CentraleSupélec
Dominante Informatique Numérique
Mention Intelligence Artificielle

Responsables du cours Systèmes de Décision
Vincent Mousseau – Wassila Ouerdane – Anäelle Wilczynski

PROJET SYSTÈMES DE DÉCISION : ADMISSION D'ÉTUDIANTS

Membres du groupe :
Alexandre Forestier
Morgane Senejko

01/02/2022

Introduction

L'objectif du projet est de comparer trois solveurs d'optimisation sur le thème d'admissions d'étudiants. Les jeux de données générés sont constitués de notes d'étudiants sur différentes matières. Selon ces notes, et à l'aide des coefficients pour chaque matière ainsi qu'un seuil d'admissibilité, les étudiants appartiennent à une certaine classe (par exemple : Admis ou Refusé dans le cas de deux classes).

Dans un premier temps, nous décrirons la manière dont nous générons nos datasets, qui seront utilisés par nos modèles.

Ensuite, nous décrirons chacun des trois modèles que nous avons implémentés : le Inv-MR-Sort à l'aide d'un solveur d'optimisation (Gurobi), le Inv-NCS à l'aide d'un solveur SAT (Gophersat), et enfin le Inv-NCS avec une formulation MaxSAT.

Puis nous comparerons ces trois méthodes en termes de temps de calcul et de capacité de généralisation, mais également leur capacité à prendre en compte des données bruitées.

Enfin, nous discuterons sur les résultats obtenus et les pistes d'amélioration.

I/ Génération de dataset

Pour que les datasets que nous générons soient résolvables par nos différents solveurs, il est important de pouvoir leur donner un dataset cohérent. C'est-à-dire qu'un étudiant n'ayant que des notes inférieures à un autre étudiant ne pourra pas être classé dans une catégorie supérieure.

Pour cela, nous fixons tout d'abord le nombre d'étudiants, le nombre de critères (matières) et le nombre de classes. Ces valeurs sont des arguments d'entrée de notre fonction de génération de dataset. Ensuite, nous générons de façon aléatoire des coefficients pour chaque matière, une liste de frontières de classes ordonnées, une variable lambda correspondant à la somme des coefficients minimale à avoir pour être catégorisé dans une classe et enfin nous générons de façon aléatoire des notes aux étudiants. Cela nous donne donc une matrice de notes, dont le nombre de lignes correspond au nombre d'élèves et le nombre de colonnes au nombre de critères. Ensuite, la meilleure classe possible est attribuée à chaque étudiant (à l'aide d'une fonction spécifique). Cela nous retourne une liste ayant comme taille le nombre d'étudiants.

Ces deux résultats sont convertis en une liste de matrices correspondant chacune à une classe donnée. Nous aurons donc finalement une matrice par classe, le nombre de lignes correspondant donc au nombre d'étudiants appartenant à cette classe et le nombre de colonnes correspondant au nombre de critères.

II/ Description des modèles étudiés

1) Inv-MR-Sort

MR-Sort (Majority Rule Sorting) est une méthode de classement multi-critères permettant de classer des objets, dans notre cas des étudiants, parmi des catégories ordonnées, à partir de certains paramètres. Nous possédons justement un dataset d'étudiants qui ont des notes sur différentes matières et qui ont une classe attribuée (par exemple Validé ou Non validé dans le cas de deux classes). Pour cette partie, nous avons repris les formulations du papier "Learning the Parameters of a Multiple Criteria Sorting Method Based on a Majority Rule", écrit par Agnès Leroy et al, datant de 2011.

Un modèle Inv-MR-Sort permet à partir de ce dataset de retrouver les paramètres suivants :

- $\mathbf{b_i}$ correspondant aux frontières du $i^{ème}$ critère.
- $\mathbf{w_i}$ correspondant aux poids appliqué au $i^{ème}$ critère.
- $\boldsymbol{\lambda}$ correspondant au critère de majorité pour pouvoir valider.

