Streaming Algorithms II
Modeling

Recap: Outline of Data Mining Process

- Data acquisition
  - Collect data from external sources
- Data pre-processing
  - Prepare data for further analysis: data cleaning, data interpolation (missing data), data normalization (heterogeneous sources), temporal alignment and data formatting, data reduction
- Data transformation
  - Select an appropriate representation for the data
  - Select or compute the relevant features (*feature extraction/selection*)
- Modeling (data mining)
  - Identify interesting patterns, similarity and groupings, partition into classes, fit functions, find dependencies and correlations, identifying abnormal data
- Evaluation
  - Use of the mining model and evaluation of the results

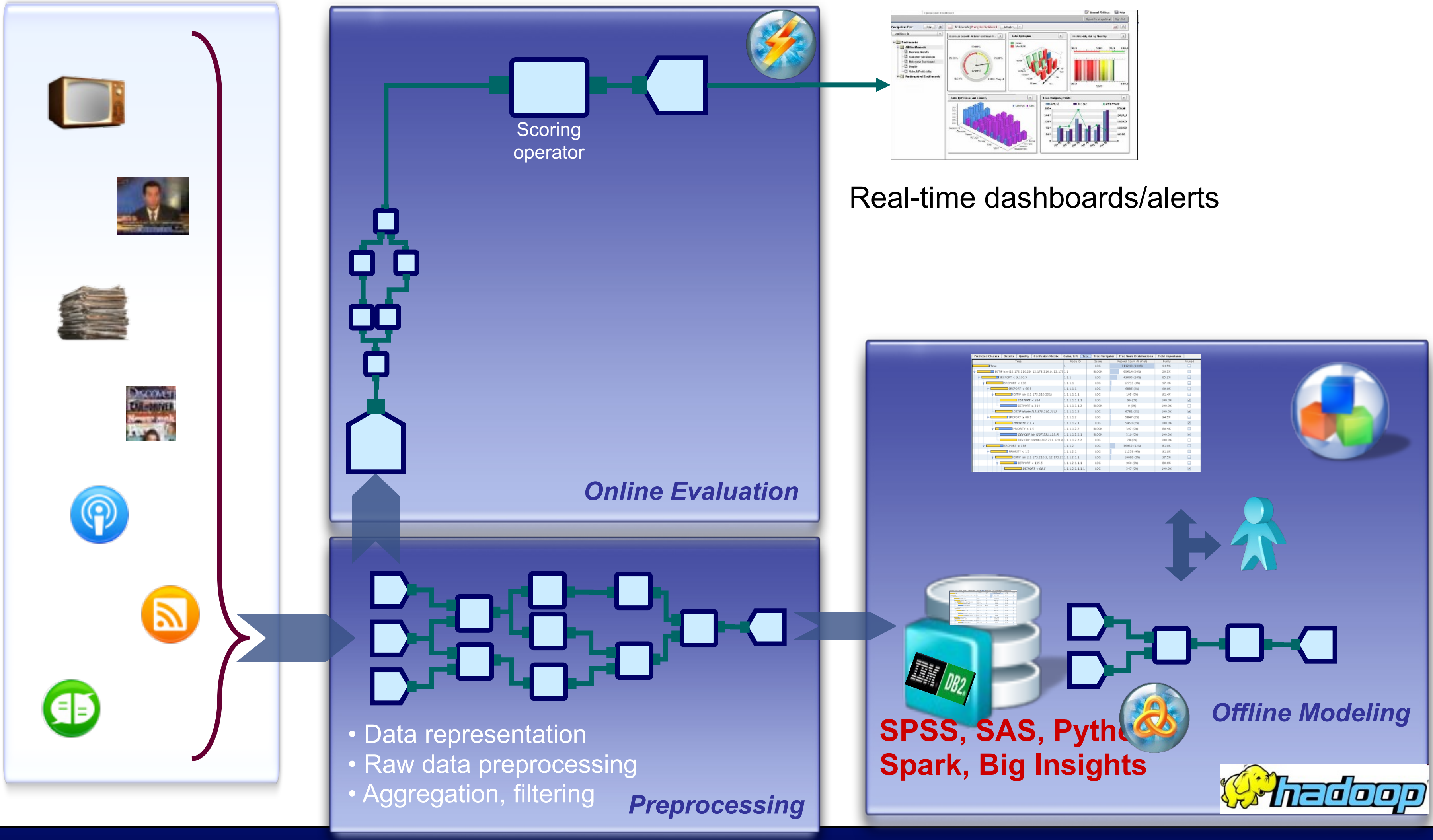
Modeling and Evaluation

- Modeling (model learning)
  - the process of applying knowledge discovery algorithms to the data to extract patterns and trends of interest, resulting in one or more *model*
  - Example: Clustering
    - data: grades of students
    - algorithm: *k*-means
    - model: k groups with mean grades
  - Usually the model is small compared to the data
- Two kinds
  - Supervised: training data is labeled (e.g., classification, regression)
  - Unsupervised: otherwise (e.g., clustering, frequent pattern mining)
- Various tools and libraries exist for modeling
  - Python, Spark, Matlab, R, SPSS, SAS, Weka

Modeling and Evaluation

- Model scoring (evaluation)
  - using the generated model on new data items, to predict what patterns they match
  - Example: Given a new grade, determine which category it belongs
- Model scoring is usually fast
- Model scoring is naturally a stream processing task

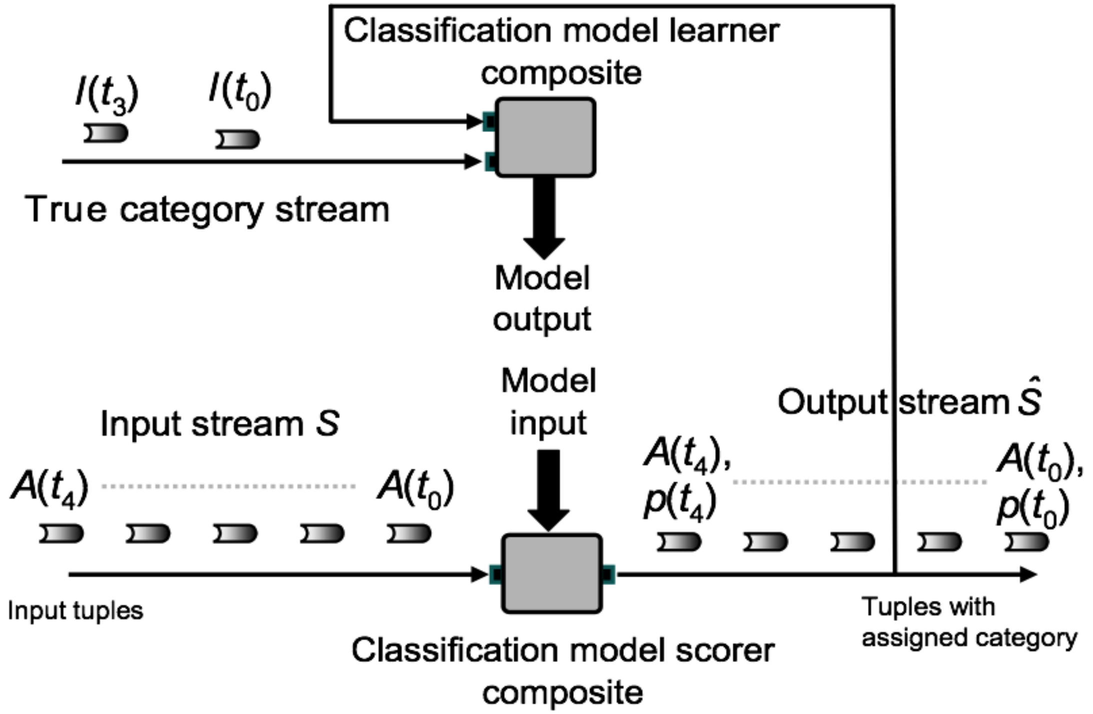
Offline Modeling and Online Evaluation



Data Stream Classification

- Assign one or more categories to each tuple
- Example scenario
  - Tuples: Credit card transactions
  - Categories: Fraudulent or not
- Modeling:
  - Use a training data set that is labeled (for each transaction, there is a label which specifies its category)
  - Build a classification model on the training data, online

Online Classification



- In an online setup, the labels will often come with some delay
  - Human in the loop
- As such, the model learner needs to do buffering

Some Basics on Measurement

- Accuracy: How accurate are our predictions of the category?
  - False positives: We predict that a transaction is fraudulent, but it is not
  - False negatives: We predict that a transaction is not fraudulent, but it is
- This can be represented in a confusion matrix:

	Predicted *fraudulent*	**Predicted *legitimate***
True *fraudulent*	N_{TP}	N_{FN}
True *legitimate*	N_{FP}	N_{TN}

Precision: Fraction of positive predictions that are true positives

$$prec = \frac{N_{TP}}{N_{TP}+N_{FP}}$$

Recall: Fraction of true positives that are predicted as positive

$$rec = \frac{N_{TP}}{N_{TP}+N_{FN}}$$

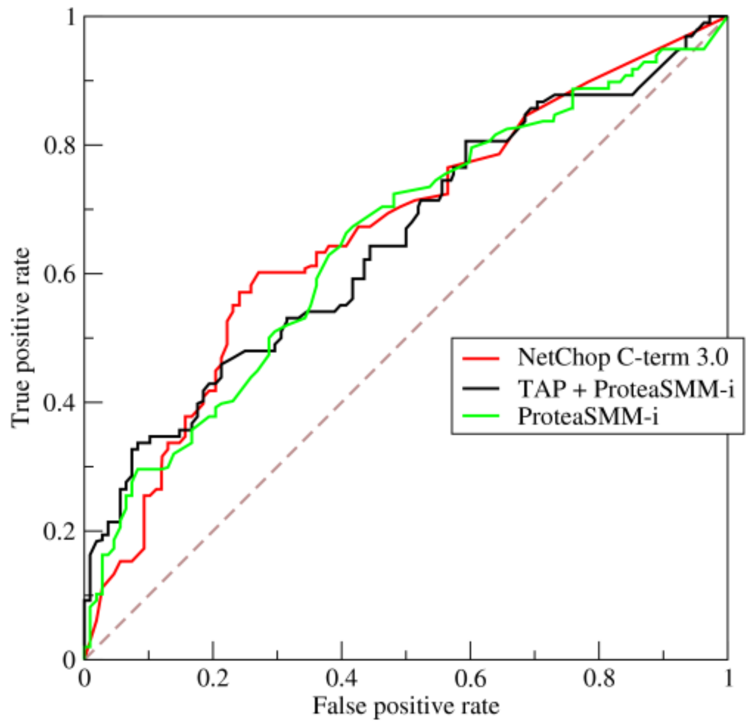
Probability of Detection: Fraction of true positives that are predicted positive

