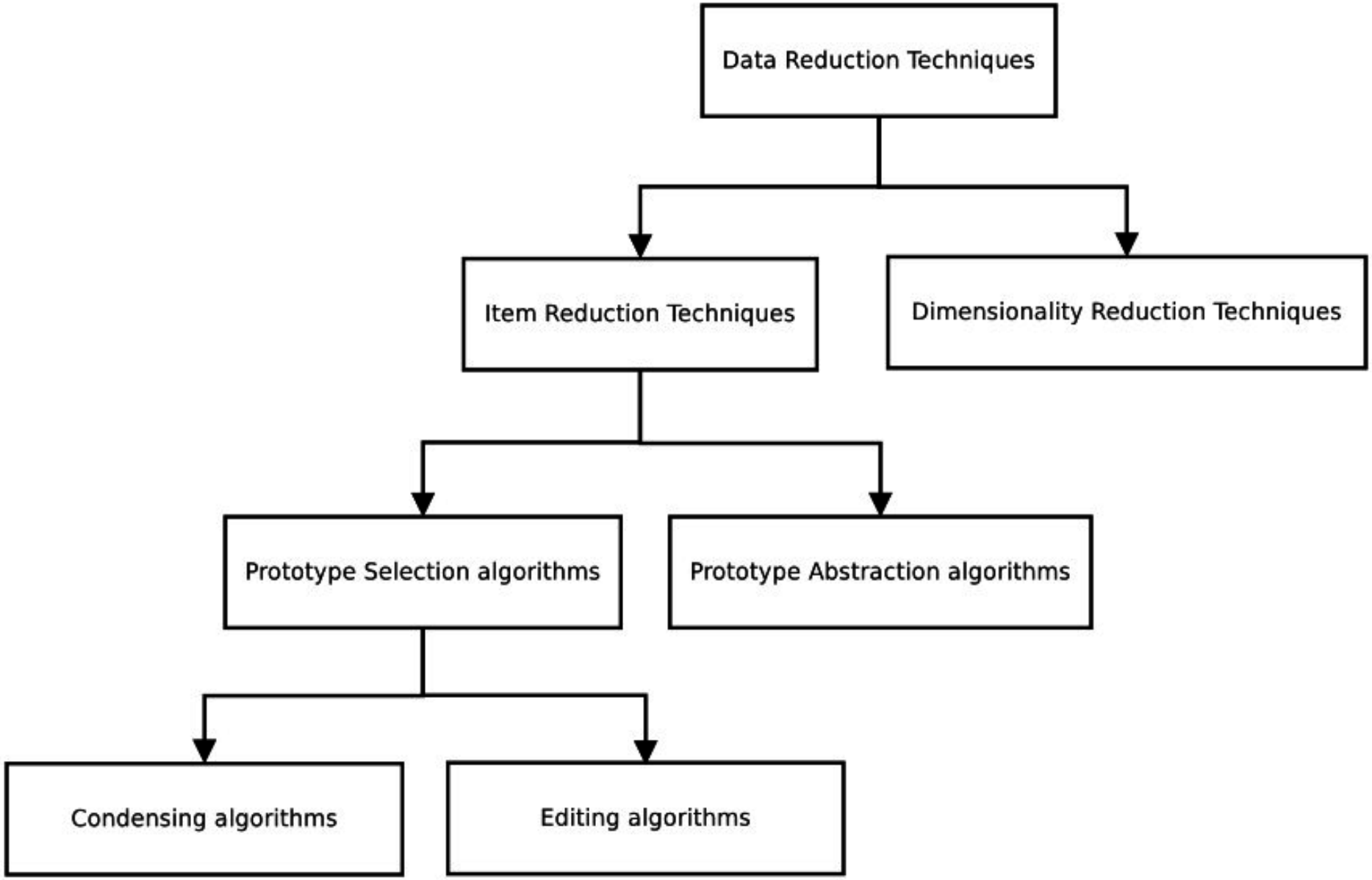
k-NN Classification through Data Reduction

Stefanos Ougiaroglou
stoug@uom.gr



PhD student
Information Management lab
Dept. of Applied Informatics, UoM

Data Reduction Techniques (1/6)



Data Reduction Techniques (2/6)

Condensing and Prototype Abstraction (PA) algorithms

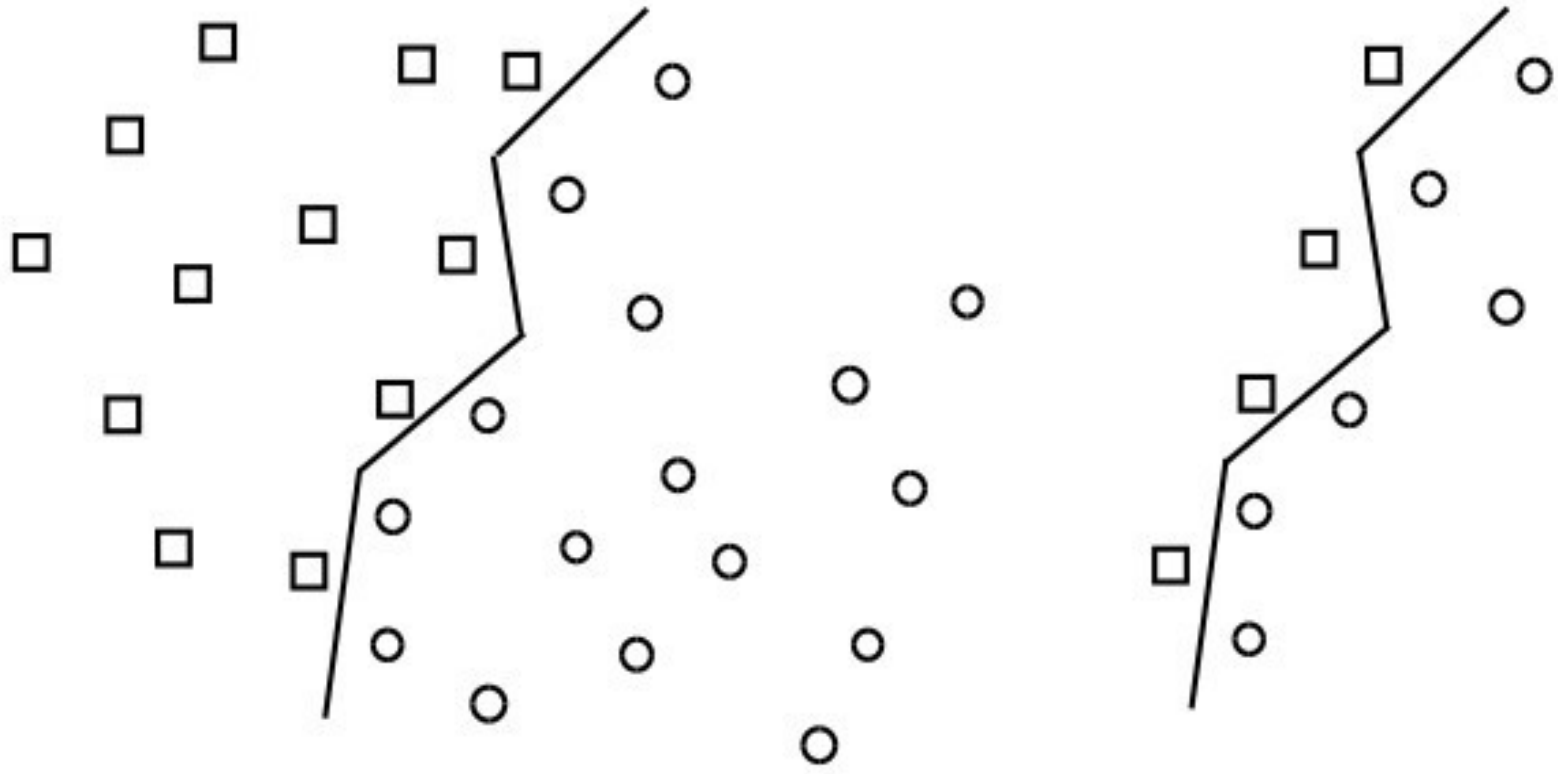
- They deal with the drawbacks of high computational cost and high storage requirements by building a small representative set **(condensing set)** of the training data

- Condensing algorithms **select** and PA algorithms **generate** prototypes

- The idea is to apply *k*-NN on this set attempting to achieve as high accuracy as when using the initial training data at much lower cost and storage requirements

Data Reduction Techniques (3/6)

PA & condensing algorithms:



(a) Training set

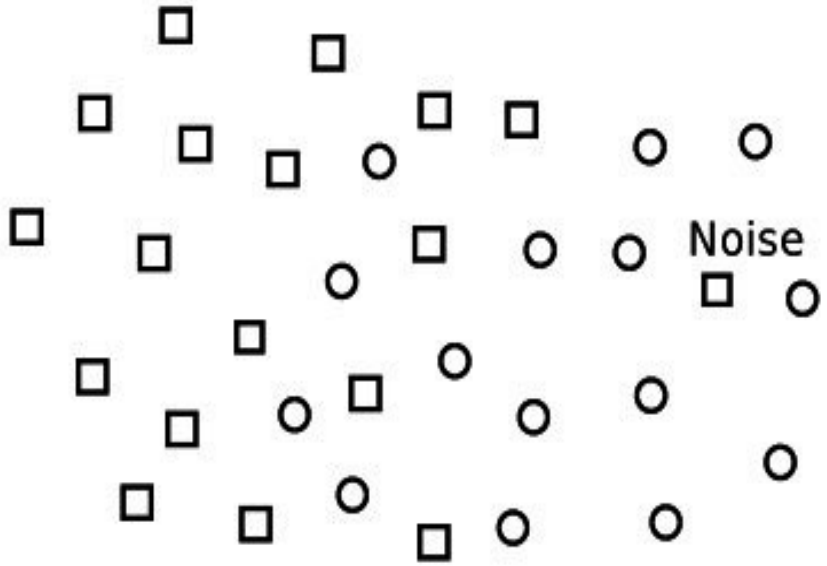
(b) Condensing set

Data Reduction Techniques (4/6)

