Rapport Projet IFT-712

Frédéric Laberge 13 102 554
Dominic Desjardins-Côté 14 100 910

12 Décembre 2019

1 Introduction

Dans le cadre du cours IFT-712, nous devons prendre une base de données en ligne et appliquer les techniques d'apprentissage automatique vues en classe. Plus précisément, nous utilisons les six modèles suivants : un perceptron, un perceptron multicouche, une machine à vecteurs de support, un arbre de décision, un réseau logistique et un "bagging" sur chacun des modèles. De plus, nous devons nous assurer de faire une recherche efficace d'hyper paramètre selon chaque modèles. Dans ce rapport, tout d'abord, nous allons décrire la base de données utilisée. Ensuite, nous décrivons et évaluons chaque modèle. Pour finir, nous comparons les modèles entre eux.

2 Base de données

L'objectif est de classifier des données cartographiques d'une forêt et de trouver sa catégorie de couverture[1]. Nous avons sept classes possibles :

1. Épicéa / Sapin
2. Pin tordu
3. Pin ponderosa
4. Peuplier / Saule pleureur
5. Tremble
6. Sapin de Douglas
7. Krummholz

La base de données contient 15 120 observations. Chaque observation contient 54 variables cartographiques et une variable d'Id que nous enlevons. Nous séparons la base de données en 5000 données d'entraînement et 500 données

[1] https://www.kaggle.com/c/forest-cover-type-prediction

tests pour obtenir un ratio 90-10. Pour chaque modèle, nous l'entraînons sur l'ensemble d'entraînement. Après le modèle entraîné, nous évaluons leur performance. Sans faire une description complète des variables, une observation a 6 variables réelles positives et 48 variables de classes discrètes. Pour le traitement

de données, nous avons seulement ajouté un biais. De plus, nous n'avons pas normalisé les données. Car, nous obtenons une erreur quand nous entraînons les modèles avec ses données normalisées. Plus précisément, nous obtenons des valeurs infinies ou des divisions par zéro dans les données.

3 Méthodologie

Pour chaque modèle, nous avons décidé une liste d'hyperparamètres. Par la suite, nous appliquer une validation croisée (80-20) à 5 échantillons pour trouver le meilleur candidat. Ensuite, nous entraînons 10 modèles différents avec les mêmes hyperparamètres et les mêmes données d'entraînement pour vérifier la robustesse du modèle. C'est-à-dire qu'on veut vérifier si le biais et la variance sont faibles ou élévés. Donc, nous calculons sa précision moyenne, sa variance, la meilleure précision et sa matrice de confusion associée. De plus,

pour les 5 modèles, nous validons, avec la "learning curve", que plus il y a de données dans l'ensemble d'entraînement plus que le modèle devrait être performant. Dans notre cas, nous prenons 500 données de l'ensemble d'entraînement et nous appliquons une validation croisée. Nous l'entraînons avec les mêmes hyperparamètre que dans l'analyse. Nous calculons moyenne et variance de ce sous-ensemble. Ensuite, nous ajoutons 500 données de l'ensemble entraînement et nous itérons les mêmes étapes jusqu'à obtenir 5 000 données. Donc, ceci nous permet d'indiquer si la performance du modèle augmente avec la quantité de données. Par contre, nous avons fait une erreur durant l'entraînement des

modèles. Nous avons oublié d'afficher l'erreur d'entraînement. Mais, la courbe d'apprentissage nous permet de vérifier et de tirer des conclusions semblables.

4 Les modèles

Dans cette section, nous allons préciser les modèles choisis et afficher les résultats que nous avons obtenus.

4.1 Perceptron

4.1.1 Description

L'algorithme du Perceptron est un algorithme qui se base sur la classification des données à l'aide d'hyperplan. L'apprentissage du Perceptron se passe comme

suit. L'algorithme passe à travers les données d'entraînement et calcule l'erreur associée à une donnée (ou à un ensemble de données). Le vecteur w sera modifié seulement par les données mal classées afin de converger vers une solution où toutes les données sont bien classées.

Les avantages du perceptron sont qu'il ne nécessite pas un coefficient d'apprentissage à cause de la nature de sa fonction erreur (Perceptron criterion). Aussi, le modèle est mis à jour seulement lorsqu'il fait une erreur ce qui implique qu'il est plus rapide à converger que modèle qui se met à jour à chaque donnée. Les désavantages du Perceptron sont que le modèle a une faible capacité. C'est-à-dire que le modèle à de la difficulté à modéliser de complexes interactions entre les variables et les classes. De plus, pour que l'algorithme converge, il faut que les données (ou la transformation non linéaire des données $\phi(x)$) soientt linéairement séparables.

Le seul hyper-paramètre pour ce modèle est la constante de régularisation.

4.1.2 Résultats

1. **Recherche d'hyper-paramètre par validation croisé**

   Lorsque l'on observe la courbe de cross-validation pour le perceptron, on voit que le meilleur hyper-paramètre *alpha* de régularisation est 0.000001, la plus petite valeur testée. Cela sera la valeur de la constante de régularisation pour le reste de l'analyse du perceptron. On remarque ici que la précision semble être affectée par de grandes valeurs de régularisation. En effet, ceci peut s'expliquer par le fait que le modèle du perceptron semble avoir une trop faible capacité pour expliquer le problème. C'est pourquoi un petit paramètre de régularisation n'aura pas pour effet de sur-apprendre puisque de toute façon, il n'en est pas capable (trop faible capacité).

2. **Courbe d'apprentissage**

   Lorsque l'on observe la courbe d'apprentissage du modèle perceptron, on remarque que l'algorithme n'est pas stable (très forte variance) et semble osciller autour d'une précision de 0.35. On ne voit pas une tendance à la hausse ou à la base du score en fonction du nombre de données utilisé. Ceci laisse croire que le modèle semble "saturé" avec un petit nombre de données et ne peut pas s'améliorer avec plus de données dû à la trop faible capacité du modèle du perceptron pour notre problématique.