Nous introduisons ensuite certaines variables nécessaires à la formalisation du problème :

- $\mathbf{c_{i,j}}$ = { 0 si $a_{i,k} < w_j$, w_j sinon}
- $\delta_{,ij}$ = {0 si $a_{i,k} < w_j$, 1 sinon}
- x_j qui viennent compenser l'écart à la majorité quand l'élève a validé
- y_j qui viennent compenser l'écart à la majorité quand l'élève n'a pas validé
- un ε qui va nous permettre de transformer des inégalité strictes en égalités
- un M qui est une grande constante positive arbitraire fixée

Nous cherchons ensuite à maximiser α avec le problème linéaire suivant que nous donnons à Gurobi: max α tel que:

- $\Sigma_{i\in N}\, c_{i,j} + x_j + \varepsilon = \lambda$ $\quad \forall a_j \in A^{*1}$ (élèves acceptés)
- $\Sigma_{i\in N}\, c_{i,j} = \lambda + y_j$ $\quad \forall a_j \in A^{*2}$ (élèves refusés)
- $\alpha \le x_j$, $\alpha \le y_j$ $\quad \forall a_j \in A^*$
- $c_{i,j} \le w_i$ $\quad \forall a_j \in A^*, \forall i \in N$
- $c_{i,j} \le \delta_{i,j}$ $\quad \forall a_j \in A^*, \forall i \in N$
- $c_{i,j} \ge \delta_{i,j} - 1 + w_i$ $\quad \forall a_j \in A^*, \forall i \in N$
- $M\,\delta_{i,j} + \varepsilon \ge g_i(a_j) - b_i$ $\quad \forall a_j \in A^*, \forall i \in N$
- $M(\delta_{i,j} - 1) \le g_i(a_j) - b_i$ $\quad \forall a_j \in A^*, \forall i \in N$
- $w_i \in [0, 1]$ $\quad \forall i \in N$
- $c_{i,j} \in [0, w_j]$, $\delta_{i,j} \in \{0, 1\}$ $\quad \forall a_j \in A^*, \forall i \in N$
- x_j , $y_j \in R$ $\quad \forall aj \in A^*$
- $\alpha \in R$

Une fois le modèle résolu, il nous suffit de récupérer les $\mathbf{b_i}$ les $\mathbf{w_i}$ et $\boldsymbol{\lambda}$ puis de les appliquer à notre dataset de test. Avec numpy il s'agit d'un one Liner: (X_test >= b) @ w >= λ

2) Inv-NCS avec une formulation SAT

Le problème auquel nous avons été confronté peut également se résoudre à l'aide d'un modèle Inv-NCS (Inv-Non Compensatory Sorting Model). Il s'agit d'une méthode MCDA (Multiple Criteria Decision Analysis) basée sur des relations de surclassement.

Pour cela, nous avons dû représenter le problème à l'aide d'une formulation SAT, c'est-à-dire avec un ensemble de formules de logique propositionnelle. Le modèle créé devait alors déterminer s'il existe une assignation des variables propositionnelles qui rend les formules vraies.

Pour cette partie, nous avons repris les formulations du papier "An efficient SAT formulation for learning multiple criteria non-compensatory sorting rules from examples", écrit par Belahcène et al, datant de 2018.

Description des variables et des clauses

Deux types de variables binaires sont utilisées pour notre modèle :

- x(i, h, k) : True ssi la note k sur le critère i est au-dessus de la frontière h, False sinon.
- yB : True ssi la coalition de critères B est suffisante, False sinon.

Afin d'être exploitables par le Solveur SAT, toutes ces variables ont été traduites en un dictionnaire dont les clés sont nos variables et les valeurs associées des nombres entiers correspondant au numéro de la variable.

Les clauses de notre modèle sont une conjonction de 5 familles de clauses.

1) Clauses 1 : Échelle ascendante

Pour chaque critère i et pour toutes les frontières entre les classes adjacentes h, pour chaque notes ordonnées k<k' : **x(i,h,k') ∨ ¬x(i,h,k)**
Cela signifie que pour 2 notes k et k' sur un critère i, telles que k<k', soit k n'est pas au-dessus de la frontière h (dans ce cas on a ¬x(i,h,k)), soit k est au-dessus de la frontière h donc k' est également au-dessus de la frontière h (dans ce cas on a x(i,h,k')).

