Résistance aux antibiotiques

Prédiction par apprentissage statistique

Vadim BERTRAND, Cheikh-Darou BEYE

9 janvier 2023

Sommaire

1 L'antibiorésistance

Les antibiotiques sont développés pour contrer les infections dues aux bactéries. Certaines bactéries peuvent acquérir une résistance à des antibiotiques, via l'obtention de nouveaux gènes ou par la mutation de gènes existants. L'antibiorésistance représente un grand risque pour la santé publique, il est donc important de la limiter. Cela passe notamment par une meilleure compréhension des mécanismes de résistances comme l'identification de gènes résistants ou des bactéries résistantes. Cette résistance de certaines bactéries à des antibiotiques peut être traitée comme une tâche de classification en apprentissage statistique.

Dans cette étude nous aurons à notre disposition un jeu de données constitué de 3 matrices de régresseurs et une matrice réponse pour 414 bactéries :

- X_gpa, codant la présence ou l'absence de 16005 gènes,
- X_nsps, codant la présence ou l'absence de 72236 mutations génétiques,
- X_genexp, représentant l'expression génétique de 6026 gènes ;
- Y, codant la résistance ou la sensibilité à 5 antibiotiques : la Ceftazidime, la Ciprofloxacine, la Colistine, le Méropenème et la Tobramycine.

Notre objectif est de prédire la résistance des bactéries aux antibiotiques à partir des régresseurs et d'identifier quelles matrices de régresseurs sont les plus intéressantes pour cette tâche, selon l'antibiotique considéré.

Dans un premier temps, nous procéderons à une courte exploration des données. Puis, nous détaillerons notre démarche : pré-traitements utilisés sur les données, proposition d'approches de réduction de dimension, classifieurs considérés et mise en œuvre via la librairie *scikit-learn*. Enfin, nous présenterons les résultats obtenus et nous proposerons quelques pistes d'amélioration.

2 Exploration des données

Avant de nous lancer dans la prédiction de la résistance aux antibiotiques, nous avons souhaité nous pencher sur les données que nous manipulons.

Naturellement nous avons commencé par observer les types de données que nous manipulons et l'éventuelle présence de données manquantes. Sur les 4 matrices à notre disposition, 3 contiennent des données binaires (Y, X_gpa, X_nsps) tandis que X_genexp contient des données quantitatives.

Comme le montre la table 1, les matrices de régresseurs ne contiennent pas de données manquantes, mais certaines informations de résistance aux antibiotiques sont manquantes, notamment pour la Ceftazidim avec 20% de données absentes. Etant donné que la taille du jeu de données est réduite, que nous procèderons par la suite à une validation croisée et que nous ne disposerons donc pas d'un jeu de test, nous avons choisi de ne pas imputer les données manquantes afin d'éviter de fausser la généralisation des résultats. Par conséquent, les bactéries dont la résistance à un antibiotique est manquante ne seront pas utilisées lors de l'évaluation des classifieurs sur l'antibiotique correspondant.

Nous pouvons également observer sur la table 1 que les variables réponses ne sont pas toujours équilibrées : 2 fois plus de bactéries résistantes à la Méropenem, et à l'inverse 2 à 3 fois plus de bactéries susceptibles à la Tobramycin et la Colistin. De même, les gènes ou les mutations sont bien plus souvent absentes que présentes.

Table 1: Résumé des variables réponses et des régresseurs.

	Résistance					Présence		
	Tobramycin	Ceftazidim	Ciprofloxacin	Meropenem	Colistin	gpa	snps	genexp
# **NA**	8	80	56	60	0	0	0	0
# **VRAI**	130	165	199	244	85	3581	8218	NaN
# **FAUX**	276	169	159	110	329	12424	64018	NaN

Pour aller un peu plus loin, nous avons représenté nos données regroupées par clustering hiérarchique avec des cartes de chaleur afin de faire apparaître des structures. La figure 1 correspondant à la carte de chaleur ainsi obtenue pour la matrice X_gpa permet par exemple de supposer que cette matrice porte de l'information intéressante pour prédire la résistance à la Tobramycin et la Ciprofloxacine, mais probablement moins pour la Colistin. Pour celle-ci, nous avons observé par le même biais que la matrice X_genexp sera sûrement indispensable.

