SY19 - TP03
Réseaux de neurones : Optimisation de l'architecture

Alice Ngwembou - Antoine Hars

December 12, 2013

Introduction

Le but de ce TP est d'étudier les propriétés des réseaux de neurones et leur capacités de classification. Nous étudierons notamment l'influence des différents paramètres (nombre de neurones dans la couche cachée, taille, régularisation ...) sur leur efficacité.

Exercice I

Dans cette première partie, nous considérons un réseau de neurones à 2 entrées + 1 biais, 1 neurone caché + 1 biais et un neurone de sortie. Les poids de notre réseau sont les suivants :

Couche d'entrée vers couche cachée	$v_0 = -1.5$	$v_1 = 1$	$v_2 = 1$	
Couche cachée vers couche de sortie	$w_0 = -0.5$	$w_1 = 1$	$w_2 = 1$	$w_3 = -2$

Chaque neurone étant un neurone de McCulloch-Pitts, leur fonction d'activation φ vérifie :

$$\varphi(u) = \begin{cases} 1 & \text{si} \quad u \geq 0 \\ 0 & \text{si} \quad u < 0 \end{cases}$$

où u correspond à la somme somme des grandeurs reçues en entrées par le neurone, pondérées par les poids des liaisons.

D'après les paramètres de notre réseau on obtient la sortie S_1 du neurone caché, puis d, la sortie de notre réseau qui dépend de S_1 :

$$S_1 = \varphi(\underbrace{v_0 + v_1 x_1 + v_2 x_2}_{u_1}) = \begin{cases} 1 & \text{si} \quad u_1 \geq 0 \\ 0 & \text{si} \quad u_1 < 0 \end{cases}$$

$$d = \varphi(\underbrace{w_0 + w_1 x_1 + w_2 x_2 + w_3 S_1}_{u_2}) = \begin{cases} 1 & \text{si} \quad u_2 \geq 0 \\ 0 & \text{si} \quad u_2 < 0 \end{cases}$$

Pour calculer les frontières de décision de notre réseau, on résout les équations :

$$\begin{cases} u_1 = 0 & \iff v_0 + v_1 x_1 + v_2 x_2 = 0 & \iff -1.5 + x_1 + x_2 = 0 & \iff x_2 = 1.5 - x_1 \\ u_2 = 0 & \iff w_0 + w_1 x_1 + w_2 x_2 + w_3 S_1 = 0 & \iff -0.5 + x_1 + x_2 - 2S_1 = 0 & \end{cases}$$

Pour résoudre u_2, sachant que S_1 ne peut avoir que deux valeurs, à savoir 0 ou 1, que si $u_1 < 0$ alors $S_1 = 0$ et que si $u_1 \geq 0$ alors $S_1 = 1$, on obtient deux équations pour u_2 :

$$\begin{cases} \text{si} \quad S_1 = 0 \quad \text{alors} \quad x_2 = 0.5 - x_1 \\ \text{si} \quad S_1 = 1 \quad \text{alors} \quad x_2 = 2.5 - x_1 \end{cases}$$

On a donc 3 frontières de décisions. Pour tous les points du plan qui sont dans la partie inférieure à la droite d'équation $x_2 = 1.5 - x_1$, c'est la droite d'équation $x_2 = 0.5 - x_1$ qui détermine la classe à laquelle ils appartiennent : si ils sont au dessus de cette droite, ils sont de la classe 1, sinon de la classe 0. De même, pour tous les points du plan qui sont dans la partie supérieure à la droite d'équation $x_2 = 1.5 - x_1$,

c'est la droite d'équation $x_2 = 2.5 - x_1$ qui détermine la classe à laquelle ils appartiennent : si ils sont au dessus de cette droite, ils sont de la classe 1, sinon de la classe 0.

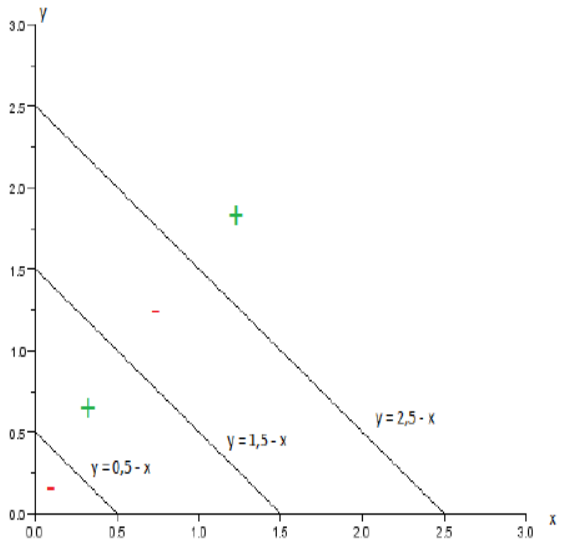


Figure 1: Représentation des frontières de décisions du réseau de neurones

Le but de notre démarche est de montrer que ce réseau peut répondre au problème XOR, dont la table de vérité est la suivante :

$$XOR \begin{cases} x_1 & x_2 & classe \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{cases}$$

Figure 2: Table de vérité du problème XOR

On teste les sorties du réseau pour toutes les entrées possible du problème XOR :

x_1	x_2	S_1	sortie d
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 3: Sorties du réseau pour les quatres types d'entrées du problème XOR

Si on compare ce tableau à la table de vérité du problème XOR, on retrouve bien les mêmes sorties pour deux entrées données, par conséquent notre réseau de neurone permet de répondre au problème XOR.

Exercice 2

Travail Préliminaire

Question 1 :

1. Quelles sont les 4 lois utilisées pour générer les observations de l'ensemble d'apprentissage appartenant à 4 places ?

Nous avons 4 lois Normales $\mathcal{N}(\mu_1, \Sigma_1)$, $\mathcal{N}(\mu_2, \Sigma_2)$, $\mathcal{N}(\mu_3, \Sigma_3)$ et $\mathcal{N}(\mu_4, \Sigma_4)$ avec :

$$\mu_1 = (4,\ 6) \quad \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \quad \mu_2 = (6,\ 1) \quad \Sigma_2 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\mu_3 = (\text{-}4,\ 4) \quad \Sigma_3 = \begin{pmatrix} 1.5 & 0 \\ 0 & 2 \end{pmatrix} \quad \mu_4 = (0,\ 0) \quad \Sigma_4 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

2. Donner la règle de Bayes qui permet de répondre à un problème de classification à c classes.

La formule de Bayes donne la probabilité, dîte probabilité *a posteriori*, pour qu'un individu décrit pas le descripteur x appartienne à la classe k : $P(c_k|x) = \frac{f_k(x)}{\sum f_i(x)} = \frac{Pr_k * f_k(x)}{\sum Pr_i * f_i(x)}$, où c est le nombre de classes et Pr_k est la probabilité *a priori* que l'individu appartienne à la classe k.

3. Développer la règle de Bayes pour arriver à la solution donnée dans le code ci-dessus.

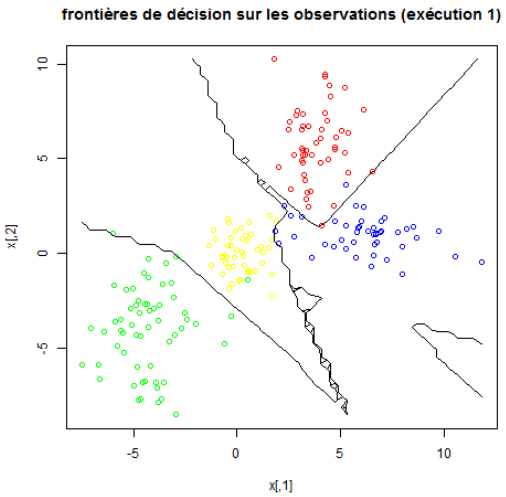
Pour appliquer la formule de Bayes, on calcule les densités de probabilité de chacune des lois qu'on multiplie à leur probabilité π, ce qui nous permet de les comparer entre elles pour chaque point.
Cela se traduit dans le code R par la définition d'une grille sur le graphique qui nous permet de comparer la probabilité d'appartenance à l'une des lois en chacun des points de cette grille. Ce traitement nous permet de récupérer et d'afficher (au moyen de la fonction *contour*) le contour de chacune des densités supérieures à 0 des 4 classes.