Cette clause se ramène à l'implication x(i,h,k) => x(i,h,k'). Avoir une note k sur le critère i supérieure à la frontière h implique que la note k' (>k) sur le critère i est également supérieure à la frontière h.

2) Clauses 2 : Hiérarchie des profils

Pour chaque note k sur un critère i et pour toutes les paires de frontières telles que $1 \leq h < h' < p-1$: **$x(i,h,k) \vee \neg x(i,h',k)$**
Cela signifie que pour une note k donnée sur le critère i, soit cette note est supérieure à la frontière h (dans ce cas on a x(i,h,k)), soit elle est inférieure à cette même frontière et donc elle est également inférieure à la frontière du dessus (dans ce cas on a ¬x(i,h',k)).

Cette clause se ramène à l'implication x(i,h',k) => x(i,h,k). En effet, avoir une note k sur le critère i supérieure à la frontière h' implique que cette note est supérieure également à la frontière h (<h').

3) Clauses 3 : Force de coalitions

Pour toutes coalitions de critères B et B' telles que $B \subset B' \subseteq N$: **$yB' \vee \neg yB$**
Cela signifie que soit la coalition B n'est pas suffisante (¬yB), soit elle l'est et dans ce cas B' l'est également (yB').

Cette clause se ramène à l'implication yB => yB'. La coalition B (= {Mathématiques} par exemple) étant suffisante, la coalition B' (= {Mathématiques, Français} par exemple) est également suffisante.

Les familles de clauses 4 et 5 sont plus spécifiques au dataset que nous avons généré.

4) Clauses 4

Pour toutes coalitions de critères $B \subseteq N$ et pour toutes frontières $1 \leq h \leq p-1$ et pour tout étudiant *u* juste en-dessous de la frontière h : **$Vi \in B(\neg x(i,h,ui)) \vee \neg yB$**
Cette clause signifie que soit la coalition B n'est pas suffisante, soit elle l'est et dans ce cas il y a au moins une note de *u* (qui a été évalué au-dessus de la frontière h-1) qui ne peut pas être au-dessus de la frontière h.
Cette clause se ramène à l'implication : $yB \Rightarrow Vi \in B(\neg x(i,h,ui))$
Elle permet d'assurer que chaque étudiant ne peut pas être approuvé par une coalition suffisante à un niveau d'exigence au-dessus de celui correspondant à sa catégorie.

5) Clauses 5

Pour toutes coalitions de critères B ⊆ N et pour toutes frontières 1 ≤ h ≤ p-1 et pour tout étudiant *a* juste au-dessus de la frontière h : **Vi∈B(x(i,h,ai)) V yN\B**
Elle permet d'assurer que chaque étudiant est approuvé par une coalition suffisante à un niveau d'exigence correspondant à sa catégorie.

Afin de lancer la résolution, l'ensemble des clauses a été transformé en un fichier au format DIMACS qui a pu être exécuté par Gophersat.

Exploitation des sorties du modèle

Le solveur nous retourne en sortie l'ensemble des variables binaires que nous avons créées avec leur valeur de vérité : True ou False.
Afin d'exploiter ce résultat, nous avons créé une fonction permettant de convertir les sorties obtenues en une liste indiquant la classe obtenue pour chaque élève. Les différentes étapes sont décrites dans la figure 1 ci-dessous.

