

# **Mémoire M1**

Prédiction de l'issue d'un match de tennis professionnel

Guillaume Levesque  
Rayan Nitcheu Touko  
Astrid Benamou

2021

## Remerciements

Nous souhaitons remercier M. Julien Stoehr et M. Pierre Cardaliaguet qui nous ont accompagné pour mener à bien ce mémoire et ont répondu à nos interrogations.

Merci également à M.Gabriel Turinici, Mme.Angelina Roche et M.David Gontier qui ont pris le temps de répondre à nos questions et nous ont permis d'avancer.

Enfin et tout particulièrement, nous voulons remercier M. Tristan Cazenave d'avoir accepté de nous encadrer tout au long de l'année, de nous avoir aidé à développer nos connaissances en machine learning et nos compétences en programmation, ainsi que de nous avoir indiquer des ouvrages et des axes de réflexion qui nous ont été très utiles.

## Avant-propos

**Encadrant :** Monsieur Tristan CAZENAVE (tristan.cazenave@lamsade.dauphine.fr)

**Motivation** "Comment pourrait-on modéliser et prédire cet évènement le plus justement possible..." est une problématique de plus en plus récurrente aujourd'hui. Mais qu'est-ce qu'une bonne prédiction ? Outre sa pertinence, c'est le niveau de confiance qu'on peut lui attribuer et son adéquation à la réalité qui nous fera l'apprécier, ou non. En clair, peut-on honnêtement prétendre à généraliser notre modèle comme un outil de prédiction fiable et adroit.

Quelles données retenir pour construire notre modèle, et quel(s) modèle(s) choisir ? Quel outil mathématique utiliser pour exploiter au mieux ces données ? Et enfin quelle conclusion en tirer ? Ce sont, d'un point de vue global, les questions auxquelles nous proposerons des réponses au cours de ce mémoire.

Et pour finir cet avant-propos, pourquoi le tennis ?

Au delà de notre intérêt commun pour ce sport, le fait que l'issue d'un match soit binaire, et que l'essentiel de la prédiction repose sur l'état de forme et le passif des deux joueurs, contrairement à d'autres disciplines, nous pensons tendre vers un modèle beaucoup plus précis (les autres paramètres étant déterministes (Surface sur laquelle le match se déroule, compétition concernée ...)). Aussi, cela permettra d'éviter un éventuel égarement à essayer de prendre en compte trop de paramètres pour la prédiction, comme nous aurions pu être tenté de le faire dans d'autres sports. Ainsi, nous allons pouvoir nous focaliser d'avantage sur la comparaison des différents modèles qu'il est possible d'utiliser, des différentes méthodes d'apprentissage et de prédiction, et de pouvoir comparer les résultats obtenus avec ces différentes méthodes.

# Table des matières

<b>1 Fondements mathématiques</b>	<b>6</b>
1.0.1 Paradigme de l'apprentissage statistique . . . . .	6
i Fonction de décision . . . . .	6
ii Espace des hypothèses . . . . .	7
iii Fonction de coût . . . . .	7
iv Risque . . . . .	7
1.0.2 Généralisation et sur/sous apprentissage . . . . .	9
i Sur/sous apprentissage . . . . .	9
ii Généralisation et compromis Biais - Variance . . . . .	9
iii Régularisation . . . . .	10
1.0.3 Critère de performance et sélection de modèle . . . . .	11
i Erreur de généralisation . . . . .	11
ii Jeu d'entraînement, jeu de test et jeu de validation . . . . .	11
iii Représentativité des données . . . . .	11
iv Critère de performance dans le cadre de la classification . . . . .	12
1.0.4 Cadre Bayésien . . . . .	12
i Loi de Bayes . . . . .	13
ii Règles de décision . . . . .	13
iii Théorie de la décision bayésienne . . . . .	14
iv Modélisation paramétrique . . . . .	17
<b>2 Construction de notre jeu de données</b>	<b>18</b>
2.1 Sélection et fusion des jeux de données . . . . .	18
2.2 Sélection des variables pertinentes . . . . .	18
2.3 Traitement des variables . . . . .	19
<b>3 Approche Machine Learning</b>	<b>20</b>
3.1 Première modélisation . . . . .	20
3.2 Régression Logistique ou "modèle logit" . . . . .	20
3.2.1 Feature scaling . . . . .	21
i Min-max normalization . . . . .	21
ii Mean normalization . . . . .	21
iii Standardization . . . . .	22
iv Conclusion . . . . .	22
3.2.2 Recherche linéaire . . . . .	22
i Recherche par tâtonnements . . . . .	22
ii Recherche linéaire exacte : Gradient pas optimal . . . . .	23
iii Méthode de Broyden Fletcher Goldfarb Shanno . . . . .	24
iv Gradient pas quasi-optimal : critère de Wolfe-Armijo . . . . .	25
3.2.3 Choix du point de départ . . . . .	26
i Initialisation aléatoire . . . . .	26
ii Initialisation pré-déterminée . . . . .	26
3.2.4 Sur-apprentissage et choix des variables explicatives . . . . .	26
i Choix du bon modèle . . . . .	26
ii Pénalisation Ridge . . . . .	27
iii Validation croisée V-fold . . . . .	28
iv Transition vers une approche plus géométrique . . . . .	28
3.3 Support Vector Machine (SVM) . . . . .	29
3.3.1 Séparabilité linéaire . . . . .	29
3.3.2 SVM à marge rigide (Hard SVM) . . . . .	30
3.3.3 SVM à marge souple (Soft SVM) . . . . .	34
3.3.4 Consistance du SVM . . . . .	34
3.3.5 Minimisation du risque empirique . . . . .	40
3.3.6 Astuce du noyau (Kernel trick) . . . . .	44

3.3.7	Complexité du SVM . . . . .	46
3.3.8	Résultats pratique . . . . .	47
i	Premier test . . . . .	47
ii	Sélection des hyperparamètres . . . . .	47
3.4	KNN (K-Nearest Neighbours) . . . . .	48
3.4.1	Plus proche voisin . . . . .	48
3.4.2	Diagramme de Voronoï . . . . .	49
3.4.3	k - Plus proche voisins . . . . .	49
3.4.4	Choix du nombre k de plus proche voisins . . . . .	50
3.4.5	Complexité du kNN . . . . .	60
3.4.6	Résultats pratique . . . . .	60
i	Premier test . . . . .	60
ii	Sélection des hyperparamètres . . . . .	60
iii	Généralisation du modèle . . . . .	60
3.5	Forêts aléatoires . . . . .	61
3.5.1	Théorie sur les forêts aléatoires . . . . .	61
3.5.2	classification supervisée . . . . .	62
3.5.3	Algorithme CART : Classification and Regression Trees . . . . .	62
3.5.4	Implémentation de l'algorithme . . . . .	63
3.6	Réseau de neurones . . . . .	65
3.6.1	Motivation . . . . .	65
3.6.2	Réseau de neurones à deux couches cachées . . . . .	65
i	Forward Propagation . . . . .	66
ii	Rétro-propagation . . . . .	66
iii	Descente de gradient stochastique . . . . .	67
iv	Vérification par différentiation finie de la Rétro-propagation . . . . .	70
v	Initialisation des poids . . . . .	71
vi	Algorithme de Polyak's Momentum - Amélioration de Nesterov . . . . .	71
vii	Recherche linéaire . . . . .	72
viiRMSprop	. . . . .	72
ix	ADAptive Moment estimation - ADAM & ADAMax . . . . .	73
x	Décroissance programmée du taux d'apprentissage . . . . .	73
<b>4</b>	<b>Approche Deep Learning</b> . . . . .	<b>74</b>
4.1	Introduction . . . . .	74
4.2	Création d'un réseau de neurones avec Keras . . . . .	75
4.2.1	Choix du nombre de couches du réseaux et de neurones dans chaque couche . . . . .	75
i	Pour la couche d'entrée . . . . .	75
ii	Pour la couche de sortie . . . . .	75
iii	Pour les couches cachées . . . . .	75
4.2.2	Choix d'autres hyperparamètres . . . . .	76
i	Epoch . . . . .	77
ii	Batch_size . . . . .	77
4.2.3	Choix des fonctions d'activation pour chaque couche et de la fonction de perte . . . . .	77
i	Fonction d'activation . . . . .	77
ii	Choix dans les couches cachées . . . . .	77
iii	Choix pour la dernière couche . . . . .	78
4.2.4	Comment vérifier la performance de notre réseau de neurones ? . . . . .	79
i	Division du jeu de données entre train et test . . . . .	79
ii	Validation_split . . . . .	79
iii	Métriques . . . . .	79
4.2.5	Conclusion . . . . .	79

<b>5 Conclusion</b>	<b>81</b>
5.1 Conclusion à mi-parcours . . . . .	81
5.2 Conclusion finale . . . . .	81

# 1 Fondements mathématiques

Dans cette partie nous allons définir les fondements théoriques qui nous permettront de mener à bien notre projet de prédire l'issue d'un match de tennis professionnel en fonction de données obtenus sur les principaux joueurs professionnels.

## 1.0.1 Paradigme de l'apprentissage statistique

On a un jeu de données que l'on considère indépendantes  $\mathcal{D}_n = (x_i, y_i)_{1 \leq i \leq n}$ , on note  $\mathcal{X}$  et  $\mathcal{Y}$  les espaces d'origines respectivement de nos observations et de leurs étiquettes associées (dans le cadre de notre mémoire il s'agira bien souvent de  $\mathbb{R}^p$  et  $\{0, 1\} \cup \{-1, 1\}$ ). On note également  $\mathcal{D}_x$  et  $\mathcal{D}_y$  respectivement les observations et les étiquettes de notre jeu de données. Étant donné notre jeu de donnée  $\mathcal{D}_n$ , on souhaite trouver une fonction qui permettrait de prédire l'étiquette associée à une nouvelle observation. On souhaite donc "extraire" des informations de notre jeu de données afin de pouvoir effectuer des prédictions.

### i Fonction de décision

**Définition 1** Soit  $(x, y) \in \mathcal{X} \times \mathcal{Y}$  une observation et son étiquette associée. On appelle modèle de prédiction ou modèle prédictif la fonction  $f$  définie par : 
$$\begin{cases} f: \mathcal{X} \longrightarrow \mathcal{Y} \\ x \longmapsto f(x) = y \end{cases}$$
 Il s'agit de la fonction qui à chaque observation lui associe son étiquette.

On a alors besoin d'une fonction intrinsèquement liée au modèle qui nous permettra de décider quelle étiquette affecter à chaque observation, on appelle cette fonction la *fonction de décision* ou *fonction discriminante*.

**Définition 2** On note pour tout ensemble  $E$ ,  $\#E := \text{Card}(E)$ . On partitionne alors  $\mathbb{R}$  en  $\#\mathcal{D}_y$  sous ensembles  $\mathbb{D}_i$ . On appelle fonction de décision la fonction  $g: \mathcal{X} \longrightarrow \mathbb{R}$  tel que :  $\forall x \in \mathcal{X}, f(x) = y_i \Leftrightarrow g(x) \in \mathbb{D}_i$

Supposons que nous ayons  $\mathcal{D}_y = \{0, 1\}$ , alors on peut poser par exemple  $g$  tel que :

$$\forall x \in \mathcal{X} \begin{cases} f(x) = 0 \Leftrightarrow g(x) \leq 0 \\ f(x) = 1 \Leftrightarrow g(x) > 0 \end{cases}$$

Dans ce cas, on décide que si notre fonction de décision est strictement positive pour notre observation  $x$  alors notre modèle prédira la valeur 1 pour notre observation, sinon elle prédira la valeur 0.

**Définition 3**  $\forall 1 \leq i \leq \#\mathcal{D}_y$  on appelle région de décision l'ensemble  $\mathcal{R}_i$  tel que :

$$\mathcal{R}_i = \{g \in \mathbb{D}_i\}$$

**Remarque 1** On remarque alors que la fonction de décision partitionne notre ensemble de départ  $\mathcal{X}$  en différentes régions de décision. On a alors :

$$\mathcal{X} = \bigcup_{i=0}^{\#\mathcal{D}_y} \mathcal{R}_i$$

## ii Espace des hypothèses

Lorsque l'on souhaite faire de la prédiction, il est nécessaire de choisir un modèle. Pour cela, un éventail de fonctions s'offrent à nous. Il est possible de décrire mathématiquement tous ce qui se rapporte au choix de notre modèle.

**Définition 4** On note l'espace d'hypothèses  $\mathcal{F}$  l'ensemble inclus dans  $\mathcal{Y}^{\mathcal{X}}$  qui représente des modèles que l'on va considérer pour un problème de prédiction donné. Supposons par exemple que l'on soit dans le cas où  $\mathcal{X} = \mathbb{R}^2$  et  $\mathcal{Y} = \{0, 1\}$ , si notre jeu de donnée semble séparé par une droite, alors il semblerait pertinent de choisir notre ensemble d'hypothèse comme étant l'ensemble des fonctions affines de  $\mathbb{R}^2$  vers  $\mathbb{R}$ , c'est à dire :

$$\mathcal{F} = \{h \in \mathcal{Y}^{\mathcal{X}} \mid \forall \alpha \in \mathbb{R}, \forall x, y \in \mathbb{R}^2, h(\alpha x + (1 - \alpha)y) = \alpha h(x) + (1 - \alpha)h(y)\}$$

Si l'on devait alors résumer le paradigme de l'apprentissage statistique mathématiquement :

- On se donne un jeu de données  $\mathcal{D}$
- On suppose qu'il existe une fonction  $\phi$  tel que  $\forall (x, y) \in \mathcal{X} \times \mathcal{Y}, y = \phi(x) + \epsilon$  avec  $\epsilon$  un bruit aléatoire
- On cherche la fonction  $f$  ou encore le modèle qui approche le mieux  $\phi$
- Pour trouver cette fonction  $f$  on essaye de déterminer une fonction de décision  $g$
- Pour déterminer cette fonction  $g$ , on essaye d'observer notre jeu de données afin trouver un ensemble d'hypothèses  $\mathcal{F}$ .

D'emblée, on voit que l'espace des hypothèses a une importance capitale dans le cadre de la prédiction. En effet, si notre espace  $\mathcal{F}$  ne contient pas "*la bonne fonction*", notre modèle ne sera jamais à même de prédire l'étiquette des observations de manière optimale.

## iii Fonction de coût

Afin de déterminer si notre modèle est précis, il va nous falloir des "*critères de précision*". Pour cela nous allons définir une fonction qui mesure à quel point notre prédiction est juste.

**Définition 5** Soit  $f$  un modèle prédictif, pour tout  $(x, y) \in \mathcal{X} \times \mathcal{Y}$  on définit la fonction de cout par une fonction  $L$  tel que : 
$$\begin{cases} L: \mathcal{Y}^2 \longrightarrow \mathbb{R} \\ (y, f(x)) \longmapsto L(y, f(x)) \end{cases}$$

Ou  $L$  est la fonction qui permet de quantifier l'écart entre notre prédiction et la vrai étiquette de notre observation. Ainsi, plus la fonction de coût est élevée et plus notre prédiction est éloignée de la réalité.

Il existe plusieurs fonctions de coût, mais toutes les fonctions de coût ne sont pas adaptées à tous les problèmes de prédiction. Le choix de la fonction va alors dépendre essentiellement de la forme du problème de prédiction. On a désormais un critère de précision afin de choisir notre fonction. Cependant, ce critère ne nous donne pas en fine d'indication suffisante sur l'optimalité de la méthode choisie. Afin de pallier ce problème nous allons donc définir des "*critères d'optimalité*".

## iv Risque

Afin de définir des critères d'optimalité, nous allons utiliser la notion de risque.

**Définition 6** Soit  $h$  une fonction de notre ensemble d'hypothèses, on définit le risque comme la fonction  $\mathcal{R}$  tel que :

$$\begin{cases} \mathcal{R} : \mathcal{F} \longrightarrow \mathbb{R} \\ h \longmapsto \mathcal{R}(h) = \mathbb{E}_{\mathcal{X}}[L(y, h(x))] \end{cases}$$

Avec  $\mathbb{E}_{\mathcal{X}}[h] = \mathbb{E}[\mathbb{1}_{\mathcal{X}}h]$ , ou on a deux cas possibles :

cas 1 :  $\#\mathcal{X}$  est fini ou infini dénombrable

$$\mathbb{E}[\mathbb{1}_{\mathcal{X}}h] = \sum_{x \in \mathcal{X}} h(x)$$

cas 2 :  $\#\mathcal{X}$  est infini non dénombrable

$$\mathbb{E}[\mathbb{1}_{\mathcal{X}}h] = \int_{\mathcal{X}} h(x)dx$$

Cette fonction calcul l'erreur moyenne réalisée par notre modèle sur l'ensemble des valeurs de  $\mathcal{X}$ .

On a alors un critère d'optimalité qui se dégage naturellement pour la sélection de notre modèle. On souhaiterait que notre choix se porte sur la fonction qui minimise l'erreur moyenne de prédiction sur toutes les valeurs de  $\mathcal{X}$ , c'est à dire :

$$f \in \arg \min_{h \in \mathcal{F}} \mathcal{R}(h)$$

Sans plus d'hypothèses, il existe cependant un certain nombre de problèmes avec ce critère d'optimalité. En effet, premièrement nous ne connaissons pas toutes les étiquettes des points de  $\mathcal{X}$ , on ne connaît que celles des observations de  $\mathcal{D}_x$ . Pour pallier ce problème nous allons nous restreindre à minimiser l'erreur moyenne de prédiction sur toutes les valeurs de  $\mathcal{D}_x$ .

**Définition 7** Soit  $h$  une fonction de notre ensemble d'hypothèses, on définit le risque empirique comme la fonction  $\mathcal{R}_n$  tel que :

$$\begin{cases} \mathcal{R}_n : \mathcal{F} \longrightarrow \mathbb{R} \\ h \longmapsto \mathcal{R}_n(h) = \frac{1}{n} \mathbb{E}_{\mathcal{D}_x}[L(y, h(x))] = \frac{1}{n} \sum_{i=1}^n L(y_i, h(x_i)) \end{cases}$$

Cette fonction calcul l'erreur moyenne réalisée par notre modèle sur l'ensemble des valeurs de notre jeu de données.

**Remarque 2** La loi des grands nombres nous assure que pour  $h$  fixé, le risque empirique converge vers le risque. Cependant, en posant  $h^* \in \arg \min \mathcal{R}_n(h)$  et  $g^* \in \arg \min \mathcal{R}(g)$ , nous n'avons aucune garantie que  $\mathcal{R}_n(h^*) \rightarrow \mathcal{R}(g^*)$ . Donc nous n'avons aucune certitude que la fonction qui minimise le risque empirique soit un bon estimateur de la fonction qui minimise le risque.

On a alors un nouveau critère d'optimalité qui cette fois est a priori nous donne un problème d'optimisation solvable :

$$f \in \arg \min_{h \in \mathcal{F}} \mathcal{R}_n(h)$$

Cependant ce n'est pas forcément le cas. En effet, il y aurait plusieurs remarques à faire :

- Selon le choix de  $\mathcal{F}$ , le modèle  $f$  peut avoir ou non une expression analytique explicite
- Souvent ce problème n'admet pas de solution unique globale

Pour ces raisons, dans la suite nous allons essayer de choisir (dès que possible) un espace d'hypothèses dans lequel notre problème a une solution analytique explicite, mais également une fonction de coût  $L$  convexe pour se retrouver dans le cadre de l'optimisation convexe et ainsi avoir une solution unique. Bien que la minimisation du risque soit un bon critère d'optimalité théorique, dans la pratique sans hypothèse en plus il est souvent insuffisant.

Supposons que l'on se donne un jeu de données  $\mathcal{D}$ , et que l'on choisit le modèle prédictif suivant :

$$\forall x \in \mathcal{X}, f(x) = \begin{cases} y_i & \text{si } (x, y_i) \in \mathcal{D} \\ \omega_i & \text{une valeur aléatoire} \end{cases}$$

Alors l'erreur empirique de notre modèle est minimale, pourtant sur des données réelles, notre modèle va prédire des étiquettes aléatoirement, il va donc avoir de mauvaises performances de manière générale. Cet exemple nous montre qu'il ne suffit pas que notre modèle performe sur les données mises à notre disposition, il faut en plus qu'il se "*généralise*" bien.

### 1.0.2 Généralisation et sur/sous apprentissage

#### i Sur/sous apprentissage

**Définition 8** Dans un problème de prédition, on parle de "surapprentissage" lorsque modèle apprend le bruit inhérent à notre jeu de données. Ou autrement dit, notre modèle a d'excellentes performances sur notre jeu de données mais de mauvaises performances en situation réelle.

**Définition 9** Dans un problème de prédition, on parle de "sousapprentissage" lorsque le modèle est trop simple pour pouvoir bien apprendre à partir de notre jeu de données. Ou autrement dit, le modèle a de mauvaises performances sur notre jeu de données (et a fortiori en situation réelle).

On en déduit de ces définitions qu'afin que notre modèle se généralise bien, il faudrait trouver un compromis entre apprendre suffisamment des données pour avoir de bonnes performances, mais, ne pas trop apprendre afin de ne pas apprendre l'aléa lié à nos données.

#### ii Généralisation et compromis Biais - Variance

Pour bien comprendre le risque lié à un modèle  $f$ , on peut comparer théoriquement les performances de ce dernier avec celle de la fonction mesurable de risque minimale.

**Définition 10** On note  $\mathcal{F}(\mathcal{Y}^{\mathcal{X}})$  l'ensemble des fonctions mesurables de  $\mathcal{X}$  vers  $\mathcal{Y}$ . On définit alors le risque minimale étendue à l'ensemble des fonctions mesurables par :

$$\mathcal{R}^* := \min_{h \in \mathcal{F}(\mathcal{Y}^{\mathcal{X}})} \mathcal{R}(h)$$

**Remarque 3** Puisque  $\mathcal{F} \subseteq \mathcal{F}(\mathcal{Y}^{\mathcal{X}})$  alors on a  $\mathcal{R}(f) \leq \mathcal{R}^*$ . On considère  $\mathcal{R}^*$  comme les meilleures performances que l'on peut avoir en théorie.

On peut alors décomposer l'écart de performance entre un modèle  $f$  et le meilleur modèle théorique, on appelle cette quantité *l'excès de risque* :

$$\mathcal{R}(f) - \mathcal{R}^* = \underbrace{[\mathcal{R}(f) - \min_{h \in \mathcal{F}} \mathcal{R}(h)]}_{(1)} + \underbrace{[\min_{h \in \mathcal{F}} \mathcal{R}(h) - \mathcal{R}^*]}_{(2)}$$

(1) Quantifie l'erreur entre un modèle  $f$  choisi et le meilleur modèle d'un ensemble d'hypothèses donné, on appelle cette erreur *l'erreur d'estimation*. Il s'agit de l'erreur dû à la sélection de notre ensemble d'hypothèses. A  $f$  fixé, plus nous choisissons un espace d'hypothèse vaste et plus nous aurons de modèles complexes à notre disposition, ainsi plus  $\min_{h \in \mathcal{F}} \mathcal{R}(h)$  aura tendance à baisser et donc l'erreur d'estimation augmentera mécaniquement. A contrario, à  $\mathcal{F}$  fixé, choisir un modèle plus complexe aura tendance à diminuer cette erreur d'estimation

(2) Quantifie l'erreur entre le meilleur modèle de notre ensemble d'hypothèses donné et le modèle théorique le plus performant, on appelle cette erreur *l'erreur d'approximation*. Il s'agit d'une erreur incompressible dans les faits. Non seulement cette erreur est constante à  $\mathcal{F}$  fixé, mais en plus, le seul moyen de diminuer cette erreur est de choisir un ensemble d'hypothèse plus vaste et donc encore une fois de choisir un modèle trop complexe.

Ainsi, dans la pratique, réduire l'erreur d'approximation (agrandir notre espace d'hypothèse) revient augmenter mécaniquement l'erreur d'estimation car plus l'espace d'hypothèse est grand, plus la solution est complexe et moins nous serons à même de la trouver, donc  $\min_{h \in \mathcal{F}}$  va diminuer sans que  $\mathcal{R}(f)$  diminue pour autant. Néanmoins, l'augmentation ou la diminution de ces 2 erreurs n'est pas linéaire. Par exemple, l'erreur (1) diminuera probablement plus vite en passant de  $\mathcal{F} = \mathbb{P}_2[x]$  à  $\mathbb{P}_3[x]$  plutôt qu'en passant de  $\mathbb{P}_{99}[x]$  à  $\mathbb{P}_{100}[x]$  car dans les 2 derniers cas il est de toute façon peu probable que l'on trouve la solution optimale. Il va donc falloir trouver un compromis entre ces 2 erreurs.

### iii Régularisation

La régularisation est une méthode qui permet de contrôler la complexité d'un modèle d'apprentissage. Le principe consiste à complexifier l'apprentissage de l'algorithme en ajoutant une contrainte au problème d'optimisation.

**Définition 11** Soit  $\Omega \in \mathcal{F}(\mathbb{R}^{\mathcal{F}})$  et  $\lambda \in \mathbb{R}^+$  un hyperparamètre. On appelle régularisation le problème suivant :

$$\arg \min_{h \in \mathcal{F}} \lambda \Omega(h) + \mathcal{R}_n(h)$$

Dans le cas où  $\lambda \rightarrow 0$ , alors l'importance du terme de contrainte  $\lambda \Omega(h)$  est négligeable devant le risque empirique. On se retrouve donc dans le cas d'un problème d'apprentissage classique où l'algorithme peut apprendre sans contrainte. Plus la valeur de  $\lambda$  est élevée et plus le terme régularisateur a d'importance, jusqu'à ce que l'erreur empirique soit négligeable devant le régularisateur et que seule la minimisation du régularisateur compte.