Réseaux de neurones sur des données simulées

Question 2 :

1. Dessiner le frontières de décision obtenues avec un réseau de neurones sur les données (pour decay = 0 et size = 5)

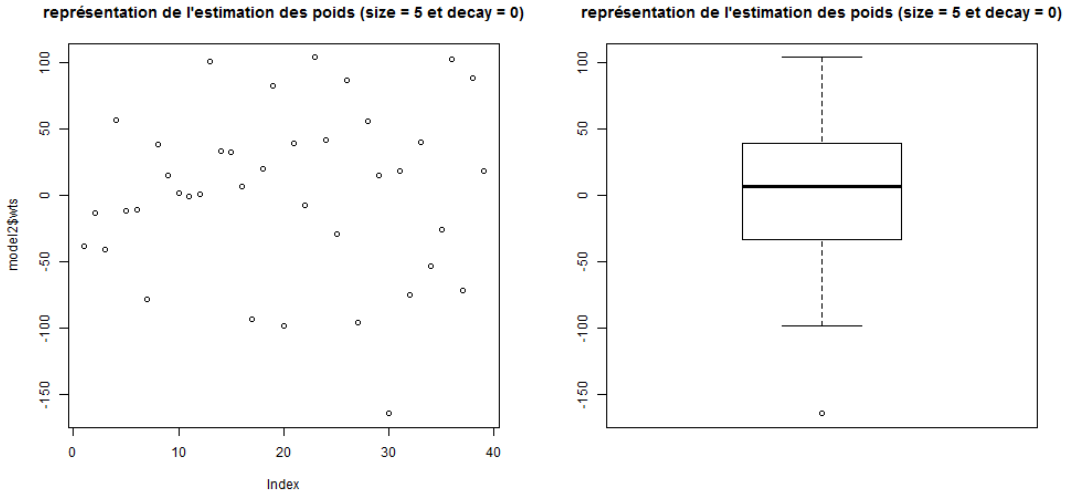
L'exécution du code donné nous permet d'observer le graphique suivant :



Nous pouvons observer sur ce graphiques les différentes observations de l'ensemble d'apprentissage avec une couleur précise et les frontières de décision obtenues. On peut dire que tous les points de chaque loi normale ne se situent pas exactement du bon coté des frontières de décision.
Donc il y a une certaine probabilité d'erreur qu'un individu appartienne à une classe.

2. Visualiser l'estimation des poids.

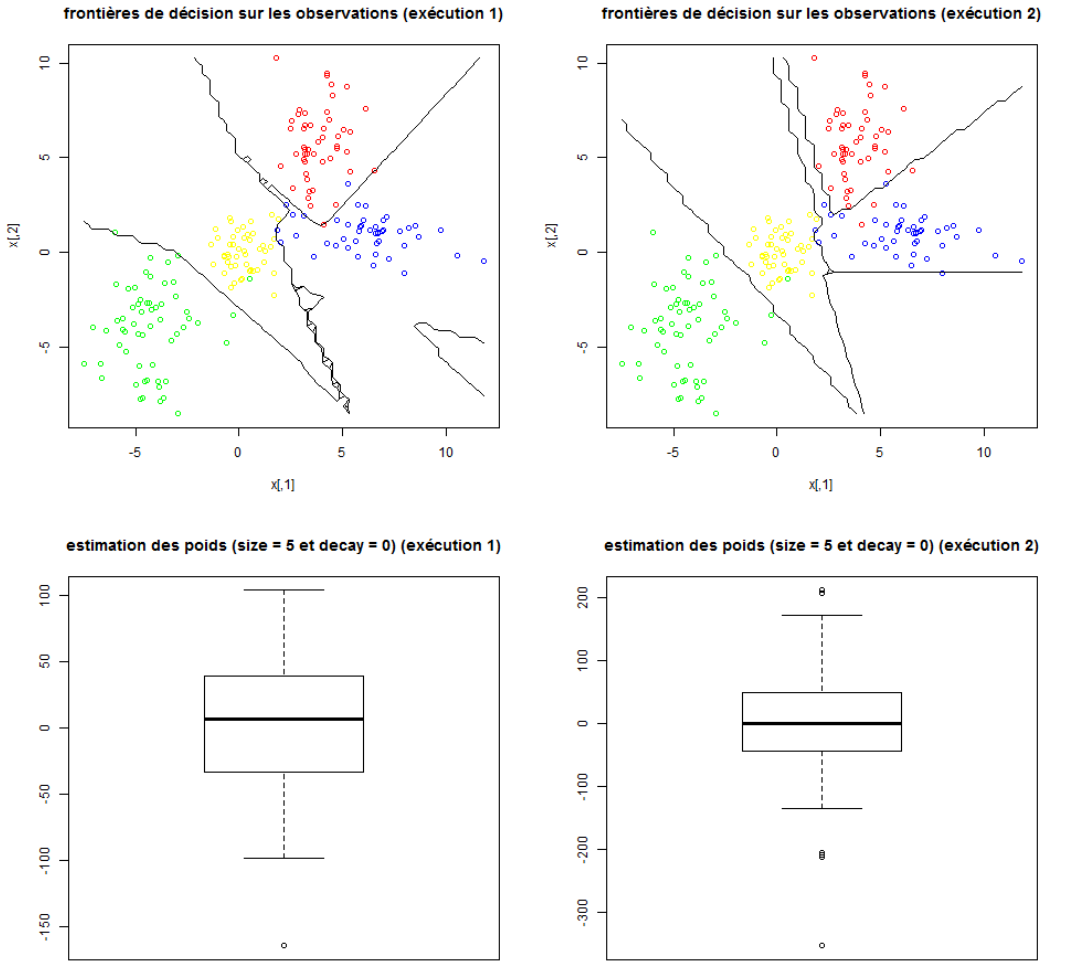
L'estimation des poids pour decay = 0 et size = 5 nous donne le graphique suivant :



On peut voir que la majorité des poids sont compris entre -40 et 40.

3. Obtient-on les mêmes résultats à chaque fois pour l'estimation des poids et des frontières de décision lorsqu'on lance plusieurs fois la procédure `nnet`

En relançant la procédure `nnet`, nous n'obtenons pas les mêmes frontières de décision.



Nous pouvons observer que les probabilités d'erreur ne semblent pas être les mêmes entre les 2 exécutions, bien que les frontières séparent pour les deux modèles assez bien les différentes classes.

Du coté de l'estimation des poids, nous obtenons des valeurs plutôt semblables entre -50 et 50, cependant pour les valeurs extrêmes, il n'y a pas du tout le même ordre de grandeur, de 105 à -100 pour le modèle obtenu lors de la première exécution et de 170 à -140 pour le second modèle obtenu lors de la deuxième exécution.

On voit donc bien que l'exécution de `nnet` donne pour résultats des modèles différents, même si les paramètres `size` et `decay` sont identiques. Il y'a donc un autre paramètre qui influence les solutions de `nnet`.