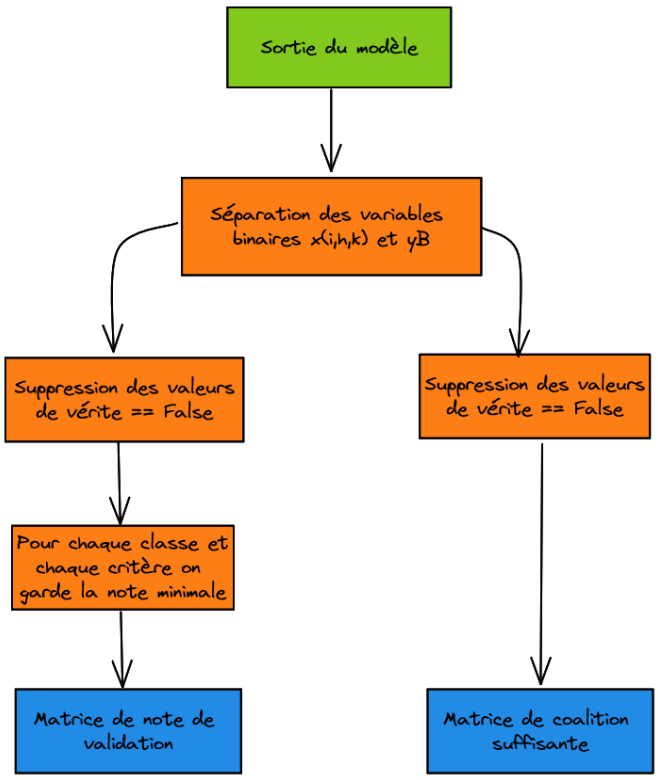

Figure 1 : Différentes étapes pour l'exploitation des résultats du modèle Inv-NCS

3) Inv-NCS avec une formulation MaxSAT

La formulation MaxSAT est la version optimisée du SAT, et permet de résoudre des problèmes de décision qui étaient insatisfiables avec le SAT. En effet, lorsque le learning set n'est pas totalement représentable, le solveur SAT renvoyait False. L'objectif du MaxSAT est de représenter le plus grand nombre d'exemples du learning set et de restituer un modèle Inv-NCS (le meilleur possible) dans tous les cas.

Ajout d'une nouvelle variable binaire et modification des clauses non structurelles

Pour cela, une nouvelle variable binaire z peut être créée afin de représenter l'ensemble des classifications données aux étudiants par le modèle. Si la clause est respectée, la variable associée z(x) prendra la valeur True et False sinon.
Les clauses structurelles du SAT (1 à 3) sont les mêmes mais les clauses 4 et 5 peuvent être modifiées afin de prendre en compte les nouvelles variables binaires z(x).
D'après le papier "Learning non-compensatory sorting models using efficient SAT/MaxSAT formulations", écrit par Tlili et al, datant de 2022, en gardant les mêmes noms de variables que la partie précédente, nous obtenons :

- Clause 4bis :

Pour toutes coalitions de critères $B \subseteq N$ et pour toutes frontières $1 \leq h \leq p-1$ et pour tout étudiant *u* juste en-dessous de la frontière h : $\mathbf{\bigvee_{i \in B}(\neg x(i,h,u)) \vee \neg y_B \vee \neg z(u)}$

- Clause 5bis :

Pour toutes coalitions de critères $B \subseteq N$ et pour toutes frontières $1 \leq h \leq p-1$ et pour tout étudiant *a* juste au-dessus de la frontière h : $\mathbf{\bigvee_{i \in B}(x(i,h,a)) \vee y_{N \setminus B} \vee \neg z(a)}$

- Clause 6 :

Par ailleurs, une clause "objectif" est rajoutée, et permet de représenter le fait qu'il faut que notre modèle Inv-NCS maximise la proportion d'étudiants correctement classifiés : $\mathbf{\bigwedge_{x \in X} z(x)}$

En ajoutant donc notre nouvelle variable z, une 6ème clause ainsi qu'en modifiant les clauses 4, 5, il est alors possible d'obtenir un modèle Inv-NCS même dans le cas où le dataset contiendrait des incohérences, mais le modèle obtenu sera le meilleur possible car il essaye de maximiser le nombre d'étudiants correctement classifiés (clause 6).

Autre approche : modification du fichier DIMACS

Afin de résoudre un problème MaxSAT, il est également possible de modifier tout simplement le fichier au format DIMACS contenant l'ensemble des variables et clauses de notre problème (celles présentées dans la partie SAT → on oublie ici l'ajout d'une nouvelle variable et 6ème clause ainsi que la modification des clauses 4 et 5). L'extension du fichier passera alors de .cnf à .wcnf. (w pour weighted MaxSAT).
La première ligne du fichier (hors commentaires) est de la forme `p wcnf #VARIABLES #CLAUSES` avec #VARIABLES : le nombre de variables et #CLAUSES : le nombre de clauses.

