Projet 2 : Classification de la surface de jeu de parties de tennis

 Lien GitHub

Jean-Thomas Baillargeon
Christopher Blier-Wong
Stéphane Caron
Sofia Harrouch

Pour le cours STT-7330
Méthode d'analyse des données

Présenté le 20 avril 2018 à la professeure

Anne-Sophie Charest

Département de mathématiques et de statistiques
Faculté des sciences et de génie
Université Laval

Faculté des sciences et de génie
Université Laval
Hiver 2018

Table des matières

1 Introduction

L'apprentissage statistique est une discipline de l'intelligence artificielle permettant de transmettre des connaissances à une machine apprenante. La nature des connaissances transmises est très variée de telle sorte qu'il est possible d'apprendre à une machine à réaliser des tâches sur n'importe quel sujet. De plus, si ces connaissances contiennent des relations de causalité entre les différentes variables, il sera possible d'entraîner une machine à faire des généralisations sur les données. Ces généralisations peuvent, par la suite, lui permettre de faire des prédictions sur de nouvelles données qui n'ont jamais été observées.

Le projet réalisé par notre équipe porte sur l'utilisation de techniques dans le contexte d'une tâche d'apprentissage supervisé. La tâche à accomplir est de prédire, suite à une partie de tennis, la surface sur laquelle cette partie a été jouée. Pour accomplir cette tâche, différentes techniques d'exploration de données et d'apprentissage automatique à complexité variée ont été utilisées. Dans ce rapport, l'équipe présente les algorithmes utilisés, les résultats obtenus ainsi que leurs conclusions quant à la tâche à accomplir.

2 Méthodologie

2.1 Présentation du jeu de données

Le jeu de données utilisé dans ce travail est composé de différentes statistiques sur les parties officielles de l'association des professionnels de tennis (ATP) depuis 1968. Les données sont distribuées par Jeff Sackmann via **GitHub** et accessibles sous ce lien.

Le jeu de données comprend entres autres la surface sur laquelle un match de tennis a été joué. C'est cette variable que la machine apprenante devra prédire. Il s'agit d'une variable catégorielle pouvant prendre 3 valeurs soit : terre battue (*clay*), dur (*hard*) et gazon (*grass*) avec des fréquences 10981, 18253 et 3651 respectivement. Il y a environ 50 autres variables qui sont utilisées pour tenter d'expliquer la variable à prédire. Ces valeurs représentent des caractéristiques se rapportant à 3 catégories :

1. informations sur le match
   (a) nom du tournoi
   (b) ville du tournoi
   (c) ...
2. informations sur les joueurs
   (a) nom du joueur
   (b) nationalité du joueur
   (c) ...

3. statistiques de la partie
   (a) temps en minutes du match
   (b) nombre d'as réalisés par le gagnant
   (c) nombre d'as réalisés par le perdant
   (d) ...

2.2 Prétraitement des données

L'information contenue dans le jeu de données était très clairsemée. Le fléau de la dimensionnalité rend l'apprentissage beaucoup plus difficile dans un tel contexte. L'entraînement de modèles basé sur de telles données augmente donc les chances d'obtenir des modèles non optimaux ou peu fiables. Afin de pallier ce problème, nous avons décidé d'améliorer la qualité du jeu de données. Les différentes étapes ont été de nettoyer les données, d'identifier et traiter les valeurs manquantes, d'intégrer et transformer certaines variables et enfin de réduire la dimensionnalité.

2.2.1 Nettoyage des données

Afin d'obtenir des données pouvant être utilisées par nos algorithmes, nous avons effectué quelques transformations sur le jeu de données initial. Premièrement, nous avons enlevé tous les matchs ayant comme surface tapis (*carpet*), car il n'y avait en avait pas suffisamment et cela rendait la tâche de classification multi-classes beaucoup plus complexe.