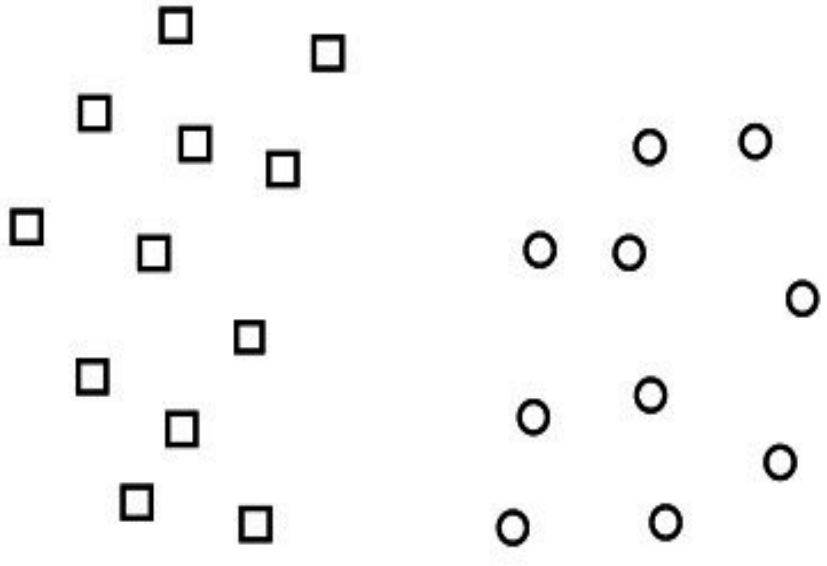
Editing algorithms

- They aim to improve accuracy rather than achieve high reduction rates
- They remove noisy and mislabeled items and smooth the decision boundaries. Ideally, they build an a set without overlaps between the classes
- The reduction rates of PA and condensing algorithms depend on the level of noise in the training data
- Editing has a double goal: accuracy improvement and effective application of PA and condensing algorithms

Data Reduction Techniques (5/6)

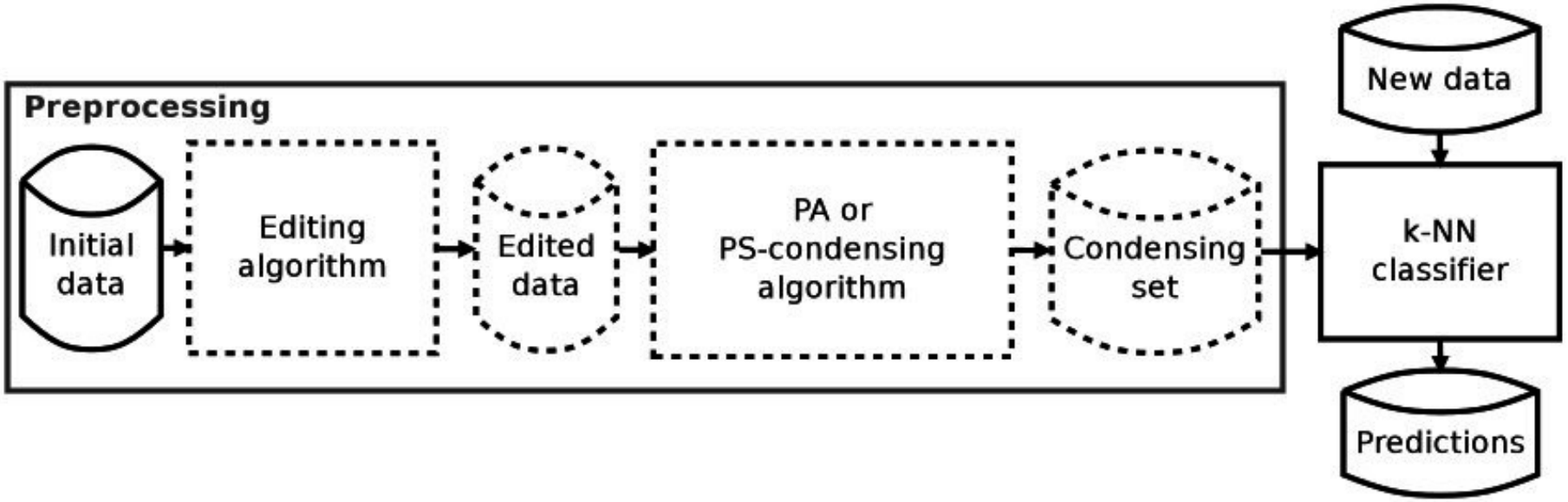


(a) Initial training set



(b) Edited set

Data Reduction Techniques (6/6)



Data Reduction Techniques

DRT performance evaluation criteria

- Classification accuracy
- Reduction rate
- Preprocessing cost

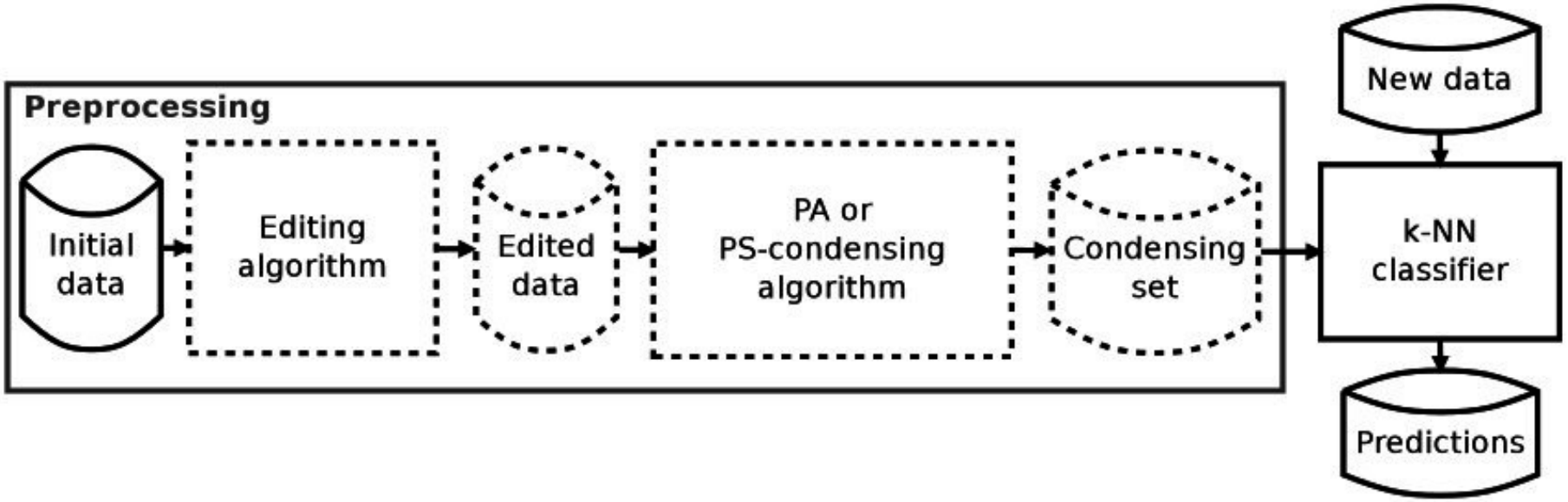
Editing algorithms for noise removal

Stefanos Ougiaroglou
stoug@uom.gr



PhD student
Information Management lab
Dept. of Applied Informatics, UoM

Data Reduction Techniques (6/6)



Edited Nearest Neighbor rule (1/2)

ENN-rule (or Wilson's editing)

- The first editing algorithm and the basis of all other editing algorithms
- A training item is considered as noise and removed if its class label is different from the majority of its *k* nearest neighbours

Drawbacks

- It is parametric algorithm: *k* should be adjusted through time-consuming trial-end-error procedures
- High preprocessing cost: ENN-rule needs to compute: $\frac{N*(N-1)}{2}$ distances

Edited Nearest Neighbor rule (2/2)

Algorithm ENN-rule

Input: TS, k
Output: ES

1: $ES \leftarrow TS$
2: **for** each $x \in TS$ **do**
3: $NNs \leftarrow$ find the k nearest to x neighbors in $TS - \{x\}$
4: $majorClass \leftarrow$ find the most common class of NNs
5: **if** $x_{class} \neq majorClass$ **then**
6: $ES \leftarrow ES - \{x\}$
7: **end if**
8: **end for**
9: **return** ES