Ensuite, chaque ligne du fichier correspond à une clause composée d'entiers (positifs ou négatifs) et se termine par 0, comme pour les fichiers .cnf. La seule différence est qu'un poids est associé à chaque clause en début de ligne.

Les clauses structurelles (familles de clauses 1 à 3) permettent d'assurer que le modèle de classification est bien Inv-NCS. Tandis que les familles de clauses 4 et 5 permettent d'assurer que les étudiants du dataset sont correctement classifiés. Ces dernières doivent donc avoir un poids moins important que les premières. En effet, le but ici est de trouver un modèle Inv-NCS qui garantit le maximum de classification correcte, mais il est totalement acceptable que certaines clauses des familles 4 et 5 ne soient pas respectées.

Nous avons implémenté cette deuxième méthode en utilisant des poids égaux à 100000 pour les clauses structurelles et des poids égaux à 1 pour les autres.

III/ Comparaison des différents modèles

1) Temps de calcul en fonction du nombre d'étudiants

Tout d'abord, nous avons voulu comparer nos modèles en termes de temps de calcul. Celui-ci dépend du dataset donné en entrée, dépendant lui-même du nombre d'étudiants, du nombre de critères (matières) et du nombre de classes.
Notre implémentation du modèle Inv-MR-Sort ne fonctionnant que dans le cas de 2 classes, nous avons fixé ce nombre (classes_count = 2).
Dans un premier temps, nous avons fixé le nombre de critères à 4 et fait varier le nombre d'étudiants. Vous pouvez observer les temps d'entraînement obtenus pour nos deux modèles sur les figures 2 et 3.

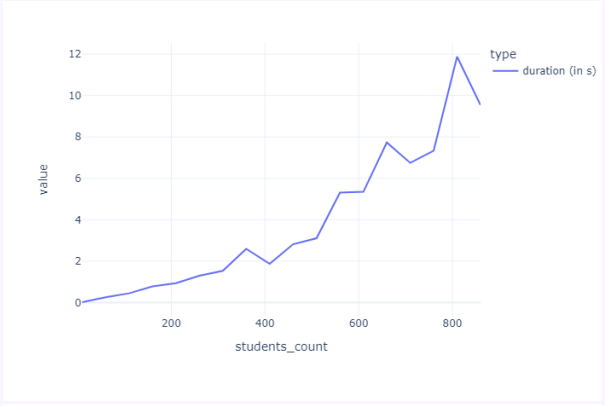

Figure 2 : Temps de calcul (en secondes) en fonction du nombre d'étudiants pour le modèle Inv-MR-Sort

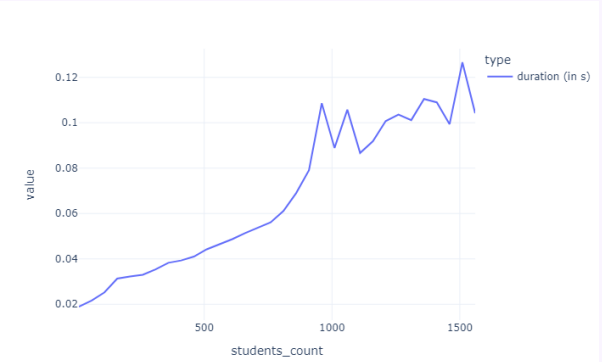

Figure 3 : Temps de calcul (en secondes) en fonction du nombre d'étudiants pour le modèle Inv-NCS

A première vue, les courbes semblent similaires. Mais elles ne correspondent pas à la même échelle. En effet, le modèle Inv-NCS est 100 fois plus rapide à s'entraîner que le modèle Inv-MR-Sort. Pour un dataset contenant 800 élèves, le modèle Inv-MR-Sort prend 10 secondes pour s'entraîner contre 0,1 secondes pour le Inv-NCS.