3. **Résultats sur données de Test**

   - Précision des 10 entraînements : [0.288 ; 0.3 ; 0.38 ; 0.394 ; 0.404 ; 0.352 ; 0.17 ; 0.416 ; 0.396 ; 0.35]
   - Moyenne de précision : 0.34500000000000003
   - Variance de précision : 0.07121937938510838

- Meilleure précision : 0.416

Nous obtenons la matrice de confusion suivante :

0	0	0	1	4	1	71
0	4	1	2	1	0	53
0	13	19	23	2	18	2
0	0	1	73	1	6	0
0	13	0	2	9	1	36
0	3	14	13	3	31	7
0	0	0	0	0	0	72

Nous remarquons que ce modèle n'est pas très bon. Bien qu'il soit plus facile de se tromper lorsque le modèle doit prédire 7 classes, environ 1 chance sur 3 de prédire la bonne classe n'est pas un modèle acceptable. En effet, il est clair que le modèle du perceptron n'avait pas une capacité suffisamment grande pour modéliser notre problème. On remarque aussi que la variance de la précision est élevée ce qui confirme que le modèle du perceptron a une faible stabilité puisque le modèle ne semble pas converger vers les mêmes résultats donc très sensibles à son initialisation. Finalement, on remarque à l'aide de la matrice de confusion que le modèle semble avoir de la difficulté à prédire des données de la classe 1,2 et 5 qu'il prédira tous à la classe 7. Seule la classe 4 et 7 semble avoir bien été prédite.

4.2 Perceptron multi couches

4.2.1 Description

Le Perceptron multi couches est un algorithme qui partage beaucoup de similarité avec le modèle du Perceptron. En effet, la différence fondamentale entre ces 2 modèles de classification est que le Perceptron multi-couches est un réseau de neurones profonds tandis que le Perceptron est un réseau de neurones sans couches cachées.

Les avantages du Perceptron peuvent être étendus au Perceptron multi couche puisque ces modèles utilisent la même fonction d'erreur. De plus, le perceptron multi couches a la capacité d'apprendre des modèles non linéaires complexes. Par contre, cette caractéristique vient avec un certain coût. Les couches cachées du perceptron engendrent un problème d'optimisation non convexe pour l'apprentissage. C'est-à-dire que l'algorithme peut converger vers différents minimum locaux n'ayant pas du tout la même précision. Ainsi, ce modèle est très sensible à l'initialisation des paramètres w. De plus, si l'on commence à utiliser un grand nombre de couches cachées et/ou neurones, l'apprentissage sera long sans toujours améliorer le modèle de manière significative.

De plus, le choix des hyper-paramètres pour ce modèle est difficile à cause de la complexité de ceux-ci. En effet, le nombre de couches cachées, le nombre de neurones par couche, le nombre d'itérations et la constante de régularisation sont tous des hyper-paramètres pour ce modèle. Dû à la complexité et temps d'apprentissage du perceptron multi couches, nous avons décidé de tester pour 4 combinaisons d'hyper-paramètres liés aux couches cachées.

1. 1 couche cachée avec 200 neurones (avec biais)
2. 1 couche cachée avec 600 neurones (avec biais)
3. 3 couches cachées avec 200, 200 et 200 neurones respectivement (avec biais)
4. 3 couches cachées avec 200, 400 et 600 neurones respectivement (avec biais)

L'idée ici était de tenter d'expliquer le comportement de l'algorithme pour notre modèle en augmentant le nombre de couches cachées, le nombre de neurones par couche et la progression du nombre de neurones par couche.

4.2.2 Résultats

1. **Recherche d'hyper-paramètre par validation croisé** Pour les 4 différents modèles du perceptron multicouches, on a fait une cross-validation sur l'hyper-paramètre de régularisation. Pour le modèle 1 (200,), on a obtenu une valeur la meilleure constante de régularisation est 0.01. Pour le modèle 2 (600,), celle-ci sera de 0.1 tandis que pour le modèle 3 (200,200,200) et modèle 4 (200,400,600), elle sera de 10.0. On peut expliquer cette différence par le fait que les 4 modèles du perceptron multi couches ont une capacité différente. En effet, le modèle 4 peut expliquer des modèles beaucoup plus complexes que le modèle 1. En règle général si l'on augmente le nombre de couches cachées ou le nombre de neurones dans un réseau de neurones, on augmente la capacité du modèle. Cette caractéristique explique pourquoi les modèles avec une plus grande capacité nécessitent des constantes de régularisation plus grande puisque ces modèles ont la "capacité" de sur-apprendre.

   On remarque aussi par la même logique que le modèle 3 semble avoir une plus grande capacité que le modèle 2 malgré le même nombre de neurones au total. Ce résultat laisse croire que le perceptron multi couches gagne plus rapidement en capacité en augmentant sa profondeur (nombre de couches cachées) quant augmentant sa largeur (nombre de neurones).

2. **Courbe d'apprentissage**

   Lorsque l'on observe les 4 différentes courbes d'apprentissage, on peut voir une nette distinction entre les modèles à une couche cachée vs ceux à 3 couches cachées. En effet, les modèles 1 et 2 semblent converger vers une précision de 0.55 tandis que les modèles 3 et 4 semblent converger vers

une précision de 0.7. Ces résultats confirment l'hypothèse que les modèles 3 et 4 ont une plus forte capacité et du même coup ont la possibilité de modéliser la complexité de notre problème.

Le deuxième élément qu'on peut observer est la vitesse de convergence des différents modèles. Nous avons testé jusqu'à 12 000 données et il n'est pas trivial de croire que certaines de ces courbes pourraient bien continuer d'augmenter avec plus de données, surtout pour le modèle 3. Le modèle 1 et 2 donne leurs meilleurs résultats autour de 8000 données. Pour le modèle 4, la précision converge autour de 7000 données ce qui est très étonnant. Ces résultats me font croire que le modèle gagne en capacité par son nombre de couches cachées et gagne en temps d'apprentissage par sa progression de neurones par couches cachées. Cette progression du nombre de neurones le long des différentes couches semble accélérer ou améliorer l'apprentissage du modèle.

3. Résultats sur données de Test

Modèle 1 (200,)

- Précision des 10 entraînements : [0.414 ; 0.45 ; 0.472 ; 0.466 ; 0.436 ; 0.572 ; 0.458 ; 0.55 ; 0.502 ; 0.458]
- Moyenne de précision : 0.47780000000000006
- Variance de précision : 0.04713767071037771
- Meilleure précision : 0.572

Nous obtenons la matrice de confusion suivante :

65	5	0	0	0	5	2
47	4	0	1	0	7	2
5	5	13	22	0	32	0
0	0	0	75	0	6	0
27	9	0	7	14	4	0
7	4	4	9	1	46	0
55	4	0	0	1	0	12

Modèle 2 (600,)

- Précision des 10 entraînements : [0.504 ; 0.434 ; 0.43 ; 0.412 ; 0.512 ; 0.42 ; 0.538 ; 0.584 ; 0.394 ; 0.486]
- Moyenne de précision : 0.47139999999999993
- Variance de précision : 0.059424237479331614
- Meilleure précision : 0.584

Nous obtenons la matrice de confusion suivante :

