Matériel supplémentaire de l'article "Apprentissage multi-labels et multi-tâches en continu pour données tabulaires: proposition d'un protocole de création de tâches et évaluation des principaux classifieurs"

1 Algorithmes de partitionnement envisagés

Plusieurs méthodes de partitionnement ont été comparés : kmeans (Ahmed et al., 2020) avec initialisation kmeans++, bisecting kmeans (Kumar et al., 2000) avec initialisation kmeans++, spectral coclustering (Ng et al., 2001), kmeans (Chao et al., 2017) et spherical kmeans (Hornik et al., 2012). Finalement, l'algorithme spherical kmeans, a été retenu. De complexité temporelle linéaire $\mathcal{O}(MNd)$, avec M le nombre d'itérations, N le nombre d'instances et d le nombre d'attributs considérés pour le partitionnement, le spherical kmeans est moins complexe que les autres méthodes envisagées. Nous avons aussi constaté dans des expérimentations préliminaires qu'il permet également d'obtenir une bonne diversité des signatures de labels des partitions sur une majorité des jeux de données.

2 Algorithmes et valeurs des paramètres testées pour les algorithmes utilisés

— Neural network (NN) (A1) : un réseau de neurone implémenté grâce à Pytorch. La fonction d'activation en sortie est une fonction sigmoid. La fonction de perte utilisée est la binary cross entropy, souvent utilisée pour les problèmes de classification multi-labels (Maxwell et al., 2017), et l'optimisation utilisée est adam. Complexité temporelle pour une instance en entrée : $\mathcal{O}(d * l)$ avec d le nombre d'attributs en entrée et l le nombre de labels. Paramètres : learning rate [0.1, 0.01 , 0.001].

— Neural network and targeted loss (NN_TL) (A2) : il s'agit du modèle NN mais un avec un calcul de la fonction de perte effectué uniquement sur les labels appartenant à la tâche évaluée, les pertes sur les autres labels sont donc fixées à 0. Complexité temporelle pour une instance en entrée : $\mathcal{O}(d * l)$ avec d le nombre d'attributs en entrée et l le nombre de labels. Paramètres : learning rate [0.1, 0.01 , 0.001].

— Neural network, targeted loss and hidden layer (NN_TLH) (A3) : il s'agit de l'algorithme NN_TL avec l'ajout d'une couche cachée avec une fonction d'activation ReLU. Complexité temporelle pour une instance en entrée : $\mathcal{O}(d * h + h * l)$ avec d le nombre d'attributs en entrée, h la taille de la couche cachée et l le nombre de labels. Paramètres : learning rate [0.1, 0.01 , 0.001], taille de la couche cachée [200, 2000].

— Neural network, targeted loss, hidden layer and fifo memory (NN_TLH_fifo) (A4) : il s'agit de NN_TLH avec l'ajout d'une mémoire FIFO. Chaque instance du flux est mémorisée dans un réservoir qui supprime l'instance mémorisée la plus ancienne pour libérer une place. Lors de l'apprentissage, un nombre n d'instances mémorisées sont sélectionnées aléatoirement dans la mémoire pour entraîner le réseau de neurones. Complexité temporelle pour une instance en entrée : $\mathcal{O}(r_f * (d * h + h * l))$ avec d le nombre d'attributs en entrée, h la taille de la couche cachée, l le nombre de labels et r_f le nombre d'instances rejouées depuis la mémoire FIFO. Paramètres : learning rate [0.1, 0.01 , 0.001], taille de la couche cachée [200,2000], taille de la mémoire fifo [100, 1000], nombre d'instance de la mémoire fifo rejouées en ligne par phase d'apprentissage [5,10].

— Neural network, targeted loss, hidden layer and reservoir sampling (NN_TLH_sampling) (A5) : il s'agit de NN_TLH avec l'ajout d'un reservoir sampling. Chaque instance du flux est mémorisée dans un réservoir qui supprime aléatoirement une des instances mémorisées pour libérer une place. Lors de l'apprentissage, un nombre n d'instances mémorisées sont sélectionnées aléatoirement dans le réservoir pour entraîner le réseau de neurones. Complexité temporelle pour une instance en entrée : $\mathcal{O}(r_s * (d * h + h * l))$ avec d le nombre d'attributs en entrée, h la taille de la couche cachée, l le nombre de labels et r_s le nombre d'instances rejouées depuis le réservoir. Paramètres : learning rate [0.1, 0.01 , 0.001], taille de la couche cachée [200,2000], taille du réservoir [100, 1000], nombre d'instance du réservoir rejouées en ligne par phase d'apprentissage [5,10].