2) Temps de calcul en fonction du nombre de matières

Nous avons fixé le nombre d'étudiants à 600 (et conservé 2 classes) et fait varier le nombre de critères. Vous pouvez observer les résultats concernant le temps d'entraînement des modèles Inv-MR-Sort et Inv-NCS sur les figures 4 et 5.

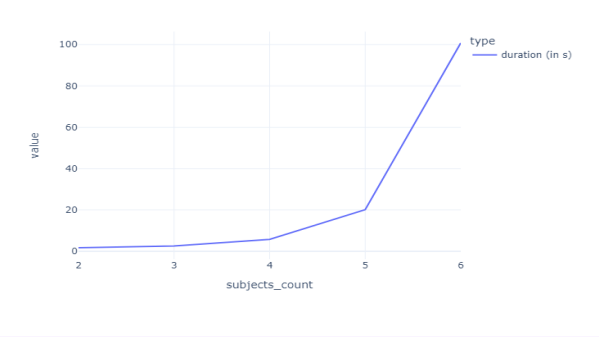

Figure 4 : Temps de calcul (en secondes) en fonction du nombre de critères pour le modèle Inv-MR-Sort

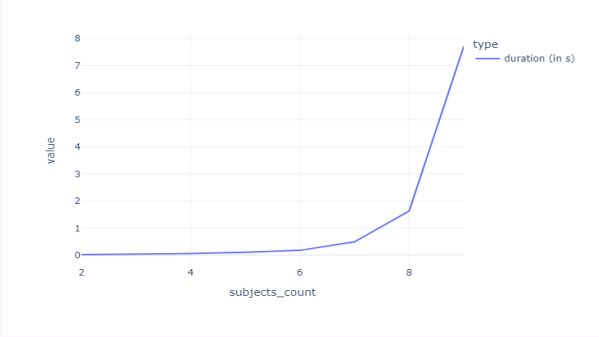

Figure 5 : Temps de calcul (en secondes) en fonction du nombre de critères pour le modèle Inv-NCS

Les tendances sont les mêmes : plus le nombre de critères augmente, plus les modèles sont longs à entraîner (avec une tendance exponentielle). Le modèle Inv-NCS est encore environ 100 fois plus rapide que le Inv-MR-Sort. En effet, pour 6 critères différents, le premier ne met qu'une demi-seconde alors que le deuxième presque 1 minute.

3) Performances en fonction du nombre d'étudiants

Pour cette comparaison, le nombre de critères est fixé à 4, le nombre de classes est fixé à 2, et nous faisons varier le nombre d'étudiants. Les résultats obtenus en figures 6 et 7 présentent les performances selon différentes métriques pour nos deux modèles.

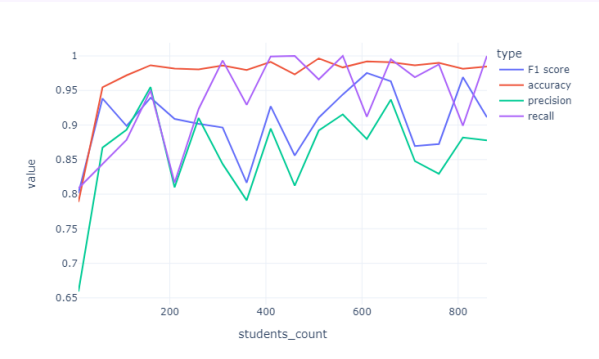

Figure 6 : Performances en fonction du nombre d'étudiants pour le modèle Inv-MR-Sort