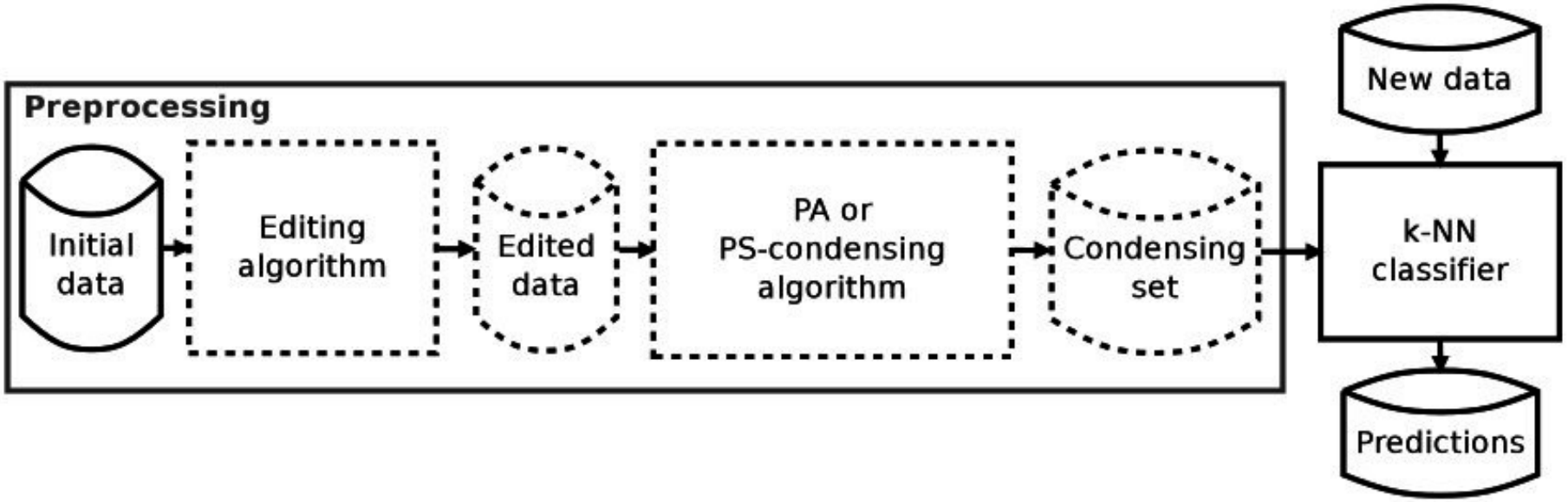
Prototype Abstraction and Condensing Algorithms

Stefanos Ougiaroglou
stoug@uom.gr



PhD student
Information Management lab
Dept. of Applied Informatics, UoM

Data Reduction Techniques (6/6)



CNN-rule (1/2)

Algorithm CNN-rule

Input: TS **Output:** CS

1: $CS \leftarrow \varnothing$
2: pick an item of TS and move it to CS
3: **repeat**
4: $stop \leftarrow TRUE$
5: **for** each $x \in TS$ **do**
6: $NN \leftarrow$ Nearest Neighbour of x in CS
7: **if** $NN_{class} \neq x_{class}$ **then**
8: $CS \leftarrow CS \cup x$
9: $TS \leftarrow TS - x$
10: $stop \leftarrow FALSE$
11: **end if**
12: **end for**
13: **until** $stop == TRUE$ {no move during a pass of TS}
14: discard TS
15: **return** CS

CNN-rule (2/2)

The main idea behind CNN-rule is that if an item is misclassified, it is close to a border area and so it must be placed into the CS

CNN-rule ensures that all removed TS items can be correctly classified by the content of CS

Advantages: Non parametric

Drawback: Order dependent
Memory-based
non-incremental

IB2 (1/2)

Algorithm IB2

Input: TS **Output:** CS

1: $CS \leftarrow \varnothing$
2: pick an item of TS and move it to CS
3: **for** each $x \in TS$ **do**
4: $\quad NN \leftarrow$ Nearest Neighbour of x in CS
5: $\quad$ **if** $NN_{class} \neq x_{class}$ **then**
6: $\quad\quad CS \leftarrow CS \cup x$
7: $\quad$ **end if**
8: $\quad TS \leftarrow TS - x$
9: **end for**
10: **return** CS

IB2 (2/2)

IB2 is a fast one-pass version of CNN-rule

Like CNN-rule:

- IB2 is non-parametric
- IB2 is order dependent

Contrary to CNN-rule:

- IB2 does not ensure that all discarded items can be correctly classified by the CS
- IB2 builds its condensing set **incrementally** (appropriate for dynamic/streaming environments)
- IB2 does not require that all training data reside into the main memory

Chen's algorithm (1/3)

It retrieves the most distant items, x and y in the training set

Then, it divides the training set into two subsets: items that lie closer to x are placed in Sx whereas items that lie closer to y are placed in Sy

It proceeds by selecting to divide non-homogeneous subsets. The non-homogeneous subset with the largest diameter is divided first

If all subsets are homogeneous, it continues by dividing the homogeneous subsets

This procedure continues until the number of subsets becomes equal to a user specified value.

For each created subset S, it averages the items of S and creates a mean item that is assigned the label of the majority class in S. The mean items created constitute the final condensing set

Chen's algorithm (2/3)

Algorithm 6 CJA

Input: TS, n
Output: CS

1: $S \leftarrow \varnothing$
2: add(S, TS)
3: **for** $i = 2$ to n **do**
4: $C \leftarrow$ select the non-homogeneous subset $\in S$ with the largest diameter
5: **if** $C == \varnothing$ {All subsets are homogeneous} **then**
6: $C \leftarrow$ select the homogeneous subset $\in S$ with the largest diameter
7: **end if**
8: $(S_x, S_y) \leftarrow$ divide C into two subsets
9: add(S, S_x)
10: add(S, S_y)
11: remove(S, C)
12: **end for**
13: $CS \leftarrow \varnothing$
14: **for** each subset $T \in S$ **do**
15: $r \leftarrow$ compute the mean item by averaging the items in T
16: $r.label \leftarrow$ find the most common class label in T
17: $CS \leftarrow CS \cup \{r\}$
18: **end for**
19: **return** CS

Chen's algorithm (3/3)

It selects the next subset that will be divided by examining its diameter. The idea is that a subset with a large diameter probably includes more training items. Therefore, if this subset is divided first, a higher reduction rate will be achieved.

A desirable property is that It builds the same condensing subset regardless of the ordering of the data in the training set.

Weak points:

The first is that the algorithm is parametric. The user has to specify the number of prototypes

The second is that the items that do not belong to the most common class of the subset are not represented in the condensing set

Reduction by Space Partitioning

RSP algorithms are based on the Chen's algorithm

RSP1 computes for each group as many mean items as the different classes in the group. It selects the next group to be split the group with the largest diameter

RSP1 and RSP2 differ on the way they select the next group to be divided. **RSP2** uses as a splitting criterion the overlapping degree of each group

RSP3: All non-homogeneous groups are split. The size of the Condensing Set is determined automatically

RSP3 (1/2)

Algorithm 7 RSP3

Input: TS
Output: CS

1: $S \leftarrow \varnothing$
2: add(S, TS)
3: $CS \leftarrow \varnothing$
4: **repeat**
5: $C \leftarrow$ select the subset $\in S$ with the highest splitting criterion value
6: **if** C is homogeneous **then**
7: $r \leftarrow$ calculate the mean item by averaging the items in C
8: $r.label \leftarrow$ class of items in C
9: $CS \leftarrow CS \cup \{r\}$
10: **else**
11: $(D_1, D_2) \leftarrow$ divide C into two subsets
12: add(S, D_1)
13: add(S, D_2)
14: remove(S, C)
15: **end if**
16: **until** IsEmpty(S)
17: **return** CS

RSP3 (2/2)

RSP3 is the only RSP algorithm (CJA included) that automatically determines the size of the condensing set

The condensing set built by RSP3 does not depend on the data order in the training set

RSP3 generates few prototypes for representing non close-class-border areas, and many prototypes for representing close-class-border areas.

Retrieving of the most distant items in each subset implies the computations of all distances between the items of the subset

Editing and/or data Abstraction through Homogeneous Clusters

Stefanos Ougiaroglou
stoug@uom.gr



PhD student
Information Management lab
Dept. of Applied Informatics, UoM

RHC: Reduction through Homogeneous Clusters (1/9)

RHC properties:

- RHC is an abstraction DRT
- Fast execution of the reduction procedure (low pre-processing cost)
- High reduction rates
- High classification accuracy
- Non-parametric algorithm
- Algorithm based on the well-known k-Means clustering
- RHC condensing set does not depend on the order of training data

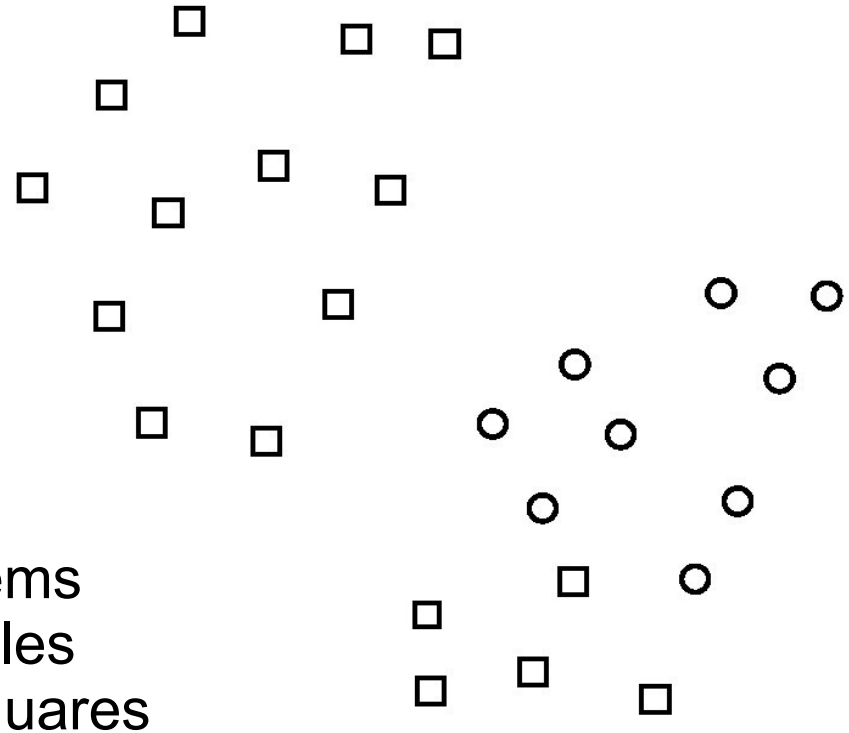
RHC: Reduction through Homogeneous Clusters (2/9)