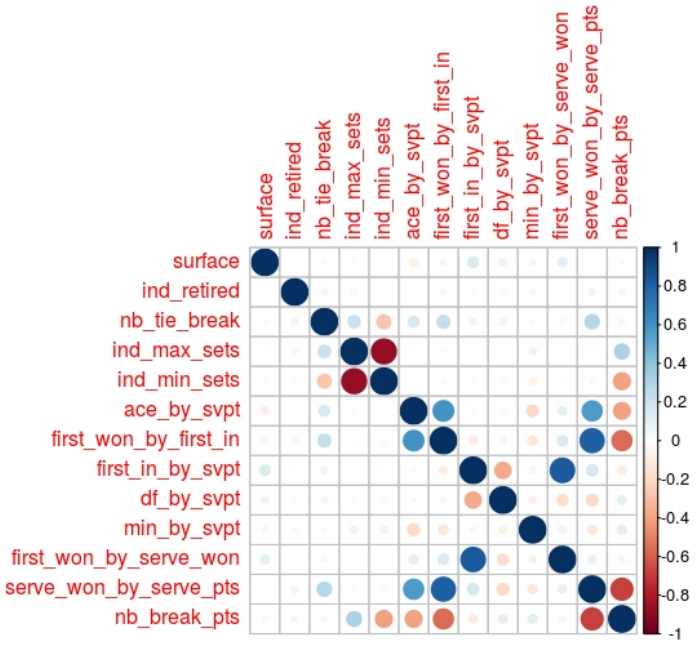
Les données dépendantes de la surface du jeu ont aussi été enlevées. En effet, dans le cas où ces données auraient été conservées, la machine apprenante aurait pu tout simplement utiliser ces variables et la classification aurait été triviale. Par exemple, le tournoi de Winbledon est toujours joué sur gazon, cela devient donc équivalent à laisser la variable réponse. Finalement, les variables considérées non informatives (tels le nom du joueur, la grandeur et le poids du joueur, etc.) ont été enlevées.

2.2.2 Intégration et transformation des variables (*Feature engineering*)

Cette étape est cruciale dans un problème de modélisation, car elle permet de créer d'autres variables non linéairement dépendantes des autres. Cela permet d'avoir des informations additionnelles qui n'étaient pas données en entrée au modèle dans la structure initiale des données. Il s'agit donc d'aider le modèle en lui donnant certains indices par la connaissance du problème a priori. Le choix de ces variables est généralement basé sur l'expérience et l'intuition du domaine et du jeu de données. Les variables qui ont été créées sont essentiellement des ratios entre différents éléments du jeu de données ou des déclencheurs lorsque certains évènements se sont produits lors du match.

Dans les variables retenues, on présente dans la figure 1 la matrice des corrélations.

FIGURE 1 – Matrice de corrélation visuelle des variables



On remarque une corrélation positive entre la variable de surface et les variables `first_won_by_first_in` et `first_won_by_serve_won`. La variable `ace_by_svpt` est corrélée négativement avec la surface. Dans les figures suivantes, on présente le *boxplot* et l'histogramme de `ace_by_svpt` selon la surface.

FIGURE 2 – *Boxplot* des as par service en fonction de la surface

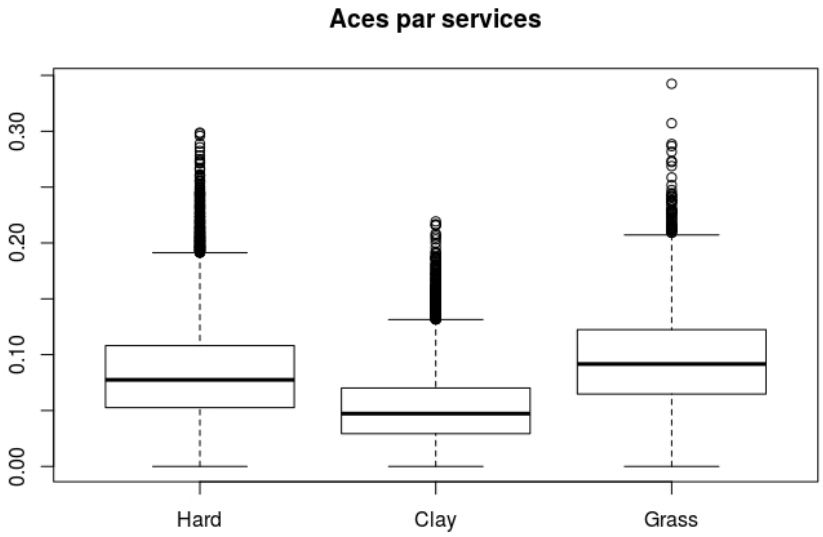
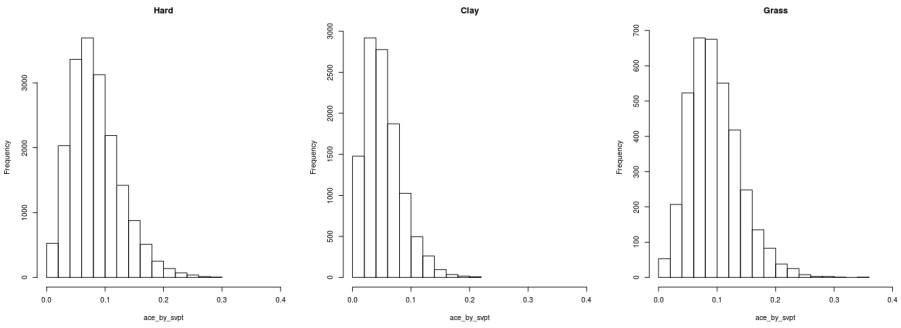


