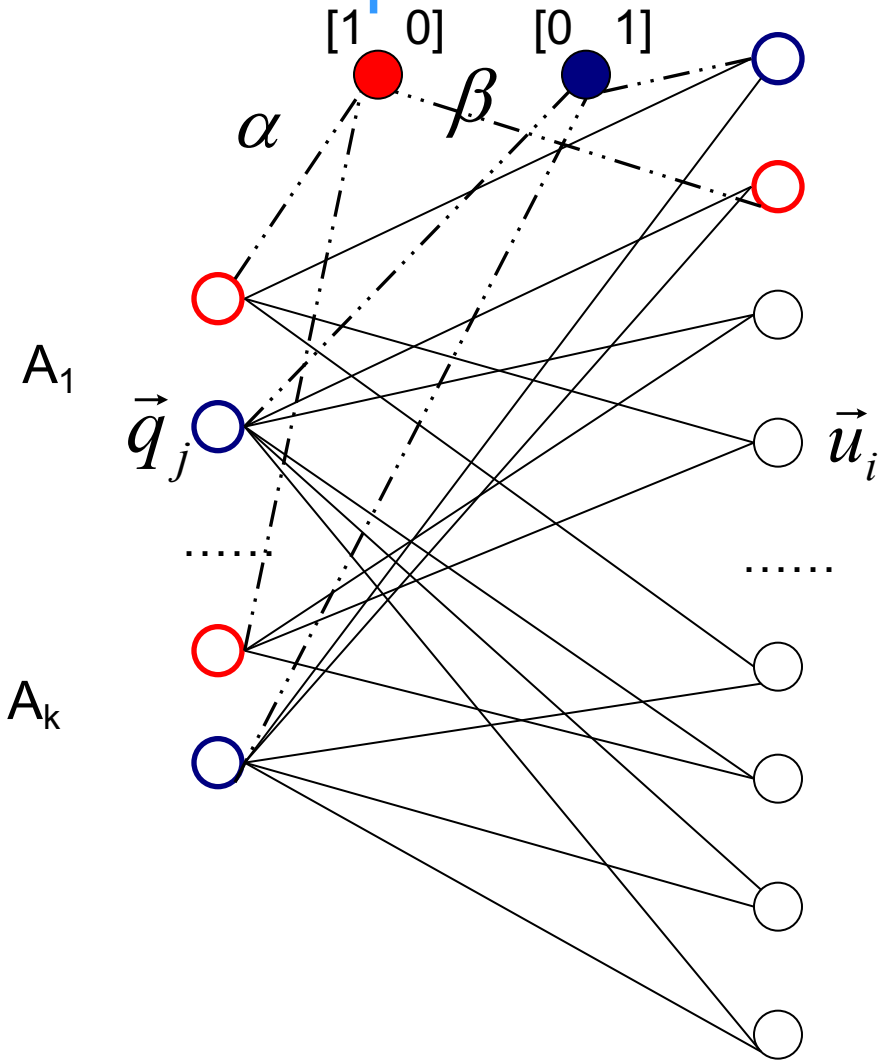
Propagation Example



Extensions

- Semi-supervised
  - Know the labels of a few tuples in advance
  - Improve the performance of combined detectors by incorporating this knowledge
- Active learning
  - At each iteration, identify the tuple with the least confident probability estimates and require its label
  - Incorporate the label of this tuple into the next iteration
- Incremental
  - Tuples arrive continuously
  - Incrementally update the combined probabilities

Semi-Supervised Consensus Maximization



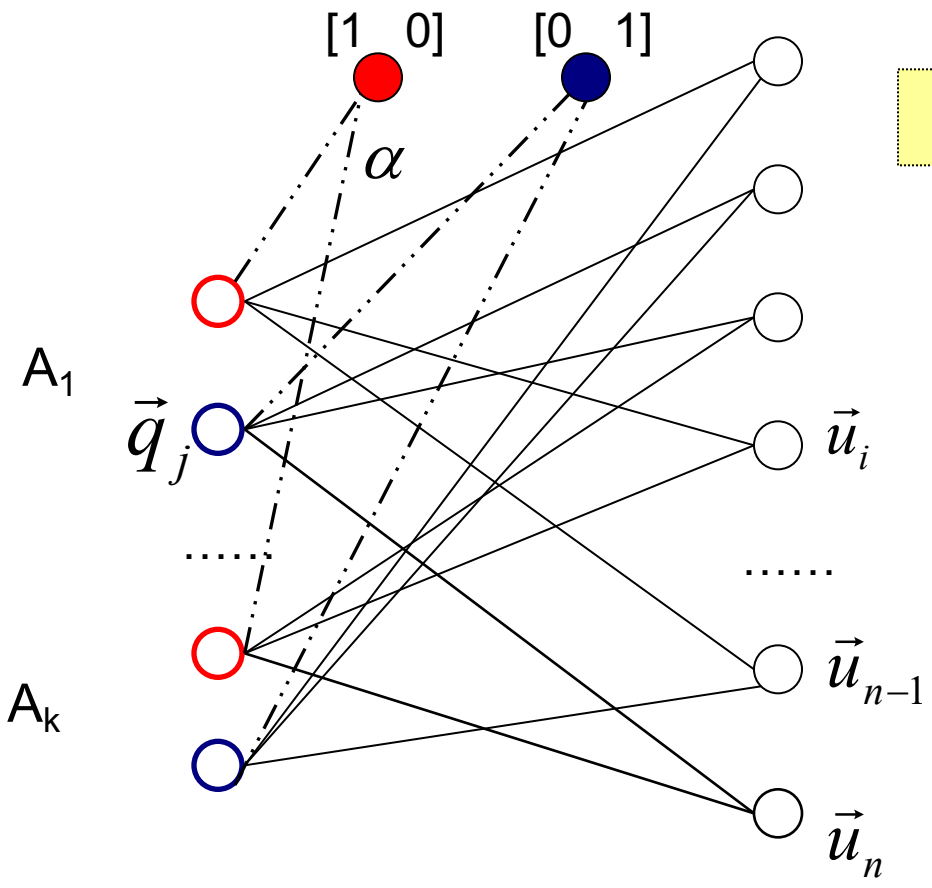
Iterate until convergence

$$\vec{q}_j = \frac{\sum_{i=1}^{n} a_{ij}\vec{u}_i + \alpha\vec{y}_j}{\sum_{i=1}^{n} a_{ij} + \alpha}$$

$$\vec{u}_i = \frac{\sum_{j=1}^{v} a_{ij}\vec{q}_j}{\sum_{j=1}^{v} a_{ij}}$$ unlabeled

$$\vec{u}_i = \frac{\sum_{j=1}^{v} a_{ij}\vec{q}_j + \beta\vec{f}_i}{\sum_{j=1}^{v} a_{ij} + \beta}$$ labeled

Incremental Consensus Maximization



When a new record arrives

Update detector probability

$$\vec{q}_j = \frac{\sum_{i=1}^{n-1} a_{ij}\vec{u}_i + a_{nj}\vec{u}_n + \alpha\vec{y}_j}{\sum_{i=1}^{n-1} a_{ij} + a_{nj} + \alpha}$$