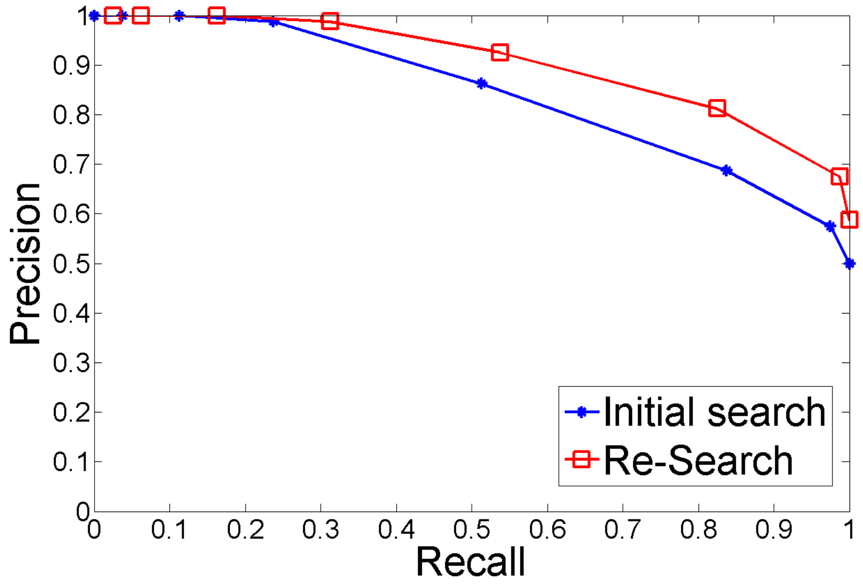
$$p_D = \frac{N_{TP}}{N_{TP}+N_{FN}}$$

Probability of False Alarm: Fraction of true negatives predicted positive

$$p_F = \frac{N_{FP}}{N_{FP}+N_{TN}}$$

ROC and Precision-Recall Curves





Classification Techniques

- Techniques Used
  - Naïve Bayes
    - GMM (Gaussian Mixture Models)
  - *k*-Nearest Neighbors
  - Hidden Markov Models (HMM)
  - Support Vector Machines (SVM)
  - Neural Networks
  - Decision Trees
- Not all these can be adapted for streaming
  - We will look at Decision Trees and a technique called
    - Very Fast Decision Trees (VFDTs)

Aside on Some Techniques

Likelihood

Class Prior Probability

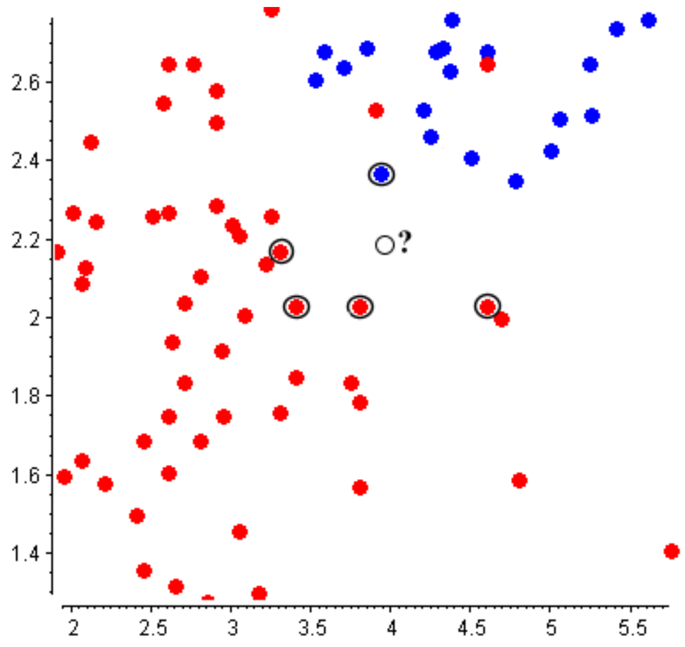
$$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

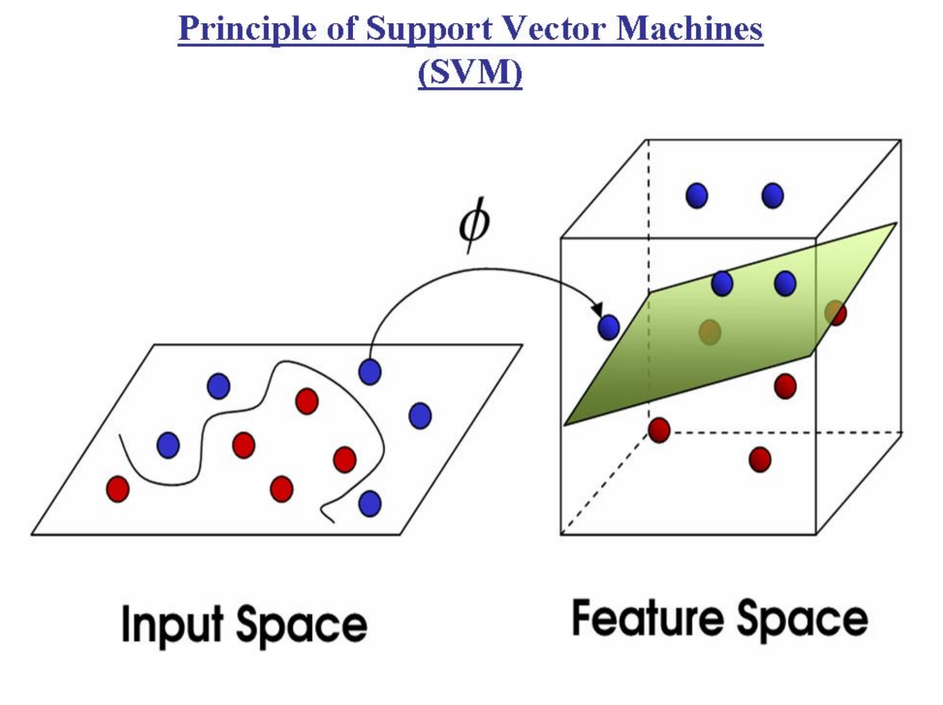
$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Naïve Bayes

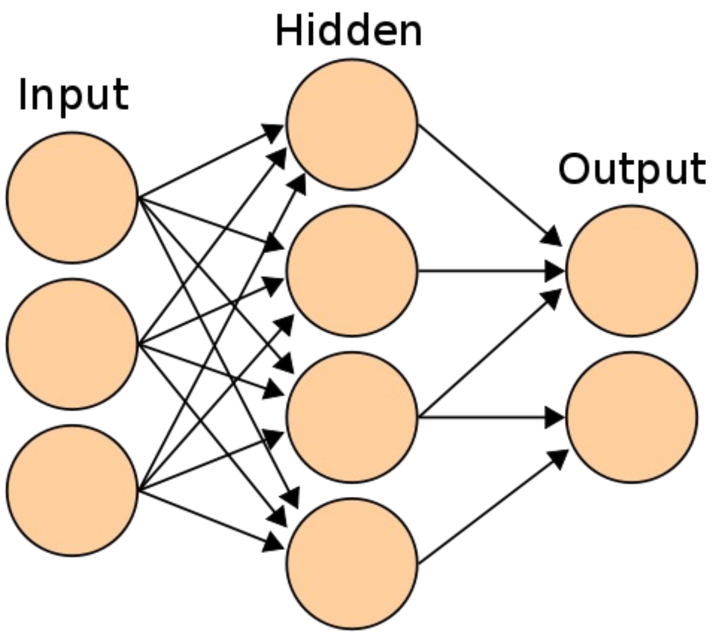


Nearest Neighbor Classifier

Aside on Some Techniques

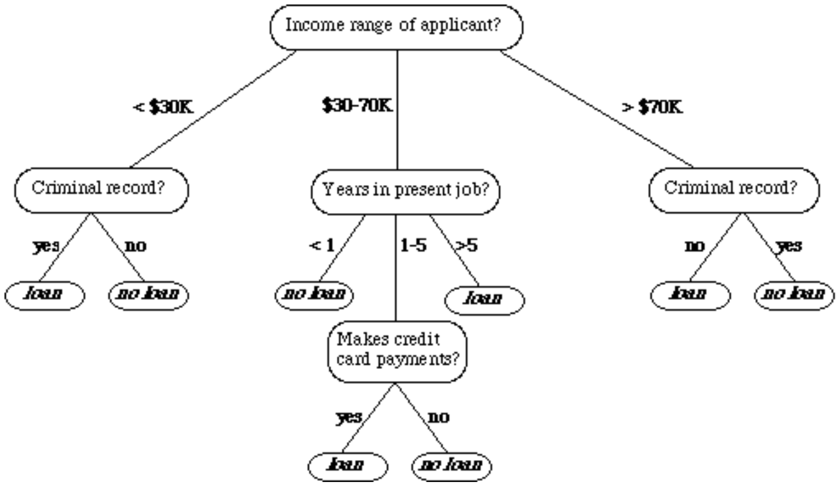


Support Vector Machines

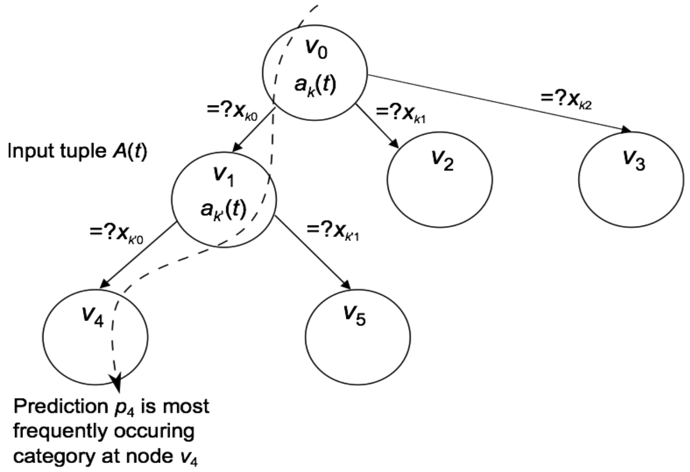


Neural Network

Decision Trees



Decision tree



- Decision Tree Learning
  - Learn the tree structure that classifies training data into multiple classes
- Decision Tree Scoring
  - Traverse the tree to determine labels for unseen values

Very Fast Decision Tree (VFDT)

- A technique that supports incremental learning and scoring
- The classification problem
  - Given a set of Q training tuples $A(t_i)$, each with N attributes $a_j(t_i)$, and their associated true categories $l(t_i)$, the goal is to determine a function Γ to map tuples to their categories
- The learning phase
  - Determine a tree structure that classifies tuples
  - At each node, the tree divides the data using one of the attributes
    - This continues until the labels within each leaf is homogenous or the number of data items is small
  - The attribute to use is determined based on a heuristic
    - An example is *information gain* based on entropy

Information Gain

- Information Gain:

$$IG(A;B) = H(A) - H(A|B)$$

- H is the entropy

$$H(A) = -\sum_a P(A=a) \cdot \log_2(P(A=a))$$

$$H(A|B) = \sum_b P(B=b) \cdot H(A|B=b)$$

- Entropy is a measure of diversity
  - It is 0 when all data items have the same category
- Picking the attribute split that maximizes the IG
  - Picking the option that reduces the entropy the most

Example

B (Attribute)	A (Category)
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

P(A=yes)=0.5
P(A=no) = 0.5

H(A) = 1

P(A=yes| B=Math) = 0.5
P(A=no| B=Math) = 0.5

H(A|B=Math) = 1

P(A=yes| B=History) = 0
P(A=no| B=History) = 1

H(A|B=History) = 0

P(A=yes| B=CS) = 1
P(A=no| B=CS) = 0

H(A|B=CS) = 0

H(A|B) = H(A|B=CS)1/4 +
H(A|B=History)1/4 +
H(A|B=Math)1/2
= 0.5

IG(A;B) = H(A)-H(A|B) = 0.5

Learning a Decision Tree

- Learning a decision tree offline is well understood
  - For each leaf node, for each attribute that is not used so far (reaching to that node from the root), compute the information gain with respect to the label
  - Split the leaf node that brings the highest IG
  - Remove leaves from consideration when their H is 0 or when the number of data items for them is small
- How can we do this when the data is very large?
- How can we do this when we get to see the data once?
- VFDT is an algorithm for doing this