4. Comment expliquer ce phénomène ?

Il semble que ce phénomène vienne du fait que le générateur de nombre aléatoire de R n'est pas réinitialisé entre les 2 exécutions de `nnet`. En effet la méthode de rétro-propagation qui est utilisée dans cet algorithme afin d'ajuster les poids lors de l'apprentissage, requiert au préalable une initialisation des poids du réseau de neurones avec des valeurs aléatoires. Ces valeurs aléatoires sont obtenues grâce à la fonction `rand` ne sont donc pas les même d'une exécution de `rand` à l'autre, c'est pourquoi les valeurs initiales des poids seront également différentes d'une exécution de `nnet` à l'autre, ce qui entraîne des combinaisons de poids différentes à la fin de l'algorithme.

Dans notre cas, nous souhaitons par la suite examiner l'influence d'autres critères que les poids du réseau sur la pertinence de la solution obtenue, il sera donc nécessaire de prendre des poids initiaux identiques afin que ce paramètre n'influence pas sur la solution données par `nnet`. Pour cela il suffit d'utiliser la fonction `set.seed(1)` avant chaque appel de `nnet`. Cette fonction permet de réinitialiser le générateur de nombre aléatoire. On obtiendra donc les mêmes poids initiaux pour tous nos modèles.

Question 3 :

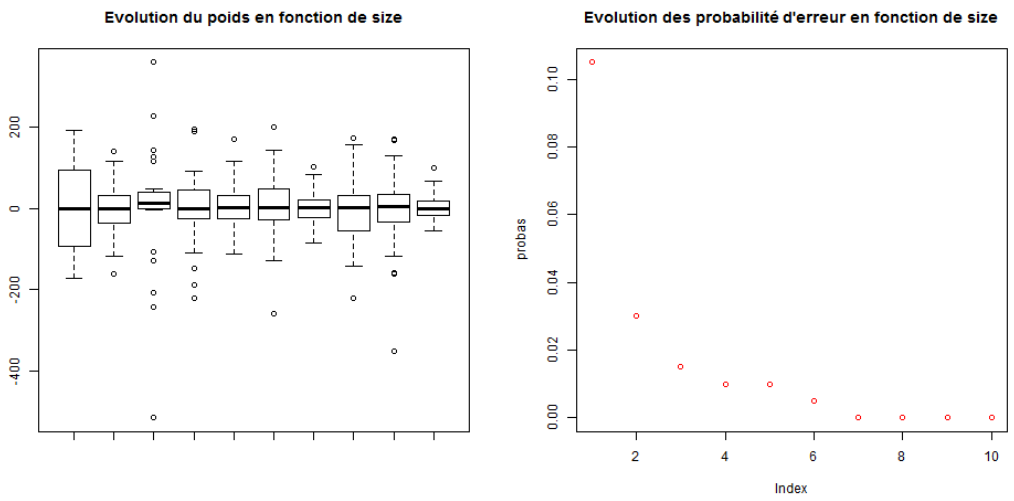
On désire voir l'influence du nombre de neurones dans la couche cachée. Pour cela, pour decay = 0, on fait varier `size` de 1 à 10 (nombre de neurones dans la couche cachée).

1. Commenter le nombre de "poids" à estimer.

Selon l'architecture du modèle, le nombre de poids à estimer varie, chaque poids étant associé à une liaison du réseau. Il dépend du nombre d'entrées k (+ biais éventuel), du nombre de neurones cachés l (+ biais éventuel), et du nombre de neurones de la couche de sortie m. Dans notre cas où chaque neurone d'une couche est relié à tous les neurones et biais de la couche précédente, on obtient l'équation :

$$\text{nbre de poids} = (k+1) * l + (l+1) * m$$

La variation de `size` nous donne les graphiques suivants par rapport à l'estimation des poids optimaux:



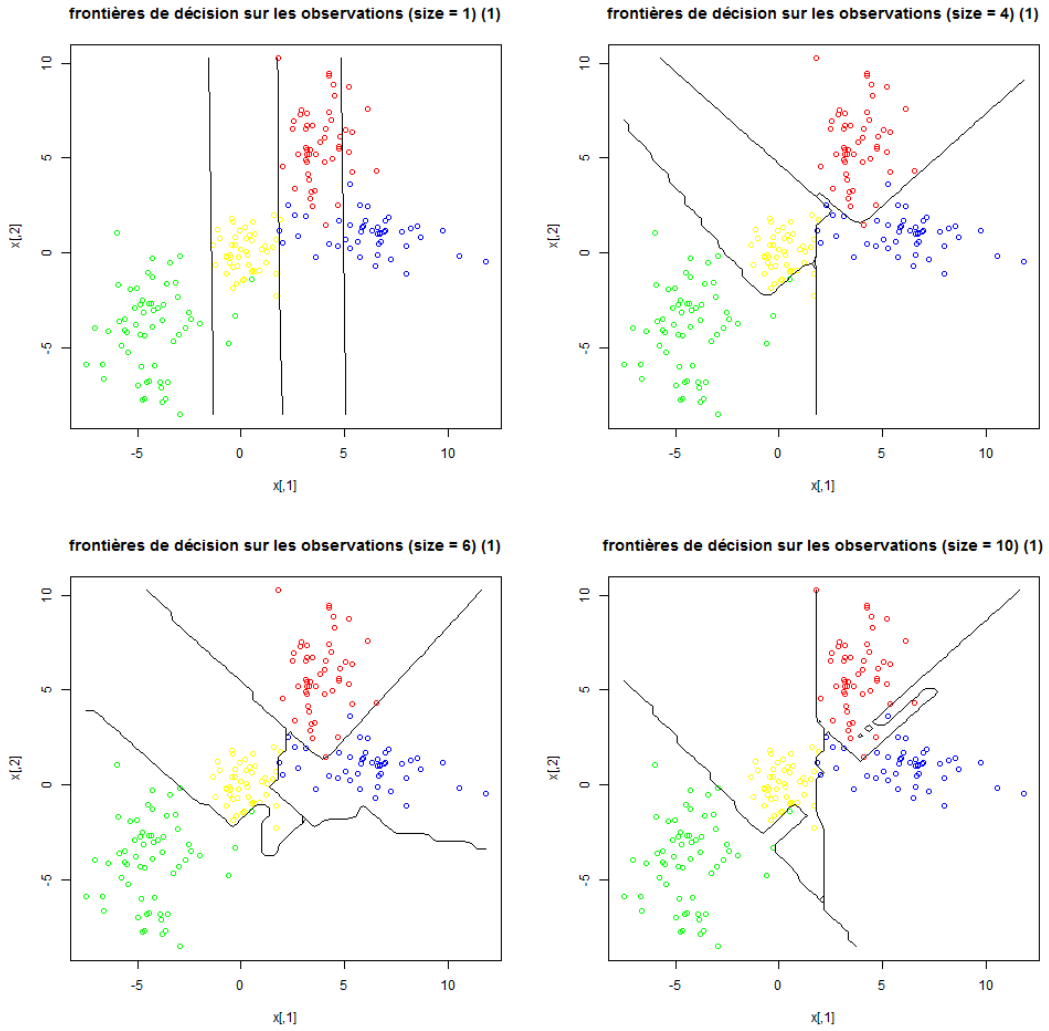
Nous pouvons observer que pour les plus grandes valeurs de `size`, la probabilité d'erreur sur les poids optimaux est très faibles. Si le nombre de neurones cachés est faible (par exemple ici avec `size` = 1), il y a donc une plus grande chance qu'un point soit mal classé, ce qui s'explique par la faible complexité du modèle et de la fonction de décision générée par ce dernier.

On observe, sur l'estimation des erreurs empiriques en fonction de `size` que la probabilité d'erreur empirique diminue entre 1 et 4, puis se stabilise à 5, pour redescendre entre 6 et 8, et se stabiliser finalement à une valeur très basse.