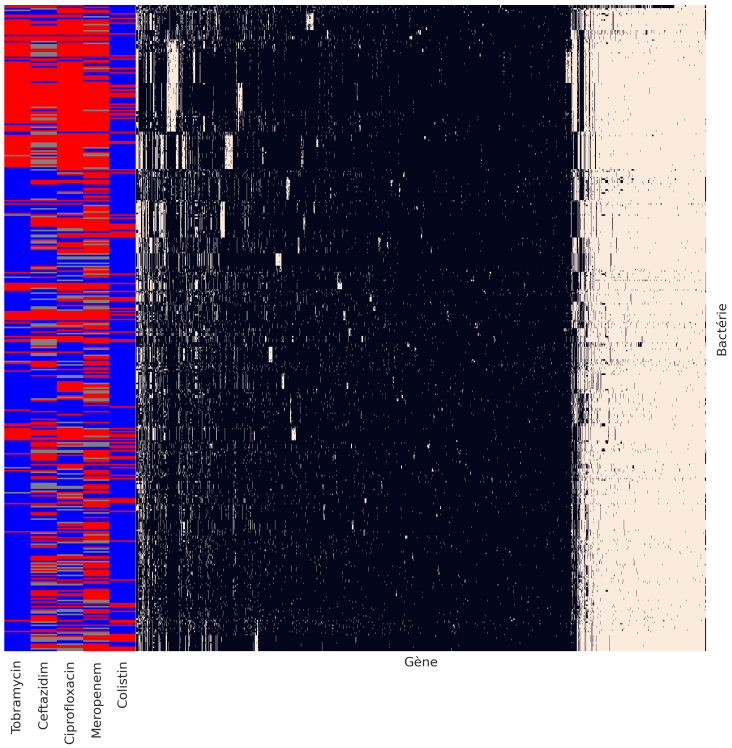


Figure 1: Carte de chaleur du clustering hiérarchique sur les lignes et les colonnes de la matrice X_gpa. Les couleurs à gauche des lignes permettent de déterminer si la bactérie est sensible (bleu) ou résistante (rouge) à l'antibiotique, le gris correspond aux données manquantes.

3 Approche pour la prédiction

3.1 Pré-traitements

Comme expliqué dans la section §2, nous avons fait le choix de supprimer les données manquantes. Cette suppression est faite de manière "intelligente" en ce sens où les bactéries dont la résistance est

absente sont éliminées uniquement pour les antibiotiques concernés et demeurent disponible pour les autres antibiotiques.

Nous nous sommes ensuite contentés de centrer/réduire les expressions génétiques de la matrice $X\_$ *genexp* grâce au transformateur **StandardScaler** de *scikit-learn*.

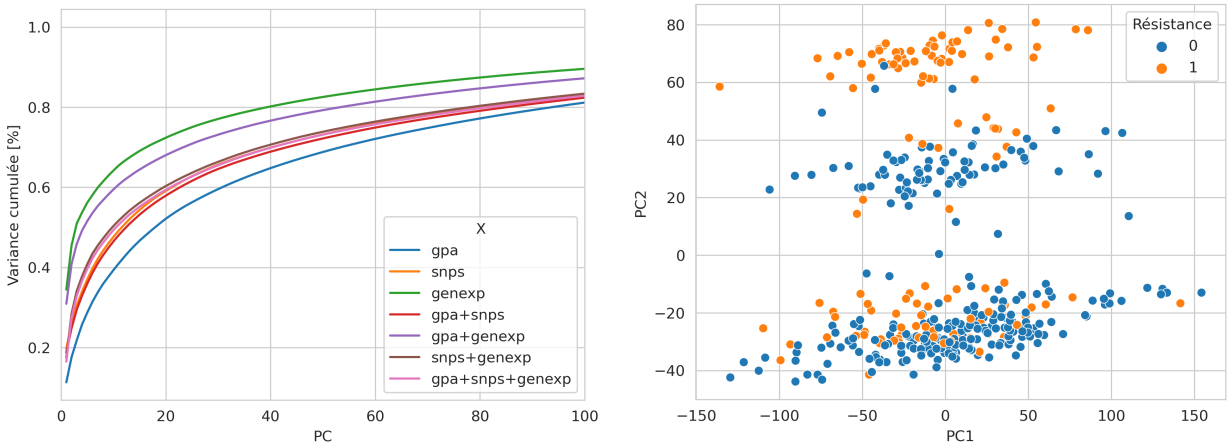
S'est ensuite posée la question de la mutualisation des informations contenues dans les 3 matrices de classifieurs. En traçant les cartes de chaleur de ces matrices, nous avons aperçu que toutes ne sont pas nécessairement pertinentes pour tous les antibiotiques. C'est pourquoi nous avons fait le choix de ne pas systématiquement agréger les matrices, mais de considérer les 7 arrangements possibles : 1 seule matrice (3), 2 matrices (3) et les 3 matrices (1).