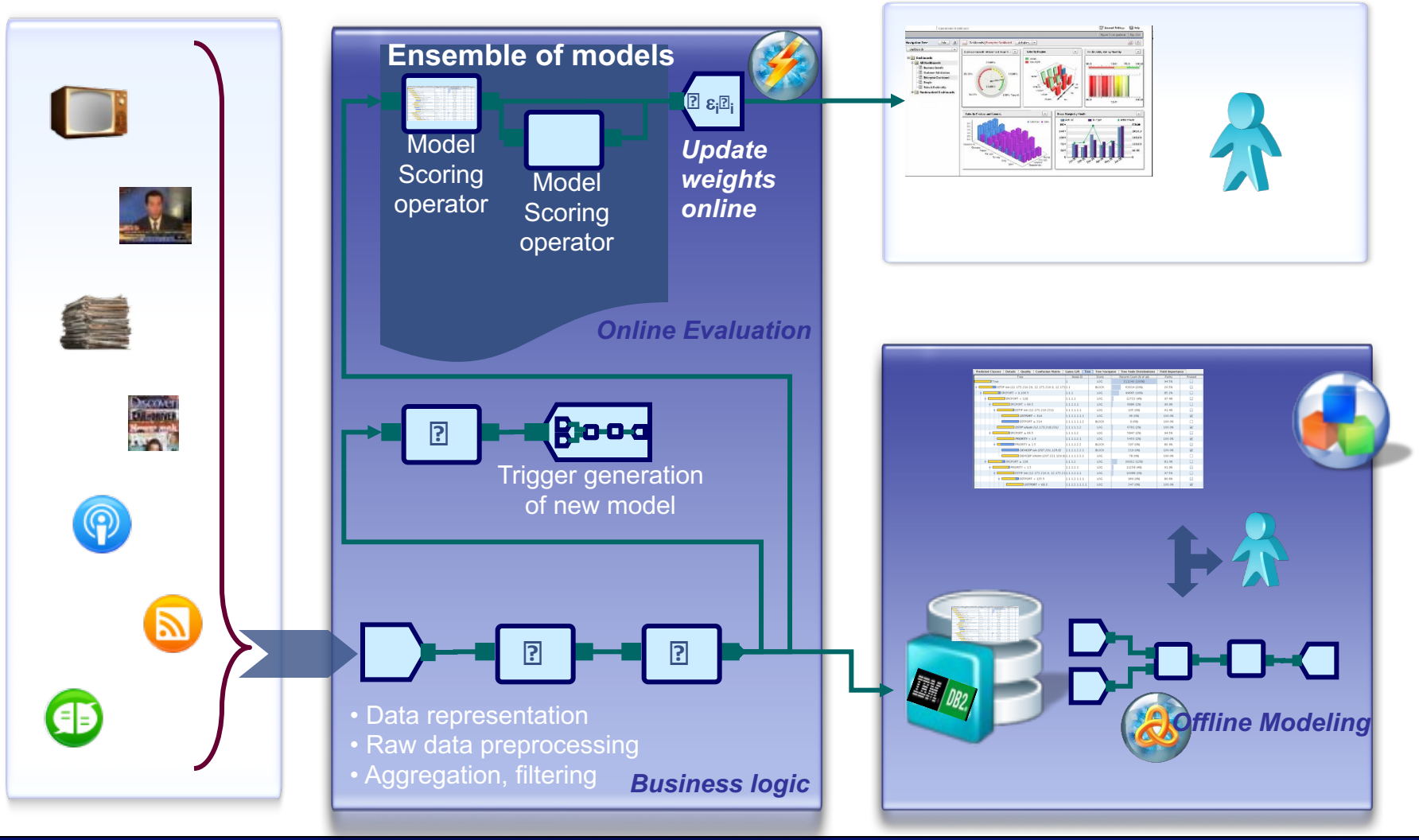
Update record probability

$$\vec{u}_i = \frac{\sum_{j=1}^{v} a_{ij}\vec{q}_j}{\sum_{j=1}^{v} a_{ij}}$$

Advanced Topics

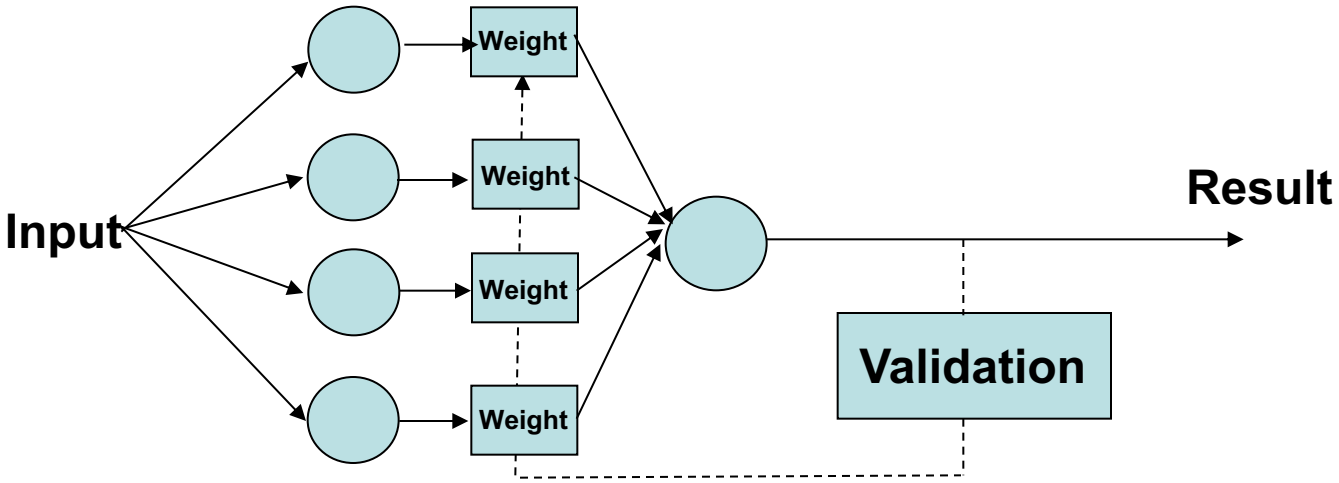
Meta-Learning for Ensembles

From Offline to Online Analysis



Ensemble Learning of Models

- Build collection of models
  - Late Fusion
- Combine them using different weighting approaches
  - Build new classifiers on different time-subsets of data
  - Tune weights for classifiers based on measured accuracy
    - validation set, most recent data with labels etc.

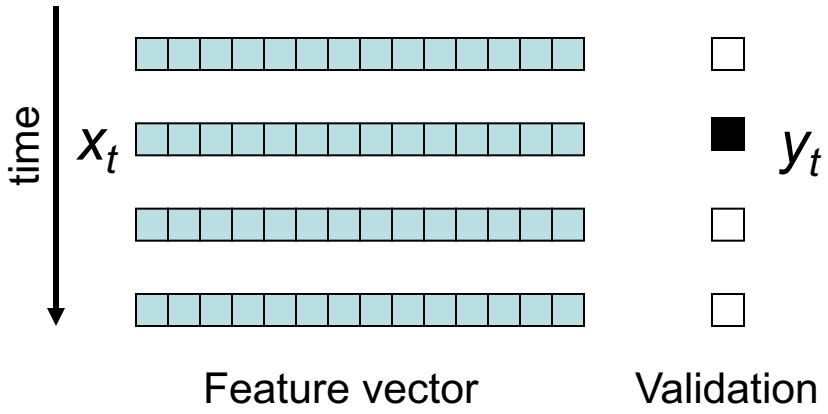


Techniques for Ensemble Learning

- Ensemble learning can use another incremental learner
  - Linear Regression
  - Decision Tree
- In general need to do this under budget constraints in an online setting
  - Exploration problem

Base Learning Problem

Prediction, Classification, Regression

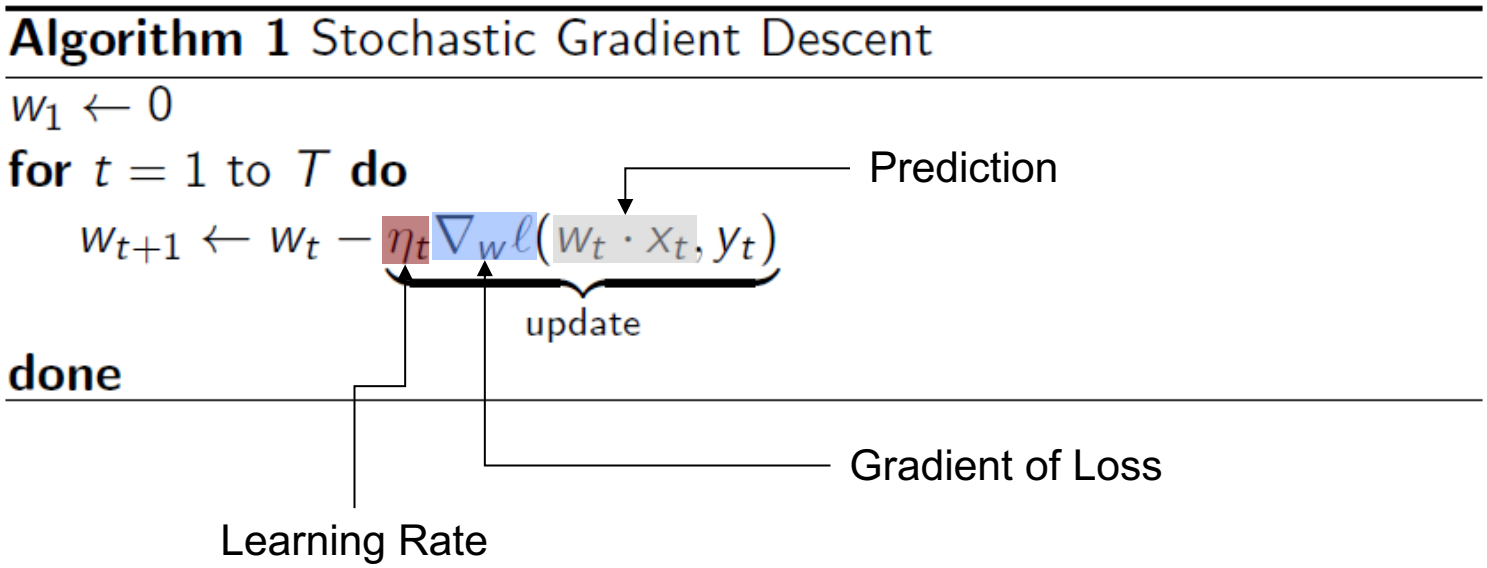


Learn **linear function** that maps feature vector onto validation in an online manner

Linear learning is very scalable and particularly competitive in sparse, high dimensional spaces

