To eat mushroom or not to eat ?

Projet Data Mining

Réalisé par Corentin Ducloux et Guillaume Devant

Printemps 2023 - Semestre 8

Contents

Introduction & Découverte des données

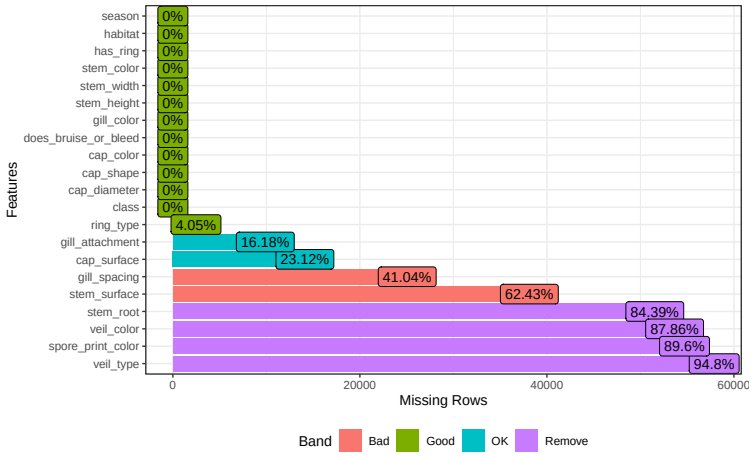
Alors que la saison des champignons approche à grands pas, des cueilleurs viennent solliciter deux modélisateurs très renommés. Les cueilleurs demandent à ces deux modélisateurs de prédire si un champignon est toxique ou non. En effet, la saison dernière, nos cueilleurs en herbe ont eu de très mauvaises surprises en dégustant leur récolte. Alors cette année, c'est décidé, ils ne se feront pas avoir une seconde fois.

Nous, les deux modélisateurs, avons accepté cette mission. Nous n'allons pas nous baser sur les 3 ou 4 souvenirs de nos cueilleurs pour effectuer des prédictions. Nous allons utiliser une base de données répertoriant 61069 champignons avec vingt et une variables. Notre objectif est très simple : nous voulons déterminer si un champignon est toxique ou non. Pour cela, nous allons tester une demi-douzaine de modèles afin de trouver celui qui sauvera la santé de nos cueilleurs.

Nous commencerons d'abord par découvrir nos données, préparer les modèles, puis nous commencerons nos modélisations. Ensuite, nous essaierons nos modèles sur des données de test pour voir s'ils sont efficaces en situation réelle ! Finalement, nous pourrons livrer à nos deux amoureux de la forêt le modèle qu'ils pourront utiliser avant de cuisiner leur récolte de champignons.

Données manquantes

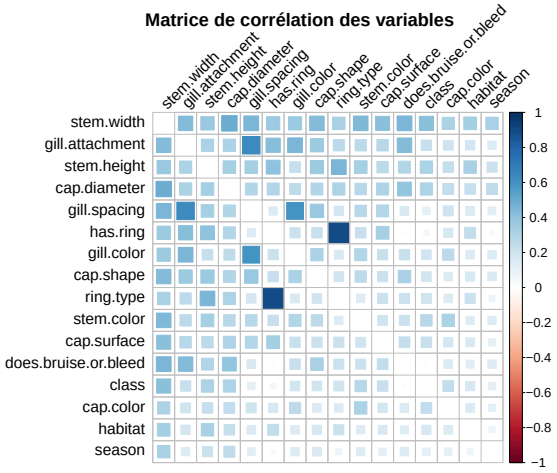
Notre base de données contient 21 variables : la variable que nous cherchons à prédire `class` et 20 variables explicatives. Nous voulons en premier lieu connaître la proportion de données manquantes pour chacune des variables.



On constate que la variable que nous cherchons à prédire ne comporte aucune donnée manquante, ce qui est une bonne nouvelle. En revanche, certaines variables comportent énormément de données manquantes, comme `veil_type` avec 94,8 % de valeurs manquantes. Nous ne pouvons pas tirer beaucoup d'informations de ces variables, c'est pourquoi nous décidons de retirer les variables ayant plus de 50 % de données manquantes. Pour les autres variables, nous allons procéder ultérieurement à des imputations pour combler les données manquantes.