— Neural network, targeted loss, hidden layer and fifo memory + reservoir sampling (NN_TLH_memories) (A6) : il s'agit de NN_TLH avec l'ajout d'une mémoire FIFO et d'un reservoir sampling. Chaque instance du flux est mémorisée dans le reservoir et la mémoire FIFO, et lors de l'apprentissage, plusieurs instances sont aléatoirement sélectionnées dans la mémoire et le réservoir pour entraîner le réseau de neurones. Complexité temporelle pour une instance en entrée : $\mathcal{O}((r_s + r_f) * (d * h + h * l))$ avec d le nombre d'attributs en entrée, h la taille de la couche cachée, l le nombre de labels, r_s le nombre d'instances rejouées depuis le réservoir et r_h le nombre d'instances rejouées depuis la mémoire FIFO. Paramètres : learning rate [0.1, 0.01 , 0.001], taille de la couche cachée [200,2000], taille de la mémoire fifo [100, 1000], nombre d'instance de la mémoire fifo rejouées en ligne par phase d'apprentissage [5,10], taille du réservoir [100, 1000], nombre d'instance du réservoir rejouées en ligne par phase d'apprentissage [5,10].

— Neural network, targeted loss, hidden layer and fifo memory + reservoir sampling (NN_TLH_mini_memories) (A7) : il s'agit d'une variante de (A6). Lors de l'apprentissage, plusieurs instances sont aléatoirement sélectionnées dans la mémoire et le réservoir pour former un mini-batch destiné à entraîner le réseau de neurones en une seule passe. Complexité temporelle pour une instance en entrée : $\mathcal{O}((r_s + r_f) * (d * h + h * l))$ avec d le nombre d'attributs en entrée, h la taille de la couche cachée, l le nombre de

labels, r_s le nombre d'instances rejouées depuis le réservoir et r_h le nombre d'instances rejouées depuis la mémoire FIFO. Paramètres : learning rate [0.1, 0.01 , 0.001], taille de la couche cachée [200,2000], taille de la mémoire fifo [100, 1000], nombre d'instance de la mémoire fifo rejouées par phase d'apprentissage [5,10], taille du réservoir [100, 1000], nombre d'instance du réservoir rejouées par phase d'apprentissage [5,10].

— BR-HT (A8) : le classifieur hoeffding tree (Hulten et al. (2001)) proposé par River adapté au problème multi-labels avec la méthode binary relevance (Zhang et al. (2018)). Complexité temporelle pour une instance en entrée : $\mathcal{O}(l * d)$ avec d le nombre d'attributs en entrée et l le nombre de labels. Paramètres : grace period [100, 200], delta [1e-06,1e-07], tau [0.05, 0.1].

— LC-HT (A9) : le classificateur hoeffding tree proposé par River adapté au problème multi-labels avec la méthode pruned set (Read et al. (2008)) aussi parfois appelée label combination (déjà implémentée dans River). Complexité temporelle pour une instance en entrée : $\mathcal{O}(2^l * d)$ avec d le nombre d'attributs en entrée et l le nombre de labels. Paramètres : grace period [100, 200], delta [1e-06,1e-07], tau [0.05, 0.1].

— CC-HT (A10) : un hoeffding tree adapté au problème multi-labels avec la méthode des classifier chains (Read et al. (2009)) (déjà implémentée dans River). Complexité temporelle pour une instance en entrée : $\mathcal{O}(l * d)$ avec d le nombre d'attributs en entrée et l le nombre de labels. Paramètres : grace period [100, 200], delta [1e-06,1e-07], tau [0.05, 0.1].