Cela vérifie bien la théorie que l'erreur empirique doive tendre vers 0 plus le nombre de neurones cachés augmente. Cependant, si notre modèle ne commet que très peu d'erreur sur les données d'apprentissage qui lui sont fournis, il n'est pas dit qu'il en soit de même pour de nouvelles données. Le modèle s'étant trop adapté aux données d'apprentissage, sa capacité à trouver une solution qui puisse bien généraliser le problème peut en être affectée et engendrer de mauvais résultats.

2. Afficher les frontières de décision (4 seulement). Que constater sur la forme des frontières et sur le nombre de points mal-classés quand `size` augmente ?

Nous choisissons de fixer les poids initiaux et `decay` mais de faire varier `size`. On affiche les frontières de décision pour `size` égal à 1, 4, 6 et 10 d'après l'évolution des probabilités d'erreur empirique vue précédemment.



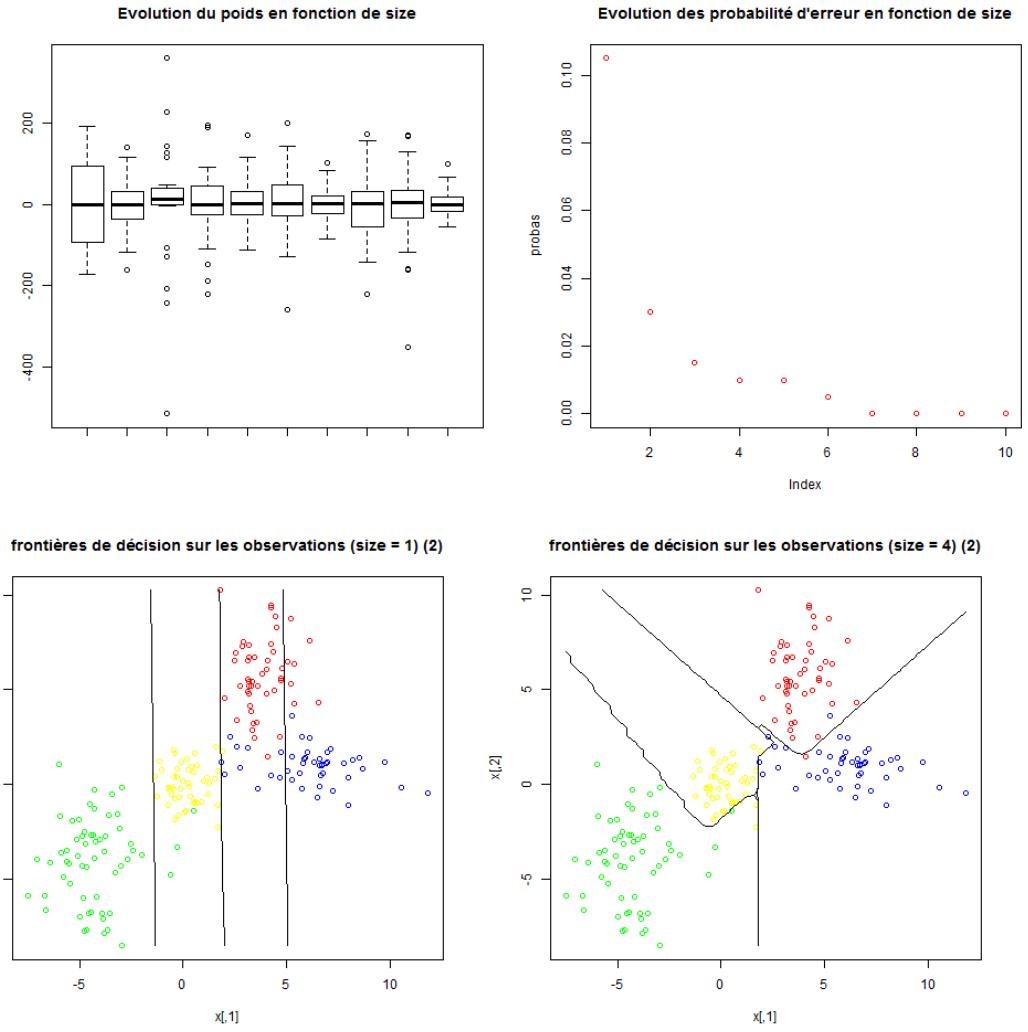
Pour le premier graphique, où la probabilité d'erreur empirique est la plus élevée, les frontières de décision sont des droites. Par conséquent le nombre de points mal-classé se retrouve à être assez conséquent par rapport aux autres graphiques, dont les frontières de décision sont des courbes. Cela s'explique par la simplicité du modèle (`size` =1), qui ne permet pas de générer une fonction de décision assez complexe pour être adaptée à notre problème.

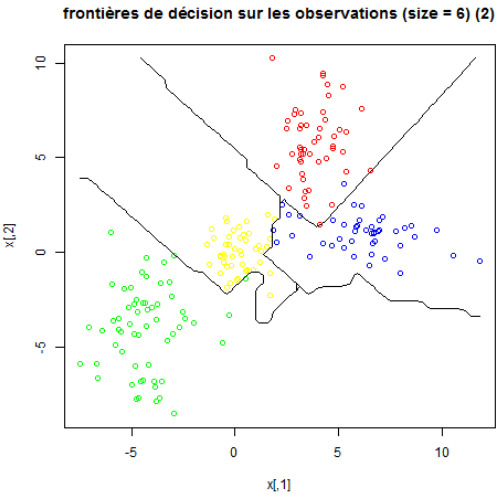
On remarque plus `size` augmente, plus les courbes sont complexes et délimitent les classes précisément.

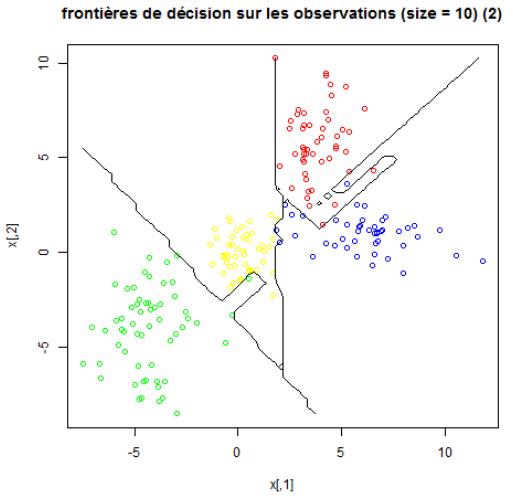
Par exemple, entre le troisième et le quatrième graphique, certains points jaunes se situent sur la frontière entre la classe bleue et la classe verte pour `size` = 4, tandis qu'ils sont inclus dans la classe jaune pour `size` = 6. Plus `size` augmente plus les courbes semblent faire des "détours" afin d'inclure des points particuliers (point bleu parmi les points rouges dans le quatrième graphique). Cela prouve bien que plus le nombre de neurones cachés augmente, plus le modèle s'adapte précisément aux données d'apprentissage qui lui sont fournies (d'où les faibles erreurs empiriques des modèles avec `size` > 6), au détriment de la capacité de généralisation du modèle.

Si le modèle à `size` = 4 semble faire plus d'erreurs, notamment pour les points particuliers, il semble mieux généraliser la classification et donc être plus adapté pour la classification de futures données que les modèles avec `size` = 6 et `size` = 10.

3. Relancer la procédure sur un nouveau jeu de données en faisant varier size de 1 à 10.







Nous pouvons observer que les frontières de décision et les estimations des poids optimaux ne changent pas avec un nouveau jeu de données du fait que nous veillons à ré-initialiser le générateur de nombre aléatoire du logiciel R avant toute exécution de l'algorithme.