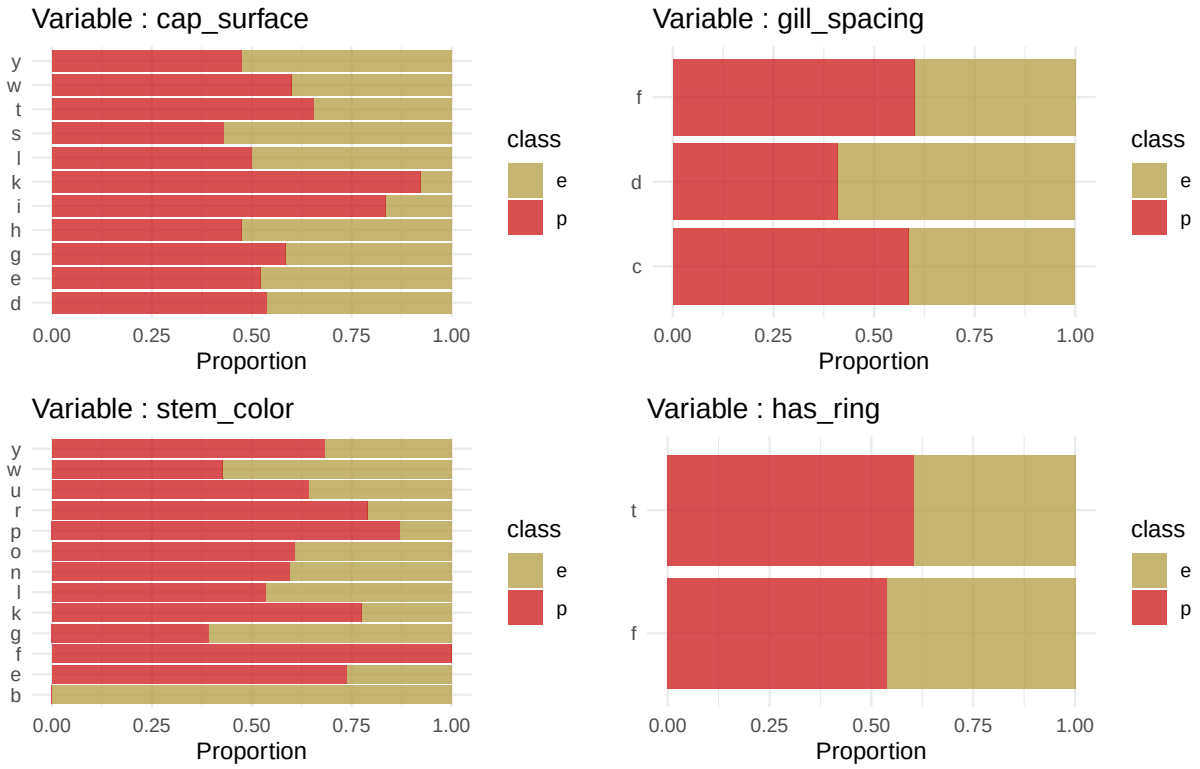
Matrice de corrélation

Nous pouvons observer les corrélations entre nos variables. Il n'y a soit pas de corrélation, soit des corrélations positives ; aucune variable présente dans la base de données n'est corrélée négativement avec une autre variable. Les corrélations les plus importantes se trouvent à l'intérieur des groupes de variables faisant référence aux lames (`gill`), aux pieds (`stem`) et aux anneaux (`ring`) des champignons.



Variables qualitatives

On procède ensuite à une analyse de la proportion des modalités par variable et par classe (comestible ou non) afin de déterminer si certaines modalités sont fortement associées à la classe **comestible** ou **non-comestible**. Nous n'allons pas nous attarder sur l'intégralité des variables catégoriques présentes dans la base de données, mais on peut par exemple s'intéresser aux quatre variables présentées sur le graphique ci-dessous.



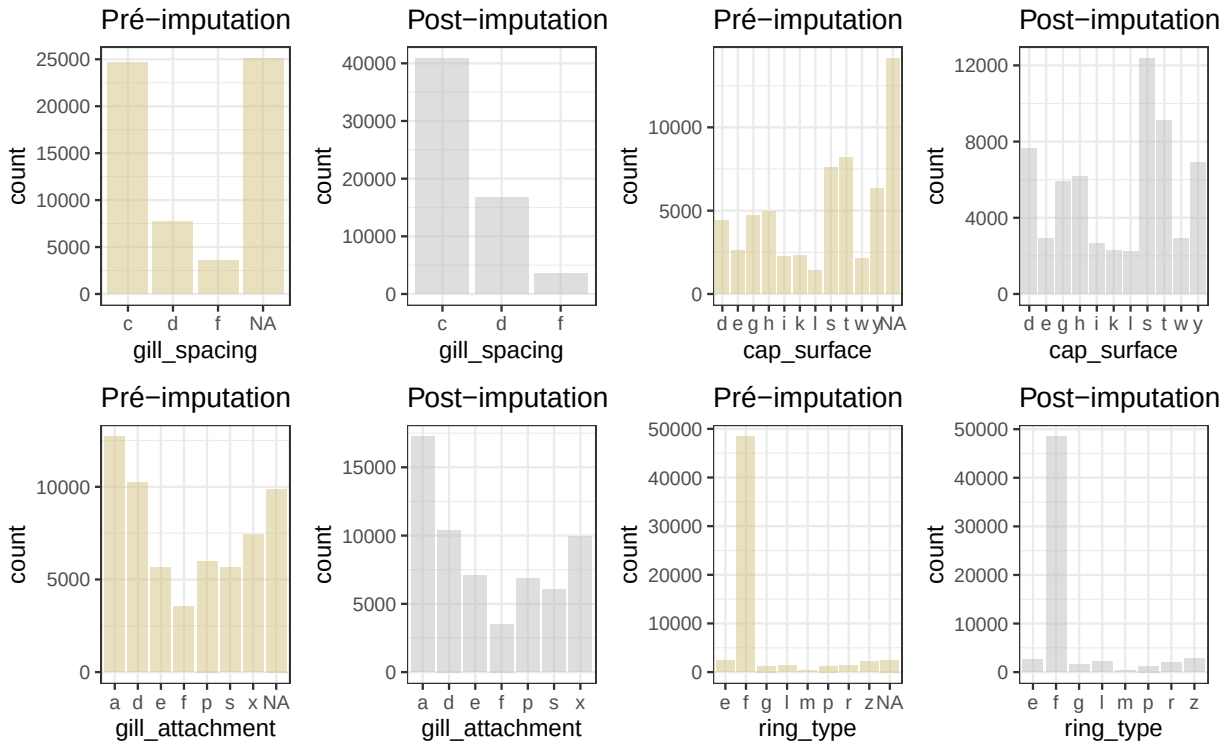
On s'aperçoit pour la variable `cap_surface` (surface du chapeau) que lorsque le champignon a la modalité `i` (fibreux) ou `k` (soyeux), alors dans plus de 80 % des cas, le champignon est toxique. Pour la variable `stem_color` (couleur du pied), si le champignon a une couleur de pied `b` (beige), alors il est comestible dans tous les cas.