5	21	15	0	0	0	36
1	32	16	1	0	1	10
0	2	69	6	0	0	0
0	0	21	60	0	0	0
0	10	41	2	1	4	3
0	0	58	6	0	7	0
0	1	2	0	0	0	69

Modèle 3 (200, 200, 200)

- Précision des 10 entraînements : [0.474 ; 0.546 ; 0.324 ; 0.402 ; 0.502 ; 0.566 ; 0.594 ; 0.5 ; 0.514 ; 0.58]
- Moyenne de précision : 0.5002
- Variance de précision : 0.07947553082553144
- Meilleure précision : 0.594

Nous obtenons la matrice de confusion suivante :

12	37	5	0	3	0	20
5	35	8	0	10	1	2
0	1	64	7	2	3	0
0	0	7	74	0	0	0
0	11	9	0	41	0	0
0	2	57	6	0	6	0
1	13	0	0	0	0	58

Modèle 4 (200, 400, 600)

- Précision des 10 entraînements : [0.71 ; 0.67 ; 0.69 ; 0.726 ; 0.614 ; 0.722 ; 0.678 ; 0.728 ; 0.662 ; 0.644]
- Moyenne de précision : 0.6844
- Variance de précision : 0.03615300817359461
- Meilleure précision : 0.728

Nous obtenons la matrice de confusion suivante :

21	41	2	1	4	0	8
2	38	10	0	4	7	0
0	0	59	10	0	8	0
0	0	1	78	0	2	0
0	3	10	0	41	7	0
0	1	18	15	0	37	0
12	12	0	0	0	0	48

Le premier élément à mentionner est qu'on peut observer que la quantité de données d'entrainement utilisé pour entrainer ces modèles n'était définitivement pas assez grande. Si nous nous fions aux courbes d'apprentissages une quantité de 12000 données aurait été appropriée. On remarque que le seul modèle qui donne de bons résultats est le modèle 4 puisqu'on sait qu'avec 5000 données d'entrainement, il peut donner de bons résultats. De plus, on peut voir que le modèle 4 est aussi le plus stable malgré qu'il semble le plus complexe. Le modèle 3 est celui avec la plus grande variance de précision et ainsi le moins stable. Lorsqu'on observe la matrice de confusion, on peut voir que les différents modèles semblent converger vers différents minimum locaux en gardant quand même certaines caractéristiques. La classe 1 et 2 sont souvent mélangés et il y a souvent une classe qui est sur-représentée et une qui est sous-représentée.

4.3 Arbre de décision

4.3.1 Description

Le troisième modèle est l'arbre de décision. Ce modèle construit un arbre binaire où que les noeuds contiennent une inégalité de type avant ou derrière un hyperplan. Si l'inégalité est satisfaite, nous prenons une branche. Sinon, nous choisissons l'autre branche. Aux feuilles de cet arbre contiennent une classe que nous attribuons à la donnée. Donc, pour classifier une donnée, nous la partons à la racine. Ensuite, nous parcourons l'arbre selon les inégalités dans les noeuds. Pour finir, la donnée arrive à une feuille et nous l'assignons à cette classe. Dans la libraire de sklearn, l'arbre utilise des stumps, c'est-à-dire que les inégalités se font sur une variable à la fois.

4.3.2 Résultats

1. **Recherche d'hyper-paramètre par validation croisé**

   Les hyper paramètres choisis sont le critère de gini ou le critère entropy et la profondeur maximale de l'arbre allant de 2 à 40. Si nous fixons la profondeur de l'arbre, alors nous pouvons empêcher que l'arbre de décision fasse du sur-apprentissage. Nous obtenons la courbe de validation pour le critère de Gini et la courbe de validation de Entropy (7 et 8). Dans notre entraînement, nous trouvons que les meilleurs hyperparamètres sont avec le critère de gini et une profondeur maximale de 15 que nous conserverons pour le reste de l'analyse.

2. **Courbe d'apprentissage**

   Lorsque nous observons la courbe d'apprentissage du modèle de l'arbre de décision (16), nous remarquons que la variance est faible en général. De plus, l'écart entre le score de l'entraînement et le score de test diminue. Si la tendance se maintient, avec encore plus de données, nous nous rapprochons du cas qu'il n'y a pas de sur-apprentissage ni se sous-apprentissage tout en gardant une précision supérieure à 80%.

3. **Résultats sur les données de Test**

   - Précision des 10 entraînements : [0.71 ; 0.7 ; 0.716 ; 0.708 ; 0.708 ; 0.71 ; 0.696 ; 0.704 ; 0.702 ; 0.708]
   - Moyenne de précision : 0.7062
   - Variance de précision : 0.0055
   - Meilleure précision : 0.716

   Nous obtenons la matrice de confusion suivante :

46	17	0	0	3	0	11
15	33	2	0	8	3	0
0	2	45	9	1	20	0
0	0	5	72	0	4	0
1	9	0	0	50	1	0
0	0	15	8	1	47	0
5	2	0	0	0	0	65

   Avec ce modèle, nous nous trompons, en moyenne, que 3 fois sur 10. De plus, la variance est faible. Le modèle est loin d'être parfait, mais c'est nettement meilleur que du bruit.

4.4 Machine à vecteur de supports

Le quatrième modèle est la machine à vecteur de supports. Nous avons utilisé un noyau rbf qui à la forme suivante : $\exp(-\gamma\|x-x'\|^2) * \gamma$ avec γ positif. Nous avons pris le choix de γ avec une valeur 'auto' dans Sklearn. Car, en fessant des tests durant le développement du code, la valeur de γ affectait peu le résultat final.

4.4.1 Résultats

1. **Recherche d'hyper-paramètre par validation croisé**

   L'hyperparamètre choisi est C qui est le paramètre de régularisation. Nous faisons une recherche entre 10^{-3} à 10^3 avec une échelle logarithmique. Après la validation croisée, la valeur de l'hyperparamètre choisie est 1. Nous obtenons la courbe de validation (6).

2. **Courbe d'apprentissage**

   Nous remarquons que la courbe d'apprentissage du SVM avec un noyau rbf fait toujours du surapprentissage même si la quantité de donnée augemente. Nous remarquons déjà que ce modèle va donner de mauvais résultats.

