Rapport du Package mcpredomics

Fabien KAMBU MBUANGI

Résumé

Dans ce rapport, nous présentons le package mcpredomics(multiclasse predomics) conçu pour étendre les fonctionnalités du package predomics. Predomics est un package qui recherche des modèles prédictifs simples et interprétables à partir de données omiques et plus spécifiquement métagénomiques. Cependant, l'approche à l'état actuel de predomics ne peut pas classer les données en plusieurs classes.

Ainsi, le package mcpredomics utilise l'approche one vs one (OVO) qui permet d'étendre les concepts de l'approche predomics en permettant de faire de la classification multiclasses.

Introduction

Le microbiote intestinal est l'ensemble des microorganismes du tractus digestif humain, c'est-à-dire tout le système gastro-intestinal. Le microbiome intestinal est composé de toutes les bactéries, commensales et pathogènes, résidant dans le tractus gastro-intestinal. Il joue un rôle essentiel dans notre santé, un microbiote équilibré contribue à rester en bonne santé.

Le microbiome intestinal est impliqué dans un nombre croissant de maladies humaines telles que maladie de Crohn, Obésité, Diabète, Allergies, Cancer colorectal, etc. L'étude du microbiote se fait par une méthode de séquençage appelée métagénomique. La métagénomique consiste à séquencer le matériel génétique de tous les organismes présents dans l'échantillon. Les données métagénomiques sont des informations génétiques provenant d'un échantillon environnemental, souvent utilisées pour étudier les microbiomes et leurs relations avec les hôtes, ces données sont sous forme de tables d'abondance et sont utilisées par des approches d'apprentissage statistique (IA) afin d'apprendre des modèles permettant de classer les échantillons/individus, mais aussi de prédire des évènements futurs à partir du microbiome.

Les nouvelles approches de séquençage 16S rRNA et de séquençage de l'ADN métagénomique ont permis de générer d'énormes quantités de données microbiomiques, ouvrant la voie à de nouvelles opportunités pour la classification multiclasses.

Dans ce rapport, nous présentons le package mcpredomics développé pour étendre les fonctionnalités de predomics afin de pouvoir faire de la classification en multiclasses.

Données utilisées

Pour tester notre package, nous avons utilisé un échantillon de taille de 894 constitué de 4 entérotypes (“Rum : 183”, “Bact1: 299”, “Bact2 : 168”, “Prev : 244”) extrait de la base de données métacardis.

```
[1] 894
```

```
Data_set
Bact1 Bact2  Prev   Rum
  299   168   244   183
```

Méthode utilisée

One-vs-all (ova) et One-v-one (ovo) sont les deux principales approches d'apprentissage automatique pour résoudre un problème de classification multiclasse. Dans ces deux approches, le choix est transparent et le résultat renvoyé à l'utilisateur sera toujours les valeurs ou classes finales. Mais il est très important de comprendre ces deux approches pour optimiser un modèle d'apprentissage automatique et toujours choisir la meilleure approche lors de la résolution d'un problème de classification multiclasse.

One-vs-all est probablement la stratégie la plus courante. Dans cette approche, le nombre n de modèles de classification est entraîné en parallèle avec le nombre n de classes de sortie en considérant qu'il existe toujours une séparation entre la classe réelle et les classes restantes.

One-vs-one est une approche alternative au One-vs-all. Cela signifie former un modèle d'apprentissage automatique pour chaque paire de classes. La complexité temporelle de cette approche n'est donc pas linéaire et la bonne classe est déterminée par la classe majoritaire. En général, le modèle One-v-one est plus coûteux que le modèle One-vs-all et ne doit être adopté que lorsqu'une comparaison de l'ensemble complet des données n'est pas préférable. Dans ce package, nous avons utilisé l'approche One-vs-one pour rechercher les variables entre chacune des paires des classes de notre échantillon.

Par exemple, considérons notre jeu de données multiclasse avec quatre classes : « Bact1 », «Bact2» et «Rum », «Prev ». Cela pourrait être divisé en six ensembles de données de classification binaire comme suit :

- **Problème de classification binaire 1** : Bact1 contre Bact2
- **Problème de classification binaire 2** : Bact1 contre Rum
- **Problème de classification binaire 3** : Bact1 contre Prev

- **Problème de classification binaire 4** : Bact2 contre Rum
- **Problème de classification binaire 5** : Bact2 contre Prev
- **Problème de classification binaire 6** : Rum contre Prev

La formule pour calculer le nombre d'ensembles de données binaires, et donc de modèles, est la suivante :

- (NumClasses * (NumClasses – 1)) / 2

Chaque modèle de classification binaire peut prédire une étiquette de classe et la moyenne de ce modèle est prédit par la stratégie un contre un.

Algorithme Terga1_OVO

Dans predomics plusieurs algorithme ont été implémenté tels que: 'terga1', 'terga2', 'terbeam', 'terda' et 'metal', Notons que chaque algorithme possède ses propres paramètres. Dans notre package, nous avons développé un nouvel algorithme terga1_ovo qui est dérivé de l'algorithme Terga1 de predomics afin de pouvoir faire de la classification en multiclasse. Terga1 est un algorithme qui est basé sur des algorithmes génétiques. C'est-à-dire, il introduit la notion de population de modèles, qui est un ensemble d'individus/modèles qui peuvent être mutés, croisés, évolués et sélectionnés sur de nombreuses générations (époques). Pour lancer notre expérience, nous avons utilisé les paramètres par défaut et définit uniquement « nCores = 1 ». Si nCores > 1, l'exécution se déroulera en parallèle, nous avons défini « seed = 1 » (si plusieurs graines sont fournies, l'algorithme s'exécutera plusieurs fois) enfin, nous avons fixé plot à TRUE. Lorsque 'plot = TRUE', les graphiques avec le processus d'évolution sont fournis au format pdf.

```
[1] "Summary of Classifier object"
========== Experiment ==========
    --- id: terga1_ovo
    --- description: terga1_ovo Wed Oct 11 18:28:09 2023
    --- save: nothing
========== Learner ==========
    --- learner: terga1_ovo
    --- sparsity: 1,2,3,4,5,6,7,8,9,10
    --- current_sparsity: NA
    --- size_pop: 100
    --- size_world: NULL
    --- max.nb.features: 1000
    --- nb_generations: 100
    --- unique_vars: FALSE
```

```
    --- popSourceFile: NULL
    --- popSaveFile: NULL
    --- objective: auc
    --- estimate_coefs: FALSE
    --- select_type: mixed
    --- select_perc1: 20
    --- select_perc2: 30
    --- perc_best_ancestor: 10
    --- mutate_size: 70
    --- mutate_rate: 50
    --- convergence: TRUE
    --- convergence_steps: 10
    --- evolve_k1: TRUE
    --- plot: TRUE
    --- verbose: TRUE
    --- warnings: FALSE
    --- debug: FALSE
    --- print_ind_method: short
    --- nCores: 1
    --- parallel: FALSE
    --- parallelize.folds: TRUE
    --- parallel.local: FALSE
    --- seed: 1
    --- evalToFit: fit_
    --- k_penalty: 0
    --- language: terinter
    --- epsilon: NULL
    --- scoreFormula: function
    --- intercept: NULL

[1] TRUE

[1] "list"
```

