CentraleSupélec

Rapport de projet pour le cours :

Systèmes de décision

Présenté par :

Baddou Othmane

Bakoury Ayoub

Benaija Ismail

Le 02 février 2022

Table des matières

Introduction *3*

Le MR-sort *4*

Présentation de l'algorithme **4**

Particularités du MR-sort **5**

Nos résultats **6**

L'algorithme NCS (version SAT) *10*

NCS (version MaxSAT) *11*

NCS avec critères non monotones : **13**

Nos résultats **14**

Introduction

Dans le cadre du cours de systèmes de décision de l'école CentraleSupelec, il nous a été demandé de travailler sur la modélisation et l'implémentation de plusieurs algorithmes de décision.

Tout d'abord, rappelons le contexte décisionnel du projet. Considérons une situation dans laquelle un comité pour un programme d'enseignement supérieur doit décider de l'admission d'étudiants sur la base de leurs évaluations dans 4 cours. Pour être accepté dans le programme, le comité considère qu'un étudiant doit obtenir au moins une certaine note à une "majorité" de cours, sinon, l'étudiant est refusé. Du point de vue de la commission, tous les cours (critères) n'ont pas la même importance. Pour définir la majorité de cours requise, le comité attribue un poids $w_j \geq 0$ à chaque cours de telle sorte que leur somme soit égale à 1 ; un sous-ensemble de cours $C \subseteq \{M, P, L, H\}$ est considéré comme majoritaire si $\sum_{j \in C} w_j \geq \lambda$, où $\lambda \in [0, 1]$ est un niveau de majorité requis.

Dans le cadre de ce rapport, nous nous intéressons particulièrement au cas d'apprentissage des paramètres de ce modèle décisionnel à partir d'un jeu de données. Nous allons donc présenter plusieurs algorithmes qui permettent de répondre à cette problématique.

Plus précisément, nous allons présenter :

- Le Mr-sort algorithme
- La version SAT de l'algorithme NCS
- La version MaxSat de l'algorithme NCS

Le MR-sort

Présentation de l'algorithme

MR-Sort est un système de décision qui trie les éléments en classes en fonction de leur évaluation sur chaque critère en utilisant certains paramètres. L'objectif ici est d'apprendre ces paramètres à partir des décisions qui ont été prises (dans le jeu de données).

Dans le cas de deux catégories, le programme mathématique correspondant est le suivant (voir référence numéro 1 dans la table des matières) :

$$
\begin{cases}
max\ \alpha & \\
\quad \sum_{i \in N} c_{ij} + x_j + \varepsilon = \lambda & \forall a_j \in A^{*1} \\
\quad \sum_{i \in N} c_{ij} = \lambda + y_j & \forall a_j \in A^{*2} \\
\quad \alpha \leq x_j,\ \alpha \leq y_j & \forall a_j \in A^{*} \\
\quad c_{ij} \leq w_i & \forall a_j \in A^{*}, \forall i \in N \\
\quad c_{ij} \leq \delta_{ij} & \forall a_j \in A^{*}, \forall i \in N \\
\quad c_{ij} \geq \delta_{ij} - 1 + w_i & \forall a_j \in A^{*}, \forall i \in N \\
\quad M\delta_{ij} + \varepsilon \geq g_i(a_j) - b_i & \forall a_j \in A^{*},\ \forall i \in N \\
\quad M(\delta_{ij} - 1) \leq g_i(a_j) - b_i & \forall a_j \in A^{*},\ \forall i \in N \\
\quad \sum_{i \in N} w_i = 1,\ \lambda \in [0.5, 1] & \\
\quad w_i \in [0, 1] & \forall i \in N \\
\quad c_{ij} \in [0, 1],\ \delta_{ij} \in \{0, 1\} & \forall a_j \in A^{*},\ \forall i \in N \\
\quad x_j, y_j \in \mathbb{R} & \forall a_j \in A^{*} \\
\quad \alpha \in \mathbb{R} &
\end{cases}
$$

Par soucis de concision nous ne présenterons pas le cas à plus de deux catégories. Le lecteur pourra toujours se référer à l'annexe 1 pour plus d'informations à ce sujet à la section 3.1 du document.