3. **Résultats sur les données de Test**

   - Précision des 10 entraînements : [0.144 ; 0.144 ; 0.144 ; 0.144 ; 0.144 ; 0.144 ; 0.144 ; 0.144 ; 0.144 ; 0.144]

- Moyenne de précision : 0.144
- Variance de précision : 0
- Meilleure précision : 0.144

Nous obtenons une matrice de confusion suivante pour un entraînement :

0	0	0	0	0	0	77
0	0	0	0	0	0	61
0	0	0	0	0	0	77
0	0	0	0	0	0	81
0	0	0	0	0	0	61
0	0	0	0	0	0	71
0	0	0	0	0	0	72

Nous remarquons que ce modèle est très mauvais. Nous avons que $\frac{1}{7} \approx 0.143$. Donc, ce modèle est équivalent à du bruit. De plus, ce modèle a eu du sur-apprentissage et affecté toutes les autres données à la septième classe. Un modèle à éviter.

Nous avons aussi essayé avec un noyau polynomial. Il avait de meilleurs résultats durant les tests que nous avons faits. Par contre, durant l'entraînement, il y a eu une erreur et nous n'avons pas eu assez temps pour relancer l'entraînement. Cet entraînement était très long.

4.5 Réseau logistique

4.5.1 Description

La classification par réseaux logistique se base sur la recherche des probabilités conditionnelles de chaque classe. L'apprentissage du réseau logistique se fait par descente de gradient stochastique comme dans le cas du Perceptron où l'on tente de minimiser une différente fonction de coût (entropie croisée). Dans ce cas, toutes les données (bien ou mal classée) vont influencer le gradient lors de l'apprentissage. Dans le cas d'une classification à 7 classes, on utilise une fonction activation softmax afin de modéliser une probabilité de classe.

Le réseau logistique est un algorithme plus stable que le perceptron. C'est-à-dire que l'algorithme aura tendance à converger plus souvent au même endroit ou que sa variance est faible. De plus, le réseau logistique fonctionne mieux pour des données non séparables qui vient au coût d'un plus grand temps d'apprentissage comparé au perceptron. Un peu comme dans le cas du Perceptron, le réseau logistique est un algorithme de classification linéaire et ainsi est limité en capacité pour modélisé de complexe problème comparé aux réseaux de neurones profonds (comme le perceptron multi couches).

Les hyper-paramètres de ce modèle sont le type de pénalité choisi pour la régularisation (L1 ou L2) ainsi que la constante de régularisation. Nous avons choisi une régularisation L2 et nous avons fait une cross-validation afin de trouver le meilleur paramètre de régularisation C.

4.5.2 Résultats

1. **Recherche d'hyper-paramètre par validation croisé**

   Lorsque l'on observe la courbe de cross-validation pour le réseau logistique, on voit que le meilleur hyper-paramètre C est d'environ 2154. Cela sera la valeur de C utilisé pour le reste de l'analyse du réseau logistique. On remarque que la précision semble être affectée pour de petite valeur de C. Cela équivaut à une forte regularisation ce qui est bien sûr attendu. Par contre, on remarque à partir des valeurs de C autour de 1, la courbe semble rester constante. Ceci peut s'expliquer par le fait que le modèle n'a pas une assez forte capacité ce qui fait que réduire la régularisation (en augmentant la valeur de C) n'aura pas l'effet de sur apprendre sur les données d'entrainement et ainsi diminuer la précision des données valides.

2. **Courbe d'apprentissage**

   Lorsque l'on observe la courbe d'apprentissage du modèle en fonction du nombre de données utilisées, on remarque l'algorithme semble données de bons résultats à partir d'environ 6000 données et ne semble pas s'améliorer de manière significative en augmentant la quantité de données. Ceci peut encore une fois s'expliquer par le fait qu'avec 6000 données, on a "saturé" le modèle et on ne peut pas s'améliorer avec plus de données (trop faible capacité).

   De plus, on peut observer qu'avec un nombre faible de données, on semble faire du sur-apprentissage puisque la précision d'entrainement est beaucoup plus forte que la précision sur les données valides. Ceci peut s'expliquer par le fait qu'on a choisi une valeur de C relativement élevé et ainsi une faible régularisation qui peut engendrer du sur-apprentissage si l'on n'a pas assez de données.

3. **Résultats sur données de Test**

   - Précision des 10 entraînements : [0.656 ; 0.656 ; 0.656 ; 0.656 ; 0.656 ; 0.656 ; 0.656 ; 0.656 ; 0.656 ; 0.656]
   - Moyenne de précision : 0.6559999999999999
   - Variance de précision : 1.1102230246251565e-16
   - Meilleure précision : 0.656

   Nous obtenons la matrice de confusion suivante :

43	11	0	0	8	1	14
13	29	0	0	14	5	0
0	0	36	12	6	23	0
0	0	5	70	0	6	0
0	10	4	0	45	2	0
0	1	14	6	5	45	0
9	2	0	0	1	0	60

À première vue, on peut dire que le modèle du réseau logistique semble donner des résultats satisfaisants. En effet, malgré qu'une précision moyenne de 0.656 peut sembler basse. Dans le contexte complexe de notre projet (55 variables, 7 classes), c'est une précision acceptable. De plus, on remarque que le modèle semble plutôt stable et converge vers des modèles relativement similaires (variance de précision pratiquement nulle). On remarque à l'aide de la matrice de confusion que le modèle a eu de la difficulté à différentier la classe 1 vs 2 ainsi que la classe 3 vs 6.

4.6 Bagging

4.6.1 Description

Le Bagging est une technique de combinaison de modèles de classification visant à améliorer la performance d'un modèle à l'aide d'un ensemble de modèles. Le bagging entraine plusieurs modèles ayant les mêmes hyper-paramètres sur les mêmes données choisies différemment (voir Bootstraping) fait un vote majoritaire de ces différents modèles entrainés comme prédiction du modèle bagging. Ce processus a pour but d'améliorer la performance des modèles ayant une forte capacité en réduisant leur variance.

Les hyper-paramètres de ce modèle sont le modèle utilisé à la base (celui qui sera entrainé plusieurs fois) et le nombre de modèles à entrainer. En plus, des hyper-paramètres du modèle à la base utilisé. Ici, on va utiliser les meilleurs hyper-paramètres des modèles précédemment trouvés. On va tester le modèle pour tous les modèles étudiés précédemment ainsi que 10 et 25 qu'on nombre de modèle à entraîner.

4.6.2 Résultats

Les résultats du bagging ne sont pas interessanst. En effet, pour chaque modèle on observe une variance de précision des différents modèles d'autour 0 pour tous les modèles avec les deux nombres de modèles différents (10 et 25). On pourrait croire que le choix des données d'entrainement n'a pas été bien réalisé puisque l'on semble entrainer exactement le même modèle et ainsi le bagging est complètement absurde à analyser.

5 Tableau récapitulatif

Un tableau récapitulatif regroupant la précision moyenne, la variance et la meilleure précision de chaque modèle sur l'ensemble de tests.