VFDT Algorithm

- Attribute i takes one of the values in set Z_i
- Number of categories/classes is L
- The algorithm starts with a root node
- Incrementally splits leaf nodes
  - A leaf node cannot be split at any time
  - It can only be split after seeing *enough* number of samples
  - *Hoeffding Bound* is used to decide when it is the time to split
- To compute information gain when needed, counters are kept
  - n_{ijkm}: # of tuples whose ith attribute value has index j in Z_i , belong to the kth category, and that map to tree node m
  - This is all the info we need to compute IG for any attribute for any node in the tree

VFDT Algorithm Analysis

- Let's say the error in the estimate of IG is ΔIG

$$IG^{est} = IG + \Delta IG$$

  - The higher the number of samples seen is, the lower the error observed is
- The error is bounded by the Hoeffding Bound
  - *Hoeffding Bound* tells us that after seeing q tuples, the error is bounded by ε with probability 1-δ, where

$$|\Delta IG| \leq \epsilon = \sqrt{\frac{\log(|L|)^2 \log(1/\delta)}{2 \cdot q}}$$

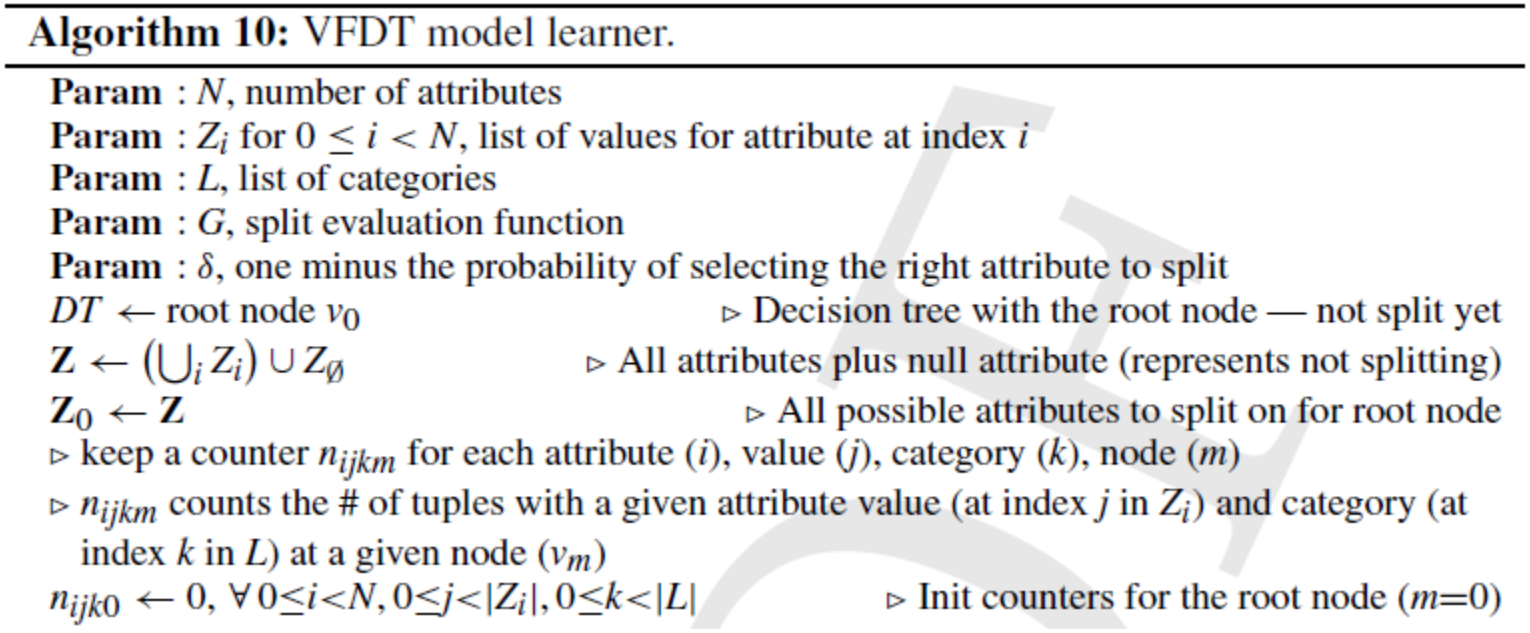
  - Then split is almost as good as the true split if

$$IG^{est}{}_1 - IG^{est}{}_2 \geq 2\,\epsilon$$

VFDT Algorithm

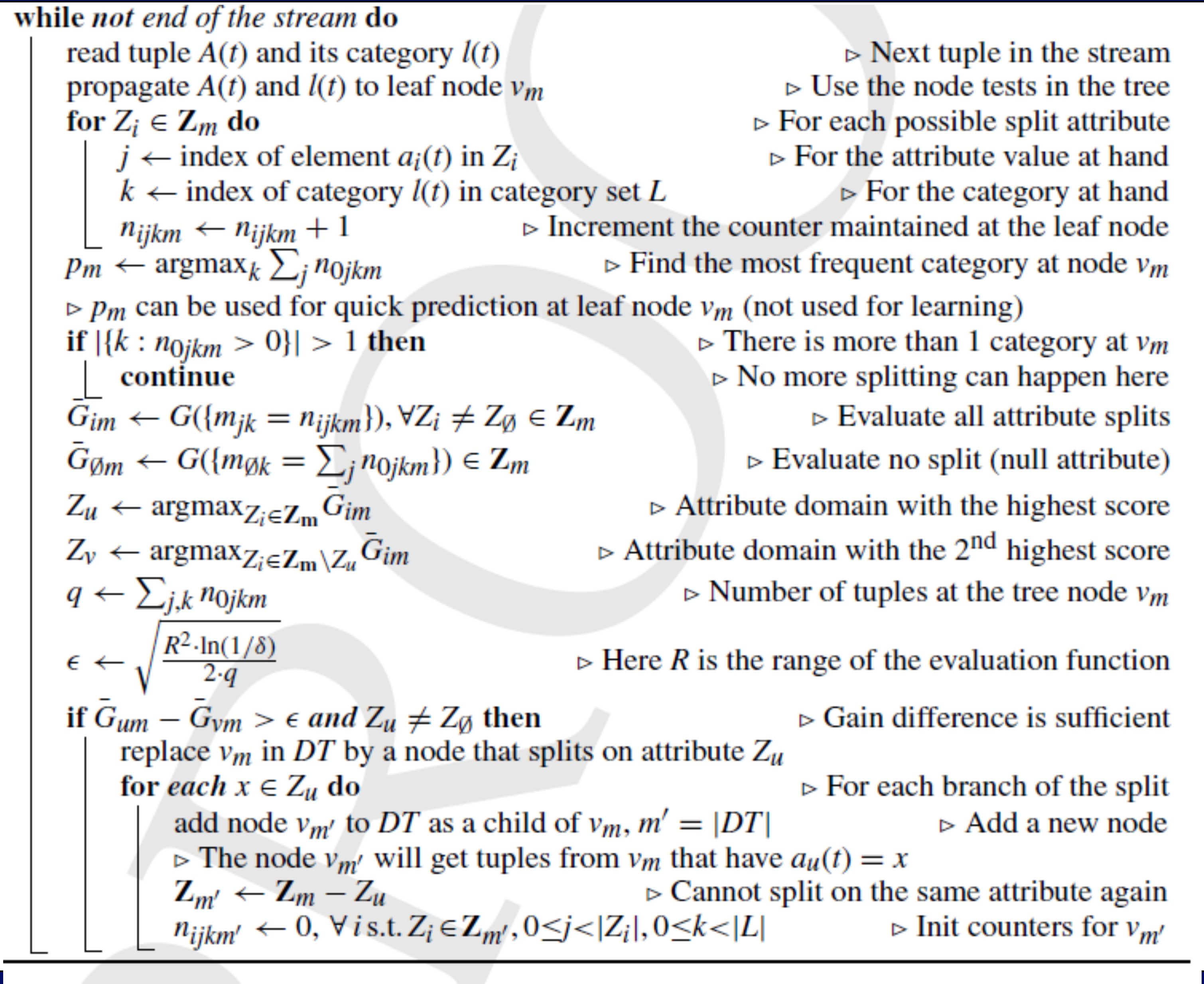
- For each update
  - Propagate down the tree to the leaf node
  - Update the leaf node's relevant counter
  - Compute the top two best splits gains
  - Compute the error bound
  - If the difference between the gains of the first two top splits is greater than the error bound
    - Perform the split

VFDT Algorithm Pseudocode



Algorithm 10: VFDT model learner.

```
Param : $N$, number of attributes
Param : $Z_i$ for $0 \leq i < N$, list of values for attribute at index $i$
Param : $L$, list of categories
Param : $G$, split evaluation function
Param : $\delta$, one minus the probability of selecting the right attribute to split
$DT \leftarrow$ root node $v_0$                    ▷ Decision tree with the root node — not split yet
$\mathbf{Z} \leftarrow \left(\bigcup_i Z_i\right) \cup Z_\emptyset$        ▷ All attributes plus null attribute (represents not splitting)
$\mathbf{Z}_0 \leftarrow \mathbf{Z}$                             ▷ All possible attributes to split on for root node
▷ keep a counter $n_{ijkm}$ for each attribute ($i$), value ($j$), category ($k$), node ($m$)
▷ $n_{ijkm}$ counts the # of tuples with a given attribute value (at index $j$ in $Z_i$) and category (at
  index $k$ in $L$) at a given node ($v_m$)
$n_{ijk0} \leftarrow 0, \forall\, 0 \leq i < N, 0 \leq j < |Z_i|, 0 \leq k < |L|$      ▷ Init counters for the root node ($m$=0)
```