On note tout de même que ces MILP peuvent s'avérer infaisables dans le cas où les affectations des alternatives dans l'ensemble d'apprentissage sont

incompatibles avec tous les modèles de MR sort. Le document de l'annexe 1 présente une solution pour remédier à ce problème à la section 3.3.

Particularités du MR-sort

Il est important de souligner que le modèle MR-Sort est très exigeant en termes de quantité de données. Un grand nombre d'exemples d'affectation est nécessaire pour obtenir de manière fiable de tels modèles de tri : d'environ 20 exemples pour le modèle à 7 paramètres avec 2 catégories et 3 critères à environ 75 exemples pour le modèle à 16 paramètres avec 3 catégories et 5 critères ; des ensembles d'apprentissage de cette taille garantissent moins de 10% d'erreurs d'affectation lors de l'utilisation du modèle appris.

Le besoin de beaucoup de données est également attesté par le fait que le modèle MR-Sort est capable « d'apprendre » à la fois d'ensembles d'apprentissage avec des erreurs mais aussi de ceux générés par un modèle différend.

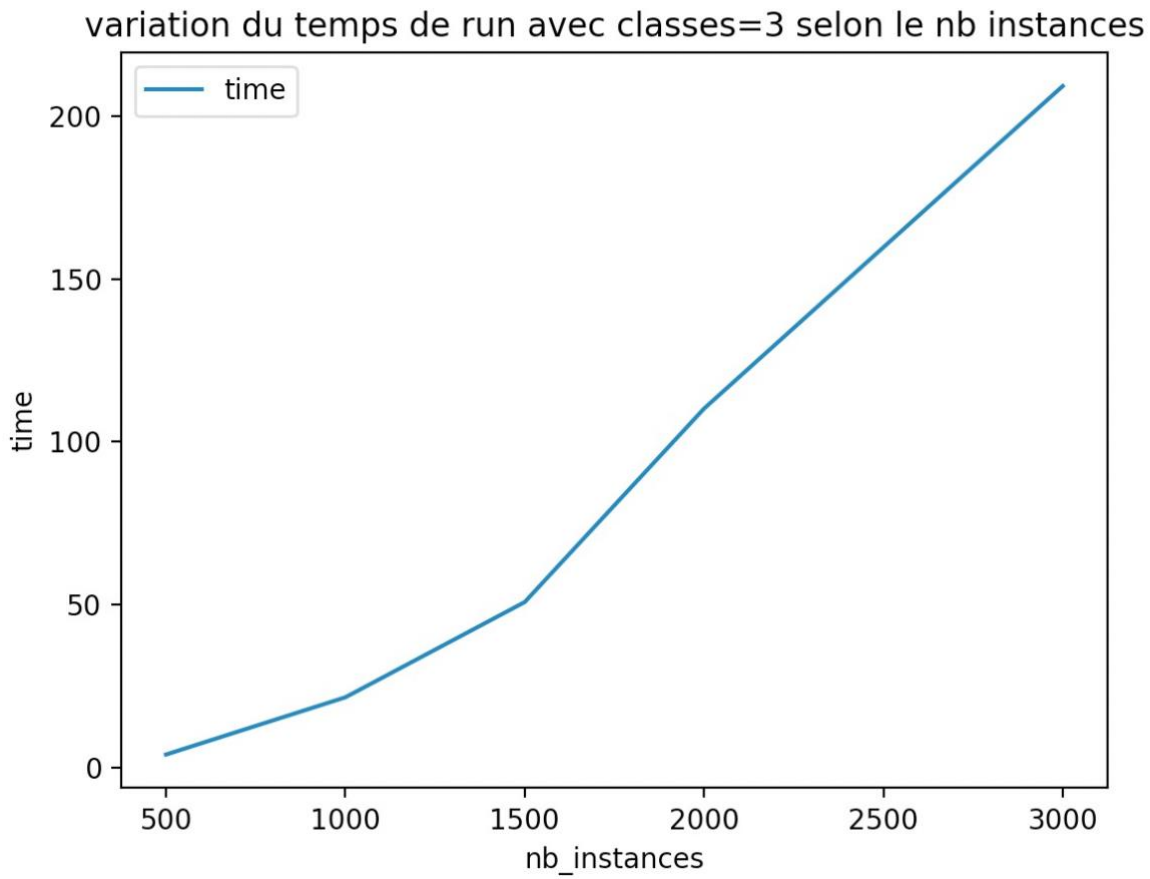
Nos résultats

Tout d'abord nous utilisons un script pour générer la donnée du problème. Ce même script est utilisé pour l'ensembles des algorithmes étudiés ici. L'image ci-dessous montre à quoi ressemblerai cette donnée sous forme de fichier CSV :

	0	1	2	3	4	accepted
0	15	12	16	9	17	0
1	0	14	11	14	10	0
2	3	14	10	7	14	0
3	2	19	16	13	6	1
4	16	9	6	18	9	0
5	19	15	14	19	0	1
6	20	0	9	11	14	0
7	11	16	17	3	12	0
8	18	13	4	16	17	0
9	14	15	15	8	6	1
10	8	12	20	3	13	0

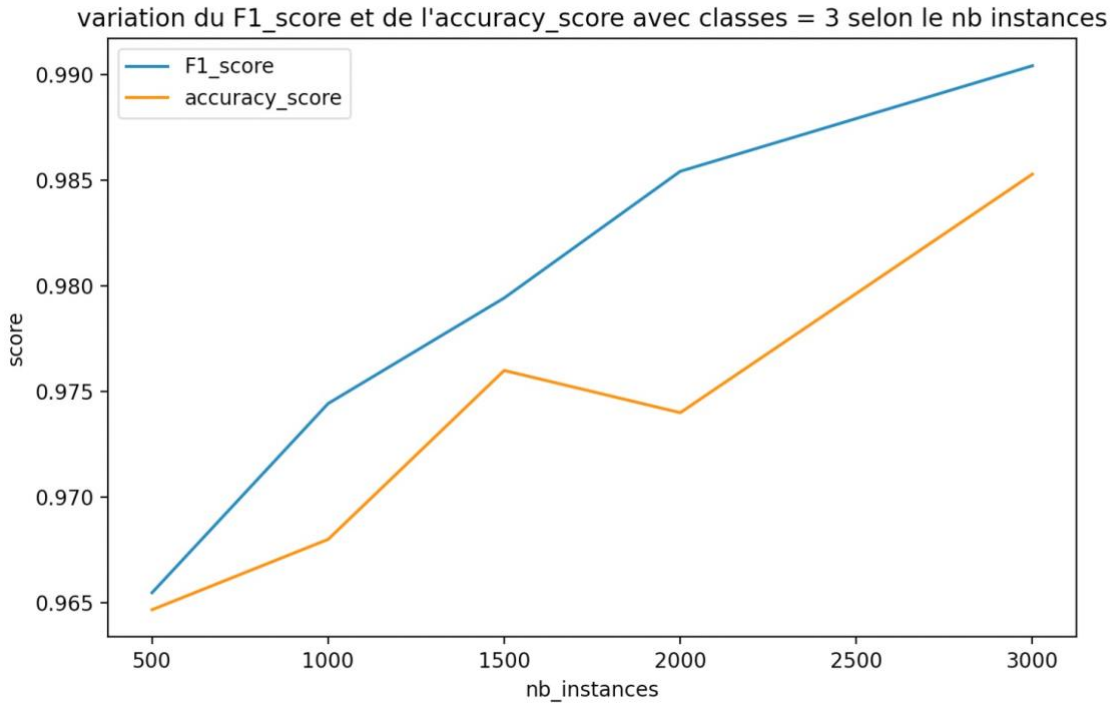
Pour les résultats présentés ci-dessous, on note que les valeurs sont des moyennes calculées sur trois essais.