	Moyenne	Variance	Meilleur Score	Rang
Perceptron	0.345	0.07	0.416	7
MLP (200,)	0.478	0.047	0.572	6
MLP (600,)	0.471	0.059	0.584	5
MLP(200, 200, 200)	0.500	0.079	0.594	4
MLP(200, 400, 600)	0.684	0.036	0.728	1
Arbre de décision	0.706	0.006	0.716	2
SVM avec RBF	0.144	0.000	0.144	8
Réseau logistique	0.656	0.000	0.656	3

6 Conclusion

En conclusion, on peut dire que le meilleur modèle dans le contexte de cette problématique est le perceptron multi couches (200, 400, 600). Il a la meilleure précision sur les données de test et est un des modèles les plus stables malgré sa forte capacité.

Toutefois, de nombreux choix auraient été faits différemment a posteriori de notre analyse. En effet, utiliser 5000 données pour entrainer un modèle peut sembler suffisant pour un modèle de régression logistique, il n'en est pas de même pour les modèles plus complexes comme le perceptron multi-couches. Les résultats obtenus sont influencés par le choix du nombre donné utilisé pour entrainer les différents modèles. Il aurait plus intelligent de prendre plusieurs quantités de données différents afin de voir le "potentiel" de chaque modèle.

Finalement, il aurait été plus intéressant d'implémenter un processus de "boosting". En effet, à la suite de notre analyse, il semble évident que nos modèles n'avaient pas assez de capacité pour expliquer la complexe problématique que nous avions. Le boosting aurait aidé nos modèles à augmenter leur capacité et ainsi probablement augmenter leur performance et pour les modèles à forte capacité, cette forte capacité ne semblait pas être le problème pour ces modèles.

A Les figures

Pour les courbes de validation croisées, la courbe bleue représente le score moyen et la bande bleue représente la variance du score sur l'ensemble de validation. La courbe orange représente le score moyen et la bande orange représente la variance du score sur l'ensemble d'entraînement. Pour les courbes d'apprentissages, la courbe rouge représente le score moyen et la bande rouge représente la variance du score sur l'ensemble d'entraînement. La courbe verte représente le score moyen et la bande verte représente la variance du score sur l'ensemble de tests.

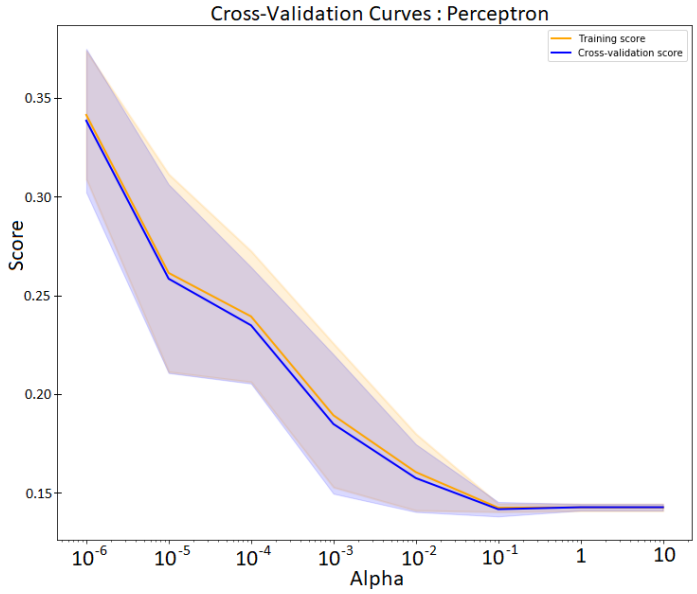

Figure 1: Courbes de validation croisée du Perceptron.

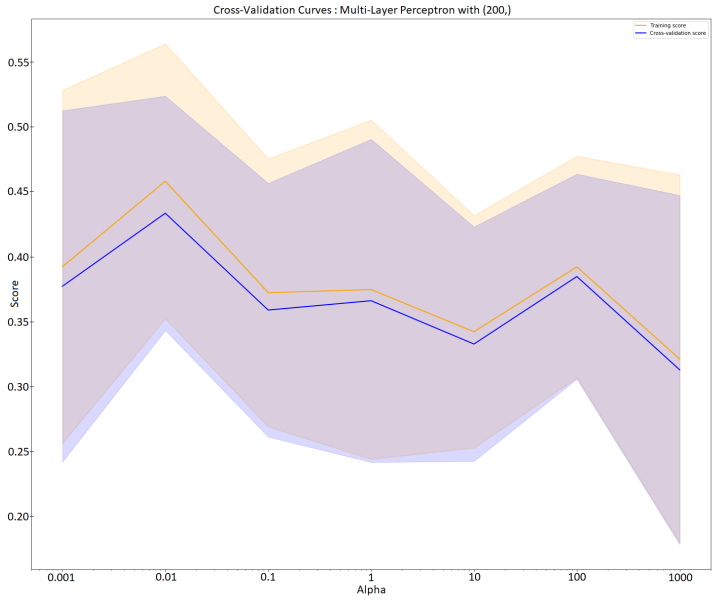

Figure 2: Courbes de validation croisée du Percerptron multi couches (200).

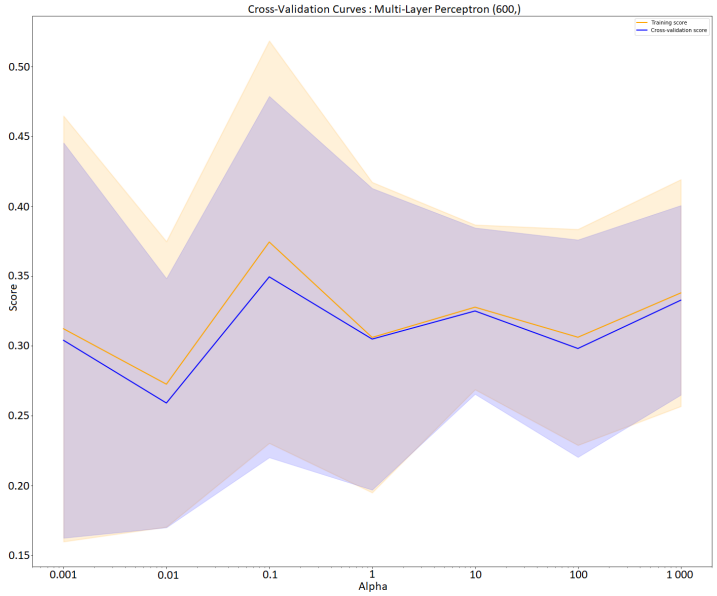

Figure 3: Courbes de validation croisée du Percerptron multi couches (600).