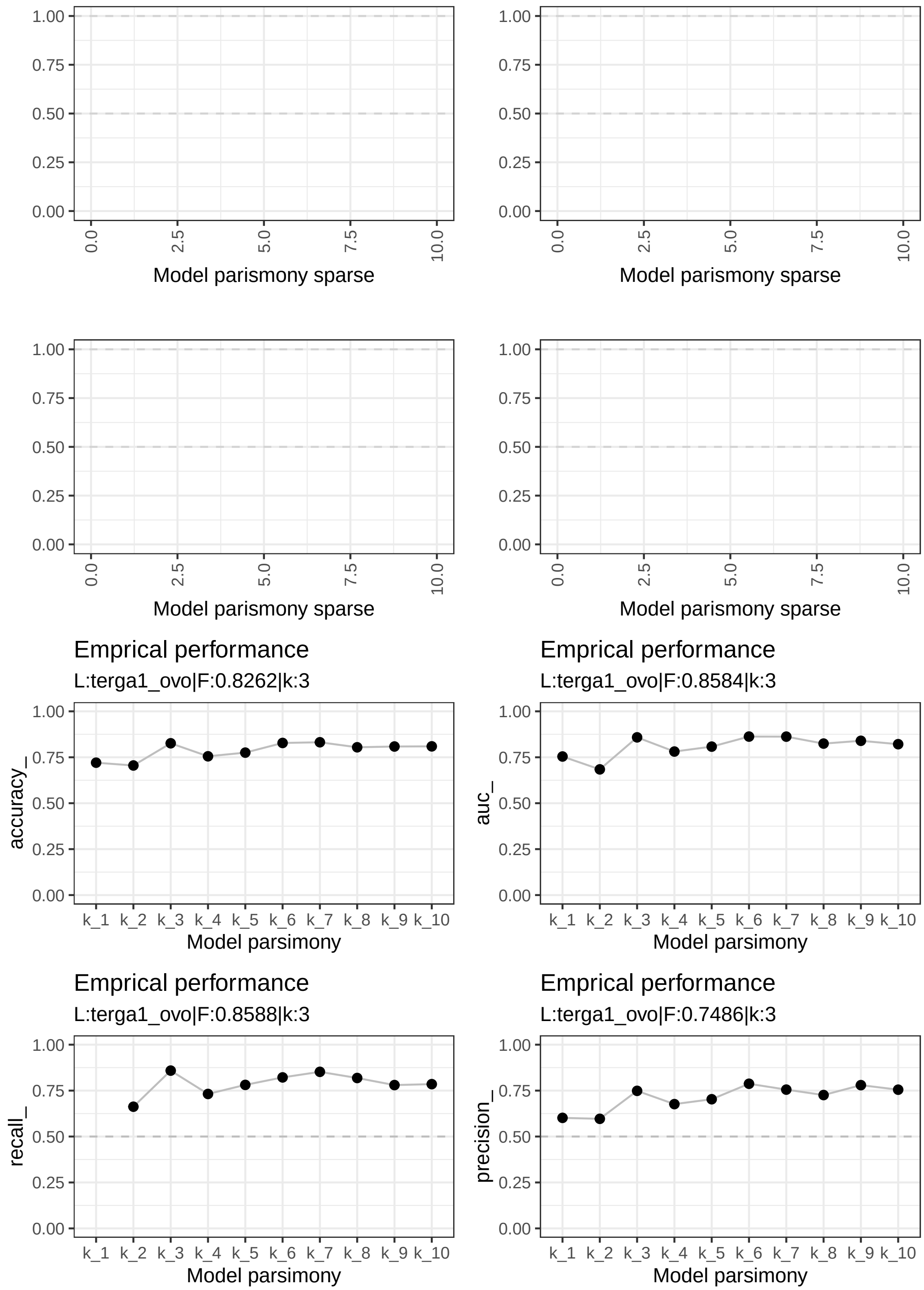
Expérience de l'apprenant

```
Après avoir configuré le classificateur, nous pouvons exécuter l'expérience d'apprentissage.
```

Le temps d'exécution de notre expérience était de 30.20973 minutes. Il y a eu 862 modèles dans cette population. Après filtrage de la population, 11 modèles ont été retenus. La prévalence des fonctionnalités était calculée et, il y a au final 37 fonctionnalités à considérer.

Peformance du modèle

Les figures ci-dessous présentent nos premiers résultats après digestion de l'expérience de l'apprenant. La performance de notre modèle sur ces différentes métriques (accuracy: 0.82, auc: 0.85, recall: 0.85, precision: 0.74.

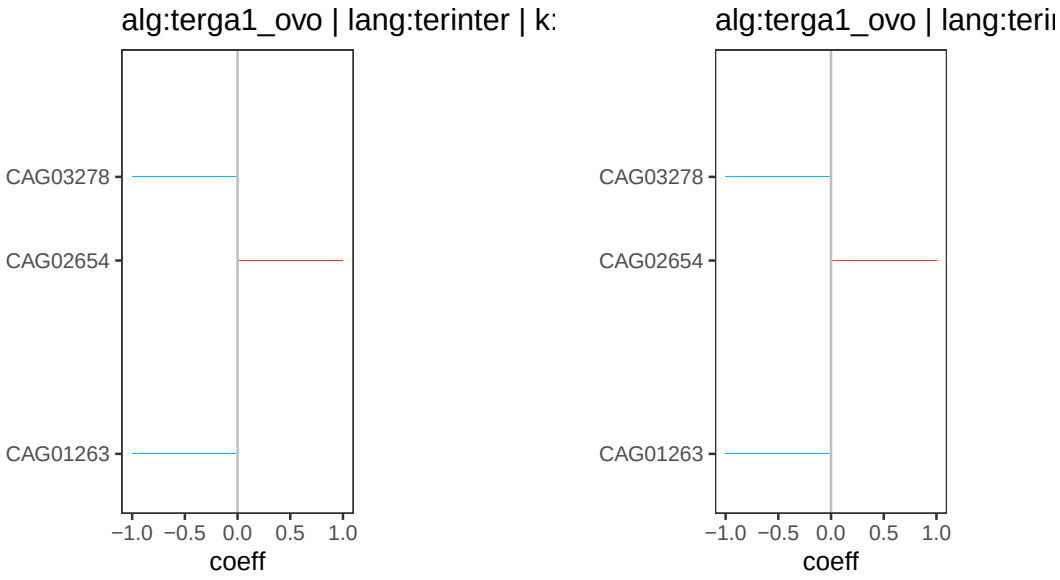

Model parismony sparse
Model parismony sparse
Model parismony sparse
Model parismony sparse
Emprical performance
L:terga1_ovo|F:0.8262|k:3
accuracy_
Model parsimony
Emprical performance
L:terga1_ovo|F:0.8584|k:3
auc_
Model parsimony
Emprical performance
L:terga1_ovo|F:0.8588|k:3
recall_
Model parsimony
Emprical performance
L:terga1_ovo|F:0.7486|k:3
precision_
Model parsimony

Visualiser le meilleur modèle

Le meilleur modèle appris a été imprimée avec des informations résumées sur les performances et la taille du modèle. De plus, nous pouvons explorer davantage le modèle en le visualisant à l'aide du tracé de code-barres '?plotModel'. Enfin, l'importance de chaque fonctionnalité peut également être affichée en utilisant la même fonction, comme illustré dans la figure ci-dessous.

```
[1] "Summary of Model object"
```

```
[1] "|(+ CAG02654)  -  (+ CAG01263+ CAG03278) > -2.1e-08 then class = Prev| (F=0.8262|K=3|L=
```