— BR-ARF (A11) : l'algorithme adaptive random forest (Gomes et al. (2017)) proposé par River adapté au problème multi-labels avec la méthode binary relevance. Complexité temporelle pour une instance en entrée : $\mathcal{O}(k * l * d)$ avec d le nombre d'attributs en entrée, l le nombre de labels et k le nombre de classifieurs. Paramètres : nombre de modèles [5, 10, 15], grace period [100, 200], delta [1e-06,1e-07], tau [0.05, 0.1].

— iSOUPtree (A12) : Incremental Structured Output Prediction Tree (Osojnik et al. (2018)) est un arbre capable de faire de la régression multi-cibles. Cet algorithme est déjà implémenté dans River. Complexité temporelle pour une instance en entrée : $\mathcal{O}(l * d)$ avec d le nombre d'attributs en entrée et l le nombre de labels. Paramètres : grace period [100, 200], delta [1e-06,1e-07], tau [0.05, 0.1].

3 Métriques utilisées

Voici les métriques utilisées pour l'évaluation en ligne :

— La macro-averaged balanced accuracy (BA_{macro}) : après avoir utilisé un seuil sur le score en sortie du modèle pour chaque label prédit (0 si $\hat{y}_{t_i} < 0.5$ et 1 $\hat{y}_{t_i} > 0.5$). Une balanced accuracy est calculée et mémorisée pour chaque label considéré puis une moyenne est réalisée. Cette mesure est implémentée avec River.

— L'écart quadratique moyen ($RMSE$) est lui calculé avant le seuillage. Cette mesure permet de faire la différence entre deux algorithmes qui auraient des performances similaires après le seuillage mais n'ont en réalité pas le même niveau de confiance dans leur prédiction.

— La précision à k ($precision@k$) est calculée en comptant le nombre de prédictions correctes parmi les k prédictions où le modèle est le plus confiant ($\hat{y}_{t_i}$ le plus éloigné possible de 0.5 avant seuillage), puis en divisant par k. Dans le protocole utilisée $k = 3$

sauf dans le cas où une tâche possède moins de 3 labels dans sa signature dans le flux. Dans ce cas précis, k prend la valeur du nombre de labels dans la tâche qui en possède le moins dans sa signature.

Après le passage d'une expérience d'apprentissage, le modèle est figé puis évalué sur les ensembles d'évaluation de chacune des tâches du flux pour générer une matrice de macro-averaged balanced accuracy $\mathbf{R} \in [0,1]^{n_r \times n_t}$ avec n_r le nombre d'expériences d'apprentissage passées dans le flux, et n_t le nombre de tâches, comme dans le cas de figure évoqué par (Lopez-Paz et Ranzato, 2017) où le nombre de lignes de la matrice $\mathbf{R}$ est beaucoup plus élevé que le nombre de tâches. $R_{i,j}$ est la macro-averaged balanced accuracy de test de h_i sur t_j. Avec la matrice $\mathbf{R}$, nous pouvons ensuite calculer 3 métriques communes de la littérature en apprentissage continu en observant les accuracy $R_{i,j}$ lors de l'apparition d'une nouvelle tâche dans le flux (Lin et al., 2021) :

— Accuracy moyenne (AV_ACC) (Díaz-Rodríguez et al., 2018) : cette accuracy moyenne mesure la performance moyenne sur toutes les tâches connues du modèle au moment de l'évaluation continue, puis moyenne les valeurs obtenues lors de toutes les évaluations continues afin d'obtenir une performance globale.

— Transfert en avant ou forward transfer (FWT) : nous considérons le FWT comme étant la contribution d'une expérience d'apprentissage sur la performance du modèle sur une tâche qu'il ne connaît pas encore. Dans le cadre de ce papier, il s'agit donc de la différence entre la performance obtenue sur une tâche inconnue avant et après le passage de l'expérience d'apprentissage. Nous moyennons ensuite les valeurs obtenues sur toutes les tâches inconnues lors d'une évaluation continue, puis sur toutes les évaluations continues où le FWT a pu être mesuré. Sa valeur est comprise dans l'intervalle $[-1, 1]$.