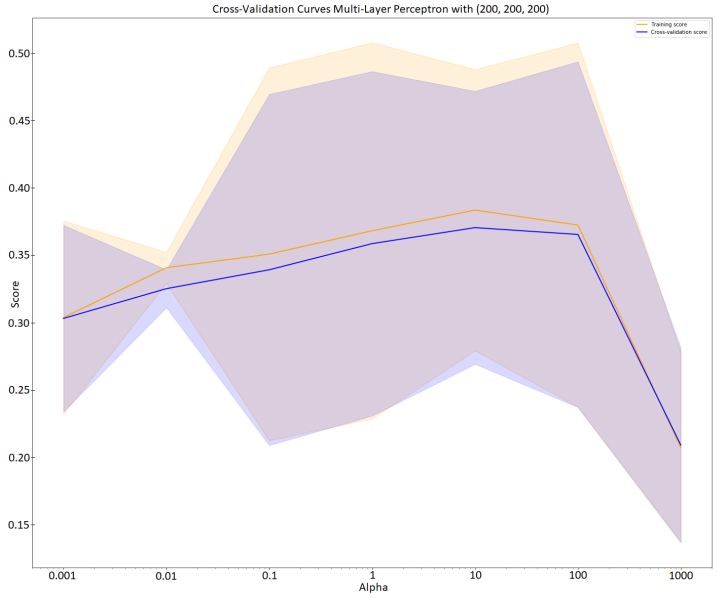

Figure 4: Courbes de validation croisée du Percerptron multi couches (200, 200, 200).

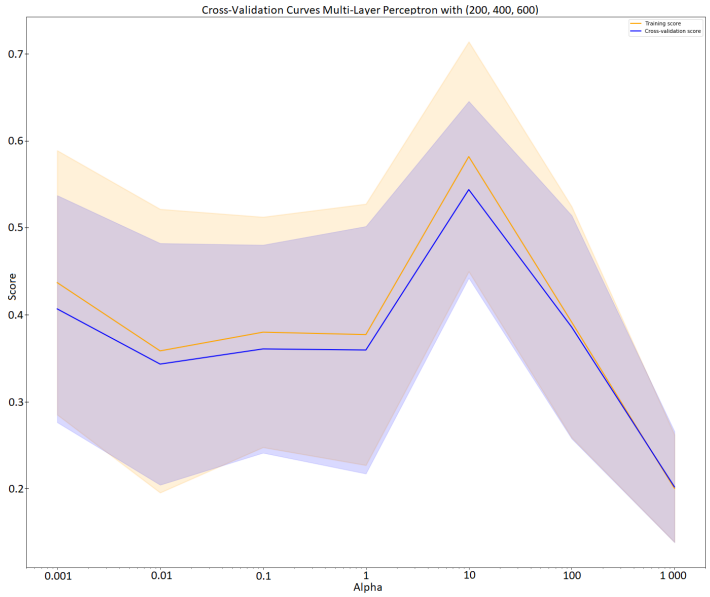

Figure 5: Courbes de validation croisée du Percerptron multi couches (200, 400, 600).

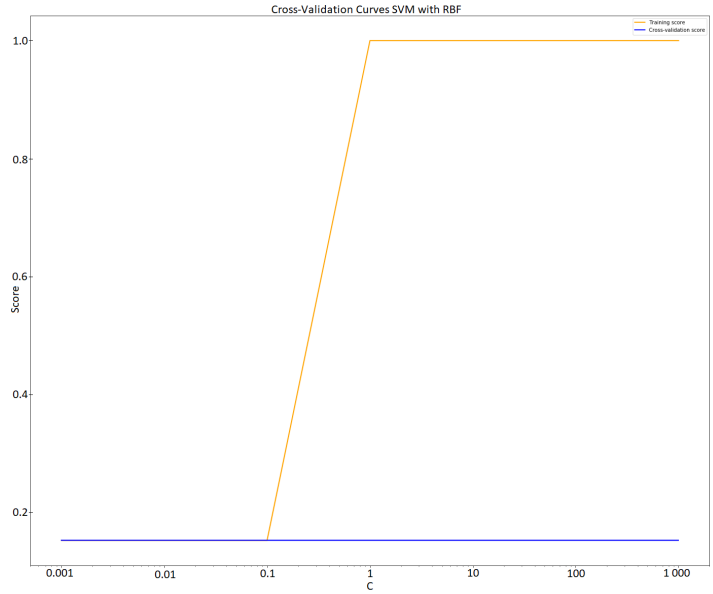

Figure 6: Courbes de validation croisée du SVM avec un noyau RBF.

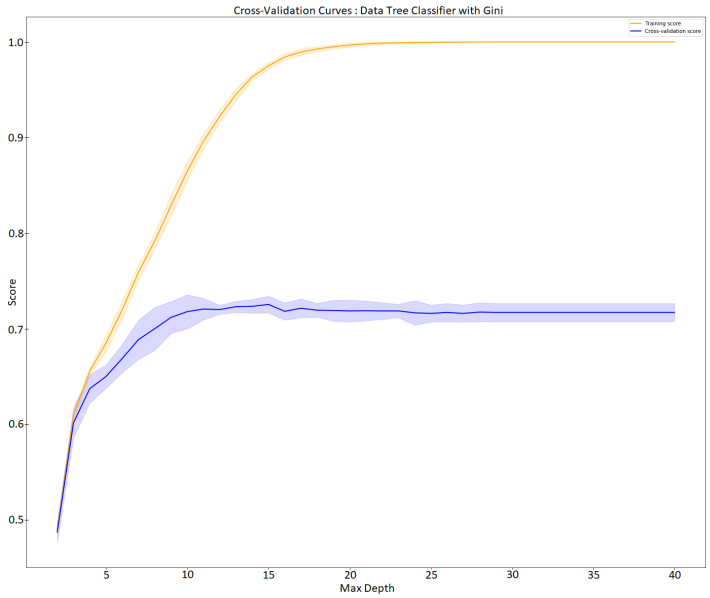

Figure 7: Courbes de validation croisée de l'arbre de décision avec un critère gini.