3.2 Réduction de la dimensionnalité

Nous avons vu que nos matrices de régresseurs contiennent beaucoup de covariables, jusqu'à 94267 lorsque nous les concaténons entres-elles, relativement aux nombres d'observations dont nous disposons. Il est donc impératif d'envisager de réduire cette dimensionnalité. Cela peut se faire en amont de la tâche de classification, ou alors de manière intégrée en incluant une pénalisation sur les poids du modèle associés aux regresseurs. Nous avons considéré les deux approches et nous détaillerons dans cette partie les 3 méthodes que nous avons employées en amont.

3.2.1 ACP (à Noyau)

L'Analyse en Composantes Principales (ACP) est une approche bien connue permettant de représenter les observations dans un sous-espace vectoriel dont les composantes sont décorrélées. Les composantes sont ordonnées en ordre décroissant de variance expliquée, de sorte qu'il est simple d'utiliser le nombre de composantes conservées pour représenter les données comme un hyper-paramètre permettant de plus ou moins réduire la dimensionnalité tout en maximisant le pourcentage de variance expliquée pour un nombre de composantes fixé.



(a) Variance cumulée exprimée en pourcentage. (b) Nuage de points selon les 2 premières dimensions.

Figure 2: Représentations de l'ACP avec un noyau linéaire.

Nous avons appliqué l'ACP à notre jeu de données, en considérant tous les arrangements possibles d'utilisation des matrices de descripteurs et comme nous pouvons le voir sur la figure 2, les résultats sont mitigés. D'un côté, la figure 2a ne montre pas de point d'inflexion franc dans la courbe de variance cumulée, ce qui suggère qu'il n'existe pas de point au delà duquel les composantes

sont peu importantes. En revanche, nous pouvons voir sur la figure 2b que la projection dans les deux premières dimensions de l'ACP (en ayant agrégé l'ensemble des matrices de régresseurs) de nos données permet de partiellement séparer les bactéries sensibles et résistantes, ce qui est encourageant en vue de l'étape de classification.

Par ailleurs, grâce au *kernel-trick* il est possible de mettre en place une variante de l'ACP permettant une réduction de dimensions non linéaire en utilisant divers noyaux. Via *scikit-learn*, il suffit alors d'utiliser la classe **KernelPCA**.

3.2.2 Stability selection

La *stability selection* est une approche permettant de corriger l'instabilité de la sélection de variables de l'estimateur Lasso en présence de covariables corrélées. Pour ce faire, des estimateurs pénalisés avec la norme $L1$ sont ajustés sur de multiples ré-échantillonnages du jeu de données. Chaque estimateur produit un chemin de régularisation le long duquel les poids associés aux covariables les moins importantes pour la tâche de régression ou de classification sont progressivement annulés quand la pénalisation augmente. À partir de ces chemins de régularisation, l'estimateur de *stability selection* construit un chemin de stabilité représentant la fréquence de sélection des covariables parmi l'ensemble des estimateurs Lasso. Au moyen d'un seuil de stabilité, il est donc possible de sélectionner les variables alors considérées comme *stables*. Cette procédure est généralement utilisée comme un estimateur plutôt que comme une méthode de réduction de dimensions, mais nous avons souhaité la tester ainsi (étant donné que nous avons intégré le Lasso aux classifieurs considérés par la suite, l'approche originelle est cependant également employée).

La *stability selection* n'est pas proposée par *scikit-learn* (uniquement le *bootsrap Lasso*), nous avons donc implémenté notre propre estimateur héritant des classes de bases **TransformerMixin** et **BaseEstimator** afin de pouvoir l'utiliser ensuite dans un **Pipeline**. La figure 3 illustre le chemin de régularité obtenu en utilisant l'ensemble des matrices de régresseurs pour construire la matrice de covariables et avec un seuil de régularité de 0.6. Un tel seuil conduit à conserver environ 1250 covariables sur plus de 96000 au départ.