FIGURE 3 – Histogramme des as par service en fonction de la surface



On observe que la distribution de `ace_by_svpt` est de queue moins lourde pour la surface terre battue.

Les autres statistiques ne sont pas très intéressantes pour l'analyse. On ne s'attend pas à ce qu'une seule variable soit plus importante pour les autres pour faire la classification des types de surface.

2.2.3 Traitement des valeurs manquantes

Il n'y avait pas réellement de données manquantes dans le jeu de données après avoir fait le premier prétraitement. Ainsi, nous avons utilisé l'analyse des cas complets étant donné qu'un pourcentage très minime des données étaient manquant (environ 3%). Nous avons cependant imputé des valeurs de 0 pour les ratios conçus précédemment pour lesquels il y avait une division par zéro.

2.2.4 Réduction de la dimensionnalité

Tel que mentionné un peu plus tôt, le jeu de données comprend environ une cinquantaine de variables explicatives servant à entraîner un modèle. Ceci peut causer un problème si certaines de ces variables sont non-informatives ou redondantes. Nous avons donc considéré l'approche de faire une analyse en composantes principales (*PCA*) sur le jeu de données et comparer les résultats ceux obtenus sur le jeu de données non réduit. Les projections produites par l'ACP permet d'obtenir une représentation réduite conservant une grande partie de l'information, produisant les mêmes (ou presque) résultats analytiques. L'ACP a réduit le nombre des variables utilisées à 6, tout en conservant une variance totale supérieure à 80 %.

2.3 Classifieurs utilisés

Les algorithmes d'apprentissage automatique utilisés pour le projet varient en complexité. Nous désirions tester différents niveaux de complexité afin de déterminer si une augmentation de la complexité était justifiée relativement à la qualité des prédictions. Nous avons librement sous-divisé les algorithmes utilisés en 3 catégories distinctes.

En apprentissage automatique, il y a un compromis à faire entre les connaissances des données et la quantité de données disponibles. Dans le cas où le statisticien connaît bien les données, un modèle très simple ou il peut utiliser son opinion pour ajuster une distribution de probabilité aux données, comme c'est le cas dans les méthodes paramétriques simples présentées plus loin. En revanche, si le statisticien ne connaît pas la distribution des données, il doit utiliser des modèles plus complexes non paramétriques, comme des réseaux de neurones. Dans le milieu de ce compromis, on retrouve des modèles tels que le SVM qui requiert de la connaissance sur le noyau et seulement les hyperparamètres doivent être déterminés.

2.3.1 Algorithmes simples

La première catégorie est composée d'algorithmes simples et ne demandant qu'une compréhension générale du modèle afin de les utiliser. Les modèles sont :

1. le classifieur de naïf de Bayes, du paquetage `e1071` [Meyer et al., 2017]
2. le classifieur régression logistique, du paquetage `nnet` [Venables and Ripley, 2002a]

3. l'analyse discriminante linéaire, du paquetage `MASS` [Venables and Ripley, 2002b]
4. l'analyse discriminante quadratique, du paquetage `MASS` [Venables and Ripley, 2002b]

Nous considérons qu'ils sont simples, car ils sont utilisables directement avec une fonction R et ne demandent généralement pas de sélection d'hyperparamètres complexes.