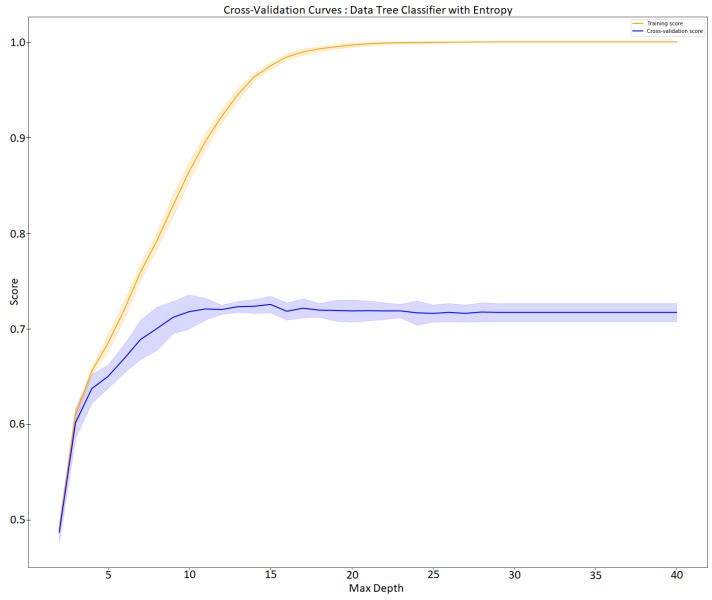

Figure 8: Courbes de validation croisée de l'arbre de décision avec un critère entropy.

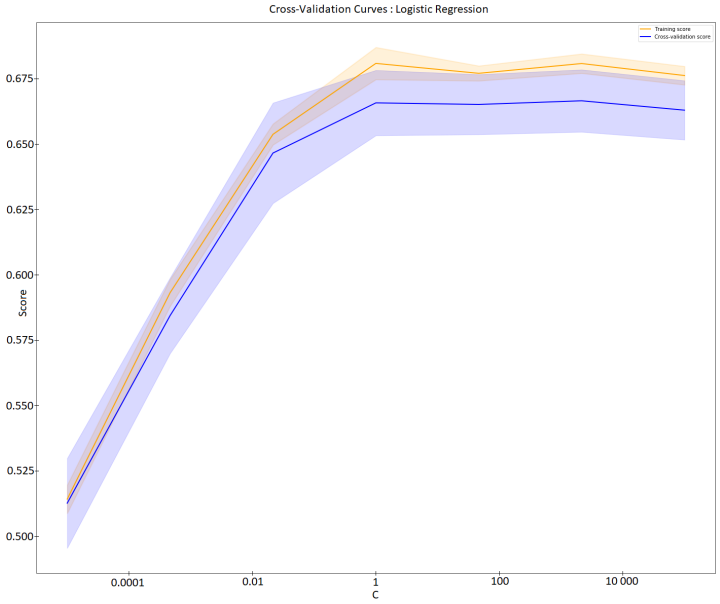

Figure 9: Courbes de validation croisée du réseau logistique.

Figure 10: Courbes d'apprentissages du Perceptron.

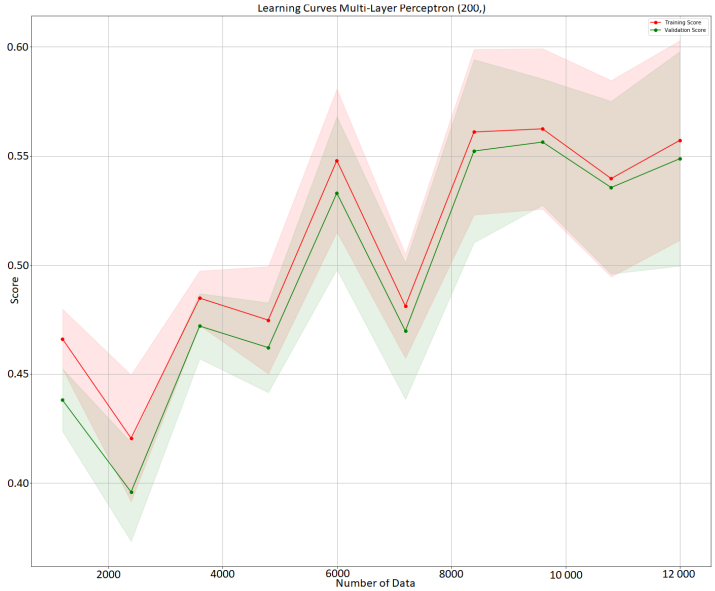

Figure 11: Courbes d'apprentissages du Perceptron multi couches (200).

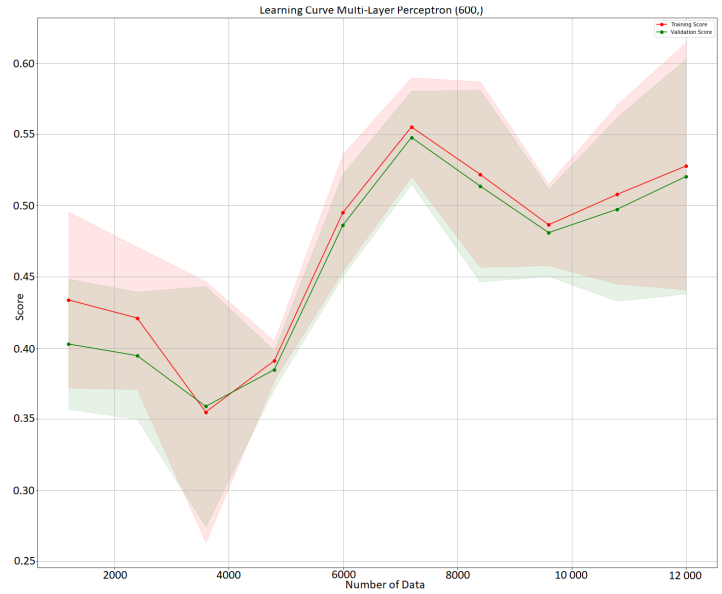

Figure 12: Courbes d'apprentissages du Perceptron multi couches (600).

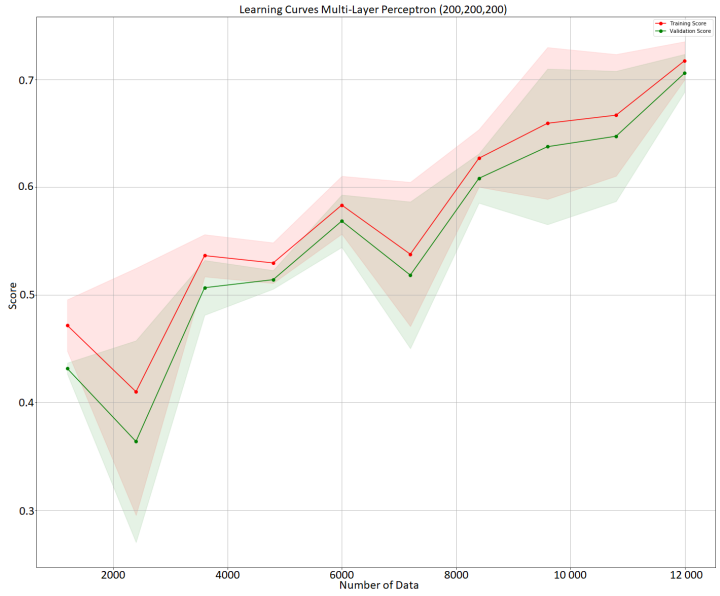

Figure 13: Courbes d'apprentissages du Perceptron multi couches (200, 200, 200).