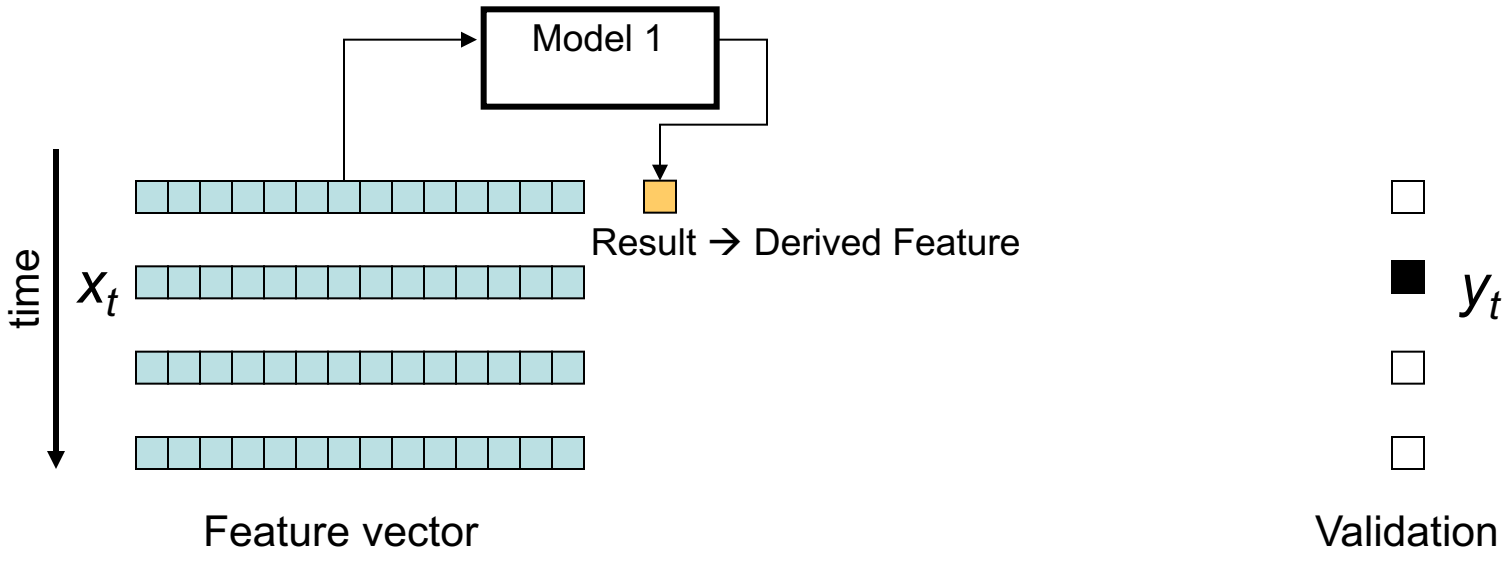
Base Learning Problem

Prediction, Classification, Regression



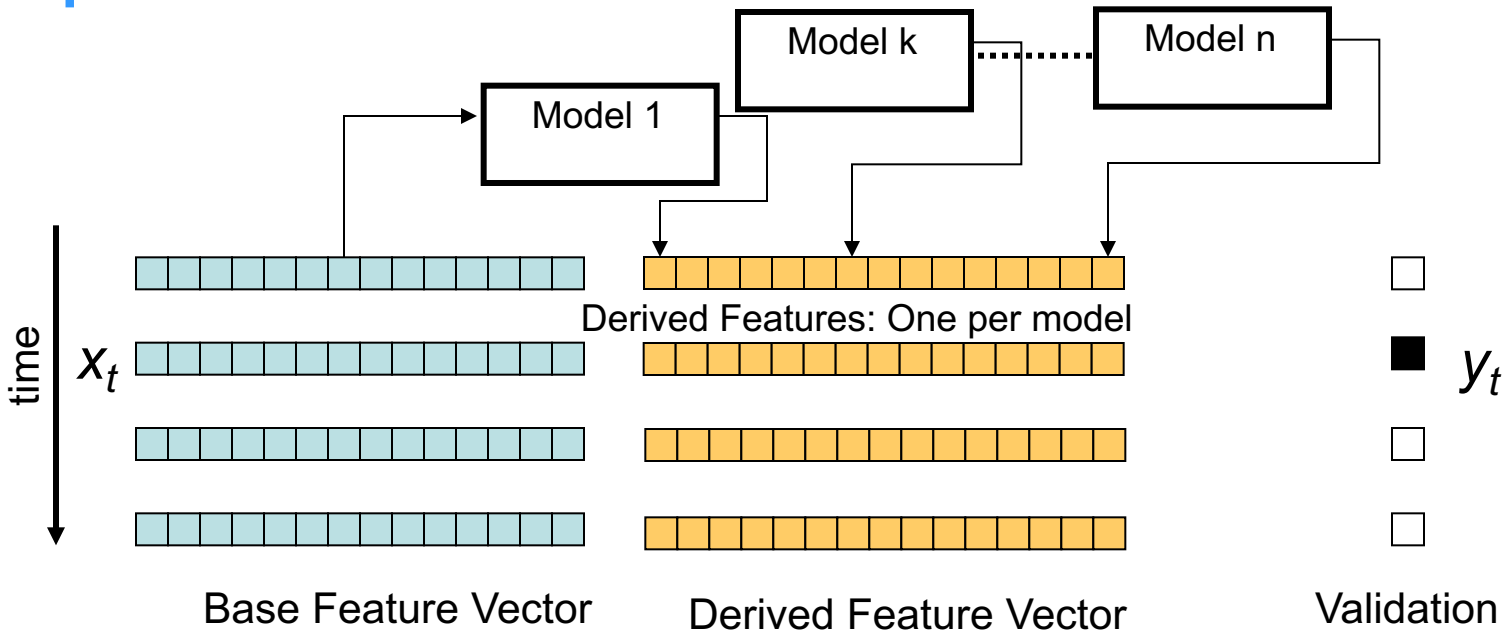
Well understood, and several standard optimizations available

Exploration Problem: First Take



Learn **linear function** that maps combined base feature vector and derived feature vector onto validation in an online manner

Exploration Problem: First Take



Learn **linear function** that maps combined base feature vector and derived feature vector onto validation in an online manner

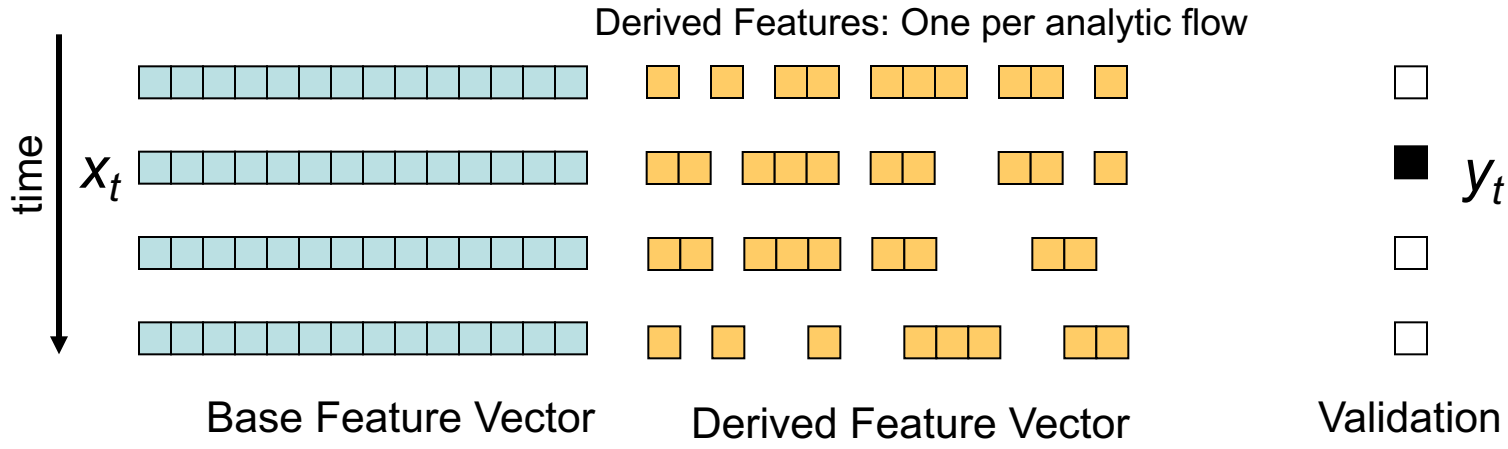
Creating a linearly weighted ensemble of the analytic flows

Exploration Problem: A Closer Look

- Each derived feature is associated with a compute cost
  - Analytic flow processing
- Potential number of derived features may be large
  - Parameter model combinations grow rapidly
- Cannot deploy all models all the time
  - Compute resource constraints in the system
- Need to explore analytic flows to find good combination
  - Not known apriori – may be different over time and for different problems

Exploration Problem: Sampling

Sample different combinations to run to explore analytic space while meeting resource constraints



Limited-resource exploration via careful sampling of models to run
Allows exploration of models without exceeding resources
Can ensure that weights converge to same weights as if all models were run

Online Convex Optimization

- Regression problem: given feature vector $\mathbf{x}$, and response y, predict using linear combination $\mathbf{w}^{\mathrm{T}}\mathbf{x}$, and incur loss $(y - \mathbf{w}^{\mathrm{T}}\mathbf{x})^2$.
- Restriction: cost of feature i is c_i. Total budget B on cost of all features computed for any example.
- Learning = solving the following optimization problem (vector $\mathbf{w}$, subset S of features are the decision variables)

$$\min \sum_t (\sum_{i \in S} w_i x_{t,i} - y_t)^2$$

$$\sum_{i \in S} c_i \leq B$$

$$\forall i : \ |w_i| \leq R$$

Convex Optimization: Solution Outline

- The convex relaxation can be solved using stochastic gradient descent (SGD)
- Main idea is to sample features using the p_i's as probabilities, and unbias the feature values by importance weighing the sampled features by inverse probabilities
- Advantage is that examples can be processed online: i.e. optimization is iterative, using examples one-by-one as they arrive
- Allows adaptation to changing examples over time.

$$\min \sum_t (\sum_i v_i x_{t,i} - y_t)^2$$

$$\sum_i c_i p_i \leq B$$

$$\forall i: \; p_i \in [0,1]$$

$$\forall i: \; |v_i| \leq R p_i$$

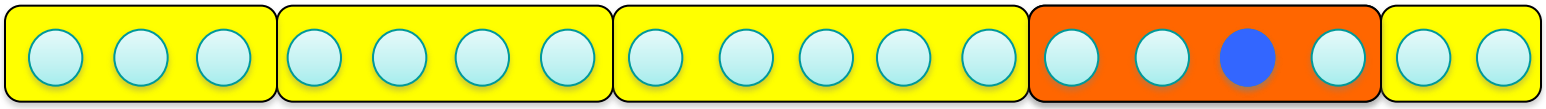
Can use SGD to solve this convex optimization

Budgeted Expert Sampling

- Alternate Formulation: Find single best expert

Group all flows into bins of total cost at most B

Algorithm maintains a weight $w(f)$ for each model f



Bin X chosen with probability ~ total weight of models in X

Model f chosen from X with probability ~ $w(f)$

Run all models in X, predict according to f (i.e. predict $f(t)$)

Update weights of flows in X using standard multiplicative update rule with importance weighting
Provable performance bounds!