— Transfert en arrière ou backward transfer (BWT) : nous considérons le BWT comme étant la contribution d'une expérience d'apprentissage sur la performance du modèle sur une tâche déjà connue. Dans le cadre de ce papier, il s'agit donc de la différence entre la performance obtenue sur une tâche connue avant et après le passage de l'expérience d'apprentissage. Nous moyennons ensuite les valeurs obtenues sur toutes les tâches connues lors d'une évaluation continue, puis sur toutes les évaluations continuesoù le BWT a pu être mesuré. Sa valeur est comprise dans l'intervalle $[-1, 1]$.

Concernant le score de frugalité, nous utilisons la formule proposée par (Evchenko et al., 2021) :

$$Frug = ACC - \frac{w}{1 + \frac{1}{C}}$$

avec ACC la BA_{macro} obtenue à la fin du passage de la première expérience d'apprentissage dans la recherche par cadrillage des hyperparamètres, ou l'accuracy moyenne des évaluations continues obtenue sur tout le flux de données, C la consommation totale estimée durant le passage du flux, et $w = 1$ le poids donné à la frugalité dans notre évaluation. Cette métrique présente l'avantage de donner un score dans l'intervalle $[-1, 1]$ et d'être modifiable en fonction de l'importance que nous accordons à la frugalité dans notre analyse. Nous avons choisi la valeur du poids w en suivant les recommandations de (Evchenko et al., 2021), pour obtenir un score qui favorise des algorithmes rapides avec une accuracy raisonnable.

4 Jeux de données synthétiques

Trois jeux de données synthétiques ont été générés. Chacun de ces jeux de données possèdent 4 attributs et 4 labels. Les attributs sont tirés aléatoirement à partir d'une loi uniforme entre 0 et 1. Nous avons ensuite créé 3 matrices M différentes de dimensions 4×4. Les lignes de ces matrices correspondent chacune à une tâche dans le jeu de données. $M_{i,j}$ représente le coefficient multiplicateur appliqué à l'attribut j dans la tâche i. Une fois le coefficient multiplicateur appliqué à l'attribut, le label correspondant à cet attribut vaut 1 si la valeur de l'attribut est supérieure à 0.5, et 0 dans le cas contraire.

— synth_monolab : ce jeu de données utilise une matrice diagonale.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

— synth_bilab : ce jeu de données utilise une matrice avec deux coefficients multiplicateurs par tâche.

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

— synth_rand : ce jeu de données utilise une matrice générée aléatoirement et de rang 4, à laquelle on a ajouté une constante à 0.5 afin d'éviter un trop grand nombre de labels absents après seuillage à 0.5.

$$M = \begin{bmatrix} 1.1 & 1.3 & 1.4 & 1.3 \\ 0.9 & 1 & 1.5 & 1.1 \\ 1 & 1.3 & 0.6 & 1.2 \\ 0.7 & 0.6 & 0.6 & 1.4 \end{bmatrix}$$