**Remarque 4** En général, le choix des hyperparamètres se fait de manière empirique : on évalue les performances et la généralisation du modèle selon plusieurs hyperparamètres et on sélectionne qui a le meilleur compromis performance/généralisation.

Nous avons donc déterminé tout un ensemble de critères théoriques d'optimalité et de précision, maintenant nous allons voir comment en pratique ces critères nous permettent de sélectionner quel modèle utiliser.

### 1.0.3 Critère de performance et sélection de modèle

#### i Erreur de généralisation

Reprenez notre critère d'optimalité et de performance principal, le risque empirique. On a vu précédemment le minimiseur du risque empirique du modèle n'était pas forcément un bon estimateur du minimiseur du risque réel. Considérer uniquement le risque empirique pour la sélection de notre modèle nous rend donc vulnérable au surapprentissage. Il est donc impératif d'évaluer les performances de notre modèle sur des observations dont on dispose des étiquettes associées.

#### ii Jeu d'entraînement, jeu de test et jeu de validation

**Définition 12** Soit  $\mathcal{D}$  notre jeu de données, afin de pallier au surapprentissage on partitionne notre jeu de données en 3 sous jeux de données disjoints :

$$\mathcal{D} = \mathcal{D}_{train} \uplus \mathcal{D}_{test} \uplus \mathcal{D}_{validation}$$

- $\mathcal{D}_{train}$  est le jeu de données sur lequel nous allons entraîner nos modèles
- $\mathcal{D}_{validation}$  est le jeu de données sur lequel nous allons sélectionner nos modèles
- $\mathcal{D}_{test}$  est le jeu de données sur lequel nous allons évaluer l'erreur de généralisation de nos modèles

**Remarque 5** En général on essaye de donner des tailles à peu près similaire à chacun des sous jeux de données.

#### iii Représentativité des données

la partition notre jeu de données est forcément arbitraire, de ce fait il est entièrement possible que nous ayons créé des sous jeux de données non représentatifs du jeu de données global. Pour éviter cet écueil de l'aléa, on dispose de plusieurs méthodes statistiques de *réduction de l'aléa*.

Nous utiliserons essentiellement une méthode, la Validation croisée. Le principe est simple, on se propose de réaliser plusieurs fois la procédure de partitionnement puis de moyenner les résultats obtenus.

**Définition 13** La *N*-Validation croisée s'effectue en 3 étapes distinctes :

(1) On partitionne  $\mathcal{D}$  en  $N$  ensembles de tailles sensiblement similaires

(2)  $\forall 1 \leq i \leq N$   $\begin{cases} (a) \text{On entraîne le modèle sur } \uplus_{j \neq i} \mathcal{D}_j \\ (b) \text{On évalue la performance du modèle sur } \mathcal{D}_i \end{cases}$

#### iv Critère de performance dans le cadre de la classification

Nous allons maintenant voir les principaux critères permettant d'évaluer la performance prédictive de nos modèles. Pour cela, nous allons établir différentes métriques de mesures de l'erreur commise par notre modèle.

Tout d'abord il est nécessaire d'effectuer une distinction entre les différents types d'erreur que peut commettre un modèle car toutes les erreurs ne se valent pas. En effet, si l'on se place dans le cadre d'un modèle prédisant la présence ou non d'une tumeur d'un patient, prédire à tort la présence d'une tumeur est une erreur plus "acceptable" que de ne pas prédire la présence tumeur chez un patient réellement malade. Afin d'établir cette distinction nous allons définir 4 erreurs.

**Définition 14** Étant donné un problème de classification binaire (ie :  $\#\mathcal{D}_y = 2$ ), on définit en cohérence avec notre problème une étiquette positive (+) ainsi qu'une négative (-). On se donne alors 4 types d'erreurs :

- *TP (Vrai positif)* : Il s'agit des exemples positifs correctement classifiés
- *FP (Faux positif)* : Il s'agit des exemples classifiés positivement à tort, ou erreur de 1ère espèce
- *TN (Vrai négatif)* : Il s'agit des exemples négatifs correctement classifiés
- *FN (Faux négatif)* : Il s'agit des exemples classifiés négativement à tort, ou erreur de 2nd espèce

**Remarque 6** La définition est extrapolable dans le cas où  $\#\mathcal{D}_y > 2$  mais nous allons nous limiter au cas binaire dans le cadre de notre mémoire.

A partir de ces différents types d'erreur nous pouvons dériver des critères d'évaluation de performance.

**Définition 15** On appelle précision (ou accuracy) le rapport entre les exemples correctement étiquetés et le nombre total d'exemples. On le note :

$$\text{Accuracy} = \frac{TP + TN}{\#\text{Exemples}}$$

C'est le principal critère de mesure de performance utilisé en apprentissage statistique, bien que ce seul indicateur ne soit pas à même d'extraire toutes les informations sur la performance de notre modèle.

#### 1.0.4 Cadre Bayésien

Jusqu'à présent nous avons considéré le cadre de l'apprentissage hors de toute considération de modélisation de l'information à notre disposition de manière probabiliste (on parle alors de cadre "fréquentiste"). En effet, dans la pratique nos données ne sont pas forcément le fruit d'un processus aléatoire (par exemple si on modélise le choix d'un jouet en fonction de l'âge et du sexe), cependant si dans les faits la répartition des choix suit une loi de probabilité connue il peut être intéressant de disposer de tous les outils probabilistes disponibles. Dans cette partie nous allons donc formaliser le concept de classification grâce à des modèles probabilistes.

On se donne toujours un jeu de données  $\mathcal{D} = (X_i, Y_i)_{1 \leq i \leq n}$ , cette fois on suppose que les  $X_i, Y_i$  sont des réalisations iid respectivement d'un couple de variables aléatoires  $(X, Y)$  c'est à dire qu'à la différence de notre première partie l'on rajoute directement l'aléa dans nos données. Afin de répondre à la question "*Comment nos données ont-elles été générées ?*", on va se munir de la loi jointe  $\mathbb{P}_{X,Y}$  qui mesure les relations de dépendance entre les variables aléatoires "*génératrices*" de nos données.

Considérons que  $\mathbb{P}(Y = c|X = x)$  soit la probabilité que l'observation  $x$  appartienne à la classe  $c$ , on va alors séparer notre problèmes d'apprentissage en 2 parties :

- (1) Déterminer les lois de probabilité de  $\mathbb{P}(Y = c|X = x)$  à partir de nos données (*Inférence*)
- (2) Utiliser ces lois afin de déterminer les classes de nouvelles observations (*Prédiction ou Décision*)

### i Loi de Bayes

Afin d'exprimer la distribution conditionnelle  $\mathbb{P}(Y|X)$  on va utiliser la loi de Bayes et définir une interprétation pour chaque terme.

**Définition 16** Soit  $x \in \mathcal{X}$  une observation et  $c \in \mathcal{Y}$  une classe, on exprime la loi de la distribution conditionnelle  $\mathbb{P}(Y|X)$  avec :

$$\mathbb{P}(Y = c|X = x) = \frac{\mathbb{P}(Y = c)\mathbb{P}(X = x|Y = c)}{\mathbb{P}(X = x)}$$

- (1)  $\mathbb{P}(Y = c)$  est la distribution à priori des étiquettes, c'est à dire la distribution des étiquettes avant toute prise en compte de l'observation.
- (2)  $\mathbb{P}(X = x|Y = c)$  est la vraisemblance. Il s'agit de la vraisemblance que l'on observe bien l'observation  $x$  en sachant que l'on considère l'étiquette  $c$ .
- (3)  $\mathbb{P}(Y = c|X = x)$  est la distribution des étiquettes à posteriori, c'est à dire celle des étiquettes après avoir considéré l'observation  $x$ .
- (4)  $\mathbb{P}(X = x)$  est la distribution marginale des observations en dehors de toute considération concernant leur classe.

### ii Règles de décision

Ici nous allons faire un parallèle entre la prise de décision dans le cadre classique et le cadre bayésien. Ici on considère  $x \in \mathcal{X}$  une observation,  $y \in \mathcal{Y}$  son étiquette et  $\hat{y} \in \mathcal{Y}$  l'étiquette prédite par notre modèle (ie  $\hat{y} := f(x)$ ). Notre but est de trouver une règle de décision qui va nous permettre de décider de quelle manière classer une nouvelle observation. Une règle de décision naturelle serait de prédire pour notre observation la classe qui maximise la valeur de la probabilité à posteriori.

**Définition 17** Supposons que  $\mathcal{Y} = \{0, 1\}$ , on appelle règle de décision par maximum à posteriori la règle définie par :

$$\hat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(Y = 1|X = x) > \mathbb{P}(Y = 0|X = x) \\ 0 & \text{sinon} \end{cases}$$

En utilisant la loi de Bayes puis en simplifiant par  $\mathbb{P}(X = x)$ , on obtient une nouvelle écriture :

$$\hat{y} = \begin{cases} 1 & \text{si } \mathbb{P}(X = x|Y = 1)\mathbb{P}(Y = 1) > \mathbb{P}(X = x|Y = 0)\mathbb{P}(Y = 0) \\ 0 & \text{sinon} \end{cases}$$

**Définition 18** On définit la fonction de régression de  $Y$  sur  $X$  par la fonction  $\eta : \mathcal{X} \mapsto \mathbb{R}$  tel que :

$$\begin{aligned}\forall x \in \mathcal{X}, \eta(x) &= \mathbb{E}[Y|X=x] \\ &= \mathbb{P}[Y=1|X=x]\end{aligned}$$

La dernière ligne est obtenu en utilisant la formule d'espérance conditionnelle discrète. En reprenant la règle initiale et en passant au complémentaire, on obtient la règle de décision de Bayes :

$$\hat{y} = \begin{cases} 1 & \text{si } \eta(x) \geq \frac{1}{2} \\ 0 & \text{sinon} \end{cases}$$

On l'écrit également sous la forme  $\forall x \in \mathcal{X}, \Phi_{Bayes}(x) = \mathbb{1}_{\{\eta(x) \geq \frac{1}{2}\}}$ .

**Définition 19** On appelle 'règle de décision plug-in' la règle qui consiste à prendre une décision en fonction de l'estimateur de  $\eta$ . On note cette règle :

$$\hat{y}_n = \begin{cases} 1 & \text{si } \eta_n(x) \geq \frac{1}{2} \\ 0 & \text{sinon} \end{cases}$$

### iii Théorie de la décision bayésienne

Nous avons vu dans la partie précédente qu'à l'instar du cadre de l'apprentissage statistique fréquentiste, nous pouvions définir pour notre modèle des règles afin de pouvoir prendre une décision quant aux prédictions qu'il devait réaliser. Nous allons maintenant voir comment choisir une des règles de décision. Pour cela nous allons nous appuyer sur des notions de théorie de la décision.

On pose les variables aléatoires :  $\begin{cases} X : \Omega \mapsto \mathcal{X}, \text{ représentant les données observées} \\ Y : \Omega \mapsto \mathcal{Y}, \text{ représentant une vérité cachée à déterminer} \end{cases}$  ainsi que la fonction  $a : \mathcal{X} \mapsto \mathcal{Y}$ , représentant une décision.

Nous allons commencer par définir le risque d'une décision comme nous l'avons fait dans la première partie.

**Définition 20** On définit le risque d'une décision  $\mathcal{R}(a)$  par la probabilité de réaliser une erreur de prédiction avec la décision  $a$ , c'est à dire :

$$\begin{aligned}\mathcal{R}(a) &:= \mathbb{P}(a \text{ réalise une erreur}) \\ &= \mathbb{P}(\exists \omega \in \Omega | a(X(\omega)) \neq Y(\omega)) \\ &= \mathbb{P}(a(X) \neq Y) \\ &= \mathbb{E}[\mathbb{P}(a(X) \neq Y | X)]\end{aligned}$$

On a alors une définition naturelle qui nous vient pour décider si une décision est optimale ou non.

**Définition 21** Soit  $\mathcal{A}$  l'ensemble des décisions possibles de prendre (dans le cadre des problèmes de classification binaire on prend  $\mathcal{A} = \mathcal{F}(\{0, 1\}^{\mathcal{X}})$ ). On appelle alors décision optimale de Bayes la fonction  $a^*$  qui minimise le risque sur  $\mathcal{A}$ , c'est à dire :

$$a^* = \arg \min_{a \in \mathcal{A}} \mathcal{R}(a)$$

**Proposition 1** Dans le cadre d'un problème de classification binaire, la règle de décision de Bayes  $\Phi_{Bayes}$  est la décision optimale de Bayes, c'est à dire  $a^* = \Phi_{Bayes}$ .

Preuve de la proposition :

Soit  $a \in \mathcal{A}$ , on a alors  $\mathcal{R}(a) = \mathbb{P}(a(X) \neq Y) = \mathbb{E}[\mathbb{1}_{a(X) \neq Y}] = \mathbb{E}[\mathbb{E}[\mathbb{1}_{a(X) \neq Y}|X]] = \mathbb{E}[\mathbb{P}(a(X) \neq Y|X)]$ . Cherchons premièrement à déterminer  $\mathbb{P}(a(X) \neq Y|X)$ .

Cas 1 : Si  $a(x) = 0$ , alors  $\mathbb{P}(a(x) \neq Y|X = x) = \mathbb{P}(Y = 1|X = x)$

Cas 2 : Si  $a(x) = 1$ , alors  $\mathbb{P}(a(x) \neq Y|X = x) = \mathbb{P}(Y = 0|X = x)$

On a alors :

$$\begin{aligned} \mathbb{P}(a(X) \neq Y|X) &= \mathbb{1}_{\{a(X)=0\}}\mathbb{P}(Y = 1|X) + \mathbb{1}_{\{a(X)=1\}}\mathbb{P}(Y = 0) \\ &= \mathbb{1}_{\{a(X)=0\}}\eta(X) + \mathbb{1}_{\{a(X)=1\}}(1 - \eta(X)) \\ &\geq \eta(X) \wedge (1 - \eta(X)) \end{aligned}$$

On remarque qu'en réalisant la disjonction de cas sur  $\{\eta(x) \geq 1/2\} \uplus \{\eta(x) < 1/2\}$ , on obtient :

$$\mathbb{P}(\Phi_{Bayes}(X) \neq Y|X) = \eta(X) \wedge (1 - \eta(X))$$

En intégrant par rapport à la loi de  $X$  on obtient que pour toute décision  $a$  :

$$\mathcal{R}(a) \geq \mathbb{E}[\eta(X) \wedge (1 - \eta(X))] = \mathcal{R}(\Phi_{Bayes})$$

Fin de la preuve.

**Remarque 7** On appelle la quantité  $\mathcal{R}^* := \mathbb{E}[\eta(X) \wedge (1 - \eta(X))]$  le risque de Bayes. Cette quantité sera très importante dans la suite de ce mémoire. En effet, premièrement cela nous donne une écriture explicite du modèle optimal dans le cadre de la classification binaire et de son risque. En outre, cette expression va nous permettre d'avoir une écriture explicite de l'excès de risque de tout modèle de classification binaire. Et pour finir, cette quantité nous donne une condition sur  $Y$  pour réaliser une classification sans erreur.

**Théorème 1** (Théorème de Györfi, Mammen et Tsybakov) Soit une décision  $a \in \mathcal{A}$ , on a une écriture explicite de l'excès de risque :

$$\mathcal{R}(a^*, a) = \mathbb{E}[\mathbb{1}_{\{a(X) \neq a^*(X)\}}|2\eta(X) - 1|]$$

*Preuve du théorème :*

On reprend notre équation sur les probabilités conditionnelle afin de calculer l'excès de risque et à la place de réaliser la disjonction de cas sur  $\{\eta(x) \geq 1/2\} \cup \{\eta(x) < 1/2\}$  on la réalise sur  $\{a^*(X) = 0\} \cup \{a^*(X) = 1\}$ . On a alors :

$$\begin{aligned}\mathbb{P}(a(X) \neq Y|X) - \mathbb{P}(a^*(X) \neq Y|X) &= \eta(X)(\mathbb{1}_{\{a^*(X)=1\}} - \mathbb{1}_{\{a(X)=1\}}) + (1 - \eta(X))((\mathbb{1}_{\{a^*(X)=0\}} - \mathbb{1}_{\{a(X)=0\}}) \\ &= (2\eta(X) - 1)(\mathbb{1}_{\{a^*(X)=1\}} - \mathbb{1}_{\{a(X)=1\}}) \\ &= |2\eta(X) - 1|\mathbb{1}_{\{a^*(X) \neq a(X)\}}\end{aligned}$$

Il suffit ensuite de primitiver sur la loi de  $X$  pour obtenir le résultat voulu.

*Fin de la preuve.*

**Définition 22** *On parle de problème de classification zéro-erreur lorsqu'il est possible de trouver un modèle qui réalise une classification sans erreur.*

**Proposition 2** *Si l'on se trouve dans un problème de classification zéro-erreur alors  $Y$  est une certaine fonction mesurable déterministe de  $X$   $\mathbb{P}-p.s.$*

*Preuve de la proposition :*

Un problème de classification binaire est zéro-erreur si  $\exists a \in \mathcal{A}$  tel que  $\mathcal{R}(a) = 0$ , or puisque d'après la proposition précédente on a  $\Phi_{Bayes}$  la décision optimale de Bayes, on a a fortiori  $\mathcal{R}^* = 0$ . D'après la définition du risque de Bayes, cela équivaut à  $\mathbb{P}(\Phi_{Bayes}(X) = Y) = 1$  c'est à dire  $Y = \Phi_{Bayes}(X)$   $\mathbb{P}-p.s$ .

*Fin de la preuve.*

On a désormais un nouvel objectif d'apprentissage pour notre modèle, il faudrait dans l'idéal que les performances de ce dernier se rapprochent de celles de la règle de Bayes. Or, l'accomplissement de cet objectif dépend énormément de la taille de notre jeu de données. Afin de bien voir le rapport de dépendance entre les données et notre modèle, nous allons désormais noter  $\mathcal{D} := \mathcal{D}_n$  et notre règle  $\forall x \in \mathcal{X}, a_n(x) := a_n(x, \mathcal{D}_n)$ . Dans ce nouveau paradigme, la qualité d'une règle est mesurée par la probabilité d'erreur conditionnellement à notre jeu de données

$$\mathcal{R}(a_n) = \mathbb{P}(a_n(X) \neq Y | \mathcal{D}_n)$$

De par l'aléa amener par notre jeu de données, on étudiera la convergence de notre modèle avec les outils de convergence de variables aléatoires.

**Définition 23** *On dit qu'une règle de classification est convergente (on parle également de consistance) si :*

$$\mathcal{R}(a_n) \xrightarrow{L^1} \mathcal{R}^*$$

*De la même manière, on dit qu'une règle est fortement convergente (on parle également de consistance forte) si :*

$$\mathcal{R}(a_n) \xrightarrow{\mathbb{P}} \mathcal{R}^*$$

A l'instar de la première partie, puisqu'on ne peut pas connaître la valeur du risque dans la pratique, nous allons devoir passer par une minimisation de l'approximation du risque .

**Définition 24** *On définit le risque empirique d'une règle  $\mathcal{R}_n$  de la manière suivante :*

$$\forall a \in \mathcal{A}, \mathcal{R}_n(a) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{a(x_i) \neq y_i\}}$$

Ainsi notre objectif devient alors de trouver  $a_n^* \in \arg \min_{a \in \mathcal{A}} \mathcal{R}_n(a)$  en espérant que  $\mathcal{R}(a_n^*) \approx \inf_{a \in \mathcal{A}} \mathcal{R}(a)$ . Nous verrons dans la suite de ce mémoire que l'obtention d'une telle décision dépend grandement de la forme de nos ensemble  $\mathcal{A}$  et  $\mathcal{F}$ .

#### iv Modélisation paramétrique

Si l'on ne connaît pas la vraisemblance des observations  $\mathbb{P}(X|Y)$  alors nous allons la paramétriser. Nous allons considérer que la vraisemblance des observations suit une loi de paramètre  $\theta \in \Theta$  avec  $\Theta$  un espace vectoriel de dimension finie. Pour cela nous allons poser les bases de l'estimation paramétrique.

Supposons que nous ayons un échantillon de  $n$  observations iid  $\mathcal{D}_x := (x_i)_{1 \leq i \leq n}$  d'une variable aléatoire  $X : \Omega \rightarrow \mathcal{X}$ , on suppose également que  $X$  suit une loi connue de paramètre  $\theta$  inconnu. Étant donnée la répartition de nos observations, nous souhaitons choisir le  $\theta$  qui maximise la vraisemblance d'observer nos données.

**Définition 25** *On appelle estimateur du maximum de vraisemblance l'estimateur  $\hat{\theta}^{MLE}$  qui maximise la probabilité d'observer nos données sachant le paramètre, soit :*

$$\hat{\theta}^{MLE} = \arg \max_{\theta^* \in \Theta} \mathbb{P}(\mathcal{D}_x | \theta = \theta^*)$$

*En sachant que nos données sont iid on a :*

$$\mathbb{P}(\mathcal{D}_x | \theta = \theta^*) = \mathbb{P}\left(\bigcap_{x_i \in \mathcal{D}_x} X = x_i | \theta = \theta^*\right) = \prod_{x_i \in \mathcal{D}_x} \mathbb{P}(X = x_i | \theta = \theta^*)$$

L'estimateur du maximum de vraisemblance peut être une bonne manière d'estimer le paramètre d'une loi inconnue lorsque l'on a aucune information sur le paramètre. Or, la pertinence de ce procédé peut être limitée lorsque l'on ne dispose pas d'un grand nombre d'observations. Dans ce cas, l'estimation Bayésienne peut se révéler très intéressante. Supposons par exemple que nous disposions d'un expert du domaine duquel nous avons extrait nos données et que cet expert ait l'intuition que le paramètre a une valeur approximativement égale à un certain  $\theta^*$  aléatoire.

**Définition 26** *Étant donné une fonction de coût  $L$ , on définit l'estimateur de Bayes par le réel  $\hat{\theta}^{Bayes}$  tel que :*

$$\hat{\theta}^{Bayes} = \arg \min_{\theta \in \Theta} \mathbb{E}[L(\theta, \theta^*)]$$

Nous avons maintenant définis toutes les bases théoriques qui nous serviront dans la réalisation de notre mémoire. Nous pouvons donc désormais commencer la résolution de notre problématique.

## 2 Construction de notre jeu de données

**Vous avez accès au code utilisé pour obtenir le résultat dans le notebook 'Preprocessing.ipynb'**

### 2.1 Sélection et fusion des jeux de données

Il existe un nombre gigantesque de bases de données regroupant des résultats de matchs de tennis et autres statistiques sur les joueurs. Trouver ces données n'a pas été difficile mais beaucoup des jeux de données étaient incomplets. Sélectionner les bons jeux de données n'est pas évident, nous avons dû nous y reprendre plusieurs fois au fur et à mesure du travail pour obtenir un jeu de données capable de nous fournir des résultats satisfaisants.

En effet, nous avons voulu combiner plusieurs jeux de données ce qui c'est avéré impossible, dans certains cas, car l'identification d'un match n'était jamais la même. Cependant grâce aux outils présents dans la bibliothèque Pandas de Python nous avons pu réaliser le tri de manière efficace.

Nous avons donc créé deux jeux de données, contenant pour chaque ligne les données d'un match, les plus complets possible. Le premier regroupant les matchs ayant eu lieu avant 2000 et le second entre 2000 et 2020. Nous nous intéresserons surtout au plus récent.

Nous avons modifié le jeux de données afin de pouvoir lui appliquer les différentes méthodes qui seront présentées dans la seconde partie.

### 2.2 Sélection des variables pertinentes

Nous partons de 20 jeux de données répertoriant 20 années de matchs de tennis. Dans cette base, les informations indiquées pour chaque match sont :

Le nom du tournois, la série du tournois (Grand slam, Masters, ATP500 ...), la ville où il a lieu, la date du match, la surface sur laquelle le match est joué (Hard, Clay, Grass, Carpet), l'étape du tournois (finale, demi-finale, etc), le nombre de sets qui doivent être joués au minimum durant le match.

Ainsi que, le nom du gagnant et du perdant, leur rang dans le classement ATP, le nombre de jeux gagnés par chaque joueur pour chaque set du match, le nombre de sets gagnés et les cotes attribuées à chaque joueur avant le match par différents organismes, ainsi que la moyenne de ses cotes et la cote maximale parmi toutes.

Les variables numériques peuvent être utilisées comme telles (ex : l'âge d'un joueur, son rang...) mais d'autres posent problème pour plusieurs algorithmes. Pour utiliser une variable telle que le nom des joueurs ou bien d'un tournois il faut pouvoir retranscrire ces informations au numérique sans que l'algorithme puisse comprendre un ordre dans ces noms. ( ex : trier les joueurs par ordre alphabétique et leur attribuer un numéro de 0 à n ne fonctionne pas).

Attribuer un numéro à chaque valeur d'une colonne ne peut être fait que lorsqu'il existe une hiérarchie entre les valeurs de cette colonne. Sinon, il faut trouver une autre méthode pour rendre les valeurs numérique.

**2.2.0.1 Suppression de certaines colonnes** Les informations qui sont utilisées au début de l'algorithme doivent être connues en amont du match. C'est pourquoi on ne lui donne pas accès aux scores.

Aussi, pour les variables qualitative, le choix de garder ou non cette variable dépend grandement du nombre de valeurs qu'elle peut prendre. En effet, en utilisant le one-hot-encoding, on ajoute autant de variables que de valeur dans la colonne et le nombre de variables total p ne peut pas être trop grand. Il s'agit donc d'évaluer la valeur de l'information apportée par cette variable face au coût qui correspond au nombre de colonnes que l'on doit ajouter au dataframe.