Tester le modèle avec un jeu de données test

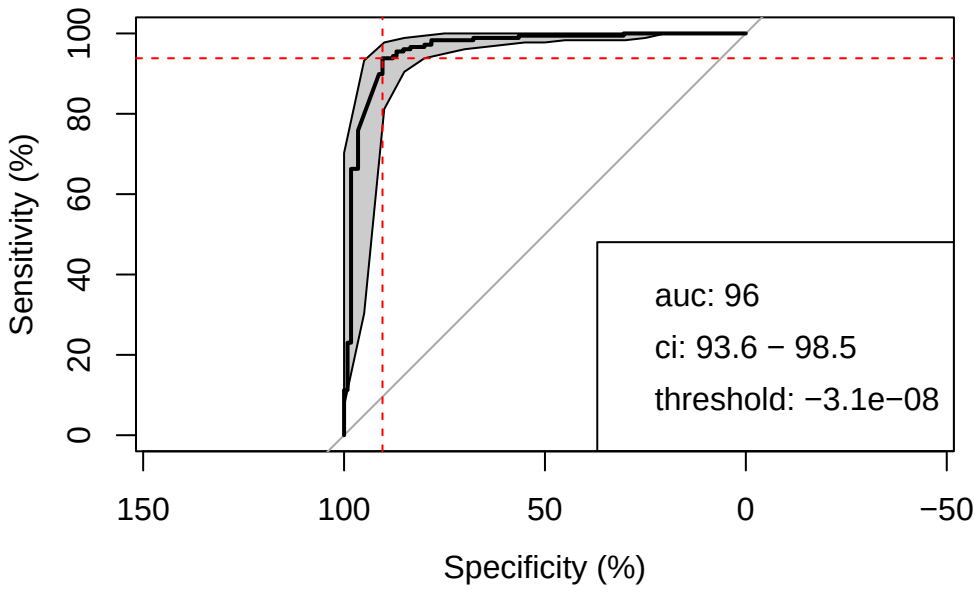
Maintenant, testons ce modèle avec le 30% restant de données réservé pour le test. Pour cela, nous devons travailler dans le même espace de fonctionnalités que celui avec lequel nous avons travaillé jusqu'à présent. L'accuracy du modèle dans l'ensemble de données d'entraînement est de 0,82, tandis que dans l'ensemble de données de test, elle est de 0,65.

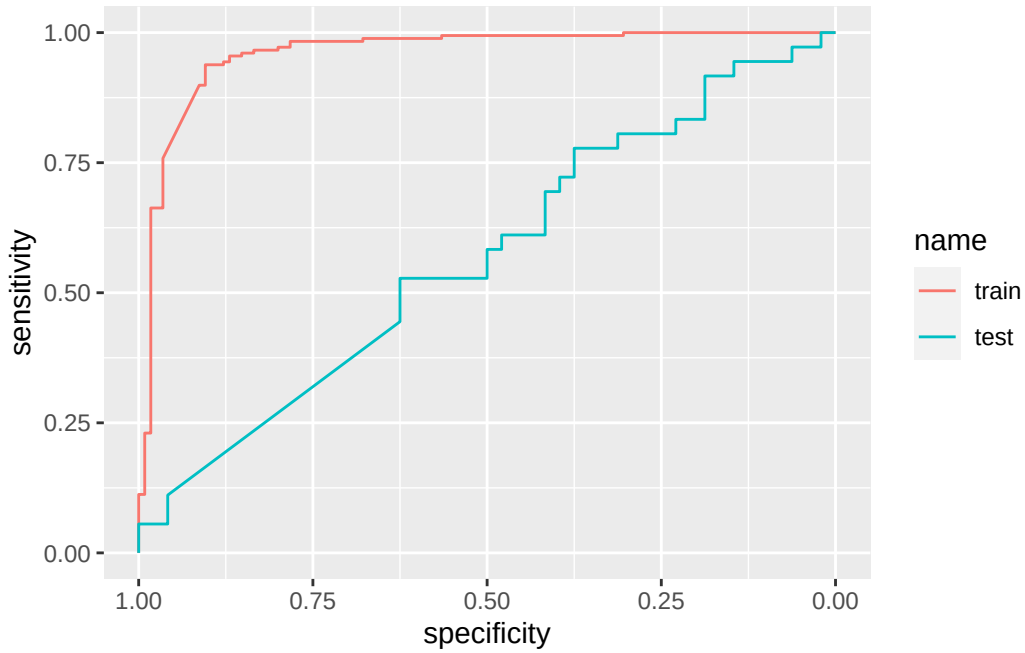
```
[1] "Summary of Model object"
```

```
[1] "|(+ CAG02654)  -  (+ CAG01263+ CAG03278) > -2.1e-08 then class = Rum| (F=0.6574|K=3|L=t
```

Visualisation des performances du modèle AUC

Les performances du modèle peuvent être visualisées à l'aide de la fonction '?plotAUC' qui utilise le jeu d'outils du package pROC. Il fournit également l'intervalle de confiance de l'aire sous la courbe pour l'analyse des caractéristiques de fonctionnement du récepteur. Nous pouvons également comparer plusieurs modèles en un seul en utilisant '?ggroc' du package *pROC*. Pour cela, nous devons créer des objets roc pour chaque modèle et les lister ensemble avant le tracé. Il convient de noter que les performances du modèle dans l'ensemble de données de test correspondent à l'intervalle de confiance du premier graphique ci-dessous.





Affichage de la famille de meilleures modèles

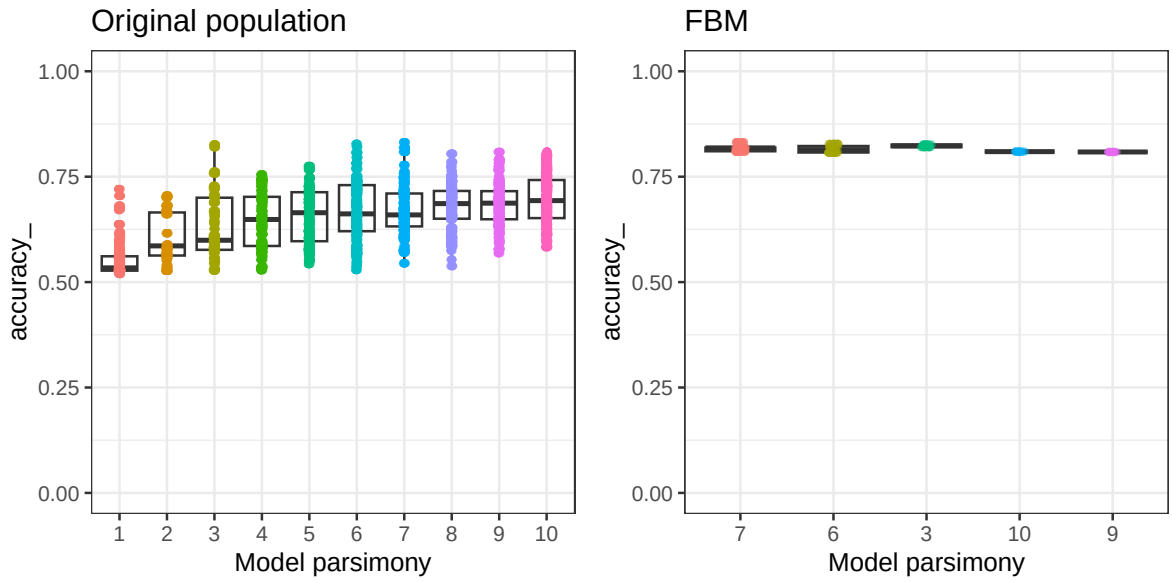
Une famille des meilleurs modèles est définie comme l'ensemble des modèles renvoyés par l'algorithme prédomique, dont la précision se situe dans une fenêtre donnée de précision du meilleur modèle. Cette fenêtre est définie en calculant un seuil de signification en supposant que la précision suit une distribution binomiale (p<0,05). Un FBM peut être analysé en détail pour distiller des informations biologiques dans un contexte prédictif. Tout d'abord, nous regroupons tous les modèles de l'objet d'expérience avec '?modelCollectionToPopulation'. Une population de 862 modèles est obtenue, qui peut être transformée avec « ?populationToDataFrame » sur une trame de données pour une exploration plus approfondie. Ensuite, nous sélectionnons le FBM composé de 13 modèles avec '?selectBestPopulation'. La figure ci-dessous affiche la distribution de l'accuracy par taille de modèle avant et après la sélection.

```
[1] "Summary of a population of models with 862 models"
[1] "1:|0  -  (+ CAG01263) > -9e-08 then class = Prev| (F=0.7206|K=1|L=terga1_ovo|La=terinte
[1] "2:|(+ CAG01321)  -  0 > 2.2e-06 then class = Prev| (F=0.7048|K=1|L=terga1_ovo|La=terinte
[1] "3:|0  -  (+ CAG01780) < -3.6e-07 then class = Prev| (F=0.6812|K=1|L=terga1_ovo|La=terint
[1] "4:|(+ CAG01015)  -  0 > 7.2e-08 then class = Prev| (F=0.6768|K=1|L=terga1_ovo|La=terinte
[1] "5:|0  -  (+ CAG03038) > -4.7e-08 then class = Prev| (F=0.6731|K=1|L=terga1_ovo|La=terint
[1] "..."
```

```
      learner language      auc_ accuracy_    intercept_ eval.sparsity sign_
```

```
mod_1 tergа1_ovo terinter 0.7541693 0.7205582 -9.017635e-08             1     >
mod_2 tergа1_ovo terinter 0.6864117 0.7047686  2.159633e-06             1     >
mod_3 tergа1_ovo terinter 0.6142060 0.6811928 -3.637867e-07             1     <
mod_4 tergа1_ovo terinter 0.6913670 0.6767718  7.239215e-08             1     >
mod_5 tergа1_ovo terinter 0.7071829 0.6731298 -4.670323e-08             1     >
mod_6 tergа1_ovo terinter 0.6359189 0.6710804 -1.602952e-08             1     <
      precision_   recall_
mod_1  0.6016431       NaN
mod_2  0.5942938 0.6587657
mod_3  0.6832614 0.6569093
mod_4  0.3750034       NaN
mod_5  0.4255773       NaN
mod_6  0.5485990       NaN
```