4.1 Résultats complémentaires

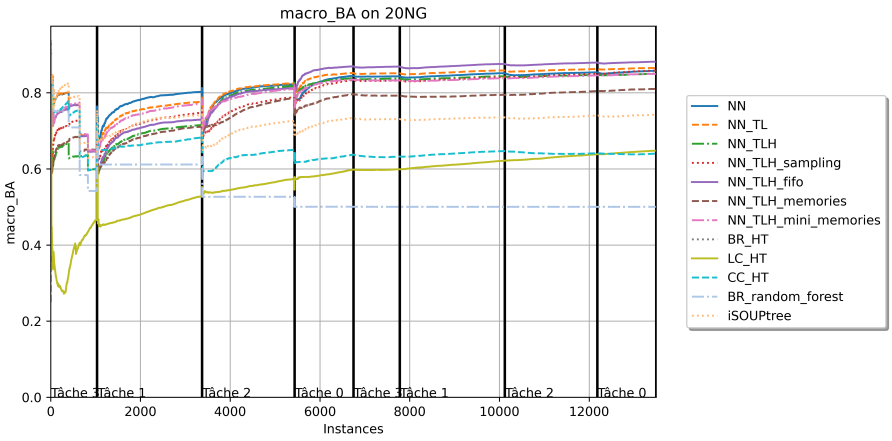

FIG. 1 – *BA_{macro} de 12 stratégies sur le jeu de données 20NG*

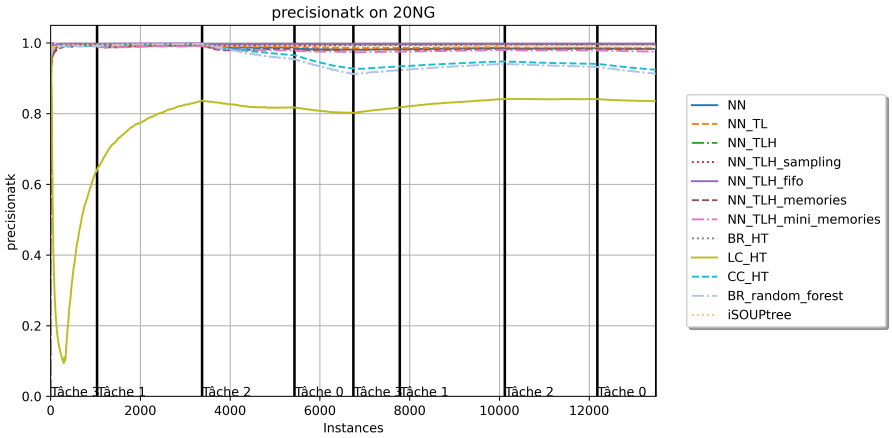

FIG. 2 – *$precision@k$ de 12 stratégies sur le jeu de données 20NG*

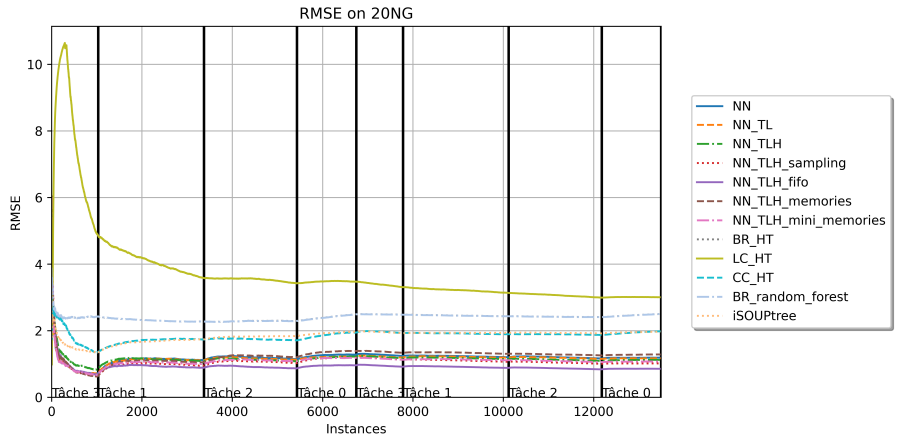

FIG. 3 – *RMSE de 12 stratégies sur le jeu de données 20NG*

	20NG	Mediamill	Scene	Yeast	Synth_monolab	Synth_bilab	Synth_rand	Avg. value	Avg. rank
NN	0.660	0.566	0.607	0.530	0.679	0.771	0.873	0.670	6.710
NN_TL	0.671	0.567	0.597	0.529	0.955	0.956	0.874	0.735	5.571
NN_TLH	0.670	0.564	0.594	0.532	0.957	0.910	0.886	0.730	6.000
NN_TLH_sampling	**0.682**	0.564	0.594	0.561	**0.971**	0.971	**0.907**	0.750	3.286
NN_TLH_fifo	0.681	0.564	0.594	**0.573**	0.931	**0.974**	0.898	0.745	4.429
NN_TLH_memories	0.673	0.564	0.691	0.571	0.955	0.969	0.905	0.761	3.429
NN_TLH_mini_memories	0.672	**0.569**	**0.706**	0.570	0.965	0.972	0.904	**0.766**	**2.286**
BR_HT	0.617	0.558	0.495	0.500	0.590	0.573	0.661	0.571	11.000
LC_HT	0.586	0.436	0.658	0.557	0.651	0.657	0.767	0.616	8.857
CC_HT	0.623	0.566	0.615	0.538	0.667	0.813	0.903	0.675	6.000
BR_random_forest	0.625	0.558	0.500	0.498	0.573	0.559	0.659	0.568	11.286
iSOUPtree	0.630	0.565	0.573	0.515	0.655	0.755	0.849	0.649	8.571