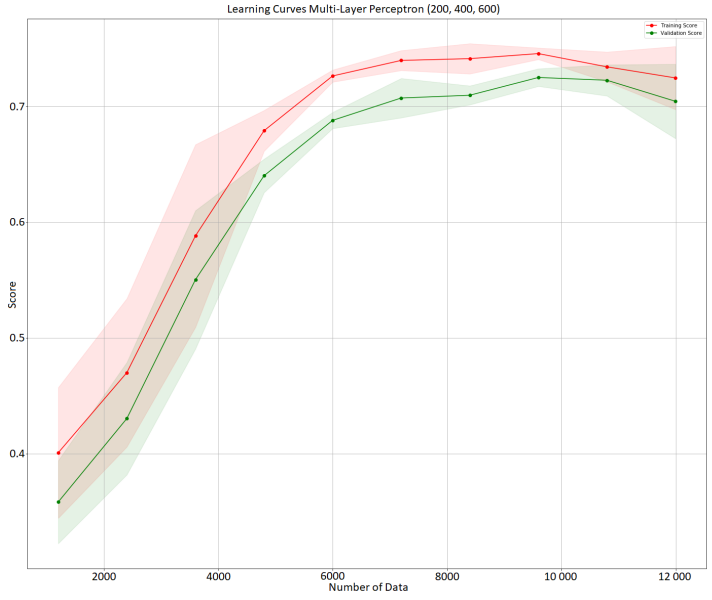

Figure 14: Courbes d'apprentissages du Perceptron multi couches (200, 400, 600).

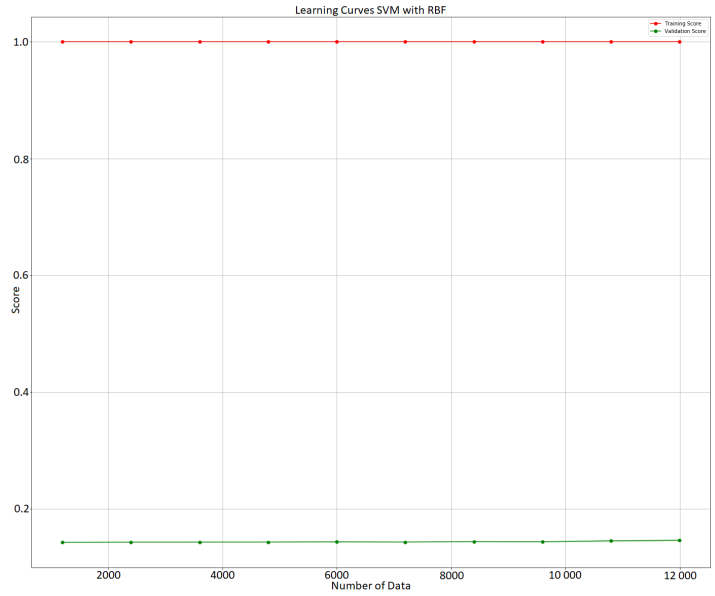

Figure 15: Courbes d'apprentissages du SVM avec un noyau RBF.

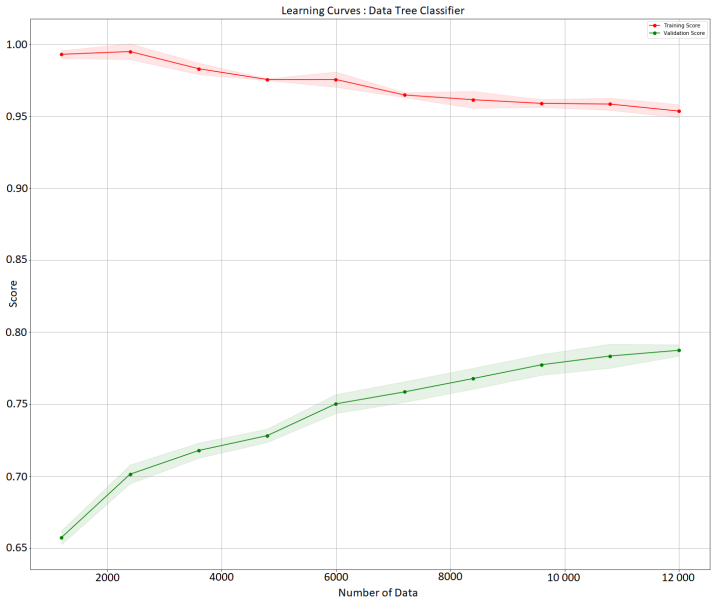

Figure 16: Courbes d'apprentissages de l'arbre de décision.

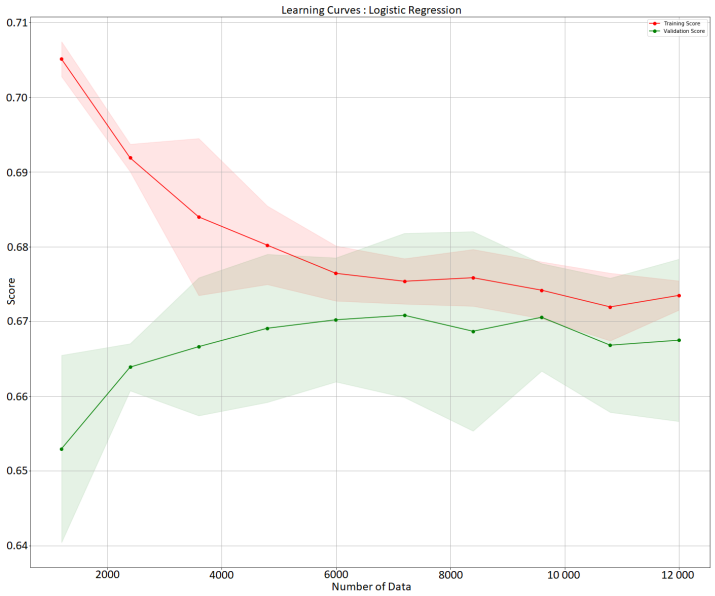

Figure 17: Courbes d'apprentissages du réseau logistique.