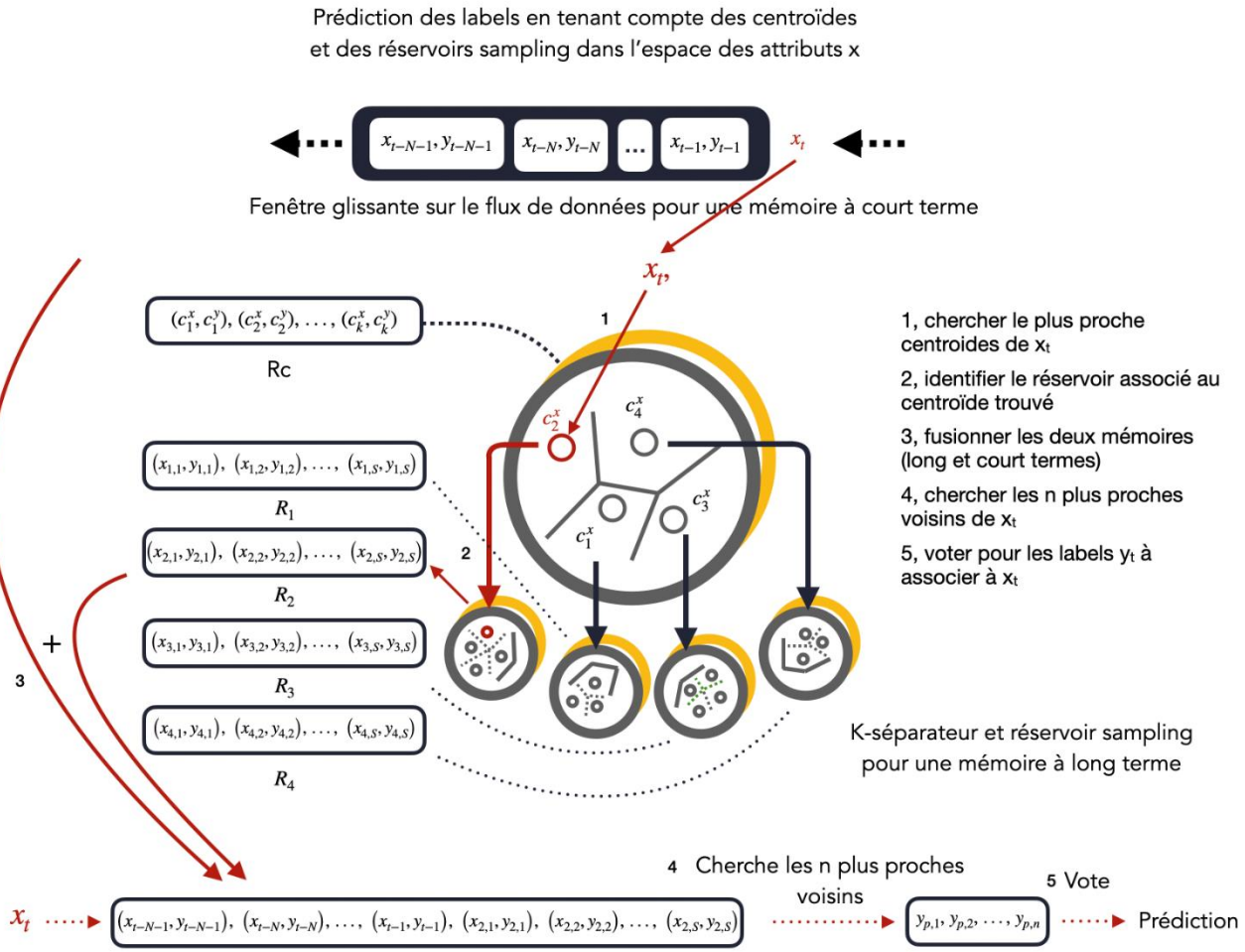
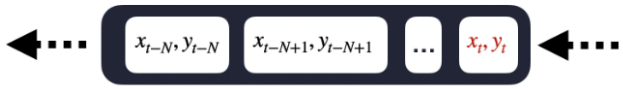
Algorithme OMk