Tout d'abord, comme on peut le voir sur le graphique ci-dessous, le temps de computation augmente considérablement de manière plus ou moins linéaire en fonction du nombre d'instances utilisées :



Cependant il est important de souligner que l'augmentation du nombre d'instances a un impact positif considérable sur les performances de l'algorithme. Comme on le voit sur le graphique ci-dessous, l'accuracy ainsi que

le F1-Score augmentent :

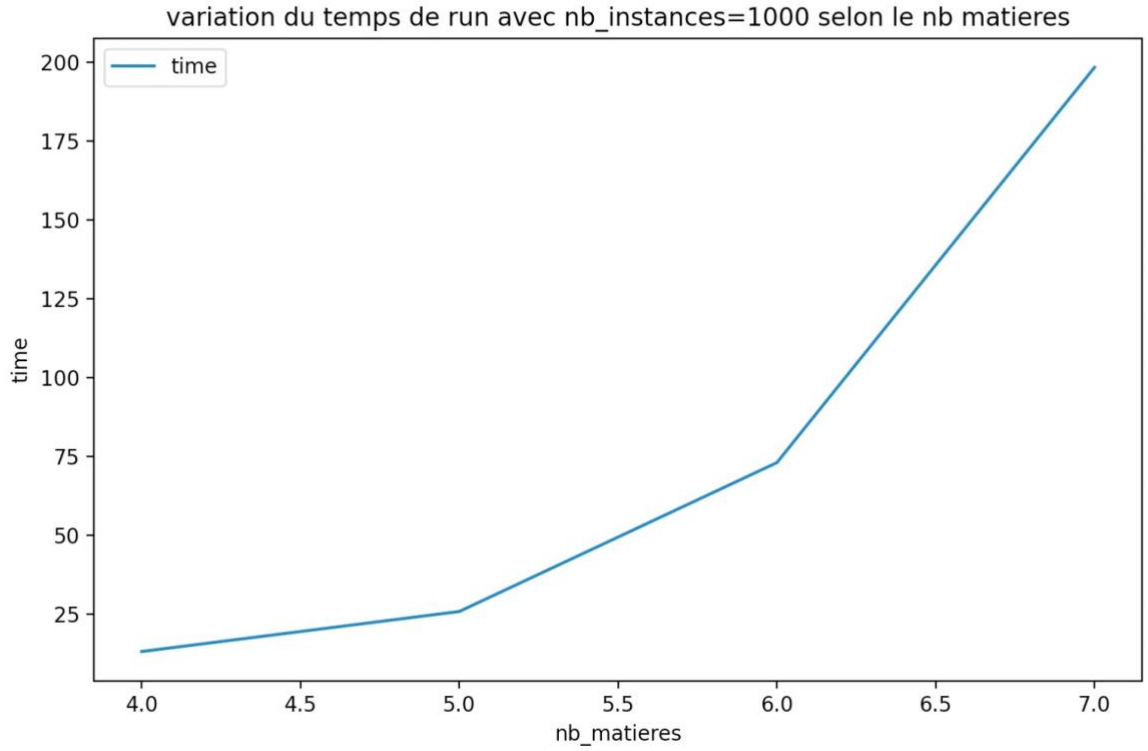


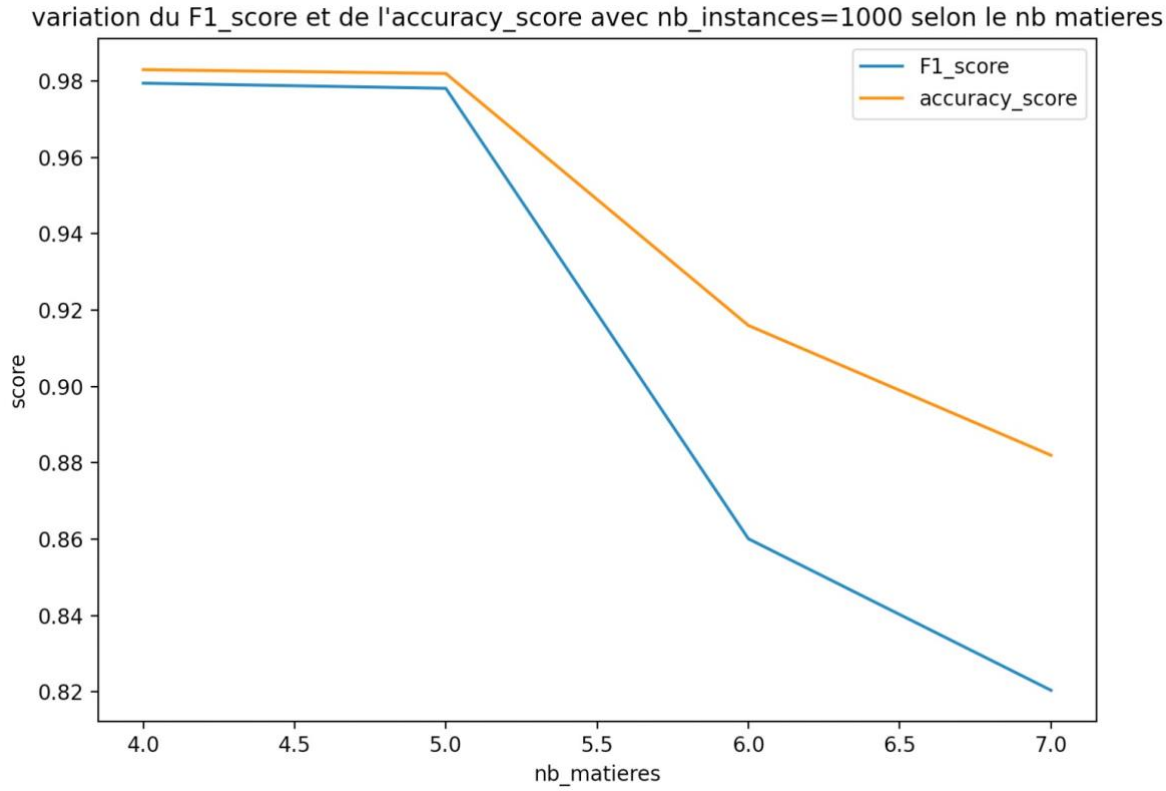
Dans notre script permettant de générer la donnée, nous avons inclut la possibilité d'introduire du bruit. L'image ci-dessous présente les résultats du Mr-sort sur un jeu de donné bruité à 3% :

```
Nombre de signaux bruités: 31
lambda truth: 0.59
lambda mrsort: 0.63
weight truth: ['0.20', '0.22', '0.16', '0.21', '0.21']
weight mrsort: ['0.25', '0.25', '0.00', '0.25', '0.25']
Temps d'exécution : 1.8e+02s
F1-score: 0.6024742041493113
accuracy: 0.66
```

On constate que c'est bien satisfiable mais que les performances baissent.

De plus, les deux courbes suivantes présentent le temps de computation ainsi que les performances en fonction du nombre de matières.





Comme on peut le constater, le nombre de matières impacte les performances ainsi que le temps de calcul considérablement.

L'algorithme NCS (version SAT)

Les modèles de tri non compensatoire (NCS) attribuent des alternatives à des classes sur la base de la façon dont elles se comparent à des profils multicritères séparant les classes consécutives.

Dans le cadre du cours de systèmes de décision nous étudions une méthode « efficace » pour déterminer les paramètres de ce modèle dans un cas multicritère.