2.3.2 Algorithmes intermédiaires

La deuxième catégorie est composée d'algorithmes un peu plus complexes et demandant un ajustement spécifique de leurs hyperparamètres. Ces modèles sont discriminants, c'est-à-dire qu'ils ne font pas d'hypothèse sur la distribution des données. Les modèles sont :

1. le classifieur par arbre, du paquetage `rpart` [Therneau et al., 2017]
2. le classifieur par forêt aléatoire, du paquetage `randomForest` [Liaw and Wiener, 2002]
3. le classifieur SVM, du paquetage `e1071` [Meyer et al., 2017]

La méthodologie entourant la sélection des hyperparamètres est présentée dans une section ultérieure.

2.3.3 Algorithmes complexes

Finalement, la troisième catégorie est composée d'algorithmes plus complexes et demandant l'utilisation de l'expérience et l'intuition du statisticien. Ce sont des modèles qui n'ont pas été étudiés dans le cours STT-7330 mais qui ont été explorés, car ils sont généralement très performants dans les tâches de classification . Les modèles sont :

1. les réseaux de neurones, du paquetage `keras` [Allaire and Chollet, 2018]
2. le modèle par ensemble, développé dans le cadre du projet

L'utilisation et l'architecture de ces modèles seront présentées dans une section ultérieure.

2.4 Ajustement des hyperparamètres

L'ajustement des hyperparamètres a été fait par une recherche exhaustive. La technique de recherche par quadrillage (*grid-search*). Afin de ne pas surajuster les modèles lors de cette recherche, nous avons utilisé la validation croisée avec 10 plis.

La recherche par quadrillage est une technique permettant de trouver les meilleurs hyperparamètres en essayant toutes combinaisons entre les différentes valeurs prédéfinies de chacun de ces hyperparamètres. Comme notre connaissance du jeu de données était plutôt limitée, nous avons décidé de considérer une large gamme de valeurs et de faire une recherche exhaustive pour chacune des combinaisons d'hyperparamètres possibles.

2.4.1 Hyperparamètres du modèle par arbre

Pour le modèle par arbre, les hyperparamètres utilisés sont maxdepth et minsplit.

L'hyperparamètre maxdepth contrôle la profondeur maximale de l'arbre. La profondeur maximale de l'arbre est en quelque sorte le nombre de divisions pouvant être réalisées dans le jeu de données. En termes du compromis biais-variance, une grande profondeur augmente la variance et réduit le biais. Nous avons considéré les valeurs 1, 3, 5 et 10.

L'hyperparamètre minsplit contrôle le nombre de points de données minimum dans un noeud pour qu'il soit divisé. Ainsi plus ce paramètre est grand, plus l'arbre aura des feuilles fournies. En termes du compromis biais-variance, un paramètre minsplit élevé donnera à l'arbre une variance faible, mais un biais élevé. Nous avons considéré les valeurs de 2, 5, 8, 10, 15 et 20.

2.4.2 Hyperparamètres du modèle de forêt aléatoire

Pour le modèle de forêt aléatoire, les hyperparamètres utilisés sont mtry, ntree et nodesize.

L'hyperparamètre mtry contrôle le nombre de variables explicatives sélectionnées aléatoirement lors de la création d'un noeud. En termes du compromis biais-variance, l'augmentation de mtry causera un biais plus petit, mais aura une variance plus grande. Nous avons considéré les valeurs 4, 8, 12 et 16.

L'hyperparamètre ntree contrôle le nombre d'arbres générés aléatoirement pour créer le classifieur. Il est important d'avoir un nombre suffisamment grand afin de faire baisser la variance et ainsi se rapprocher d'une valeur optimale au niveau du compromis biais-variance. Nous avons considéré les valeurs de 700, 1000 et 2000.

L'hyperparamètre nodesize contrôle le nombre minimal de données inclus dans le noeud terminal. Par défaut, le paramètre est 1 et cette valeur sur-ajuste les données. Nous avons considéré les valeurs de 100 et 500.