Préparation des modèles

Imputation des NA : Traitement des données manquantes

Il y a 5.27 % de données manquantes au total Nous imputons les données manquantes des variables restantes à l'aide de la librairie `simputation` et de l'algorithme CART. Afin de réaliser les imputations, nous utilisons toutes les autres variables excepté `class` qui est la variable à prédire.

Nous vérifions également que la distribution des modalités par variable n'a pas trop changé :



Découpage et seuil

Nous avons choisi une valeur de découpage classique, c'est-à-dire que $\frac{2}{3}$ de notre base de données sera utilisé pour entraîner nos modèles et le $\frac{1}{3}$ restant sera utilisé pour tester nos modèles.

Les modèles sont construits par défaut pour minimiser l'erreur globale.

Lorsque nous appliquerons nos modèles sur **les données de test**, nous allons également ajuster le seuil. En effet, ce qui nous intéresse est plutôt d'éviter aux cueilleurs de s'intoxiquer, nous cherchons donc à maximiser la TER (**spécificité**) la plus "sûre" :

$$P(class = \text{poisonous}|X = x) > z$$

avec z étant le seuil qui minimise le taux de vrais comestibles.

Spécification des modèles

La variable à prédire est `class` et présente les caractéristiques suivantes :

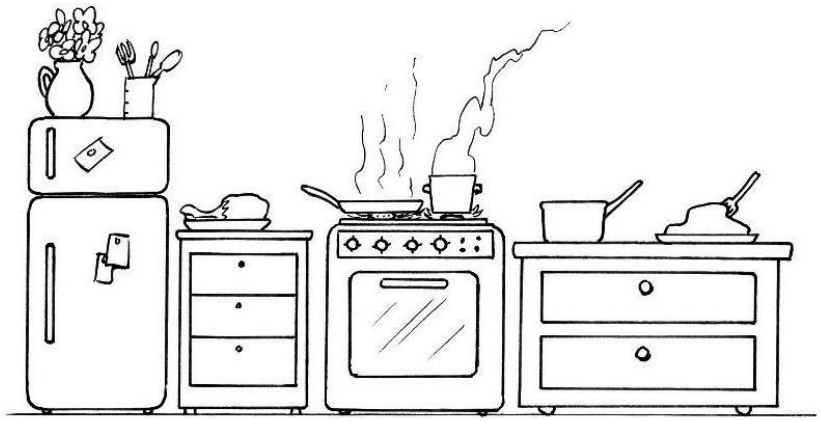
class	n
e	27181
p	33888

Ce qui est intéressant avec cette information, c'est que si l'on prend un champignon au hasard, on a une chance de 45% d'avoir un champignon comestible. L'objectif est de trouver un modèle qui serait plus efficace que le hasard.

Tous les algorithmes présentés ne sont pas en mesure de prendre en charge les mêmes recettes, il convient donc d'adapter les recettes de spécification selon les modèles. On va donc procéder à deux recettes différentes:

- `df_recipe_mixt` prédit la classe du champignon avec 10 variables.
- `df_recipe_qda` prédit la classe du champignon avec 8 variables adaptées pour la QDA.

Nous ne pouvons pas simplement prédire notre variable explicative en fonction de toutes nos variables qualitatives. En effet, on peut être confronté au problème de colinéarité, c'est-à-dire que certaines variables explicatives sont fortement corrélées entre elles, ce qui peut engendrer des résultats faux, un modèle biaisé ou peu précis. Pour résoudre ce problème, il suffit de retirer les variables fortement corrélées entre elles. Le problème a été pris en compte lors de la réalisation des recettes.



En cuisine !

LDA et QDA

LDA : Analyse linéaire discriminante

Pour notre tout premier modèle, nous allons effectuer une LDA avec la recette `df_recipe_mixt`.

Table 2: Matrice de confusion du modèle : LDA

		Réalité		
		comestibles	toxiques	Total
Prédiction	comestibles	11212	5482	**16694**
	toxiques	6816	17202	**24018**
	Total	**18028**	**22684**	**40712**

Table 3: LDA

	GE	TER	TPR
Seuil : 0.5	30.21	62.19	75.83

[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

Le modèle n'est pas terrible, en effet il se trompe environ dans 30% des cas sur les données d'entraînement. Le TER est également très élevé. En utilisant ce modèle, nos cueilleurs courent un grand risque.

QDA : Analyse quadratique discriminante

Pour la QDA, nous ne pouvons pas utiliser la même recette. Nous devons utiliser `df_recipe_qda`, qui comporte notamment deux variables en moins que la recette utilisée précédemment.