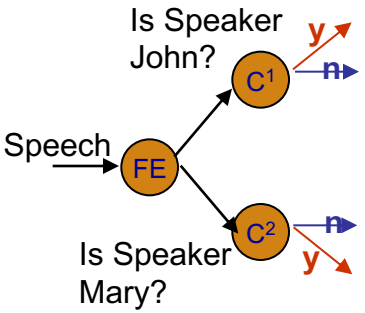
Issues to Consider

- Switching Cost
- *What about non-linear meta-learners?*

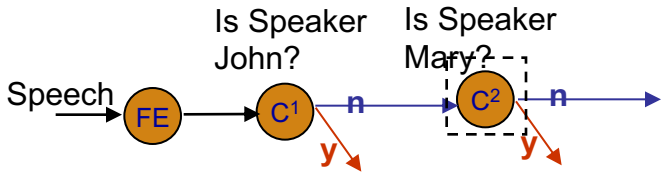
Topologies and Compositions of Algorithms

- Can we construct and optimize topologies of classifiers in a stream processing setting?

Classifier Topology for Stream Mining



70% of speech from John, 20% from Mary, 10% unknown



"Mary" detector needs to process ~30% of data on average

Multi-concept (binary) detection

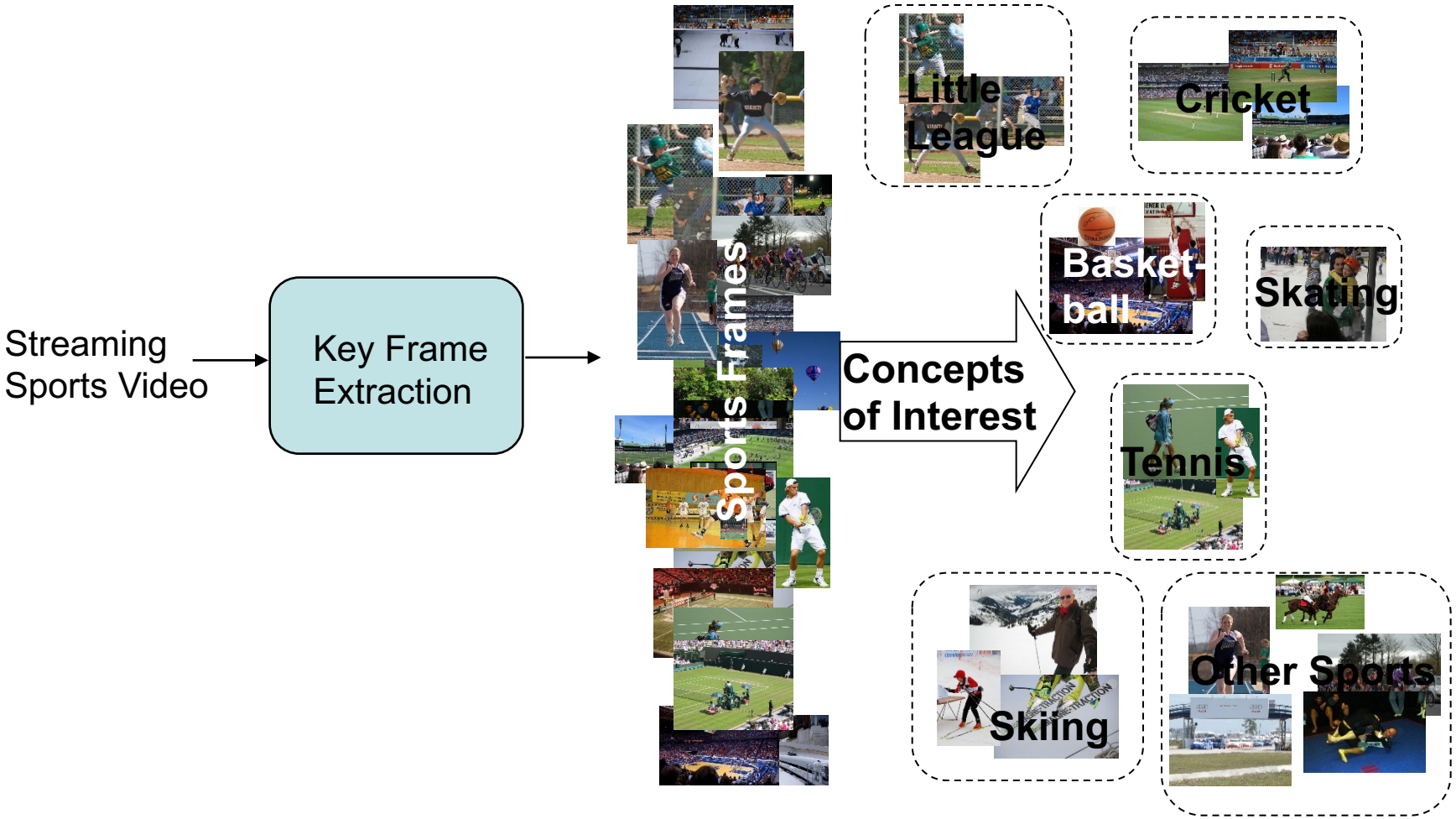
Hierarchical Multi-concept (binary) detection

Significant Advantage in Resource Constrained Scenarios

Hierarchy can account for data density skew ⇒ limit unnecessary data processing by downstream classifiers

