Anomaly Detection in Evolving Data Streams

COMP90073
Security Analytics

Sarah Erfani, CIS

Semester 2, 2021

Outline

- Introduction to data streams
- Windowing techniques
- HS-Trees
- Incremental LOF (iLOF)
  - Memory-efficient iLOF (MiLOF)

Data Streams

Data stream is a sequence of data points, which is *continues*, *unbounded*, and *nonstationary*.

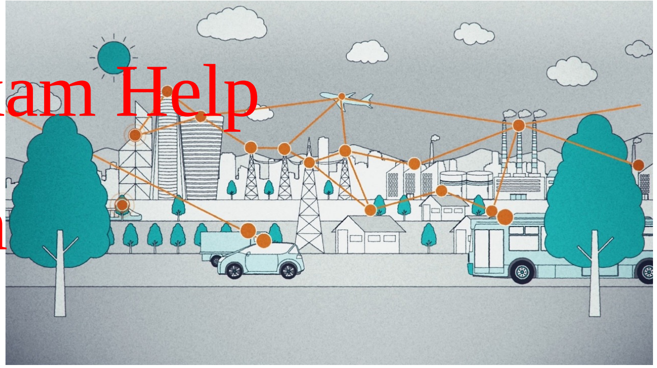
- **Streamlining Analysis**

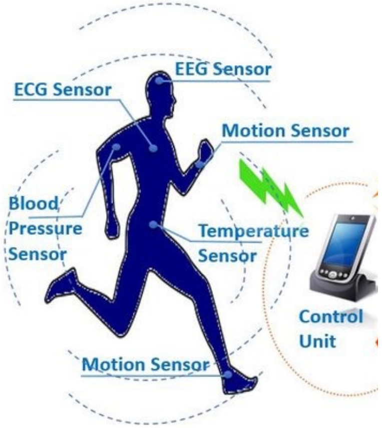
  – Large volume of data

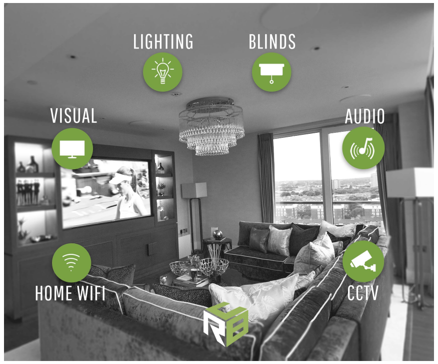
  – Short/real-time response

  – Limited memory

  – Energy/communication constraints







Batch Learning vs. Incremental Learning

Batch Leaning: Data points are *stored until they can be analysed* at the end of a monitoring period. Batch learning methods

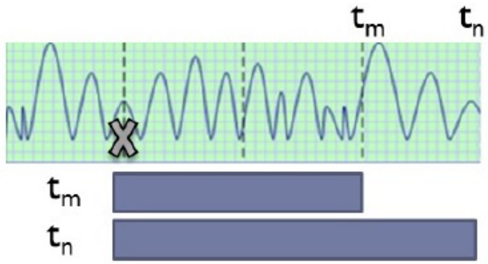
- Can be computationally efficient
- Their accuracy is heavily dependent on a good choice for the training period and the quality of the training data
- Cannot be applied in *streaming environments*, where the measurements arrive as a continuous stream of data
- Cannot be used in *resource constraint devices* to buffer all the measurements
- Cannot identify anomalous points *as they occur*
- Cannot *adapt to changes* in the environment (e.g., drift)

Incremental Learning: Data points are (usually) *analysed once* and there is *no need to buffer the data*. Incremental methods

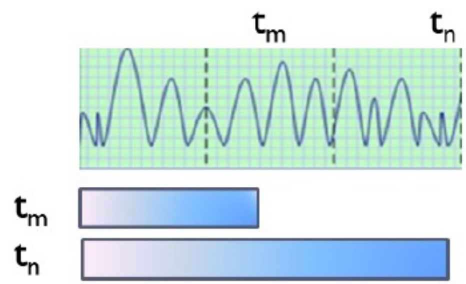
- Start with a set of initial parameters for the selected model and they becomes more accurate as more data arrives

Different Windowing Techniques for Data Streams

- **Landmark windows:** A fixed point in the data stream is defined as a landmark and processing is done over data points between the landmark and the present data point.

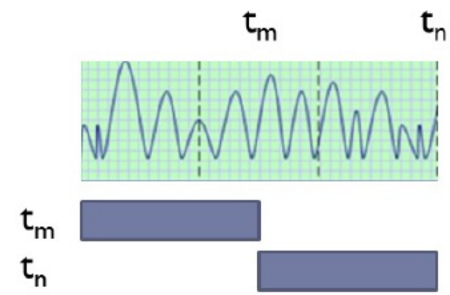


- **Damped windows:** A weight is assigned to each data point in such a way that the old data points are given smaller weights. Therefore, the most recent data points are used with higher weights.

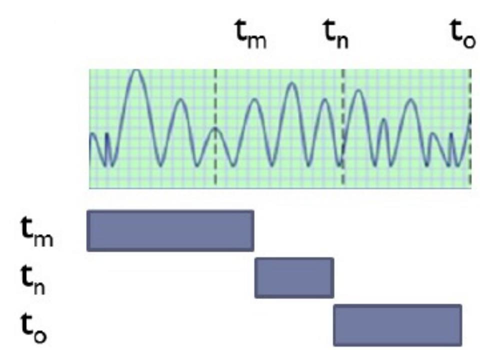


Different Windowing Techniques for Data Streams

- **Sliding windows:** A sliding window size *w* is considered in this technique. It processes the last *w* data points in the stream, while older data points are discarded.



- **Adaptive windows:** The window size *w* would change as the data stream evolves. In this technique, the more the data points evolve, the smaller *w* becomes. In contrast, if data points remain constant, the value of *w* increases.



Streaming Half Space (HS)-Trees [1]

A fast one-class anomaly detector for evolving data streams.

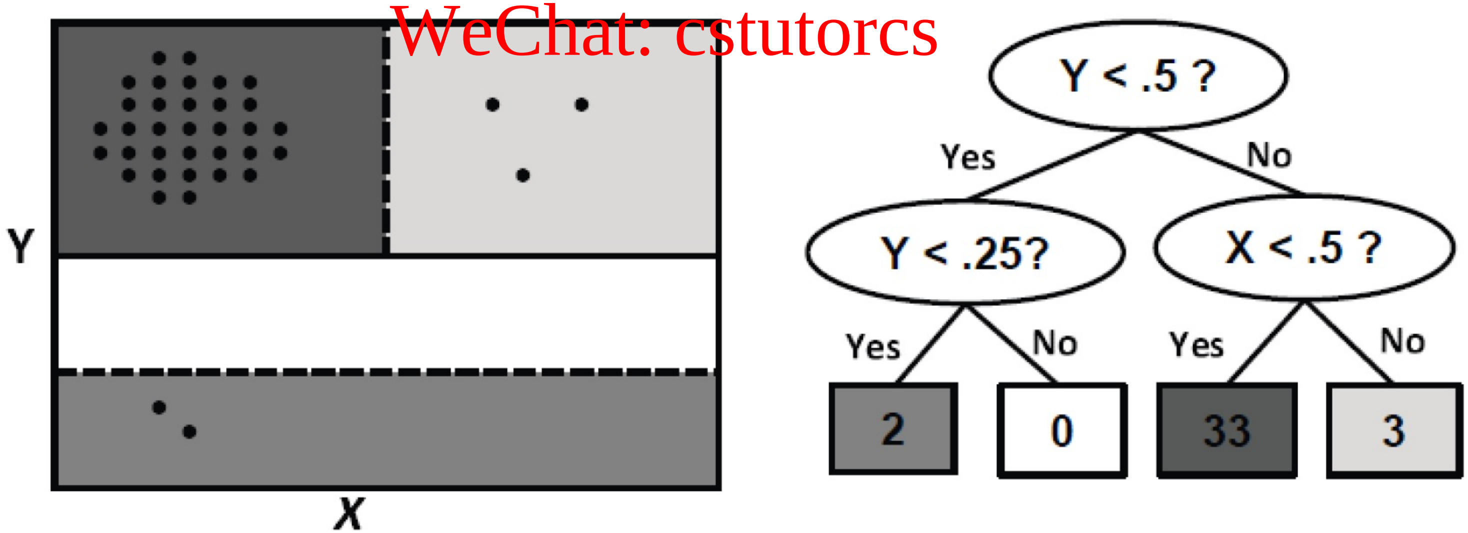
- A random tree model
- Builds tree structure without data
- Detects anomalies in one pass
- Adapts to distribution changes by regular model updates
- Updates model in constant time $O(t(h+\psi))$
- Requires constant amount of memory $O\left(t2^h\right)$

t: number of trees, h: depth of tree, ψ: window size

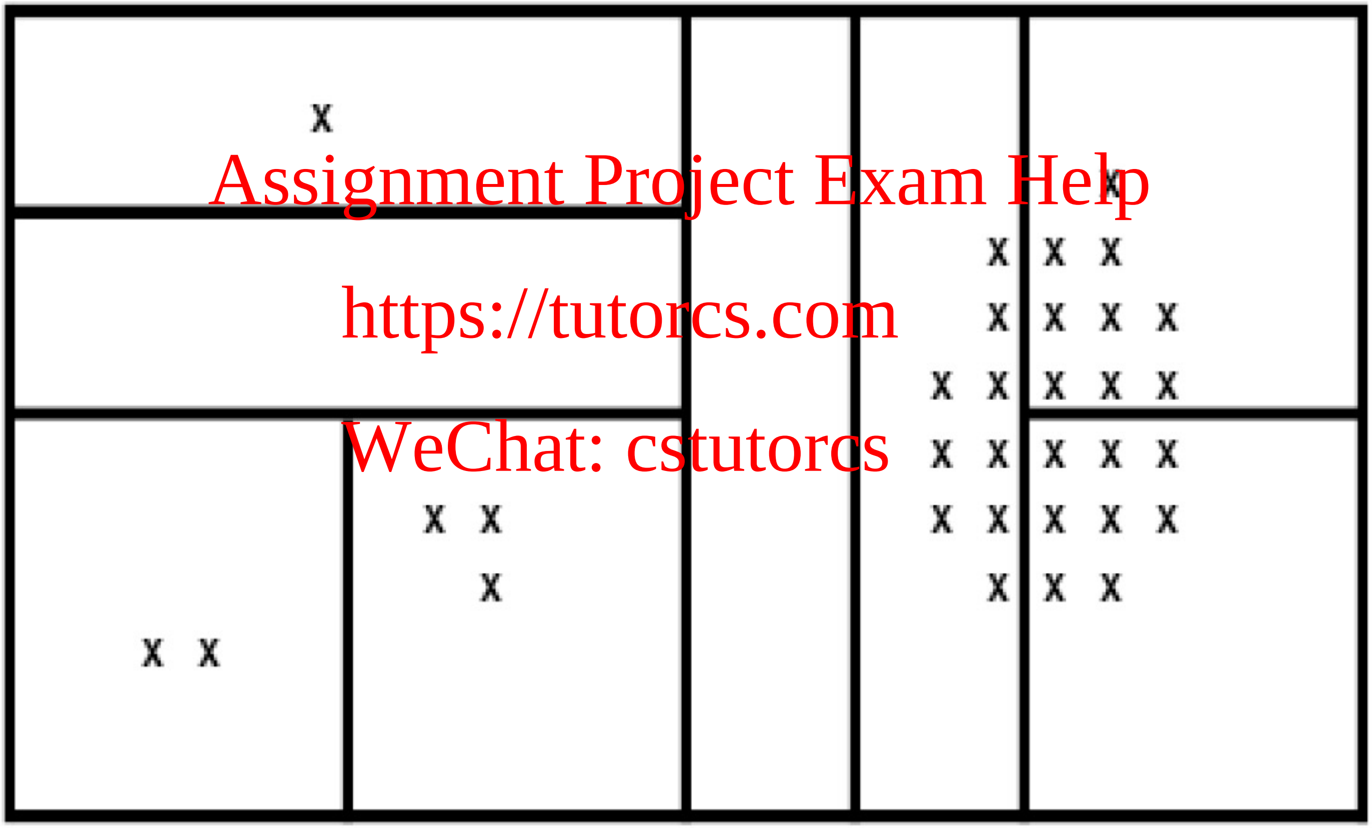
Half Space-Tree

An HS-Tree is a full binary tree, which all leaves are at the same depth *h*.