Table 4: Matrice de confusion du modèle : QDA

		Réalité		
		comestibles	toxiques	Total
Prédiction	comestibles	12554	7039	**19593**
	toxiques	5474	15645	**21119**
	Total	**18028**	**22684**	**40712**

Table 5: QDA

	GE	TER	TPR
Seuil : 0.5	30.74	69.64	68.97

Bien que nous ayons deux variables en moins, nous avons un modèle à peu près identique en termes d'erreur globale par rapport à la LDA. Cependant, la QDA est bien moins efficace en ce qui concerne le taux de vrais champignons comestibles que la LDA.

LOGIT

Au-delà des modèles AFD que nous venons de réaliser, pour prédire une variable binaire, on peut également utiliser le modèle Logit.

Afin de confectionner notre modèle Logit on va utiliser la recette `df_recipe_mixt`. Pour tous les prochains modèles, nous utiliserons cette recette.

Table 6: Matrice de confusion du modèle : LOGIT

		Réalité		
		comestibles	**toxiques**	**Total**
Prédiction	**comestibles**	11273	5414	**16687**
	toxiques	6755	17270	**24025**
	Total	**18028**	**22684**	**40712**

Table 7: LOGIT

	GE	TER	TPR
Seuil : 0.5	29.89	62.53	76.13

[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

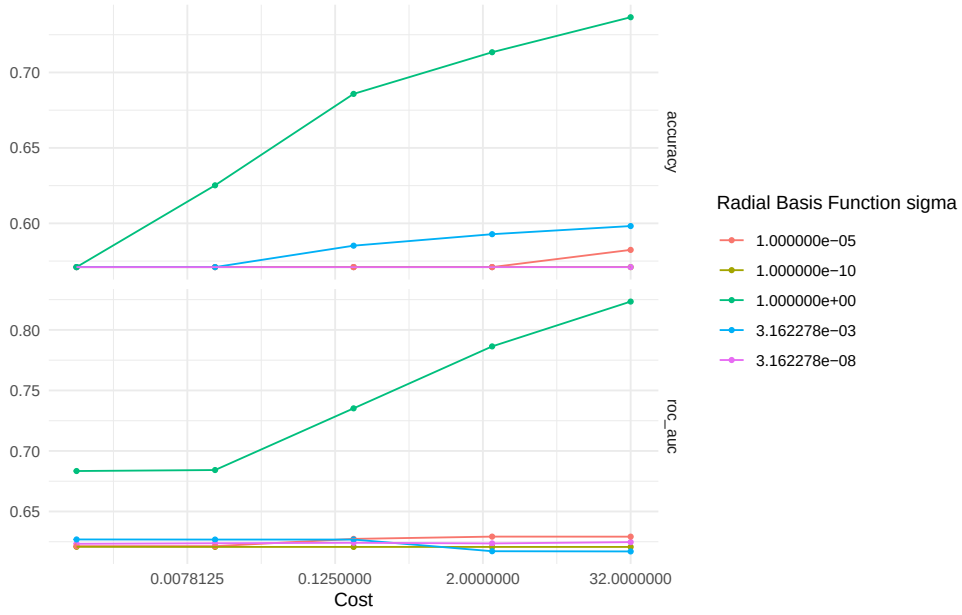
On a une nouvelle fois une erreur globale tournant autour des 30%. Le modèle Logit obtient des résultats similaires aux modèles d'AFD, les modèles paraissent identiques.

SVM

Les résultat de la **SVM Linéaire** sont très mauvais, nous ne la laissons pas ici mais on peut regarder la **SVM Radiale**.

Optimisation des paramètres

Nous devons optimiser deux paramètres, il y a `cost` qui contrôle la marge de séparation entre les classes et `rbf_sigma` qui contrôle la forme de la courbe. Pour optimiser les paramètres on va s'appuyer sur une validation croisé à 5 plis sur les données d'entrainement. Afin de choisir les meilleurs paramètres on va utiliser la métrique `accuracy`.Le modèle SVM n'est entrainé que sur un échantillon de 5000 observations car c'est un modèle qui prend beaucoup de temps à charger.



Après optimisation on obtient un cost = **32** et rbf_sigma = **1**.

Evaluation des performances

Table 8: Matrice de confusion du modèle : SVM

		Réalité		
		comestibles	toxiques	Total
Prédiction	comestibles	1448	535	**1983**
	toxiques	697	2320	**3017**
	Total	**2145**	**2855**	**5000**

Avec bien moins de données d'entrainement, la SVM Radiale est plus performante que les modèles précédents.

Table 9: SVM

	GE	TER	TPR
Seuil : 0.5	24.64	67.51	81.26

[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

KNN

Optimisation des paramètres

Avant de faire tourner notre modèle *k-Nearest Neighbors*, nous devons optimiser le paramètre `k` (le nombre de voisin).

Pour trouver le nombre optimal de $k-$plus proches voisins, on utilise une Cross Validation (validation croisée) à 5 blocs. Cela permet également d'obtenir une estimation plus robuste du biais et de la variance pour le modèle.

En optimisant le paramétre k, on obtient k = **6**.