4. Que dire de la variabilité du modèle en fonction de `size` ?

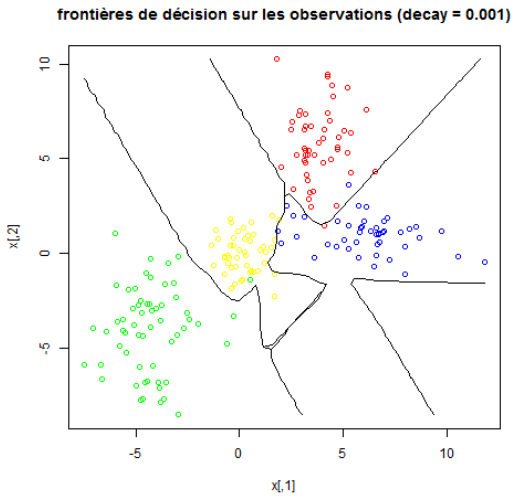
Plus la taille du réseau de neurone augmente, plus les frontières de décision seront complexes et précises, la probabilité qu'un point soit mal-classé est plus faible lorsque `size` augmente.

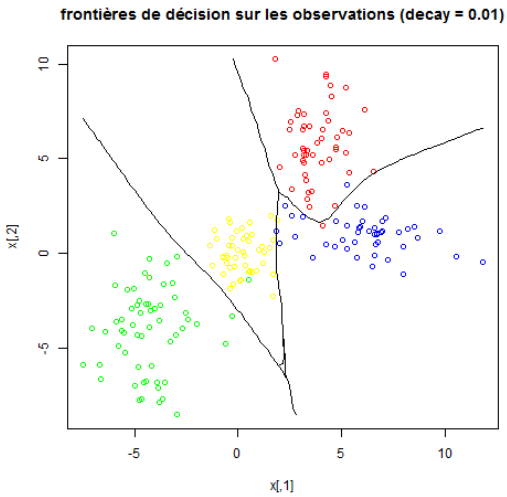
Cependant, nous avons observé que lorsque le nombre de neurones cachés se rapproche de la valeur du nombre lois utilisées pour fournir l'ensemble d'observation, la probabilité d'erreur de l'appartenance d'un point à une classe est plus faible que pour les valeurs de `size` adjacentes. De manière générale plus le nombre de neurones cachés augmente, plus le modèle s'adapte précisément aux données d'apprentissage qui lui sont fournies (d'où les faibles erreurs empiriques des modèles avec une valeur `size` élevée), au détriment de la capacité de généralisation du modèle. Un modèle s'étant trop adapté aux données d'apprentissage aura une capacité médiocre à trouver une solution qui puisse bien généraliser le problème et engendrer de mauvais résultats.

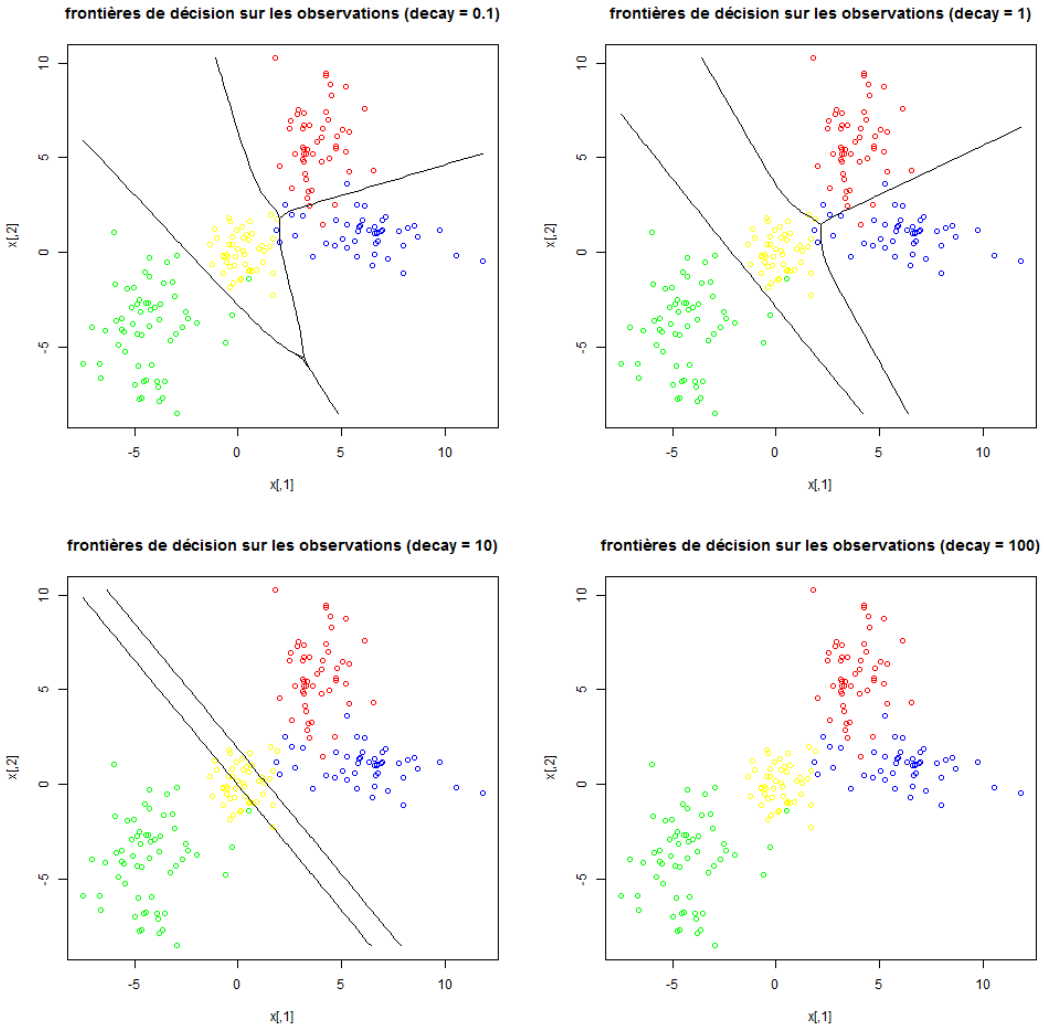
Question 4 :

On désire voir l'influence du paramètre de régularisation `decay` utilisé pour limiter le surapprentissage dans un réseau de neurones. Il s'agit d'une pénalité ajoutée à la fonction d'erreur qui pénalise les poids trop grands. En effet plus il est important et moins les paramètres ou poids peuvent prendre des valeurs "chaotiques" contribuant ainsi à limiter les risques de surapprentissage.

1. Pour `size` = 5, comparer l'estimation des poids et des frontières pour `decay` = 0.001, 0.01, 0.1, 1, 10, 100.

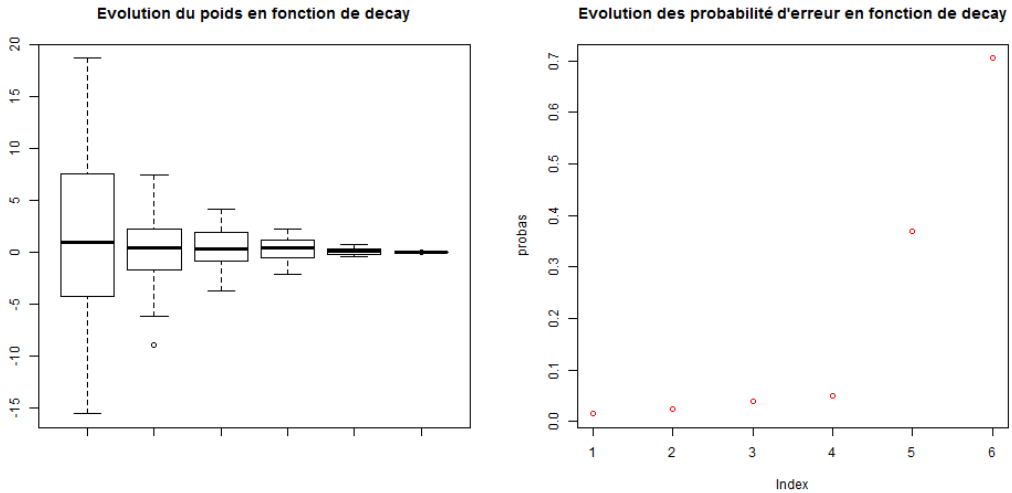






Pour la représentation des frontières de décision suivant les différentes valeurs du paramètre de régularisation, nous pouvons observer que plus la valeur de `decay` est grande, moins les frontières de décision sont précises. Pour une valeur de 100, nous n'avons même plus de frontières de décision (une seule classe), tandis que pour une valeur de 10, les frontières sont très peu précises (nous avons juste 2 droites séparant approximativement les classes). Ces deux derniers modèles sont un exemple de sous apprentissage, ils sont beaucoup trop généralistes. Pour les valeurs très petites de `decay`, les frontières de décision font plus attention au découpage des classes par rapport aux zones d'intersection des classes. On observe particulièrement pour `decay` $= 1$ un comportement de surapprentissage, on voit en effet que le frontières forment des limites complexes, quitte à inclure quelques points particuliers.