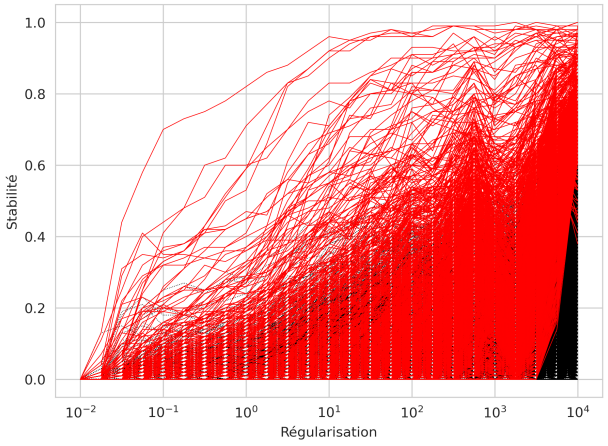


Figure 3: Illustration du chemin de stabilité obtenu par *stability selection*.

L'un des inconvénients principal de la *stability selection* est le temps d'ajustement très important en raison du nombre de ré-échantillonnages à effectuer (100 dans notre cas) et de la longueur du chemin de régularisation (fixé à 25).

3.2.3 Tests multiples

Une autre manière de réaliser la réduction de dimensionnalité pourrait être de considérer que de nombreuses variables sont inutiles pour la prédiction, car leurs distributions entre les bactéries susceptibles ou résistantes est la même.

Nous pouvons ainsi procéder à une nouvelle sélection de variables s'appuyant cette fois-ci sur deux tests statistiques : le Z-test pour les matrices *X_ gpa* et *X_ snps* et le T-test pour la matrice *X_ genexp*. Nous avons utilisé la librairie *statsmodels* qui propose l'implémentation de ces tests. En raison du grand nombre de tests réalisés de manière indépendante, il convient d'ajuster les *pvaleurs*. Nous avons choisi d'utiliser la procédure de Benjamini-Hochberg permettant de contrôler le FDR (*False Discovery Rate*), plutôt que des procédures contrôlant le FWER (*Family-Wise Error Rate*) afin de découvrir plus de variables statistiquement différentes selon les deux modalités de résistance. Nous nous sommes tourné pour cela vers la librairie *MultiPy* qui permet de choisir parmi de nombreuses méthodes d'ajustement. L'ensemble de la procédure a été encapsulée dans une classe permettant de l'utiliser comme un transformateur dans un **Pipeline**.

Comme le montre la table 2, cette approche permet de réduire très fortement le nombre de covariables utilisées par le classifieur, et le niveau α peut-être employé comme un hyper-paramètre permettant de jouer sur le nombre de variables rejetées.

Table 2: Nombre de covariables sélectionnées pour différents niveaux α.

	# régresseurs	Niveau α		
	total	0.1	0.05	0.01
gpa	16005	575	374	157
snps	72236	5258	3053	1233
genexp	6026	96	77	56

Le désavantage que nous voyons à cette approche par tests multiples est que les variables sélectionnées sont potentiellement corrélées.

3.3 Classifieurs considérés

Nous avons considéré 3 grands types de classifieurs différents : régression logistique, machines à support vectoriel (SVM) et forêts aléatoires. Nous avons également utilisé les variantes AdaBoost et Gradient Boosting des forêts aléatoires, ainsi que l'entrainement par descente de gradient pour la régression logistique et la SVM linéaire.

Dans l'univers *scikit-learn* cela revient à utiliser les classes **LogisticRegression**, **SVC**, **RandomForestClassifier**, **AdaBoostClassifier**, **GradientBoostingClassifier** et **SGDClassifier**. Nous verrons dans la section suivante quels hyper-paramètres nous avons fait varier pour ces différents estimateurs, mais pour l'ensemble d'entre eux nous avons spécifié *class_ weight="balanced"* afin de tenir compte du déséquilibre entre les nombres de bactéries sensibles et résistantes.

3.4 Mise en œuvre

Comme évoqué précédemment, nous nous sommes reposés sur la librairie *scikit-learn* pour la mise en œuvre de notre étude de prédiction.