Notation	Description
t	Le temps d'instant
k	Le nombre de réservoirs
n	Les nombres de plus proches voisins pour la prédiction
S	La taille du réservoir d'échantillonnage
N	La taille de la fenêtre temporelle glissante FIFO
$\{R_i\}_{i\in[1,k]}$	L'ensemble des k réservoirs
$R_i = \{(x_{ij}, y_{ij})\}_{j\in[1,S]}$	L'i-ème réservoir
T_i	Le nombre de données traitées par l'i-ème réservoir
$c^{(x)}_i$	Les valeurs moyennes des attributs des mémoires stockées dans R_i
$c^{(y)}_i$	Les valeurs moyennes des labels des mémoires stockées dans R_i
$\{(c^{(x)}_i, c^{(y)}_i)\}_{i\in[1,k]}$	Les séparateurs
$F_t = \{(x_{t-i}, y_{t-i})\}_{i\in[1,N]}$	La fenêtre temporelle glissante
Initial_ Stream	Les données pour initialiser le module
Initial_size	La taille de l'Initial_ *Stream # 1%-5% de la taille de Data Stream*

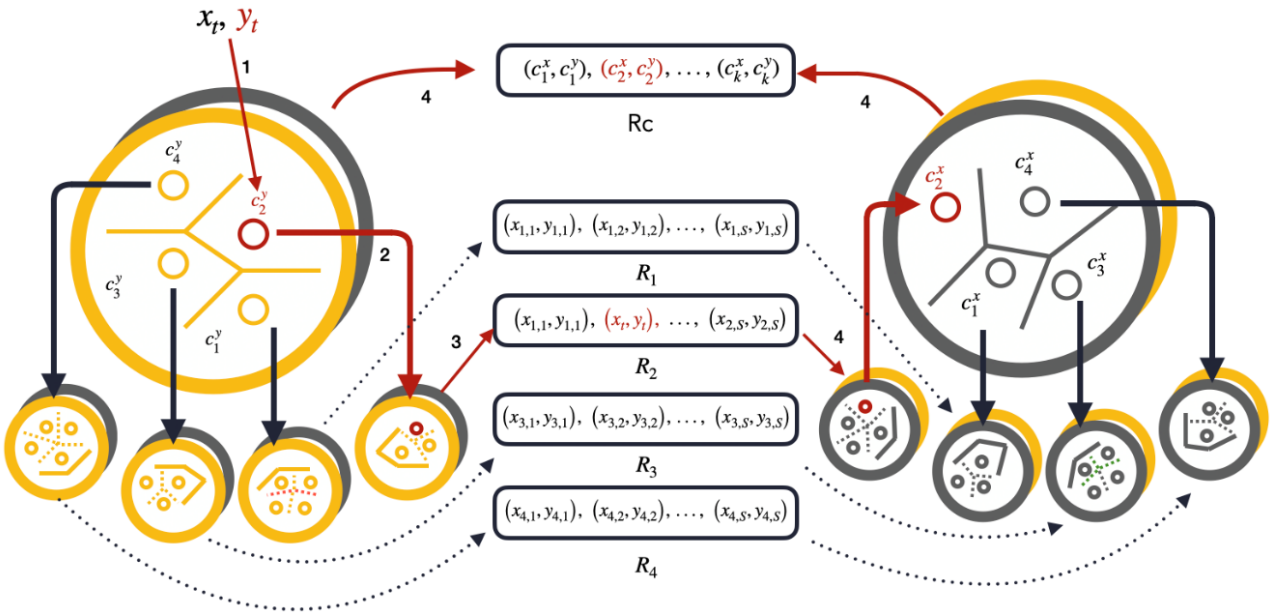
Vue schématique du processus d'apprentissage