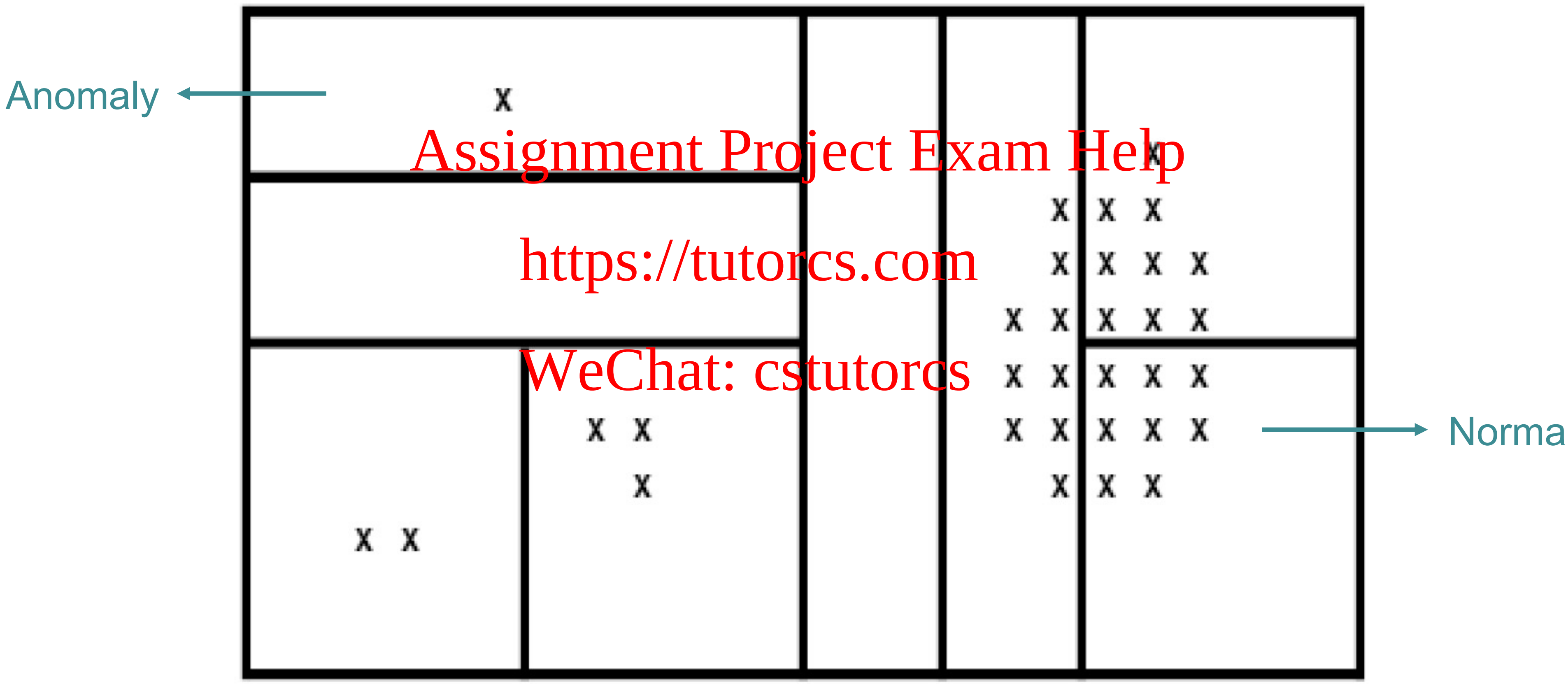
- Randomly select an attribute *d*
- Bisects the space into two half-spaces, using the mid-point of *d* (assume that attributes' ranges are normalised to [0, 1])
- Continue expansion until the maximum depth *h* of all nodes is reached.
- Employs mass as a measure to rank anomalies.



Separating the Regions: HS-Trees



Ranking by Mass



Stream Processing

- Divide data stream into fixed-size windows: $W_1, W_2, \ldots, W_n$
- Each window is a fixed number of sequenced data instances
- ***Initial Learning:*** Train model M_1 using instances in W_1
- ***Subsequent Learning and Anomaly Scoring***

  For each window W_k (where k >1)

    Train model M_k using instances in W_k

    Test instances in W_k using model M_{k-1}

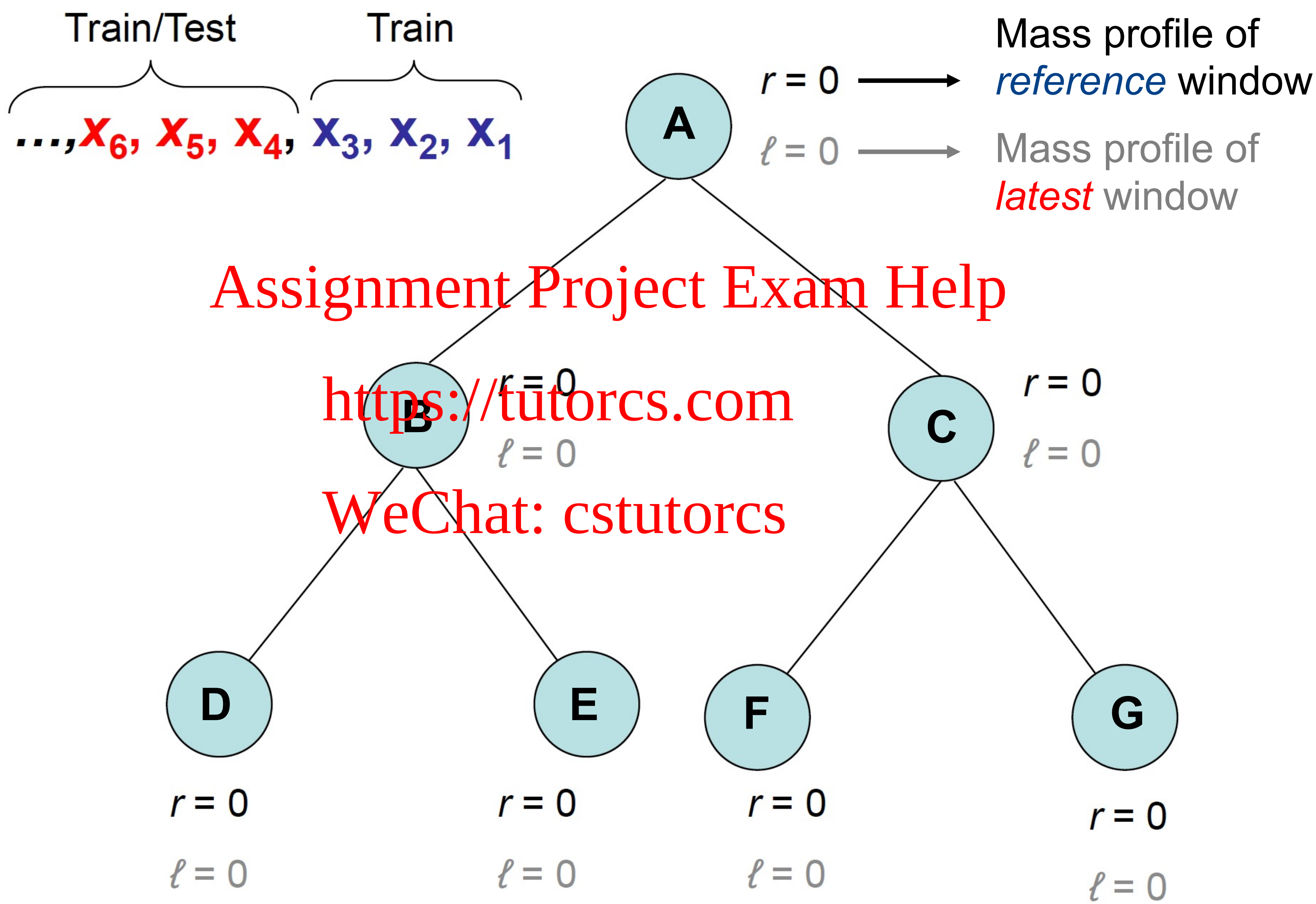
  Next window

Count-based Windows

- Let window size = 3
- Initial stage
- W1: reference window
  - Train HS-Trees and update mass r