Evaluation des performances

Table 10: Matrice de confusion du modèle : KNN

		Réalité		
		comestibles	**toxiques**	**Total**
Prédiction	**comestibles**	17980	38	**18018**
	toxiques	48	22646	**22694**
	Total	**18028**	**22684**	**40712**

Table 11: KNN

	GE	TER	TPR
Seuil : 0.5	0.21	99.73	99.83

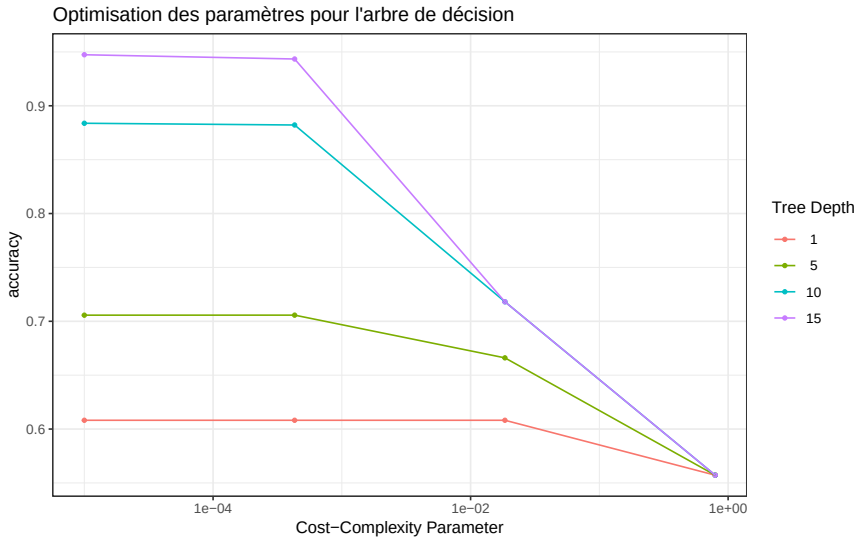
[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

Le modèle KNN donne des résultats exceptionnels, en effet, son erreur globale est inférieure à 1 %. Cependant, ces résultats sont basés sur nos données d'entraînement. Nous allons ensuite tester notre modèle sur des données de test pour vérifier que nous ne sommes pas en présence d'un modèle souffrant de **sur-apprentissage**.

Arbre de décision (CART)

Optimisation des paramètres

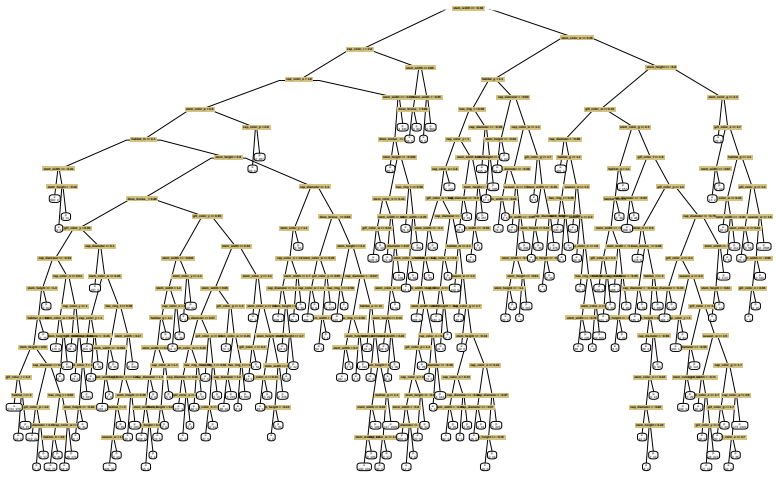
Pour trouver le paramètre optimal de **cost complexity**, on utilise une Cross Validation (validation croisée) à 5 plis. Le paramètre permet ici de trouver un compromis entre la complexité de l'arbre de décision et son adéquation aux données d'entraînement.



Afin de choisir la profondeur optimale de l'arbre, on sélectionne celle qui a la précision (`accuracy`) la plus élevée. Sur le graphique ci-dessus, c'est la courbe la plus élevée. Pour le paramètre `cost-complexity` (cp), nous utilisons la même méthode. En fait, nous choisissons le point "le plus haut du graphique" pour déterminer à la fois la profondeur de l'arbre et la valeur de cp.

Nous obtenons une valeur optimale de `cost-complexity` : **0.00001**

- Visualisation de l'arbre obtenu :



Evaluation des performances

Table 12: Matrice de confusion du modèle : TREE

		Réalité		
		comestibles	**toxiques**	**Total**
Prédiction	**comestibles**	17453	1374	**18827**
	toxiques	575	21310	**21885**
	Total	**18028**	**22684**	**40712**

Table 13: TREE

	GE	TER	TPR
Seuil : 0.5	4.79	96.81	93.94

[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

Notre arbre de décision a une erreur globale inférieure à 5%, ce qui nous donne un bon modèle. Cependant, comme pour les modèles **KNN**, il est important de vérifier que le modèle ne souffre pas de sur-apprentissage en testant sur des données de test.