Entraînement du modèle par la mise à jour des centroïdes
et des réservoirs sampling dans l'espace des labels y



Fenêtre glissante sur le flux de données pour une mémoire à court terme



K-séparateur et réservoir sampling pour une mémoire à long terme

1, chercher le plus proche centroide de y_t

2, identifier le réservoir associé au centroïde trouvé

3, ajouter la nouvelle donnée au réservoir identifié

4, mettre à jour les centroïdes sur X et Y

Algo 1: Initialisation

Input : *Initial_Stream*
Result: *Initial_Module*
k_subset_Index = k_Clustering(Initial_Stream.labels);
k_séparateur = { };
F_t *={};*
# initialisation du k-séparateur et des reservoirs sampling
For *i from 1 to k* **do**
  For *index∈k_subset_Index[i]* **do**
    # with reservoir sampling algo
    Initial_Stream[index] est stockée dans R_i ;
  EndFor
  $c^{(x)}_i$ *= (1/* R_i*.size) * *R_i*.attributs.sum();*
  $c^{(y)}_i$ *= (1/* R_i*.size) * *R_i*.labels.sum();*
  k_séparateur.add(($c^{(x)}_i$, $c^{(y)}_i$));
EndFor
# initialisation de la fenêtre glissante (mémoire courte)
For *i from (Initial_Stream.size – N) to Initial_Stream.size* **do**
  F_t*.add*(x_i,y_i));
EndFor

Algo 2: Prédiction

Input : x_t
Result: $Prédiction_t$
distances1 = Euclidean(x_t, { $c^{(x)}_i$} $_{i\in[1,k]}$);
P = l'index de la plus petite valeur de distances1
R_p *= Identifier le réservoir plus proche*
distances2 = Euclidean(x_t, R_i.attributs);
distances3 = Euclidean(x_t, F_t.attributs);
{p_1,..., p_n} = les index du n plus petites valeurs de *distances2 et de distances3*
votes = {} ;
For *index* $\in$ *{ p_1,..., p_n }* **do**
labels = (R_p U F_t).labels[index];
votes.add(labels) ;
EndFor
$Prédiction_t$ *<- votes.*majoritaire() *;*
EndFor

Algo 3: Apprentissage

Input : *(x_t,y_t)*
Result: *Module d'apprentissage*
distances1 = Euclidean(x_t, { $c^{(y)}_i$} $_{i\in[1,k]}$);
P = l'index de la plus petite valeur de distances1
R_p *= Identifiez le réservoir plus proche*
(x_t,y_t) est stocké dans R_p; # with reservoir sampling algo
$c^{(x)}_p$ *= (1/R_p.Size) * R_p.Attributs.sum();*
$c^{(y)}_p$ *= (1/ R_p.Size) * R_p. Labels.sum();*
k_séparateur.update(($c^{(x)}_p$, $c^{(y)}_p$));
F_t*.removeOldest();*
F_t*.add((x_t,y_t)) ;*

OMk en multi-label en flux

Input : *Data_Stream*
Result: OMk_*Module*
Initial_Stream = {} ;
While *Data_Stream.hasMoreInstance()* **do**
(x_t,y_t) <- Data_Stream.nextInstance();
# Cas particulier en Initialisation
If *Initial_Stream.size < Initial_size* **Then**
Prédiction (faire un vote majoritaire sur les n plus proches voisins de la donnée x_t en compte tenu des données dans *Initial_Stream*)**;**
Apprentissage (ajouter la donnée *(x_t,y_t)* dans *Initial_Stream*) = *Initiale_Stream.add((x_t,y_t)) ;*
# Cas général d'usage d'OMk
Else
Initialisation (Algo 1); (à exécuter une seule fois) ;
Prediction (Algo 2);
Apprentissage (Algo 3);
End
EndWhile