$$\overbrace{x_3,\ x_2,\ x_1}^{\mathbf{W1}}$$

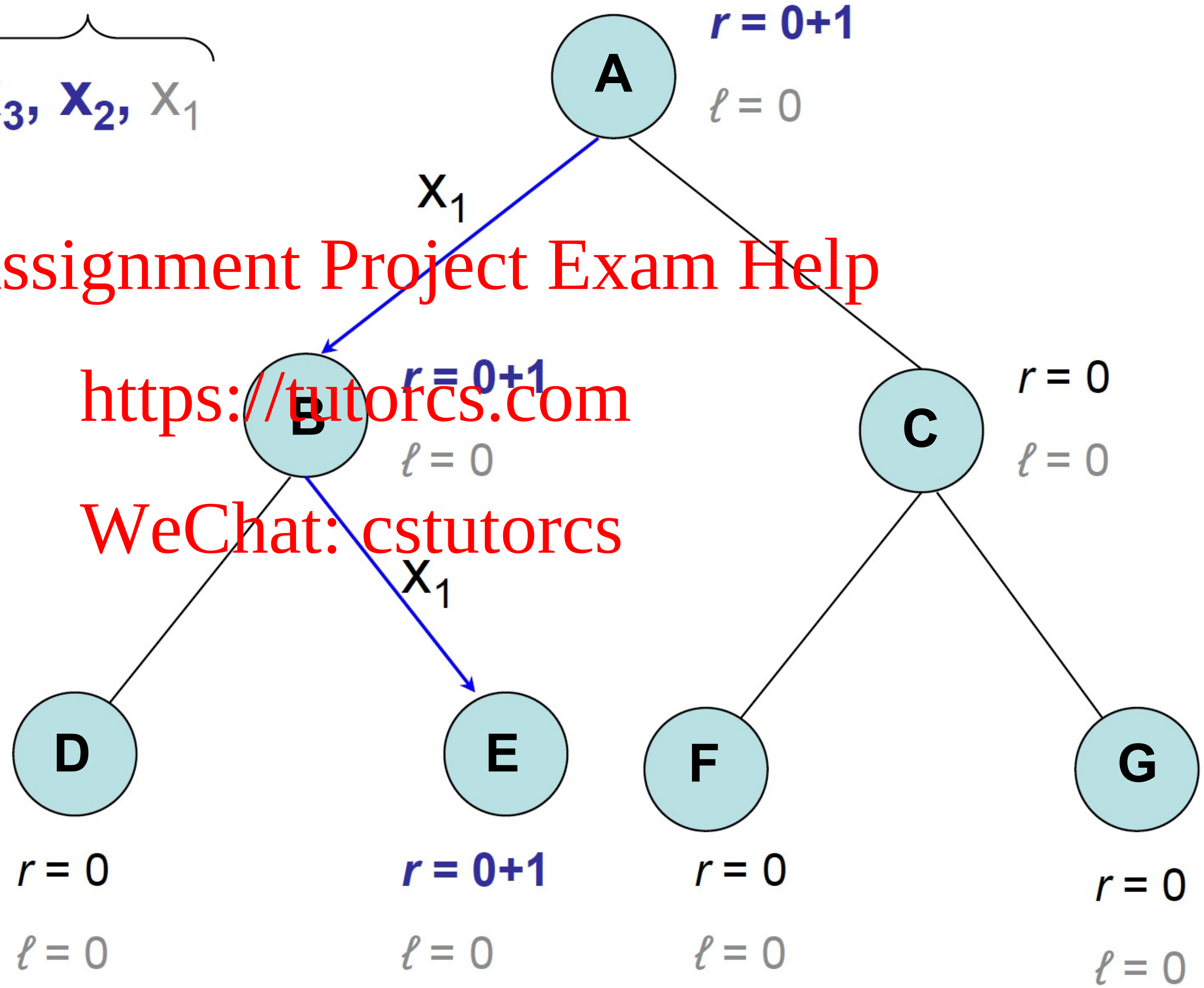
Streaming HS-Trees – Example



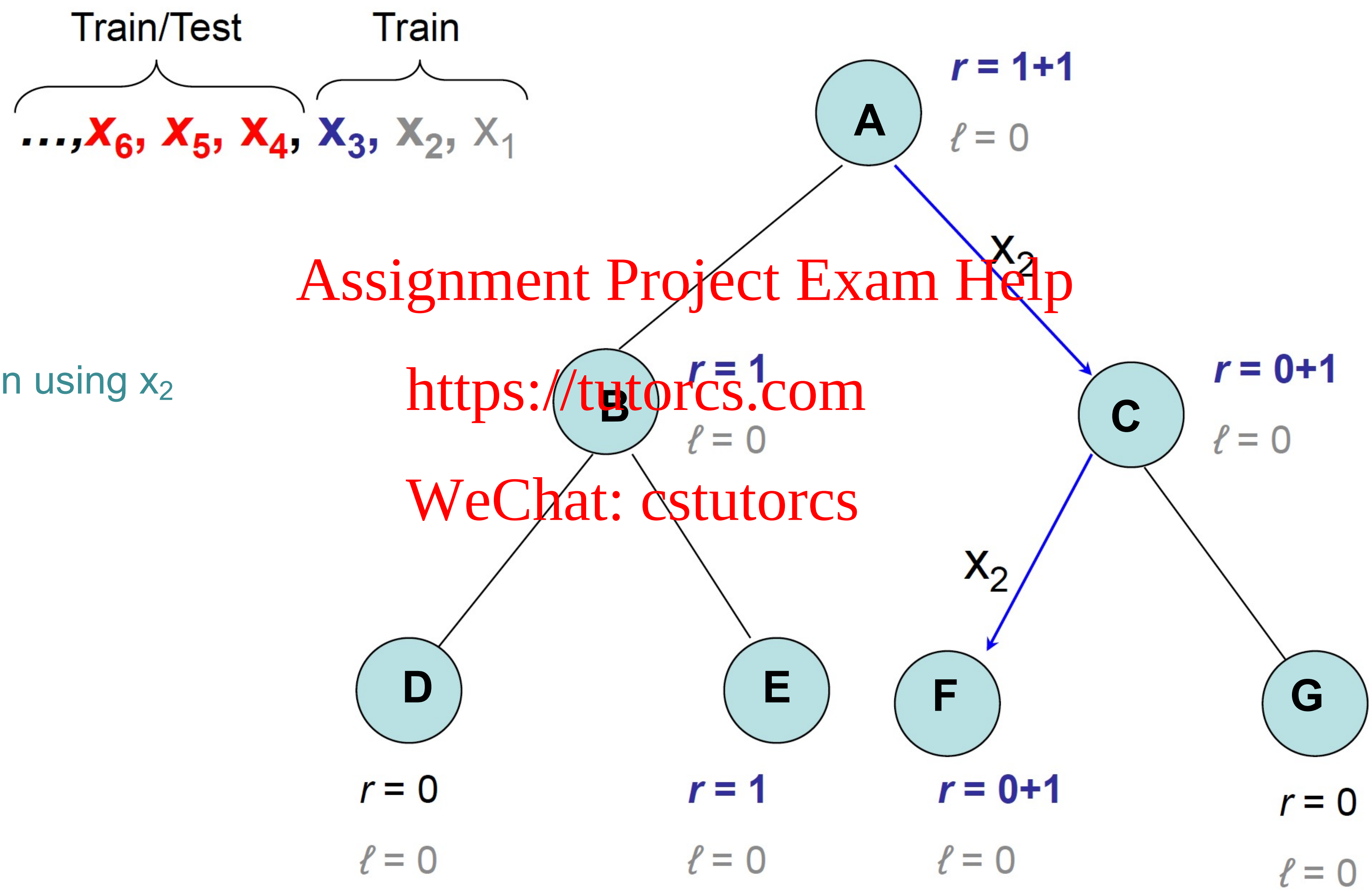
Streaming HS-Trees – Example

$$\overbrace{\dots, x_6, x_5, x_4}^{\text{Train/Test}}, \overbrace{x_3, x_2, x_1}^{\text{Train}}$$