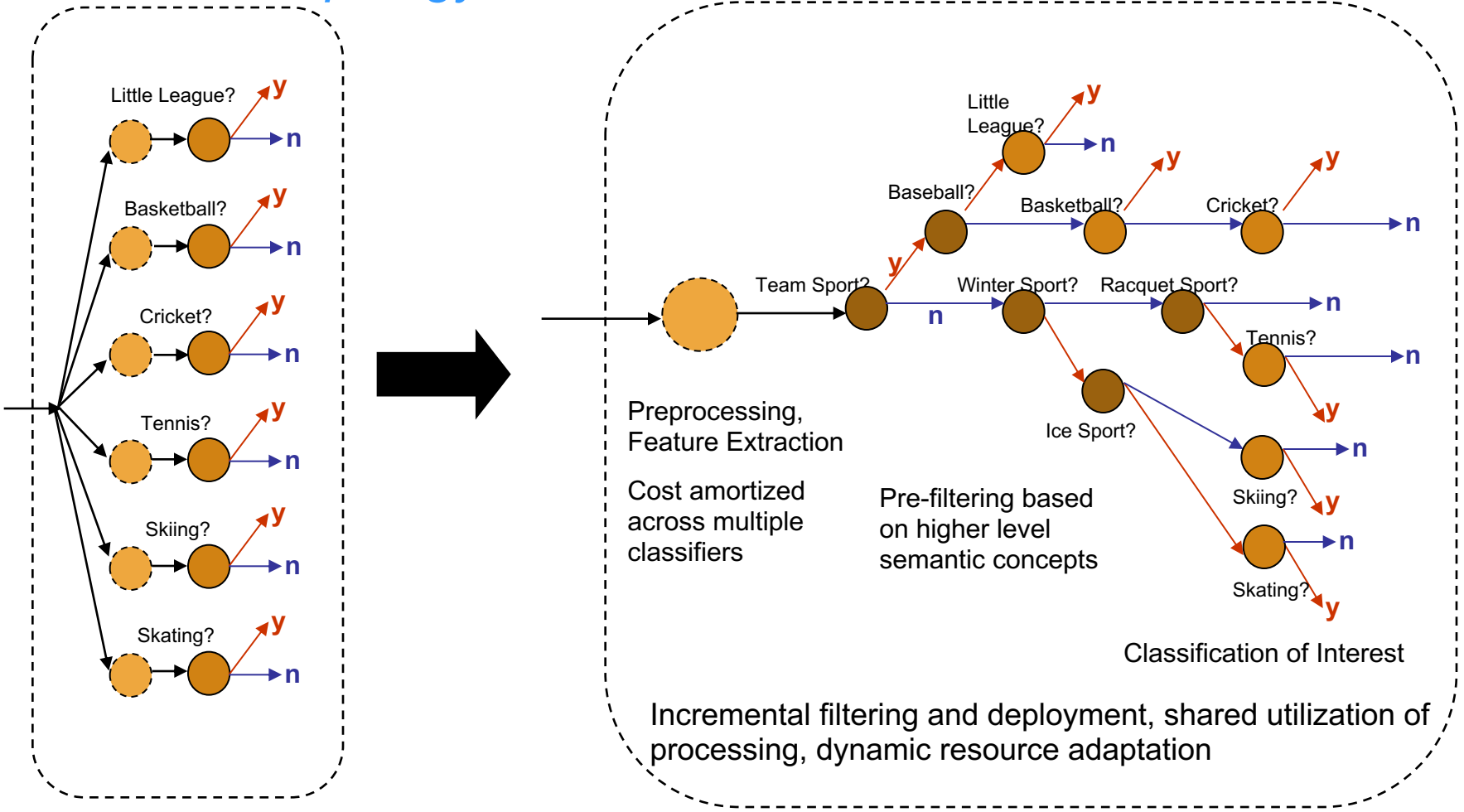
Improve Classification Accuracy

Example: Semantic Concept Detection



Constructing Classifier Topology

Semantic Topology

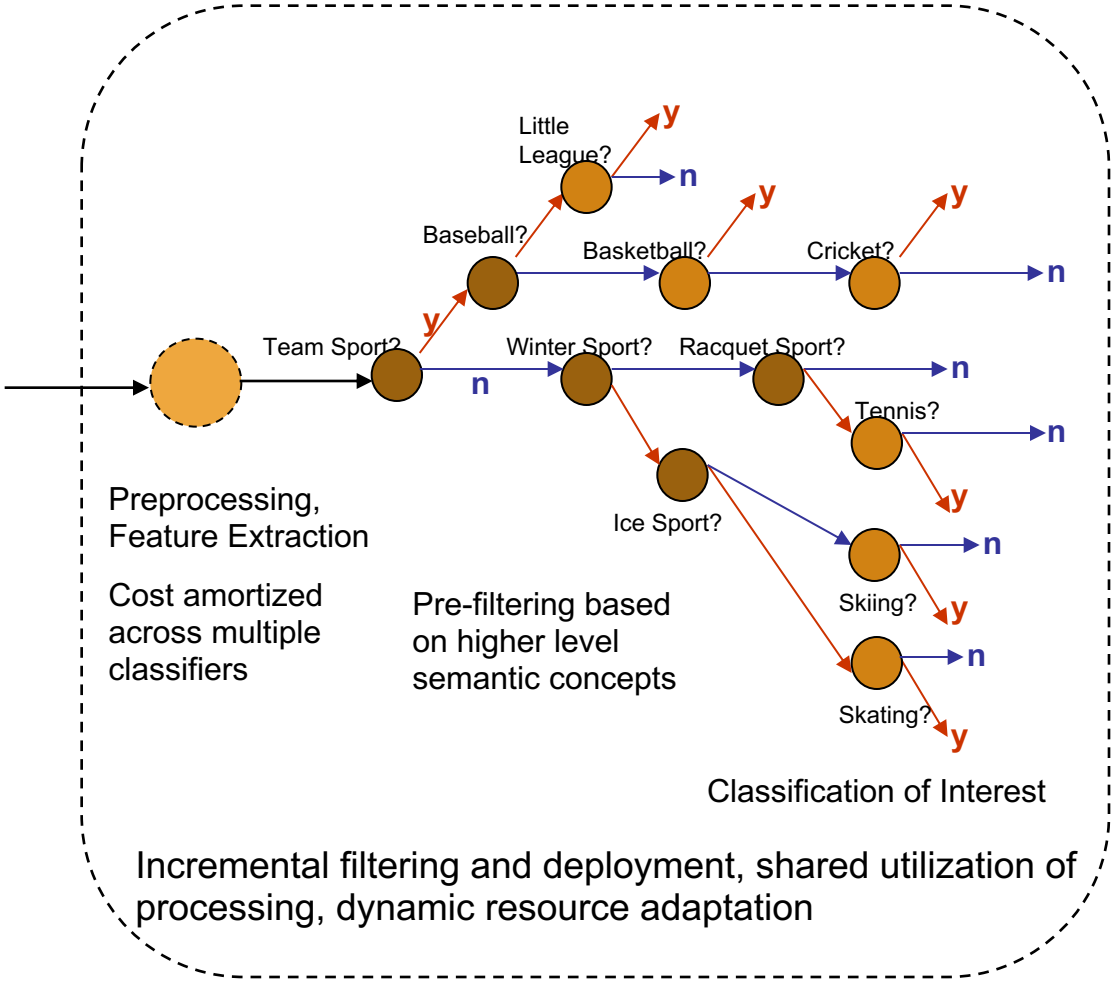


Classifier Topology and Implementation

IMARS Analytics/Classifiers

Classifiers use SVMs trained individually over 20000 sports key frames.

Experimental tree consists of 11 classifiers with a semantic hierarchy.

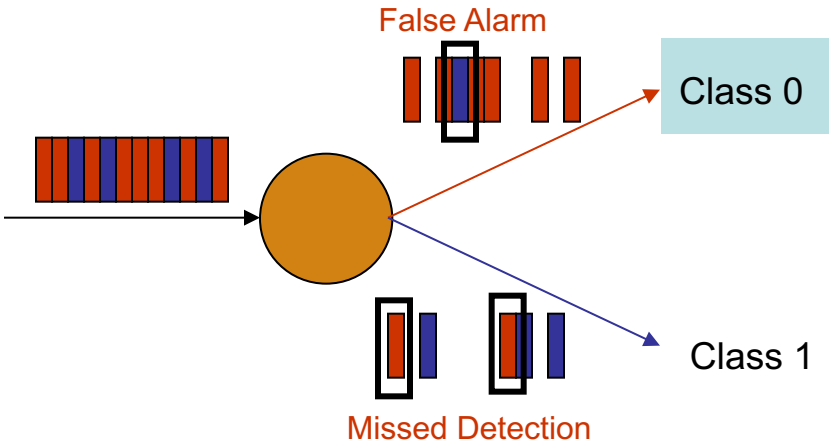


Classifier Topology for Stream Mining

- Topology Construction
  - How can such a topology be built automatically?
- Topology Configuration
  - How can the topology be configured?
  - Central or Distributed Optimization
  - Multi-agent Learning

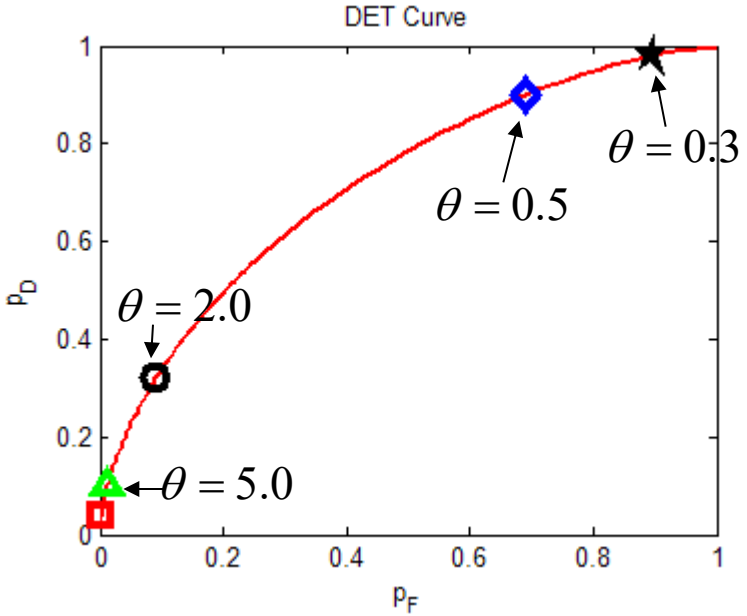
- *In resource constrained scenarios…*

Binary Classifier Model



Filters Data into Two Classes

Classifier Accuracy Characterization
DET curve i.e. p_D versus p_F curve



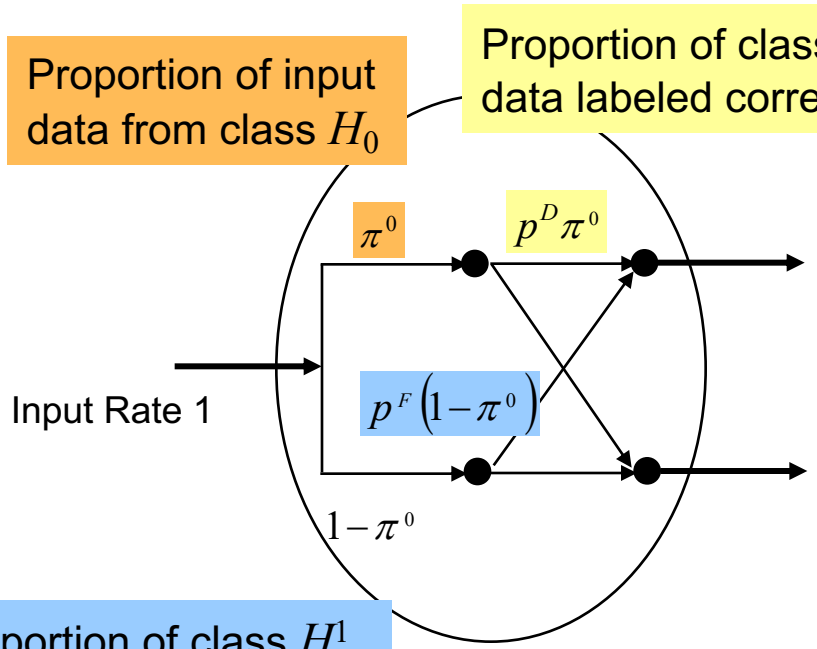
Operating point on curve determined by desired tradeoff between p^D and p^F

Definitions

- Throughput (t)
  - Total rate output by classifier
- Goodput (g)
  - Correctly classified portion of throughput
- Apriori Probabilities
  - Unconditional probability (π) of data belonging to class H^0 or H^1 for a classifier
  - Conditional probability (ϕ) given filtering by previous classifier

Binary Classifier/Filter Model

Data from two classes: H^0 and $H^1 \rightarrow$ Consider data **labeled** as class H^0