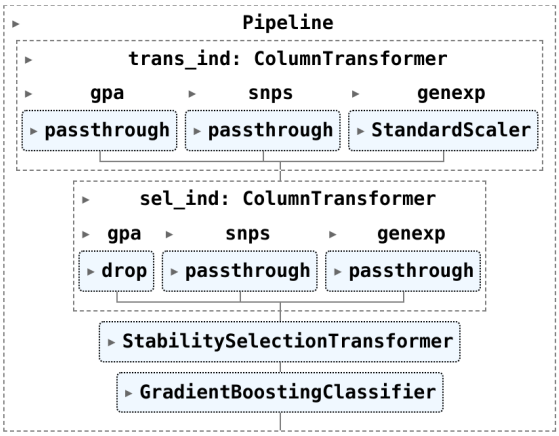


Figure 4: Représentation du **Pipeline** construit.

La figure 4 illustre le **Pipeline** que nous avons utilisé. Le premier niveau, *trans_ind*, applique les pré-traitements de manière distincte pour les différentes matrices de régresseurs. Le niveau *sel_ind* permet de générer les différents arrangements possibles pour l'utilisation des matrices de régresseurs. Le troisième niveau correspond à celui de la réduction de dimension. Et enfin, le dernier estimateur est le classifieur.

Ce **Pipeline** nous permet de "facilement" faire varier les différents estimateurs et leurs hyper-paramètres en utilisant la classe **GridSearchCV** pour trouver la combinaison la plus performante. "Facilement" car construire une grille d'hyper-paramètres avec notre structure n'est pas si évident que cela, et car le nombre de combinaisons d'hyper-paramètres augmente très vite. Nous avons indiqué à l'object **GridSearchCV** que la métrique à utiliser pour mesurer la performance est la *balanced accuracy* afin de tenir compte du déséquilibre dans les classes à prédire.

Table 3: Hyper-paramètres envisagés dans la grille de recherche.

	hyper-paramètres
PCA	kernel: [linear,poly,rbf,sigmoid], n_components: [64,128,256]
StabilitySelection	threshold: np.linspace(0.6,0.9,4)
MultipleTesting	alpha: [0.01,0.05,.1]
LogisticRegression	C: np.logspace(-1,1,3)
SVC	C: np.logspace(-1,1,3), kernel: [linear,poly,rbf,sigmoid]
RandomForestClassier	n_estimators: [100,300,500], max_depth: [None,10,100], max_features: [sqrt,log2]
AdaBoostClassifier	learning_rate: np.logspace(-2,0,3)
GradientBoostingClassifier	learning_rate: np.logspace(-2,0,3)
SGDClassifier	loss: [hinge,log_loss], alpha: np.logspace(-5,-3,3)

La table 3 donne à voir l'ensemble des hyper-paramètres considérés dans la grille de recherche. Au total, cela représente 7200 combinaisons d'estimateurs et d'hyper-paramètres par antibiotique. Afin de limiter le temps de recherche des hyper-paramètres, en plus du parallélisme, nous avons utilisé la librairie *scikit-learn-intelex* qui propose des implémentations plus rapides de certains estimateurs pour les processeurs Intel ; et nous nous sommes également servi de l'argument *memory* de **GridSearchCV** qui permet de ne pas ré-ajuster les estimateurs lorsque leurs paramètres d'entrée ne changent pas. Malgré cela, la recherche des hyper-paramètres a nécessité plusieurs heures.

4 Résultats

Dans un premier temps nous nous sommes intéressés à la moyenne et à la déviation standard des scores obtenus par validation croisée pendant la procédure de recherche des hyper-paramètres. Nous avons fait figurer sur la table 4 les meilleurs résultats selon ces critères pour chacun des antibiotiques. La table permet également de voir quelles matrices de régresseurs ont alors été utilisées, ainsi que les estimateurs de réduction de dimension et de classification retenus.

Table 4: Meilleurs scores obtenus par validation croisée et architectures associées.

	Balanced accuracy		Présence			Estimateur	
	moyenne	déviation	gpa	snps	genexp	réduction	classifieur
Ceftazidim	0.84	0.05	True	False	True	StabilitySelection	AdaBoostClassifier
Ciprofloxacin	0.89	0.03	True	True	False	StabilitySelection	AdaBoostClassifier
Colistin	0.73	0.06	True	False	True	StabilitySelection	SVC
Meropenem	0.90	0.03	True	True	True	MultipleTesting	LogisticRegression
Tobramycin	0.93	0.04	True	False	True	StabilitySelection	RandomForestClassifier

En terme de scores, nous pouvons immédiatement noter que prédire la résistance à la Colistin semble bien plus compliqué que pour les autres antibiotiques, comme pouvait nous le laisser présager la figure 1. La Ceftazidim vient ensuite, puis la Ciprofloxacin et la Meropenem dont les performances de prédiction sans assez proches, tandis que la Tobramycin apparait comme l'antibiotique auquel la résistance est la plus prévisible. En parallèle des scores moyens, nous pouvons être relativement satisfait de la déviation standard de la *balanced accuracy* qui est inférieure ou égale à 0.05, exception faite de la Colistin.