Train using x_1

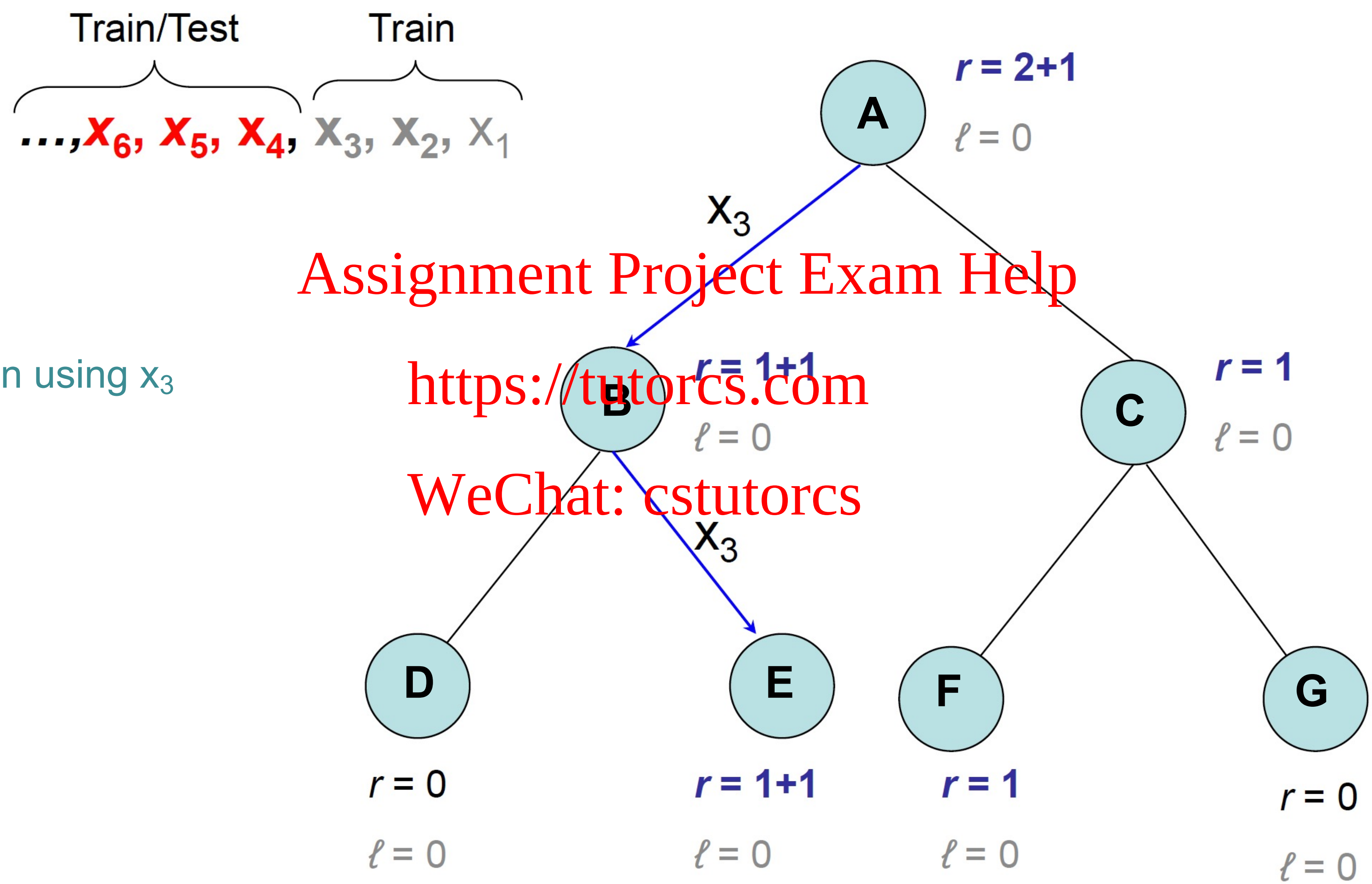


Streaming HS-Trees – Example



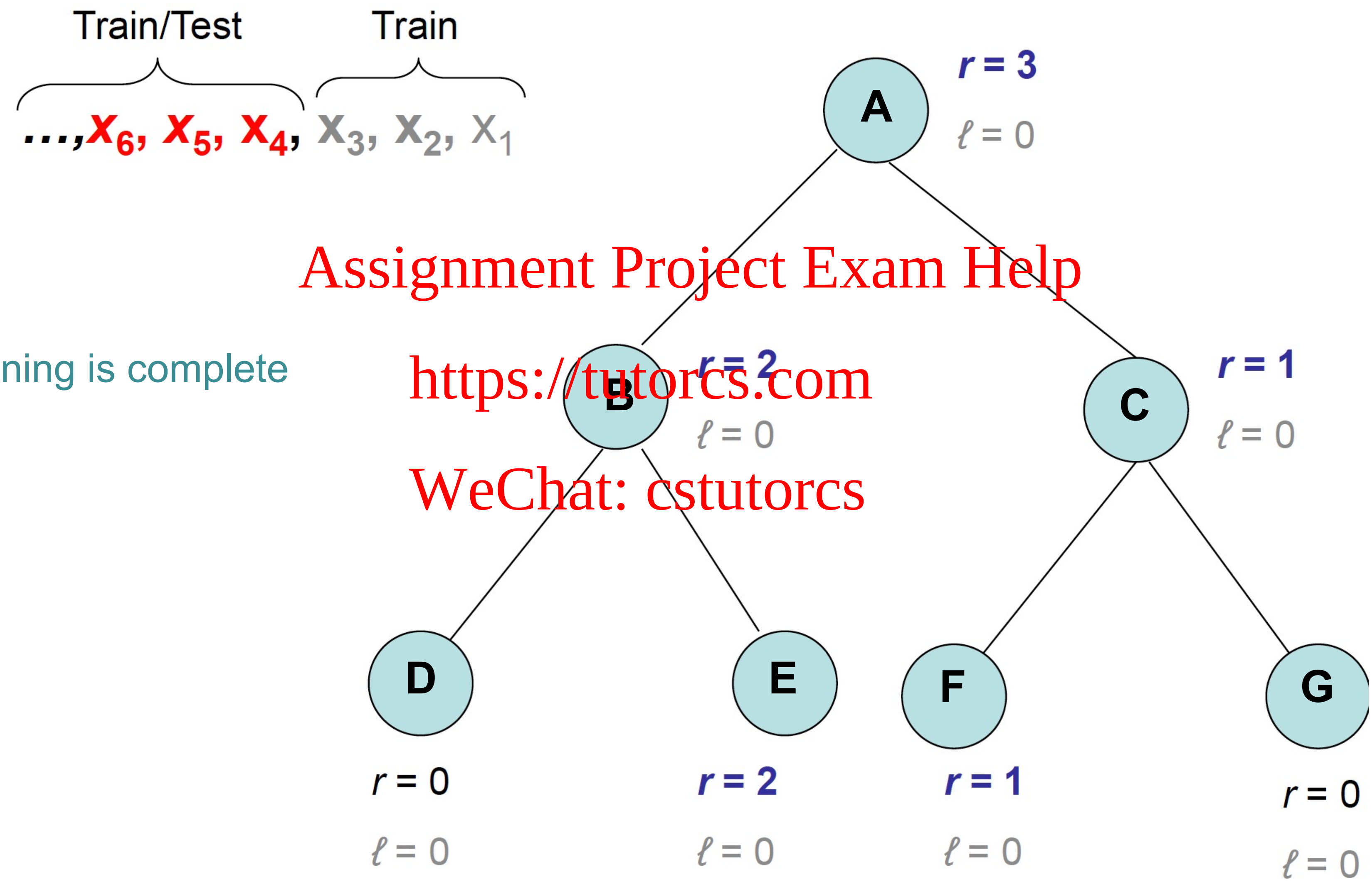
Train using x_2

Streaming HS-Trees – Example



Train using x_3

Streaming HS-Trees – Example



Training is complete

Count-based Windows

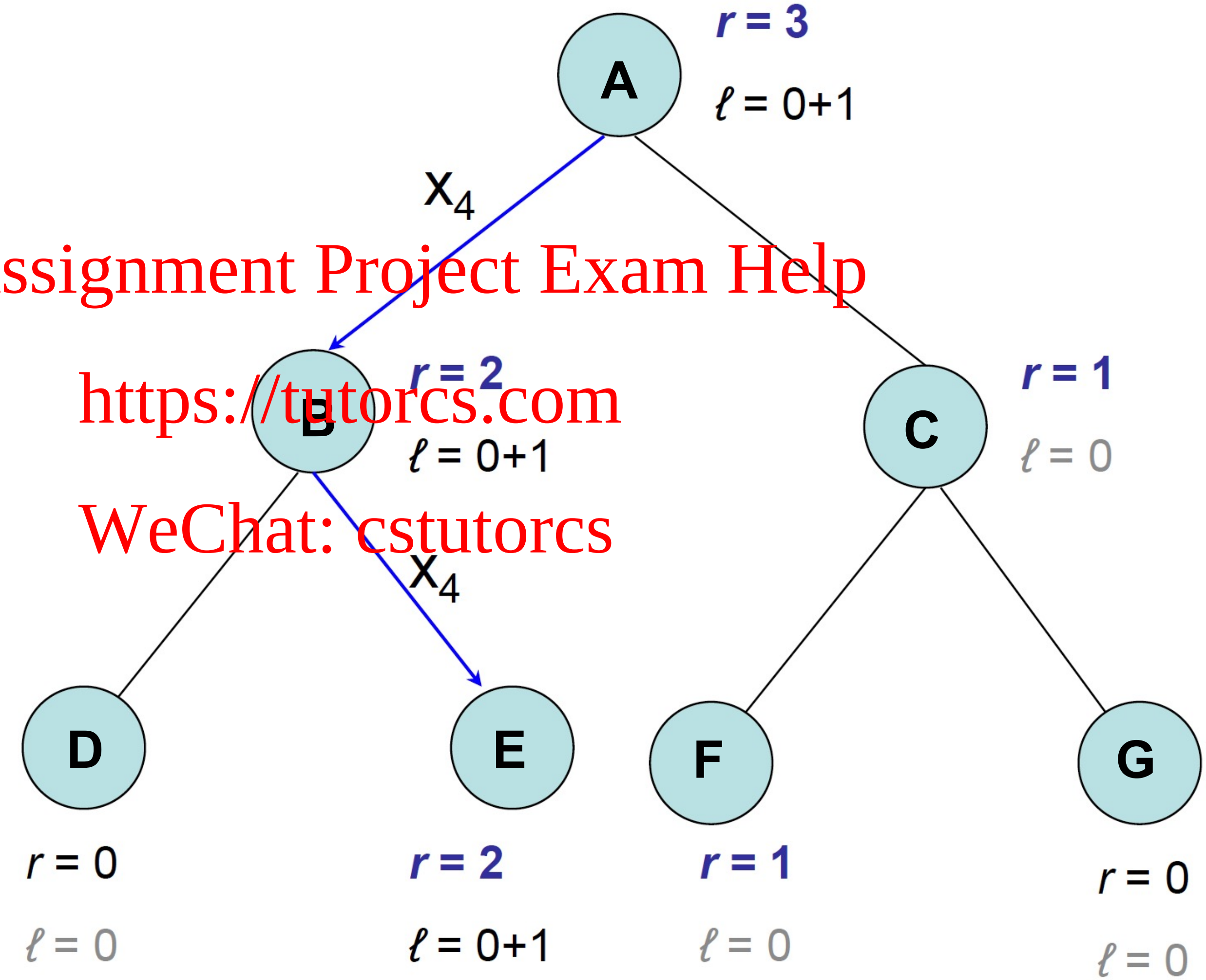
- Let window size = 3
- Initial stage
- W1: reference window
  - Train HS-Trees and update mass r
- W2: latest window
  - Instances in W2 for training HS-Trees (mass ℓ)
  - Instances in W2 for testing HS-Trees (mass r)