RHC idea:

- RHC continues constructing clusters until all of them are homogeneous
- A cluster is homogeneous if all items that have been assigned to it are of a specific class
- The centroids of the homogeneous clusters constitute the condensing set

RHC: Reduction through Homogeneous Clusters (3/9)

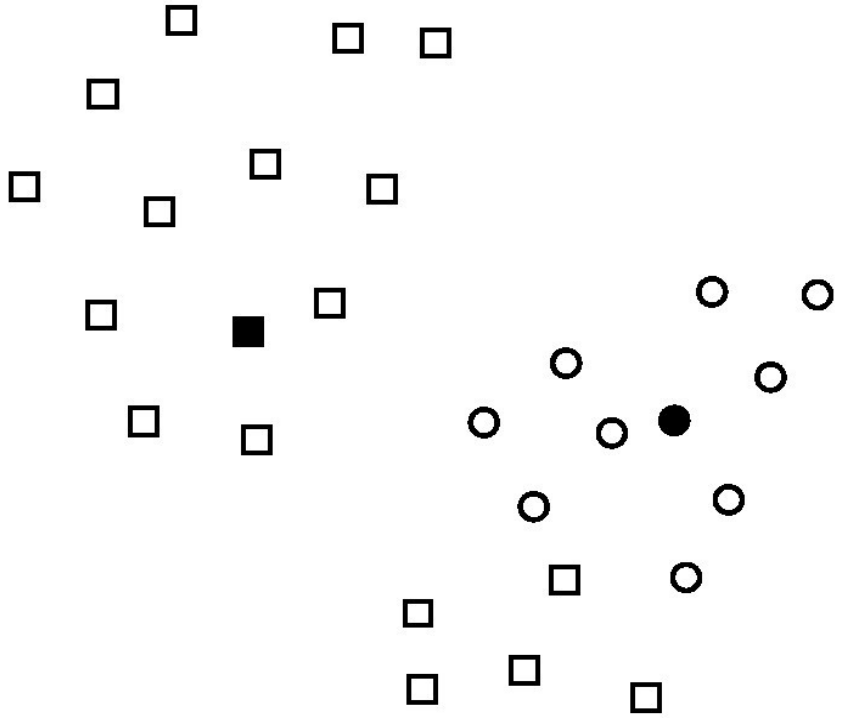
Initially, RHC considers the dataset as a non-homogeneous cluster



26 items
9 circles
17 squares

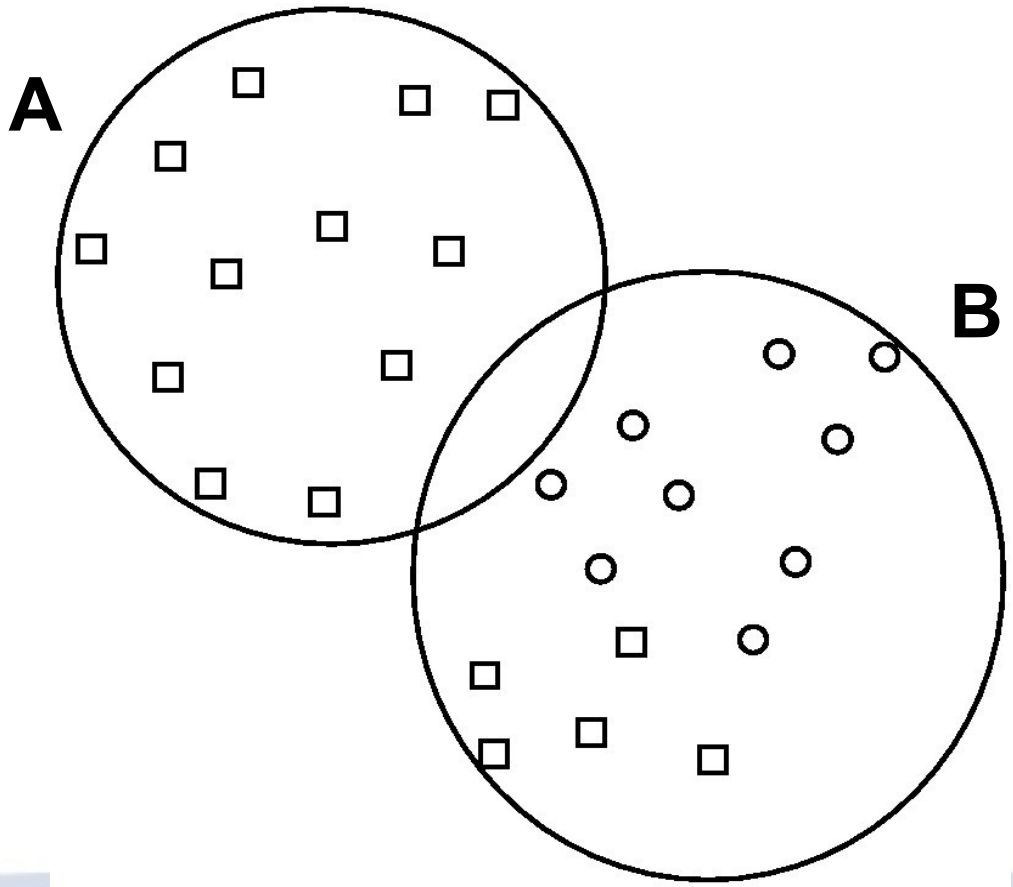
RHC: Reduction through Homogeneous Clusters (4/9)

RHC computes the mean item for each class in the data (class centroid)



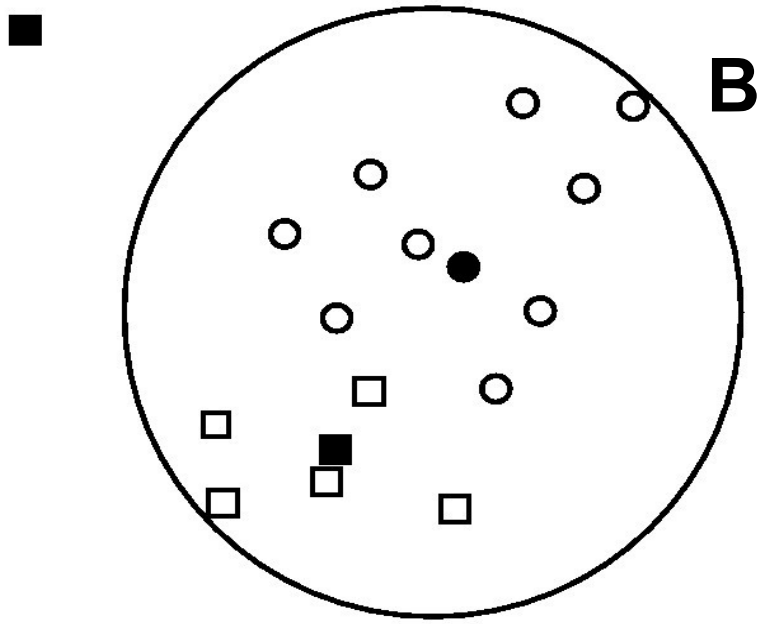
RHC: Reduction through Homogeneous Clusters (5/9)

RHC executes k-means clustering using the two class centroids as initial means and builds two clusters



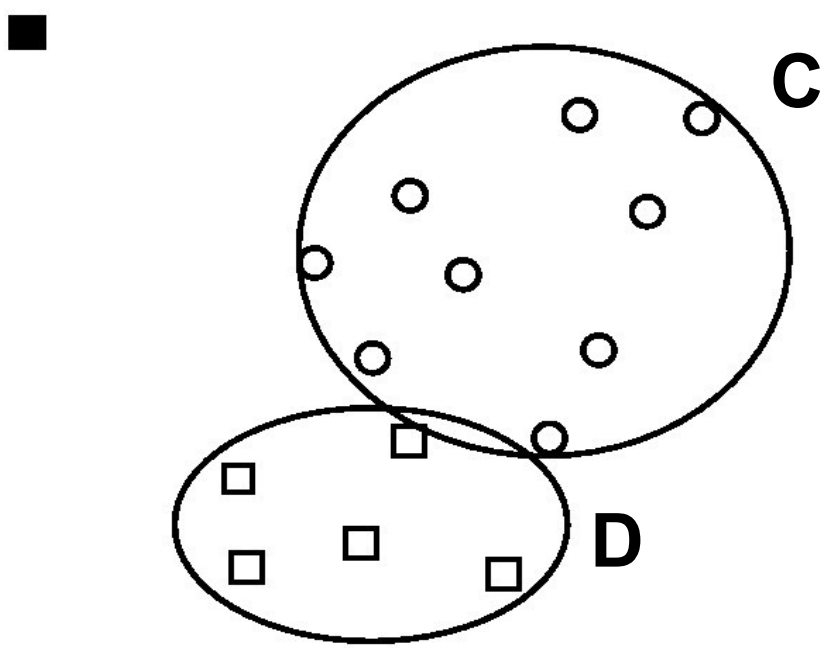
RHC: Reduction through Homogeneous Clusters (6/9)