2.4.3 Hyperparamètres du modèle SVM

Pour le modèle SVM, les hyperparamètres utilisés sont cost et gamma (c et γ dans la notation utilisée dans le cours).

L'hyperparamètre cost contrôle la fréquence et la sévérité des violations par rapport à la marge. En termes du compromis biais-variance, l'augmentation de cost causera un biais plus grand, mais aura une variance plus petite. Nous avons considéré les valeurs 1, 10, 100 et 1000. L'hyperparamètre gamma est un paramètre de noyau. Nous avons considéré les valeurs 0.001, 0.01, 0.1, 1).

2.5 Ajustement des modèles complexes

2.5.1 Réseau de neurones

Finalement, nous avons fait la classification en entraînant un réseau de neurones profond. Les réseaux de neurones sont des modèles très flexibles, mais le choix d'une bonne architecture important. Ces choix sont souvent basés sur beaucoup d'expérience du scientifique des données. Étant donné que le but premier de ce projet n'est pas de construire un réseau de neurones et trouver l'architecture idéale en soi, nous avons choisi une architecture relativement simple. Nous avons donc choisi un réseau à propagation avant avec 2 couches cachées et une couche de sortie à 3 neurones (un pour chacune des catégories possibles) avec du *dropout*. Puisque c'est un problème de classification à multiples classes, nous avons choisi comme mesure de perte la *categorical-crossentropy* et comme métrique à optimiser la précision. La librairie utilisée est celle de `keras`.

Considérant la nature de ce type de méthode, il est important de mentionner le fait que nous n'avons pas structuré les données de la même façon pour ce type de modèle. En effet, celui-ci est mieux adapté pour trouver des interactions dans les variables et il n'est pas nécessaire de travailler en basse dimension. Ainsi, nous n'avons pas construit de variables précises (comme dans la méthode classique), nous avons plutôt donné en entrée les données brutes au modèle. De plus, les pays d'origine des joueurs sont utilisés (une fois dichotomisés, les pays d'origine des joueurs représentent 156 variables). Un total de 193 variables sont utilisées pour construire le modèle de réseau de neurones.

2.5.2 Modèles par ensemble

Les modèles par ensemble sont inspirés du proverbe de la sagesse des foules. Par exemple, s'il y a trois classifieurs indépendants qui ont une précision de 0.65 et qu'on utilise un modèle de vote majoritaire, la précision du voteur (de l'électeur) est

$$1 - \binom{3}{1} 0.65^1 \times 0.35^2 + \binom{3}{0} 0.65^0 \times 0.35^3 = 0.71825.$$

Bien sûr, les modèles ne sont pas indépendants, car ils utilisent les mêmes données (et parfois des hypothèses similaires). On s'attend quand même à une augmentation de la performance. Ce type de modèle est similaire au modèles AdaBoost et forêt aléatoire qui agrègent plusieurs apprenants faibles, mais dans notre cas ce sont des apprenants forts.

Le vote utilisé est pondéré par la précision du modèle, i.e. un modèle avec une meilleure performance en classification reçoit plus de poids. À partir du jeu de données test, on compare chaque combinaison des voteurs (chaque permutation

des 1, 2, . . ., 9 voteurs) et on sélectionne la combinaison qui permet de minimiser l'erreur sur le jeu de données de test.

La sortie du modèle est une prédiction basée sur une combinaison de modèles, alors beaucoup d'interprétabilité des résultats est perdue.

3 Résultats

Maintenant que les différents modèles à entraîner ainsi que leurs hyperparamètres respectifs sont définis, il ne reste plus qu'à évaluer leur performance et choisir le modèle final. Une fois le modèle final déterminé, il ne restera plus qu'à faire une brève analyse de celui-ci avec une dernière évaluation de sa performance sur les données de validation, qui n'ont toujours pas été vues par le modèle.

3.1 Évaluation des performances

Dans le but d'orienter notre choix de modèle, nous devons définir des mesures de performances qui permettront de comparer les modèles entre eux.