Throughput $t = \left[\pi^0 p^D + (1-\pi^0)p^F\right]$

Goodput $g = \pi^0 p^D$

Can similarly define throughput $\bar{t}$ and goodput $\bar{g}$

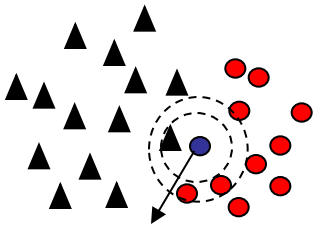
Selected operating point/s control throughput and goodput

Binary Classifier Model

Classifier complexity $\rightarrow$ Underlying Model Complexity $\rightarrow$ Test complexity
measured in terms of processing (CPU and/or Memory) resource requirements

Processing resource consumption r proportional to input rate ρ

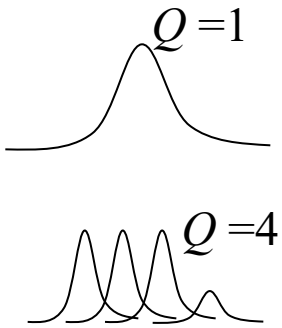
Non-parametric: k -Nearest Neighbor



Test Sample

Test Complexity
$r \propto k\rho$

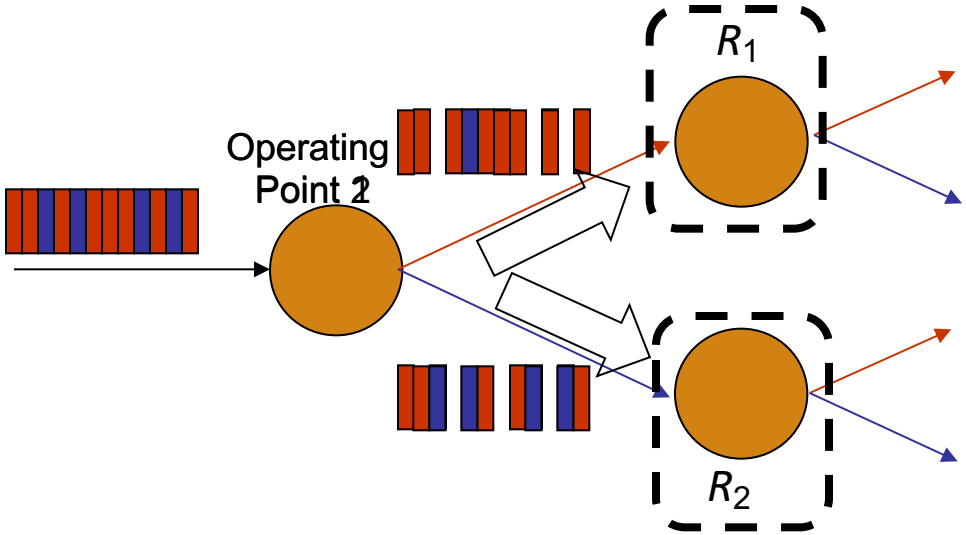
Parametric: Gaussian Mixture Model



Test Complexity
$r \propto Q\rho$

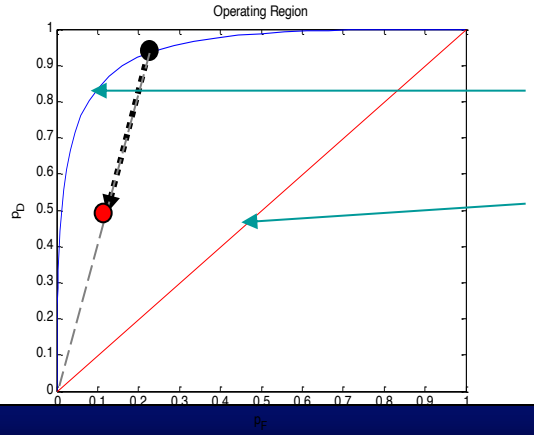
Classifier Resource Consumption Model: $r = \alpha\rho$

Topology of Classifiers



It may not be feasible to meet tight resource constraints $\Rightarrow$ **Arbitrary Load Shedding at next classifier**

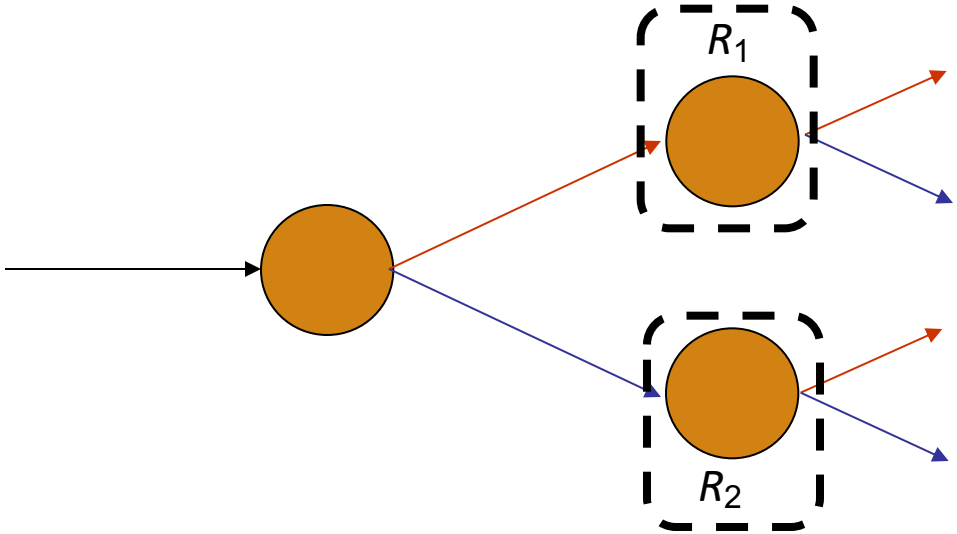
Operating point controls rate flowing through (resource consumption of) each successive classifier



DET curve

Random data forwarding curve

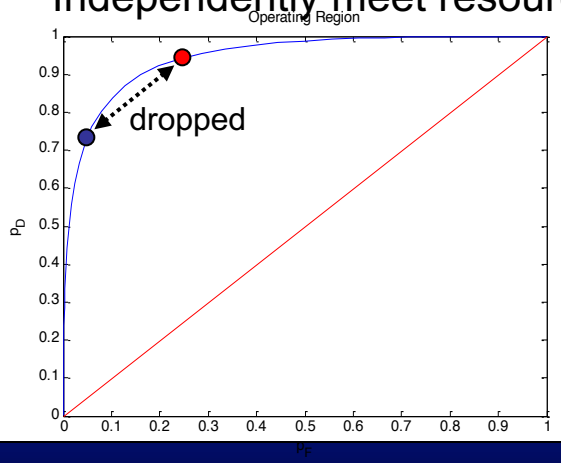
Topology of Classifiers



Any load shedding at current classifier ⇒ **Intelligent Load Shedding**

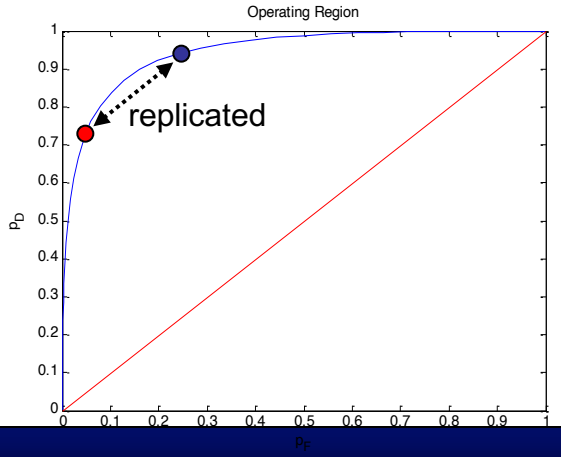
Also support **Replication** when resources are available

Decouple operating points for positive and negative branches
Independently meet resource constraints of downstream classifiers



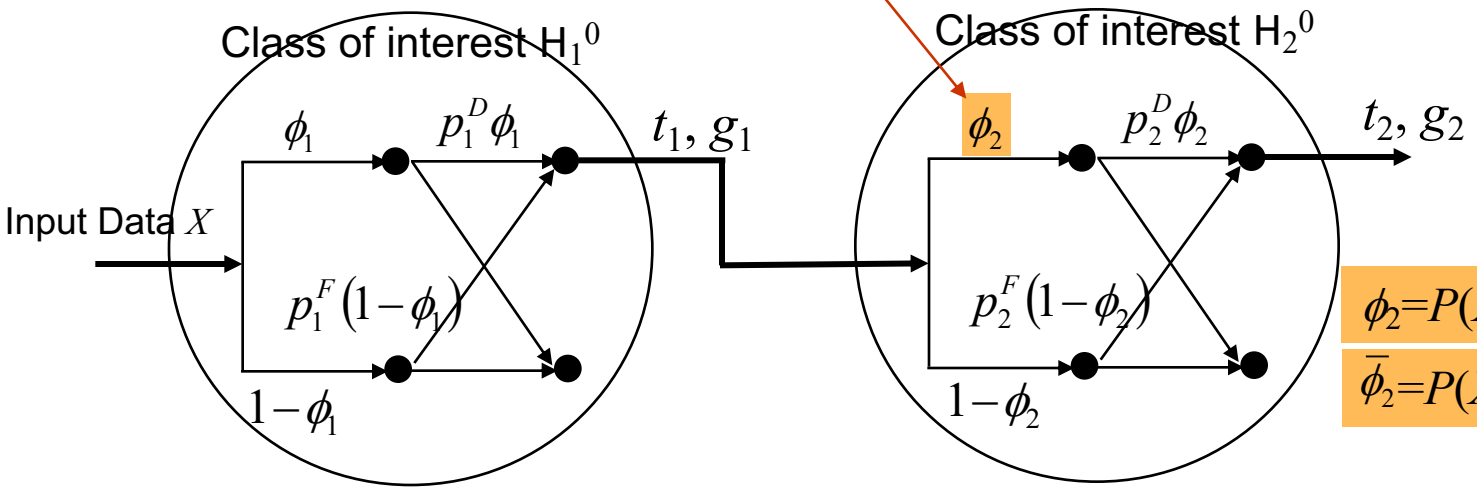
Blue = positive OP

Red = negative OP



Cascade of Binary Classifiers

Conditional probability of data belonging to class H_2^0 given it belongs to class H_1^0