RHC stores the centroid of cluster A to the condensing set and computes a class centroids for each class in cluster B



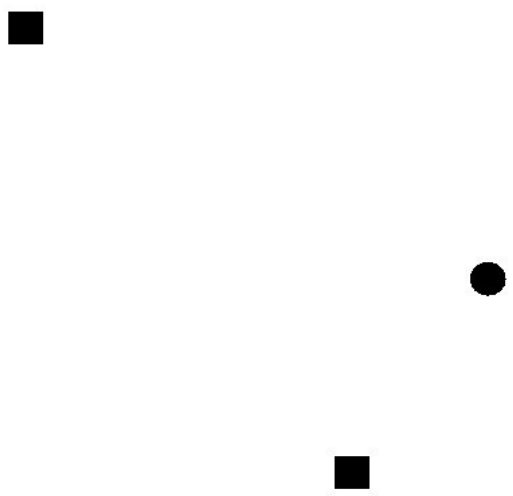
RHC: Reduction through Homogeneous Clusters (7/9)

k-Means is executed on the data of cluster B using as initial means the centroids of the two classes and produces two clusters



RHC: Reduction through Homogeneous Clusters (8/9)

Clusters C and D are homogeneous. RHC stores their centroids to the condensing set



RHC Condensing set

RHC Algorithm (9/9)

Algorithm 1 RHC Algorithm

Input: TS **Output:** CS

```
1:  Enqueue(Queue, TS)
2:  repeat
3:    C ← Dequeue(Queue)
4:    if C is homogeneous then
5:      Put the mean vector (centroid) of C into CS
6:    else
7:      ClassCentroids ← ∅
8:      for each Class L in C do
9:        Centroid_L ← Compute the mean vector of items
          that belong to L
10:       ClassCentroids ← ClassCentroids ∪ Centroid_L
11:     end for
12:     NewClusters ← K-MEANS(C, ClassCentroids)
13:     for each cluster X in NewClusters do
14:       Enqueue(Queue, X)
15:     end for
16:   end if
17: until IsEmpty(Queue)
18: return CS
```

Editing through Homogeneous Clusters

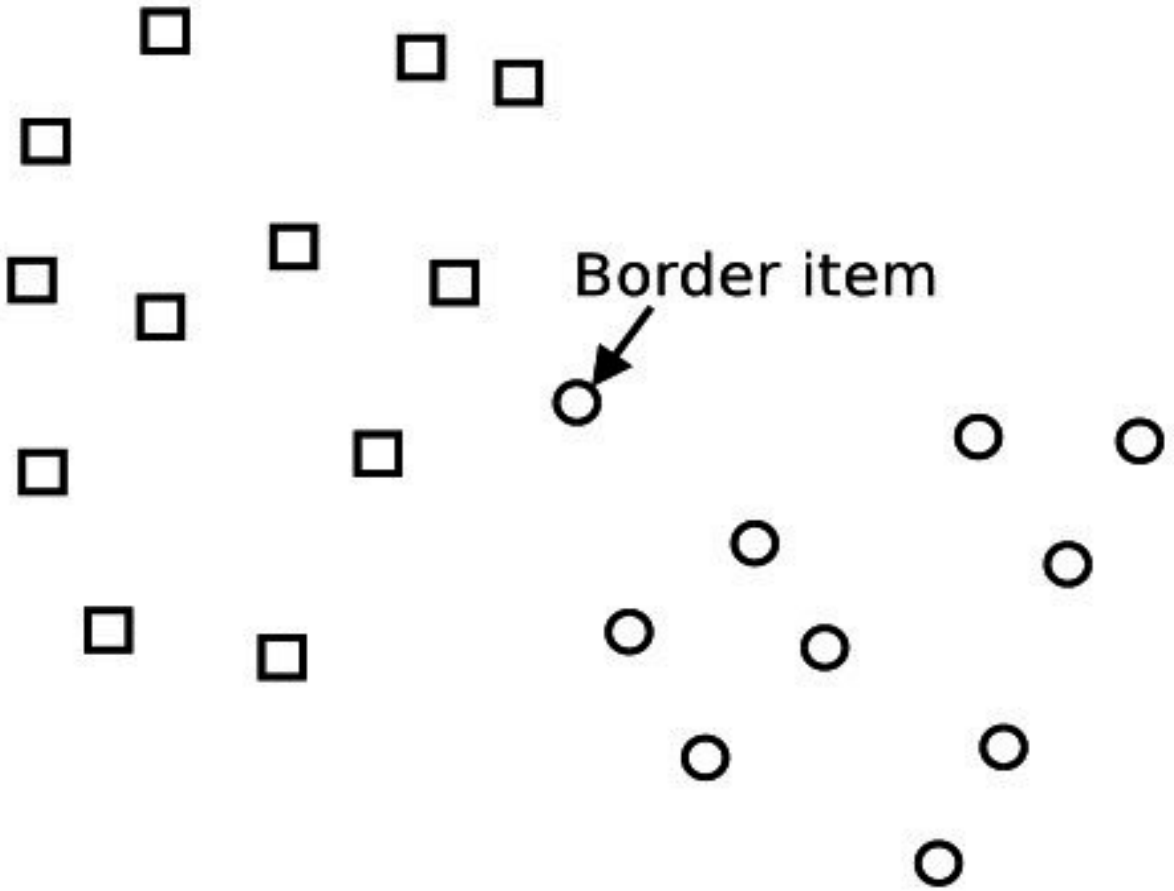
EHC properties

- It follows completely different strategy from that of ENN-based approaches
- Fast execution
- Non-parametric
- It is based on *k*-means clustering

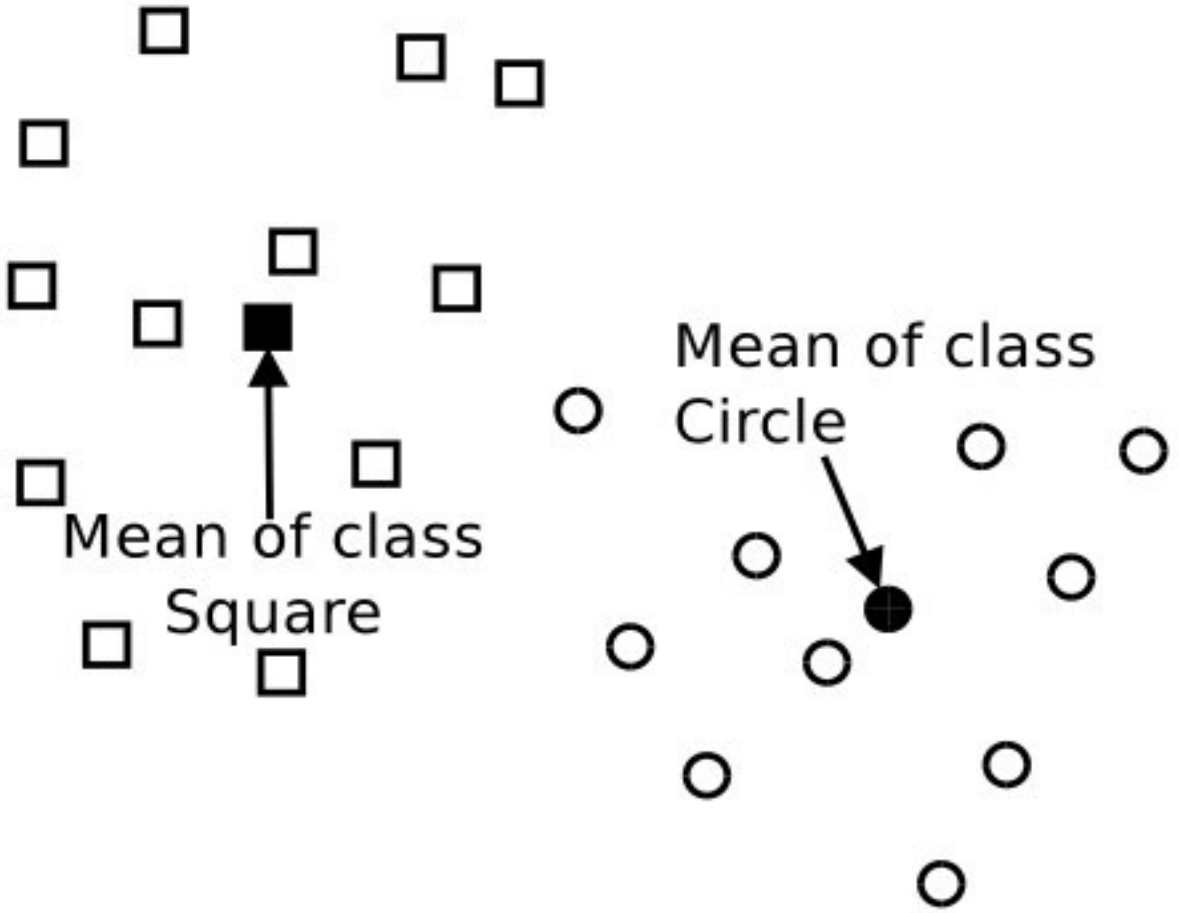
EHC idea:

- It continues constructing clusters until all of them are homogeneous (they include items of a specific class only)
- It removes the clusters that contain only one item (they are considered as outliers, noisy or close-border items)

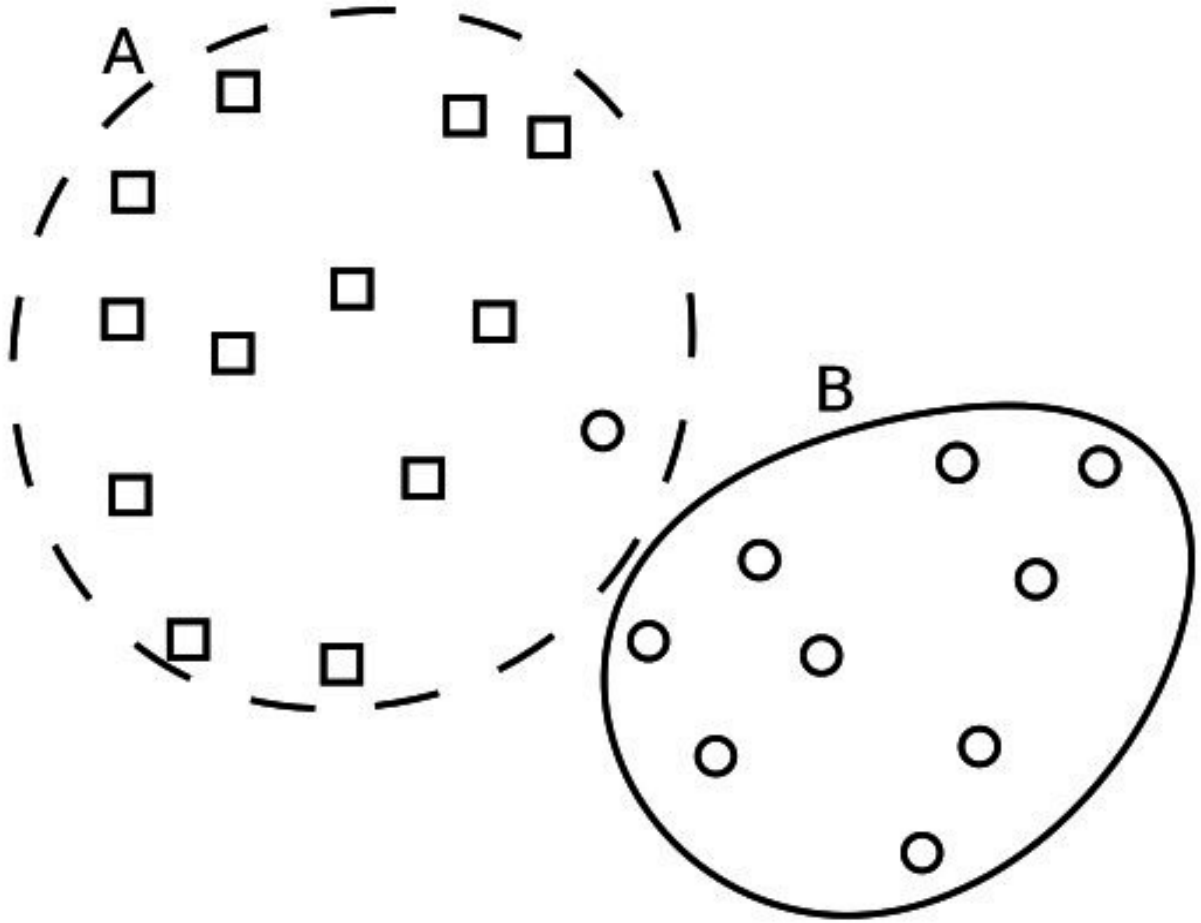
EHC: Remove of a border item (1/6)



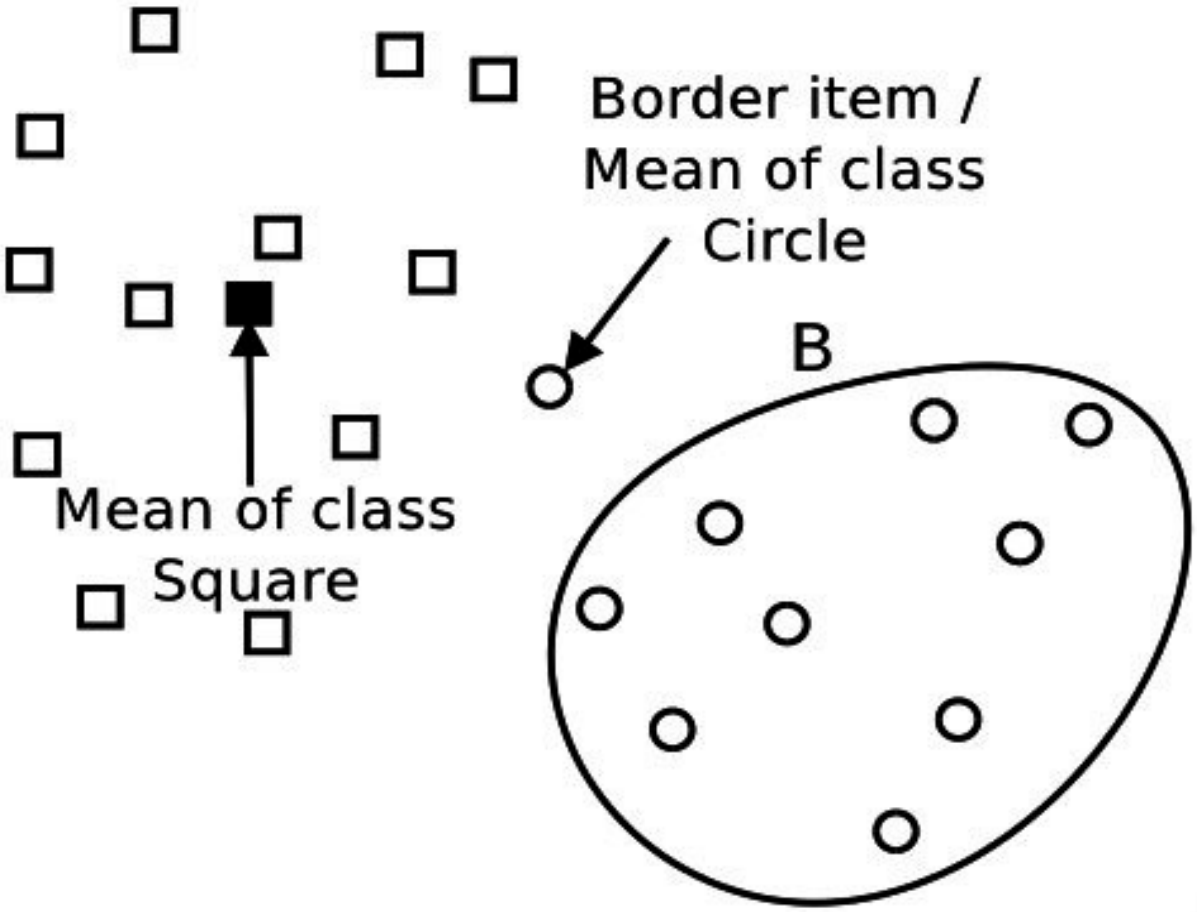
EHC: Remove of a border item (2/6)



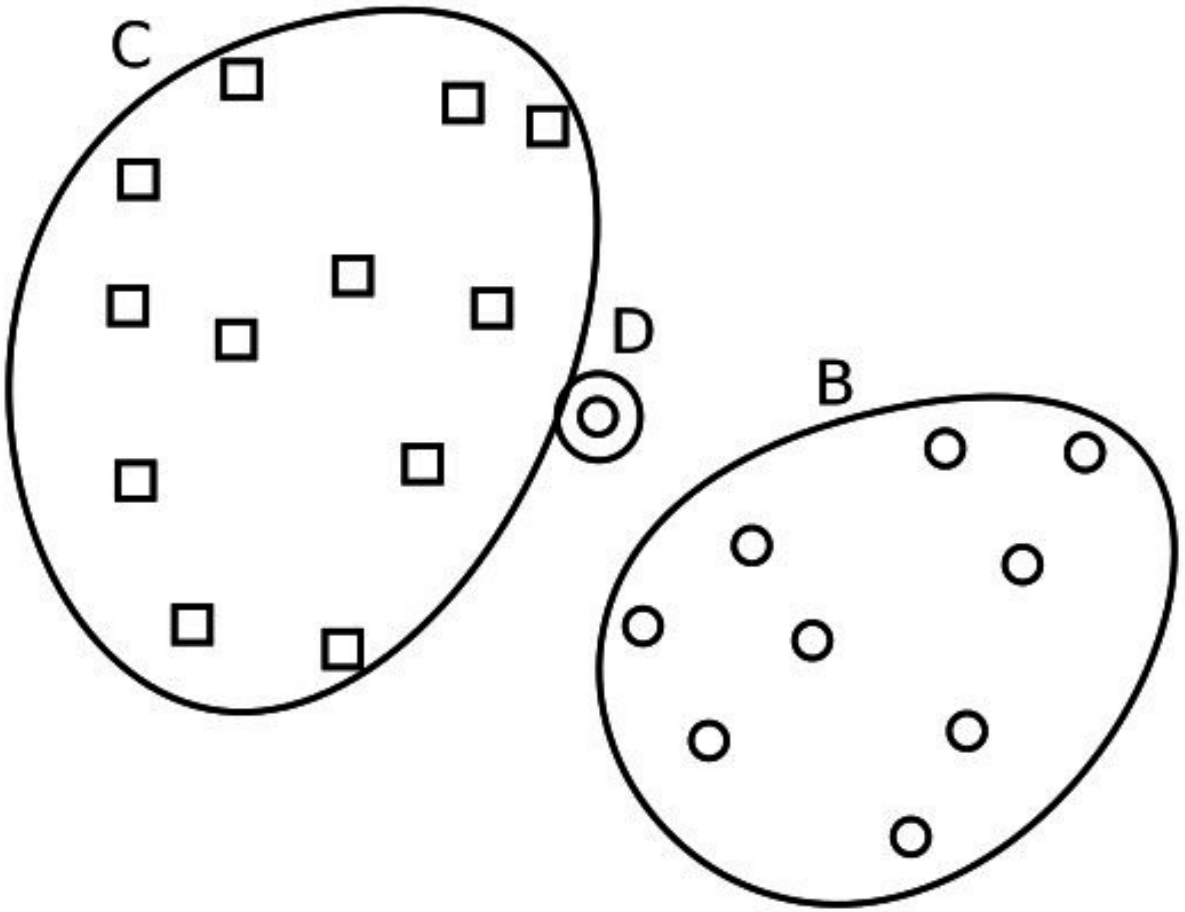
EHC: Remove of a border item (3/6)