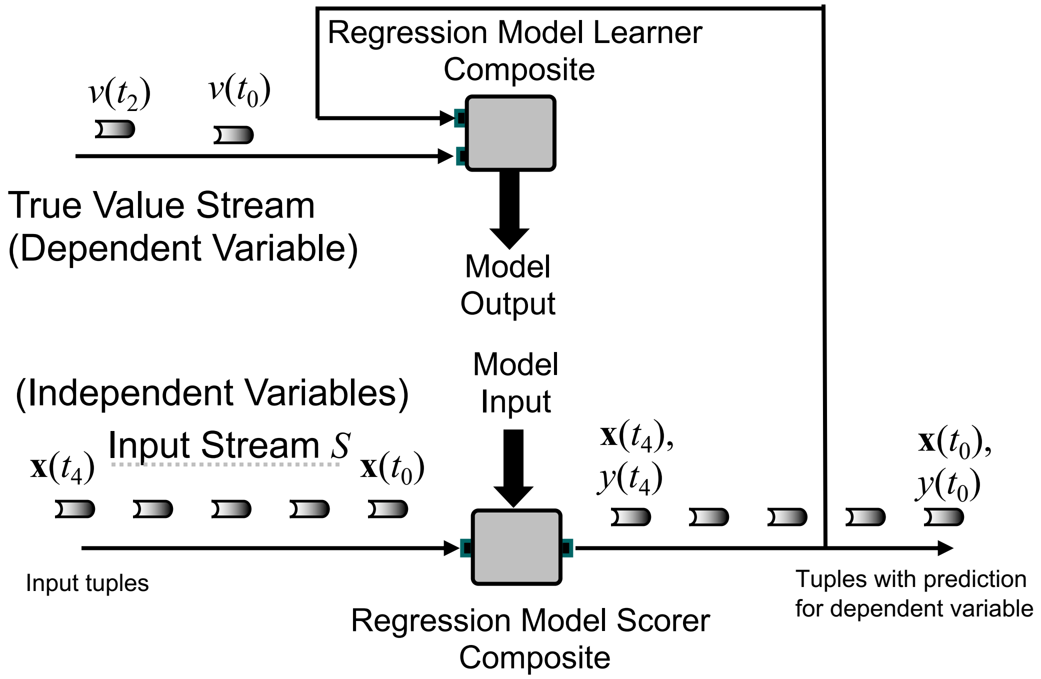


```
while not end of the stream do
    read tuple $A(t)$ and its category $l(t)$                                    ▷ Next tuple in the stream
    propagate $A(t)$ and $l(t)$ to leaf node $v_m$                                ▷ Use the node tests in the tree
    for $Z_i \in \mathbf{Z}_m$ do                                                 ▷ For each possible split attribute
        $j \leftarrow$ index of element $a_i(t)$ in $Z_i$                         ▷ For the attribute value at hand
        $k \leftarrow$ index of category $l(t)$ in category set $L$               ▷ For the category at hand
        $n_{ijkm} \leftarrow n_{ijkm} + 1$                                        ▷ Increment the counter maintained at the leaf node
    $p_m \leftarrow \text{argmax}_k \sum_j n_{0jkm}$                              ▷ Find the most frequent category at node $v_m$
    ▷ $p_m$ can be used for quick prediction at leaf node $v_m$ (not used for learning)
    if $|\{k : n_{0jkm} > 0\}| > 1$ then                                          ▷ There is more than 1 category at $v_m$
        continue                                                                  ▷ No more splitting can happen here
    $\bar{G}_{im} \leftarrow G(\{m_{jk} = n_{ijkm}\}), \forall Z_i \neq Z_\emptyset \in \mathbf{Z}_m$     ▷ Evaluate all attribute splits
    $\bar{G}_{\emptyset m} \leftarrow G(\{m_{\emptyset k} = \sum_j n_{0jkm}\}) \in \mathbf{Z}_m$        ▷ Evaluate no split (null attribute)
    $Z_u \leftarrow \text{argmax}_{Z_i \in \mathbf{Z_m}} \bar{G}_{im}$            ▷ Attribute domain with the highest score
    $Z_v \leftarrow \text{argmax}_{Z_i \in \mathbf{Z_m} \setminus Z_u} \bar{G}_{im}$   ▷ Attribute domain with the 2nd highest score
    $q \leftarrow \sum_{j,k} n_{0jkm}$                                            ▷ Number of tuples at the tree node $v_m$
    $\epsilon \leftarrow \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2 \cdot q}}$        ▷ Here $R$ is the range of the evaluation function
    if $\bar{G}_{um} - \bar{G}_{vm} > \epsilon$ and $Z_u \neq Z_\emptyset$ then   ▷ Gain difference is sufficient
        replace $v_m$ in $DT$ by a node that splits on attribute $Z_u$
        for each $x \in Z_u$ do                                                   ▷ For each branch of the split
            add node $v_{m'}$ to $DT$ as a child of $v_m$, $m' = |DT|$            ▷ Add a new node
            ▷ The node $v_{m'}$ will get tuples from $v_m$ that have $a_u(t) = x$
            $\mathbf{Z}_{m'} \leftarrow \mathbf{Z}_m - Z_u$                       ▷ Cannot split on the same attribute again
            $n_{ijkm'} \leftarrow 0, \forall\, i$ s.t. $Z_i \in \mathbf{Z}_{m'}, 0 \leq j < |Z_i|, 0 \leq k < |L|$   ▷ Init counters for $v_{m'}$
```

Regression

- Used to estimate relationship between a set of *independent* variables and a *dependent* variable
  - Works on numeric values
  - e.g. predicting stock price
- Similar to classification
  - Dependent variable takes discrete values
- Regression can be used to solve classification problems
- Modeling – estimation of regression function
- Evaluation – using the function to make predictions

Regression



Regression Model Learner Composite
$v(t_2)$
$v(t_0)$
True Value Stream (Dependent Variable)
Model Output
Model Input
(Independent Variables)
Input Stream S
$\mathbf{x}(t_4)$
$\mathbf{x}(t_0)$
$\mathbf{x}(t_4)$, $y(t_4)$
$\mathbf{x}(t_0)$, $y(t_0)$
Input tuples
Tuples with prediction for dependent variable
Regression Model Scorer Composite

Modeling (learning) and Evaluation (scoring) are both done online

Parametric Regression

- Uses finite number of parameters to represent relationship between dependent and independent variables
- Assumes a functional form for the relationship
- E.g. Linear Regression Model

$$\hat{y}(t) = \sum_{i=0}^{N-1} w_i x_i(t)$$

Estimate of Dependent variable

Independent variables

Weights (parameters)

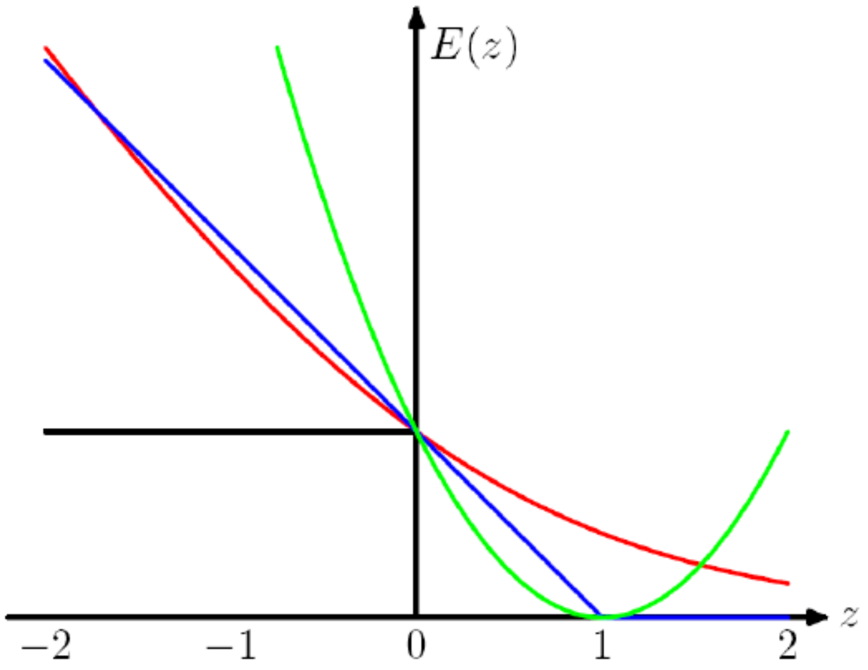
or

$$\hat{y}(t) = \mathbf{w}^T \mathbf{x}(t)$$

Parametric Regression

Loss associated with predicting $\hat{y}$ on x when the true label is y:

1. Hinge loss $\ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\}$ linear SVM
2. Logistic loss $\ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y}))$
3. Squared loss $\ell(\hat{y}, y) = (\hat{y} - y)^2 = (1 - y\hat{y})^2$
4. Median loss $\ell(\hat{y}, y) = |\hat{y} - y| = |1 - y\hat{y}|$ (and arbitrary quantiles)



$z = y\hat{y}$

- Modeling step involves learning the parameters
  - e.g. weights
  - Optimize some error function between estimate of independent variable and actual value

Non-Parametric Regression

- Makes no assumption on functional form of relationship between independent and dependent variables
- Non-Parametric Multiplicative Regression (NPMR)
- Kernel Regression
  - Use a kernel function to smooth out influence of points
- Regression Trees
  - Split criterion based on error between prediction and value as opposed to information gain
- Often require more training data for modeling than parametric methods

Stochastic Gradient Descent

- Online algorithm to estimate linear regression model

Algorithm 1 Stochastic Gradient Descent

```
$w_1 \leftarrow 0$
**for** $t = 1$ to $T$ **do**
    $w_{t+1} \leftarrow w_t - \underbrace{\eta_t \nabla_w \ell(w_t \cdot x_t, y_t)}_{\text{update}}$
**done**
```

Here $\nabla_w \ell(w_t \cdot x_t, y_t)$ is the gradient of the loss with respect to w evaluated at the t-th prediction $w_t \cdot x_t$ and label y_t, and η_t is the learning rate.

Stochastic Gradient Descent

Quadratic Loss $l\left(\mathbf{w}_t^T\mathbf{x}(t), y(t)\right) = \left(y(t) - \mathbf{w}_t^T\mathbf{x}(t)\right)^2$

Gradient $\nabla_{\mathbf{w}} l\left(\mathbf{w}_t^T\mathbf{x}(t), y(t)\right) = -2\left(y(t) - \mathbf{w}_t^T\mathbf{x}(t)\right)\mathbf{x}(t)$

Update $\mathbf{w}_{t+1} = \mathbf{w}_t + 2\eta_t\left(y(t) - \mathbf{w}_t^T\mathbf{x}(t)\right)\mathbf{x}(t)$

Learning Rate $\eta_t = \dfrac{\eta_0}{\sqrt{t}}$

If data is stationary, reducing the learning rate makes sure that weights do not oscillate too much

SGD: Safe Updates

Solution: Enforce safe updates, for which the residual $w_t^\top x_t - y_t$ doesn't change its sign after the update (i.e., the learner doesn't move its prediction on the current example beyond the label as a result of the update)

Enforcing safe updates allows for a more aggressive learning rate, which improves the learned predictor at essentially no computational cost.

For squared loss, we use

$$w_{t+1} \leftarrow w_t - \frac{w_t \cdot x_t - y}{x_t^\top x_t} \left(1 - e^{-\eta_t}\right) x_t$$

Safe updates are particularly helpful when different examples have different importances (η_t just gets multiplied by the importance), but it improves the learned predictor even if all examples have the same weight.

SGD: Adaptive Gradients

Adaptive Gradients:

- Use an individual learning rate for each feature, adjusted automatically according to a data-dependent schedule.
- The learner effectively ups the importance of very predictive but rarely seen features. So we can expect to see the most improvement when different features are present at different rates, but it improves the learned predictor even on dense datasets.
- This learning schedule is more agressive from the start, so it's very useful when combined with safe updates.

Vowpal Wabbit Library

- An open source learner currently sponsored by Microsoft Research
  - http://hunch.net/~vw/
- Very scalable public learner with state of the art predictive technology
- Online learning algorithm
- Parallel implementation
- Inherently modular, making it a perfect vehicle for advanced learning research
- Open source project.

VW Library

Input Features

Label [Importance] [Base] |Namespace Feature ... |Namespace Feature ... ...