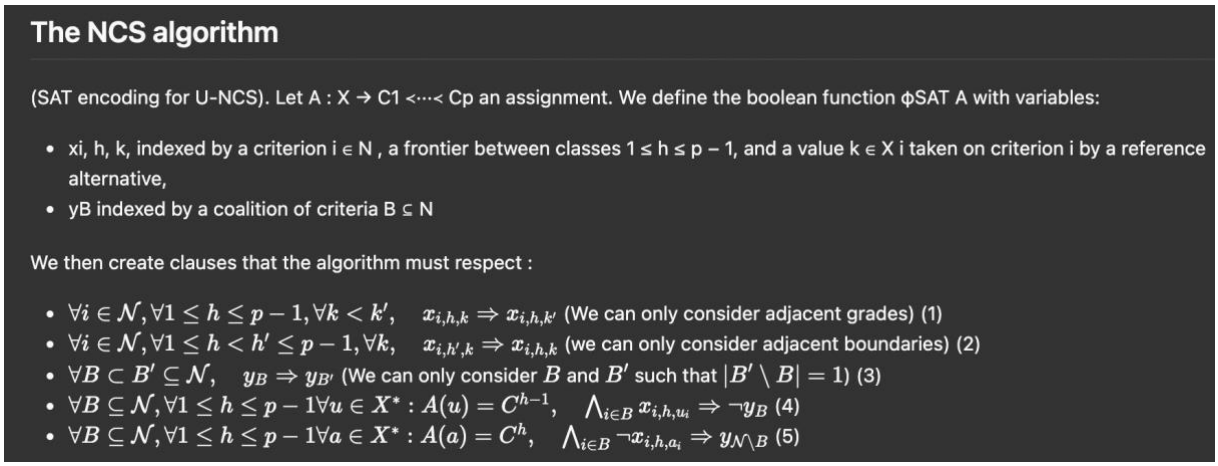
On note qu'il existe d'autres méthodes permettant d'apprendre ces paramètres. Certaines se basent sur des programmes de type MILP qui assurent une bonne restitution mais qui n'est pas applicable à de larges jeux de données.

D'autres méthodes basées sur des heuristiques peuvent être appliquées à de larges jeux de données mais n'assurent pas que les paramètres sont « les meilleurs » pour représenter les relations du jeu de données.

En se basant sur le document de l'annexe 2, nous allons donc voir une méthode de type SAT qui permet d'assurer de trouver les meilleurs paramètres tout en étant efficace sur de larges jeux de données.

Cette dernière a été testé et s'avère être plus performante en termes de temps de calcul par rapport aux solutions MILP existantes.

La solution proposée par le document de l'annexe 2 consiste à transformer le problème du choix des paramètres du NCS sous la forme d'un SAT. On note donc l'ajout des clauses suivantes :



NCS (version MaxSAT)

Dans la section précédente nous avons présenté le NCS sous forme SAT. Bien que les résultats soient prometteurs, cette approche ne convient pas au problème à partir de données réelles, car elle ne tolère pas la présence de bruit dans les données. Il existe de nombreuses raisons pour lesquelles les données d'entrée ne reflètent pas parfaitement le modèle (pour plus de détails voir la section 5.1 de l'annexe 2).

Dans cette section, nous la solution proposant une relaxation des formulations de décision : au lieu de trouver un modèle NCS qui restaure tous les exemples de l'ensemble d'apprentissage nous essayons de trouver le modèle qui en restaure le plus : d'où l'appellation MaxSAT.

Ainsi nous introduisons la variable « z » ainsi que trois clauses au programme SAT présenté précédemment.

this purpose, we define the following additional binary variables:

- 'z' variables, indexed by an alternative x, represent the set of alternatives properly classified by the inferred model, with the following semantic: $z_x = 1 \Leftrightarrow \alpha^{-1}(x) = NCS_\omega(x)$ i.e. the alternative x is properly classified

These variables are introduced in some clauses to serve as switches:

- For any exigence level $k \in [2.p]$, let $B \subseteq \mathcal{N}$ a coalition of criteria, and x an alternative assigned to C^{k-1} by α. If $z_k = 1$ and $B \subseteq \left\{i \in \mathcal{N} : x \in \mathcal{A}_i^k\right\}$ then $t_{B,k} = 0$. This leads to the following conjunction of clauses:

$$\phi_\alpha^{\widetilde{C5}} = \bigwedge_{B \subseteq \mathcal{N},\ k \in [2.p]} \bigwedge_{x \in \alpha^{-1}(C^{k-1})} \left(\bigvee_{i \in B} \neg a_{i,k,x} \vee \neg t_{B,k} \vee \neg z_x\right)$$