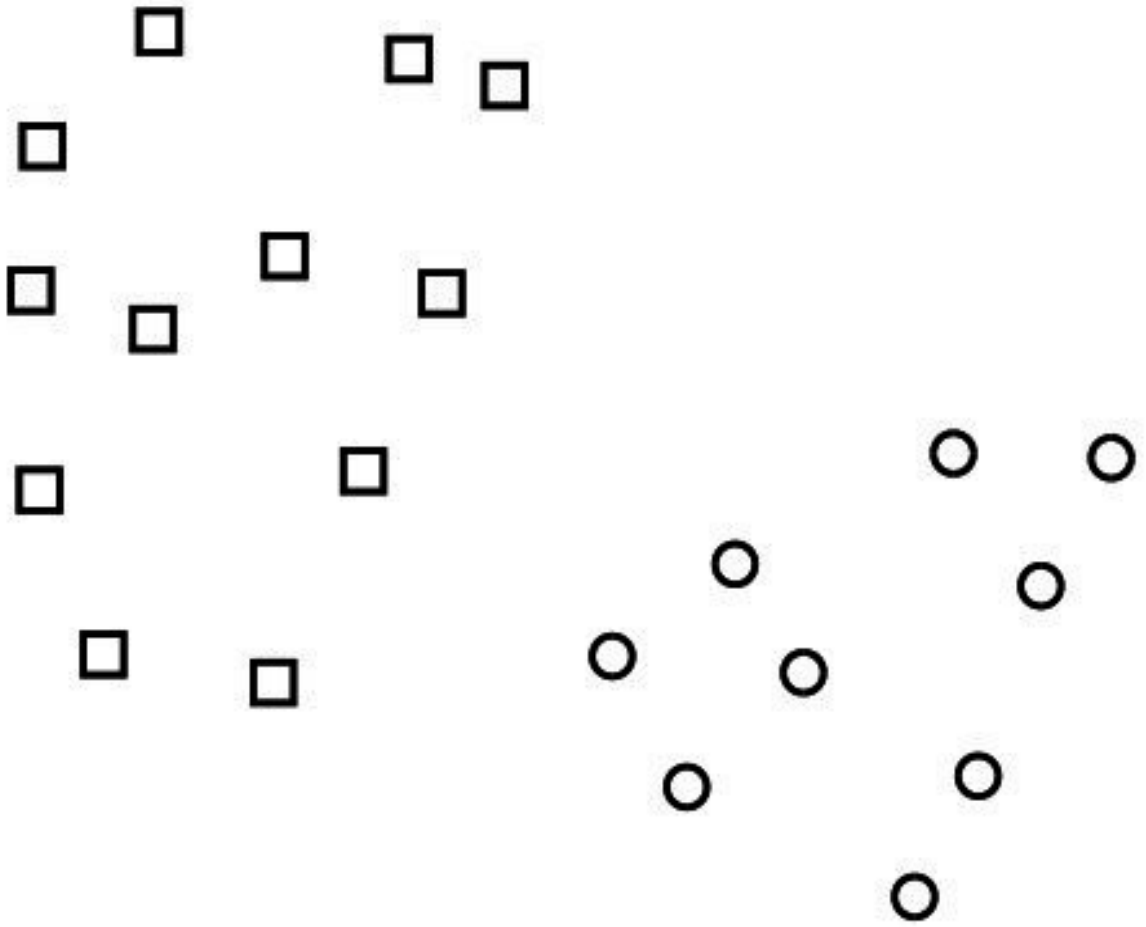
EHC: Remove of a border item (4/6)



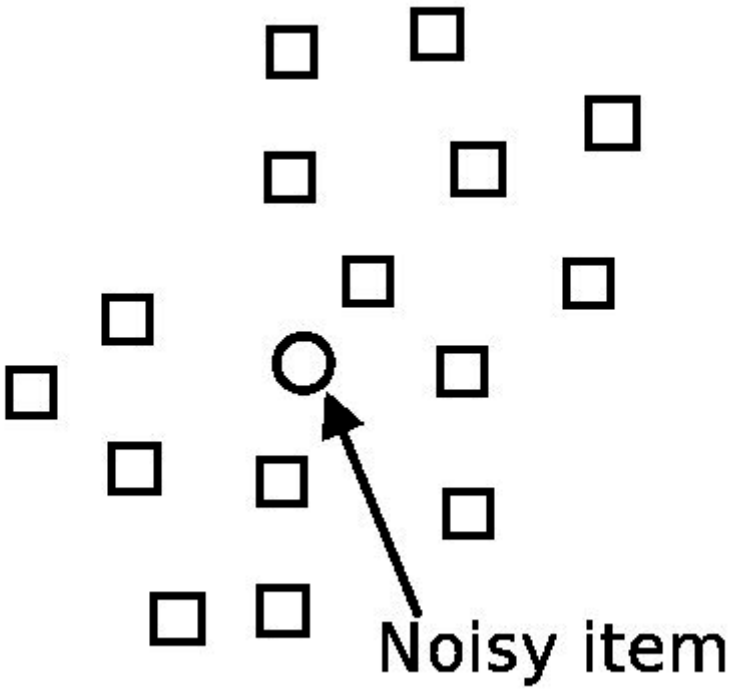
EHC: Remove of a border item (5/6)



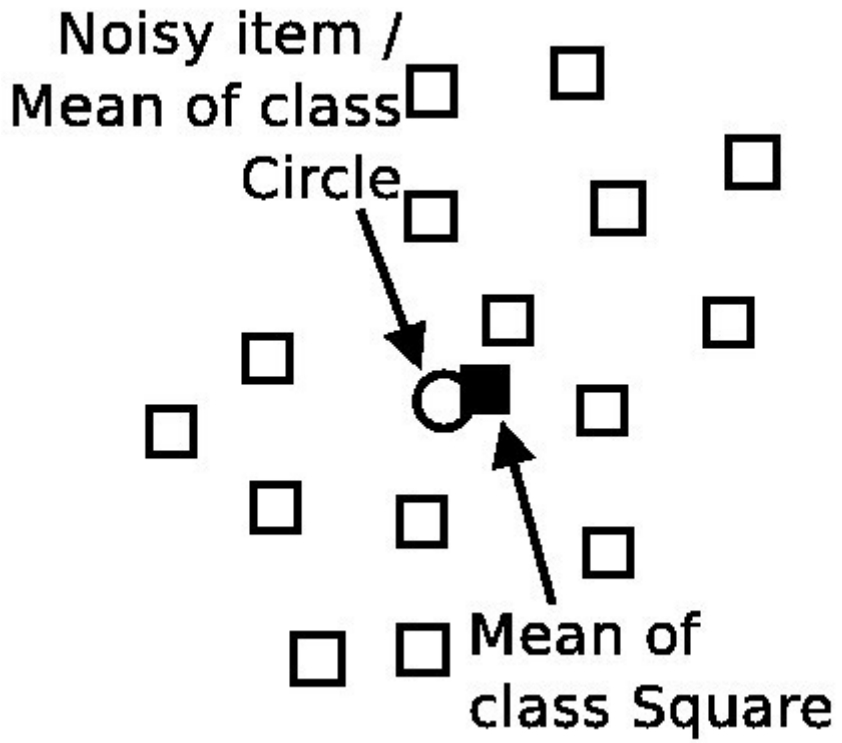
EHC: Remove of a border item (6/6)