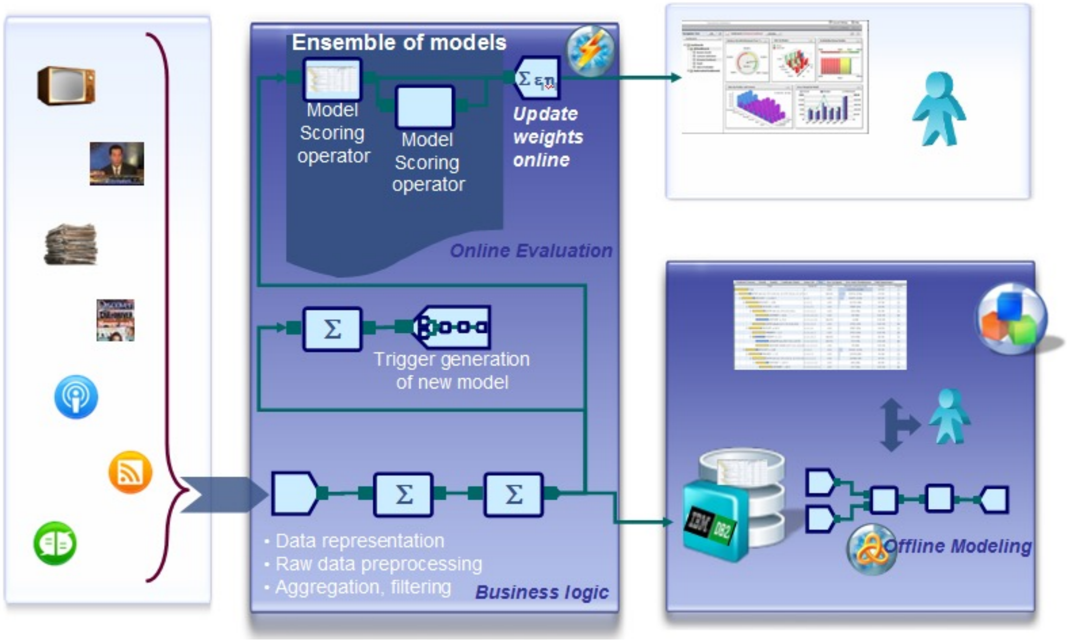
Namespace is a mechanism for feature manipulation and grouping:
Namespace = String[:Float]
Feature = String[:Float]
Importance is multiplier on learning rate, default 1.
Base is a baseline prediction, default 0.
To specify an example for scoring, omit the label. Example:

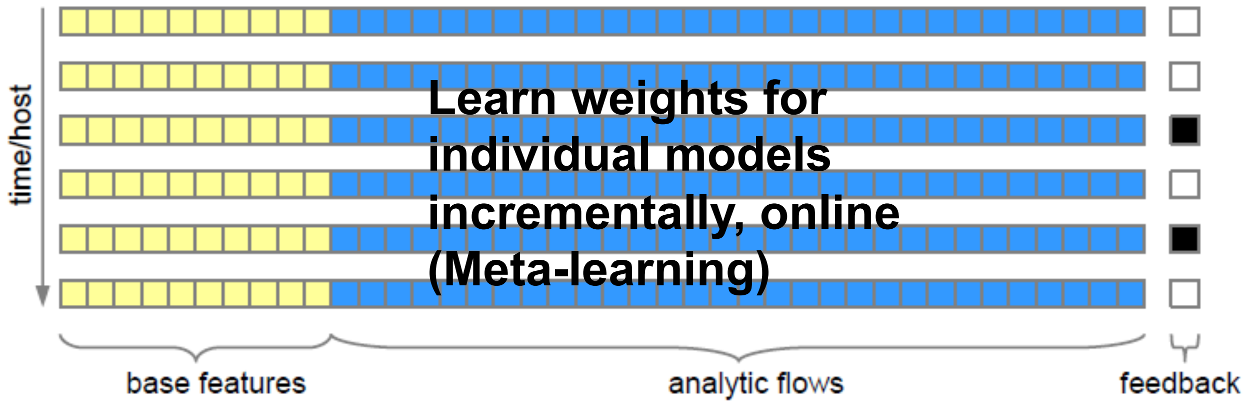
1 |tweet Nicholas Lemann asks what happened to the environmental movement |account screen_name:NewYorker followers_count:2,272,893 tweets:11,237 verified:YES . . .

Algorithms

On-the-fly feature manipulators with namespaces {ngrams, skipgrams, ignored, quadratic, cubic}
Core problems (importance weighted) binary classification, Regression
Optimizers for core problems fonline, CG, LBFGSg
Representations (linear, matrix factorization, latent dirichlet allocation (unsupervised), neural networks)
Losses squared, hinge, logistic, quantile
Multiclass one-against-all, error-correcting tournaments
Cost-sensitive multiclass, one-against-all, weighted all pairs
Active (importance weighted) learning
Structured searn, imperative searn
Cluster parallel mode, daemon mode

Linear Regression and Ensemble Learning





Learn weights for individual models incrementally, online (Meta-learning)

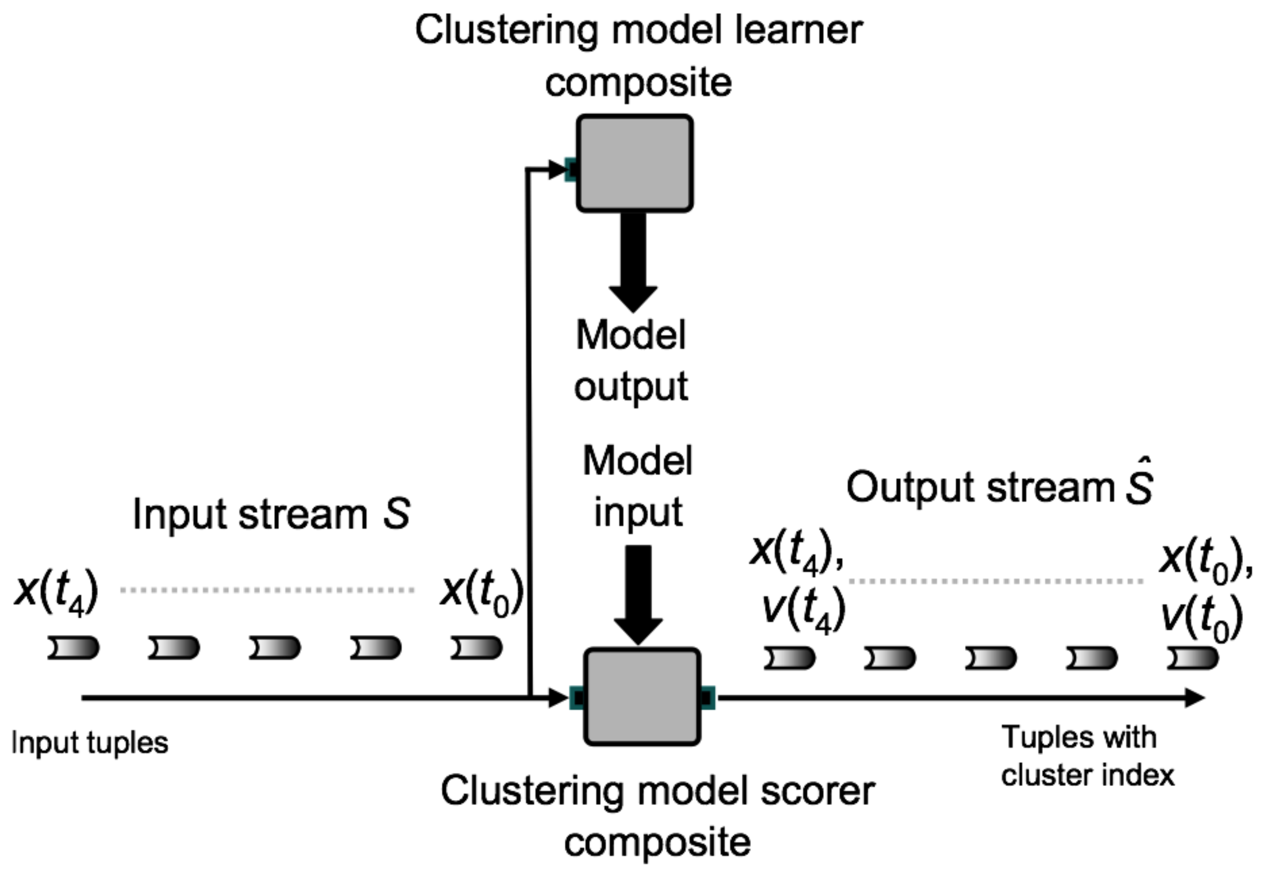
Linear Regression Summary

- Parametric and Non-parametric models
- Linear models popular
  - Very easy to compute
  - Can be made arbitrarily complex by creating appropriate features
  - Outputs from other models can be features (meta-learning)
  - Can support incremental feature discovery
- How to do linear regression under budget constraints for meta-learning?

Clustering

- Widely studied problem in data mining literature
- An unsupervised technique
  - Given a set of items, divide them into groups
    - Similar items are in the same group
- There needs to be a *similarity metric* defined over pairs of data items
  - For numeric values, this can be a distance function
- Example scenario:
  - You have CDRs (call detail records) in telecommunication
  - Cluster users according to their call profiles
    - Users that are similar in terms of their calling behavior are grouped together
- Classical approach to clustering uses offline learning, online scoring
  - Run clustering algorithms on historical data
  - Use results for assigning clusters to new data items

Online Clustering Model



Modeling (learning) and Evaluation (scoring) are both done online

Types of Clustering Algorithms

- Kinds of clustering algorithms
  - Centroid-based clustering
  - Distribution-based clustering
  - Hierarchical clustering
  - Density-based clustering

Centroid Based Clustering

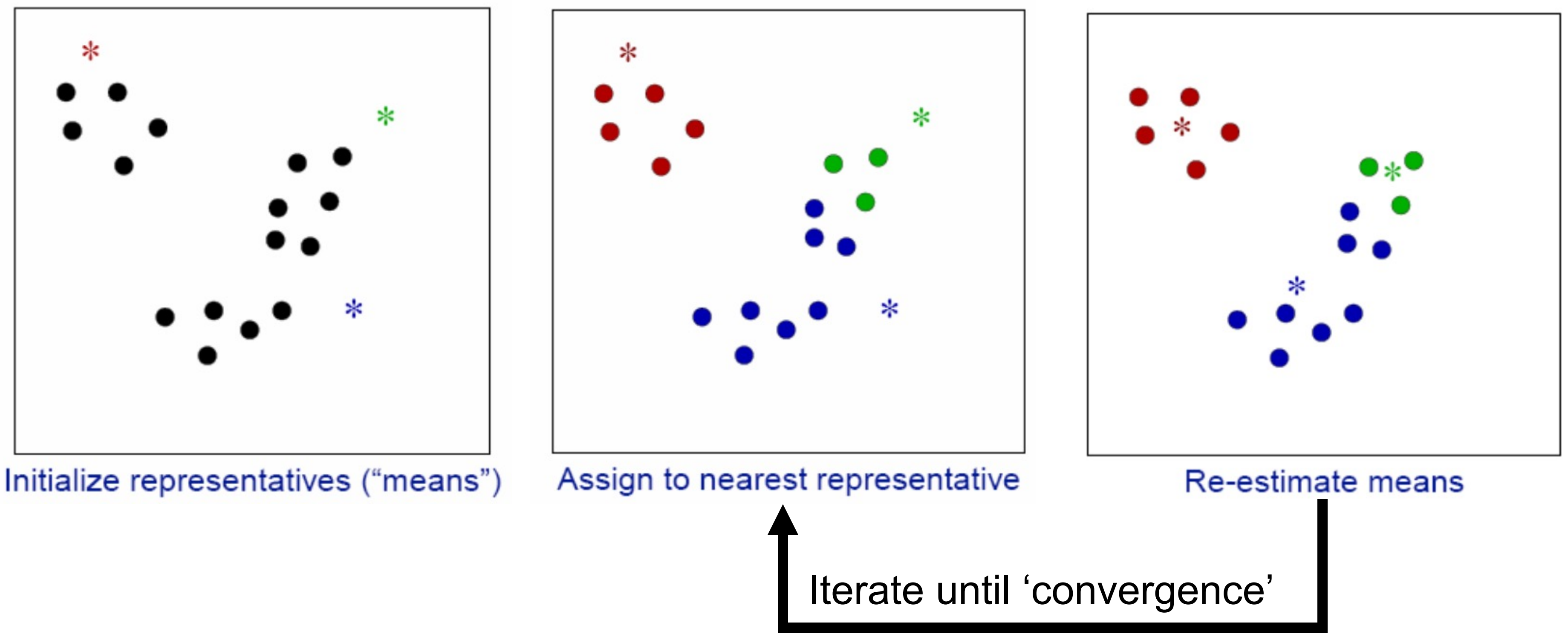
- Each cluster C_j is represented by a centroid vector μ_j
- Each data item is assigned to the cluster with the closest centroid vector
- The problem can be represented as an optimization, where the goal is to minimize the squared error