TAB. 1 – *Accuracy moyenne de 12 stratégies sur 7 jeux de données.*

	20NG	Mediamill	Scene	Yeast	Synth_monolab	Synth_bilab	Synth_rand	Avg. value	Avg. rank
NN	0.001	0.000	−0.075	−0.009	−0.046	0.044	0.002	−0.012	4.860
NN_TL	−0.046	−0.004	−0.003	−0.005	−0.139	−0.000	0.003	−0.028	7.571
NN_TLH	−0.041	0.001	0.000	−0.019	−0.159	−0.013	**0.007**	−0.032	6.429
NN_TLH_sampling	−0.055	0.002	0.000	−0.020	−0.167	0.003	−0.018	−0.037	8.429
NN_TLH_fifo	−0.055	0.001	0.000	−0.023	−0.129	0.008	−0.003	−0.029	6.714
NN_TLH_memories	−0.041	0.004	−0.086	−0.025	−0.162	0.012	−0.008	−0.044	7.571
NN_TLH_mini_memories	−0.050	−0.000	−0.090	−0.025	−0.137	0.008	−0.010	−0.044	9.286
BR_HT	**0.008**	**0.005**	−0.005	0.000	**0.000**	0.025	−0.004	**0.004**	**3.286**
LC_HT	−0.029	0.000	−0.103	−0.030	−0.005	0.044	−0.018	−0.020	7.714
CC_HT	−0.010	−0.002	−0.080	−0.003	−0.023	0.041	−0.010	−0.013	6.429
BR_random_forest	0.000	0.003	**0.001**	0.001	**0.000**	0.033	−0.013	0.004	3.429
iSOUPtree	−0.008	−0.002	−0.061	**0.004**	−0.040	**0.046**	−0.025	−0.012	5.714

TAB. 2 – *Transfert en avant (FWT) moyen de 12 stratégies sur 7 jeux de données.*

	20NG	Mediamill	Scene	Yeast	Synth_monolab	Synth_bilab	Synth_rand	Avg. value	Avg. rank
NN	**1.223**	**6.002**	**0.045**	**0.068**	0.048	0.053	0.061	**1.071**	**2.710**
NN_TL	1.285	6.191	0.046	0.069	0.049	0.054	0.064	1.108	3.714
NN_TLH	1.942	6.928	0.096	0.138	0.112	0.108	0.115	1.348	5.714
NN_TLH_sampling	40.854	12.649	1.150	0.612	0.374	0.371	0.689	8.100	9.714
NN_TLH_fifo	23.136	40.929	0.541	0.554	0.266	0.672	0.662	9.537	10.429
NN_TLH_memories	14.892	29.656	0.674	0.577	0.925	1.255	0.978	6.994	11.000
NN_TLH_mini_memories	4.317	8.092	0.062	0.109	0.066	0.072	0.100	1.831	5.000
BR_HT	6.314	9.791	0.089	0.137	0.024	0.028	**0.037**	2.346	3.429
LC_HT	14.937	60.176	0.163	0.675	**0.021**	**0.027**	0.040	10.863	6.571
CC_HT	11.488	25.640	0.164	0.198	0.030	0.034	0.042	5.371	6.143
BR_random_forest	5.973	27.519	0.230	0.239	0.138	0.196	0.200	4.928	8.429
iSOUPtree	7.575	12.114	0.179	0.166	0.025	0.028	**0.037**	2.875	4.857

TAB. 3 – *Energie consommée par la CPU (Wh) de 12 stratégies sur 7 jeux de données.*