Dans les tâches de classification, les métriques que nous utilisons mettent l'emphase sur différents objectifs de classification. On pourrait vouloir par exemple mesurer la capacité du classifieur à prédire une bonne classe, la capacité de ne pas faire de fausses prédictions et/ou de ne jamais manquer de cas positifs. Ces mesures sont utilisées dans le cas binaire, où le classifier doit choisir soit une classe ou l'autre. Dans le problème actuel, l'utilisation de 3 classes impose qu'une transformation vers plusieurs problèmes binaires soit faite afin de pouvoir calculer ces métriques.

Afin de transformer le problème de classification multi-classe en classification binaire, chaque classe sera prise indépendamment et sera confrontée aux 2 autres agrégées ensemble (méthode tous contre un). Cela revient à faire 3 classifications binaires. Une stratégie d'agrégation doit être mise en place pour faire un sommaire des statistiques des 3 classifieurs.
Une stratégie serait d'attribuer un poids identique à chacune des métriques générées par les 3 classifieurs binaires. Ce sont les métriques de type macro qu'on utilise lorsqu'une classe minoritaire doit être considérée aussi importante que les autres classes.
Une autre stratégie serait d'attribuer des poids proportionnels à la quantité de données appartenant à chacune des classes aux métriques générées par les 3 classifieurs binaires. Ce sont les métriques de type micro qu'on utilise lorsqu'aucune classe n'a plus d'importance qu'une autre - ou qu'il n'y a pas d'incitatif à prédire une classe plus qu'une autre.

Dans le cadre de ce problème, nous utiliserons les métriques de type micro car il n'y a pas de classe ayant besoin d'une attention accrue. Formellement, pour chaque classe l, les métriques utilisées sont :

L'exactitude (Taux de bonnes détections)
L'exactitude est le pouvoir du classifieur à prédire la bonne classe pour une observation donnée. Lors d'une classification, l'optimisation de cette métrique maximise le taux de bonne détection.

$$\text{Exactitude} = \frac{\sum_{i=1}^{l} \frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i}}{l}$$

La précision
La précision est la capacité d'un classifieur à ne trouver que les données d'une classe en particulier. Lors d'une classification, l'optimisation de cette métrique minimise les faux positifs.

$$\text{Précision} = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} (tp_i + fp_i)}$$

Rappel (sensibilité)
Le rappel est la capacité du classifieur à bien trouver toutes les données d'une classe en particulier. Lors d'une classification, l'optimisation de cette métrique maximise les vrais positifs.

$$\text{Rappel} = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} (tp_i + fn_i)}$$

F1-score
Le F1-score est la moyenne harmonique de la précision et du rappel. Elle sert de mesure agrégée sur laquelle prendre une décision sur une classe.

$$\text{F1-score} = \frac{2 * \text{Précision} * \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

3.2 Choix du modèle

Dans le tableau 1, nous présenterons l'exactitude et le F1-score pour chacun des classifieurs pour le jeu de données complet et à dimensionnalité réduite. Le choix du modèle final sera fait en considérant ces deux statistiques.

Comme le F1-score combine les concepts de précision et de rappel, il a été choisi de ne pas présenter ces deux dernières. En effet, le but de notre classification est de donner la meilleure prévision, peu importe la classe en particulier, c'est-à-dire qu'il n'y a pas d'erreur plus grave qu'une autre. L'exactitude et le F1-score sont donc deux métriques appropriées pour l'évaluation de notre classifieur. Dans le cas où il y aurait une très grande différence entre le F1-score et l'exactitude, il

pourrait être intéressant de bien comprendre d'où vient cet écart en analysant les ratios intermédiaires de précision et de rappel.

TABLE 1 – Performances des différents modèles sur le jeu de données de test selon les différentes mesures de performance.