- For any exigence level $k \in [2.p]$, let $B \subseteq \mathcal{N}$ a coalition of criteria, and x an alternative assigned to C^k by α. If $z_k = 1$ and $B \subseteq \left\{i \in \mathcal{N} : x \in \mathcal{A}_i^k\right\}$ then $t_{\mathcal{N}\setminus B,k} = 0$. This leads to the following conjunction of clauses:

$$\phi_\alpha^{\widetilde{C6}} = \bigwedge_{B \subseteq \mathcal{N},\ k \in [2.p]} \bigwedge_{x \in \alpha^{-1}(C^{k})} \left(\bigvee_{i \in B} a_{i,k,x} \vee t_{\mathcal{N}\setminus B,k} \vee \neg z_x\right)$$

The objective in the MaxSAT formulation is to maximize the portion of alternatives properly classified, this is the subject of the following soft clause:

$$\phi_\alpha^{goal} = \bigwedge_{x \in \mathbb{X}^*} z_x \tag{8}$$

The MaxSAT extension of the first formulation is the conjunction of the first four clauses of the SAT formulation given in Definition 4.1 and clauses $\phi_\alpha^{\widetilde{C5}}$, $\phi_\alpha^{\widetilde{C6}}$ and ϕ_α^{goal}.

Clauses composing the conjunctions ϕ_α^{C1}, ϕ_α^{C2}, ϕ_α^{C3}, ϕ_α^{C4}, $\phi_\alpha^{\widetilde{C5}}$ and $\phi_\alpha^{\widetilde{C6}}$ are hard, associated to the weight $w_{\max}$, and we associate to ϕ_α^{goal} the weight w_1 such that $w_{\max} > |\mathbb{X}^\star| w_1$.

NCS avec critères non monotones :

Dans certaines applications de classification, les données sur les critères ne sont pas nécessairement monotones, comme c'est le cas dans la méthode NCS. Pour ce type de critère dit single-peaked, il ne s'agit pas de maximiser ou minimiser la valeur du critère, mais d'identifier un intervalle de "bonnes" valeurs.

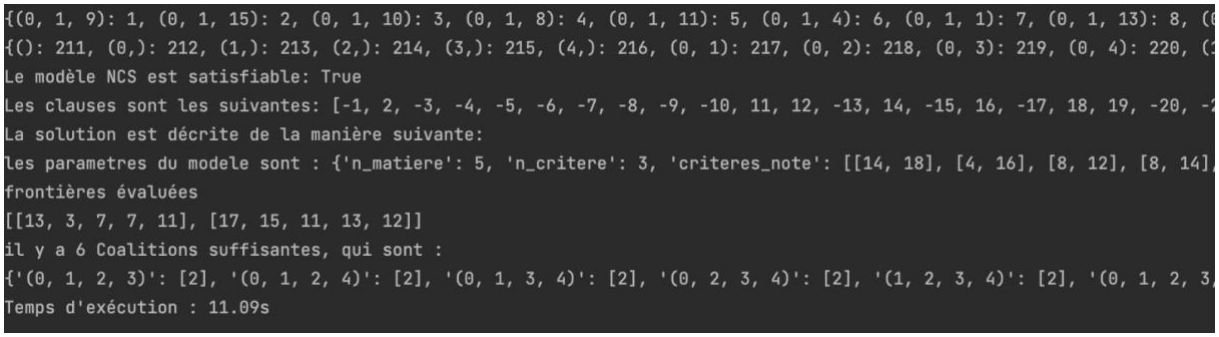
On note que la monotonie des valeurs sur les critères est encodée par une clause. On peut ainsi la modifier en remplaçant l'inégalité par un intervalle. Ainsi on obtient la clause suivante :

```
## Clauses 2a et 2b : Si un élève valide avec la note k,
## alors tous ceux qui valident avec une note k'>k valident aussi
## Adaptation sous forme d'intervalle pour la version single peaked

if single_peak:
    clause2a = []
    for i in subjects:
            for h in range(1, val):
                for k in X[i]:
                    for k1 in X[i]:
                        for k2 in X[i]:
                            if k < k1 and k1 < k2:
                                clause2a.append(
                                    [x[(i, h, k1)],-x[(i, h, k)],-x[(i, h, k2)]])
else:
    clause2a = [
        [x[i, h, p], -x[i, h, k]] for h in range(1, val) for i in subjects for k in X[i] for p in X[i] if k < p
    ]
```