C'est pourquoi on supprime les colonnes indiquant le nom du tournois et la localisation qui rajouterais beaucoup trop de variables pour notre jeu de données.

## 2.3 Traitement des variables

**2.3.0.1 One-Hot-Encoding** Pour les variables qualitatives, nous avons dû utiliser la méthode du One-Hot encoding. En appliquant la commande "pandas.get\_dummies()" à une colonne de notre dataframe contenant d valeurs différente, d nouvelles colonnes sont créées. Elles sont remplies de 1 aux lignes où la valeur correspondante apparaît et de 0 sinon.

C'est une méthode lourde car elle augmente considérablement le nombre de colonnes.

On applique le one-hot-encoding aux colonnes :

- Joueur 1 et Joueur 0
- Series
- Court = intérieur ou extérieur
- Surface = pelouse, sol dur, synthétique ou terre battue
- Round = finale, demi-finale, etc

Pour certaine colonnes comme 'Round' on aurait pu attribuer des valeurs en fonction de l'importance du match ('Final' = 1, 'Demi-finale' = 2 ...) mais le one hot encoding fonctionne tout aussi bien et ça ne rajoute pas beaucoup de variables.

**2.3.0.2 Noms des joueurs** Le jeu de données de base comporte un trop grand nombre de joueurs différents. Nous avons donc décidé de les regrouper en fonction de leur rang comme suit : -Les joueurs ayant un classement entre 0 et 20 gardent leur nom -les joueurs classés entre 20 et 50 s'appelleront désormais tous 'classes-20-50' et idem pour les classés entre 50 et 100, 100 et 200, 200 et 500, 500 et 1000 et enfin une classe pour ceux classés au-delà de 1000.

Cela réduit considérablement le nombres de valeurs dans nos colonnes de gagnants et perdants.

Ensuite, il faut faire en sorte que l'algorithme ait accès aux noms des joueurs sans savoir lequel a gagné. Nous avons donc créé des colonnes 'player1' qui pour chaque match correspond au nom du joueur le mieux classé et 'player0' qui correspond au joueur avec le moins bon classement, enfin, une colonne 'winner' qui contient 1 si le joueur 1 gagne et 0 sinon et qui sera la colonne à prédire.

Enfin on applique le one-hot-encoding pour créer les colonnes 'player1\_nom' et 'player0\_nom' pour chaque joueur de la colonne player1 et player0.

**2.3.0.3 Dates** Pour les dates, on les sépare en trois colonnes "Year", "Month" et "Day", puis on leur donne un type numérique afin que les algorithmes puisse les lire.

Les années seront numérotées de 1 à 20 (au lieu de 2001 à 2020) et seront l'information dont disposeront la plupart des algorithmes en ce qui concerne les dates.

Il faut ensuite s'assurer que le type indiqué pour chaque colonne de notre base de données est désormais numérique.

On obtient une base de données idéale pour appliquer nos algorithmes.

### 3 Approche Machine Learning

#### 3.1 Première modélisation

Tout d'abord posons le cadre de notre modèle. Nous nous confrontons à un problème d'apprentissage supervisé de type "classification" où la variable à expliquer, que nous appellerons  $y$ , peut prendre deux valeurs : 0 ou 1. Nous considérerons que 0 représente l'évènement "le joueur 1 perd le match" et 1 l'évènement "le joueur 1 gagne le match".

Soit  $X$  la matrice regroupant toutes nos variables explicatives auxquelles nous avons ajouté un intercept :

$$X = \begin{pmatrix} 1 & x_1^1 & \dots & x_1^m \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 1 & x_n^1 & \dots & x_n^m \end{pmatrix} \text{ avec } n, m \in \mathbb{N} \text{ et } y \in \{0, 1\}^n$$

Où  $n$  est le nombre d'observations et  $m$  le nombre de variables explicatives. La première chose à laquelle nous pensons est alors la régression logistique.

#### 3.2 Régression Logistique ou "modèle logit"

Ce modèle est justement utilisé lorsque  $y$  est une variable réponse de type qualitative et binaire (ici match gagné ou perdu, 1 ou 0). Il a pour but de construire un prédicteur tel que si on nous donne une nouvelle observation  $x_{n+1}$ , on soit en mesure de prédire  $y_{n+1}$ . Le but est donc de déterminer le vecteur de paramètres  $\theta = (\theta_0, \dots, \theta_m)^T \in \mathbb{R}^{m+1}$  tel que  $h_\theta(x_{n+1}) = g(\theta^T x_{n+1}) = \mathbb{P}_\theta(y = 1|x_{n+1})$  soit le plus proche possible de  $y$ , où  $g$  est appelée fonction de lien réciproque. Dans le modèle logit, elle est définie par  $g(x) = \frac{\exp^x}{\exp^x + 1} = \frac{1}{1 + \exp^{-x}}$  qui est en fait la fonction sigmoïde. On laissera pour l'instant variable le seuil de décision  $s$  afin de pouvoir se laisser le choix du niveau de confiance pour prédire l'issue du match. Pour rappel le seuil de décision  $s$  est défini par :

$$\begin{cases} \text{"On prédit } y=1 \text{" si } h_\theta(x) \geq s \\ \text{"On prédit } y=0 \text{" si } h_\theta(x) < s \end{cases}$$

Revenons à l'objectif premier : déterminer le vecteur de paramètres  $\theta$  permettant de construire notre prédicteur  $h_\theta(x) = \mathbb{P}_\theta(y = 1|x)$ . On remarque donc que  $y$  suit, sous  $\theta$ , une loi de Bernoulli de paramètre  $p = h_\theta(x) = \frac{1}{1 + \exp^{-x^T \theta}}$ . Ainsi, trouver  $\theta$  maximisant la fonction de vraisemblance des  $y$  de notre jeu de données permettra d'obtenir le meilleur vecteur de paramètres  $\theta$  permettant de construire notre prédicteur. Nous cherchons donc à résoudre :

$$\begin{aligned} \arg \max_{\theta} \mathcal{L}(y|\theta) &= \arg \max_{\theta} (\log \mathcal{L}(y|\theta)) \\ &= \arg \max_{\theta} \left( \log \prod_{i=1}^n p^{y_i} (1-p)^{1-y_i} \right) \\ &= \arg \min_{\theta} -\log \left( \prod_{i=1}^n p^{y_i} (1-p)^{1-y_i} \right) \\ &= \arg \min_{\theta} -\sum_{i=1}^n (\log(p^{y_i} (1-p)^{1-y_i})) \\ &= \arg \min_{\theta} -\sum_{i=1}^n y_i \log(h_\theta(x_i)) + (1-y_i) \log(1-h_\theta(x_i)) \end{aligned} \tag{1}$$

Ce qui revient à trouver  $\theta$  minimisant :

$$-\frac{1}{n} \sum_{i=1}^n [y_i \log(h_\theta(x_i)) + (1-y_i) \log(1-h_\theta(x_i))] = J(\theta)$$

Pour cela nous allons utiliser l'algorithme de descente de gradient défini par récurrence de la façon suivante.

$$\{\text{Tant que } J(\theta) > \epsilon \text{ répéter } \forall j \in \{0, \dots, m\} : \} \quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Calculons alors  $\frac{\partial}{\partial \theta_j} J(\theta)$  :

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{1}{n} \sum_{i=1}^n \left[ y_i \frac{\partial \log(h_\theta(x_i))}{\partial \theta_j} + (1-y_i) \frac{\partial \log(1-h_\theta(x_i))}{\partial \theta_j} \right] \\
&= -\frac{1}{n} \sum_{i=1}^n \left[ y_i \frac{\partial \log(\frac{1}{1+\exp^{-\theta^T x_i}})}{\partial \theta_j} + (1-y_i) \frac{\partial \log(1-\frac{1}{1+\exp^{-\theta^T x_i}})}{\partial \theta_j} \right] \\
&= -\frac{1}{n} \sum_{i=1}^n \left[ y_i \frac{-\partial \log(1+\exp^{-\theta^T x_i})}{\partial \theta_j} + (1-y_i) \frac{\partial(-\theta^T x_i - \log(1+\exp^{-\theta^T x_i}))}{\partial \theta_j} \right] \\
&= -\frac{1}{n} \sum_{i=1}^n \left[ -y_i \frac{-x_i^j \exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}} + (1-y_i)(-x_i^j - \frac{-x_i^j \exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}}) \right] \\
&= -\frac{1}{n} \sum_{i=1}^n x_i^j \left[ \frac{y_i \exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}} + y_i - 1 + \frac{(1-y_i) \exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}} \right] \\
&= -\frac{1}{n} \sum_{i=1}^n x_i^j \left[ \frac{y_i \exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}} - \frac{1+\exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}} + \frac{(1-y_i) \exp^{-\theta^T x_i}}{1+\exp^{-\theta^T x_i}} + y_i \right] \\
&= -\frac{1}{n} \sum_{i=1}^n x_i^j \left[ -\frac{1}{1+\exp^{-\theta^T x_i}} + y_i \right] = \frac{1}{n} \sum_{i=1}^n x_i^j \left[ \frac{1}{1+\exp^{-\theta^T x_i}} - y_i \right] = \frac{1}{n} \sum_{i=1}^n x_i^j [h_\theta(x_i) - y_i]
\end{aligned} \tag{2}$$

La convergence de cet algorithme se justifie intuitivement par le fait que le gradient d'une fonction pointe vers ses lignes de niveau associées à des plus grandes valeurs, ainsi l'inverse du gradient permettra d'approcher un minimum de la fonction  $J$  en un certain nombre d'itérations (selon le taux d'apprentissage  $\alpha$  choisi et la précision  $\epsilon$  souhaitée).

Très rapidement nous rencontrons un problème assez connu et courant en machine learning : Certaines de nos variables explicatives prennent des valeurs trop grandes relativement à d'autres, nous sommes confrontés à des problèmes d'échelle. Ces derniers ralentissent la convergence de l'algorithme de descente de gradient et le rendent moins performant. Pour éviter ce problème, il faut effectuer au préalable sur notre jeu de données une "renormalisation" de nos variables explicatives appelée "Feature scaling".

### 3.2.1 Feature scaling

Il existe plusieurs techniques de renormalisation. Nous allons essayer les trois principales et conserver la plus performante pour la suite.

**NB** : Il est important de garder en tête lors de l'implémentation qu'on renormalise toutes nos variables mais pas l'intercept. Le renormaliser aurait pour conséquence de transformer l'intercept en une colonne de 0, ce qui n'a pas de sens. Dans la suite on appellera  $x$  une variable explicative quelconque.

#### i Min-max normalization

$$x' = \frac{x - \min x}{\max x - \min x}$$

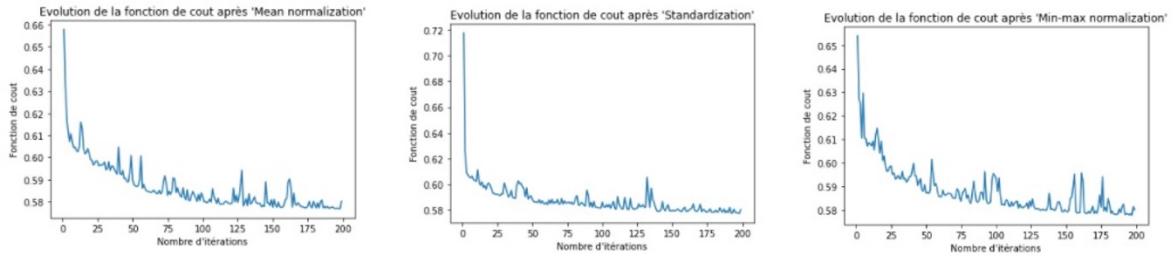
#### ii Mean normalization

$$x' = \frac{x - \bar{x}}{\max x - \min x}$$

### iii Standardization

$$x' = \frac{x - \bar{x}}{\sigma}$$

### iv Conclusion

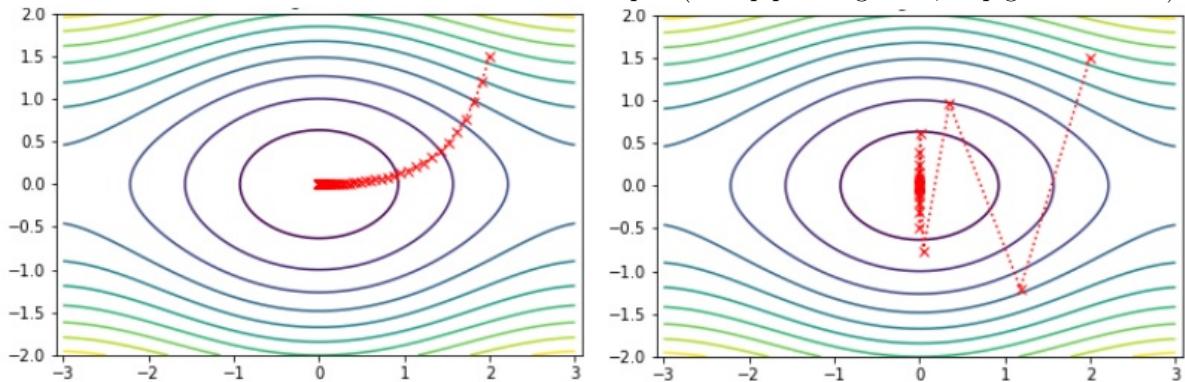


Nous nous apercevons que c'est la technique de 'Standardization' qui nous donne de meilleurs résultats dans le cadre de l'application d'une descente de gradient (où le taux d'apprentissage est fixé arbitrairement). Au vue de ces résultats, il paraît donc intéressant de garder cette renormalisation pour la suite afin de ne pas subir les potentielles variations d'échelle entre les variables explicatives lors de l'apprentissage du modèle.

Il semble désormais intéressant de se pencher sur le choix du taux d'apprentissage  $\alpha$  au travers de la question suivante : qu'est ce qu'un "bon  $\alpha$ " ?

#### 3.2.2 Recherche linéaire

La recherche linéaire est la façon dont on choisit le taux d'apprentissage  $\alpha$ . Nous allons voir qu'il en existe plusieurs et nous les présenterons après une bref discution autour du choix de  $\alpha$  : le taux d'apprentissage ne peut pas être choisi au hasard. Si il est pris trop grand alors l'algorithme ne va pas converger vers le minimum et peut même, dans le pire des cas, s'en éloigner. Inversement, si il est choisi trop petit alors la convergence sera très lente. Ceci est très bien illustré par ces deux figures tirées du cours de L3 de Monsieur Gontier - Méthodes numériques ( $\alpha$  trop petit à gauche, trop grand à droite) :



Nous allons donc voir comment apporter une réponse à cette problématique.

#### i Recherche par tâtonnements

Cela consiste à tracer l'évolution de la fonction de coût au cours des itérations de l'algorithme de descente de gradient, pour plusieurs valeur de  $\alpha$ , afin d'approcher et de conserver une valeur qui semble être la plus adapté au problème.

C'est la méthode que nous avons choisi pour une première approche dans la section précédente. Bien qu'elle soit très populaire sur internet, nous allons tenter de trouver plus rigoureux.

## ii Recherche linéaire exacte : Gradient pas optimal

L'idée est de choisir automatiquement le pas optimal à chaque itération. On va alors adapter notre algorithme de descente de gradient défini plus haut et le réécrire comme suit :

$$\{\text{Tant que } J(\theta^n) > \epsilon \text{ répéter } \forall j \in \{0, \dots, m\} : \} \quad \theta_j^{n+1} := \theta_j^n - \alpha^n \frac{\partial}{\partial \theta_j^n} J(\theta^n) \quad (3)$$

Où  $\theta^n$  est le nouveau vecteur de paramètres obtenu avec la  $n^{\text{ième}}$  itération de notre descente de gradient à pas optimal et  $\alpha^n$  le pas optimal de la  $n^{\text{ième}}$  itération. Ce pas optimal, pour la  $n^{\text{ième}}$  itération, est défini par :

$$\alpha^n = \arg \min \{J(\theta^n - \alpha \nabla J(\theta^n)), \alpha \in \mathbb{R}_+\} \quad (4)$$

Mais en quoi ce pas est-il optimal ? Quel est l'intérêt d'un tel choix de  $\alpha$  ?

Si  $\alpha^n$  est solution du problème ci-dessus alors  $\alpha^n$  est solution de :

$$\frac{\partial}{\partial \alpha} J(\theta^n - \alpha \nabla J(\theta^n)) = 0$$

Or

$$\frac{\partial}{\partial \alpha} J(\theta^n - \alpha \nabla J(\theta^n)) = < \frac{\partial}{\partial \alpha} (\theta^n - \alpha \nabla J(\theta^n)), \nabla J(\theta^n - \alpha \nabla J(\theta^n)) > = < -\nabla J(\theta^n), \nabla J(\theta^n - \alpha \nabla J(\theta^n)) >$$

$\alpha^n$  vérifie donc :

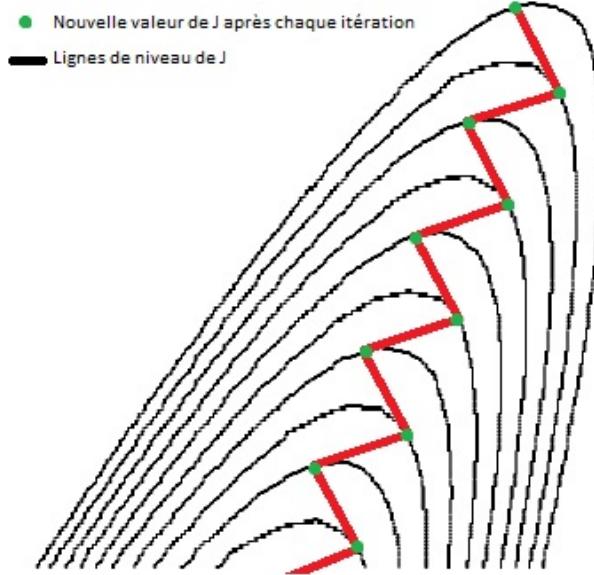
$$< \nabla J(\theta^n), \nabla J(\theta^n - \alpha^n \nabla J(\theta^n)) > = 0$$

Or  $\theta^{n+1} := \theta^n - \alpha^n \nabla J(\theta^n)$  d'après (3), ainsi :

$$< \nabla J(\theta^n), \nabla J(\theta^{n+1}) > = 0$$

On peut en déduire qu'un tel choix de  $\alpha$  permet donc d'avoir une relation d'orthogonalité entre les gradients de chaque itération. Une simple illustration permet de se rendre compte de l'avantage en terme de rapidité de convergence que nous apporte un tel résultat :

Illustration d'une descente de gradient à pas optimal



Maintenant que nous avons conscience de la pertinence d'un tel choix de  $\alpha$ , il nous faut le déterminer. Nous avons ici affaire à un problème d'optimisation en dimension 1 : (4). Nous allons pouvoir utiliser la méthode de la section dorée, étudiée en L3, afin d'approcher ce  $\alpha$  optimal. Le seul problème est

que  $\alpha \in \mathbb{R}_+$  qui n'est pas compact. Nous supposerons cependant ici, plutôt logiquement, que le taux d'apprentissage ne peut pas être supérieur à 1 000 (ce dernier étant très souvent inférieur à 1, cette marge semble confortable). Cela nous ramène donc à un problème d'optimisation d'une fonction continue (par composition de fonctions continues) sur un compact :  $[0; 1000]$ . Nous allons donc pouvoir appliquer la méthode de la section dorée comme suit : **(nous reviendrons ultérieurement sur la problématique d'unimodalité de la fonction  $J(\theta^n - \alpha \nabla J(\theta^n))$ )**

- On note  $\xi = \frac{\sqrt{5}-1}{2}$  le nombre d'or.
  - On dit que le triplet  $x < y < z \in [0; 1000]$  est admissible si :
- $$J(\theta^n - y \nabla J(\theta^n)) < \min\{J(\theta^n - x \nabla J(\theta^n)), J(\theta^n - z \nabla J(\theta^n))\}$$
- Et on définit par récurrence le quadruplets avec lesquels nous allons travailler :

$$(a_{n+1}, c_{n+1}, d_{n+1}, b_{n+1}) = \begin{cases} (a_n, x, c_n, d_n) & \text{si } (a_n, c_n, d_n) \text{ est admissible} \\ (c_n, d_n, x, b_n) & \text{si } (c_n, d_n, b_n) \text{ est admissible} \end{cases}$$

Avec  $x$  défini par :

$$x = \begin{cases} \xi a_n + (1 - \xi) d_n & \text{si } (a_n, c_n, d_n) \text{ est admissible} \\ \xi b_n + (1 - \xi) c_n & \text{si } (c_n, d_n, b_n) \text{ est admissible} \end{cases}$$

Ainsi nous appliquerons cette méthode à chaque itération pour obtenir une approximation de  $\alpha^n$ . Maintenant que nous avons un taux d'apprentissage "optimal" (en réalité nous avons approché l'optimal donc il devrait plutôt être qualifié de "quasi optimal", mais nous reviendrons sur ce point à la fin de la prochaine sous-section), ne pourrait-on pas, au lieu d'utiliser la direction de descente classique  $-\nabla J(\theta)$  de l'algorithme de descente de gradient, utiliser une direction de descente plus "optimale". C'est justement l'objectif de la méthode BFGS qui est une méthode actuelle très utilisée et qu'on retrouve très souvent dans les librairies permettant de traiter des problèmes du type "logit".

### iii Méthode de Broyden Fletcher Goldfarb Shanno

L'idée est d'améliorer la direction de descente en introduisant une approximation de la hessienne de  $J$  en  $\theta$  de la façon suivante :  $-H^{-1}\nabla J(\theta_n)$  où  $H$  est la Hessienne de la fonction  $J$  en  $\theta$  qui sera approchée à chaque itération par  $B_k$  (pour la  $k^{\text{ième}}$  itération). Mais pourquoi ? Rappelons que nous cherchons maintenant à chaque itération deux choses :  $\alpha_k$  le taux d'apprentissage (obtenu, pour l'instant, par la méthode du gradient pas optimal, mais nous verrons dans la section suivante qu'il en existe d'autres) et la direction de descente  $d_k$  (qui était jusqu'à présent  $-\nabla J(\theta)$ ) que nous allons chercher à déterminer tel que :

$$\theta_{k+1} = \theta_k - \alpha_k d_k$$

Qui peut se réécrire :

$$\theta_{k+1} - \theta_k = -\alpha_k d_k$$

L'idée qui motive la définition de  $d_k$  par  $-H^{-1}\nabla J(\theta_n)$  provient du développement de Taylor suivant :

$$J(\theta_{k+1}) \simeq J(\theta_k) + \nabla J(\theta_k)^T(\theta_{k+1} - \theta_k) + \frac{1}{2}(\theta_{k+1} - \theta_k)^T \nabla^2 J(\theta_k)(\theta_{k+1} - \theta_k)$$

Qu'on peut appliquer au gradient de la fonction  $J$  et obtenir :

$$\nabla J(\theta_{k+1}) \simeq \nabla J(\theta_k) + \nabla^2 J(\theta_k)(\theta_{k+1} - \theta_k) \quad (5)$$

ou encore :

$$\nabla J(\theta_{k+1}) - \nabla J(\theta_k) \simeq \nabla^2 J(\theta_k)(\theta_{k+1} - \theta_k)$$

Ainsi d'après (5) on obtient la relation dite de "quasi-Newton" suivante :

$$\theta_{k+1} - \theta_k \simeq [\nabla^2 J(\theta_k)]^{-1}(\nabla J(\theta_{k+1}) - \nabla J(\theta_k))$$

Et la méthode consistera en la construction, à chaque itération, d'une matrice  $B_k$  définie par récurrence telle que  $d_k = -B_k^{-1}\nabla J(\theta_k)$  et :

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

vérifiant la relation de "quasi-Newton" :  $\theta_{k+1} - \theta_k = B_{k+1}^{-1}(\nabla J(\theta_{k+1}) - \nabla J(\theta_k))$ , avec  $y_k = \nabla J(\theta_{k+1}) - \nabla J(\theta_k)$ ,  $s_k = \alpha_k d_k$  et où dans la pratique nous initialiserons  $B_0$  comme étant la matrice identité (et on remarquera alors que la première itération est équivalente à une descente de gradient pas optimal classique).

Ainsi, nous avons, grâce aux deux dernières sous-sections, un choix de taux d'apprentissage et de direction de descente optimisé.

**Remarque :** On remarque cependant que le taux d'apprentissage n'est pas littéralement "optimal" puisqu'on l'approche avec la méthode de la section dorée, comme vu précédemment. On dira que notre pas est plutôt "quasi-optimal". Il existe alors de nombreuses autres méthodes pour trouver un pas "quasi-optimal" dont la plus connue est celle de Wolfe-Armijo. Nous nous proposons donc de l'étudier dans la prochaine sous-section.