L'estimation des poids associés aux différentes valeurs de `decay` nous donne les graphiques suivants :



On observe que plus la valeur du paramètre de régularisation augmente, plus les poids sont de valeurs faibles voire proches de 0, tandis qu'une faible valeur de `decay` permet une plus grande amplitude des poids. De plus on remarque que la probabilité d'erreur augmente si `decay` est grand.

2. Que dire de la valeur des poids optimaux quand `decay` augmente ?

On peut voir sur le graphique des probabilités d'erreur en fonction de `decay` que pour des valeurs comprises entre 0.001 et 1, les probabilité d'erreur n'excède pas 10%, ce qui donne donc des valeurs de poids optimaux plutôt intéressantes alors que pour des valeurs supérieures à 1, la probabilité d'erreur augmente beaucoup pour être à environ 32% pour `decay` = 10 et 72% pour `decay` = 100, les valeurs de poids optimaux ne semblent donc pas être pertinantes.

Nous pouvons faire le lien de ces observations avec le `boxplot` de l'évolution du poids en fonction de `decay` où nous avons des boîtes à moustache dont les quartiles sont tous proches de zéro pour les 2 dernières valeurs du paramètre de régularisation (10 et 100), donc les 2 valeurs les moins probantes pour avoir un modèle pertinent. Pour une faible valeur de `decay`, les valeurs de poids varient beaucoup, avec une forte amplitude.

3. Comment expliquer ce phénomène et comment cela se traduit-il sur la forme des frontières ?

Ce phénomène provient de l'ajout du terme de pénalisation P au critère d'erreur R sur la régulation du modèle :

$$C = R + \lambda P, \ \lambda \text{ étant un } \textit{hyperparametre}.$$

Ce critère permet de limiter la grandeur des poids afin d'éviter le surapprentissage. Par conséquent, plus la valeur de `decay` est petite, moins les poids sont limité et plus le modèle proposé est complexe, alors que plus `decay` est grand, plus le modèle aura des poids faibles et sera donc plus généraliste et simple. Au niveau des frontières de décision, cela se traduit pas des frontières très précises pour un `decay` proche de 0 et des frontières moins précises, plus droites pour un grand `decay`. Pour un `decay` trop élevé, il n'y a plus de frontières, les données appartiennent toutes à la même classe.

Exercice 3 : Classification des données `crabs`

Dans cette partie nous utiliserons les données `crabs`. Elles concernent des mesures effectuées sur 200 crabes de deux variétés différentes, qui sont mâles ou femelles et peuvent être de couleur bleue ou orange. Pour chacun de ces crabes nous tiendront compte de deux mesures : le taille de leur lobe frontal (FL) et la largeur de leur arrière (RW).

Dans un premier temps nous représentons les crabes en coordonnées logarithmiques selon leur sexe (femelles en rose et mâles en cyan) et selon leur couleur (crabes oranges en orange et crabes bleus en bleu).

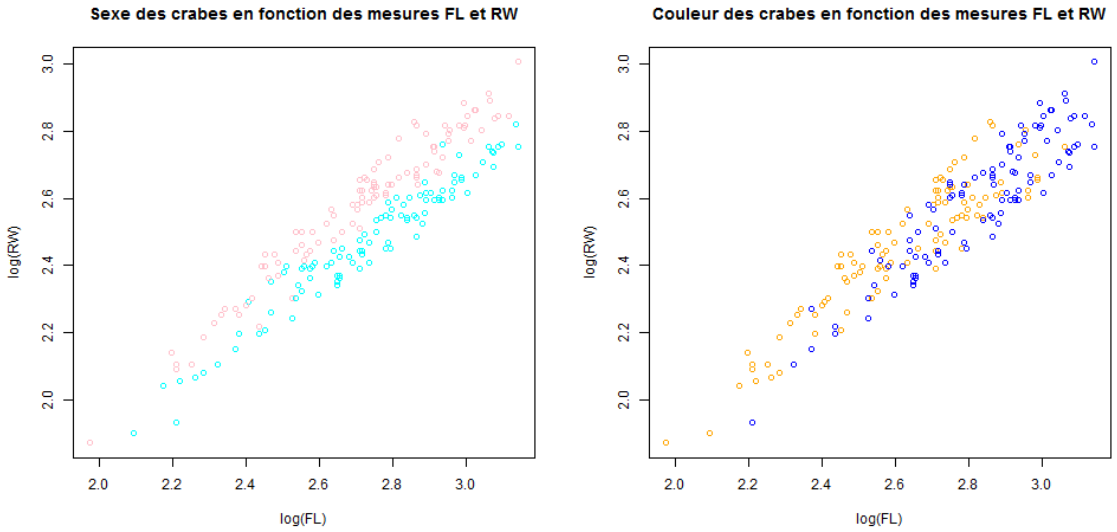


Figure 4: Représentations du sexe et de la couleur des crabes selon leur mesures FL et RW

Notre but est de trouver une règle de classification permettant de prédire le sexe et la couleur de ces crabes à partir de leurs mesures LF et RW.

Pour cela nous utilisons la méthode de validation croisée. Elle consiste à partitionner l'échantillon des données initiales $\mathcal{D}$ en N parties de tailles égales $\mathcal{D}_1, .., \mathcal{D}_N$. On entraîne alors N modèles différents en faisant pour chaque modèle l'apprentissage sur $\mathcal{D} \backslash \mathcal{D}_k$ et en évaluant sa probabilité d'erreur sur la partie restante qui sert donc de données de test. Après avoir calculé la probabilité d'erreur $\hat{\mathcal{R}}_{(k)}$ de chaque modèle, on calcule la moyenne des probabilités d'erreur $\hat{\mathcal{R}}$, ce qui nous permet alors d'avoir un bon apercu des performances générales.

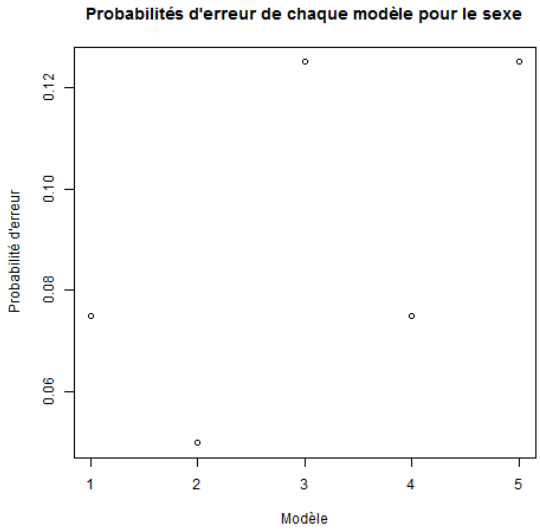
Cette méthode permet de mesurer le risque estimé (erreur que fait le modèle sur des données nouvelles) et non le risque empirique (erreur que fait le modèle sur les données d'apprentissage) ce qui est une méthode moins optimiste mais plus proche de la réalité, le but étant que notre modèle minimise le plus possible le risque sur de nouveaux exemples, et donc qu'il ait de bonnes capacités de généralisation. Bien que cette méthode entraîne des temps de calculs plus importants, elle permet une bonne utilisation des données (puisque chaque exemle est utilisé tour à tour pour l'apprentissage et pour le test).