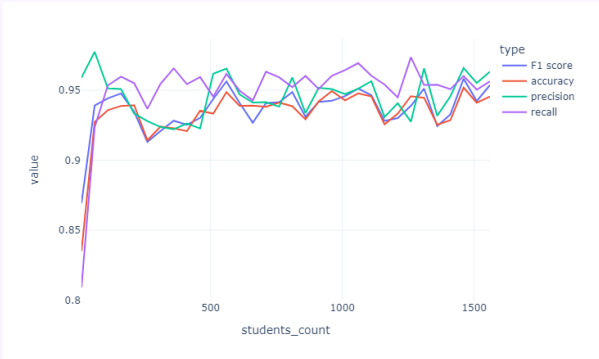

Figure 7 : Performances en fonction du nombre d'étudiants pour le modèle Inv-NCS

Le modèle Inv-MR-Sort présente une meilleure accuracy que le modèle Inv-NCS. En revanche, la précision n'est pas très bonne et oscille entre 80 et 90%.
Concernant le modèle Inv-NCS, sa précision est indexée sur l'accuracy.

4) Performances en fonction du nombre de critères

Pour cette comparaison, le nombre d'étudiants est fixé à 600, le nombre de classes est fixé à 2, et nous faisons varier le nombre de critères. Les résultats obtenus en figures 8 et 9 présentent les performances selon différentes métriques pour nos deux modèles.

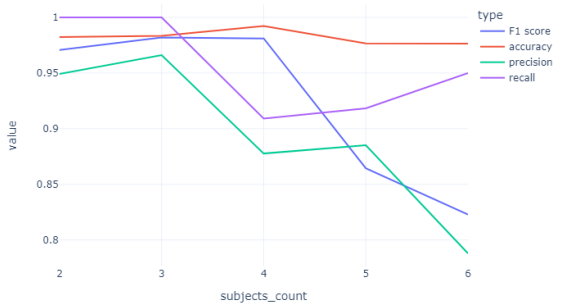

Figure 8 : Performances en fonction du nombre de critères pour le modèle Inv-MR-Sort

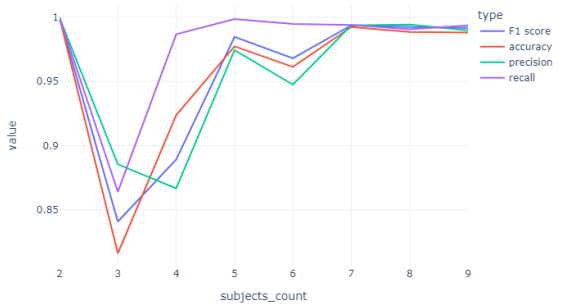

Figure 9 : Performances en fonction du nombre de critères pour le modèle Inv-NCS

Les résultats de ce dernier exemple est très intéressant : il montre que nos 2 modèles sont complémentaires. Inv-MR-Sort propose de bonnes performances sur des modèles avec peu de critère tandis que le modèle Inv-NCS est meilleur sur des dataset contenant plus de critères.

5) Performances en fonction du bruit

Nous avons fixé le nombre d'étudiants à 500, le nombre de critères à 4 et le nombre de classes à 2. Nous avons généré un dataset que nous avons séparé en train et en test et nous avons ajouté du bruit au dataset de test en ajoutant à toutes les notes un bruit gaussien centré dont nous avons fait varier l'écart-type. Les résultats obtenus en figures 10 et 11 présentent les performances en fonction du bruit selon différentes métriques pour nos deux modèles.

Nos observations précédentes sont confirmées, le Inv-MR-Sort est meilleur sur des données peu bruitées, en revanche le modèle Inv-NCS gère mieux les données plus bruitées.

Figure 10 : Performances en fonction du bruit pour le modèle Inv-MR-Sort

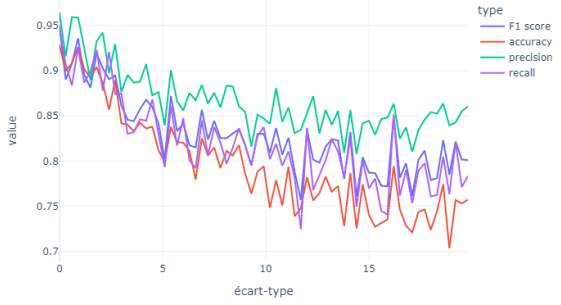

Figure 11 : Performances en fonction du bruit pour le modèle Inv-NCS

6) Performances du solveur MaxSat

Nous avons fixé le nombre d'étudiants à 300, le nombre de critères à 4 et le nombre de classes à 2. Nous avons généré un dataset que nous avons séparé en train et en test et nous avons ajouté un bruit gaussien centré dont nous avons fait varier l'écart type aux datasets de train et de test. La figure 12 représente le nombre de clauses ignorées par le modèle Inv-NCS en fonction du bruit et la figure 13 les performances du modèle en fonction du bruit.