$$\overbrace{x_6, x_5, x_4}^{\text{W2}}, \overbrace{x_3, x_2, x_1}^{\text{W1}}$$

Streaming HS-Trees – Example

$\overbrace{\ldots, x_6, x_5, x_4,}^{\text{Train/Test}}$

Train/Test using x_4

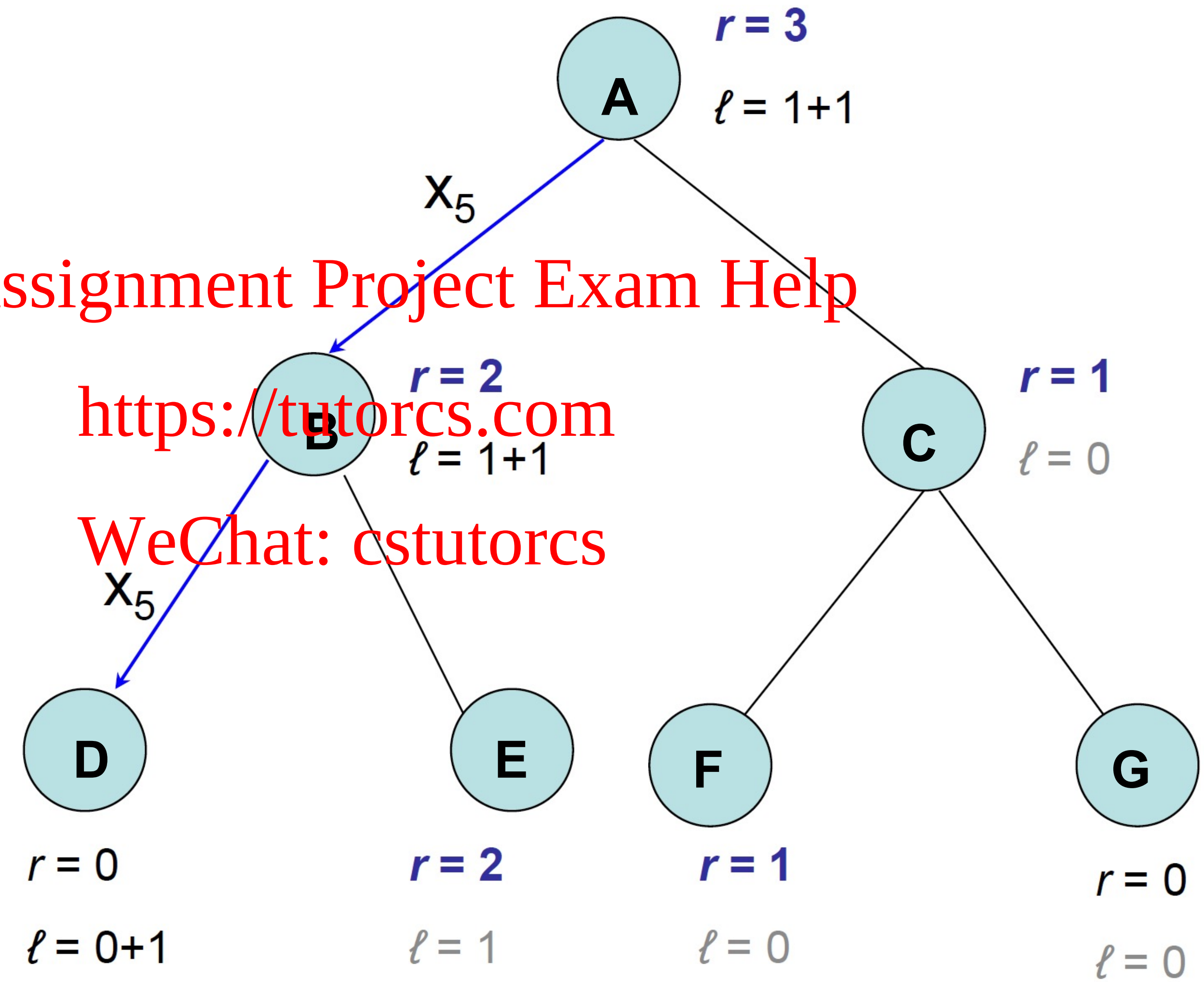


Streaming HS-Trees – Example

Train/Test

$\underbrace{\ldots, x_6, x_5, x_4,}_{}$

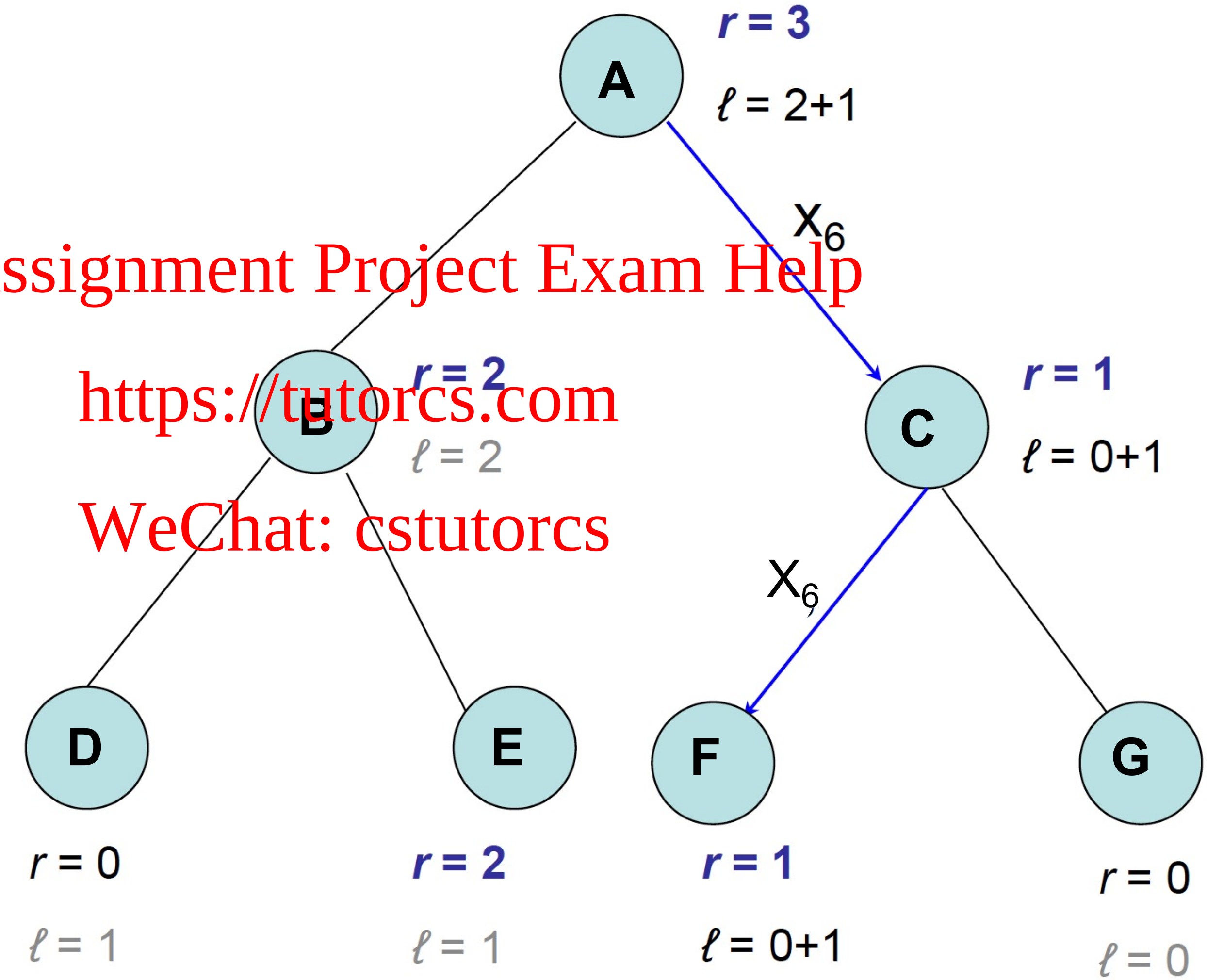
Train/Test using x_5



Streaming HS-Trees – Example

$\overbrace{\ldots, x_6, x_5, x_4,}^{\text{Train/Test}}$

Train/Test using x_6



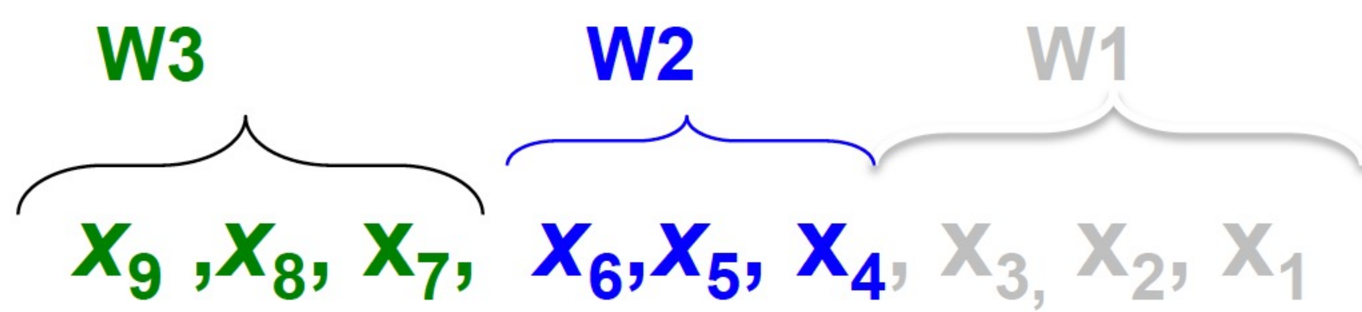
THE UNIVERSITY OF MELBOURNE

Count-based Windows

When all instances in W2 are processed

- Model update occurs
- W2 becomes the new reference window
  - Transfer all mass ℓ values to mass *r* values
  - Reset all mass ℓ values to zero
- W3 becomes the latest window
  - Instances in W3 for training HS-Trees (mass ℓ)
  - Instances in W3 for testing HS-Trees (mass r)

And so on…

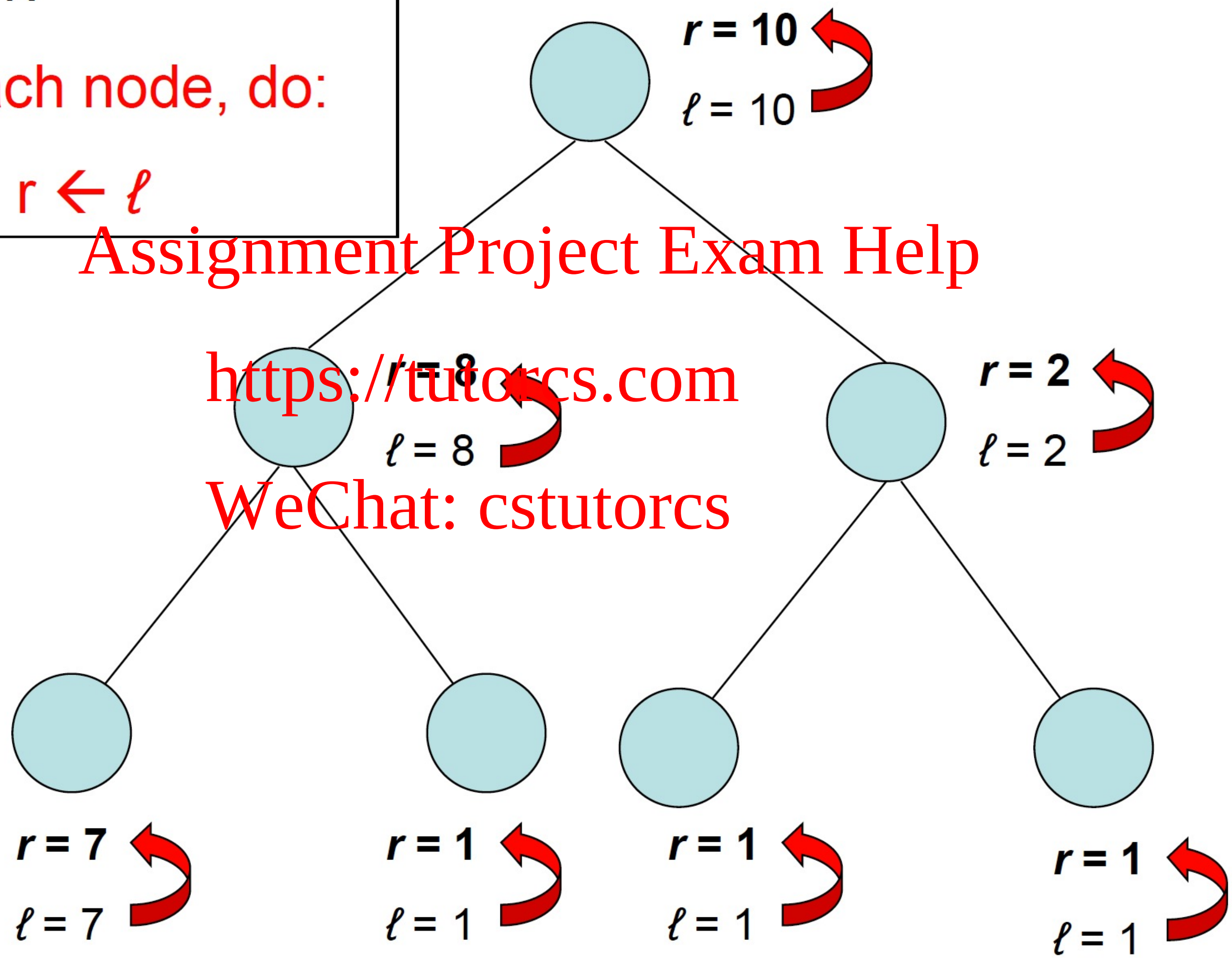


Model Update

Step 1:

for each node, do:

$r \leftarrow \ell$

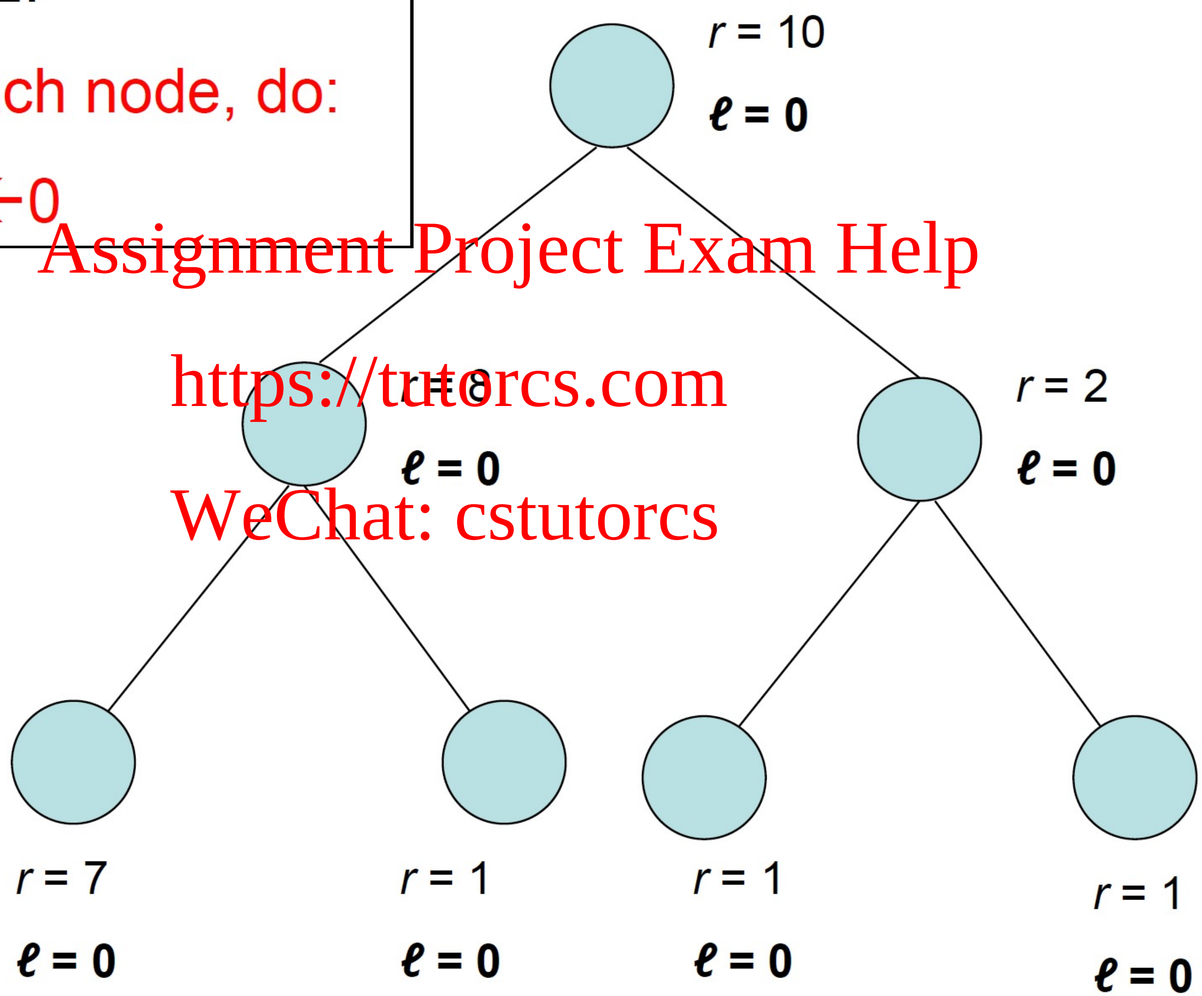


Model Update

Step 2:

for each node, do:

$\ell \leftarrow 0$



Anomaly Score in HS-Tree

- The final score for x is the sum of scores obtained from each HS-Tree in the ensemble

$$anomaly\ score(x) = \sum_{t \in T} Score(x, t_i)$$

$$Score(x, t_i) = Node_r \times 2^{Node_k}$$

r value at node

Depth of node

LOF – Revision

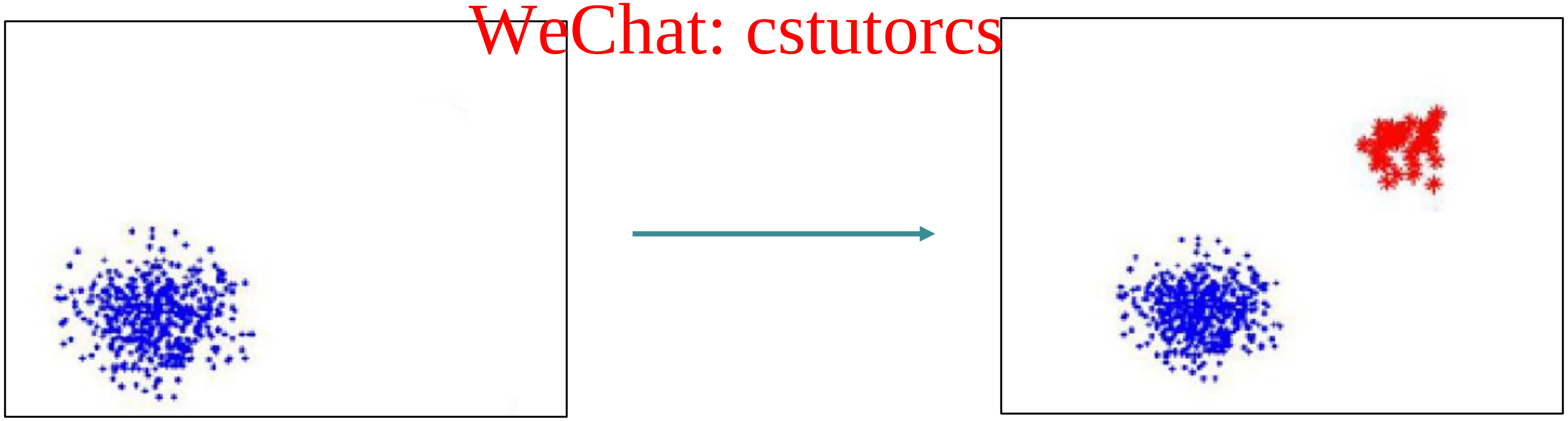
Advantages of LOF for anomaly detection:

- Detects anomalies regardless the data distribution of normal behaviour, since it does not make any assumptions about the distributions of data records.
- Detects anomalies with respect to density of their neighbouring data records; not to the global model.
- Directly applying LOF to data streams would be extremely computationally inefficient and/or very often may lead to incorrect prediction results.