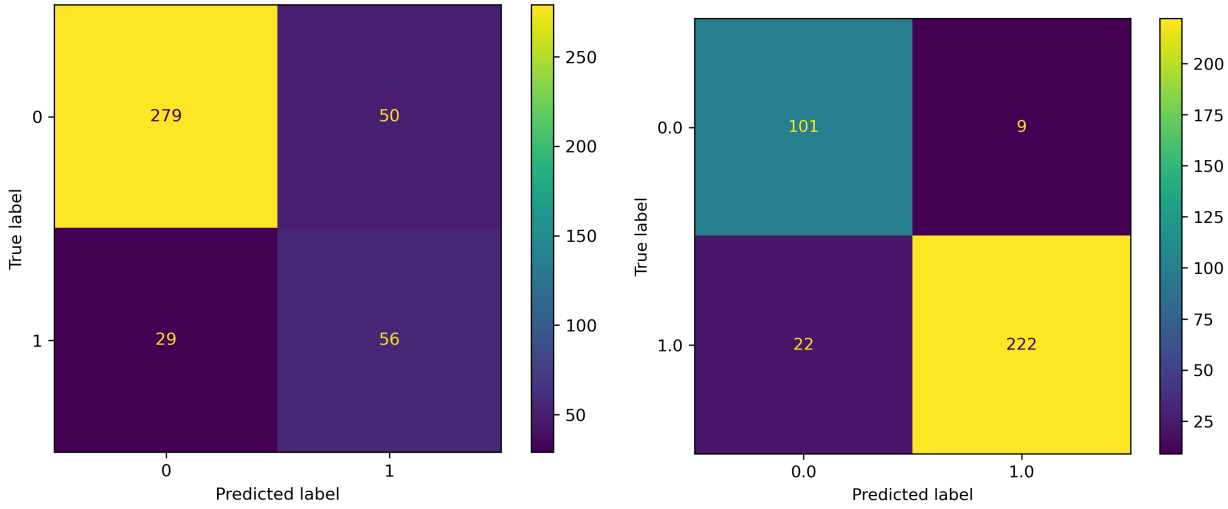
Le choix de considérer la *stability selection* et les tests multiples pour réduire la dimensionnalité semble aussi avoir été payant, étant donné qu'ils se retrouvent dans les estimateurs de réduction de dimensions des meilleures architectures, au contraire de l'ACP.

À la vue des indications de présence ou d'absence des matrices de régresseurs, X_gpa semble importante dans la prédiction de l'ensemble des résistances, X_genexp a été ignorée uniquement dans le cas de la Ciprofloxacin et X_snps serait celle la moins souvent utile car conservée deux fois seulement. Pour approfondir ces observations nous avons observé le pourcentages de sélection des matrices de régresseurs parmi les 10 meilleures architectures pour chaque antibiotique. Nous pouvons voir sur la table 5 qu'il est confirmé que X_snps est bien nécessaire pour prédire la résistance à la Ciprofloxacin mais très peu utile pour les autres antibiotiques. En revanche, l'importance prépondérante de X_gpa pour l'ensemble des antibiotiques est nuancée, notamment en comparaison avec celle de X_genexp qui est sélectionnée 100% du temps 3 fois, contre 2 fois pour X_gpa. Ainsi X_genexp semble absoluement indispensable pour la prédiction de la Tobramycin, la Colistin et la Ceftazidim ; tandis que X_gpa le serait également pour la Tobramycin, et pour la Meropenem.