	20NG	Mediamill	Scene	Yeast	Synth_monolab	Synth_bilab	Synth_rand	Avg. value	Avg. rank
NN	**6**	117	0.895	1.355	0.893	1.017	1.170	18	3.57
NN_TL	**6**	121	0.915	1.377	0.933	1.038	1.242	19	4.429
NN_TLH	10	139	1.745	2.564	1.977	1.898	2.019	23	5.857
NN_TLH_sampling	244	246	23.539	11.834	10.641	7.107	12.996	80	10.286
NN_TLH_fifo	143	798	11.021	10.711	5.168	12.712	12.432	142	9.857
NN_TLH_memories	89	575	13.567	11.068	17.614	23.845	18.606	107	10.143
NN_TLH_mini_memories	26	**47**	**0.354**	**0.565**	0.381	**0.420**	**0.510**	**11**	**1.571**
BR_HT	97	185	1.586	2.528	0.387	0.469	0.668	41	4.000
LC_HT	314	1230	3.256	12.817	**0.315**	0.437	0.720	223	7.000
CC_HT	221	524	3.290	7.902	0.515	0.603	0.764	108	6.857
BR_random_forest	121	580	4.568	4.586	2.533	3.767	3.823	103	8.571
iSOUPtree	164	254	3.516	3.148	0.402	0.473	0.663	61	5.714

TAB. 4 – *Energie consommée par la RAM (mWh) de 12 stratégies sur 7 jeux de données.*

	20NG	Mediamill	Scene	Yeast	Synth_monolab	Synth_bilab	Synth_rand	Avg. value	Avg. rank
NN	**103.626**	**508.436**	**3.779**	**5.769**	4.028	4.526	5.154	**90.760**	**2.710**
NN_TL	108.861	524.408	3.875	5.851	4.191	4.603	5.440	93.890	3.714
NN_TLH	164.499	586.906	8.109	11.690	9.473	9.185	9.705	114.224	5.714
NN_TLH_sampling	3460.604	1071.545	97.439	51.809	31.679	31.441	58.341	686.123	10.571
NN_TLH_fifo	1959.808	3466.991	45.796	46.934	22.551	56.947	56.092	807.874	10.286
NN_TLH_memories	1261.444	2512.122	57.073	48.873	78.361	106.282	82.821	592.425	10.857
NN_TLH_mini_memories	365.704	685.521	5.243	9.221	5.585	6.112	8.481	155.124	5
BR_HT	534.876	829.395	7.581	11.581	2.033	2.375	3.174	198.716	3.571
LC_HT	1265.399	5097.527	13.786	57.166	**1.760**	**2.247**	3.368	920.179	6.429
CC_HT	973.204	2172.107	13.929	16.733	2.549	2.897	3.541	454.994	6
BR_random_forest	506.052	2331.365	19.499	20.225	11.672	16.631	16.941	417.484	8.286
iSOUPtree	641.696	1026.282	15.161	14.049	2.088	2.396	**3.153**	243.546	4.857

TAB. 5 – *Durée de traitement complet du flux (s) de 12 stratégies sur 7 jeux de données.*

	20NG	Mediamill	Scene	Yeast	Synth_monolab	Synth_bilab	Synth_rand	Avg. value	Avg. rank
NN	**1.279**	**6.389**	**0.048**	**0.072**	0.051	0.057	0.064	**1.137**	**2.71**
NN_TL	1.348	6.590	0.049	0.073	0.053	0.058	0.068	1.177	3.714
NN_TLH	2.045	7.375	0.104	0.147	0.121	0.117	0.123	1.433	5.714
NN_TLH_sampling	42.969	13.467	1.227	0.651	0.395	0.393	0.732	8.548	10.571
NN_TLH_fifo	24.340	43.604	0.575	0.590	0.281	0.716	0.705	10.116	10.286
NN_TLH_memories	15.680	31.596	0.718	0.616	0.986	1.335	1.041	7.424	10.857
NN_TLH_mini_memories	4.537	8.502	0.065	0.116	0.069	0.075	0.107	1.925	5.000
BR_HT	6.678	10.391	0.097	0.145	0.027	0.031	**0.040**	2.487	3.429
LC_HT	15.877	63.957	0.175	0.717	**0.023**	**0.029**	0.043	11.546	6.571
CC_HT	12.192	27.205	0.174	0.208	0.033	0.037	0.045	5.699	6.571
BR_random_forest	6.343	29.266	0.245	0.254	0.146	0.207	0.212	5.239	7.571
iSOUPtree	8.062	12.877	0.189	0.175	0.027	0.031	**0.040**	3.058	4.571