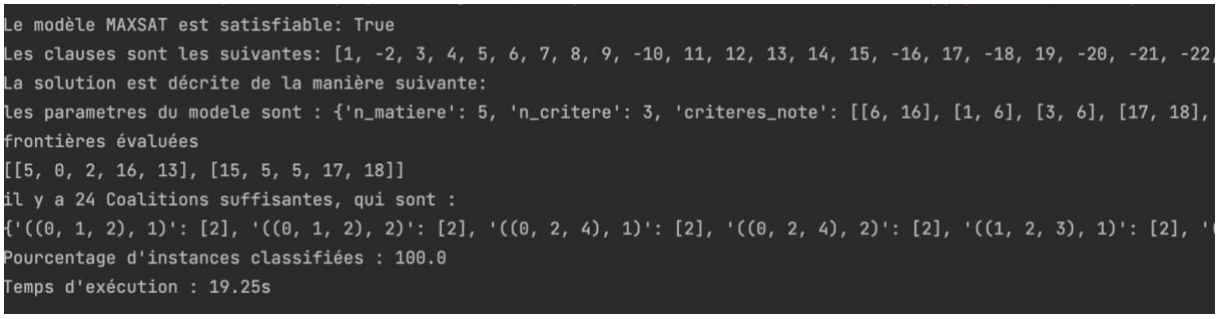
Nos résultats

Nous présentons ci-dessous les résultats de notre algorithme NCS SAT :

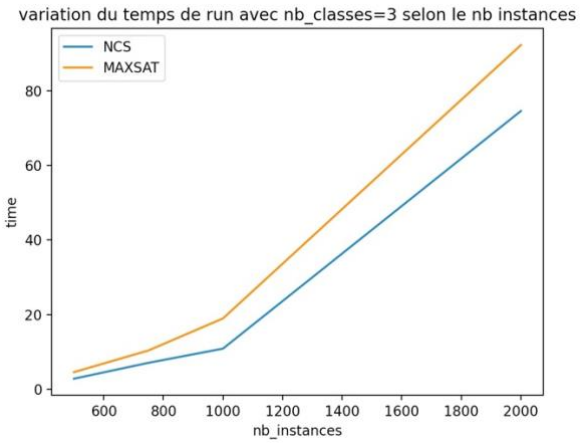


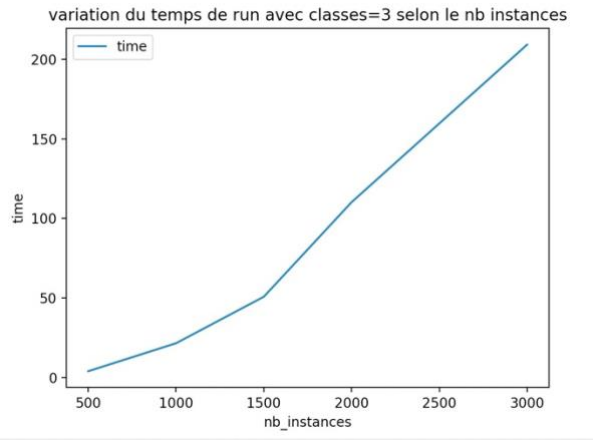
Cependant, comme attendu, en essayant d'ajouter du bruit dans le jeu de donné, l'algorithme NCS ne fonctionne pas : le problème est non satisfiable.

On peut voir par la suite les résultats de notre algorithme NCS MAXSAT :



Comme on peut le voir, le temps d'exécution est bien plus élevé avec la version MAXSAT. Cependant, en reprenant le graphe du temps d'exécution du MR-sort, on remarque que le NCS devient plus efficace plus le nombre d'instances augmente. La figure ci-dessous permet d'effectuer une comparaison du temps de calcul entre ces deux versions.





Mr-sort

Finalement, on peut voir sur la figure ci-dessous que pour la versions « single-peaked » le temps de calcul augmente avec le nombres d'instances :