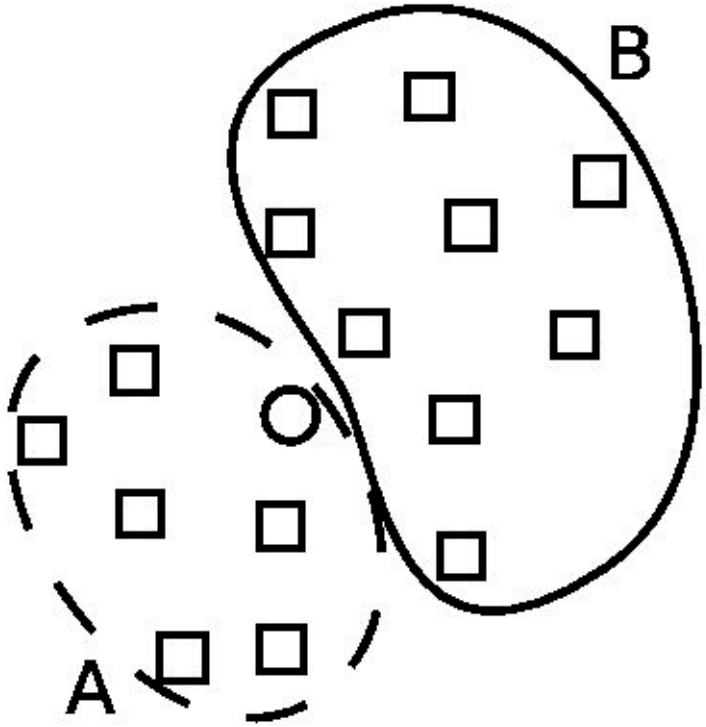
EHC: Remove of a noisy item (1/6)



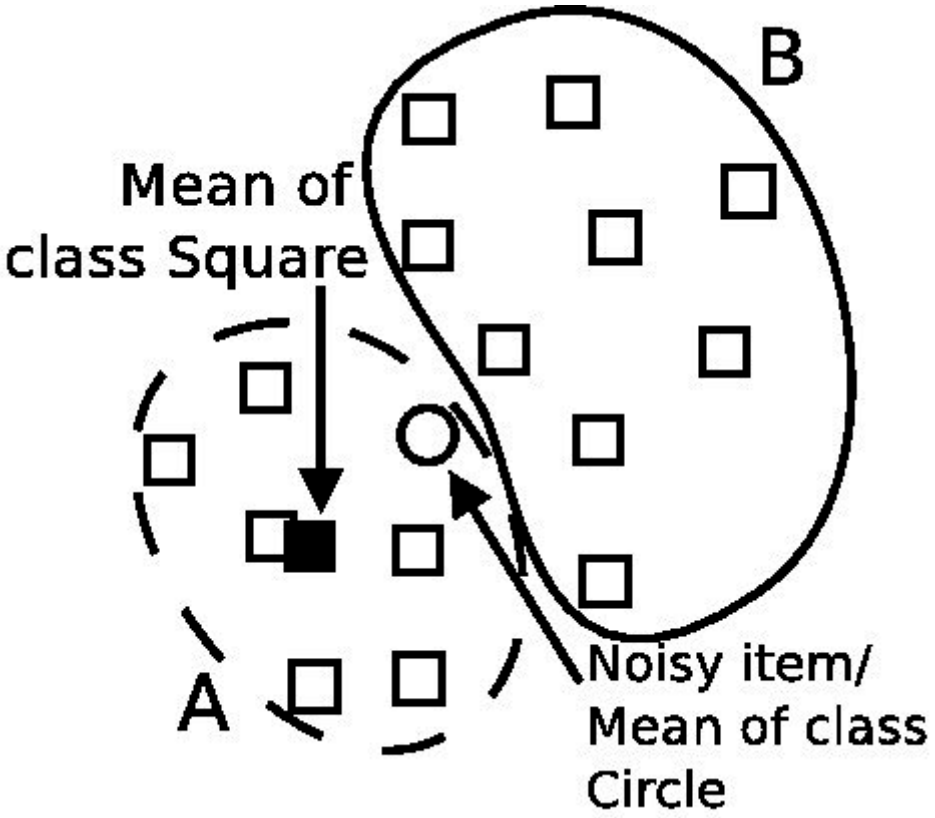
EHC: Remove of a noisy item (2/6)



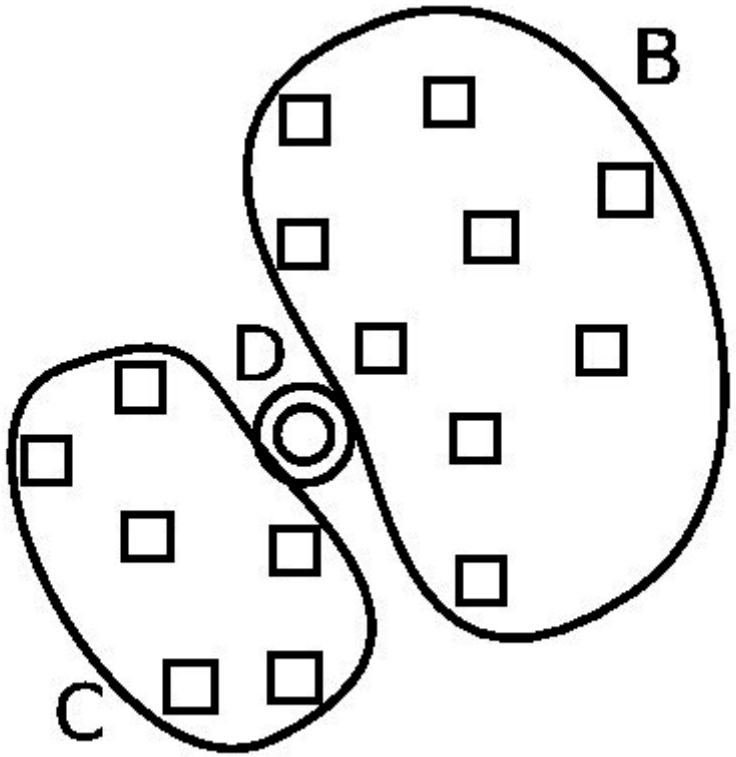
EHC: Remove of a noisy item (3/6)



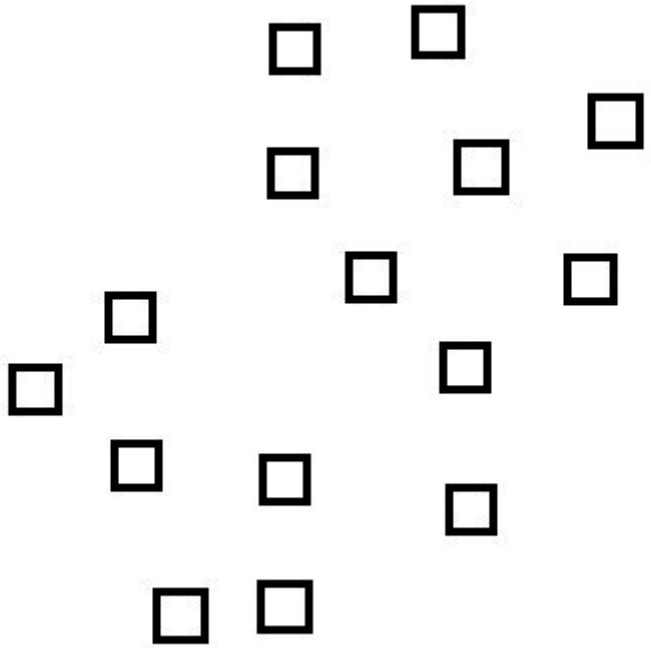
EHC: Remove of a noisy item (4/6)



EHC: Remove of a noisy item (5/6)



EHC: Remove of a noisy item (6/6)



Observations

The number of clusters constructed in each step is automatically determined. It is equal to the number of classes that exist in the cluster

The higher the level of noise in the data, the more one-item-clusters built and removed

EHC does not compute distances between “real” items. It computes distances between items and mean items

The number of distances computed by EHC is difficult to predict in advance. It exclusively depends on the item distribution in the data space

EHC builds the same ES regardless of data ordering

EHC Implementation

Algorithm EHC

Input: TS
Output: ES

1: $ES \leftarrow TS$
2: $Q \leftarrow \varnothing$
3: Enqueue(Q, TS)
4: **repeat**
5: $C \leftarrow$ Dequeue(Q)
6: **if** C is homogeneous **then**
7: **if** $|C| = 1$ **then**
8: $ES \leftarrow ES - C$
9: **end if**
10: **else**
11: $R \leftarrow \varnothing$ {R is the set of class means}
12: **for** each class M in C **do**
13: $R \leftarrow R \cup mean_of(M)$
14: **end for**
15: $Clusters \leftarrow K$-MEANS(C, R)
16: **for** each cluster $Cl \in Clusters$ **do**
17: Enqueue(Q, Cl)
18: **end for**
19: **end if**
20: **until** IsEmpty(Q) {until all constructed clusters are homogeneous}
21: **return** ES

ERHC (1/2)

ERHC combines EHC and RHC:

- It is a RHC variation that can effectively manage datasets with noise
- It works similarly to RHC but it discards clusters with one items (They are considered to be noise)

ERHC Implementation

Algorithm 12 ERHC

Input: TS
Output: CS

```
1: Queue ← ∅
2: Enqueue(Queue, TS)
3: CS ← ∅
4: repeat
5:    C ← Dequeue(Queue)
6:    if C is homogeneous then
7:       if |C| > 1 then
8:          r ← mean of C
9:          CS ← CS ∪ {r}
10:      end if
11:   else
12:      M ← ∅ {M is the set of class means}
13:      for each class L in C do
14:         m_L ← mean of L
15:         M ← M ∪ {m_L}
16:      end for
17:      NewClusters ← K-MEANS(C, M)
18:      for each cluster C ∈ NewClusters do
19:         Enqueue(Queue, C)
20:      end for
21:   end if
22: until IsEmpty(Queue)
23: return CS
```