#### iv Gradient pas quasi-optimal : critère de Wolfe-Armijo

Intuitivement le critère d'Armijo permet de choisir un pas "pas trop grand" et Wolfe "pas trop petit". L'objectif est donc de combiner les deux. (Pour rappel nous cherchons à approcher une solution de (4)). Notons  $\Phi_{\theta^n, -\nabla J(\theta^n)}(\alpha) = J(\theta^n - \alpha \nabla J(\theta^n))$ . On dit que  $\alpha$  satisfait la règle d'Armijo pour le paramètre  $0 < k < 1$  si  $\Phi : \mathbb{R}_+ \rightarrow \mathbb{R}$  est telle que  $\Phi'(0) < 0$  et :

$$\Phi_{\theta^n, -\nabla J(\theta^n)}(\alpha) \leq \Phi_{\theta^n, -\nabla J(\theta^n)}(0) + k\alpha\Phi'_{\theta^n, -\nabla J(\theta^n)}(0)$$

Dans notre cas nous avons bien que  $\Phi'(0) < 0$  car :

$$\begin{aligned} \frac{\partial}{\partial \alpha} \Phi_{\theta^n, -\nabla J(\theta^n)}(\alpha) |_{\alpha=0} &= \frac{\partial}{\partial \alpha} J(\theta^n - \alpha \nabla J(\theta^n)) |_{\alpha=0} \\ &= < \frac{\partial}{\partial \alpha} (\theta^n - \alpha \nabla J(\theta^n)), \nabla J(\theta^n - \alpha \nabla J(\theta^n)) > |_{\alpha=0} \\ &= < -\nabla J(\theta^n), \nabla J(\theta^n - \alpha \nabla J(\theta^n)) > |_{\alpha=0} \\ &= < -\nabla J(\theta^n), \nabla J(\theta^n) > \\ &= -\|\nabla J(\theta^n)\|^2 < 0 \end{aligned} \tag{6}$$

Par ailleurs, on dira que  $\alpha$  satisfait la règle de Wolfe pour les paramètres  $0 < k < w < 1$  s'il satisfait Armijo pour le paramètre  $k$ , que  $\Phi : \mathbb{R}_+ \rightarrow \mathbb{R}$  est telle que  $\Phi'(0) < 0$  (ce qui est vrai d'après (6)), et si :

$$\Phi'_{\theta^n, -\nabla J(\theta^n)}(\alpha) \geq w\Phi'_{\theta^n, -\nabla J(\theta^n)}(0)$$

Afin de satisfaire Wolfe et Armijo, nous allons maintenant appliquer la méthode de dichotomie pour trouver notre taux d'apprentissage.

Soit un couple  $(\alpha_a, \alpha_w)$ , tels que  $\alpha_a < \alpha_w$ , deux taux d'apprentissage satisfaisant respectivement Armijo et Wolfe, en posant  $\alpha_d = \frac{\alpha_a + \alpha_w}{2}$ , nous allons définir notre algorithme de dichotomie de la façon suivante :

{Tant que  $\alpha_w - \epsilon$  ne satisfait pas Armijo répéter :}

$$(\alpha_a, \alpha_w) = \begin{cases} (\alpha_d, \alpha_w) & \text{si } \alpha_d \text{ vérifie Armijo} \\ (\alpha_a, \alpha_d) & \text{sinon} \end{cases}$$

Pour une précision  $\epsilon$  souhaitée. Une fois cette boucle "Tant que" terminée nous obtiendrons notre taux d'apprentissage  $\alpha = \alpha_w - \epsilon$  satisfaisant Wolfe et Armijo.

**Remarque :** Comme mentionné dans la section précédente nous n'avons pas cherché à savoir si  $\Phi_{\theta^n, -\nabla J(\theta^n)}(\alpha)$  était unimodale (strictement convexe) sur le compact  $[0; 1000]$ . Ceci implique que notre solution approchée par la méthode de la section dorée précédemment peut converger vers un minimum locale de la fonction  $\Phi_{\theta^n, -\nabla J(\theta^n)}(\alpha)$ . On peut donc en conclure qu'on préfèrera l'utilisation du critère de Wolfe-Armijo, qui contourne ce problème, sauf si on sait que  $\Phi_{\theta^n, -\nabla J(\theta^n)}(\alpha)$  est unimodal sur  $[0; 1000]$ .

**2<sup>ème</sup> Remarque :** En prenant du recul, nous pouvons même nous apercevoir que nous retrouvons

ce problème au "niveau supérieur". Que se passe-t-il si, même avec un bon choix de  $\alpha$ , notre algorithme de descente de gradient reste coincé dans un minimum locale de la fonction de coût  $J$ , comment éviter cela ?

Même si dans la littérature, ou autres cours en ligne, il est souvent mentionné qu'il n'est "pas si grave" d'aboutir à un minimum locale de la fonction  $J$ , sachant qu'elle est directement liée à la qualité de notre prédiction, il paraît plus rigoureux de chercher à atteindre son minimum global. Pour cela un bon choix d'initialisation du vecteur de paramètres  $\theta$  est la clé.

### 3.2.3 Choix du point de départ

#### i Initialisation aléatoire

Le but est ici d'initialiser tous les poids de manière uniforme sur  $[-\epsilon; \epsilon]$  pour un  $\epsilon$  choisi. Pour cela à chaque poids nous affecterons la valeur  $(U * 2\epsilon) - \epsilon$  où  $U$  sera un tirage d'une uniforme sur  $[0, 1]$ . C'est la technique la plus populaire pour initialiser une descente de gradient. Le problème est qu'elle n'est pas compatible avec notre dernière remarque. Nous allons donc maintenant voir comment nous pourrions initialiser  $\theta$  pour apporter une réponse à ce problème.

#### ii Initialisation pré-déterminée

Après avoir consulté plusieurs de nos professeurs Dauphinois il n'existe pas de méthodes générales pour l'initialisation d'une descente de gradient qui garantirait une convergence vers le minimum global mais une forte recommandation pour s'assurer de l'approcher au mieux. Elle consiste à initialiser notre vecteur de paramètres avec des réels compris entre -1 et 1 selon notre information à priori sur la corrélation entre la variable explicative associée concernée et la variable à expliquer. Enfin on divisera ces valeurs par la racine du nombre de variables explicatives + 1 (intercept). On aura donc :

$$\theta_0 = \left( \frac{c_0}{\sqrt{m+1}}, \dots, \frac{c_m}{\sqrt{m+1}} \right)^T \in \mathbb{R}^{m+1}$$

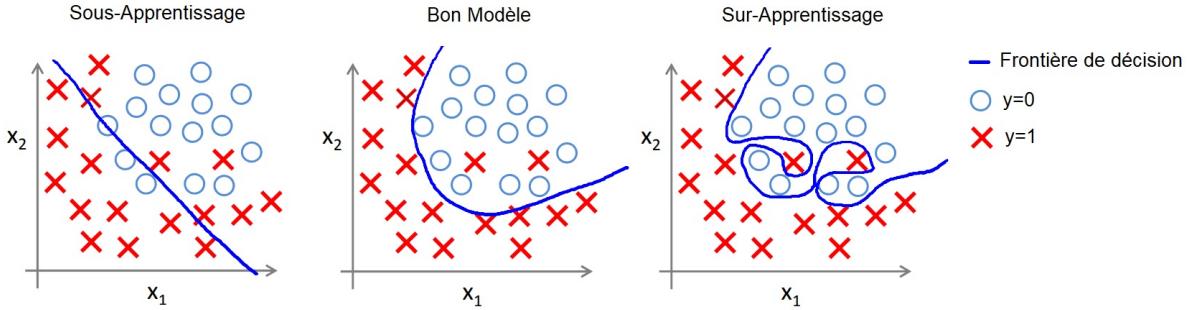
où  $c_i$  avec  $i \in \{0, \dots, m\}$ , représente la corrélation à priori entre la  $i^{\text{ème}}$  variable explicative et la variable à expliquer.

### 3.2.4 Sur-apprentissage et choix des variables explicatives

Après avoir remarqué que notre frontière de décision n'était pas linéaire nous avons dû introduire des variables explicatives polynomiales comme combinaison de nos variables explicatives existante. Plusieurs questions se posent alors : Quel degré choisir pour chaque variable ? QUID des combinaisons du type  $x_i x_j$  avec  $i \neq j$  ? Et surtout comment éviter le sur-apprentissage ?

#### i Choix du bon modèle

L'objectif de cette section est de choisir un modèle ni trop complexe qui sur-apprendrait des données de notre échantillon d'apprentissage (à l'image du graphique de droite, "on apprend du bruit", variance importante), ni trop simple qui sous-apprendrait de notre échantillon d'apprentissage (à l'image du graphique de gauche, "modélisation trop grossière", biais important). Pour illustrer cette problématique nous allons commencer avec un exemple simple d'un modèle où on chercherait à prédire  $y$  avec seulement deux variables  $x_1$  et  $x_2$  :



Pour l'instant nous sommes dans le premier cas de figure, c'est à dire que, n'ayant ajouter aucune puissance de variables explicatives existante, nous ne pouvons pas faire mieux qu'une classification linéaire. Le but est d'en introduire afin de pouvoir approcher le bon modèle sans pour autant sur-apprendre de nos données d'apprentissage et tomber dans le sur-apprentissage. La question qu'on se pose alors est : Comment savoir quelle combinaison de nos variables explicatives ajouter ?

Pour répondre à cette problématique nous allons adopter la démarche suivante : Par principe de parcimonie on commence par fixer le degré de chaque variable explicative à 1.

- Si le modèle est évalué comme pertinent alors on conserve ce choix.
- Si on sur-apprend des données : erreur sur l'échantillon d'apprentissage faible mais élevée sur l'échantillon de test alors on reprend l'étape précédente et on revoit quelles variables on pourrait retirer du modèle.
- Si on sous-apprend : erreur forte sur l'échantillon d'apprentissage et de test, alors on ajoute les puissances/ passage au logarithme/ autres transformations des variables explicatives qui ont les plus gros coefficients correspondant en valeur absolue dans notre vecteur de paramètres.
- On réitère la procédure jusqu'à se trouver légèrement au delà du bon modèle, à la limite du sur-apprentissage.

Préférant un modèle qui sur-apprend des données plutôt que l'inverse, il va désormais falloir corriger cela avec un autre levier. Le plus adapté est la modification de notre fonction de coût. Le principe sera d'introduire une pénalisation afin de réduire l'influence des variables responsables du sur-apprentissage au sein de notre modèle. On va donc introduire une pénalisation L2 : ajout, à la fonction de coût, de la norme 2 de notre vecteur de paramètres multiplié par une constante  $\lambda$  que l'on devra choisir ; Communément appelée "Pénalisation Ridge".

## ii Pénalisation Ridge

Suivant ce que nous venons de dire, la nouvelle fonction de coût sera donnée par :

$$J^{\{R\}}(\theta) = \left( -\frac{1}{n} \sum_{i=1}^n [y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))] \right) + \underbrace{\frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2}_{||\theta||_2^2} \quad (7)$$

Où on a effectivement ajouté la norme 2 de  $\theta$  au carré, avec un nouveau paramètre à déterminer  $\lambda$  et une division par  $2m$  qui est une convention mais n'affecte en rien la minimisation de la fonction de coût puisque  $m$  est un nombre fixe : le nombre de variables explicatives dans notre modèle.

Ainsi l'algorithme de descente de gradient devient :

$$\{ \text{Tant que } J^{\{R\}}(\theta) > \epsilon \text{ répéter } \forall j \in \{0, \dots, m\} : \} \quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J^{\{R\}}(\theta)$$

Où, d'après (2),  $\frac{\partial}{\partial \theta_j} J^{\{R\}}(\theta)$  est donné par :

$$\begin{cases} \frac{\partial}{\partial \theta_0} J^{\{R\}}(\theta) = \frac{1}{n} \sum_{i=1}^n x_i^0 [h_\theta(x_i) - y_i] & \text{pour } j=0 \\ \frac{\partial}{\partial \theta_j} J^{\{R\}}(\theta) = \frac{1}{n} \sum_{i=1}^n x_i^j [h_\theta(x_i) - y_i] + \frac{\lambda}{m} \theta_j & \text{sinon} \end{cases}$$

Il nous reste donc plus qu'une dernière chose à déterminer avant d'obtenir notre modèle final :  $\lambda$ .

### iii Validation croisée V-fold

Dans cette section nous allons nous intéresser au choix de la constante de pénalisation  $\lambda$ .

Tout d'abord remarquons que le biais est une fonction croissante de  $\lambda$  : plus  $\lambda$  sera grand plus on pénalise les éléments du vecteur  $\theta$  et plus on tend vers un modèle réduit à l'intercept, qui sous-apprend et donc a un fort biais. D'un autre côté la variance une fonction décroissante de  $\lambda$  pour la raison inverse : plus  $\lambda$  est petit, moins on pénalise les éléments de  $\theta$  et donc plus on est sous le joug du sur-apprentissage, et donc une forte variance. L'objectif est donc de trouver un compromis biais variance et ainsi obtenir le  $\lambda$  optimal :

$$\lambda^* = \arg \min_{\lambda \geq 0} \{\beta_\lambda^{(R)} + V_\lambda^{(R)}\}$$

L'idée de la validation croisée V-fold est alors la suivante :

Soit  $\Omega \subset [0; +\infty[$  un ensemble fini, appelé grille, de valeurs possible de  $\lambda$ , nous allons partitionner l'ensemble des observations en  $V$  ensembles disjoints de taille égale :  $I_1, \dots, I_V$ . On va ensuite implémenter l'algorithme suivant :

{Pour  $v = 1, \dots, V$  :}

{Pour  $\lambda$  dans  $\Omega$  :}

Calculer  $\theta_\lambda^{-v}$  à partir de  $\{(X_i, y_i), i \notin I_v\}$  qui minimise le biais sur cette ensemble

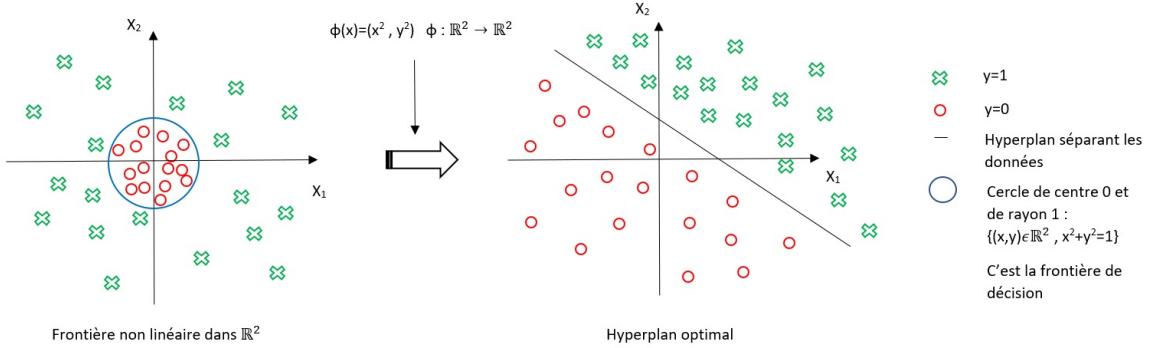
$\forall i \in I_v$  calculer la prediction  $\hat{y}_i^{(\lambda)} = x_i^T \hat{\theta}_\lambda^{(-v)}$  qui nous permet de quantifier la variance sur un échantillon

test artificiellement créé  $I_v$ , en calculant l'erreur de prédiction.

Ainsi, nous allons sélectionner le  $\lambda$  qui conduit **en moyenne** à la prédiction ayant la plus petite erreur de prédiction et nous aurons notre  $\lambda$  optimal parmi les valeurs possible de  $\lambda$  de notre grille ce qui conclut notre chapitre sur la régression logistique.

### iv Transition vers une approche plus géométrique

A l'image des graphiques présentant les problématiques de sur/sous-apprentissage de la section "Choix du bon modèle", nous remarquons qu'il est possible d'aborder notre problème d'un point de vue plus géométrique. L'idée serait, plutôt que de chercher à déterminer une relation linéaire, combinée à la fonction sigmoid, entre nos variables explicatives (d'origines & celles créées au cours de la section "Choix du modèle") et la variable à expliquer, de trouver une frontière de décision permettant de classifier de part et d'autre une étiquette négative ou positive (correspondant respectivement à  $y=0$  et  $y=1$ ). Cette approche est celle de la classification à vaste marge (ou SVM de son nom d'origine anglophone). Pour des raisons pratiques, qui feront sens au fil des prochaines pages, nous redéfinissons le support de  $y$  par l'ensemble  $\{-1; 1\}$ , où -1 correspond donc à la réponse  $y=0$  et 1 à  $y=1$ . L'idée général est, plutôt que de construire directement une fonction de prédiction  $h : X \rightarrow \{-1; 1\}$ , de construire une fonction  $f : X \rightarrow \mathbb{R}$  pour laquelle on regardera le signe qui nous donnera alors notre prédiction. La construction de cette fonction  $f$  passe d'abord par une transformation  $\phi$  des données de l'espace de départ  $X$  en vecteur dans un espace  $\mathbb{F}$  appelé "Feature space". Ainsi  $\phi : X \rightarrow \mathbb{F}$ , où  $\mathbb{F}$  est un espace de Hilbert pas nécessairement de dimension finie. Le but est de traiter la non linéarité du problème via cette transformation pour pouvoir se ramener à une séparation linéaire du type "si  $f$  négatif je predis une étiquette négative" / "si  $f$  positif je predis une étiquette positive". Pour illustrer ce principe voici ce en quoi cela consiste en dimension 2 :



Plusieurs questions se posent alors : Comment déterminer  $\phi$ ? Comment trouver l'hyperplan séparant les données? Comment éviter le sur-apprentissage avec cette approche? ...

C'est autant de questions auxquelles nous nous proposons de répondre dans la partie qui suit.

### 3.3 Support Vector Machine (SVM)

Dans cette partie on se place dans  $\mathcal{X} \times \mathcal{Y} := (\mathbb{R}^p, d_2) \times \{-1, 1\}$  et on notera sauf indications contraire  $\|\cdot\| := \|\cdot\|_2$ . On se donne un jeu de données  $\mathcal{D} = (x_i, y_i)_{1 \leq i \leq n}$  que l'on considère indépendantes.

Nous allons étudier les *machines à vecteurs de support* aussi appelé *SVM*. Cet algorithme d'apprentissage est née de la coopération entre Vladimir Vapnik et Aleksandr Lerner en 1963. Leur but était de proposer un prédicteur linéaire capable d'apprendre en haute dimension. Le principe est plutôt simple, il s'agit de séparer les observations de notre jeu de données (de dimension  $p$ ) par un hyperplan (de dimension  $p - 1$ ). Pour que cette opération soit possible, il faudrait que nos données soit *séparable de manière linéaire* par un hyperplan. Nous allons tacher de formaliser ce principe de manière rigoureuse.

#### 3.3.1 Séparabilité linéaire

**Définition 27** On note  $\mathcal{H}(w, b) := \{v \in \mathbb{R}^p \mid \langle w, v \rangle + b = 0\}$  l'hyperplan définit par le couple  $(w, b) \in \mathbb{R}^p \times \mathbb{R}$ . Alors on dit d'un jeu de données  $\mathcal{D}$  qu'il est séparable linéairement si et seulement si il existe un hyperplan  $\mathcal{H}(w, b)$  tel que :

$$\begin{aligned} \forall i \leq n, y_i &= \operatorname{sgn}(\langle w, x_i \rangle + b) \\ \Leftrightarrow \forall i \leq n, y_i &(\langle w, x_i \rangle + b) \geq 0 \end{aligned}$$

Intuitivement, cela signifie qu'il existe un hyperplan tel que toutes les observations du jeu de données étiquetées respectivement positivement (+) et négativement (-) se trouvent d'une part et d'autre de l'hyperplan.

Malheureusement, cette définition ne garantit pas l'unicité d'un tel hyperplan. La réalité est pire car en absence d'informations ou de contraintes supplémentaires, il existe une infinité d'hyperplans satisfaisant ces conditions. Notre but va alors être de poser une condition sur notre hyperplan afin de choisir celui qui sépare nos données de manière optimale. Pour cela, nous allons introduire la notion de marge.

**Définition 28** Soit  $\mathcal{H}$  un hyperplan quelconque. On définit  $\gamma$  la marge de l'hyperplan comme étant la distance minimale entre les observations de notre jeu de données et l'hyperplan satisfaisant les conditions de séparabilité :  $y_i(\langle w, x_i \rangle + b) \geq 0$ .

**Définition 29** On appelle vecteur de support tout vecteur se trouvant à une distance  $\gamma$  de l'hyperplan séparateur.

Une première idée pour le choix de notre hyperplan optimale, serait de prendre l'hyperplan qui possède la plus grande marge avec notre jeu de données. En effet, intuitivement plus la marge entre l'hyperplan et nos données est élevée et plus il sépare nos données de manière efficace. Cependant, cette condition n'est pas suffisante. En effet, notons  $(\mathcal{H}^*, x^*)$  l'hyperplan de marge maximale ainsi que l'observation pour laquelle cette distance est atteinte. Alors on peut pencher indéfiniment  $\mathcal{H}^*$  tant que  $x^*$  reste toujours l'observation de distance minimale (schema). C'est ici que l'algorithme du SVM intervient.

### 3.3.2 SVM a marge rigide (Hard SVM)

Commençons par définir quelques propositions qui nous serons utiles par la suite.

**Proposition 3** Soit  $\mathcal{H}(w, b)$  un hyperplan tel que  $\|w\| = 1$ , alors pour tout  $x \in \mathbb{R}^p$  la distance de l'observation à l'hyperplan vérifie :

$$d(\mathcal{H}(w, b), x) = |\langle w, x \rangle + b|$$

On s'autorise la condition  $\|w\| = 1$  car quitte à diviser  $(w, b)$  par  $\|w\|$  l'équation définissant l'hyperplan serait toujours vérifiée.

Preuve de la proposition :

Par définition :  $d(\mathcal{H}(w, b), x) = \min_{y \in \mathcal{H}(w, b)} \|x - y\| = \min_{y \in \mathcal{H}(w, b)} f(y)$

Posons  $v := x - (\langle w, x \rangle + b) \frac{w}{\|w\|}$ . Alors :  $\langle w, v \rangle + b = (\langle w, x \rangle + b) - (\langle w, x \rangle + b) \underbrace{\frac{\langle w, w \rangle}{\|w\|}}_{=1} = 0$

Donc on a bien  $v \in \mathcal{H}(w, b)$ . Or, pour tout  $u \in \mathcal{H}(w, b)$ , on a :

$$\begin{aligned} \|x - u\|^2 &= \|x - v + v - u\|^2 \\ &= \|x - v\|^2 + \underbrace{\|v - u\|^2}_{\geq 0} + 2 \langle x - v, v - u \rangle \\ &\geq \|x - v\|^2 + 2 (\langle w, x \rangle + b) \underbrace{(\langle w, v \rangle - \langle w, u \rangle)}_{=0} \\ &= \|x - v\|^2 \\ &= (\langle w, x \rangle + b)^2 \frac{\|w\|}{\|w\|^2} = (\langle w, x \rangle + b)^2 \end{aligned}$$

Donc on a bien pour tout  $u \in \mathcal{H}(w, b)$ ,  $f(u) = \|x - u\| \geq |\langle w, x \rangle + b|$ , c'est à dire que  $|\langle w, x \rangle + b|$  est un minorant de  $f$  indépendant de  $u$ , de plus ce minorant est atteint pour  $f$  en  $v$ . Donc on a bien :

$$d(\mathcal{H}(w, b), x) = \min_{y \in \mathcal{H}(w, b)} \|x - y\| = |\langle w, x \rangle + b|$$

Fin de la preuve.

On rappel que notre première idée était de prendre l'hyperplan vérifiant les conditions de séparabilité possédant la plus grande marge avec notre jeu de données, la résolution de ce problème d'optimisation s'appelle le *SVM a marge rigide*.

**Définition 30** On appelle SVM à marge rigide ou HSVM le problème d'optimisation suivant :

$$\arg \max_{(w,b) \in \mathbb{R}^p \times \mathbb{R}} \gamma := \arg \max_{(w,b) \in \mathbb{R}^p \times \mathbb{R}} \min_{1 \leq i \leq n} |\langle w, x_i \rangle + b| \quad \text{tel que : } \begin{cases} \forall i \leq n, y_i(\langle w, x_i \rangle + b) \geq 0 \\ \|w\| = 1 \end{cases}$$

On peut écrire de manière équivalente ce problème :

$$\arg \max_{(w,b) \in \mathbb{R}^p \times \mathbb{R}} \min_{1 \leq i \leq n} y_i(\langle w, x_i \rangle + b) \quad \text{tel que : } \|w\| = 1$$

Comme nous avons vu en première partie, l'idéal serait de se retrouver avec un problème d'apprentissage convexe afin pouvoir trouver une solution globale unique au problème. Nous allons alors reformuler le problème de tel sorte à retomber sur un problème d'optimisation quadratique convexe.

**Proposition 4** Soit  $\mathcal{H}(w, b)$  notre hyperplan séparateur. Alors on a :

$$\gamma = \frac{1}{\|w\|}$$

Preuve de la proposition :

En reprenant notre calcul sans normaliser notre vecteur  $w$  on trouve :

$$d(\mathcal{H}(w, b), x_i) = \frac{|\langle w, x_i \rangle + b|}{\|w\|} = \frac{y_i(\langle w, x_i \rangle + b)}{\|w\|} \quad (\text{condition de séparabilité})$$

On note alors  $x^+$  et  $x^-$  les vecteurs supports se trouvant respectivement du côté positif et négatif de l'hyperplan séparateur. De même, on notera  $\mathcal{H}^+(w, b)$  et  $\mathcal{H}^-(w, b)$  les hyperplans vérifiant :

$$\begin{cases} (1) \mathcal{H}^\pm(w, b) \text{ est parallèles à } \mathcal{H}(w, b) \\ (2) \mathcal{H}^\pm(w, b) \text{ se trouve à une distance } \gamma \text{ de } \mathcal{H}(w, b) \\ (3) \text{l'un des deux hyperplans passe par au moins un vecteur support} \end{cases}$$

De par la propriété (1) de ses hyperplans et par les propriétés de symétrie, leur équation caractéristique doit vérifier  $\forall 1 \leq i \leq n$  :

$$\begin{cases} \langle w, x_i^+ \rangle + b = C \\ \langle w, x_i^- \rangle + b = -C \end{cases}$$

Alors en normalisant  $(w, b)$  de tel sorte à avoir  $\forall 1 \leq i \leq n$  :

$$\begin{cases} \langle w, x_i^+ \rangle + b = 1 \\ \langle w, x_i^- \rangle + b = -1 \end{cases}$$

On en déduit que :  $\gamma = \min_{1 \leq i \leq n} \frac{y_i(\langle w, x_i \rangle + b)}{\|w\|} = \frac{1}{\|w\|}$

Fin de la preuve.