TAB. 6 – *Energie consommée estimée (Wh) de 12 algorithmes sur 7 jeux de données.*

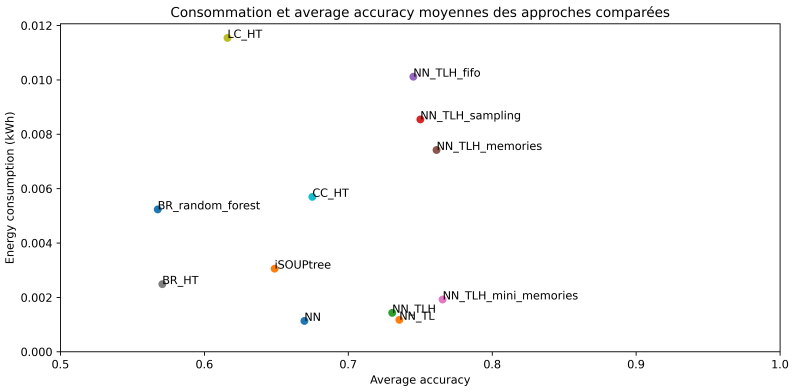

FIG. 4 – *Figure présentant consommation estimée totale pour traiter le flux complet (kWh) en ordonnée et l'accuracy moyenne en abscisse pour chacune des stratégies comparées.*

Références

Ahmed, M., R. Seraj, et S. M. S. Islam (2020). The k-means algorithm : A comprehensive survey and performance evaluation. *Electronics 9*(8), 1295.

Chao, G., S. Sun, et J. Bi (2017). A survey on multi-view clustering. *arXiv preprint arXiv :1712.06246*.

Díaz-Rodríguez, N., V. Lomonaco, D. Filliat, et D. Maltoni (2018). Don't forget, there is more than forgetting : new metrics for continual learning. *arXiv preprint arXiv :1810.13166*.

Evchenko, M., J. Vanschoren, H. H. Hoos, M. Schoenauer, et M. Sebag (2021). Frugal machine learning. *arXiv preprint arXiv :2111.03731*.

Gomes, H. M., A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, et T. Abdessalem (2017). Adaptive random forests for evolving data stream classification. *Machine Learning 106*, 1469–1495.

Hornik, K., I. Feinerer, M. Kober, et C. Buchta (2012). Spherical k-means clustering. *Journal of statistical software 50*, 1–22.

Hulten, G., L. Spencer, et P. Domingos (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 97–106.

Kumar, M. S. G. K. V., M. Steinbach, et G. Karypis (2000). A comparison of document clustering techniques. *Department of Computer Science and Engineering, University of Minnesota*.

Lin, Z., J. Shi, D. Pathak, et D. Ramanan (2021). The clear benchmark : Continual learning on real-world imagery. In *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 2)*.

Lopez-Paz, D. et M. Ranzato (2017). Gradient episodic memory for continual learning. *Advances in neural information processing systems 30*.

Maxwell, A., R. Li, B. Yang, H. Weng, A. Ou, H. Hong, Z. Zhou, P. Gong, et C. Zhang (2017). Deep learning architectures for multi-label classification of intelligent health risk prediction. *BMC bioinformatics 18*, 121–131.

Ng, A., M. Jordan, et Y. Weiss (2001). On spectral clustering : Analysis and an algorithm. *Advances in neural information processing systems 14*.

Osojnik, A., P. Panov, et S. Džeroski (2018). Tree-based methods for online multi-target regression. *Journal of Intelligent Information Systems 50*, 315–339.

Read, J., B. Pfahringer, et G. Holmes (2008). Multi-label classification using ensembles of pruned sets. In *2008 eighth IEEE international conference on data mining*, pp. 995–1000. IEEE.

Read, J., B. Pfahringer, G. Holmes, et E. Frank (2009). Classifier chains for multi-label classification. In *Machine Learning and Knowledge Discovery in Databases : European Conference, ECML PKDD 2009, Proceedings, Part II 20*, pp. 254–269. Springer.

Zhang, M.-L., Y.-K. Li, X.-Y. Liu, et X. Geng (2018). Binary relevance for multi-label learning : an overview. *Frontiers of Computer Science 12*, 191–202.