Extending LOF to Data Streams

(i) Periodic LOF. Apply LOF algorithm on the entire data set *periodically* (e.g., after every data block of 1000 data records) or after all the data records are inserted.

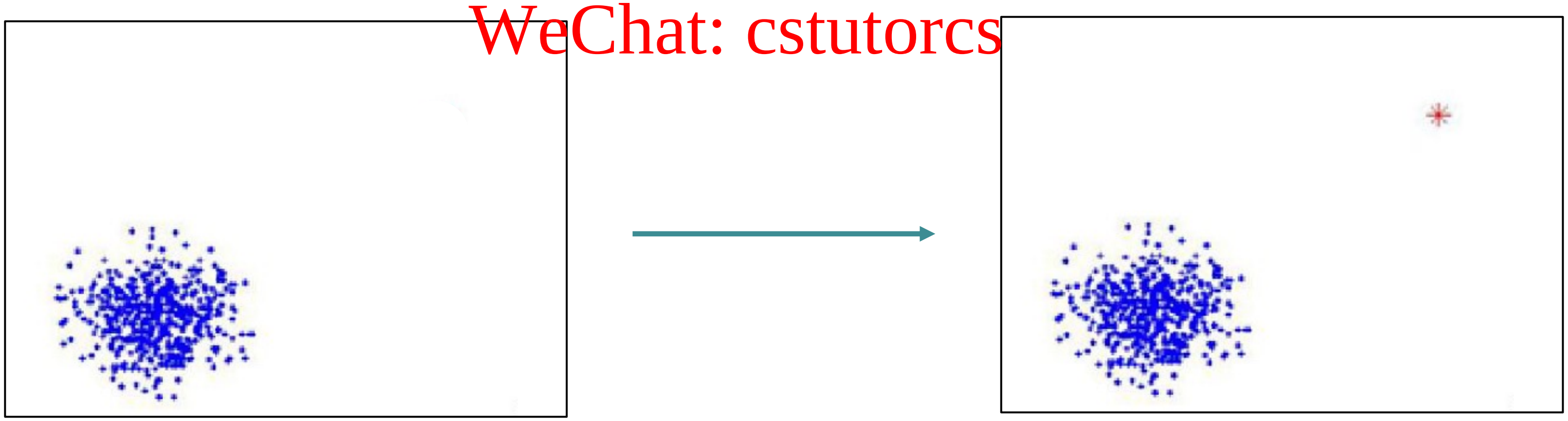
- The major problem of this approach is inability to detect anomalies related to the *beginning of new behaviour* that initially appear within the inserted block.



Extending LOF to Data Streams

(ii) Iterated LOF: Re-apply the static LOF algorithm *every time a new data* record is inserted into the dataset.

- This static LOF algorithm does not suffer from the previous problems, but is extremely computationally expensive.
- Increases LOF's time complexity to $O(n^2 \log n)$, where n is the current number of data records in the data set.



Incremental LOF (iLOF) [2]

Objectives:

- The result of the incremental algorithm must be equivalent to the result of the "batch".
- Time complexity of incremental LOF algorithm has to be comparable to the static LOF algorithm $O(n \log n)$.

Step 1 – Insertion:

- **Insertion of new record**, compute *k-dist*, *reachdist*, *lrd* and *LOF* values of a new point
- **Maintenance**, update *k-dist*, *reachdist*, *lrd* and *LOF* values for *affected existing points*.

Step 2 – Deletion: Delete certain data records (e.g., due to their obsoleteness).

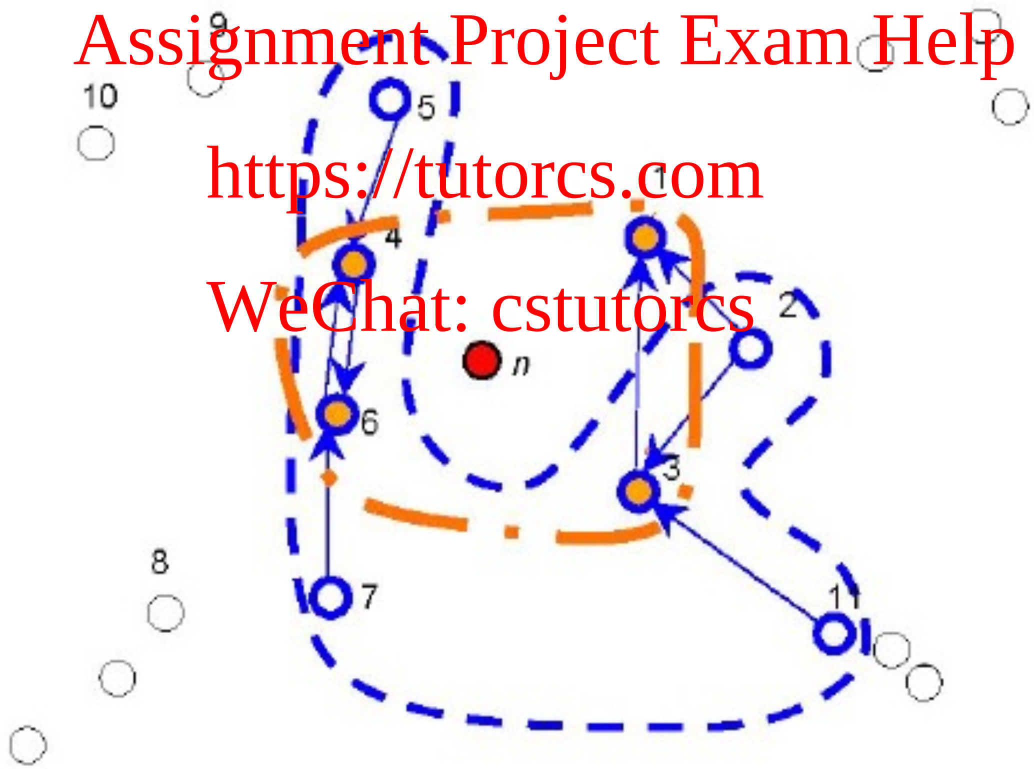
- **Maintenance**, update *k-dist*, *reachdist*, *lrd* and *LOF* values for *affected existing points*.

iLOF – Step 1 (Maintenance)

- **Updating *k-dist*:** Insertion of the point *n* may decrease the *k*-distance of certain neighbouring points, and it can happen only to those points that have the new point *n* in their *k*-neighbourhood (e.g., 2-neighbourhood).
- **Reverse Nearest Neighbour (RNN):** Find all the objects for which the new point *n* is their (*k*-)nearest neighbour.

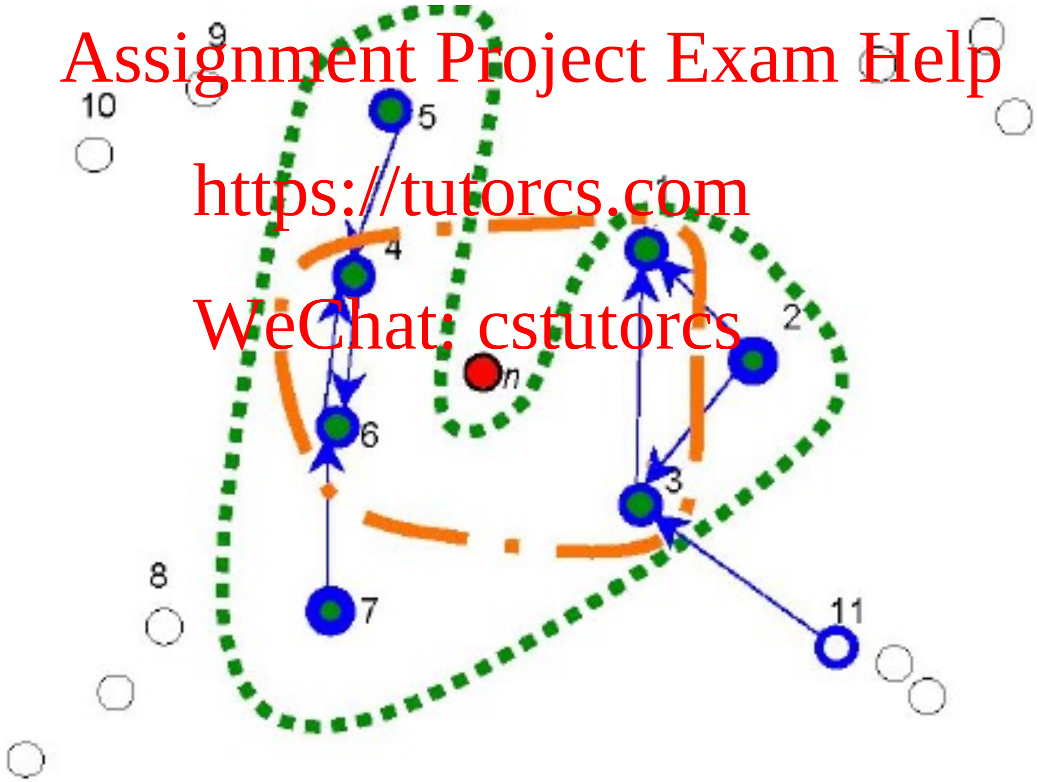
iLOF – Step 1 (Maintenance)

- **Updating *reachdist$_k$*:** When *k*-distance(*p*) changes for a point *p*, *reachdist$_k$(o,p)* will be affected only for points *o* that are in *k*-neighbourhood of the point *p*.