Random Forest

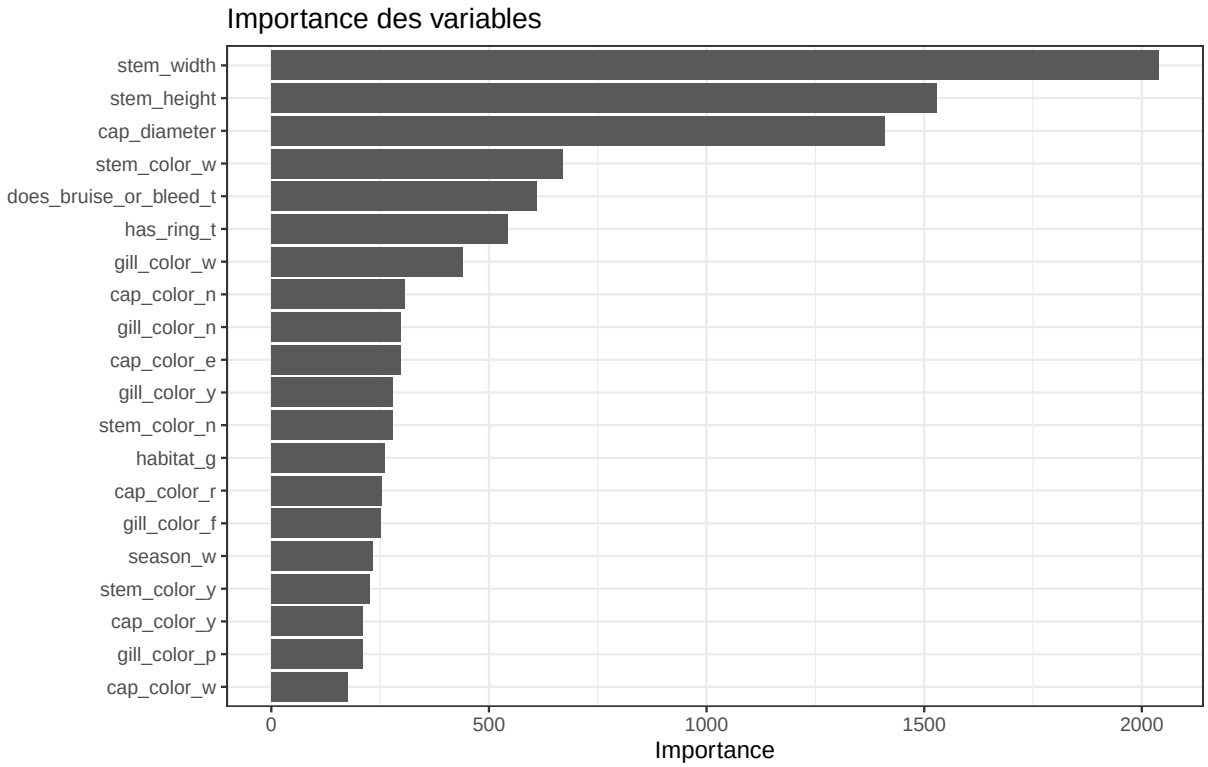
Optimisation des paramètres

Pour construire une forêt aléatoire, nous devons procéder à deux tirages aléatoires. Le premier pour le boostrapping, c'est-à-dire tirer un échantillon aléatoire d'individu avec remise pour construire chaque arbre. Le deuxième est un tirage parmi les variables, pour chaque arbre nous allons utilise `mtry` variables pour le construire. Le nombre de variables à tirer est un paramètre à sélectionner (`mtry`). Mais avant tout, combien d'arbres allons-nous choisir pour faire pousser notre forêt ? Pour cela, nous devons optimiser le paramètre `ntrees`. On optimise également le paramétre `tree_depth` afin de contrôler la profondeur de l'arbre

Après optimisiation des trois hyperparamétres, on obtient **ntrees = 500**, **mtry = 5** et **tree_depth = 10**.

- Importance des variables :

L'importance des variables est un indicateur qui permet de connaître les variables ayant le plus d'impact sur les prédictions. Nous pouvons constater que les trois variables les plus importantes sont également les seules variables quantitatives du modèle.



Mesures de performance

Table 14: Matrice de confusion du modèle : RF

		Réalité		
		comestibles	**toxiques**	**Total**
Prédiction	**comestibles**	17683	245	**17928**
	toxiques	345	22439	**22784**
	Total	**18028**	**22684**	**40712**

Table 15: RF

	GE	TER	TPR
Seuil : 0.5	1.44	98.14	98.9

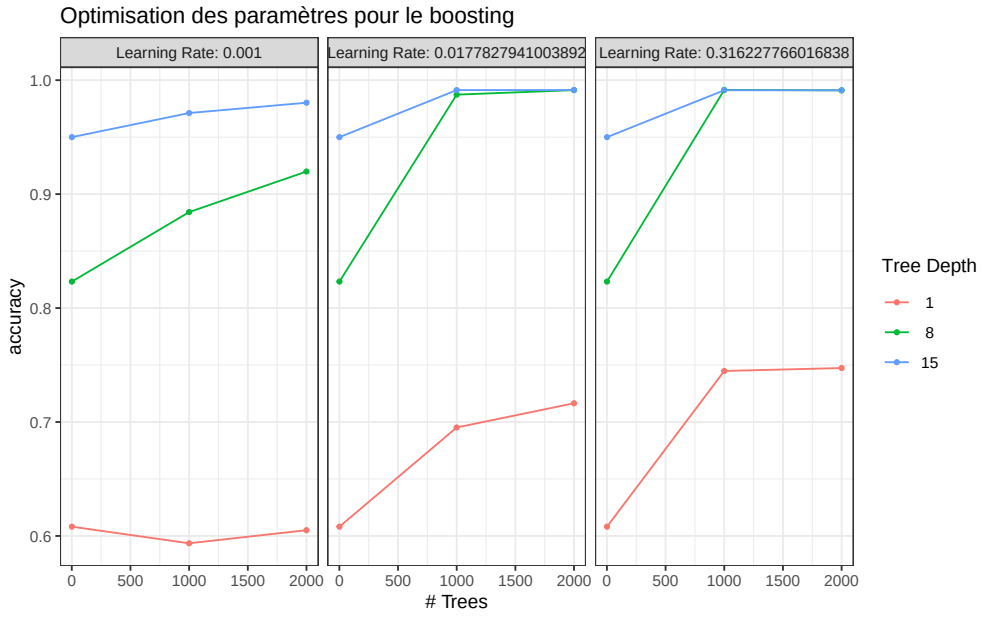
[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

Notre modèle Random Forest est meilleur qu'un simple arbre de décision.