Table 5: Pourcentage de sélection des matrices de régresseurs parmi les 10 meilleures architectures.

	gpa	snps	genexp
Ceftazidim	80%	20%	100%
Ciprofloxacin	70%	100%	60%
Colistin	60%	20%	100%
Meropenem	100%	50%	80%
Tobramycin	100%	10%	100%

Pour tenter de comprendre un peu plus les forces et les faiblesses de nos modèles de prédictions, nous avons généré les matrices de confusion entre les classes attendues et celles prédites. Dans la plupart des cas il n'est pas apparu de déséquilibre dans les fausses prédictions entre classe résistante et sensible, mis à part pour la Colistin et la Méropenem dont nous avons donc représenté les matrices sur la figure 5. Il semble sur 5a que notre modèle a eu plus de difficultés à correctement classifier la susceptibilité à la Colistin que la résistance, ce qui peut-être préférable selon l'objectif visé. A l'inverse, 5b montre que pour la Meropenem c'est la détection des bactéries résistantes qui a causé le plus de problème, mais dans des proportions moindres qu'avec la Colistin. Dans les deux cas, cela peut s'expliquer par le fort déséquilibre entre bactéries sensibles et résistantes pour ces antibiotiques, mais ce déséquilibre était également présent pour la Tobramycin dont la matrice de confusion est équilibrée.



(a) Pour la Colistin.

(b) Pour la Meropenem.

Figure 5: Matrices de confusion.

5 Conclusion

Afin de prédire au mieux la résistance d'environ 400 bactéries à 5 antibiotiques différents à partir de 3 matrices de régresseurs, nous avons développé un **Pipeline** doté de plusieurs couches permettant d'appliquer des pré-traitements distincts, de sélectionner ou non les matrices de régresseurs, de réduire la dimensionnalité des covariables utilisées par différentes approches et enfin d'appliquer plusieurs classifieurs. Les meilleures combinaisons et leurs hyper-paramètres ont ensuite été ajustés par validation croisée afin de généraliser au mieux les résultats finaux, et en considérant comme critère la *balanced accuracy* pour tenir compte du déséquilibre des classes à prédire.

Les scores obtenus par validation croisée ont montré que la Colistin est l'antibiotique pour lequel la prédiction à la résistance s'est avéré la plus compliquée, avec une *balanced accuracy* moyenne de seulement 0.73 et une déviation de 0.06. Nous considérons en revanche que la prédiction pour les autres antibiotiques est plutôt bonne, voir très bonne en ce qui concerne la Tobramycin.

Il est apparu que la prédiction de la résistance pour tous les antibiotiques ne nécessitaient pas les mêmes types d'informations sur les bactéries. Ainsi, la présence ou l'absence de gènes s'est avérée capitale pour la Meropenem et la Tobramycin, tandis que ce sont les mutations génétiques qui ont été indispensables pour la Ciprofloxacin. Les informations sur l'état cellulaire "courant"

des bactéries a-t-elle été absoluement nécessaire pour la Ceftazidim, et surtout la Colistin, et s'est trouvée être un excellent complément pour la Tobramycin.

Pour finir, nous avons identifié quelques pistes intéressantes et des questionnements sur les résultats obtenus.

Tout d'abord, il aurait peut-être été bon de se pencher sur l'impact sur la dimensionnalité des différentes méthodes de réduction de dimension et sur le lien éventuel avec les performances en classification.

Nous aurions aussi aimé inclure un estimateur réalisant un vote entre les prédictions de plusieurs classifieurs mais nous n'avons pas eu le temps de répondre à la question du choix des classifieurs utilisés pour le vote : les n meilleurs, les meilleurs sur chacun des sous-ensembles de validation, ... Nous étions également contraints par les temps d'ajustement des modèles.

De plus, afin de réduire ces temps d'ajustement, nous aurions peut-être pu utiliser la classe **HalvingGridSearchCV** permettant d'ajuster les hyper-paramètres en allouant peu de ressources au début de la recherche et en éliminant les combinaisons les plus mauvaises au fur et à mesure que plus de ressources sont allouées. Notre souci avec cette technique a été le fort déséquilibre dans les classes à prédire pour les premières itération de la recherche.

Par ailleurs, nous n'avons posé la question du temps d'ajustement et du gain de performance engendré par l'utilisation de la *stability selection* par rapport à celle des tests multiples.

Enfin, nous avons décidé de ne pas utiliser de réseaux de neurones en raison du fort risque de sûr-apprentissage dû au faible nombre d'observations à notre disposition et du grand nombre d'hyper-paramètres à considérer avec ce types de modèles. De plus, le coût calculatoire s'en serait encore trouvé augmenté.