Les équations caractéristiques définies dans la démonstration nous permettent de réécrire la condition de séparabilité sous la forme :  $\forall 1 \leq i \leq n, y_i(\langle w, x_i \rangle + b) \geq 1$ . En se rappelant que notre objectif de

départ était de maximiser la marge sous les conditions de séparabilité, on peut réécrire notre problème sous la forme :

$$\arg \max_{(w,b) \in \mathbb{R}^p} \frac{1}{\|w\|}$$

Or maximiser  $\frac{1}{\|w\|}$  sous  $w$  est équivalent à minimiser  $\frac{1}{2}\|w\|^2$  (nous utilisons cette forme afin de nous assurer de la convexité du critère) sous  $w$ . Nous avons donc bien une forme quadratique convexe pour notre problème, il ne nous reste plus qu'à nous assurer que ses solutions sont bien les solutions du SVM à marge rigide.

**Proposition 5** Soit  $(w_0, b_0)$  des solutions du problème suivant :

$$\arg \min_{(w,b) \in \mathbb{R}^p \times \mathbb{R}} \frac{1}{2}\|w\|^2 \text{ tel que : } y_i(\langle w, x_i \rangle + b) \geq 1$$

Alors  $(\hat{w}, \hat{b}) := (\frac{w_0}{\|w_0\|}, \frac{b_0}{\|b_0\|})$  est aussi solution du HSVM. C'est la forme dite "primale" du SVM et c'est celle que l'on retrouve le plus souvent.

Preuve de la proposition :

Soit  $(w^*, b^*)$  une solution du HSVM, posons alors  $\gamma^* := \min_{1 \leq i \leq n} y_i(\langle w^*, x_i \rangle + b^*)$ . Alors on a :

$$\begin{aligned} \forall i \leq n, y_i(\langle w^*, x_i \rangle + b^*) &\geq \gamma^* \\ \Leftrightarrow y_i(\left\langle \frac{w^*}{\gamma^*}, x_i \right\rangle + \frac{b^*}{\gamma^*}) &\geq 1 \end{aligned}$$

Alors le couple  $(\frac{w^*}{\gamma^*}, \frac{b^*}{\gamma^*})$  vérifie les conditions de la forme quadratique du SVM à marge rigide. De plus on a :

$$\|w_0\| \leq \left\| \frac{w^*}{\gamma^*} \right\| = \frac{\overbrace{\|w^*\|}^{=1}}{\gamma^*} = \frac{1}{\gamma^*}$$

Donc finalement :

$$\forall i \leq n, y_i(\langle \hat{w}, x_i \rangle + \hat{b}) = \frac{1}{\|w_0\|} \underbrace{y_i(\langle w_0, x_i \rangle + b_0)}_{\geq 1} \geq \frac{1}{\|w_0\|} \geq \gamma^*$$

Donc puisque  $\|\hat{w}\| = 1$  alors  $(\hat{w}, \hat{b})$  est aussi solution du HSVM.

Fin de la preuve.

Pour finir nous avons une proposition qui va nous permettre d'adapter l'apprentissage du SVM en fonction du rapport entre la dimension de nos observations et le nombre d'exemples disponibles.

**Proposition 6** Les solutions du problème primal du HSVM sont les mêmes que celles de son dual :

$$\arg \max \sum_{k=1}^n \lambda_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle \quad \text{avec : } \begin{cases} \sum_{i=0}^n \lambda_i y_i = 0 \\ \forall i \leq n, \lambda_i \geq 0 \end{cases}$$

*Preuve de la proposition*

On pose  $\begin{cases} f(w) = \frac{1}{2} \|w\|^2 \\ g_i(w) = y_i(\langle w, x_i \rangle + b - 1) \\ \mathcal{L}(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=0}^n \lambda_i y_i (\langle w, x_i \rangle + b - 1) \quad (\text{Le Lagrangien}) \\ \mathcal{G}(\lambda) = \inf_{w \in \mathbb{R}^p, b \in \mathbb{R}} \mathcal{L}(w, b, \lambda) \end{cases}$

Alors on peut définir le dual du problème initial :

$$\sup \inf \frac{1}{2} \|w\|^2 - \sum_{i=0}^n \lambda_i y_i (\langle w, x_i \rangle + b - 1) \quad \text{avec } \begin{cases} \lambda \in \mathbb{R}^{+n} \\ w \in \mathbb{R}^d, b \in \mathbb{R} \end{cases}$$

Puisque le Lagrangien  $\mathcal{L}$  est convexe en  $w$  alors son inf est atteint en  $w$  si et seulement si le gradient de  $\mathcal{L}$  en  $w$  vaut 0, ce qui équivaut à :

$$w^* = \sum_{i=0}^n \lambda_i y_i x_i$$

En outre, puisque le Lagrangien  $\mathcal{L}$  est affine en  $b$  alors son inf est atteint en  $b$  si et seulement si le gradient de  $\mathcal{L}$  en  $b$  vaut 0, ce qui équivaut à :

$$\sum_{i=0}^n \lambda_i y_i = 0$$

Or le critère  $f$  est continu, coercif et strictement convexe, de plus les contraintes sont affines, donc le minimum global du problème est atteint sur  $K$  en  $w^*$  et il est unique. On a également  $f$  et l'ensemble des contraintes  $g_i$  convexes et de classe  $C^1$ . Alors le problème dual admet une solution  $\lambda^*$ . On peut alors en déduire que le gradient de  $\mathcal{L}$  en  $b$  vaut bien 0 sinon  $\mathcal{G}(\lambda^*)$  serait égal à  $-\infty$  car  $\mathcal{L}$  est affine en  $b$ . On peut donc utiliser la solution du problème dual pour retrouver la solution  $b^*$  :

Soit  $\lambda^*$  la solution du problème dual, alors  $w^* = \sum_{i=0}^n \lambda_i^* y_i x_i$ . En reprenant les équations caractéristiques des hyperplans, on a  $\forall x^+ \in \mathcal{H}^+, \langle w^*, x^+ \rangle + b^* = 1$ . Donc  $b^* = 1 - \min_{x^+ \in \mathcal{H}^+} \langle w^*, x^+ \rangle$ .

Pour finir, on réécrit le problème dual à l'aide des solutions précédemment trouvées :

$$\max \frac{1}{2} \left\| \sum_{i=0}^n \lambda_i y_i x_i \right\|^2 - \left( \sum_{i=0}^n \lambda_i y_i \left\langle \sum_{j=0}^n \lambda_j y_j x_j, x_i \right\rangle + \underbrace{\sum_{i=0}^n \lambda_i y_i b^*}_{=0} \right) + \sum_{i=0}^n \lambda_i$$

avec  $\begin{cases} \sum_{i=0}^n \lambda_i y_i = 0 \\ \lambda_i \geq 0 \end{cases}$

En développant la norme on trouve :

$$\max \frac{1}{2} \sum_{i=0}^n \sum_{j=0}^n \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle - \sum_{i=0}^n \sum_{j=0}^n \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle + \sum_{i=0}^n \lambda_i$$

D'où le résultat voulu.

*Fin de la preuve.*

La décision de choisir si l'on optimise le HSVM dans sa forme primale ou duale se fera selon la dimension de notre jeu de données. En effet, les deux problèmes sont respectivement des problèmes d'optimisation en dimension  $p+1$  et  $n$ . De ce fait, si nous avons plus de variables que d'observations (ie :  $p+1 > n$ ) alors on choisira d'optimiser le HSVM dans sa forme duale et inversement. Étant donné notre situation ( $n \gg p$ ), on se limitera dans la suite aux SVM primaux. Tout cela est bien entendu théorique puisqu'il faudrait déjà que nos données soient linéairement séparables, ce qui est rarement le cas dans la réalité.

Maintenant que nous avons trouvé les solutions du HSVM on a une écriture pour notre modèle.

**Définition 31** Soit  $(w^*, b^*)$  les solutions du HSVM. On définit alors notre modèle prédictif par la fonction vérifiant :

$$\forall x \in \mathbb{R}^p, f_{HSVM}(x) = \mathbb{1}_{\langle w^*, x \rangle + b^* \geq 0} - \mathbb{1}_{\langle w^*, x \rangle + b^* < 0} = sgn(\langle w^*, x \rangle + b^*)$$

Donc la fonction de décision du SVM est définie par :

$$\forall x \in \mathbb{R}^p, g_{HSVM}(x) = \langle w^*, x \rangle + b^*$$

### 3.3.3 SMV a marge souple (Soft SVM)

Nous avons vu que tout le fonctionnement du HSVM se fondait sur la séparabilité linéaire de notre jeu de données, or c'est hypothèse qui n'est que très rarement vérifiée dans la pratique. Cette hypothèse intervient dans la "rigidité" de notre région de décision qui se manifeste par la linéarité de l'hyperplan. Nous allons donc introduire l'algorithme d'apprentissage du SSVM dont le principe est simple. Il s'agit d'un problème de régularisation (aussi appelé régularisation de Tikhonov) dans lequel nous allons considérer une variable d'ajustement (aussi appelé "slack variable")  $\xi_i$  dont le but va être de donner une certaine marge de manœuvre à notre hyperplan en pénalisant les contraintes, d'où une rigidité moindre. La nouvelle forme de la contrainte va alors être :  $\forall 1 \leq i \leq n, y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$ . Dans l'idéal on aimerait que  $\xi_i$  ne pénalise pas l'observation si  $(x_i, y_i)$  respecte bien la contrainte de séparabilité, et la pénalise par une quantité qui nous renseigne sur à quel point le couple "transgresse" de la contrainte de séparabilité. Si l'on devait formaliser mathématiquement  $\xi_i$  on aurait :

$$\forall 1 \leq i \leq n, \xi \begin{cases} 0 & \text{si } y_i(\langle w, x_i \rangle + b) \geq 1 \\ 1 - y_i(\langle w, x_i \rangle + b) & \text{sinon} \end{cases}$$

Il nous vient naturellement  $\forall 1 \leq i \leq n, \xi_i := L_{hinge}(y_i, \langle w, x_i \rangle + b) := [1 - y_i(\langle w, x_i \rangle + b)]_+$ . Nous avons maintenant toutes les clés en main pour définir le SSVM.

**Définition 32** On appelle SVM a marge souple ou SSVM le problème d'optimisation suivant :

$$\arg \min_{w, b} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^n [1 - y_i(\langle w, x_i \rangle + b)]_+ \quad \text{tel que : } y_i(\langle w, x_i \rangle + b) \geq 1 - [1 - y_i(\langle w, x_i \rangle + b)]_+$$

Il existe aussi une forme duale pour ce problème mais nous ne le résoudrons pas car il ne nous sera pas utile pour ce mémoire.

### 3.3.4 Consistance du SVM

Dans cette partie nous allons montrer la consistance de l'algorithme du SVM. Afin d'arriver à montrer ce résultat nous allons devoir démontrer quelques propriétés et théorèmes nous renseignant sur la complexité d'un problème apprentissage relativement à la dimension de l'ensemble des hypothèses. Dans toute cette partie nous allons considérer que nos données sont les réalisations iid d'un couple de variable aléatoire  $(X, Y)$  de loi  $\nu := \mathbb{P}_{X,Y}$  et de loi empirique sur notre jeu de données  $\nu_n$ . On note alors pour toute règle de décision  $g \in \mathcal{G}$ ,  $\mathcal{A}_g := \{(x, y) \in \mathbb{R}^p \times \{-1, 1\} \mid g(x) \neq y\}$  le borélien qui contient tous les couples sur lesquels la règle  $g$  réalise une mauvaise décision. On peut alors définir  $\mathcal{A} := \{\mathcal{A}_g \mid g \in \mathcal{G}\}$ .

**Définition 33** Soit  $\mathcal{A}$  une famille de sous ensembles de  $\mathbb{R}^p$  tel que  $\#\mathcal{A} > 1$ , étant donné  $n$  points  $u_i$  de  $\mathbb{R}^p$  on note  $\mathcal{N}_{\mathcal{A}}(u_1, \dots, u_n) := \#\{(u_1, \dots, u_n) \cap A \mid A \in \mathcal{A}\}$  le nombre de sous ensemble de  $\mathcal{A}$  que

l'on peut obtenir en réalisant une intersection entre  $(u_1, \dots, u_n)$  et les sous ensemble de  $\mathcal{A}$ . Lorsque  $\mathcal{N}_{\mathcal{A}}(u_1, \dots, u_n) = 2^n$  on dit que  $\mathcal{A}$  pulvérise  $(u_1, \dots, u_n)$ .

**Définition 34** On appelle coefficient de pulvérisation l'entier défini par :

$$\forall n \in \mathbb{N}, \mathbf{S}_{\mathcal{A}}(n) := \max_{(u_1, \dots, u_n) \in \mathbb{R}^{p \times n}} \mathcal{N}_{\mathcal{A}}(u_1, \dots, u_n)$$

**Définition 35** On appelle dimension de Vapnik-Chervonenkis (ou VC-dimension) de  $\mathcal{A}$  le plus grand entier  $n_0 \geq 1$  tel que  $\mathbf{S}_{\mathcal{A}}(n_0) = 2^{n_0}$ , c'est à dire qu'elle représente le nombre maximal de points que  $\mathcal{A}$  pulvérise. Par convention, si  $\forall n \in \mathbb{N}, \mathbf{S}_{\mathcal{A}}(n) = 2^n$ , on considère que  $\mathcal{A}$  est de  $VC\text{-dim} = +\infty$ . On la note  $\mathbf{V}_{\mathcal{A}}$ . La VC-dim permet de généraliser le concept de complexité pour un ensemble d'hypothèses. Intuitivement, plus la VC-dimension d'un ensemble d'hypothèse est élevée, plus le meilleur modèle de cet exemple est compliqué et donc mécaniquement moins on a de chance que notre modèle soit consistant.



(a) Pulvérisation de 3 points par  $\mathcal{A}$ .



(b) Schéma XOR de répartition des observations.

Ces définitions étant très théoriques nous allons avoir besoin d'exemples concret afin de bien saisir la signification. On se place dans  $\mathbb{R}^2$ , supposons que  $\mathcal{A} := \{f \mid \forall x \in \mathbb{R}^2, f(x) = sgn(\langle w, x \rangle), w \in \mathbb{R}^2\}$  l'ensemble des classifier linéaire du plan. Alors à l'aide d'un schéma, on peut voir quelque soit la répartition des étiquettes parmi un ensemble de 3 points, nous pouvons trouver un classifier linéaire qui sépare les observations étiquetées positivement et négativement, alors  $\mathcal{N}_{\mathcal{A}}(u_1, u_2, u_3) = 2^3 = 8$ , donc l'ensemble des classifier linéaires du plan pulvérise les ensembles de 3 points. De ce fait, on en déduit que  $\mathbf{V}_{\mathcal{A}} \geq 3$ . Or, si l'on observe une répartition de 4 points suivant une logique XOR (ou exclusif) alors il n'existe aucun classifier linéaire capable de séparer linéairement les 4 points. On en déduit  $\mathbf{V}_{\mathcal{A}} < 4$  et donc  $\mathbf{V}_{\mathcal{A}} = 3$ .

Maintenant que la signification de VC-dimension est plus claire, nous pouvons passer à l'énonciation et la démonstration du théorème principal de cette partie : *Le Théorème de Vapnik-Chervonenkis*.

**Théorème 2** (*Théorème de Vapnik-Chervonenkis*) Soit  $(Z_i)_{1 \leq i \leq n} := (X_i, Y_i)_{1 \leq i \leq n}$  une famille iid de variables aléatoires de loi  $\nu$  et de loi empirique  $\nu_n$ . Alors pour toute famille borelienne  $\mathcal{A} \subset \mathbb{R}^p$  et  $\forall \epsilon > 0$  :

$$\mathbb{P}\left(\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| > \epsilon\right) \leq 8\mathbf{S}_{\mathcal{A}}(n)e^{-n\epsilon^2/32}$$

*Preuve du Théorème :*

Soit  $\epsilon > 0$  fixé, on suppose  $n$  assez grand pour que  $n\epsilon^2 \geq 2$ , si ce n'est pas le cas, puisque  $\mathbf{S}_{\mathcal{A}}(n) \leq 2^n$  alors la borne de notre probabilité est plus grande que 1, or, une probabilité est toujours plus petite que 1 donc il n'y a rien à prouver.

Maintenant posons  $(Z'_i)_{1 \leq i \leq n} := (X'_i, Y'_i)_{1 \leq i \leq n}$  une famille iid de variables aléatoires indépendantes de la première famille de loi empirique  $\nu'_n$ . Prenons  $A^* \in \mathcal{A}$  un échantillon aléatoire dépendant de  $(Z_i)_{1 \leq i \leq n}$  tel que  $|\nu_n(A^*) - \nu(A^*)| > \epsilon$ . Alors :

$$\begin{aligned} \mathbb{P}\left(\sup_{A \in \mathcal{A}} |\nu'_n(A) - \nu(A)| > \epsilon/2\right) &= \mathbb{E}[\mathbb{P}(\sup_{A \in \mathcal{A}} |\nu'_n(A) - \nu(A)| > \epsilon/2 | Z_1, \dots, Z_n)] \\ &\geq \mathbb{E}[\mathbb{P}(|\nu'_n(A^*) - \nu(A^*)| > \epsilon/2 | Z_1, \dots, Z_n)] \\ &= \mathbb{P}(|\nu'_n(A^*) - \nu(A^*)| > \epsilon/2) \end{aligned}$$

Or, en utilisant l'inégalité  $||a| - |b|| \leq |a - b|$  on trouve :

$$\left\{ |\nu_n(A^*) - \nu(A^*)| > \epsilon, |\nu'_n(A^*) - \nu(A^*)| < \epsilon/2 \right\} \subset \left\{ |\nu'_n(A^*) - \nu(A^*)| > \epsilon/2 \right\}$$

Donc nous pouvons en déduire :

$$\begin{aligned} \mathbb{P}\left(\sup_{A \in \mathcal{A}} |\nu'_n(A) - \nu(A)| > \epsilon/2\right) &\geq \mathbb{P}(|\nu_n(A^*) - \nu(A^*)| > \epsilon, |\nu'_n(A^*) - \nu(A^*)| < \epsilon/2) \\ &= \mathbb{E}[\mathbb{1}_{[|\nu_n(A^*) - \nu(A^*)| > \epsilon, |\nu'_n(A^*) - \nu(A^*)| < \epsilon/2]}] \\ &= \mathbb{E}[\mathbb{E}[\mathbb{1}_{[|\nu_n(A^*) - \nu(A^*)| > \epsilon]} \mathbb{1}_{[|\nu'_n(A^*) - \nu(A^*)| < \epsilon/2]} | Z_1, \dots, Z_n]] \\ &= \mathbb{E}[\mathbb{1}_{[|\nu_n(A^*) - \nu(A^*)| > \epsilon]} \mathbb{P}(|\nu'_n(A^*) - \nu(A^*)| < \epsilon/2 | Z_1, \dots, Z_n)] \end{aligned}$$

D'après l'inégalité de Bienaimé-Tchebytchev sous  $Z_1, \dots, Z_n$  on a :

$$\mathbb{P}(|\nu'_n(A^*) - \nu(A^*)| < \epsilon/2 | Z_1, \dots, Z_n) \geq 1 - \frac{\mathbb{E}[|\nu'_n(A^*) - \nu(A^*)|^2 | Z_1, \dots, Z_n]}{\epsilon^2/4}$$

Or en observant que conditionnellement à  $Z_1, \dots, Z_n$ ,  $n\nu'_n(A^*) \sim \mathcal{B}(n, \nu(A^*))$  on obtient :

$$\begin{aligned} \mathbb{P}(|\nu'_n(A^*) - \nu(A^*)| < \epsilon/2 | Z_1, \dots, Z_n) &\geq 1 - \frac{\mathbb{V}[\nu'_n(A^*) | Z_1, \dots, Z_n]}{\epsilon^2/4} \\ &= 1 - \frac{\nu(A^*)(1 - \nu(A^*))}{n\epsilon^2/4} \\ &\geq 1 - \frac{1}{n\epsilon^2/4} \end{aligned}$$

Donc finalement on obtient :

$$\begin{aligned} \mathbb{P}\left(\sup_{A \in \mathcal{A}} |\nu'_n(A) - \nu(A)| > \epsilon/2\right) &\geq \mathbb{E}[\mathbb{1}_{[|\nu_n(A^*) - \nu(A^*)| > \epsilon]} (1 - \frac{1}{n\epsilon^2/4})] \\ &\geq \frac{1}{2} \mathbb{P}(|\nu_n(A^*) - \nu(A^*)| > \epsilon) = \frac{1}{2} \mathbb{P}\left(\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| > \epsilon\right) \end{aligned}$$

Ainsi on a bien :  $\mathbb{P}(\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| > \epsilon) \leq 2\mathbb{P}(\sup_{A \in \mathcal{A}} |\nu'_n(A) - \nu(A)| > \epsilon/2)$ .

Maintenant on se donne une famille  $(\sigma_i)_{1 \leq i \leq n}$  suivant une loi de Rademacher et indépendante des  $2n$  couples de variables aléatoires posés précédemment. On a alors  $n \sup_{A \in \mathcal{A}} |\nu'_n(A) - \nu(A)| = \sup_{A \in \mathcal{A}} |\sum_{i=1}^n (\mathbb{1}_A(Z_i) - \mathbb{1}_A(Z'_i))|$  de même loi que  $\sup_{A \in \mathcal{A}} |\sum_{i=1}^n \sigma_i (\mathbb{1}_A(Z_i) - \mathbb{1}_A(Z'_i))|$ . En effet  $\mathbb{1}_A(Z_i) - \mathbb{1}_A(Z'_i)$  étant symétrique par rapport à 0,  $\mathbb{1}_A(Z_i) - \mathbb{1}_A(Z'_i)$  et  $-(\mathbb{1}_A(Z_i) - \mathbb{1}_A(Z'_i))$  sont de même loi, or puisque  $\sigma_i \in \{-1, 1\} \mathbb{P}-p.s$  on a alors bien le résultat voulu. Donc on en déduit :

$$\begin{aligned} \mathbb{P}(\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| > \epsilon) &\leq 2\mathbb{P}\left(\sup_{A \in \mathcal{A}} \frac{1}{n} \left| \sum_{i=1}^n \sigma_i (\mathbb{1}_A(Z_i) - \mathbb{1}_A(Z'_i)) \right| > \epsilon/2\right) \\ &\leq 4\mathbb{P}\left(\sup_{A \in \mathcal{A}} \frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4\right) = 4\mathbb{E}[\mathbb{P}(\sup_{A \in \mathcal{A}} \frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n)] \end{aligned}$$

Or une fois les  $z_1, \dots, z_n$  fixés le vecteur  $(\mathbb{1}_A(z_1), \dots, \mathbb{1}_A(z_n))$  prend  $\mathcal{N}_{\mathcal{A}}(z_1, \dots, z_n)$  valeurs distinctes lorsque  $A$  varie dans  $\mathcal{A}$ , pour s'en convaincre il suffit de remarquer que  $(\mathbb{1}_A(z_1), \dots, \mathbb{1}_A(z_n))$  est le vecteur qui associe la valeur 1 à sa  $i$ ème coordonnée si  $z_i \in \mathcal{A}$ , ainsi il y a autant de valeurs possibles pour ce vecteur de sous ensemble de  $\mathcal{A}$  engendrés par  $z_1, \dots, z_n$  d'où le résultat. Soit au maximum  $\mathbf{S}_{\mathcal{A}}(n)$  par définition. Posons alors  $\mathcal{A}_0$  un sous ensemble de  $\mathcal{A}$  contenant toutes les combinaison possibles de  $Z_1, \dots, Z_n \cap A$  de cardinal inférieur ou égal à  $\mathbf{S}_{\mathcal{A}}(n)$ , on peut donc conclure :

$$\begin{aligned} \mathbb{P}(\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| > \epsilon) &\leq 4\mathbb{E}[\mathbb{P}(\exists A \in \mathcal{A} \mid \frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n)] \\ &= 4\mathbb{E}[\mathbb{P}(\exists A \in \mathcal{A}_0 \mid \frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n)] \\ &= 4\mathbb{E}[\mathbb{P}(\bigcup_{A \in \mathcal{A}_0} \frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n)] \\ &\leq 4\mathbb{E}[\sum_{A \in \mathcal{A}_0} \mathbb{P}(\frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n)] \\ &\leq 4\mathbb{E}[\sup_{A \in \mathcal{A}} \mathbb{P}(\frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n) \sum_{A \in \mathcal{A}_0} 1] \\ &\leq 4\mathbb{E}[\sup_{A \in \mathcal{A}} \mathbb{P}(\frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n) \mathbf{S}_{\mathcal{A}}(n)] \\ &= 4\mathbf{S}_{\mathcal{A}}(n)\mathbb{E}[\sup_{A \in \mathcal{A}} \mathbb{P}(\frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n)] \end{aligned}$$

Or sous  $Z_1, \dots, Z_n$ ,  $\sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i)$  est une somme de  $n$  variables aléatoires centrées, indépendantes, à valeur dans  $[0, 1]$  alors d'après l'inégalité d'Hoeffding on a :

$$\begin{aligned} \mathbb{P}\left(\frac{1}{n} \left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > \epsilon/4 | Z_1, \dots, Z_n\right) &= \mathbb{P}\left(\left| \sum_{i=1}^n \sigma_i \mathbb{1}_A(Z_i) \right| > n\epsilon/4 | Z_1, \dots, Z_n\right) \\ &\leq e^{-n\epsilon^2/32} \end{aligned}$$

D'où le résultat voulu.