Boosting

Optimisation des paramètres

Le modèle de boosting repose sur les arbres de décision, comme le modèle Random Forest. Nous allons donc optimiser les paramètres liés aux arbres `trees` (nombre d'arbres) et `tree_depth` (profondeur des arbres). Il existe également un paramètre `learn_rate` qui contrôle la vitesse d'apprentissage du modèle. Nous pouvons visualiser cette optimisation ci-dessous.



On choisit alors les paramétres suivant : **Trees = 1000**, **Tree Depth = 8** et **Learning Rate = 0.3162**

Mesures de performance

Table 16: Matrice de confusion du modèle : Boosting

		Réalité		
		comestibles	toxiques	Total
Prédiction	comestibles	18028	0	**18028**
	toxiques	0	22684	**22684**
	Total	**18028**	**22684**	**40712**

Table 17: Boosting

	GE	TER	TPR
Seuil : 0.5	0	100	100

[a] Erreur globale de classement : GE
[a] Taux de vrais comestibles : TER
[a] Taux de vrais toxiques : TPR

Le modèle de boosting est très bon, au vu de ses résultats, il fait concurrence au modèle KNN.

Degustation

Nous allons maintenant évaluer nos modèles sur les données de test. Il est possible qu'un modèle soit très performant sur les données d'entraînement mais moins bon sur les données de test, ce qui peut indiquer un problème de sur-apprentissage. Cette étape permet de confronter nos modèles à des données qu'ils n'ont pas vues auparavant et pour lesquelles nous connaissons la réponse concernant la toxicité des champignons, mais pas le modèle. Nous nous attendons à avoir des taux d'erreurs globaux plus élevés sur nos données de test que sur nos données d'entraînement, mais si un modèle est bon, la différence ne devrait pas être très importante.

Nous allons également modifier le seuil de décision afin d'augmenter le taux de vrais comestibles. Nous allons donc diminuer le seuil. Pour chaque modèle, nous utiliserons un seuil de 0.5 (celui par défaut) et un seuil plus bas, qui sera différent en fonction de chaque modèle. Il sera choisi de manière à maximiser le `TER` (Taux de vrais comestibles).

Table 18: Comparaison de tous les modèles sur les données de test

	GE	TER	TPR
LDA	30.05	68.11	71.24
QDA	31.18	64.37	72.91
LOGIT	29.57	68.75	71.60
KNN	0.68	99.35	99.30
ARBRE	5.16	92.67	96.73
RF	1.89	98.29	97.97
BOOST	0.76	99.26	99.22
SVM	23.79	74.97	77.13

Pour faire le tri entre tous ces modèles, nous allons principalement regarder l'erreur globale et si nécessaire regarder le `TER` pour les départager. On constate que la LDA, la QDA, et le LOGIT sont trois modèles qui tournent autour des 30% d'erreur globale. C'est énorme, ce sont de très mauvais modèles. Cependant, ce n'est pas une surprise car en effet, c'était à quelques points de pourcentage près les résultats que nous avions obtenus sur les données d'entraînement. La SVM est considérée comme légèrement supérieure aux modèles précédents car elle présente une erreur globale inférieure.

En regardant le modèle KNN, on change totalement de dimension. Le modèle est extrêmement performant avec une erreur globale inférieure à 1%. On se rappelle que nous craignions un surapprentissage du modèle car les résultats étaient déjà très bons sur les données d'entraînement. Cependant, ce n'est pas le cas. Le modèle est également très bon sur les données de test.

Le modèle d'arbre de décision est un modèle moyen, il y a tout de même entre 5 et 6% d'erreur sur les données de test. Mais c'est avec les modèles Random Forest et Boosting que les arbres ont tout leur intérêt. Ce sont tous les deux de très bons modèles. Boosting est le meilleur des deux car il a, comme le modèle KNN, moins de 1% d'erreur globale. On ne s'attendait pas à avoir d'aussi bons résultats sur les données tests.