Modèles	Exactitude	F1-score	Exactitude avec PCA	F1-score avec PCA
Réseau de neurones	0.7843	0.8382	NA	NA
Ensemble	0.7614	0.8210	0.7528	0.8146
Multinomial	0.7614	0.8210	0.7499	0.8124
LDA	0.7571	0.8178	0.7481	0.8111
Forêt aléatoire	0.7553	0.8165	s/o	s/o
SVM Polynomial de degré 3	0.7553	0.8165	s/o	s/o
Arbre de décision	0.7465	0.8099	0.7398	0.8048
QDA	0.7224	0.7918	0.7357	0.8018
Bayes	0.7114	0.7836	0.7431	0.8074
SVM Gaussien	0.7018	0.7763	s/o	s/o

En analysant les résultats ci-dessus, on remarque que le modèle le plus performant est le réseau de neurones, suivi de très près par des modèles beaucoup plus simples. Seulement 2% séparent les deux métriques d'intérêt. On remarque aussi l'effet de nivellement causé par l'analyse en composantes principales sur les résultats. En gros, on voit le fait de réduire la dimension et de structurer les données de manière non supervisée avant l'entraînement rapproche les performances des différents modèles. Toutefois, les modèles qui étaient les meilleurs ont perdu de la performance, ce qui n'est pas nécessairement l'effet désiré. Ainsi, nous avons décidé de ne pas allé plus loin en termes de réduction de dimensionnalité, mais il aurait pu être possible de tester différentes avenues comme l'ACP par noyaux où des relations non linéaires peuvent être ajoutées. On rappelle aussi que l'équipe désirait trouver le meilleur modèle en considérant une complexité appropriée au problème. Ainsi, malgré que le réseau de neurone soit un peu plus performant, nous avons choisi le modèle multinomial, car il est beaucoup plus simple et nous avons une meilleure compréhension générale de ce modèle, ce qui rend nous plus confiant dans notre choix. De plus, il est possible que la différence notée dans les métriques de qualité soit due au hasard, supportant une fois de plus notre choix de modèle.

3.3 Présentation du modèle final

Le modèle que nous avons choisi pour prédire les surfaces sur lesquelles les matchs de tennis ont été joués est le modèle multinomial. En utilisant un jeu de données qui avait été gardé secret tout au long du projet, ce modèle à une exactitude de 63.1%. La distribution des erreurs qu'il fait est présentée dans la matrice de confusion ci-dessous.

Table 2 – Matrice de confusion pour le modèle final.

	Hard	Clay	Grass
Hard	1495	558	307
Clay	279	536	34
Grass	17	10	32

En guise de comparaison, une attribution au hasard donne une exactitude de 57.2%. On peut donc conclure qu'une certaine apprentissage a eu lieu, mais que le problème n'est pas simple à modéliser. Cela peut d'ailleurs se voir dans la matrice de corrélation 1 présentée plus tôt. Toutefois, étant donné que nous avons tranché en faveur d'un modèle nous donnant un certain niveau d'interprétabilité, il ne faut pas seulement prendre en compte la capacité du modèle à prédire. Un modèle plus complexe comme le réseau de neurones aurait probablement donné de meilleures performances en termes de prévisions, mais nous avons choisi d'opter pour un certain compromis. Une option envisageable aurait été de tenter d'obtenir davantage de données pour ainsi bâtir d'autres prédicteurs pertinents pour le modèle.

4 Conclusion

Au final, ce projet nous a aidé à comprendre l'utilisation de modèles d'apprentissage statistique dans un contexte de classification. Nous avons bâti un canal permettant de sélectionner le modèle le plus performant par validation croisée et recherche d'hyperparamètres optimaux par quadrillage, ce qui est une expérience importante pour un scientifique des données.

Nous sommes satisfaits des performances de notre classifieur et considérerons l'éventualité de peut-être donner des redevances à notre professeure, Anne-Sophie Charest, lorsque nous vendrons ce produit à des athlètes de haut niveau.

Références

[Allaire and Chollet, 2018] Allaire, J. and Chollet, F. (2018). *keras : R Interface to 'Keras'*. R package version 2.1.4.

[Liaw and Wiener, 2002] Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3) :18–22.

[Meyer et al., 2017] Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2017). *e1071 : Misc Functions of the Department of Statistics, Probability Theory Group (Formerly : E1071), TU Wien*. R package version 1.6-8.

[Therneau et al., 2017] Therneau, T., Atkinson, B., and Ripley, B. (2017). *rpart : Recursive Partitioning and Regression Trees*. R package version 4.1-11.

[Venables and Ripley, 2002a] Venables, W. N. and Ripley, B. D. (2002a). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

[Venables and Ripley, 2002b] Venables, W. N. and Ripley, B. D. (2002b). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.