$$\text{argmin}_{\{\mathcal{C}_j,\mu_j\}} \sum_{j=0}^{k-1} \sum_{x(t)\in\mathcal{C}_j} |\mathbf{x}(t) - \mu_j|^2$$

- This is an NP-Hard problem

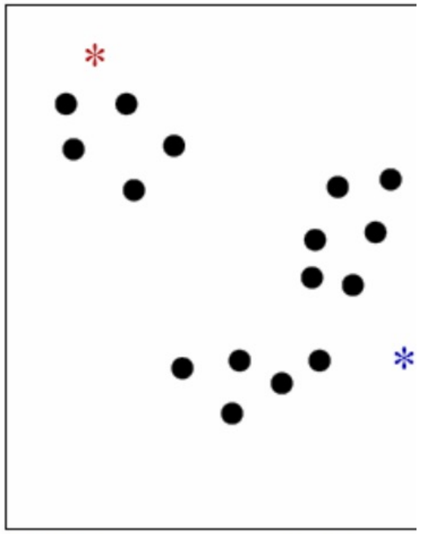
k-Means Algorithm

Gradient Descent based Algorithm for Centroid based Clustering

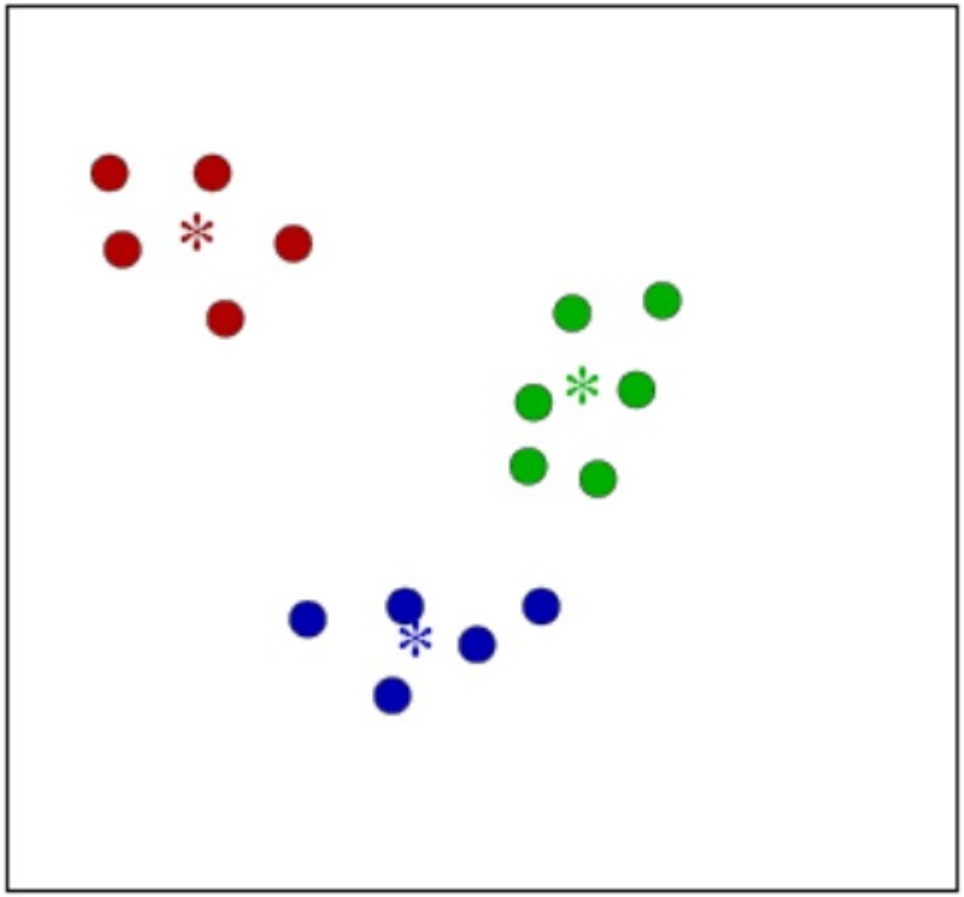

Initialize representatives (“means”)
Assign to nearest representative
Re-estimate means
Iterate until ‘convergence’

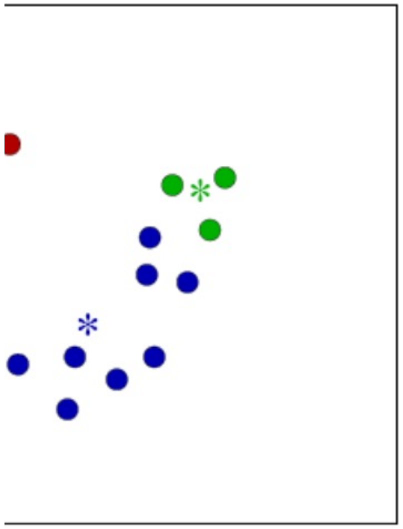
Note similarity to Lloyd algorithm for quantization

k-Means Algorithm



Initialize representatives (





-estimate means

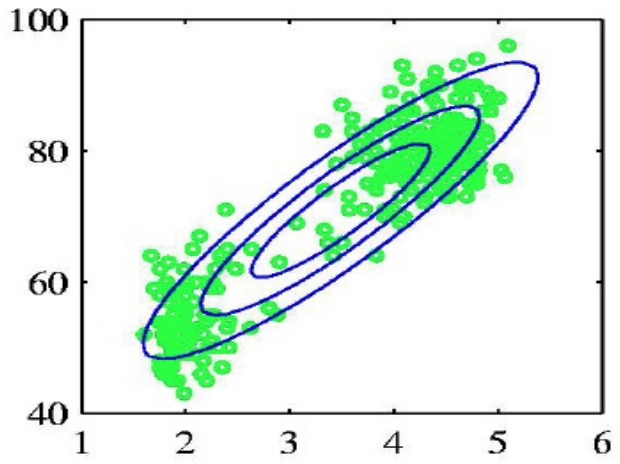
ce'

Centroid Based Clustering

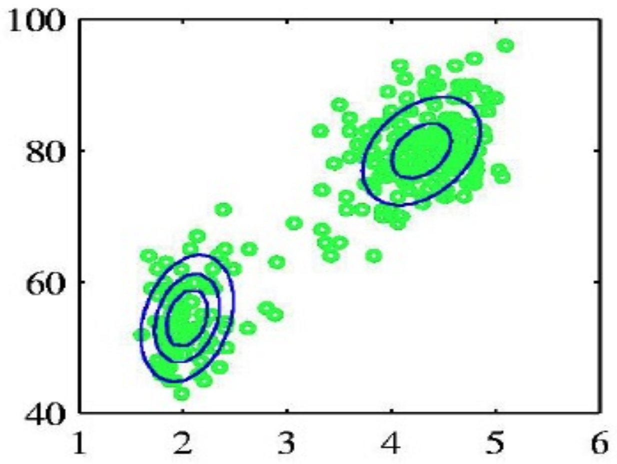
- The k-means algorithm being gradient descent
  - Can converge to local optima
  - Depends on choice of initial centroids
- k-means based algorithms (k-medoid, k-median) require the number of clusters (k) to be specified
  - There are some algorithms that can set *k* automatically. Often requires penalizing solutions with too many clusters
- Alternate Approach: Nearest Neighbor (NN) algorithm [Equitz, 1984]
  - Start with the entire training set
  - Merge the two vectors that are closest into one vector equal to their mean
  - Repeat until the desired number of vectors is reached, or the distortion exceeds a certain threshold

Distribution based Clustering

- The idea is to represent each cluster as a distribution
  - Parametric Model
- Select the best set of distributions that describe the data
- A typical approach is to use a Gaussian mixture model (GMM)



Single Gaussian



Mixture of two Gaussians

Distribution based Clustering

- Multivariate Gaussian

$$\mathcal{N}(\mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{2 \cdot \pi \cdot det(\Sigma)}} \cdot e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \times \Sigma^{-1} \times (\mathbf{x}-\mu)}$$

- Mixture of k Gaussians

$$\sum_{j=0}^{k-1} w_j \cdot \mathcal{N}(\mu_j, \mathbf{\Sigma}_j)$$

- Goal of distribution based clustering
  – find best weights, means, and the covariance matrices

Distribution based Clustering

- Using Bayes rule

$$P(\mu_j, \Sigma_j \mid x) = \frac{P(x \mid \mu_j, \Sigma_j) P(\mu_j, \Sigma_j)}{P(x)}$$

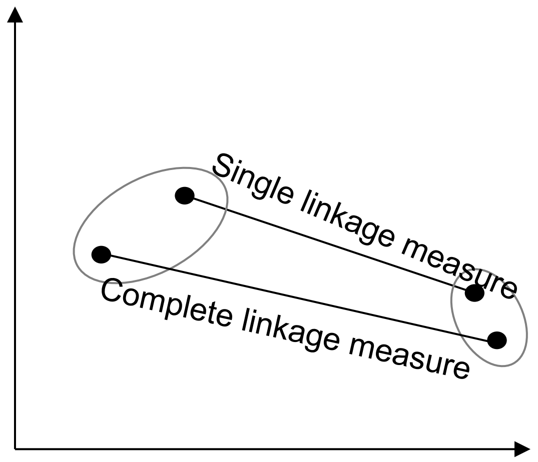
- Transform this estimation
  – Maximize the likelihood of the data (X) given the model

$$\prod_{x \in X} \sum_{j=0}^{k-1} w_j \cdot \mathcal{N}(x|\mu_j, \Sigma_j)$$

- Typically solved using Gradient Descent algorithms
  – Expectation Maximization (EM) Algorithm