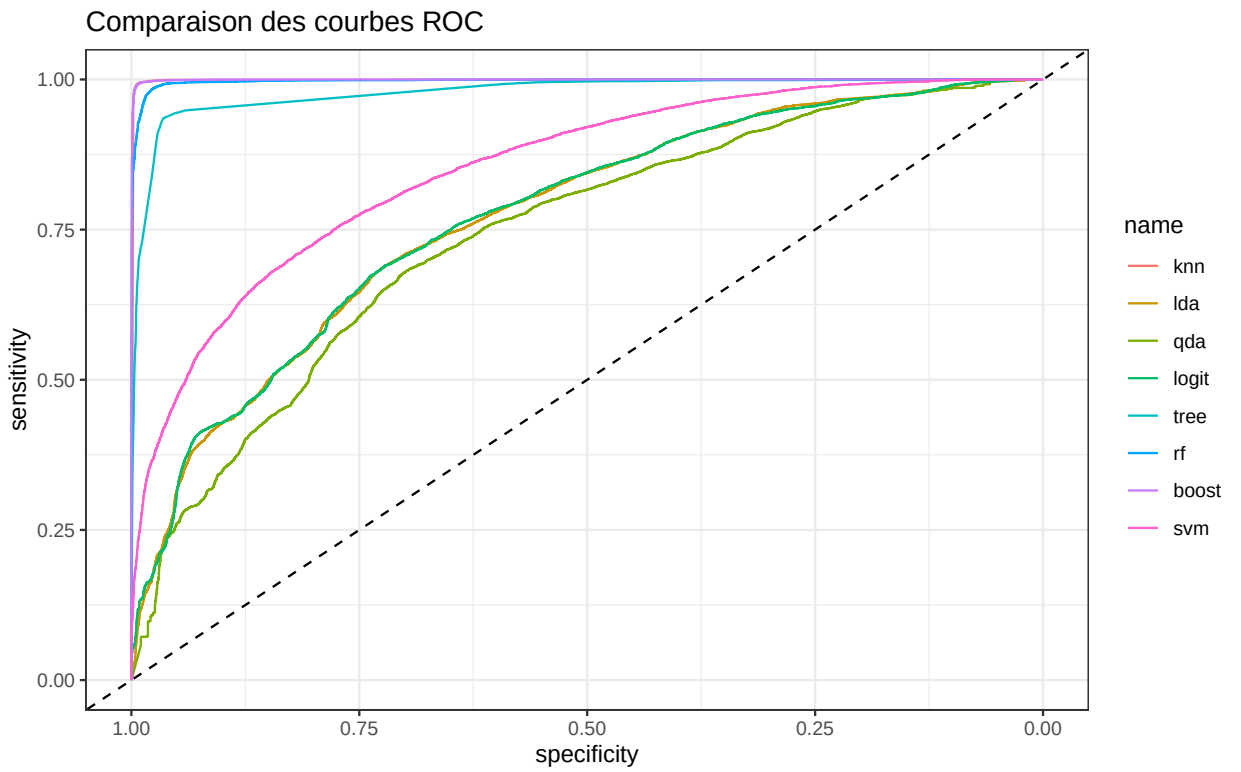
Faire varier le seuil pourra également faire gagner quelques dixièmes de point de pourcentage au `TER` à ces bons modèles. Mais ce ne sera pas suffisant pour départager les très bons modèles entre eux. Il va falloir trouver une autre solution.

Les Seuils

Nous avons des modèles suffisamment performants pour qu'une variation de seuil soit négligeable sur le résultat. Mais pour les modèles moins bons, ce n'est pas le cas. Une variation de seuil peut faire varier de plusieurs points de pourcentage le TER. L'objectif en diminuant les seuils est de faire augmenter le TER, la colonne de couleur bleue. On constate que la variation du seuil permet d'augmenter le TER de manière significative.

Seuil	GE	TER	TPR
LDA			
0.5	30.05	68.11	71.24
0.4	31.66	73.24	66.37
QDA			
0.5	31.18	64.37	72.91
0.2	32.83	69.23	66.22
LOGIT			
0.5	29.57	68.75	71.60
0.4	31.75	73.42	66.22
KNN			
0.5	0.68	99.35	99.30
0.3	0.93	99.56	98.68

Courbe ROC



En regardant les AUC on se rend compte que le modèle QDA a la moins bonne précision et les modèles Random Forest, KNN et Boosting ont les aires sous la courbe les plus importantes, ce sont donc les meilleurs modèles en terme d'AUC.

Table 19: Comparaison des AUC

Modèle	AUC
LDA	0.771
QDA	0.742
LOGIT	0.772
KNN	0.998
ARBRE	0.979
RF	0.997
BOOST	0.999
SVM	0.852

F1-Score

Jusqu'à maintenant on a essentiellement utilisé l'erreur globale pour évaluer nos modèles. Le F1-score est une autre métrique permettant d'évaluer nos modèles. Il se réfère quant à lui à la précision et au rappel.

$$\text{Precision} = \frac{VP}{VP + FP}$$

$$\text{Rappel} = \frac{VP}{VP + FN}$$

Avec VP : Vrai Positif, FP : Faux Positif et FN : Faux négatif.

$$0 < F_1 \text{ - Score} = 2 \times \frac{\text{precision} \times \text{rappel}}{\text{precision} + \text{rappel}} < 1$$

Ce score se situe entre 0 et 1. Un modèle ayant un F1-Score de 0 est très mauvais et un score proche de 1 très bon.

Table 20: Comparaison des F1-Score

Modèle	F1-Score
LDA	0.651
QDA	0.664
LOGIT	0.656
KNN	0.992
ARBRE	0.944
RF	0.979
BOOST	0.992
SVM	0.728

Conclusion

Les trois meilleurs modèles sont donc le **KNN**, le **Boosting** et le **Random Forest**. Cependant, ce dernier est légèrement moins bon que les deux premiers. Il est très difficile de départager ces deux modèles, mais nous devons choisir un modèle pour nos chers cueilleurs. Une analyse plus poussée de l'AUC et du F1-Score nous indique que l'on préférera le modèle **Boosting** - mais la différence est très légère.

Sources

- Base de données : UCI (Machine Learning Repository) Secondary Mushroom Dataset Data Set
- Image page de garde : pixabay - creozavr
- Image page 3 : Champignons sauvages - Guylaine Duval
- Image page 7 : papik.pro
- Image page fancy head : espace des sciences
- Logo : Université de Tours