Dans notre cas nous divisons les données en $N = 5$ parties, après les avoir préalablement mélangées de manière aléatoire pour éviter qu'une certaine classe n'influence plus les poids que les autres classes. Puis nous entraînons cinq modèles de réseaux différents, chacun utilisant 4 échantillons d'apprentissage différents, l'échantillon restant étant à chaque fois réservé pour le test.
Puis, pour chaque modèle, on teste la probabilité d'erreur sur l'échantilllon qui n'a pas été utilisé pour l'apprentissage.

Question 5: Prédiction du sexe des crabes grâces aux mesures FL et RW

1. En prenant `size` = 6 et `decay` = 0.001, et en utilisant la méthode de validation croisée (5 fold), donner une estimation de la probabilité d'erreur du réseau de neurones associé. Comparer avec la méthode LDA.

Avec les paramètres `size = 6` et `decay = 0.001`, et en réinitialisant les poids initiaux aux mêmes valeurs grâce à la commande `set.seed(1)` avant l'étape d'apprentissage pour chaque modèle, nous obtenons les résultats suivants :

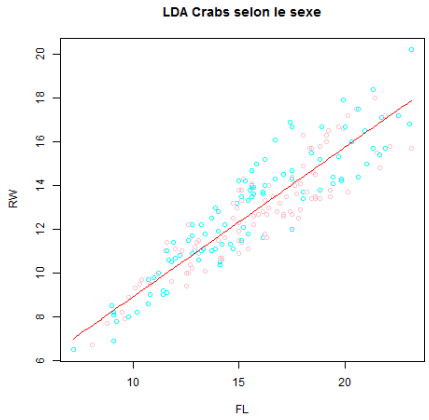


Pe_1	Pe_2	Pe_3	Pe_4	Pe_5
12.5%	12.5%	7.5%	7.5%	5%

Figure 5: Résultats de probabilités d'erreur de chaque modèle sur le sexe

Nous faisons enfin la moyenne des 5 pourcentages d'erreur obtenus pour avoir une estimation de la probabilité d'erreur, et nous obtenons $Pe_{moyenne} = 9\%$. Ce taux plutôt faible peut s'expliquer premièrement par la valeur la valeur `size` du nombre de neurones cachés qui augmente la complexité des modèles, et leur permet de générer une fonction de décision assez complexe pour être adaptée à notre problème. De plus la faible valeur de notre `decay` implique que les poids ne subissent quasiment pas le paramètre de régularisation, ce qui permet de s'adapter très facilement aux données d'apprentissage.

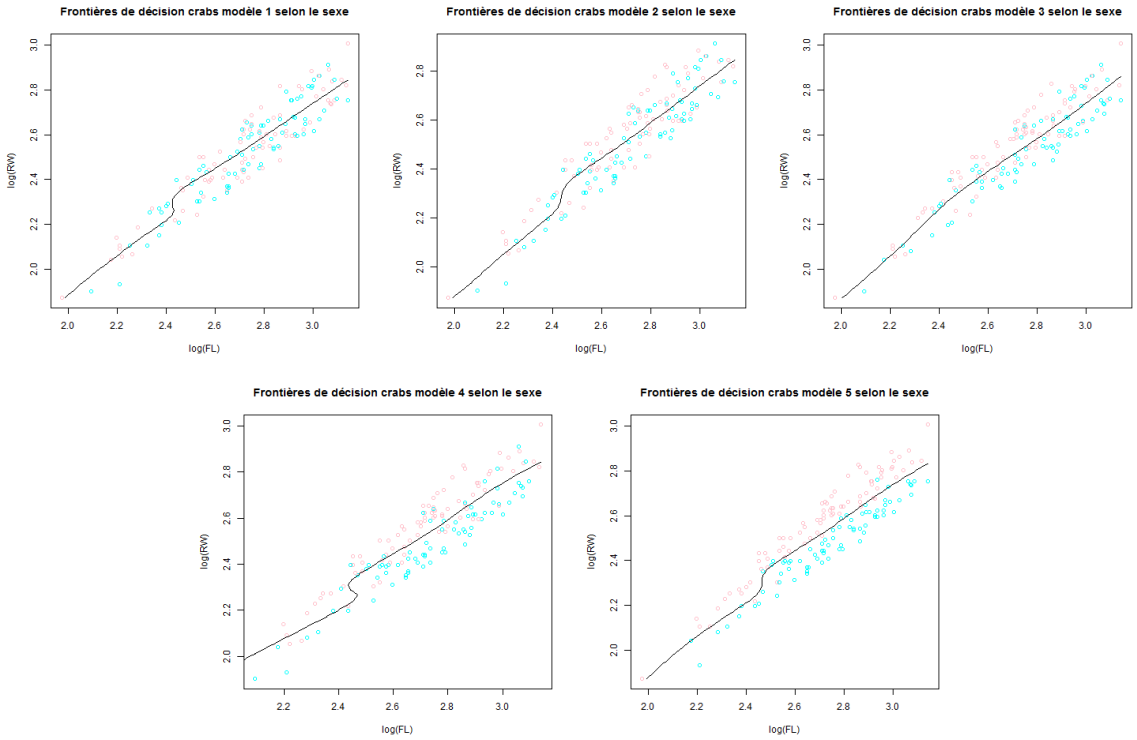
Nous appliquons ensuite l'analyse discriminante linéaire grâce à la fonction `lda`. Elle permet de prédire l'appartenance d'un individu à une classe prédéfinie à partir de ses caractéristiques mesurées à l'aide de variables prédictives en se basant sur la règle de Bayes. On obtient une erreur de 10% et la classification suivante :



Les classes femelle/homme semblant être facilement délimitable par une droite transversale et la LDA étant une méthode linéaire, elle peut être adaptée pour résoudre cette classification c'est pourquoi cette

dernière obtient une erreur presque similaire à la moyenne de nos modèles.

2. Dessiner les frontières de décision en mettant sur la figure les données d'apprentissage



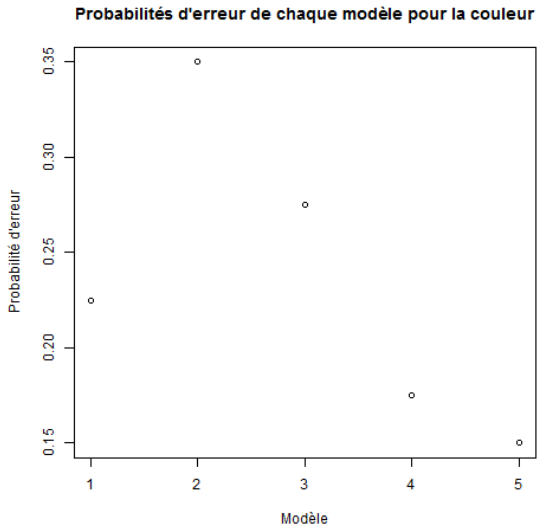
Les frontières obtenues pour les modèles semblent en adéquations avec les probabilité d'erreurs obtenues pour chaque modèle. En effet le modèle cinq semble réaliser très peu de mauvaises classification sur ses données d'apprentissage et obtient la plus faible probabilité d'erreur (5%), ce qui implique qu'il en fait également très peu sur les données de test. Ceci peut être dû à son ensemble d'apprentissage qui semble définir assez précisément la séparation de part et d'autre des classes mâle et femelles.