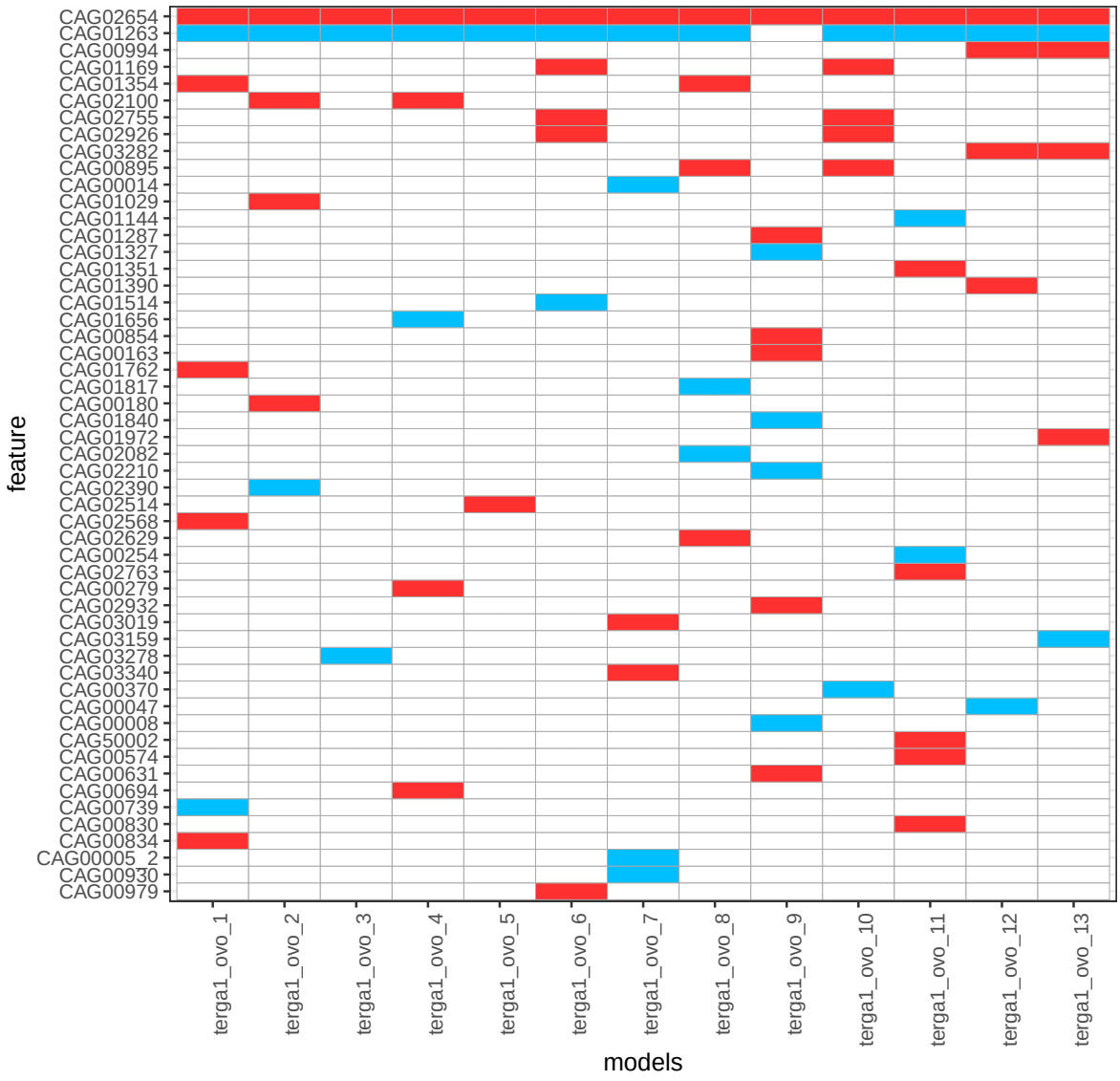
```
[1] "Summary of a population of models with 13 models"
[1] "1:|(+ CAG01354+ CAG01762+ CAG02568+ CAG02654+ CAG00834)  -  (+ CAG01263+ CAG00739) > -1
[1] "2:|(+ CAG01029+ CAG00180+ CAG02100+ CAG02654)  -  (+ CAG01263+ CAG02390) > 1.2e-07 then
[1] "3:|(+ CAG02654)  -  (+ CAG01263+ CAG03278) > -2.1e-08 then class = Prev| (F=0.8262|K=3|
[1] "4:|(+ CAG02100+ CAG02654+ CAG00279+ CAG00694)  -  (+ CAG01263+ CAG01656) > 1.2e-07 then
[1] "5:|(+ CAG02514+ CAG02654)  -  (+ CAG01263) > -7.4e-09 then class = Prev| (F=0.8201|K=3|
[1] "..."
```