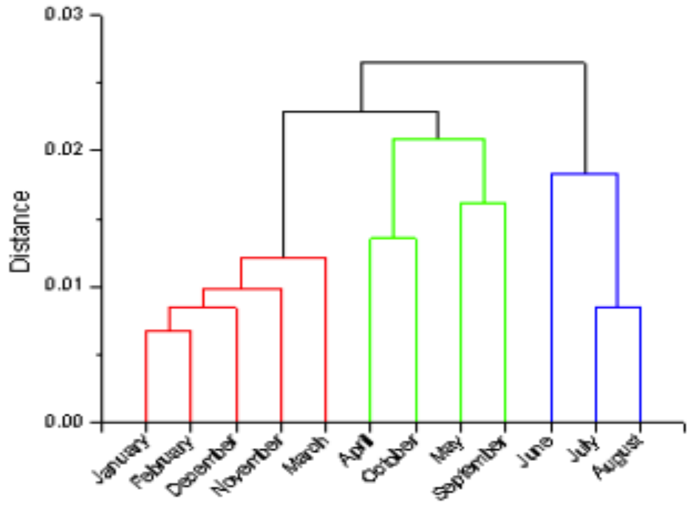
Hierarchical Clustering

- Also known as connectivity-based clustering
- Based on the idea of grouping clusters that are closest to each other
- Closeness can be defined in different ways
  - Single linkage measure: smallest distance between any pair
  - Complete linkage measure: largest distance between any pair
  - Average linkage measure: the average distance between pairs



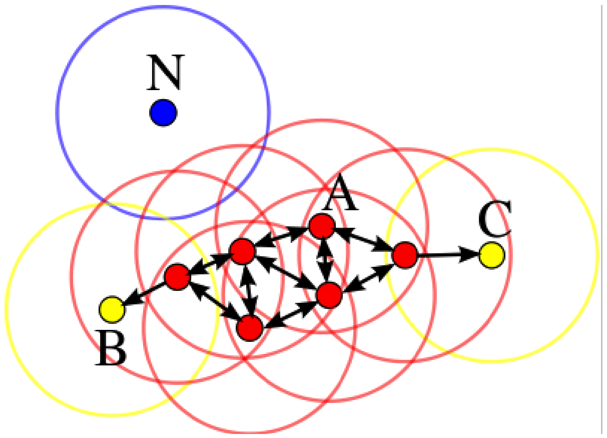
Hierarchical Clustering

- There are two approaches to this: agglomerative and divisive
  - Top-down (divisive), bottom up (agglomerative)
- Finding the right cut-off point is a challenge
- Dendrograms

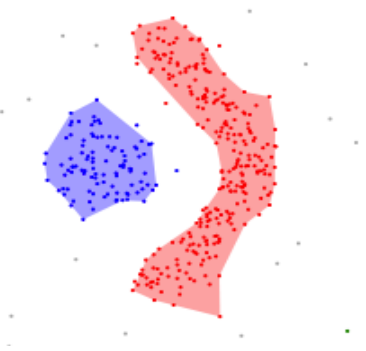


Density Based Clustering

- Data items that are ‘close enough’ are considered part of the same cluster
  - A density parameter is used for this purpose
  - Uses a linkage model
  - Data items in sparsely populated areas are considered as noise
- Number of clusters is not specified
- Can create arbitrary shape clusters
- A popular algorithm is Density Based Spatial Clustering of applications with noise (DBSCAN)
- Works well only for low dimensions



Can result in arbitrary shaped clusters



Clustering Algorithms

- Modeling step of Clustering
  - Algorithms are not incremental
    - Require access to all data
  - Gradient descent based
    - Not necessarily optimal
  - Require additional steps to determine number of clusters
- Evaluation step of Clustering
  - Nearest Neighbor labeling

CluStream: Online Clustering

- Maintain, in a streaming fashion, a set of Microclusters that can be used for more coarse-grained centroid-based clustering
- *Microcluster*: A small cluster represented by a cluster *feature vector*
- Cluster feature vector: Keeps first order and second order moments (which are additive)

CluStream: Microclusters

- A microcluster containing Q tuples stores
  - First two moments for the tuples

$$\mathbf{m1} = \sum_{i=0}^{Q-1} \mathbf{x}(t_i) \qquad \mathbf{m2} = \sum_{i=0}^{Q-1} \mathbf{x}(t_i) \bullet \mathbf{x}(t_i)$$

  - First two moments for the timestamps of tuples

$$\tau 1 = \sum_{i=0}^{Q-1} t_i \qquad \tau 2 = \sum_{i=0}^{Q-1} (t_i)^2$$

  - It also stores a micro-cluster id
- In summary, a micro-cluster is a 6-tuple:

$$\mathcal{M} = \{\mathbf{m1}, \mathbf{m2}, \tau 1, \tau 2, Q, id\}$$

CluStream

- Initialization: To bootstrap the algorithm, k-means is used with k set to the number of desired micro-clusters
  - This requires accumulating sufficient number of tuples
- Whenever a new tuple arrives, there are the following possibilities:
  - The tuple belongs to an existing microcluster
  - The tuple is an outlier and does not belong to a cluster
  - The tuple represents an evolution in the stream characteristics, and thus a new microcluster has to be created

CluStream

- To determine if a tuple belongs to an existing cluster
  - Locate the microcluster with the closest centroid
  - Compute an *extent* for the closest microcluster
  - If the distance of the tuple to the microcluster centroid is less than then extent, then it belongs to an existing cluster
- Computing the microcluster extent
  - Some constant (β) times the sum of the standard deviations in each dimension

$$E_p = \frac{\beta}{Q_p} \cdot \left(\sqrt{Q_p \cdot \mathbf{m2}_p - \mathbf{m1}_p \bullet \mathbf{m1}_p} \right)^T \times \mathbf{1}$$

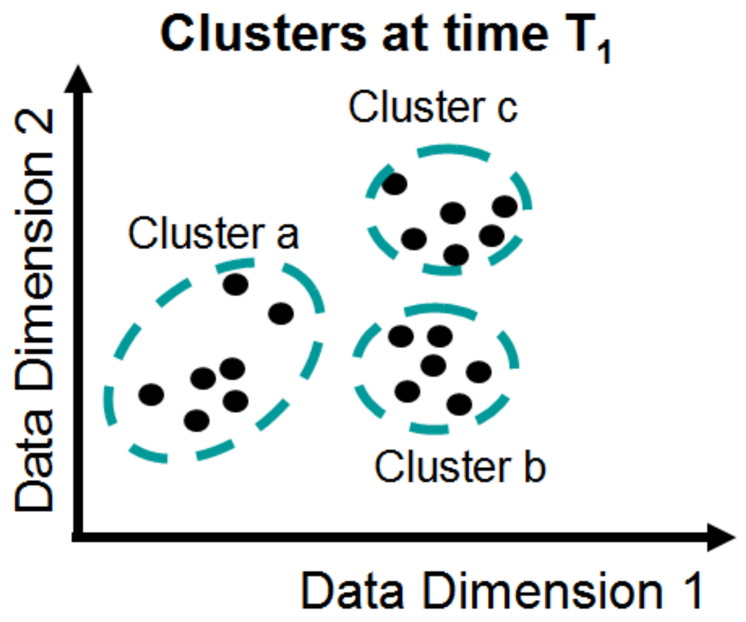
CluStream

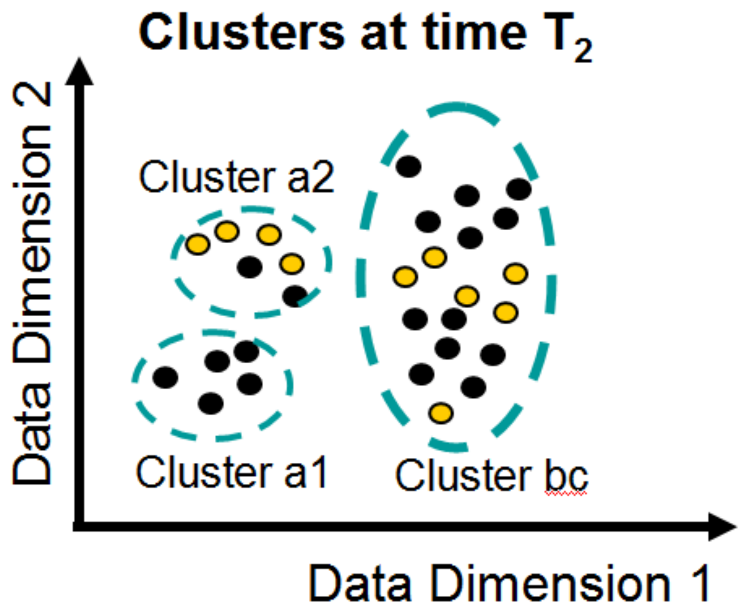
- Extent for a singleton microcluster ($Q=1$)
  - Compute extent for nearest non-singleton microcluster and scale it
- If the tuple belongs to one of the existing microclusters p, the microcluster feature vector is simply updated
  - Incrementally update the four moments and Q_p
- Otherwise, create a new cluster
  - Either remove an existing microcluster
  - Or merge two microclusters

CluStream

- First, the relevance of each microcluster is computed
  - This requires the mean and the variance of the item timestamps
  - Assume Gaussian and compute a timestamp x (the relevance) such that tuples with timestamps greater than x constitute some y% of the tuples within microcluster. Here y is an algorithmic parameter.
- If the relevance for the microcluster with the smallest relevance is less than current time - δ, then the microcluster is removed (it is too old)
- Otherwise
  - The closest two microclusters are merged
  - Merging is easy, as the moments are additive. Ids keep track of both ids of the merged clusters

CluStream



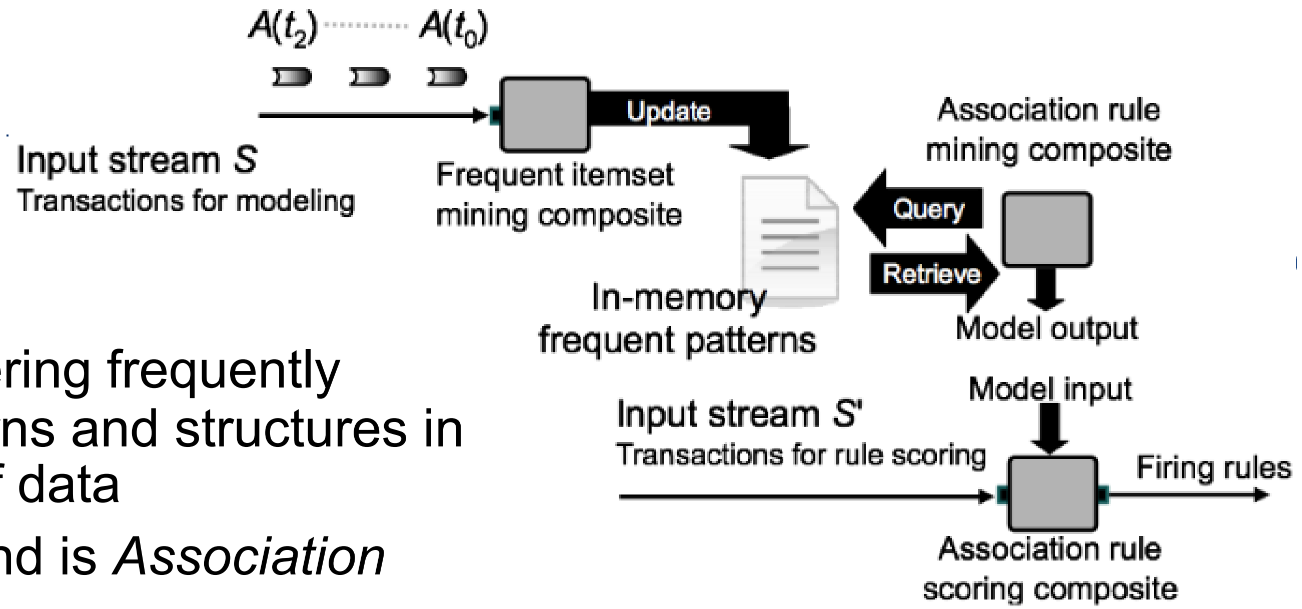


If newly arriving tuple is an outlier – it will be discarded at some point
If it represents a change in stream characteristics it will be retained
Pseudocode in the book chapter 11