*Fin de la preuve.*

Ce théorème va nous être très utile afin de montrer la consistance des algorithmes en haute dimension. D'abord remarquons qu'avec les écritures que nous avons défini plus haut  $\nu_n(A_g) = \mathcal{R}_n(g)$ . Ainsi on a :

$$\left\{ \sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)| \right\} = \left\{ \sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| \right\}$$

**Lemme 1** *On rappelle que  $\mathcal{R}^*$  représente le risque de Bayes,  $\mathcal{R}_n$  le risque empirique et  $g_n = g_n(., \mathcal{D}_n)$  une règle de décision dépendante de notre jeu de données. On a alors :*

$$|\mathcal{R}^* - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| \leq 2 \sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)|$$

*Preuve du lemme :*

On peut écrire le coté gauche de l'inégalité :

$$\begin{aligned} |\mathcal{R}^* - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| &= |\mathcal{R}^* - \mathcal{R}_n(g_n^*) + \mathcal{R}_n(g_n^*) - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| \\ &\leq |\mathcal{R}^* - \mathcal{R}_n(g_n^*)| + |\mathcal{R}_n(g_n^*) - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| \end{aligned}$$

La première quantité est clairement inférieur ou égale à  $\sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)|$  tandis que la deuxième quantité vaut :

$$|\mathcal{R}_n(g_n^*) - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| = |\inf_{g \in \mathcal{F}} \mathcal{R}_n(g) - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| \leq \sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)|$$

D'où le résultat voulu.

*Fin de la preuve.*

Ce résultat nous montre que si  $\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| \rightarrow 0$  alors notre algorithme est consistant. Nous allons maintenant utiliser tous les résultats démontrés afin de montrer que le SVM est un algorithme consistant. Pour cela nous allons avoir besoin du résultat suivant :

**Proposition 7** *Soit  $\mathcal{F} = \{f \mid \forall x \in \mathbb{R}^p, f(x) = \text{sgn}(\langle w, x \rangle), w \in \mathbb{R}^p\}$ , alors :*

$$\mathbf{V}_{\mathcal{F}} = p$$

*Preuve de la proposition :*

Soit  $(e_i)_{1 \leq i \leq p}$  la base canonique de  $\mathbb{R}^p$  alors il est possible de la pulvériser par  $\mathcal{F}$ , en effet en utilisant la propriété  $\forall i \neq j, \langle e_i, e_j \rangle = \mathbb{1}_{i=j}$  toutes les étiquettes  $(y_1, \dots, y_n) \in \{-1, 1\}^p$  peuvent être retrouvée en posant le classifieur linéaire défini par :

$$\forall x \in \mathbb{R}^p, f(x) = \text{sgn}\left(\sum_{j=0}^p y_j \langle e_j, x \rangle\right) = \text{sgn}\left(\left\langle \sum_{j=0}^p y_j e_j, x \right\rangle\right)$$

Donc d'après la définition de la VC-dimension :  $\mathbf{V}_{\mathcal{F}} \geq p$ .

Supposons maintenant qu'il existe un ensemble de  $p+1$  point  $x_1, \dots, x_{p+1}$  pulvérisés par  $\mathcal{F}$ , alors pour

chacun des  $2^{p+1}$  étiquetages possibles il existe un vecteur  $w_k$  tel que  $f_k(x) = \text{sgn}(\langle w_k, x \rangle)$  renvoie la valeur de l'étiquette. En outre on en déduit que la famille  $x_1, \dots, x_{p+1}$  est liée dans  $\mathbb{R}^p$ . Donc  $\exists x_j = \sum_{i \neq j} a_i x_i$  tel que  $a \neq 0_p$ . Posons alors un ensemble d'étiquettes défini par :

$$\begin{cases} y_j = -1 \\ \forall a_i \neq 0, y_i = \text{sgn}(a_i) \end{cases}$$

Si  $a_k = 0$  alors  $y_k$  peut être de signe arbitraire, on choisira pour la suite  $\geq 0$ . Prenons alors  $w \in \mathbb{R}^p$ , on a  $\langle w, x_j \rangle = \sum_{i \neq j} a_i \langle w, x_i \rangle$ , or  $\forall a_i \neq 0, y_i = \text{sgn}(a_i) = \text{sgn}(\langle x_i, w \rangle)$ , donc  $a_i \langle x_i, w \rangle > 0$ , il s'ensuit que  $\sum_{i \neq j} \langle a_i w, x_i \rangle > 0 \neq y_j = -1$ . On a donc une contradiction, on en déduit qu'il n'existe pas d'ensemble de  $p+1$  points pulvérisés par  $\mathcal{F}$ . D'après la définition de la VC-dimension :  $\mathbf{V}_{\mathcal{F}} \leq p$ . Ainsi  $\mathbf{V}_{\mathcal{F}} = p$ .

*Fin de la preuve.*

**Corollaire 1** Soit  $\mathcal{F}' = \{f \mid \forall x \in \mathbb{R}^p, f(x) = \text{sgn}(\langle w, x \rangle + b), (w, b) \in \mathbb{R}^p \times \mathbb{R}\}$ , alors :

$$\mathbf{V}_{\mathcal{F}'} = p + 1$$

*Preuve du corollaire :*

Soit  $(e_i)_{1 \leq i \leq p} \cup \{e_{p+1} := 0_{\mathbb{R}^p}\}$  la base canonique de  $\mathbb{R}^p$  muni de l'origine. Alors en se basant sur la démonstration précédente on peut également la pulvériser en posant le classifieur linéaire défini par :

$$\forall x \in \mathbb{R}^p, f(x) = \text{sgn}\left(\sum_{j=0}^p (y_j - y_{p+1}) \langle e_j, x \rangle + y_{p+1}\right) = \text{sgn}\left(\left\langle \sum_{j=0}^p (y_j - y_{p+1}) e_j, x \right\rangle + y_{p+1}\right)$$

En effet :

$$\forall 1 \leq i \leq p+1, f(e_i) = \begin{cases} \text{sgn}((y_i - y_{p+1}) - y_{p+1}) = y_i & \text{si } i \leq p \\ \text{sgn}(y_{p+1}) = y_{p+1} & \text{si } i = p+1 \end{cases}$$

Donc :  $\mathbf{V}_{\mathcal{F}'} \geq p + 1$

Maintenant nous allons utiliser la proposition précédente afin de montrer qu'il n'existe pas d'ensemble de  $p+2$  points pulvérisé par  $\mathcal{F}'$ . Pour cela il suffit de remarquer que tout classifieur affine de  $\mathbb{R}^p$  peut se voir comme un classifieur linéaire de  $\mathbb{R}^{p+1}$ . En effet :

$$\forall x \in \mathbb{R}^p, f(x) = \text{sgn}(\langle w, x \rangle + b) = \text{sgn}(\langle w', x' \rangle) \quad \text{tel que :} \quad \begin{cases} x' = [x \ 1]^T \\ w' = [w \ b]^T \end{cases}$$

Comme la VC-dimension des classifieurs linéaires de  $\mathbb{R}^{p+1}$  est  $p+1$  alors il n'existe pas d'ensembles de  $p+2$  point qui peuvent être pulvérisés par  $\mathcal{F}'$ . Donc on en déduit :  $\mathbf{V}_{\mathcal{F}'} = p + 1$ .

*Fin de la preuve.*

Nous pouvons déduire de ce corollaire que le SVM est un algorithme consistant.

**Théorème 3** Posons  $\mathcal{F}' = \{f \mid \forall x \in \mathbb{R}^p, f(x) = \text{sgn}(\langle w, x \rangle + b), (w, b) \in \mathbb{R}^p \times \mathbb{R}\}$ . Alors les classifieurs linéaires de  $\mathbb{R}^p$  sont des algorithmes universellement consistants.

*Preuve du théorème :*

D'après le théorème de Vapnik-Chervonenkis on a :

$$\begin{aligned}\mathbb{P}(\sup_{A \in \mathcal{A}} |\nu_n(A) - \nu(A)| > \epsilon) &= \mathbb{P}(\sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)| > \epsilon) \\ &\leq 8\mathbf{S}_{\mathcal{F}}(n)e^{-n\epsilon^2/32} \\ &\leq 8(n+1)^{d+1}e^{-n\epsilon^2/32} \rightarrow 0\end{aligned}$$

Pour la dernière inégalité nous avons admis le *corollaire du lemme de Sauer* qui dit que si  $\mathcal{A}$  une famille d'ensembles admettant une VC-dim finie  $\mathbf{V}_{\mathcal{A}}$ , alors  $\forall n \geq 1$  :

$$\mathbf{S}_{\mathcal{F}}(n) \leq (n+1)^{\mathbf{V}_{\mathcal{A}}}$$

Or,  $\sum_{n=1}^{+\infty} (n+1)^{d+1}e^{-n\epsilon^2/32} < +\infty$ , donc d'après le lemme de Borell-Cantelli on a :

$$\sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)| \rightarrow 0 \text{ } \mathbb{P} - p.s.$$

Or nous avons montré précédemment que  $|\mathcal{R}^* - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| \leq 2 \sup_{g \in \mathcal{F}} |\mathcal{R}_n(g) - \mathcal{R}(g)|$ , donc on en déduit :  $|\mathcal{R}^* - \inf_{g \in \mathcal{F}} \mathcal{R}(g)| \rightarrow 0 \text{ } \mathbb{P} - p.s..$

*Fin de la preuve.*

Le SVM faisant parti des classificateurs linéaires on en déduit que le SVM est un algorithme consistant. Maintenant que nous connaissons les propriétés de convergence de l'algorithme, nous allons voir comment l'optimiser dans les faits.

### 3.3.5 Minimisation du risque empirique

L'algorithme du SVM fait parti de la famille des algorithmes par minimisation du risque empirique, cela signifie que l'apprentissage de l'algorithme se fait via la résolution d'un problème d'optimisation ayant pour but de minimiser (nous verrons plus tard dans ce mémoire une autre famille d'algorithme d'apprentissage). Pour des questions de "difficulté" on ne minimisera que la fonction empirique du SSVM dans le cadre homogène. On rappelle que la fonction de risque du SSVM homogène est :

$$\arg \min_w \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=0}^n [1 - y_i(\langle w, x_i \rangle)]_+ \quad \text{tel que : } y_i(\langle w, x_i \rangle) \geq 1 - [1 - y_i(\langle w, x_i \rangle)]_+$$

Pour ce faire nous allons avoir besoin de quelques propriétés.

**Lemme 2** Soit  $f$  une fonction définie sur un convexe ouvert (ici  $\mathbb{R}^p$ ), on rappelle pour  $w \in \mathbb{R}^p$  on définit la sous différentielle de  $f$  en  $w$  par l'ensemble :

$$\partial f(w) = \{v \in \mathbb{R}^p \mid \forall u \in \mathbb{R}^p, f(u) \geq f(w) + \langle u - w, v \rangle\}$$

Chaque  $v \in \partial f(w)$  est appelé sous gradient de  $f$  en  $w$ . La sous différentielle est une généralisation de la différentielle pour les fonctions non différentielles. Alors  $f$  est convexe si et seulement si :

$$\begin{aligned}\forall w \in \mathbb{R}^p, \exists v \in \mathbb{R}^p \text{ tel que } \forall u \in \mathbb{R}^p, f(u) \geq f(w) + \langle u - w, v \rangle \\ \Leftrightarrow \exists v \in \partial f(w)\end{aligned}$$

*Preuve de la proposition :*

La preuve ressemble beaucoup à celle de la caractérisation de la convexité par une inégalité sur le gradient ou  $v := \nabla f(x)$  réalisée au cours *d'Optimisation* de premier semestre.

*Fin de la preuve.*

**Lemme 3** Soit  $A := \mathbb{R}^p$  un convexe ouvert et  $f$  une fonction convexe définie sur  $A$ . Alors  $f$  est  $\rho$ -lipschitzienne sur  $A$  si et seulement si :  $\begin{cases} \forall w \in A \\ \forall v \in \partial f(w) \end{cases}, \|v\| \leq \rho$

*Preuve du lemme :*

Soit  $w \in A$ , supposons  $\forall v \in \partial f(w), \|v\| \leq \rho$  alors :

$$\begin{aligned} |f(w) - f(v)| &\leq |\langle u - w, v \rangle| \quad (v \in \partial f(w)) \\ &\leq \underbrace{\|v\|}_{\leq \rho} \|w - u\| \quad (\text{Cauchy-Schwarz}) \\ &\leq \rho \|w - u\| \end{aligned}$$

Supposons que  $f$  soit  $\rho$ -lipschitzienne sur  $A$ . Alors puisque  $A$  un convexe ouvert  $\epsilon > 0$ ,  $u := w + \epsilon \frac{v}{\|v\|} \in A$ .

$$\text{Alors on en déduit : } \begin{cases} \langle u - w, v \rangle = \epsilon \|v\| \\ \|u - w\| = \epsilon \end{cases} \Leftrightarrow \begin{cases} f(u) - f(w) \geq \langle u - w, v \rangle = \epsilon \|v\| \\ |f(u) - f(w)| \leq \rho \|u - w\| = \rho \epsilon \end{cases}$$

Donc finalement :  $\epsilon \|v\| \leq \rho \epsilon \Leftrightarrow \|v\| \leq \rho$ .

*Fin de la preuve.*

Nous allons maintenant montrer à l'aide de ces 2 lemmes que notre fonction de minimisation du risque est bien convexe et lipschitzienne.

**Proposition 8** Posons :

$$\forall w \in \mathbb{R}^p, f(w) = \underbrace{\frac{\lambda}{2} \|w\|^2}_{=g(w)} + \underbrace{\frac{1}{n} \sum_{i=0}^n [1 - y_i(\langle w, x_i \rangle)]_+}_{=h(w)}$$

*Alors  $f$  est une fonction convexe et lipschitzienne.*

*Preuve de la proposition :*

Commençons par montrer que  $g$  est lipschitzienne et convexe.

Premièrement, on a :  $\nabla^2 f(w) = \lambda \mathbf{I}_p$ , donc  $g$  est  $\lambda$ -fortement convexe. On en déduit  $g$  strictement convexe et a fortiori convexe. Ensuite de par l'inégalité :  $\|a - b\| \leq |a - b|$ , on a  $g$  est  $\lambda/2$ -lipschitzienne. Montrons maintenant  $h$  est lipschitzienne et convexe.

D'abord on remarque que :  $\forall x, y, x \vee y = \frac{x+y}{2} + |\frac{x-y}{2}|$ . En posant :  $\begin{cases} j_1(x, y) = \frac{x+y}{2} \\ j_2(x, y) = \frac{x-y}{2} \\ j_3(x) = |x| \end{cases}$ , on se rend compte que  $j_1, j_2$  sont convexes car affines et  $j_3$  est convexe grâce à l'inégalité triangulaire. Donc par

composée de fonctions convexe ( $\vee$ ) :  $(x, y) \mapsto x \vee y$  est convexe donc a fortiori ( $0 \vee .$ ) :  $x \mapsto x \vee 0 = [x]_+$ .

En outre pour  $x, y$  fixé, on pose :

$$\forall w \in \mathbb{R}^p, j(w) = [1 - y \langle w, x \rangle]_+$$

Alors la fonction  $j$  est convexe car composée de fonctions convexes. Nous allons utiliser le *lemme 3* pour montrer que cette fonction est lipschitzienne. En se rappelant que si  $f$  différentiable en  $w$  alors  $\partial f(w) = \{\nabla f(w)\}$ , on a :

$$\partial f(w) = \begin{cases} \{0_{\mathbb{R}^p}\} & \text{si } y \langle w, x \rangle > 1 \\ \{-yx\} & \text{si } y \langle w, x \rangle \leq 1 \end{cases}$$

Donc  $\forall v \in \partial f(w), v = \begin{cases} 0_{\mathbb{R}^p} & \text{si } y \langle w, x \rangle > 1 \\ -yx & \text{si } y \langle w, x \rangle \leq 1 \end{cases}$ , alors  $\|v\| \leq \underbrace{|y|}_{=1} \|x\| = \|x\|$ .

Donc d'après le *lemme 3*,  $j$  est  $\|x\|$ -lipschitzienne. On en déduit que  $f$  est  $((\|x_i\| \wedge \|x_j\|)_{1 \leq i \neq j \leq n} \wedge \lambda/2)$ -lipschitzienne car somme de fonction lipschitzienne. De plus, puisque  $1/n$  est positif alors on en déduit  $g$  est convexe car somme de fonctions convexes à coefficient positif. On en déduit alors que  $f$  est convexe.

*Fin de la preuve.*

Nous avons maintenant démontré les principales propriétés qui nous serons utiles pour la suite. Nous allons utiliser la technique de la descente de gradient stochastique. Le SGD permet de diminuer le nombre d'itérations du GD classique expliqué dans la partie réseau de neurone. A la place de calculer le gradient à chaque itération nous allons plutôt calculer un estimateur du gradient. En détail le fonctionnement de l'algorithme est le suivant :

---

**Algorithm 1** Descente de gradient stochastique

---

Initialisation :  $x_0 \in \mathbb{R}^p; \eta, T > 0$

Pour  $i = 1 \rightarrow T$  :

(1) On construit un estimateur  $v_i$  non biaisé de  $\nabla f(x_i)$  (ie :  $\mathbb{E}[v_i] = \nabla f(x_i)$ ) et indépendant des précédents.

(2)  $x_{i+1} = x_i - \eta v_i$

Sortie :  $\mathbb{E}[f(\bar{x}_T)] := \frac{1}{T} \sum_{i=1}^T \nabla f(x_i)$

---

Maintenant que nous connaissons le fonctionnement de l'algorithme, nous allons démontrer que sous les hypothèses de notre critère de minimisation du risque empirique le SGD est convergent et à quelle vitesse.

**Théorème 4** Soit  $f$  une fonction convexe et  $L$ -lipschitzienne, et  $x^* = \arg \min_w f(w)$ . On considère le SGD défini avec un pas  $\eta \leq 1/L$  tel que les estimateurs  $v_i$  aient une variance bornée ( $\forall i \leq n, \mathbb{V}[v_i] \leq \sigma^2$ ). Alors le SGD défini tel que précédemment vérifie :

$$\forall k > 1, \mathbb{E}[f(\bar{x}_k)] - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\eta k} + \eta\sigma^2$$

Avec  $\bar{x}_k = \frac{1}{k} \sum_{i=1}^k x_i$ . En particulier pour  $\eta := 1/\sqrt{T}$ ,  $T := \frac{(\sigma^2 + L\|x_0 - x^*\|^2)}{\epsilon^2}$  itérations suffisent à obtenir une approximation  $2\epsilon$ -optimal (en espérance).

*Preuve de la proposition :*

Puisque  $f$  est convexe on a d'après une proposition vu dans le cours *d'Optimisation* du premier semestre :

$$\begin{aligned} f(x_{i+1}) &\leq f(x_i) + \langle \nabla f(x_i), x_{i+1} - x_i \rangle + \frac{L}{2} \|x_{i+1} - x_i\|^2 \\ &= f(x_i) - \eta \langle \nabla f(x_i), v_i \rangle + \frac{L\eta^2}{2} \|v_i\|^2 \end{aligned}$$

En passant à l'espérance des 2 cotés on a alors :

$$\begin{aligned} \mathbb{E}[f(x_{i+1})] &\leq f(x_i) - \eta \|\nabla f(x_i)\|^2 + \frac{L\eta^2}{2} \mathbb{E}[\|v_i\|^2] \quad (\mathbb{E}[\|v_i\|^2] = \|\nabla f(x_i)\|^2 + \mathbb{V}[v_i]) \\ &\leq f(x_i) - \eta \underbrace{(1 - L\eta/2)}_{\leq 1} \|\nabla f(x_i)\|^2 + \frac{L\eta^2}{2} \sigma^2 \\ &\leq f(x_i) - \frac{\eta}{2} \|\nabla f(x_i)\|^2 + \frac{\eta}{2} \sigma^2 \quad (\eta L \leq 1) \end{aligned}$$

En combinant les deux équations et en appliquant en  $x^*$  on a :

$$\mathbb{E}[f(x_{i+1})] \leq f(x^*) + \langle \nabla f(x_i), x_i - x^* \rangle - \frac{\eta}{2} \|\nabla f(x_i)\|^2 + \frac{\eta}{2} \sigma^2$$

En utilisant l'égalité :  $\mathbb{E}[v_i] = \nabla f(x_i)$  et l'inégalité :  $\|\nabla f(x_i)\|^2 = \mathbb{E}[\|v_i\|^2] - \mathbb{V}[v_i] \leq \mathbb{E}[\|v_i\|^2] - \sigma^2$  on a alors :

$$\begin{aligned} \mathbb{E}[f(x_{i+1})] &\leq f(x^*) + \langle \mathbb{E}[v_i], x_i - x^* \rangle - \frac{\eta}{2} \mathbb{E}[\|v_i\|^2] + \eta \sigma^2 \\ &= f(x^*) + \mathbb{E}[\langle v_i, x_i - x^* \rangle] - \frac{\eta}{2} \mathbb{E}[\|v_i\|^2] + \eta \sigma^2 \\ &= f(x^*) + \mathbb{E}\left[\frac{1}{2\eta} (\|x_i - x^*\|^2 - \|x_i - x^* - \eta v_i\|^2)\right] + \eta \sigma^2 \\ &= f(x^*) + \mathbb{E}\left[\frac{1}{2\eta} (\|x_i - x^*\|^2 - \|x_{i+1} - x^*\|^2)\right] + \eta \sigma^2 \end{aligned}$$

Ensuite en sommant sur  $i = 0 \rightarrow k-1$  et en simplifiant le résultat par somme télescopiques puis en prenant en compte que  $x_0$  est déterministe on obtient :

$$\sum_{i=0}^{k-1} (\mathbb{E}[f(x_{i+1})] - f(x^*)) \leq \frac{1}{2\eta} (\|x_0 - x^*\|^2 - \mathbb{E}[\|x_k - x^*\|^2]) + k\eta\sigma^2 \leq \frac{\|x_0 - x^*\|^2}{2\eta} + k\eta\sigma^2$$

Pour finir puisque  $f$  est convexe on a :

$$kf(\bar{x}_k) \leq k \frac{\sum_{i=1}^k f(x_i)}{k} = \sum_{i=1}^k f(x_i)$$

Ainsi d'après l'inégalité de Jensen appliquée aux espérances :

$$\sum_{i=0}^{k-1} (\mathbb{E}[f(x_{i+1})] - f(x^*)) \geq k\mathbb{E}[f(\bar{x}_k)] - kf(x^*)$$

On a finalement en simplifiant par  $k > 1$  :

$$\mathbb{E}[f(\bar{x}_k)] - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\eta k} + \eta \sigma^2$$

*Fin de la preuve.*

On a donc montré qu'avec les hypothèses dont nous disposions sur le critère de minimisation sur notre fonction de risque empirique il était possible de la minimiser à l'aide du SGD. On remarque cependant que le SGD nécessite que notre fonction soit différentiable, l'algorithme réalise en fait une petite astuce afin de pallier ce problème. L'algorithme remplace le gradient par la fonction  $\nabla f(w_i) := \lambda w_i + \frac{1}{n} \sum_{j=0}^n (-y_j x_j) \mathbb{1}_{[y_j \langle w_i, x_j \rangle \leq 1]}$ . Il existe également un deuxième moyen de pallier au caractère non différentiable de notre fonction, il s'agit de la méthode du *sous-gradient stochastique* mais nous ne l'explorerons pas par soucis de temps.

### 3.3.6 Astuce du noyau (Kernel trick)

Nous avons vu dans la première partie que le SVM était plus efficace lorsque nos données étaient linéairement séparables, comment obtenir les meilleures performances possibles lorsque ce n'est pas le cas ? Nous allons utiliser un espace de redescription dans lequel nous allons projeter nos données. Dans cet espace (qui est souvent de dimension supérieure) nos données vont s'exprimer de manière différentes et nous allons alors essayer de trouver un hyperplan séparateur dans ce nouvel espace de redescription.

**Définition 36** On appelle espace de redescription le Hilbert  $\mathcal{H}$  dans lequel il est souhaitable de redéfinir nos données à l'aide d'une fonction de redescription  $\phi : \mathbb{R}^p \mapsto \mathcal{H}$

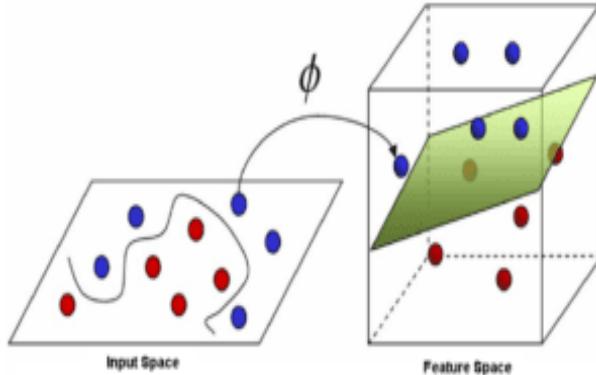


FIGURE 2 – Exemple de redescription d'un jeu de données non séparable de  $\mathbb{R}^2$  vers  $\mathbb{R}^3$

**Définition 37** On appelle noyau toute fonction  $k : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathcal{H}$  continue, symétrique, et semi-définie positive. Il vérifie donc :

$$\begin{cases} \forall N \in \mathbb{N} \\ \forall x_i \in \mathbb{R}^p \\ \forall a_i \in \mathbb{R} \end{cases}, \sum_{i=0}^N \sum_{j=0}^N a_i a_j k(x_i, x_j) \geq 0$$

Il s'agit donc de toute fonction sous la forme d'un produit scalaire des images dans un Hilbert de ses variables.