$\phi_2 = P(X \in H_2^0 | X \in H_1^0)$

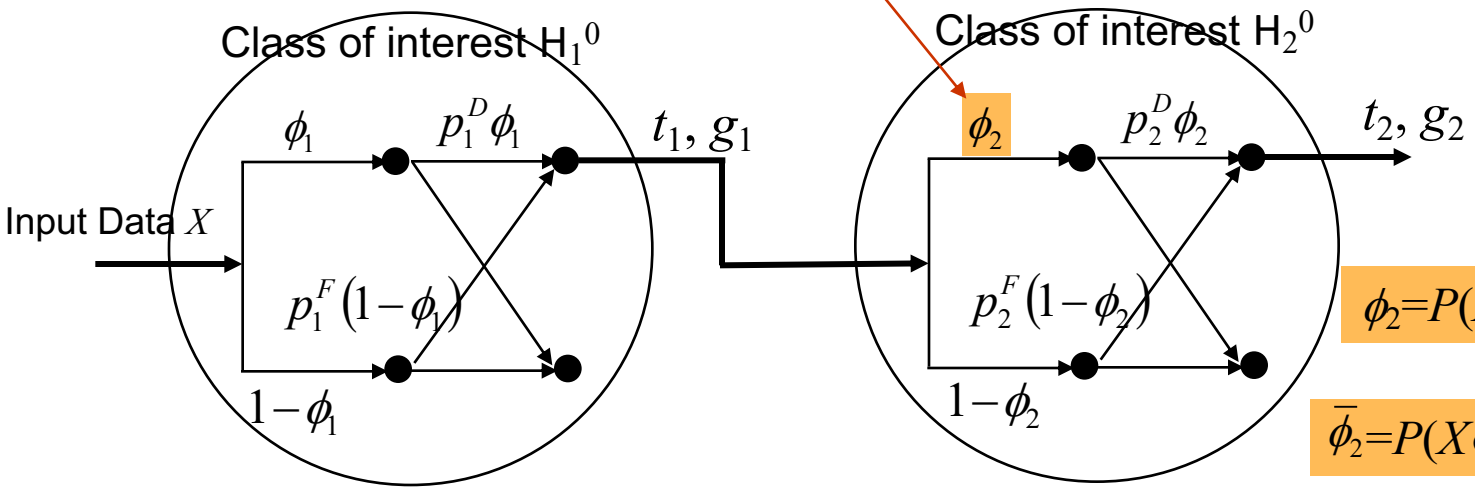
$\bar{\phi}_2 = P(X \in H_2^0 | X \notin H_1^0)$

$$\begin{bmatrix} t_2 \\ g_2 \end{bmatrix} = \begin{bmatrix} p_2^F + \bar{\phi}_2(p_2^D - p_2^F) & (\phi_2 - \bar{\phi}_2)(p_2^D - p_2^F) \\ 0 & \phi_2 p_2^D \end{bmatrix} \begin{bmatrix} t_1 \\ g_1 \end{bmatrix}$$

Recursive relationship in throughput and goodput among classifiers

Cascade of *Exclusive* Binary Classifiers

Conditional probability of data belonging to class H_2^0 given it belongs to class H_1^0



$\phi_2 = P(X \in H_2^0 | X \in H_1^0)$

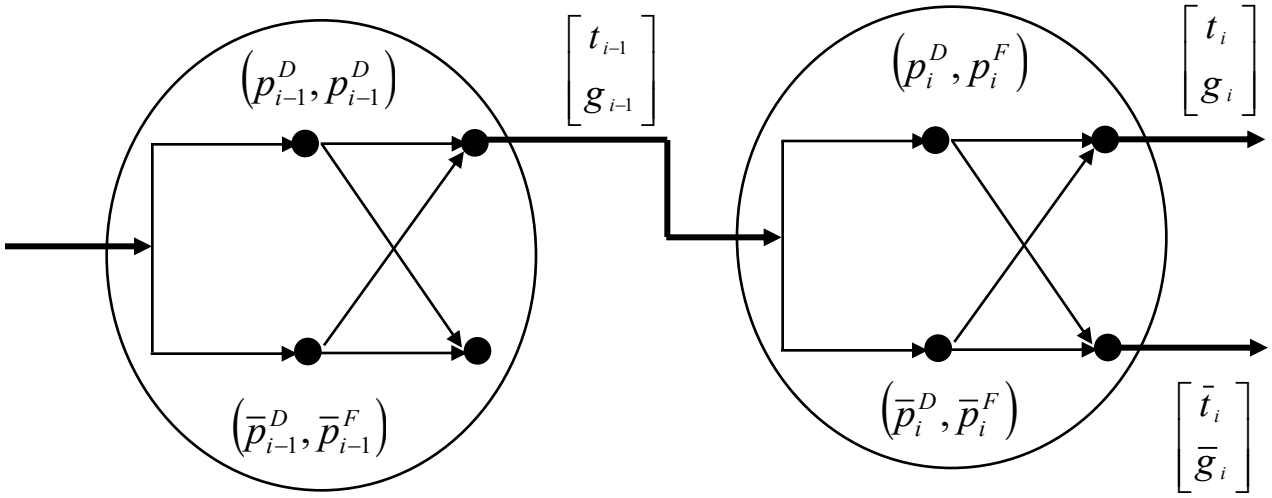
$\bar{\phi}_2 = P(X \in H_2^0 | X \notin H_1^0) = 0$

$$\begin{bmatrix} t_2 \\ g_2 \end{bmatrix} = \begin{bmatrix} p_2^F & \phi_2 (p_2^D - p_2^F) \\ 0 & \phi_2 p_2^D \end{bmatrix} \begin{bmatrix} t_1 \\ g_1 \end{bmatrix}$$

Exclusivity $\Rightarrow$ what is not of interest to parent, is not of interest to child classifier

Recursive relationship in throughput and goodput among classifiers

Cascade of *Exclusive* Binary Classifiers



$$\begin{bmatrix} t_i \\ g_i \end{bmatrix} = \begin{bmatrix} p_i^F & \phi_i\left(p_i^D - p_i^F\right) \\ 0 & \phi_i\, p_i^D \end{bmatrix} \begin{bmatrix} t_{i-1} \\ g_{i-1} \end{bmatrix}$$

where $\phi_i = P\left(X \in H_i^0 \mid X \in H_{i-1}^0\right)$

$$\begin{bmatrix} \bar{t}_i \\ \bar{g}_i \end{bmatrix} = \begin{bmatrix} \bar{p}_i^D & \phi_i\left(\bar{p}_i^F - \bar{p}_i^D\right) \\ 0 & \bar{\phi}_i\, \bar{p}_i^D \end{bmatrix} \begin{bmatrix} t_{i-1} \\ g_{i-1} \end{bmatrix}$$

where $\bar{\phi}_i = 1 - \phi_i$

Classification Accuracy and Penalty

- Consider a classifier C_k in terminal set T
  - False-Alarm Rate: t_k-g_k
  - Missed Detection rate: $t_0\pi_k$-g_k
    - t_0: Initial throughput, π_k:Apriori probability
- Misclassification Penalty Coefficients
  - c^M: Missed detection penalty per unit rate
  - c^F: False alarm penalty per unit rate
- End-to-end misclassification penalty

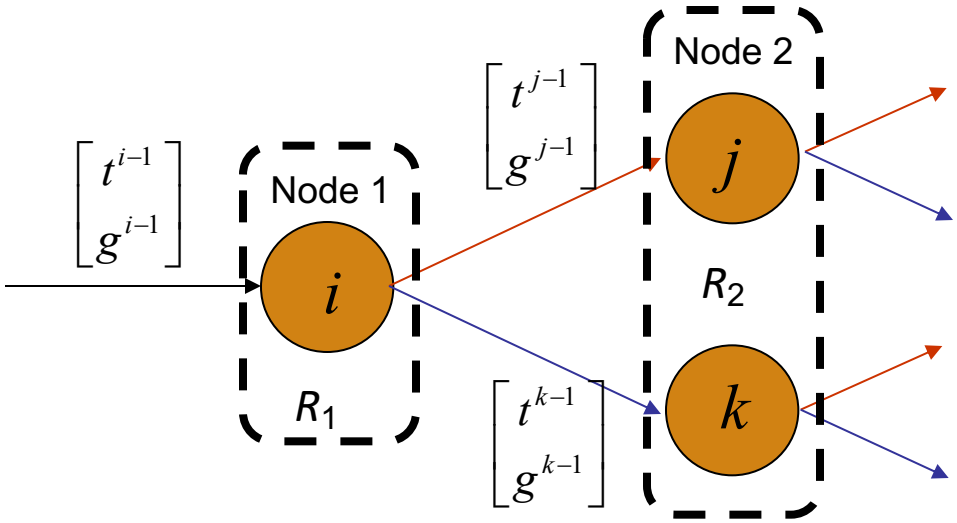
$$u = \sum_{C_k \in} c_k^F \left(t_k - g_k\right) + c_k^M \left(t_0 \pi_k - g_k\right)$$