Clustering

- CluStream
  - Can be combined with traditional k-means clustering as a post step
  - Extended to account for cases with noise
    - Uncertainty in data observations
- Other types of clustering
  - Fuzzy Clustering
  - Graph Clustering
    - Useful for clique detection

Frequent Pattern Mining


$A(t_2)$ ········ $A(t_0)$
Update
Input stream S
Transactions for modeling
Frequent itemset mining composite
Association rule mining composite
Query
Retrieve
In-memory frequent patterns
Model output
Model input
Input stream S'
Transactions for rule scoring
Firing rules
Association rule scoring composite

- Aims at discovering frequently occurring patterns and structures in various types of data
- An important kind is *Association Rule Mining*
- E.g. market basket analysis: *People who buy a, b, c tend to buy x as well*

Terminology

- An itemset is a set of items
  - Let $\mathcal{I}$ be the set of all items
  - $X \subset \mathcal{I}$ is an itemset
- A *k*-itemset is a set of items of size k
  - X = {milk, chips} is a possible 2-itemset
- A transaction $\mathcal{T}$ has unique id t and an itemset I
  - The set of all transactions is $\mathcal{D}$
- The *cover* of an itemset X over a transaction set $\mathcal{D}$ is the set of transactions $\mathcal{T}$ whose itemset I is a superset of X
- The *support* of an itemset is the size of its cover

Example

Transaction id	I: itemset
1	{milk, chips, beer}
2	{chips, beer}
3	{pizza, milk}
4	{chips, pizza}

$$cover(X, \mathcal{D}) = \{\mathcal{T}.tid \mid \mathcal{T} \in \mathcal{D}, X \subseteq \mathcal{T}.I\}$$

$$support(X, \mathcal{D}) = |cover(X, \mathcal{D})|$$

$$relative\ support(X, \mathcal{D}) = \frac{support(X, \mathcal{D})}{|\mathcal{D}|}$$

Itemset	Cover	Support
{}	{1, 2, 3, 4}	4
{chips}	{1, 2, 4}	3
{beer}	{1, 2}	2
{pizza}	{3, 4}	2
{milk}	{1, 3}	2
{beer, chips}	{1, 1}	2
{beer, milk}	{1}	1
{chips, pizza}	{4}	1
{chips, milk}	{1}	1
{pizza, milk}	{3}	1
{beer, chips, milk}	{1}	1

Frequent Itemsets and Association Rules

- An itemset is *frequent* if its support is above a threshold
  - $support(X, \mathcal{D}) \geq \sigma$
  - The set of all frequent itemsets is $\mathcal{F}(\mathcal{D}, \sigma)$
- An association rule is in the form $X \Rightarrow Y$
  - X is an itemset called the *body* (antecedent)
  - Y is an itemset called the *head* (consequent)
- The confidence of a rule is the fraction of transactions that support X, that also support Y
  - $confidence(X \Rightarrow Y) = \dfrac{support(X \cup Y, \mathcal{D})}{support(X, \mathcal{D})}$

Frequent Itemsets and Association Rules

- A rule is *confident* if its confidence is above a threshold
  - $confidence(X \Rightarrow Y) \geq \gamma$

$$\mathcal{R}(\mathcal{D},\sigma,\gamma) = \{X \Rightarrow Y \mid X \cap Y = \emptyset, \; X \cup Y \in \mathcal{F}(\mathcal{D},\sigma), \; confidence(X \Rightarrow Y, \mathcal{D}) \geq \gamma\}$$

Association Rule	Support	Confidence
{beer} ⇒ {chips}	2	100%
{beer} ⇒ {milk}	1	50%
{chips} ⇒ {beer}	2	66%
{pizza} ⇒ {chips}	1	50%
{pizza} ⇒ {milk}	1	50%
{milk} ⇒ {beer}	1	50%
{milk} ⇒ {chips}	1	50%
{milk} ⇒ {pizza}	1	50%
{beer, chips} ⇒ {milk}	1	50%
{beer, milk} ⇒ {chips}	1	100%
{chips, milk} ⇒ {beer}	1	100%
{beer} ⇒ {chips, milk}	1	50%
{milk} ⇒ {beer, chips}	1	50%

Algorithms for Association Rules

- There are several algorithms for finding such rules
  - The AIS algorithm
  - The Apriori algorithm
  - The Eclat algorithm
  - FP-growth algorithm
- All these require computing frequency of items
  - In a streaming setting, how would we find frequent items?
    - Without keeping state proportional to the unique item count
  - Recall we looked at frequency estimation using sketches
  - However, the problem there was to predict the frequency of a *given* item
  - Here, the items are not given as part of the query

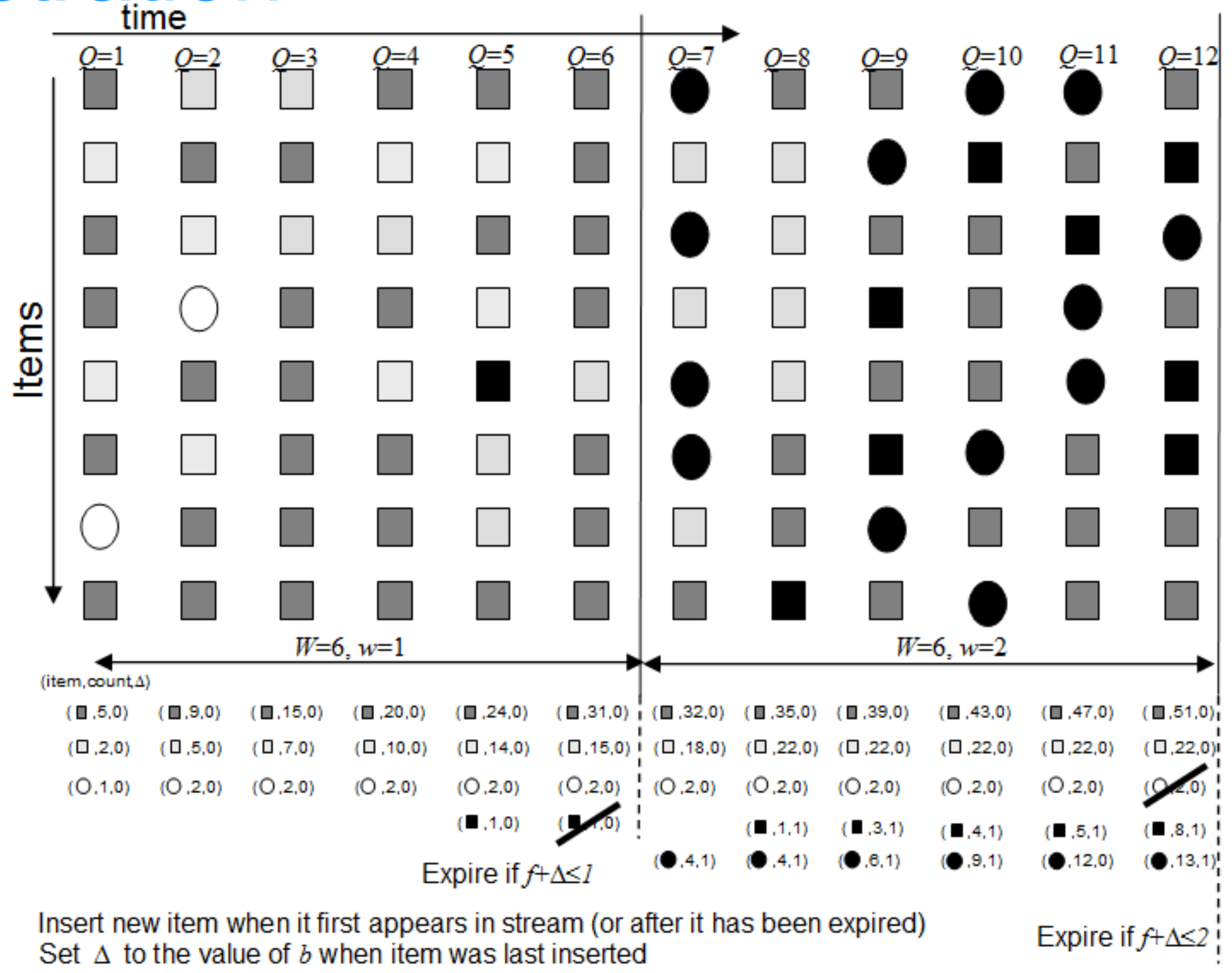
Lossy Counting Algorithm

- Given a *relative support* threshold σ, finds all items with frequency at least $\sigma - \varepsilon$
- Typically, ε is set to $\sigma/10$
- Let Q be the number of time steps so far
- The algorithm has the following properties
  - All items that have occurred $Q\times\sigma$ times are reported
  - None of the items that have occurred less than $Q\times(\sigma-\varepsilon)$ times are reported
  - The estimated support for a reported item is at most $Q\times\varepsilon$ less than its original value
  - Uses space in the order of $(1/\varepsilon)\times\log(Q\times\varepsilon)$

Lossy Counting Algorithm

- Break the stream into windows of size $W = \left\lceil \frac{1}{\varepsilon} \right\rceil$
- The algorithm keeps a simple data store
  - For each itemset in the data store, it keeps two values
    - A count for the item, f
    - An error estimate on the count, Δ
- Update process
  - Maintain a window index w, initially set to 1
  - Increment w after every 1/ϵ tuples,
  - When an item is received, if it is already in the store, increment its count, $f=f+1$
  - Otherwise, create a new entry for it, with the count set to $f=1$ and the error estimate set to $\Delta=w-1$
  - Before incrementing w, remove entries for which the count + error estimate is less than the window index, i.e., $f+\Delta \leq w$

Illustration



Insert new item when it first appears in stream (or after it has been expired)
Set Δ to the value of b when item was last inserted
If item already present, increment count
Expire items at boundaries of bucket size W

Intuition behind Lossy Counting

- It is easy to see that
  - At the window boundaries, the algorithm makes sure that all items that are guaranteed to have frequency less than ε are removed
  - The count f of an item cannot undershoot its real value by more than $Q \times \varepsilon$
- Pseudocode in book

Frequent Itemset Mining Summary

- Lossy Counting Algorithm can result in false positives
  - Can report some itemsets as frequent, when they are not
  - Optimistic estimate of the error
- Other extensions possible

Summary

- Classification
  - VFDT available as an open source library
    - http://www.cs.washington.edu/dm/vfml/vfdt.html
  - Several extensions to account for
    - Sliding windows
    - Concept adaptation
- Next Class
  - Frequent Pattern Mining, Outlier Detection