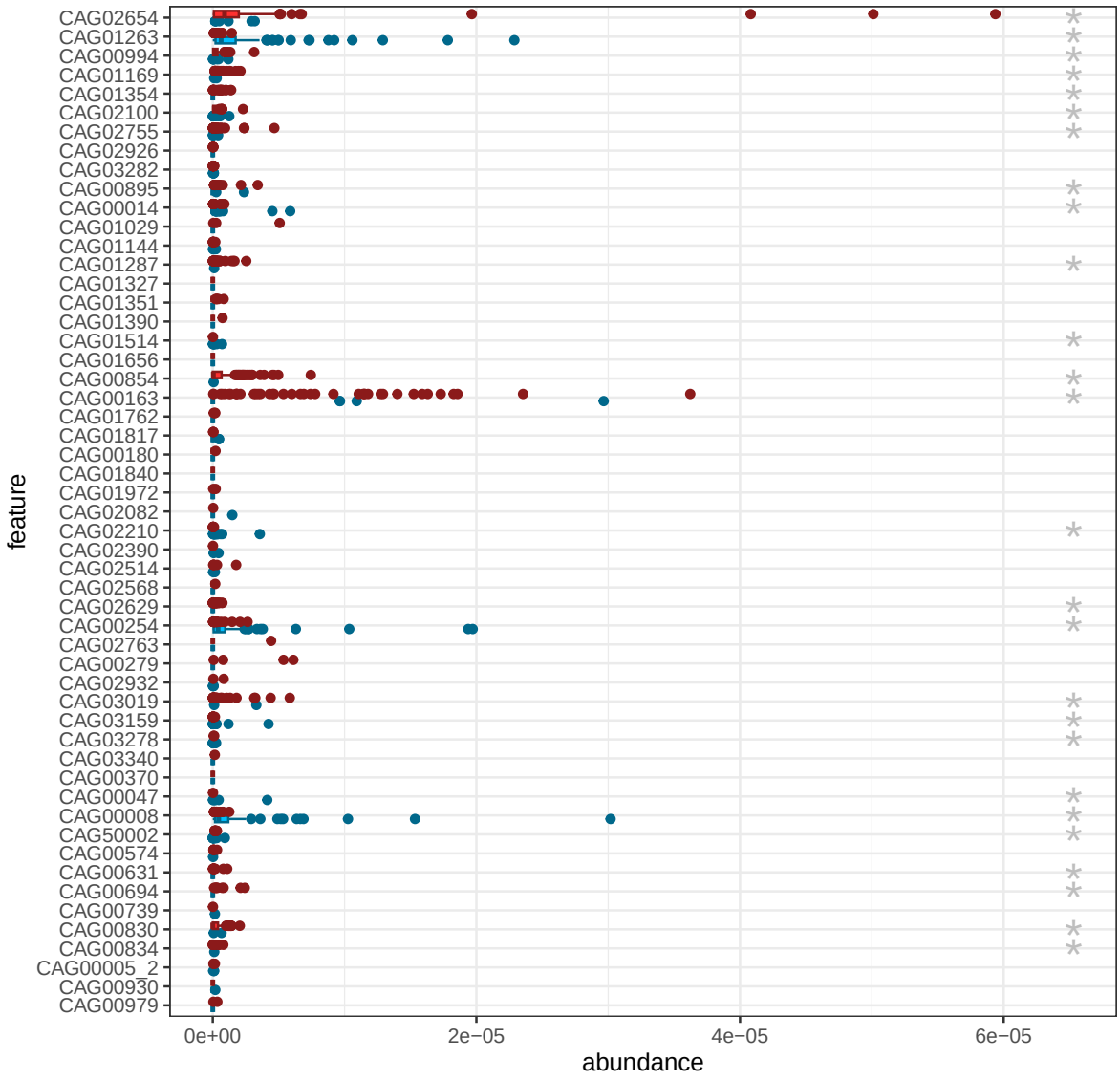


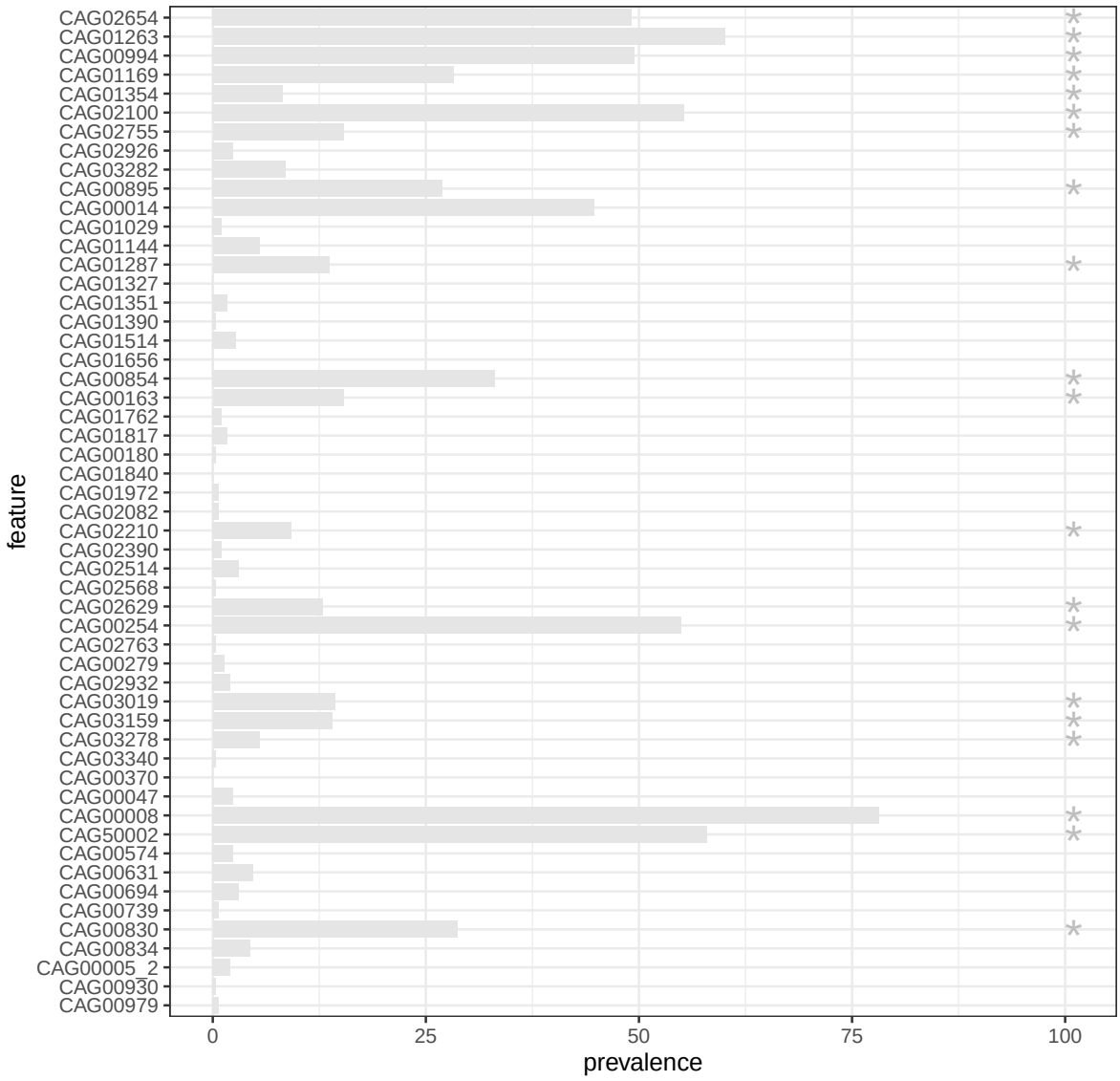
Analyse de la famille des meilleures modèles

L'analyse du FBM peut être très informative pour découvrir les variables les plus importantes dans le processus de prédiction. Voyons d'abord l'utilisation des variables dans les modèles FBM. Nous exécutons pour cela la fonction '?makeFeatureAnnot_ovo' pour obtenir la distribution des fonctionnalités dans la matrice du modèle, qui est l'élément 'pop.noz'. Les modèles du FBM sont classés par précision et le même ordre sera propagé dans la trame de données des coefficients. Il existe 44 fonctionnalités dans les modèles FBM. La figure ci-dessous indique que certaines de ces fonctionnalités sont très répandues dans le FBM, et sont probablement importantes. L'abondance et la distribution de prévalence de ces caractéristiques peuvent être explorées respectivement avec '?plotAbundanceByCalss' et '?plotPrevalence'. Les étoiles grises sur le côté droit des graphiques indiquent des différences significatives entre les groupes de prédiction.

```
[1] 53 13
```