Le modèle 2 pour lequel les classes des données d'apprentissages sont assez mélangées (les points cyans et roses sont moins bien séparés que pour les données d'apprentissage du modèle 5), il semble plus difficile de trouver une courbe de séparation optimale. Ainsi la probabilité d'erreur en est affectée et atteint 12.5%.

Question 6: Prédiction de la couleur des crabes grâces aux mesures FL et RW

1. En prenant `size` = 5 et `decay` = 0.001, et en utilisant la méthode de validation croisée (5 fold), donner une estimation de la probabilité d'erreur du réseau de neurones associé. Comparer avec la méthode LDA.

Avec les paramètres `size = 5` et `decay = 0.001`, et en réinitialisant les poids initiaux aux mêmes valeurs grâce à la commande `set.seed(1)` avant l'étape d'apprentissage pour chaque modèle, nous obtenons les résultats suivants :

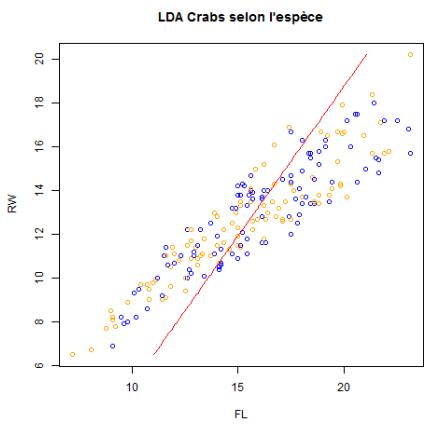


Pe_1	Pe_2	Pe_3	Pe_4	Pe_5
22.5%	35%	27.5%	17.5%	15%

Figure 6: Résultats de probabilités d'erreur de chaque modèle sur la couleur

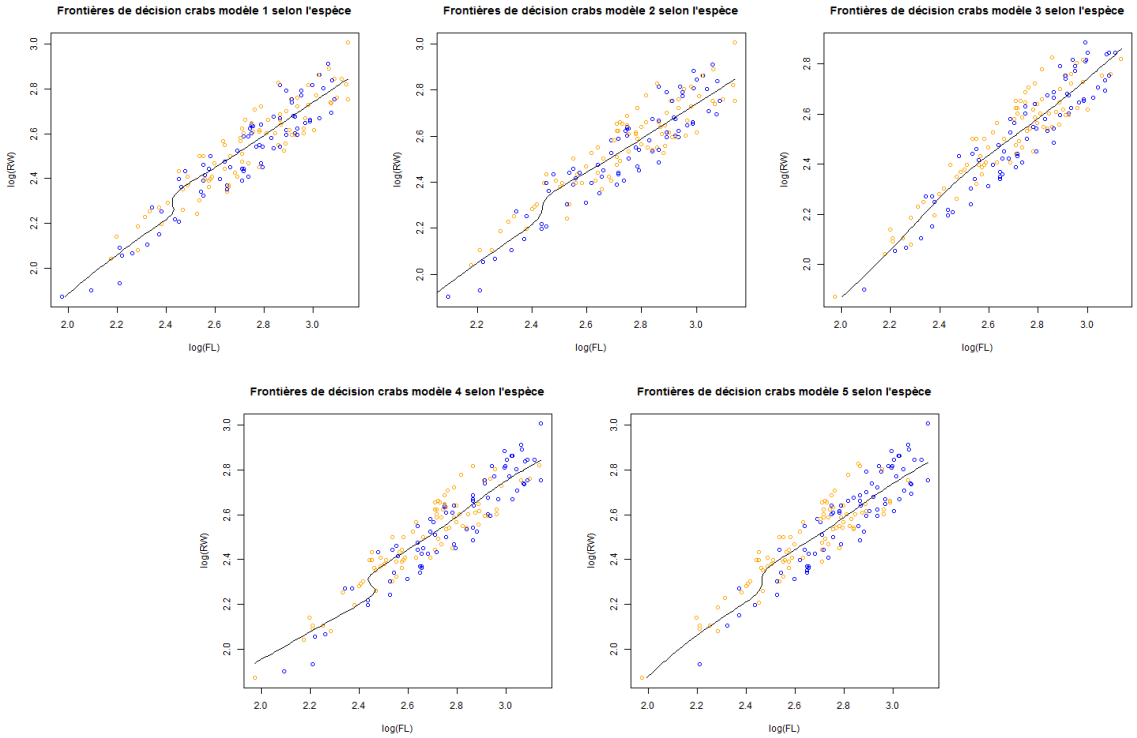
Nous faisons enfin la moyenne des 5 pourcentages d'erreur obtenus pour avoir une estimation de la probabilité d'erreur, et nous obtenons $Pe_{moyenne} = 23.5\%$. Ce taux est beaucoup plus élevé que celui de la partie précédente. Il semble que même si la faible valeur de notre `decay` implique que les poids ne subissent quasiment pas le paramètre de régularisation, cela qui permet théoriquement de s'adapter très facilement aux données d'apprentissage, d'autres raisons sont la cause de la médiocrité de notre modèle. Cela peut s'expliquer premièrement par la complexité des données : en effet on peut voir que contrairement aux classe femelle/mâle, les classes bleu/orange déterminant la couleur des crabes semblent beaucoup plus mélangées, donc dur à séparer les uns des autres et à classifier. Deuxièmement la valeur `size` = 5 du nombre de neurones cachés, moins élevée que pour le modèle précédent implique une complexité plus faible des modèles, et ne leur permet peut-être pas de générer une fonction de décision assez complexe pour être adaptée à notre problème.

Nous appliquons ensuite l'analyse discriminante linéaire grâce à la fonction `lda`. On obtient une erreur de 31% et la classification suivante :



Les classes bleu/orange n'étant pas facilement délimitables par une droite et la LDA étant une méthode linéaire, elle n'est peut être pas la plus adaptée pour résoudre cette classification c'est pourquoi cette dernière obtient une erreur bien supérieur à celle obtenue précédemment, mais aussi à celle de la moyenne de nos modèles.

2. Dessiner les frontières de décision en mettant sur la figure les données d'apprentissage



Les frontières obtenues pour les modèles semblent en adéquations avec les probabilité d'erreurs obtenues pour chaque modèle, qui sont relativement élevées (entre 15% et 35%, ce qui est très grand). On voit bien qu'aucun des modèles ne permette vraiment une classification pertinente des données, quelles que soient leur données d'apprentissage et de test.

Même si on pourrait essayer d'augmenter le nombre de neurones de la couche cachée, le réseau de neurone ne semble ici pas être la solution adaptée si on ne tient compte que des mesures RW et FL. Tester les résultats du réseau de neurone en incluant une troisième voire une quatrième entrée dans la structure du réseau (on peut utiliser les autres mesures des données `crabs` CM, CW et BD) pourrait être une solution et pourrait peut-être aider à déterminer une fonction de décision plus efficace.

Conclusion

Grâce à nos multiples expérimentations sur différents modèles de réseaux de neurones artificiels, nous avons pu expérimenter l'influence des poids initiaux qui peuvent donner des modèles différents en influant dans l'algorithme de réto-propagation qui met à jour les poids des liaisons du réseau. Nous avons pu constater la difficulté de choisir un bon rapport entre le nombre de neurones cachés et le paramètre de régularisation afin d'obtenir un modèle capable de fournir une fonction de décision assez complexe pour répondre à notre problème de façon pertinente mais tout de même générale, sans tomber dans le surapprentissage.

Enfin bien que les réseaux de neurones soient de puissants outils, ils connaissent des limitent et ne peuvent pas forcément pour autant répondre à tous les problèmes de classification si les données ne sont pas adaptées à la complexité du problème qu'elles posent.