**Définition 38** On peut définir la matrice de Gram d'un noyau par la matrice  $K \in \mathcal{M}_n(\mathbb{R})$  dont les coefficients sont définies par :  $K_{i,j} = k(x_i, x_j)$

On va alors avoir un moyen de résoudre les problèmes d'optimisation dans notre espace de redescription en résolvant le problème :

$$\min_w R(\|w\|) + f(\langle w, \phi(x_1) \rangle, \dots, \langle w, \phi(x_n) \rangle)$$

Ou  $f : \mathbb{R}^p \mapsto \mathbb{R}$  une fonction arbitraire et  $R : \mathbb{R}_+ \mapsto \mathbb{R}$  une fonction croissante.

**Théorème 5** (*Théorème de représentation de Moore - Aronszajn*) Supposons  $\phi : \mathbb{R}^p \mapsto \mathcal{H}$  alors il existe  $\alpha \in \mathbb{R}^n$  tel que  $w^* := \sum_{i=1}^n \alpha_i \phi(x_i)$  soit une solution du problème d'optimisation à noyau. Autrement dit, il existe une solution de ce problème  $w^* \in \text{Vect}(\phi(x_1), \dots, \phi(x_n))$ .

*Preuve du théorème :*

Posons  $F := \text{Vect}(\phi(x_1), \dots, \phi(x_n))$ . Supposons que  $v^*$  soit une solution du problème à noyau. Alors il est possible d'écrire  $v^*$  sous la forme :

$$v^* = \underbrace{\sum_{i=1}^n \alpha_i \phi(x_i)}_{\in F} + \underbrace{u}_{\in F^\perp}$$

Si on pose  $w^* = v^* - u$ , alors par orthogonalité on a  $\|w^*\|^2 = \|v^*\|^2 + \|u\|^2$ . Alors par croissance de  $R$  on a :  $R(\|w^*\|) \leq R(\|v^*\|)$ . En outre on a :

$$\forall 1 \leq i \leq n, \langle w^*, \phi(x_i) \rangle = \langle v^* - u, \phi(x_i) \rangle = \langle v^*, \phi(x_i) \rangle$$

De ce fait on en déduit que le  $w^*$  est bien une solution du problème à noyau.

*Fin de la preuve.*

De ce nouveau théorème nous pouvons trouver une nouvelle manière d'écrire notre problème. En effet, puisque :

$$\begin{cases} \langle w^*, \phi(x_i) \rangle = \sum_{j=1}^n \alpha_j \langle \phi(x_j), \phi(x_i) \rangle \\ \|w^*\|^2 = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j) \rangle \end{cases}$$

On peut alors poser  $k^\phi(x, y) = \langle \phi(x), \phi(y) \rangle$  le noyau associé à  $\phi$  dans  $\mathcal{H}$ . On a alors une nouvelle forme pour le problème à noyau :

$$\min_{\alpha \in \mathbb{R}^n} R(\sqrt{\left( \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j k^\phi(x_i, x_j) \right)}) + f\left( \sum_{j=1}^n \alpha_j k^\phi(x_j, x_1), \dots, \sum_{j=1}^n \alpha_j k^\phi(x_j, x_n) \right)$$

On a alors une nouvelle écriture du SVM grâce à l'astuce des noyaux.

**Définition 39** On appelle SSVM a noyau le problème suivant :

$$\min_{\alpha \in \mathbb{R}^n} \frac{\lambda}{2} \alpha^T K^\phi \alpha + \frac{1}{n} \sum_{i=0}^n [1 - y_i (K^\phi \alpha)_i]_+$$

Le problème se réécrit de manière équivalente de la manière suivante :

$$\min_w \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=0}^n [1 - y_i \langle w, \phi(x_i) \rangle]_+$$

Maintenant que nous avons vu comment incorporer l'astuce des noyaux à notre problème du SVM, nous allons passer en revue quelques noyaux usuels.

**Définition 40** On appelle noyau polynomial de degré  $d$  le noyau défini par :

$$\forall x, y \in \mathbb{R}^p, k(x, y) = (1 + \langle x, y \rangle)^d$$

**Définition 41** On appelle noyau radial gaussien de bande passante  $\sigma > 0$  (RBF) le noyau défini par :

$$\forall x, y \in \mathbb{R}^p, k(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$$

Sa fonction de redescription est donnée par :

$$\forall x \in \mathbb{R}^p, \phi(x) = e^{-\frac{\|x\|^2}{2\sigma^2}}$$

### 3.3.7 Complexité du SVM

Num	Methods	Complexity (Big-O)
1	parse_command_line	$O(1) \cdot O(n)$
2	read_problem	$O(1) \cdot 2^n \cdot O(m \cdot n)$
3	svm_predict (main) & svm_predict	$O(1) \cdot O(n) \cdot O(n^2 m)$
4	svm_check_parameter	$O(1) \cdot O(m^2 n) \cdot O(g')$
5	svm_train	$O(1) \cdot 9 \cdot O(n) \cdot 2^n \cdot O(m^2 n) \cdot O(g^2 m)$
6	svm_save_model	$O(1) \cdot 3 \cdot O(n) \cdot 2^n \cdot O(m^2 n)$
7	svm_load_model	$O(1) \cdot 2^n \cdot O(n) \cdot 2^n \cdot O(m^2 n)$
8	svm_check_probability_model	$O(1)$
9	svm_predict_probability	$O(1) \cdot 3 \cdot O(n) \cdot O(g^2)$
10	svm_predict	$O(1) \cdot 4 \cdot O(n) \cdot 2^n \cdot O(g^2)$

FIGURE 3 – Tableau de complexité pour chacune des étapes.

Le SVM fait partie des algorithmes d'apprentissage très gourmand en ressources et en temps. En effet, la complexité de l'entraînement du SVM varie selon le noyau utilisé mais aussi selon la méthode d'optimisation utilisée (par exemple par programmation quadratique ou par SGD). Elle a été estimée entre  $\mathcal{O}(n^2)$  et  $\mathcal{O}(n^3)$ . Tandis que la complexité de prédiction a été estimée à  $\mathcal{O}(n_{vsp})$ . Pour cette raison il est peu conseillé d'utiliser le SVM pour de très vastes jeu de données car il existe des algorithmes aussi précis mais moins gourmand.

### 3.3.8 Résultats pratique

#### i Premier test

Les premiers tests sans optimisation manuelle des hyperparamètres nous donne une précision d'environ 0.72. Les hyperparamètres étant sélectionnés de manière optimale en fonction de la forme du *training set*. Nous allons tenter de nous approcher de cette précision a l'aide des théorèmes que l'on a démontré précédemment.

#### ii Sélection des hyperparamètres

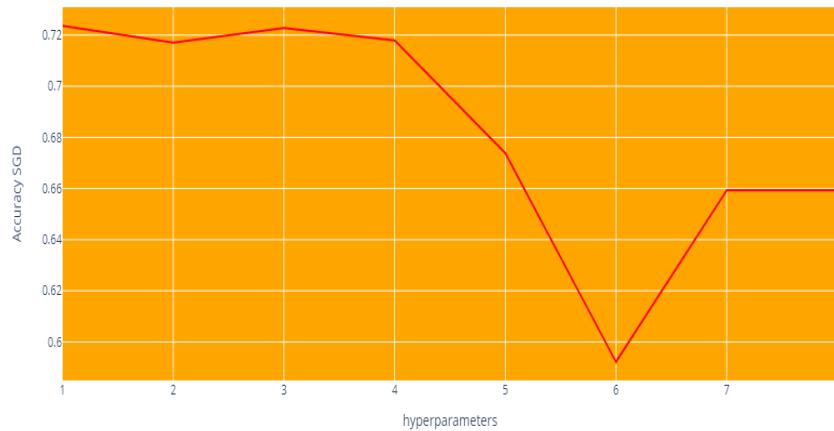


FIGURE 4 – Graphique de l'évolution de la précision du modèle en fonction du paramètre de régularisation

On peut voir que l'arbre a l'air de donner des résultats optimaux pour  $\alpha = 0.01$ . Nous allons maintenant essayer de trouver le nombre d'itérations maximales du gradient a  $\alpha$  fixé.

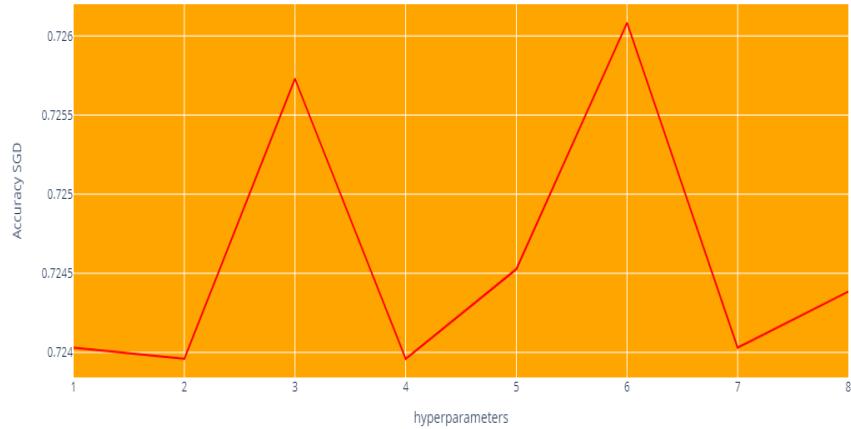


FIGURE 5 – Graphique de l'évolution de la précision du modèle en fonction du nombre d'itérations maximales.

On voit que les performances de notre modèles ne varie pas de manière conséquente. En outre, nos performances s'approchent déjà de celles du modèle par défaut.

### 3.4 KNN (K-Nearest Neighbours)

#### 3.4.1 Plus proche voisin

Dans cette partie on se place dans  $\mathcal{X} \times \mathcal{Y} := (\mathbb{R}^p, d_2) \times \{0, 1\}$  et on notera toujours sauf indications contraire  $\|\cdot\| := \|\cdot\|_2$ . On se donne un jeu de données  $\mathcal{D} = (X_i, Y_i)_{1 \leq i \leq n}$  que l'on suppose des réalisations iid d'un couple  $(X, Y)$  de variables aléatoire de loi  $\mathbb{P}_{X,Y}$ .

$\forall x \in \mathbb{R}^p$ , on ordonne nos observations par ordre croissant par rapport a leur distance avec  $x$ , on a donc :

$$\|X_{(1)}(x) - x\| \leq \dots \leq \|X_{(n)}(x) - x\|$$

**Définition 42** soit  $x \in \mathbb{R}^p$ , on appelle plus proche voisin de  $x$  l'observation  $X_{(1)}(x)$ , c'est a dire la plus proche observation de  $x$  au sens de la distance euclidienne.

**Définition 43** On appelle algorithme du plus proche voisin la méthode qui consiste a étiqueter une nouvelle observation de la même étiquette que son plus proche voisin dans le jeu d'entraînement de laquelle elle est la plus proche. On écrit le modèle prédictif associé de la manière suivante :

$$\forall x \in \mathbb{R}^p, \Phi_n^{NN}(x) = \mathbb{1}_{\{Y_{(1)} > 1/2\}}$$

Il existe un problème fondamental avec l'algorithme simple du plus proche voisin. En effet, que faire si l'une des observations de notre jeu de données est mal étiqueté (par exemple si les données de notre jeu d'entraînement ne sont pas parfaitement étiquetées dû a une erreur humaine lors de la récolte de données) ? Pour illustrer ce problème nous allons passer par les diagrammes de Voronoï.

### 3.4.2 Diagramme de Voronoï

**Définition 44** Soit  $\mathcal{S}$  un sous ensemble fini de  $\mathbb{R}^p$  et  $u \in \mathcal{S}$ . On appelle cellule de Voronoï de la germe  $u$  l'ensemble défini par :

$$Vor(u) := \{x \in \mathbb{R}^p \mid \forall v \in \mathcal{S}, \|x - u\| \leq \|x - v\|\}$$

La cellule de Voronoï de  $u$  correspond à l'ensemble des éléments de  $\mathbb{R}^p$  qui sont plus proche de  $u$  que de n'importe quel autre élément de  $\mathcal{S}$ .

**Remarque 8** Avec notre définition de cellule on peut réécrire  $\mathbb{R}^p$  comme une  $n$ -partition de la manière suivante ou chaque  $x \in \mathbb{R}^p$  est classé dans la cellule de son plus proche voisin par rapport à notre jeu de données :

$$\mathbb{R}^p = \bigcup_{i=1}^n Vor(X_i)$$

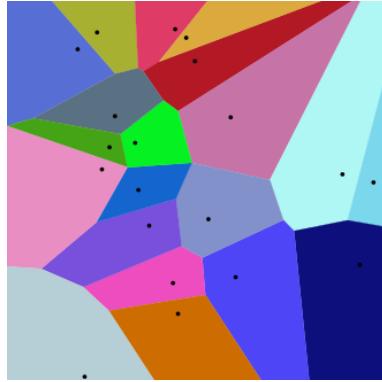


FIGURE 6 – Partition de Voronoï où chaque point représente une germe et chaque couleur sa cellule associée.

On voit alors bien où est le problème, si l'une des observations de notre jeu de données venait à être mal étiquetée alors l'ensemble des points de sa cellule de Voronoï associée seraient eux aussi mal étiqueté. Pour pallier ce problème et augmenter la robustesse de notre méthode, on se propose de "contrôler" la justesse de notre prédiction en comparant notre observation à ses  $k$  plus proches voisins, on appelle ce procédé le "majority vote". Ici,  $k$  agit comme un hyperparamètre dans le cadre d'une régularisation.

### 3.4.3 k - Plus proche voisins

**Définition 45** On appelle algorithme des  $k$  plus proche voisin la méthode qui consiste à étiqueter une nouvelle observation de la même étiquette que les  $k$  observations du jeu d'entraînement de laquelle elle est la plus proche. On écrit le modèle prédictif associé de la manière suivante :

$$\forall x \in \mathbb{R}^p, \Phi_n^{kNN}(x) = \begin{cases} 1 & \text{si } \frac{1}{k} \sum_{i \leq k} \mathbb{1}_{\{Y(i)(x)=1\}} > \frac{1}{k} \sum_{i \leq k} \mathbb{1}_{\{Y(i)(x)=0\}} \\ 0 & \text{sinon} \end{cases}$$

L'algorithme consiste à choisir le  $c$  (la valeur de l'étiquette) qui maximise la somme qui compte le nombre fois où  $x$  est étiqueté de la même manière que ses  $k$  plus proches voisins. On peut également définir la règle kNN sous la forme d'une règle de classification plug-in. Il suffit pour cela de compter le nombre de plus proches voisins étiquetés "1" et de normaliser le résultat. Si plus de la moitié de ses  $k$  plus proches voisins sont étiquetés "1", alors la somme de comptage sera  $\geq 1/2$ . On obtient alors l'estimateur défini de la manière suivante :

$$\forall x \in \mathbb{R}^p, \eta_{n,k}(x) = \frac{1}{k} \sum_{i \leq k} Y_{(i)}(x)$$

La règle de classification plug-in se réécrit donc :

$$\forall x \in \mathbb{R}^p, \Phi_n^{kNN}(x) = \mathbb{1}_{\{\eta_{n,k}(x) \geq 1/2\}}$$

### 3.4.4 Choix du nombre $k$ de plus proche voisins

Dans cette partie nous allons étudier le rapport entre la consistance du kNN et le choix de l'hyperparamètre  $k$ . Pour arriver à ce résultat nous allons devoir un résultat théorique majeur : *le Théorème de Stone*. Nous allons déjà commencer par montrer un résultat préliminaire nécessaire pour la démonstration du théorème de Stone.

**Théorème 6** Soit  $\eta_n$  un estimateur de la fonction de régression et  $\Phi_n$  la règle de décision plug-in associée, alors :

$$0 \leq \mathcal{R}(\Phi_n) - \mathcal{R}^* \leq 2 \int_{\mathbb{R}^p} |\eta_n(x) - \eta(x)| d\mathbb{P}_X(x)$$

En outre, on a :

$$\forall p \geq 1, \begin{cases} 0 \leq \mathcal{R}(\Phi_n) - \mathcal{R}^* \leq 2 (\int_{\mathbb{R}^p} |\eta_n(x) - \eta(x)|^p d\mathbb{P}_X(x))^{1/p} \\ 0 \leq \mathbb{E}\mathcal{R}(\Phi_n) - \mathcal{R}^* \leq 2 (\mathbb{E}|\eta_n(X) - \eta(X)|^p)^{1/p} = 2 \|\eta_n(X) - \eta(X)\|_{L^p} \end{cases}$$

Preuve du théorème :

On reprend l'équation utilisé dans la démonstration du théorème de Györfi, Mammen et Tsybakov, et on trouve :

$$\mathbb{P}(\eta_n(X) \neq Y | \mathcal{D}_n) - \mathbb{P}(\Phi^*(X) \neq Y | X) = |2\eta(X) - 1| \mathbb{1}_{\{\Phi^*(X) \neq \eta_n(X)\}}$$

En primitivant sur  $\mathbb{P}_X$  et en factorisant par 2 on obtient :

$$\mathcal{R}(\Phi_n) - \mathcal{R}^* = 2 \int_{\mathbb{R}^p} |\eta(x) - 1/2| \mathbb{1}_{\{\Phi^*(x) \neq \eta_n(x)\}} d\mathbb{P}_X(x)$$

Il suffit ensuite de réaliser les 2 cas sur  $\{\Phi^* \neq \eta_n\}$  pour déduire que  $|\eta(x) - 1/2| \leq |\eta(x)_n - \eta(x)|$

Cas 1 : si  $\Phi^*(x) = 1$ , alors  $\eta(x) \geq 1/2$  et donc  $|\eta(x) - 1/2| = \eta(x) - 1/2$  de plus on a  $\Phi(x) = 0$  donc  $\eta_n(x) < 1/2$ , alors  $\eta(x) - 1/2 \leq \eta(x) - \eta_n(x)$ , puisque les 2 termes sont positifs alors l'inégalité est vérifiée.

Cas 2 : si  $\Phi^*(x) = 0$  alors on applique le même raisonnement.

Ainsi on a :

$$\mathcal{R}(\Phi_n) - \mathcal{R}^* \leq 2 \int_{\mathbb{R}^p} |\eta_n(x) - \eta(x)| d\mathbb{P}_X(x)$$

En posant  $1 = \frac{1}{p} + \frac{p-1}{p}$  on obtient avec l'inégalité d'Hölder :

$$\begin{aligned} 2 \int_{\mathbb{R}^p} |\eta_n(x) - \eta(x)| d\mathbb{P}_X(x) &\leq 2 \left( \int_{\mathbb{R}^p} |\eta_n(x) - \eta(x)|^p d\mathbb{P}_X(x) \right)^{1/p} \underbrace{\left( \int_{\mathbb{R}^p} 1^{p-1} d\mathbb{P}_X(x) \right)^{1/p-1}}_{=1} \\ &= 2 \left( \int_{\mathbb{R}^p} |\eta_n(x) - \eta(x)|^p d\mathbb{P}_X(x) \right)^{1/p} \end{aligned}$$

Puis on conclut en primitivant une seconde fois et enfin en utilisant l'inégalité de Jensen appliquée à l'espérance.

*Fin de la preuve.*

Cette règle nous permet de déduire que si l'on dispose d'un estimateur consistant de la fonction de régression d'un jeu de données alors la règle de classification plug-in associé est elle aussi consistante. Le *Théorème de Stone* va nous donner un moyen de construire des estimateurs de la fonction de régression universellement constants. Mais avant de voir avec plus de profondeur le *Théorème de Stone*, il est important de voir en quoi ce théorème est incontournable pour notre problème. Pour ce faire, nous allons montrer que pour un nombre fixe de proche voisin ne dépendant pas de notre quantité de données, l'algorithme  $\Phi_n^{kNN}$  n'est pas consistant et ne fonctionne donc pas de manière optimale.

**Théorème 7** Soit  $k \geq 1$  un entier impair fixé. Alors le risque asymptotique du kNN vérifie :

$$\begin{aligned} \mathcal{R}_{kNN} &= \mathbb{E} \left[ \sum_{j=0}^k \binom{j}{k} \eta(X)^j (1 - \eta(X))^{k-j} (\eta(X) \mathbb{1}_{j < k/2} + (1 - \eta(X)) \mathbb{1}_{j \geq k/2}) \right] \\ &= \mathcal{R}^* + \mathbb{E}[2|\eta(X) - 1| \mathbb{P}(Z > k/2 | X)] \end{aligned}$$

Avec  $Z \sim \mathcal{B}(k, \eta(X) \wedge (1 - \eta(X)))$

*Preuve du théorème :*

Soit  $(X_{n+1}, Y_{n+1}) \sim \mathbb{P}_{X,Y}$  une nouvelle donnée. En appliquant l'algorithme à notre nouvelle donnée on a :

$$\Phi_n^{kNN}(X_{n+1}) \begin{cases} 1 & \text{si } \frac{1}{k} \sum_{i \leq k} Y_{(i)}(X_{n+1}) \geq 1/2 \\ 0 & \text{si } \frac{1}{k} \sum_{i \leq k} Y_{(i)}(X_{n+1}) < 1/2 \end{cases} \Leftrightarrow \begin{cases} 1 & \text{si } \sum_{i \leq k} Y_{(i)}(X_{n+1}) \geq k/2 \\ 0 & \text{si } \sum_{i \leq k} Y_{(i)}(X_{n+1}) < k/2 \end{cases}$$

Quand le nombre de données disponible est très élevé ( $n \rightarrow +\infty$ ) les  $k$  plus proches voisins de  $X$  sont de plus en plus proches de  $X$  (on le prouvera rigoureusement dans un lemme plus bas). Ainsi le risque

asymptotique du kNN s'écrit :

$$\begin{aligned}
\mathcal{R}_{kNN} &= \lim_{n \rightarrow +\infty} \mathbb{P}(\Phi_n^{kNN}(X_{n+1}) \neq Y_{n+1}) \\
&= \lim_{n \rightarrow +\infty} \mathbb{P}\left(\sum_{i \leq k} Y_{(i)}(X_{n+1}) < k/2, Y_{n+1} = 1\right) + \mathbb{P}\left(\sum_{i \leq k} Y_{(i)}(X_{n+1}) \geq k/2, Y_{n+1} = 0\right) \\
&= \lim_{n \rightarrow +\infty} \mathbb{E}\underbrace{\mathbb{P}(Y_{n+1} = 1 | X_{n+1})}_{=\eta(X_{n+1})} \underbrace{\mathbb{P}\left(\sum_{i \leq k} Y_{(i)}(X_{n+1}) \geq k/2 | X_{n+1}\right)}_{\sim \mathcal{B}(k, \eta(X_{n+1}))} \\
&\quad + \underbrace{\mathbb{P}(Y_{n+1} = 0 | X_{n+1})}_{=1-\eta(X_{n+1})} \underbrace{\mathbb{P}\left(\sum_{i \leq k} Y_{(i)}(X_{n+1}) < k/2 | X_{n+1}\right)}_{\sim \mathcal{B}(k, \eta(X_{n+1}))}
\end{aligned}$$

Avec étant donné  $X_{n+1}$ , les  $(Y_{(i)})_{i \leq k}$  iid, donc la somme suit bien une loi  $\mathcal{B}(k, \eta(X_{n+1})) \equiv \mathcal{B}(k, \eta(X))$  car comme nous l'avons dis plus haut plus nous ajoutons d'observations plus les proches voisins sont proches de  $X$  (ie :  $\|X_{(k)}(X) - X\| \rightarrow 0$ ), donc par continuité de  $\eta()$  la propriété est vraie, d'où l'égalité que nous souhaitions montrer.

On pose ensuite :

$$\mathcal{R}_{kNN} = \mathbb{E}[\psi(\eta(X))]$$

Avec :

$$\forall p \in [0, 1], \psi(p) := p\mathbb{P}(\mathcal{B}(k, p) < k/2) + (1-p)\mathbb{P}(\mathcal{B}(k, p) \geq k/2)$$

On réalise ensuite une disjonction des cas sur  $\{p \geq 1/2\} \cup \{p < 1/2\}$  et on obtient :

Cas 1 : si  $p < 1/2$ , alors  $p < 1 - p$ , donc :

$$\begin{aligned}
\psi(p) &= p(1 - \mathbb{P}(\mathcal{B}(k, p) \geq k/2)) + (1-p)\mathbb{P}(\mathcal{B}(k, p) \geq k/2) \\
&= p + (1-2p)\mathbb{P}(\mathcal{B}(k, p) \geq k/2)
\end{aligned}$$

En réalisant le même raisonnement pour  $p \geq 1/2$ , il en résulte :

$$\psi(p) = p \wedge 1 - p + |2p - 1|\mathbb{P}(\mathcal{B}(k, p \wedge 1 - p) \geq k/2)$$

En se rappelant qu'on a démontré première partie que :

$$\mathcal{R}^* = \mathbb{E}[\eta(X) \wedge (1 - \eta(X))]$$

on a bien :

$$\mathcal{R}_{kNN} = \mathcal{R}^* + \mathbb{E}[2|\eta(X) - 1|\mathbb{P}(Z > k/2 | X)]$$

*Fin de la preuve.*

Cette égalité nous permet de déduire beaucoup de propriétés du  $kNN$  à  $k$  fixé. Premièrement, sans hypothèse supplémentaire sur l'hyperparamètre  $k$ , le  $kNN$  n'est pas un algorithme consistant. Et deuxièmement de par la forme explicite de  $\mathcal{R}_{kNN}$  on peut alors borner le risque du  $kNN$  en fonction de l'hyperparamètre  $k$  fixé choisi et du risque de Bayes  $\mathcal{R}^*$ .