Figure 12 : Nombre de clauses ignorées en fonction du bruit pour le modèle Inv-NCS avec formulation MaxSAT

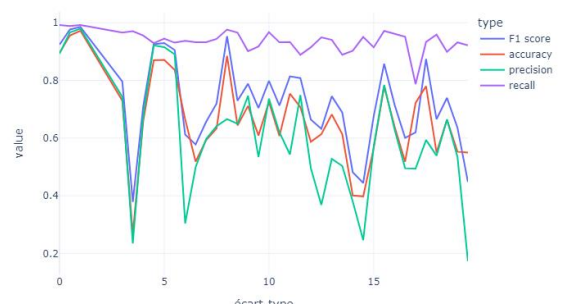

Figure 13 : Performances en fonction du bruit pour le modèle Inv-NCS avec formulation MaxSAT

Nos intuitions ont été confirmées, le nombre de clauses ignorées est proportionnel au bruit appliqué. En revanche, les performances sont très variables d'une itération à l'autre, nous observons une tendance de baisse de l'accuracy avec l'augmentation du bruit.

IV/ Adaptation à des critères single-peaked

Dans certaines applications de classification, il arrive que certains critères ne soient ni à minimiser ni à maximiser. Les bonnes valeurs doivent être situées dans un intervalle spécifique et c'est dans le cas où elles sont inférieures ou supérieures à ces limites qu'elles appartiennent à une autre catégorie.

Il est possible de modifier la formulation SAT et MaxSAT de Inv-NCS pour prendre en compte de telles données. Il suffit de modifier la famille de clauses 1 :

Pour chaque critère i et pour toutes les frontières entre les classes adjacentes h, pour chaque notes ordonnées k < k'< k'' : **x(i,h,k') ∨ ¬x(i,h,k) ∨ ¬x(i,h,k'')**

Cette nouvelle clause se ramène à l'implication : x(i,h,k) /\ x(i,h,k'') => x(i,h,k'), sachant k<k'<k''.

Conclusion

La résolution de notre problème par un modèle Inv-NCS est beaucoup plus rapide et a l'avantage d'être très adaptable.
En effet, de nouvelles variables peuvent être créées et les clauses de la formulation SAT modifiées pour obtenir une formulation MaxSAT permettent de résoudre des datasets qui contiennent des incohérences. Ainsi ils sont en mesure de traiter des datasets qui n'étaient pas résolvables par un modèle Inv-MR-Sort ou par un modèle Inv-NCS à formulation SAT.
Par ailleurs, nous avons vu en dernière partie qu'il suffisait de modifier une seule clause de notre formulation SAT afin de gérer un dataset contenant des critères single-peaked.

Les performances en termes de classification de nos implémentations sont assez proches. Nous préférons une implémentation Inv-NCS lorsque :

- Nous avons des gros datasets
- Nous avons une grande quantité de critères
- Nous avons des données assez bruitées

En revanche nous préférons une implémentation Inv-MR-Sort lorsque :

- Nous avons des petits datasets
- Le nombre de critères évalués est faible
- Les données ne sont pas bruitées

Enfin concernant la rapidité d'apprentissage de nos implémentations, Inv-NCS est bien plus rapide, cela peut s'expliquer en (toute) petite partie par l'utilisation de Gophersat qui est implémenté en GO, langage plus optimisé que le python utilisé par Gurobi. Le facteur le plus influent reste la méthode et travailler avec des variables booléennes réduit en quelque sorte l'espace de recherche. De plus, du point de vue calculatoire, il est plus rapide de travailler avec des booléen que des contraintes linéaires et leur espace de solutions associé.