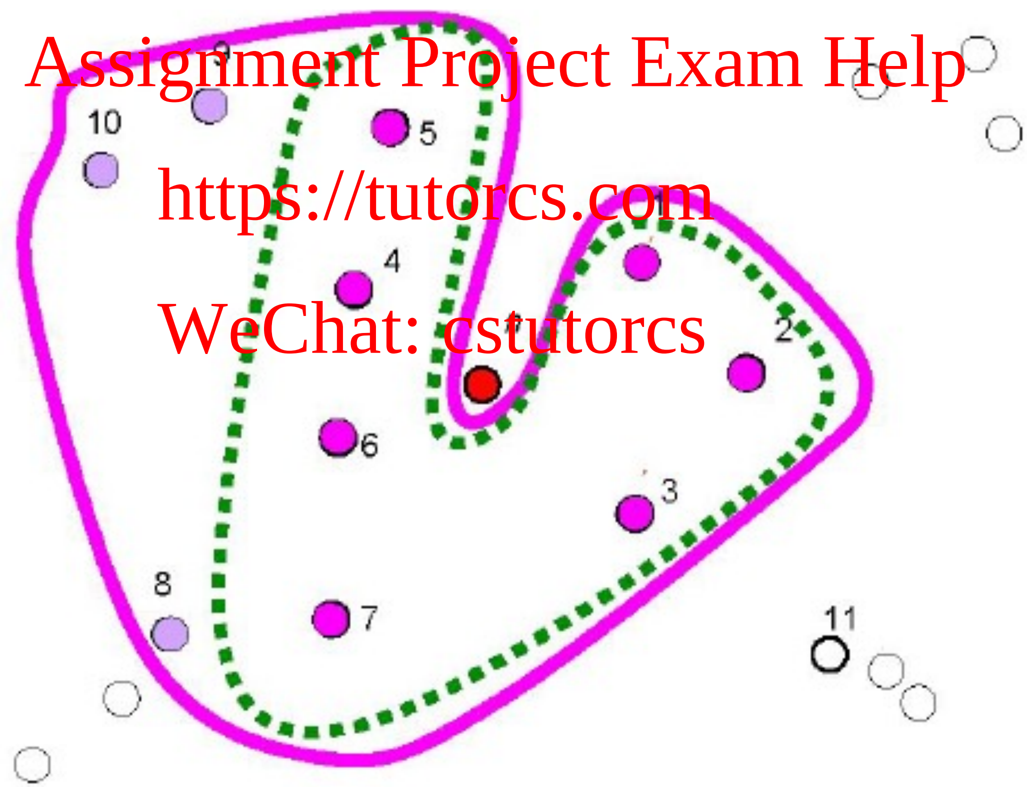
iLOF – Step 1 (Maintenance)

- **Updating lrd:** *lrd* value of a point *p* is affected if:
  - The *k*-neighbourhood of the point *p* changes,
  - *Reachdist* from point *p* to one of its *k*-neighbours changes.



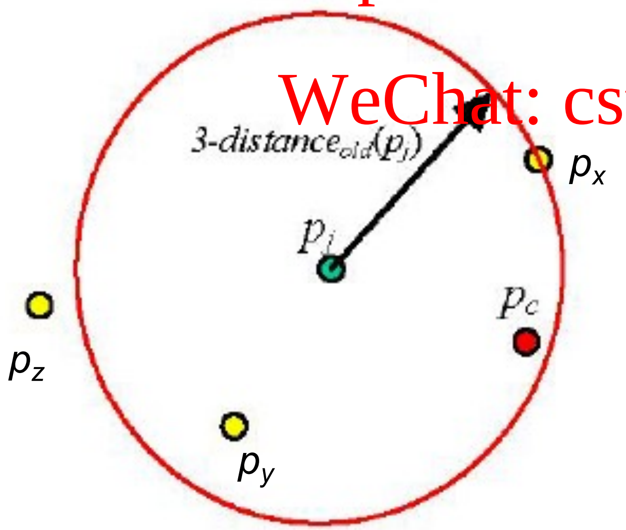
iLOF – Step 1 (Maintenance)

- **Updating *LOF* Values:** LOF values of an existing point *p* should be updated if
  - *lrd(p)* is updated, or
  - *lrd(p)* of one of its *k*-neighbours *p* changes



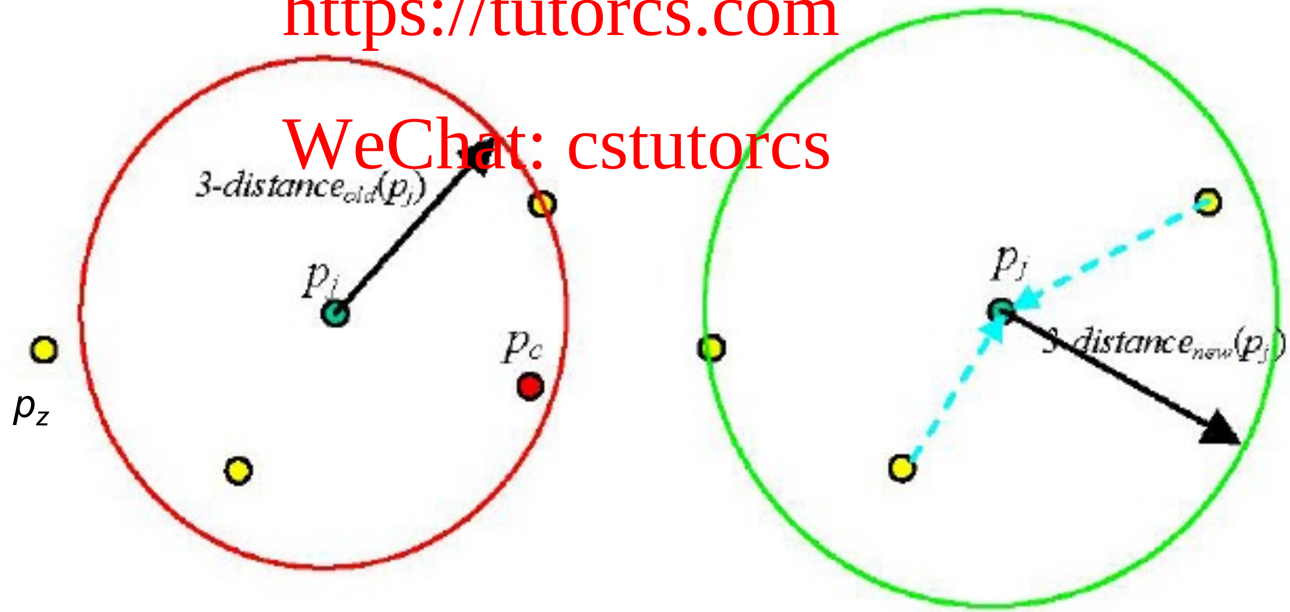
iLOF – Step 2 (Deletion)

- **Updating *k-dist*:** The deletion of each record p_c from the dataset influences the k-distances of its RNN.
  - k-neighbourhood increases for each data record p_j that is in reverse k-nearest neighbourhood of p_c. The new k-distance for p_j becomes equal to its distance to its new k-th nearest neighbour.



iLOF – Step 2 (Deletion)

- **Updating *k-dist*:** The deletion of each record p_c from the dataset influences the k-distances of its RNN.
  - k-neighbourhood increases for each data record p_j that is in reverse k-nearest neighbourhood of p_c. The new k-distance for p_j becomes equal to its distance to its new k-th nearest neighbour.



iLOF – Step 2 (Deletion)

- **Updating *reachdist:*** The reachability distances from p_j's nearest neighbours need to be updated.

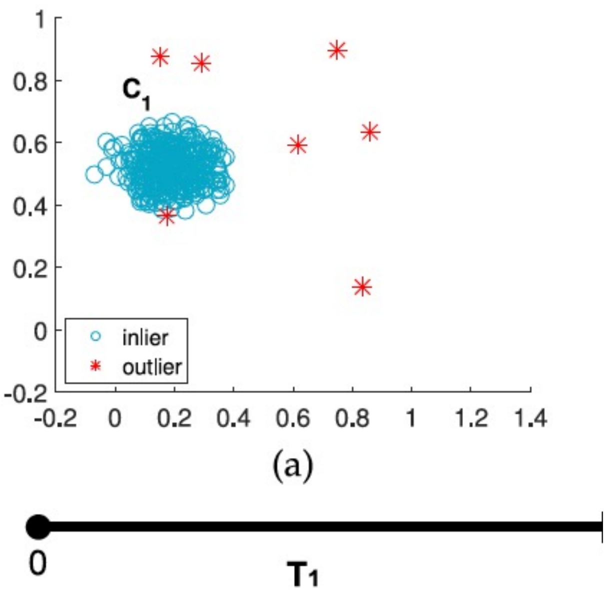
- **Updating *lrd*:** *lrd* value needs to be updated for
  - All points p_j, which *k*-distance is updated.
  - All points p_i, which is in *k*-NN of p_j and p_j is in *k*-NN of p_i.

- **Updating *LOF* Values:** LOF value is updated for
  - All points p_j, which *lrd* value is updated
  - All points p_i, which is in RNN of p_j

Shortcomings of iLOF

Deleting past data points due to memory limitations causes two problems:

- Differentiation between new and old events
- The accuracy will drop by deleting the history

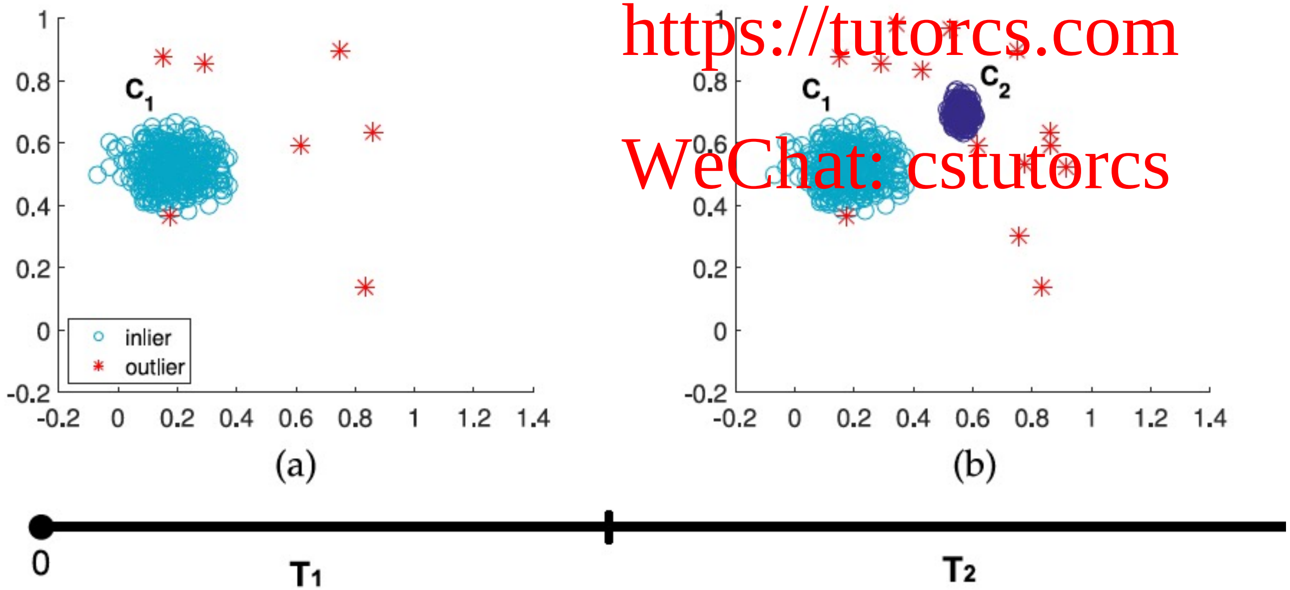


(a)

Shortcomings of iLOF

Deleting past data points due to memory limitations causes two problems:

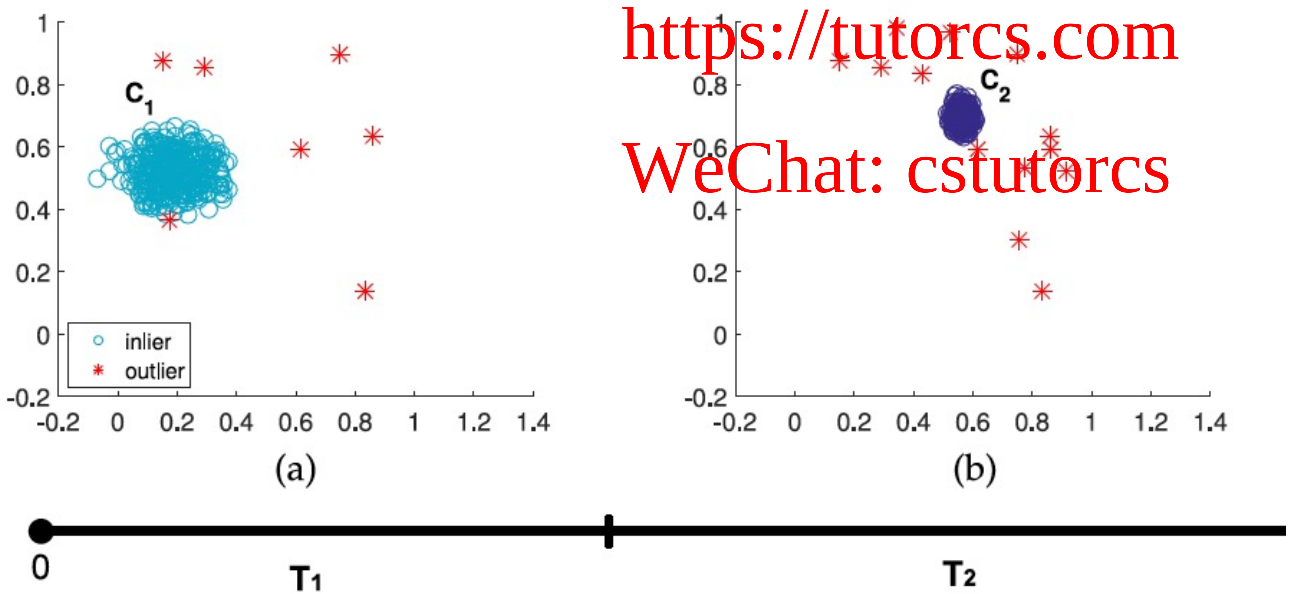
- Differentiation between new and old events
- The accuracy will drop by deleting the history



Shortcomings of iLOF

Deleting past data points due to memory limitations causes two problems:

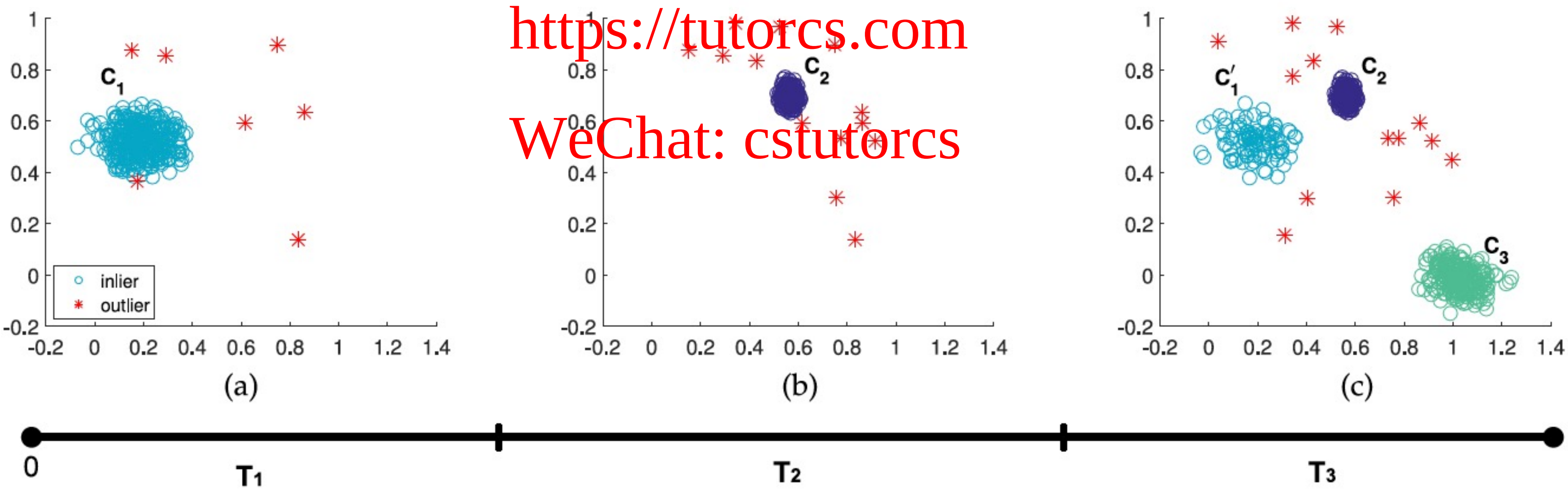
- Differentiation between new and old events
- The accuracy will drop by deleting the history



Shortcomings of iLOF

Deleting past data points due to memory limitations causes two problems:

- Differentiation between new and old events
- The accuracy will drop by deleting the history



Memory Efficient Incremental Local Outlier (MiLOF) Detection [3]

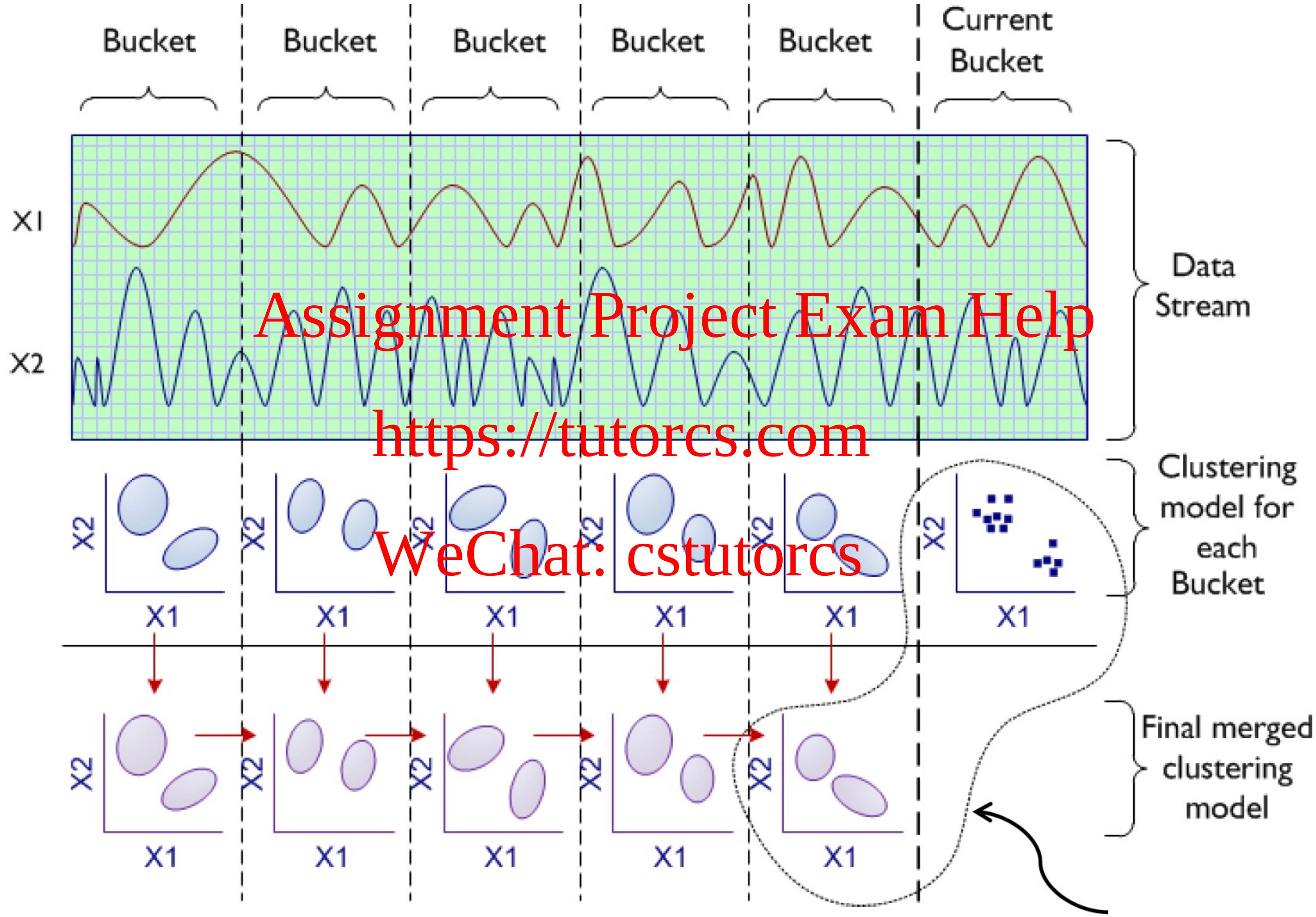
Objective: Assign an LOF value to a point p_t, under the constraint that the available memory stores only a fraction $m \ll n$ of the n points that have been observed up to time T.

- Need to choose a strategy to summarize the previous data points so that the LOF values of new points can be calculated.

MiLOF Phases:

- Summarization
- Merging
- Revised Insertion

Three Phases of MiLOF – Framework



Compute outlier score of new point using the latest data points and latest merged clustering model

Three Phases of MiLOF

Phase 1 – Summarization:

Build a summary over the past data points along with their corresponding values (*k-dist*, *lrd* and *LOF*), and deleting them from memory.

- Every bucket data points are summarized and cluster centres are generated using *k*-means clustering

Notations:

- C: points arriving at time T
- Partition C into m clusters $C = \{C_1 \cup C_2 \cup \cdots \cup C_m\}$, with cluster centres $V = \{v_1, v_2, \ldots, v_m\}$

MiLOF Measures

- ***k*-dist** of a cluster centre $v_i \in V$

$$kdist(v_i) = \frac{\sum_{p \in C_i} kdist\ (p)}{|C_i|}$$

Number of points in C_i

- **lrd** of a cluster centre $v_i \in V$

$$lrd_k(v_i) = \frac{\sum_{p \in C_i} lrd_k(p)}{|C_i|}$$

- **LOF** of a cluster centre $v_i \in V$

$$LOF_k(v_i) = \frac{\sum_{p \in C_i} LOF_k(p)}{|C_i|}$$

Three Phases of MiLOF

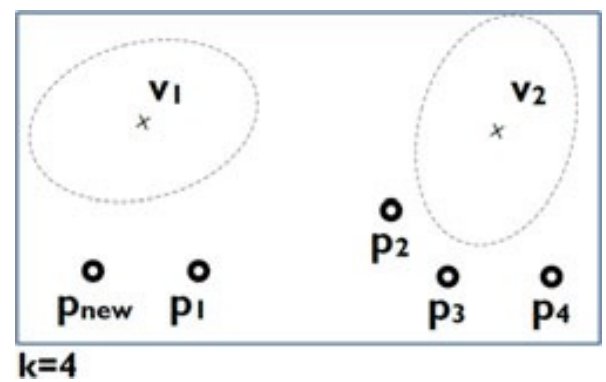
Phase 2 – Merging:

Merge the clusters with existing clusters to maintain a single set of cluster centres by the anomaly detection framework after each step.

- Using a weighted clustering algorithm (weighted *k*-means) and cluster the cluster centres
- Cluster centre's weight is equal to the *number of data points* in that cluster

Phase 3 – Revised Insertion:

- Compute LOF value of the new incoming data point *p,* w.r.t. both the recent *data points* and *cluster centres.*
  - If a cluster centre is the i^{th} NN of *p*, we stop searching for the rest of the nearest neighbours.
- Update the $kdist$, *reachdist, lrd* and *LOF* values for the existing data points



Summary

- What are different windowing techniques for data streams?
- How to apply tree based anomaly detection methods to data streams?
- How to extend LOF for incremental learning while maintaining its performance?

Next: Anomaly Detection Using Support Vector Machine

THE UNIVERSITY OF MELBOURNE

References

1. Swee Chuan Tan, Kai Ming Ting, Tony Fei Liu, “Fast Anomaly Detection for Streaming Data”, International Joint Conference on Artificial Intelligence (IJCAI), 2011
   – https://github.com/yli96/HSTree
2. Dragoljub Pokrajac, Aleksandar Lazarevic, Longin Jan Latecki, “Incremental Local Outlier Detection for Data Streams”, IEEE Symposium on Computational Intelligence and Data Mining, 2007
3. Mahsa Salehi, Christopher Leckie, James C. Bezdek, Tharshan Vaithianathan, Xuyun Zhang, “Fast Memory Efficient Local Outlier Detection in Data Streams”, IEEE Transactions on Knowledge and Data Engineering (TKDE), 2016