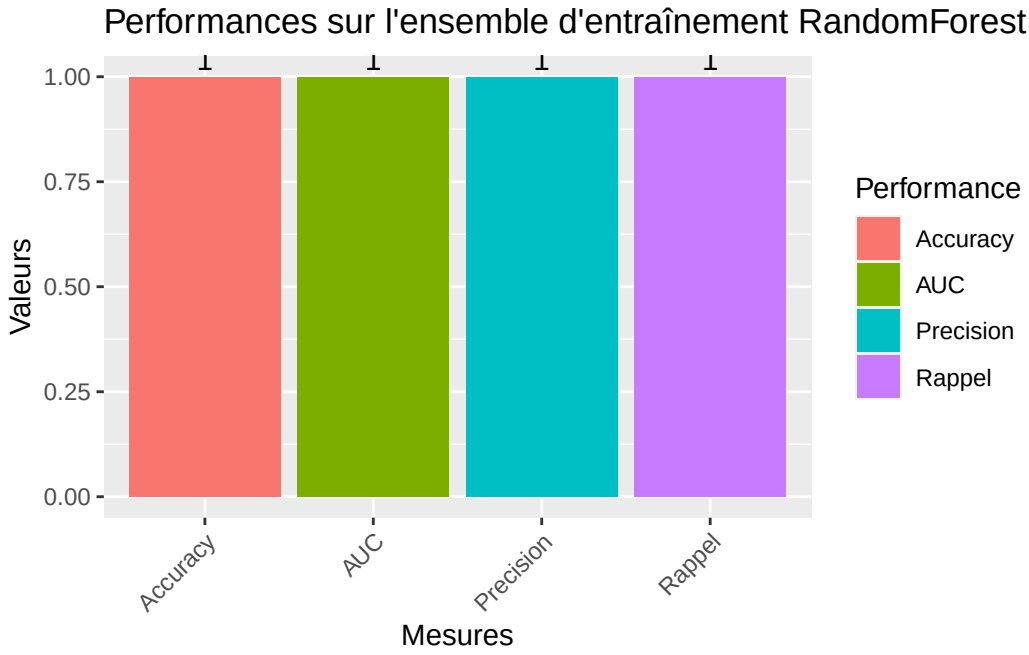

feature
CAG02654
CAG01263
CAG00994
CAG01169
CAG01354
CAG02100
CAG02755
CAG02926
CAG03282
CAG00895
CAG00014
CAG01029
CAG01144
CAG01287
CAG01327
CAG01351
CAG01390
CAG01514
CAG01656
CAG00854
CAG00163
CAG01762
CAG01817
CAG00180
CAG01840
CAG01972
CAG02082
CAG02210
CAG02390
CAG02514
CAG02568
CAG02629
CAG00254
CAG02763
CAG00279
CAG02932
CAG03019
CAG03159
CAG03278
CAG03340
CAG00370
CAG00047
CAG00008
CAG50002
CAG00574
CAG00631
CAG00694
CAG00739
CAG00830
CAG00834
CAG00005_2
CAG00930
CAG00979
terga1_ovo_1
terga1_ovo_2
terga1_ovo_3
terga1_ovo_4
terga1_ovo_5
terga1_ovo_6
terga1_ovo_7
terga1_ovo_8
terga1_ovo_9
terga1_ovo_10
terga1_ovo_11
terga1_ovo_12
terga1_ovo_13
models

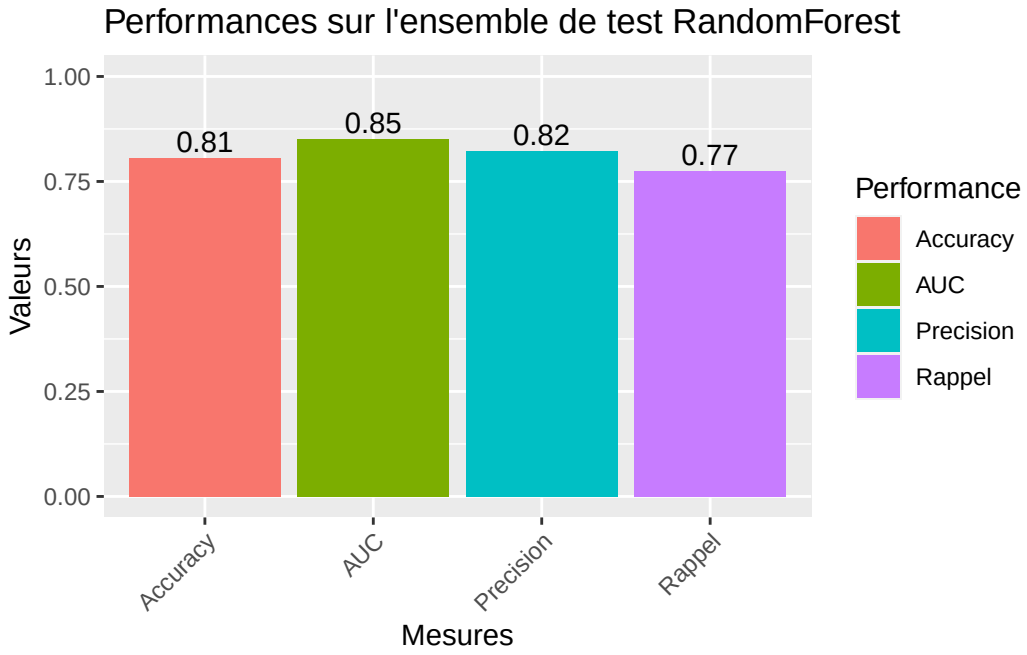

CAG02654
CAG01263
CAG00994
CAG01169
CAG01354
CAG02100
CAG02755
CAG02926
CAG03282
CAG00895
CAG00014
CAG01029
CAG01144
CAG01287
CAG01327
CAG01351
CAG01390
CAG01514
CAG01656
CAG00854
CAG00163
CAG01762
CAG01817
CAG00180
CAG01840
CAG01972
CAG02082
CAG02210
CAG02390
CAG02514
CAG02568
CAG02629
CAG00254
CAG02763
CAG00279
CAG02932
CAG03019
CAG03159
CAG03278
CAG03340
CAG00370
CAG00047
CAG00008
CAG50002
CAG00574
CAG00631
CAG00694
CAG00739
CAG00830
CAG00834
CAG00005_2
CAG00930
CAG00979
feature
0e+00
2e−05
4e−05
6e−05
abundance