Delay and Resource Consumption

- Resource consumption for classifier C_k
  - $\alpha_k t_{k\text{-}1}$
- End-to-end resource consumption for topology

$$\sum_{C_k} \alpha_k t_{k-1}$$

Placement and Resource Constraints



$$r_i = \alpha_i t_{i-1} \leq \quad 1$$

$$r_j + r_k = \alpha_j t_{j-1} + \alpha_k t_{k-1} \leq \quad 2$$

Can write out combined resource constraints as:

Placement matrix **A**

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_i t_{i-1} \\ \alpha_j t_{j-1} \\ \alpha_k t_{k-1} \end{bmatrix} \leq \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Finding Optimized Operating Points

- Constrained optimization

Minimize

$$u = \sum_{C_k \in} c_k^F \left(t_k - g_k \right) + c_k^M \left(t_0 \pi_k - g_k \right)$$

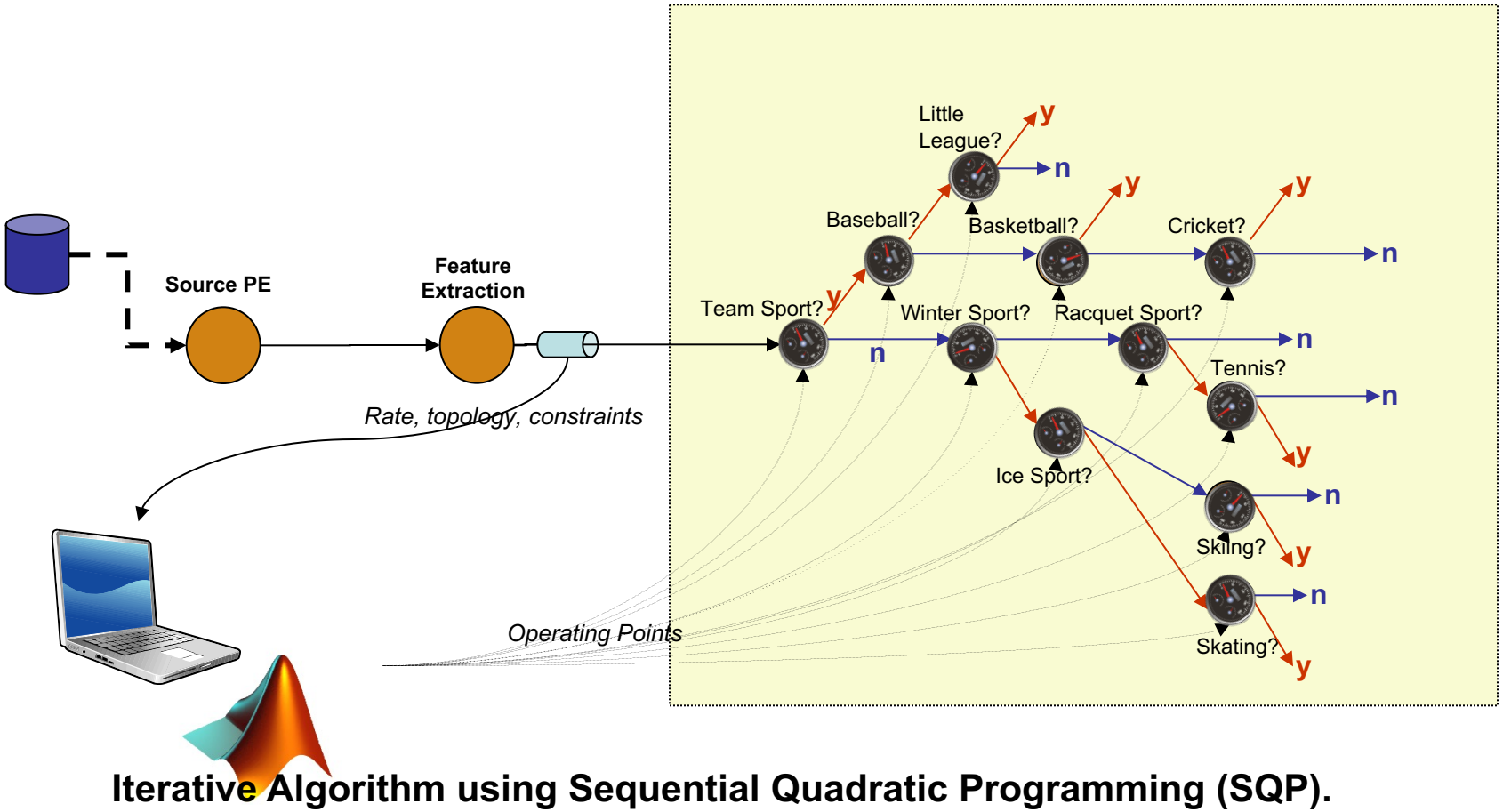
subject to

Can write out combined resource constraints as:

Placement matrix **A**

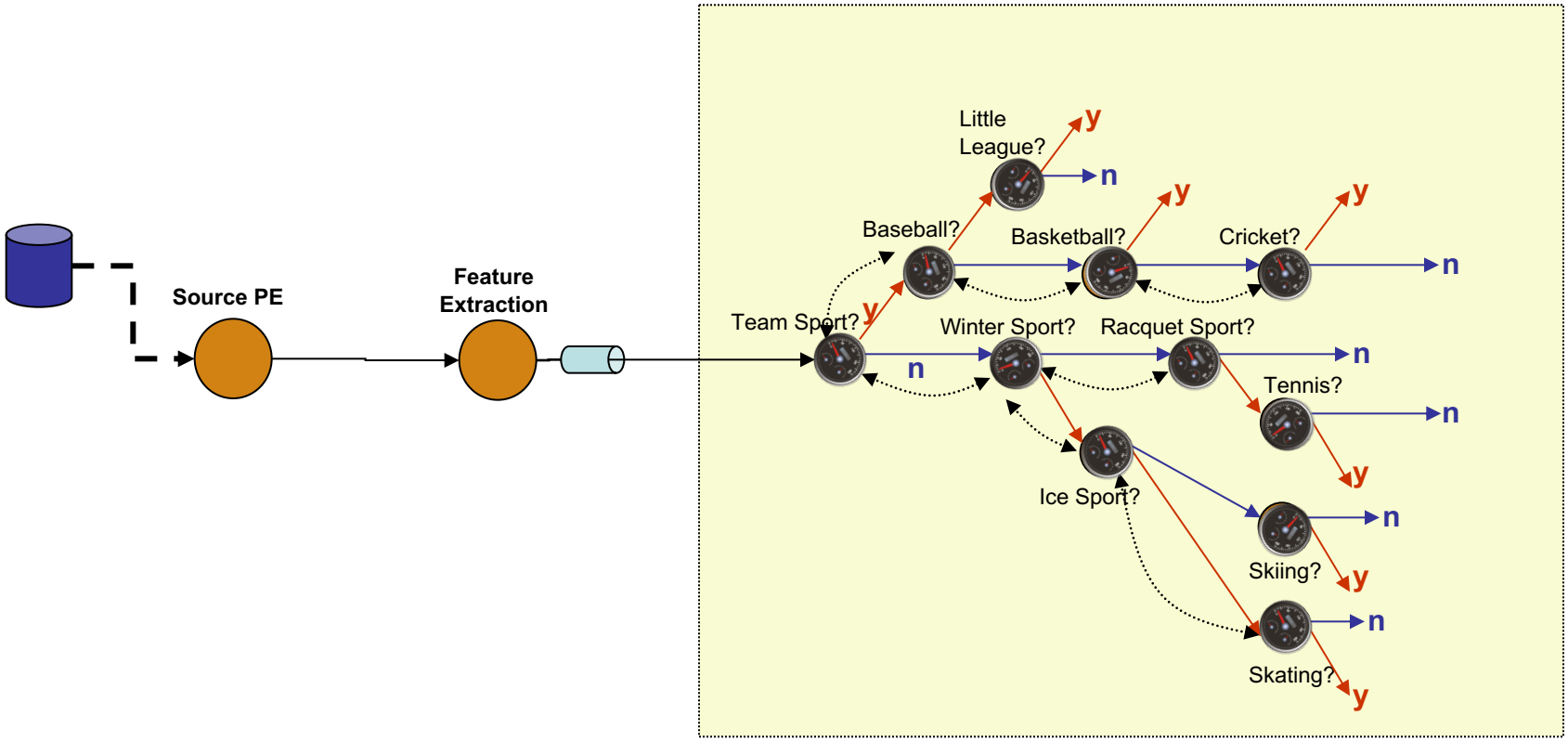
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_i t_{i-1} \\ \alpha_j t_{j-1} \\ \alpha_k t_{k-1} \end{bmatrix} \leq \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Centralized Solution: Outline



Iterative Algorithm using Sequential Quadratic Programming (SQP).
Guaranteed to converge to a local optimum
Run multiple times with different starting points for high probability of finding global optimum.

Distributed Solution: Outline



Individual classifiers exchange information Classifiers reconfigure themselves

Based on Lagrangian Dual Formulation

Individual classifier reconfiguration *always* increases end-to-end utility

Classifier Topology for Stream Mining

- Multiple Optimizations can be designed
  - Topology Construction
  - Topology Configuration
- Centralized and Distributed Solutions Possible
- Game-theoretic Formulations
- Multi-agent Learning

Summary

- Intersection of Meta-learning and Optimization
  - Several interesting open problems