RandomForest

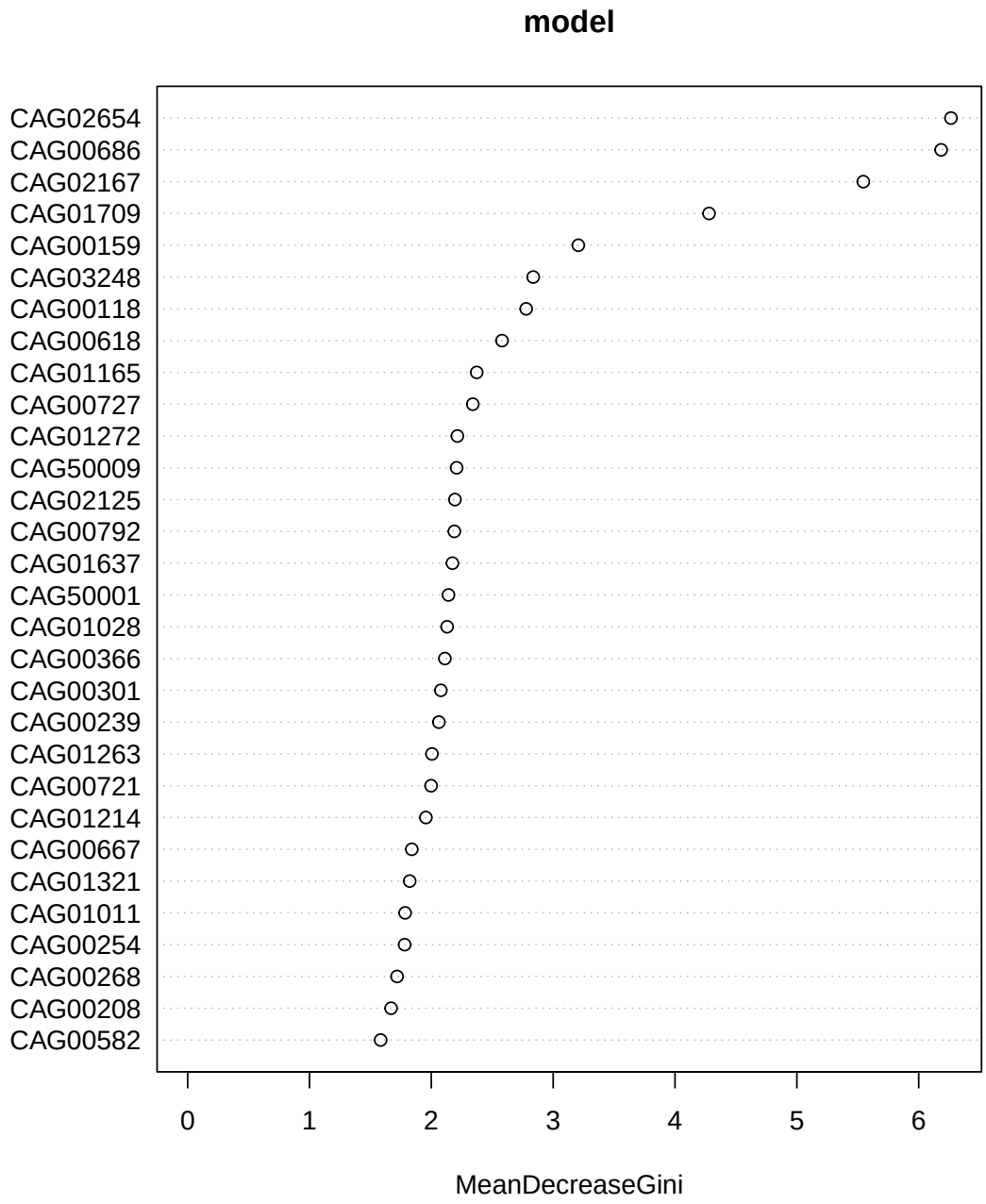
Nous avons entrainé le modèle RandomForest avec les mêmes données utilisées dans notre modèle mcpredomics. Les figures ci-dessous représentent les résultats de différentes métriques obtenues après calculs.


Performances sur l'ensemble d'entraînement RandomForest
1
1
1
1
1.00
0.75
0.50
0.25
0.00
Valeurs
Accuracy
AUC
Precision
Rappel
Mesures
Performance
Accuracy
AUC
Precision
Rappel


Performances sur l'ensemble de test RandomForest
0.81
0.85
0.82
0.77
1.00
0.75
0.50
0.25
0.00
Valeurs
Accuracy
AUC
Precision
Rappel
Mesures
Performance
Accuracy
AUC
Precision
Rappel

Importance des variables

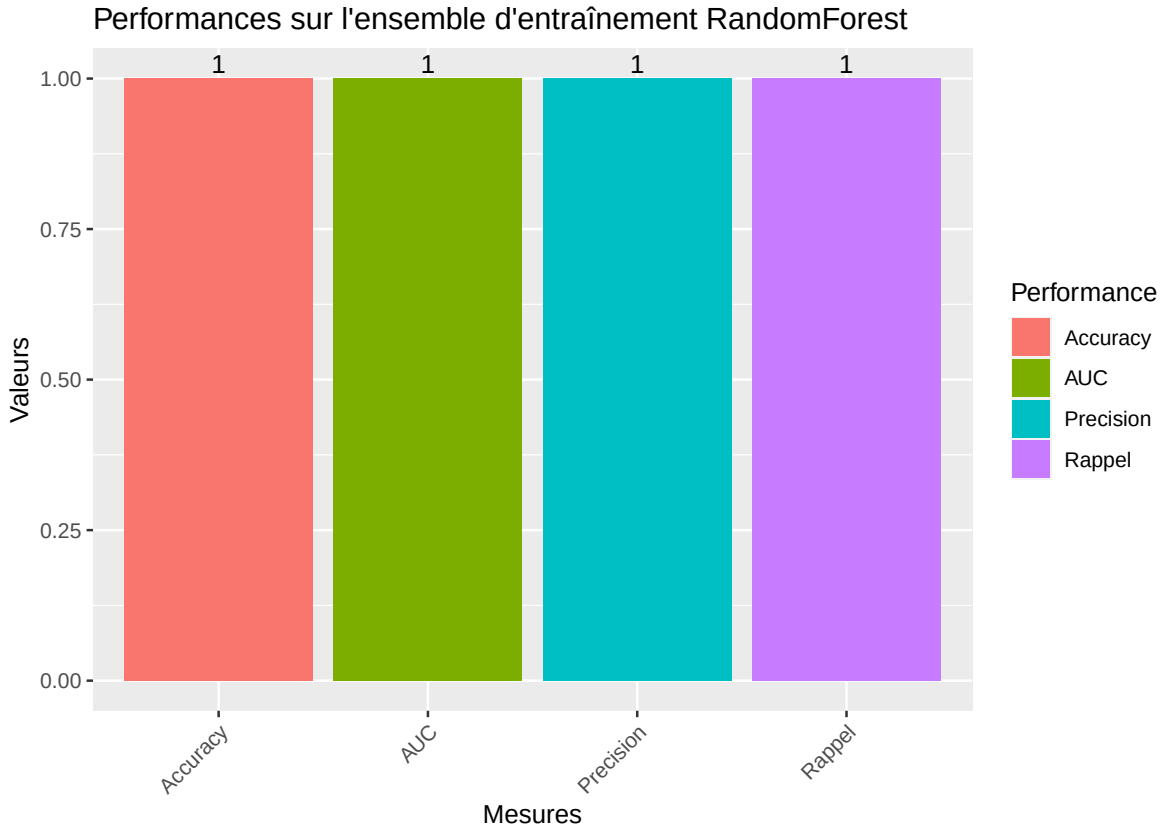
La figure ci-dessous affiche les variables importantes sélectionnées par le modèle RandomForest.

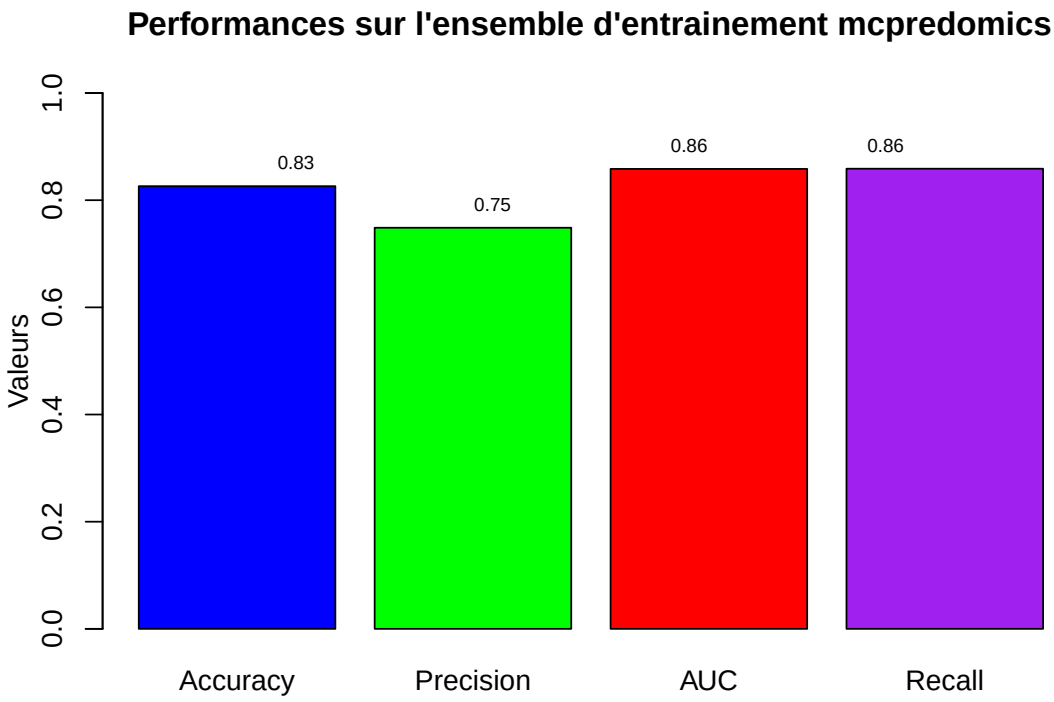


Comparaison des résultats mcpredomics et RandomForest

Phase d'entrainement

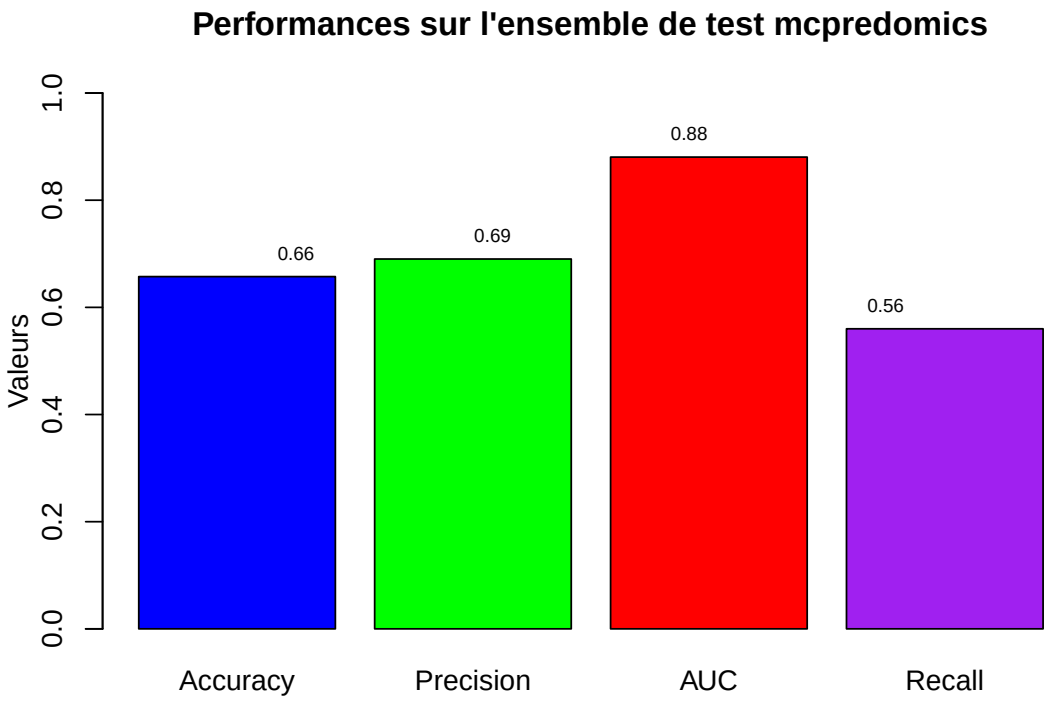
Les performances des différentes métriques sont évaluées à 100% pour RandomForest dans la phase d'entrainement, par contre mcpredomic a des performances estimées à 0.83% d'accuracy, 0.74% de précision, 0.86% d'auc et 0.86% recall.

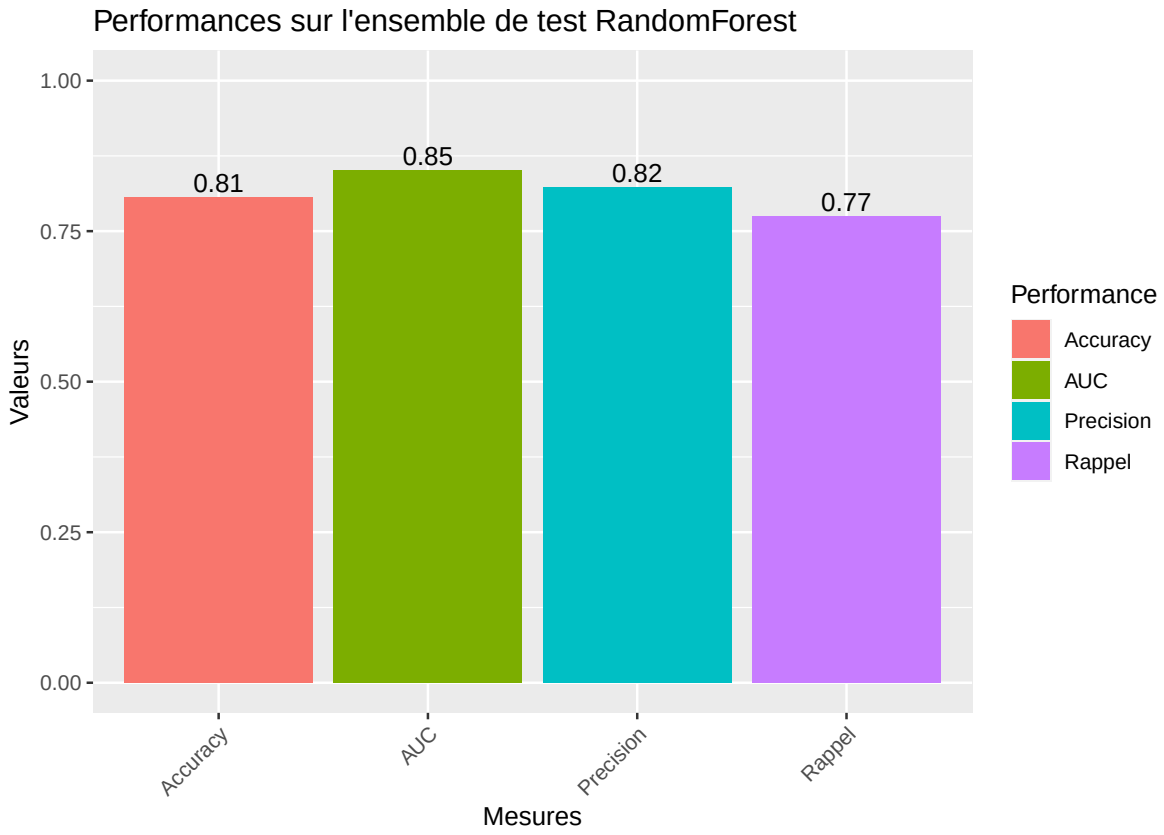




Phase de Test

Les performances de nos deux modèles de tests sont évaluées comme suit : accuracy 0.82 RF contre 0.66 mcpredomics, AUC 0.86 RF contre 0.88 mcpredomics, precision 0.83 RF contre 0.69 mcpredomics et recall 0.79 RF contre 0.56 mcpredomics. Les figures ci-dessous présentent ces différentes performances.


Performances sur l'ensemble de test mcpredomics
Valeurs
0.0
0.2
0.4
0.6
0.8
1.0
0.66
0.69
0.88
0.56
Accuracy
Precision
AUC
Recall



Bien que RandomForest nous ait généré les meilleures performances dans l'ensemble, elle ne peut être utilisée à des fins explicatives.

Discussion

Le package mcpredomics à aider à faire de la classification multiclasses interprétable sur les données du microbiome. Nous comptons expérimenter notre modèle avec d'autres jeux de données afin d'émettre des vraies hypothèses. Les résultats obtenus pour notre première expérience nous rassure pour la suite.