**Corollaire 2** (*Inégalité de Cover et Hart*) On alors un encadrement du risque du  $kNN$  en fonction du risque de Bayes. L'encadrement est le suivant :

$$\mathcal{R}^* \leq \dots \leq \mathcal{R}_{3NN} \leq \mathcal{R}_{NN} \leq 2\mathcal{R}^*(1 - R^*)$$

De plus pour tout  $k \geq 1$  entier impair fixé, alors  $\mathcal{R}_{kNN}$  vérifie :

$$\mathcal{R}_{kNN} \leq \mathcal{R}^* + \frac{1}{\sqrt{ke}}$$

Pour vérifier la première inégalité on va utiliser le fait que  $Z \sim \mathcal{B}(k, \eta(X) \wedge (1 - \eta(X)))$ . Soit  $p < 1/2$  (ce qui nous est assuré par le paramètre  $\eta(X) \wedge (1 - \eta(X))$ ) , alors  $k : \rightarrow \mathbb{P}(\mathcal{B}(k, p) \geq k/2)$  est décroissante sur  $k$ . Donc l'inégalité est bien vérifiée.

Pour vérifier  $\mathcal{R}_{NN} \leq 2\mathcal{R}^*(1 - R^*)$  asymptotiquement, on considère  $(X_{n+1}, Y_{n+1}) \sim \mathbb{P}_{X,Y}$  une nouvelle donnée. Alors d'après le même raisonnement que précédemment en considérant qu'etant donné  $X_{n+1}$ ,  $Y_{(1)}$  et  $Y_{n+1}$  indépendantes avec  $Y_{(1)(X_{n+1})} \sim \mathcal{B}(\eta(X))$  :

$$\begin{aligned}\mathcal{R}_{NN} &= \dots \\ &= \mathbb{P}(Y_{(1)}(X_{n+1}) \neq Y_{n+1}) \\ &= \dots \\ &= \mathbb{E}[2\eta(X)(1 - \eta(X))] = 2\mathbb{E}[\eta(X)(1 - \eta(X))]\end{aligned}$$

Posons  $R(X) := \eta(X) \wedge (1 - \eta(X))$ , alors en réalisant une simple disjonction des cas on a :

$$\mathbb{E}[\eta(X)(1 - \eta(X))] = \mathbb{E}[R(X)(1 - R(X))]$$

On utilise ensuite la concavité de la fonction  $x \mapsto x(1 - x)$  (dérivée seconde négative d'une fonction de  $\mathcal{C}^2([0, 1], [0, 1])$ ) et les propriétés de l'espérance pour obtenir :

$$\mathbb{E}[\eta(X)(1 - \eta(X))] = \mathbb{E}[R(X)(1 - R(X))] \leq \mathbb{E}[R(X)]\mathbb{E}[1 - R(X)] = \mathbb{E}[R(X)](1 - \mathbb{E}[R(X)]) = \mathcal{R}^*(1 - \mathcal{R}^*)$$

Pour finir, prenons  $p < 1/2$  et  $B \sim \mathcal{B}(k, p)$ . Alors :

$$\begin{aligned}(1 - 2p)\mathbb{P}(B \geq k/2) &= (1 - 2p)\mathbb{P}\left(\frac{B - kp}{k} \geq 1/2 - p\right) \\ &\leq (1 - 2p)e^{-2k(1/2 - p)^2} \\ &\leq \sup_{0 \leq u \leq 1} ue^{-ku^2/2} \\ &= \frac{1}{\sqrt{ke}}\end{aligned}$$

La première inégalité est obtenu en utilisant *l'inégalité d'Hoeffding* vu en cours de *statistiques non paramétrique* cette année, tandis que la troisième inégalité a été obtenu en posant  $u := 1 - 2p$ .

*Fin de la preuve.*

La première inégalité nous permet de voir qu'à  $k$  fixé, le kNN n'est jamais consistant sauf si  $\mathcal{R}^* \in \{0, 1/2\}$ . Si l'on souhaite pallier cette situation, il est nécessaire d'ajouter des hypothèses supplémentaire sur l'hyperparamètre. C'est tout le principe du théorème de Stone que nous allons démontrer dans les prochaines lignes. Mais avant cela, nous avons besoin de définir un nouveau type d'algorithme.

**Définition 46** Soit  $n := \#\mathcal{D}_x$  fixé,  $\forall i \leq n$ , on appelle fonction de poids la fonction  $W_{i,n} \in \mathcal{F}(\{0, 1\}^{\mathbb{R}^p})$  qui apporte une pondération des  $X_i$  en fonction de  $x$ . Cette fonction doit vérifier  $\forall x \in \mathbb{R}^p$ ,  $\sum_{i \leq n} W_{i,n}(x) = 1$ . On appelle alors estimateur de type moyenne locale l'estimateur de la régression définit par :

$$\forall x \in \mathbb{R}^p, \eta_n(x) = \sum_{i \leq n} W_{i,n}(x)Y_i(x)$$

*Intuitivement, dans certains contextes, les voisins les plus proches de  $x$  (en terme de distance) apportent davantage d'information sur  $y$  que les voisins les plus éloignés.*

Les algorithmes utilisant un estimateur de type moyenne locale sont appelés *algorithmes par moyenage local*. A la différence des *algorithmes par minimisation du risque empirique*, leur apprentissage ne passe pas par la minimisation du risque empirique via la résolution d'un problème d'optimisation mathématique, mais plutôt par l'estimation de la fonction de régression. Maintenant que nous définissons le cadre, nous avons tous les outils nécessaires pour démontrer le Théorème de Stone.

**Théorème 8 (Théorème de Stone)** *On suppose que quelque soit la loi de  $X$ , les poids satisfont les conditions suivantes :*

- (1)  $\exists c \in \mathbb{R}, \forall f \in \mathcal{F}(\mathbb{R}^p)$  tel que  $f(X) \in L^1$ ,  $\mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X)|f(X_i)|\right] \leq c \mathbb{E}|f(X)|$ ,  $\forall n \geq 1$ .
- (2)  $\forall a > 0$ ,  $\mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X) \mathbb{1}_{||X_i - X|| > a}\right] \rightarrow 0$ .
- (3)  $\mathbb{E}\left[\max_{i \leq n} W_{i,n}(X)\right] \rightarrow 0$ .

Alors la règle plug-in  $\Phi_n$  associée à l'estimateur de type locale  $\eta_n$  est universellement consistante, ou autrement dit :

$$\mathbb{E}\mathcal{R}(\Phi_n) \xrightarrow{L^1} \mathcal{R}^* \text{ peut importe } \mathbb{P}_{X,Y}$$

Intuitivement, la condition (2) signifie qu'hors des boules centrées en  $x$  la contribution apportée par la fonction poids doit être négligeable, en d'autre terme la contribution de la fonction poids doit être limitée au calcul de la moyenne. La condition (1) quant à elle contraint la fonction poids à ne pas attribuer une influence disproportionnée aux différents points. Pour finir, la condition (3) est d'ordre technique pour la mise en place du théorème.

Preuve du théorème :

D'après le précédent théorème, pour montrer que  $\Phi_n$  est consistante il suffit de montrer que  $\eta_n$  converge dans  $L^1$ , or d'après les propriétés de convergence de variables aléatoires, il nous suffit de montrer que  $\eta_n$  converge dans  $L^2$ . Posons :

$$\widehat{\eta}_n(X) := \sum_{i \leq n} W_{i,n}(X)\eta(X)$$

En utilisant l'inégalité  $(a + b)^2 \leq 2(a^2 + b^2)$  et la linéarité de l'intégrale, on obtient :

$$\begin{aligned} \mathbb{E}|\eta_n(X) - \eta(X)|^2 &= \mathbb{E}[|\eta_n(X) - \widehat{\eta}_n(X) + \widehat{\eta}_n(X) - \eta(X)|^2] \\ &\leq 2(\mathbb{E}[\eta_n(X) - \widehat{\eta}_n(X)]^2 + \mathbb{E}[\widehat{\eta}_n(X) - \eta(X)]^2) \end{aligned}$$

Il nous suffit maintenant de montrer que chacun des 2 termes tend vers 0 asymptotiquement. Puisque les poids sont positifs, l'inégalité de Jensen nous permet d'écrire que :

$$\mathbb{E}[\widehat{\eta}_n(X) - \eta(X)]^2 \leq \mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X)|\eta(X_i) - \eta(X)|^2\right]$$

Pour la suite on va supposer sans perdre en généralité dans notre situation que :

$$\eta \text{ est tel que } \begin{cases} (1) \eta(\mathbb{R}^p) = [0, 1] \\ (2) \eta \in \mathcal{C}(\mathbb{R}^p, [0, 1]) \\ (3) \text{Supp}(\eta) \text{ borné} \end{cases}$$

Alors on a  $Supp(\eta)$  borné, donc il existe  $M$  tel que  $Supp(\eta) \subseteq [-M, M]^p$  ainsi puisque  $\eta$  est continue sur un compact de  $\mathbb{R}^p$ , d'après un lemme vu dans le cours d'intégration de L3,  $\eta$  est uniformément continue. De ce fait :

$$\forall \epsilon > 0, \exists a > 0, \forall x, x' \in \mathbb{R}^p, \|x - x'\|_2 \leq a \Rightarrow |\eta(x) - \eta(x')|^2 \leq \epsilon$$

En remarquant que  $\begin{cases} (1) & \forall a > 0, \mathbb{1}_{\{\|X_i - X\| > a\}} + \mathbb{1}_{\{\|X_i - X\| \leq a\}} = 1 \quad \mathbb{P} - p.s \\ (2) & \text{sur } \{\|X_i - X\| \leq a\}, |\eta(X_i) - \eta(X)|^2 \leq \epsilon \quad \mathbb{P} - p.s \\ (3) & |\eta(X_i) - \eta(X)|^2 \leq 1 \quad \mathbb{P} - p.s \end{cases}$

On obtient que en appliquant la seconde condition :

$$\mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X) |\eta(X_i) - \eta(X)|^2\right] \leq \underbrace{\mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X) \mathbb{1}_{\{\|X_i - X\| > a\}}\right]}_{\rightarrow 0} + \underbrace{\mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X) \epsilon\right]}_{\rightarrow \epsilon}$$

On pourrait également réaliser la démonstration dans le cas général. En effet puisque  $\eta$  est bornée alors  $\eta \in L^2$ , donc d'après un lemme vu dans le cours d'intégration de L3, il existe une fonction étagée  $\psi$  tel que  $\forall \epsilon > 0, \|\eta(X) - \psi(X)\|_{L^2} < \epsilon$ . Ensuite on réalise une démonstration similaire en utilisant l'inégalité  $(a + b + c)^2 \leq 3(a^2 + b^2 + c^2)$ . Maintenant, il nous reste à contrôler  $\mathbb{E}[\eta_n(X) - \widehat{\eta}_n(X)]^2$ .

Pour commencer, on constate que  $\forall i \neq j$  :

$$\begin{aligned} \mathbb{E}[W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)(Y_j - \eta(X_j))] &= \mathbb{E}[\mathbb{E}[W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)(Y_j - \eta(X_j))|X, X_1, \dots, X_n, Y_i]] \\ &= \mathbb{E}[W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)\mathbb{E}[(Y_j - \eta(X_j))|X, X_1, \dots, X_n, Y_i]] \\ &= \mathbb{E}[W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)\mathbb{E}[(Y_j - \eta(X_j))|X_j]] \\ &= \mathbb{E}[W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)\mathbb{E}[(\eta(X_j) - \eta(X_i))]] \\ &= 0 \end{aligned}$$

La première ligne s'obtient en utilisant la propriété  $\forall \mathcal{G} \in \mathcal{T}, \mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|\mathcal{G}]]$ . Le passage de la première à la deuxième ligne est possible car  $W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)$  est  $\sigma(X, X_1, \dots, X_n, Y_i)$  mesurable. Celui de la deuxième à la troisième s'obtient en remarquant que  $Y_j - \eta(X_j)$  est indépendante de  $\sigma(X, X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_n, Y_i)$ . Pour finir, l'avant dernière ligne s'obtient par linéarité de l'espérance conditionnelle.

Ainsi :

$$\begin{aligned} \mathbb{E}[\eta_n(X) - \widehat{\eta}_n(X)]^2 &= \mathbb{E}\left[\left(\sum_{i \leq n} W_{i,n}(X)(Y_i - \eta(X_i))\right)^2\right] \\ &= \sum_{i \leq n} \sum_{j \leq n} \underbrace{\mathbb{E}[W_{i,n}(X)(Y_i - \eta(X_i))W_{j,n}(X)(Y_j - \eta(X_j))]}_{=0 \text{ si } i \neq j} \\ &= \sum_{i \leq n} \mathbb{E}[W_{i,n}^2(X) \underbrace{(Y_i - \eta(X_i))^2}_{\leq 1}] \\ &\leq \sum_{i \leq n} \mathbb{E}[W_{i,n}^2(X)] \\ &= \mathbb{E}\left[\sum_{i \leq n} W_{i,n}^2(X)\right] \\ &\leq \mathbb{E}\left[\max_{i \leq n} W_{i,n}(X) \underbrace{\sum_{i \leq n} W_{i,n}(X)}_{=1}\right] \\ &= \underbrace{\mathbb{E}\left[\max_{i \leq n} W_{i,n}(X)\right]}_{\rightarrow 0} \end{aligned}$$

On en déduit donc que  $\|\eta_n - \eta\|_{L^2} \rightarrow 0$ , d'après le théorème définit en préambule, cela implique que  $\eta_n$  est bien un estimateur consistant. De plus la forme de la loi  $\mathbb{P}_{X,Y}$  n'entre en compte à aucun moment dans la démonstration, ce qui prouve la consistance universelle de l'estimateur.

*Fin de la preuve.*

Grâce au Théorème de Stone, nous avons maintenant un ensemble de règles qui nous permettent de construire un estimateur pondéré universellement consistant et donc ayant des performances satisfaisantes. Nous allons maintenant voir une condition sur la relation entre l'hyperparamètre  $k$  et la taille de notre jeu de données  $n$  qui nous garantit la consistance universelle du kNN. Avant de passer à la démonstration, nous allons devoir démontrer quelques lemmes qui nous permettront gérer quelques cas particuliers.

**Lemme 4** Soit  $\mu$  une loi de probabilité et  $k$  un hyperparamètre dépendant éventuellement de la taille de notre jeu de données. On définit  $\text{Supp}(\mu) := \{x \in \mathbb{R}^p \mid \forall \epsilon > 0, \mathcal{B}(x, \epsilon)\}$ , il s'agit du plus petit fermé de mesure = 1 par  $\mu$ . Soit  $x \in \text{Supp}(\mu)$ , alors si  $k/n \rightarrow 0$ , on a :

$$\|X_{(k)}(x) - x\| \rightarrow 0 \quad \mathbb{P} - p.s.$$

Intuitivement, ce lemme nous dit que si la valeur choisie de l'hyperparamètre  $k$  est négligeable devant notre nombre d'observation  $n$ , alors il est très probable que plus on ajoute de nouvelles données et plus les  $k$  plus proches voisins de  $x$  ( $X_i(x)$ ) $_{i \leq k}$  seront proches de  $x$ .

*Preuve du lemme :*

Soit  $\epsilon > 0$  fixé et  $x \in \text{Supp}(\mu)$ . Montrons l'égalité d'ensemble  $\{\|X_{(k)}(x) - x\| > \epsilon\} = \left\{ \frac{1}{n} \sum_{i \leq n} \mathbb{1}_{[X_i \in \mathcal{B}(x, \epsilon)]} < \frac{k}{n} \right\}$ .

Cas 1 : Soit  $\omega \in \{\|X_{(k)}(x) - x\| > \epsilon\}$

Alors puisque le  $k$ -ième plus proche voisin de  $x$  est hors de la boule centrée en  $x$  de rayon  $\epsilon$  alors les  $k - n + 1$  suivants le sont aussi, donc on a au plus  $k - 1$  voisins de  $x$  compris dans la dite boule. On en déduit :

$$\frac{1}{n} \sum_{i \leq n} \mathbb{1}_{[X_i(\omega) \in \mathcal{B}(x, \epsilon)]} \leq \frac{k-1}{n} < \frac{k}{n}$$

Donc  $\omega \in \left\{ \frac{1}{n} \sum_{i \leq n} \mathbb{1}_{[X_i \in \mathcal{B}(x, \epsilon)]} < \frac{k}{n} \right\}$ .

Cas 2 : Soit  $\omega \in \left\{ \frac{1}{n} \sum_{i \leq n} \mathbb{1}_{[X_i \in \mathcal{B}(x, \epsilon)]} < \frac{k}{n} \right\}$

Alors on en déduit que :  $\sum_{i \leq n} \mathbb{1}_{[X_i(\omega) \in \mathcal{B}(x, \epsilon)]} \leq k - 1$ , ce qui signifie qu'il y a au plus  $k - 1$  voisins de  $x$  contenu dans la boule centrée en  $x$  de rayon  $\epsilon$ . Les  $X_i$  étant rangés dans l'ordre croissant de leur distance par rapport à  $x$  on en déduit que les  $k - 1$  voisins possiblement compris dans la boule sont forcément ses  $k - 1$  plus proches. Naturellement on en déduit que le  $k$ -ième n'est pas compris dans la boule, autrement dit  $\|X_{(k)}(x, \omega) - x\| > \epsilon$ . Donc  $\omega \in \{\|X_{(k)}(x) - x\| > \epsilon\}$ . On en déduit l'égalité par double inclusion.

On rappel que  $x \in \text{Supp}(\mu)$ , donc  $\forall \epsilon > 0, \mu(\mathcal{B}(x, \epsilon)) > 0$ . On déduit de l'égalité d'ensemble grâce à la loi forte des grands nombres et au fait que  $k/n \rightarrow 0$  que :

$$\forall \epsilon > 0, \mathbb{P}\left(\lim_{n \rightarrow \infty} \|X_{(k)}(x) - x\| > \epsilon\right) = \mathbb{P}(\mu(\mathcal{B}(x, \epsilon)) < 0) = 0$$

Donc :  $\mathbb{P}(\lim_{n \rightarrow \infty} \|X_{(k)}(x) - x\| = 0) = 1 \Leftrightarrow \|X_{(k)}(x) - x\| \rightarrow 0 \quad \mathbb{P} - p.s.$

*Fin de la preuve.*

**Lemme 5** Soit  $x' \in \mathbb{R}^p$ , et  $\nu$  une mesure de probabilités sur  $\mathbb{R}^p$ .  $\forall a \geq 0$ , on définit :

$$\mathcal{B}_a(x') = \left\{ x \in \mathbb{R}^p \mid \nu(\mathcal{B}(x, \|x - x'\|)) \leq a \right\}$$

Alors :

$$\nu(\mathcal{B}_a(x')) \leq \gamma_p a$$

ou  $\gamma_p$  une constante positive qui ne dépend que de la dimension  $d$  de l'espace dans lequel nos observations prennent leur valeurs.

Preuve du lemme :

On définit le cône de centre  $x$  et de rayon  $\theta$  par l'ensemble  $\mathcal{C}(x, \theta) = \left\{ y \in \mathbb{R}^p \mid \widehat{(x, y)} \leq \theta \right\}$ . Alors  $\forall z \in \mathbb{R}^p$ ,  $\mathcal{C}(x, \theta) + z$  représente la translation du cône par  $z$ . On admet le lemme suivant qui nous dit que pour un  $\theta \in [0, \pi/2]$  fixé il existe  $(z_i)_{i \leq d} \subset \mathbb{R}^p$  tel que :

$$\bigcup_{i=1}^d \mathcal{C}(z_i, \theta) = \mathbb{R}^p$$

Soit  $x' \in \mathbb{R}^p$ , on considère une famille de cônes centrés en  $x'$  et d'angle  $\pi/6$  :  $(\mathcal{C}_i)_{i \leq \gamma_p}$ , tel que  $\gamma_p$  soit l'entier minimal de sorte à ce que la famille de cône forme un recouvrement de  $\mathbb{R}^p$ , autrement dit :

$$\gamma_p = \arg \min \left\{ d \in \mathbb{N}, \bigcup_{i=1}^d \mathcal{C}_i = \mathbb{R}^p \right\}$$

Pour démontrer ce lemme il est important de remarquer que si  $y, z \in \mathcal{C}(x, \pi/6)$  et  $\|y\| \leq \|z\|$  alors  $\|y - z\| < \|z\|$ . En effet :

$$\begin{aligned} \|y - z\|^2 &= \|y\|^2 + \|z\|^2 - 2y.z = \|y\|^2 + \|z\|^2 - 2\|y\|\|z\|\cos(\pi/3) \\ &= \|y\|^2 + \|z\|^2 - \|y\|\|z\| < \|z\|^2 \end{aligned}$$

En outre on a :

$$\begin{aligned} \nu(\mathcal{B}_a(x')) &= \nu(\mathcal{B}_a(x') \bigcap \bigcup_{i \leq \gamma_p} \mathcal{C}_i) \\ &\leq \sum_{i \leq \gamma_p} \nu(\mathcal{B}_a(x') \cap \mathcal{C}_i) \end{aligned}$$

Soit  $x^* \in \mathcal{B}_a(x') \cap \mathcal{C}_i$  alors d'après la propriété des cônes démontrée plus haut, on a :

$$\forall y \in \mathcal{B}_a(x') \cap \mathcal{C}_i \cap \mathcal{B}(x', \|x^* - x'\|), \|y - x'\| \leq \|x^* - x'\| \Rightarrow \|y - x^*\| < \|x' - x^*\|$$

Donc :

$$\mathcal{B}_a(x') \cap \mathcal{C}_i \cap \mathcal{B}(x', \|x^* - x'\|) \subseteq \mathcal{B}(x^*, \|x^* - x'\|)$$

De ce fait on peut en déduire :

$$\nu(\mathcal{B}_a(x') \cap \mathcal{C}_i) = \nu(\mathcal{B}_a(x') \cap \mathcal{C}_i \cap \mathcal{B}(x', \|x^* - x'\|)) \leq \nu(\mathcal{B}(x^*, \|x^* - x'\|)) \leq a$$

Ainsi :

$$\nu(\mathcal{B}_a(x')) \leq \gamma_p a$$

Fin de la preuve.

La principale conséquence de ce lemme est que le nombre de points dans  $(X_i)_{i \leq n}$  pour lesquels  $X$  figure parmi les  $ppv$  ne dépasse pas une constante fois  $k$ , c'est ce que nous allons démontrer dans le corollaire suivant.

**Corollaire 3** *Si les égalités entre les distances se produisent avec probabilités zero, alors on a :*

$$\sum_{i \leq n} \mathbb{1}_{X \text{ parmi les } k-\text{ppv de } X_i \text{ dans } \{X_1, \dots, X_{i-1}, X, X_{i+1}, \dots, X_n\}} \leq k\gamma_p \quad \mathbb{P}-p.s.$$

*Il est à noter que pour que ce corollaire soit vrai, il faut qu'en probabilité il n'existe pas de  $pv$  de  $x$  à la même, ce qui est le cas lorsque  $X$  a une loi à densité. On considérera que cette hypothèse comme vrai dans la suite car dans la pratique il est peu probable que deux points différents de notre jeu de données se trouvent exactement à la même distance d'une nouvelle observation.*

*Preuve du corollaire :*

On applique le lemme avec  $a := k/n$  et  $\nu = \mu_n$  la mesure empirique associée à  $(X_i)_{i \leq n}$ . On a donc  $\mathbb{P}-p.s$  :

$$\begin{aligned} X_i \in \mathcal{B}_{k/n}(X) &\Leftrightarrow \mu_n(\mathcal{B}(X_i, \|X - X_i\|)) \leq k/n \\ &\Leftrightarrow X \text{ parmi les } k-\text{ppv de } X_i \text{ dans } \{X_1, \dots, X_{i-1}, X, X_{i+1}, \dots, X_n\} \end{aligned}$$

La deuxième équivalence vient de l'égalité d'ensemble montrée dans la démonstration du premier lemme. De ce fait en appliquant le lemme on a  $\mathbb{P}-p.s$  :

$$\begin{aligned} \sum_{i \leq n} \mathbb{1}_{X \text{ parmi les } k-\text{ppv de } X_i \text{ dans } \{X_1, \dots, X_{i-1}, X, X_{i+1}, \dots, X_n\}} &= \sum_{i \leq n} \mathbb{1}_{[X_i \in \mathcal{B}_{k/n}(X)]} \\ &= n \mu_n(\mathcal{B}_{k/n}(X)) \\ &\leq n * k/n * \gamma_p = k\gamma_p \end{aligned}$$

*Fin de la preuve.*

Il ne nous reste plus qu'un dernier lemme pour montrer que l'algorithme  $kNN$  sous de bonnes hypothèses vérifie les conditions du théorème de Stone.

**Lemme 6** *Supposons que les égalités entre les distances se produisent avec probabilités zéro. Alors pour toute fonction borélienne  $f : \mathbb{R}^p \mapsto \mathbb{R}$  tel que  $f(X) \in L^1$ , on a :*

$$\sum_{i \leq n} \mathbb{E}|f(X_{(i)}(X))| \leq k\gamma_p \mathbb{E}|f(X)|$$

*Preuve du lemme :*

En remarquant que pour toute permutation  $\sigma \in \mathcal{S}_n$ , si les  $(X_i)_{i \leq n}$  iid, alors  $(X_{\sigma(i)})_{i \leq n} = \mathcal{L}(X_i)_{i \leq n}$  en loi, alors en prenant  $f$  comme dans l'énoncé on a :

$$\begin{aligned} \sum_{i \leq n} \mathbb{E}|f(X_{(i)}(X))| &= \mathbb{E}\left[\sum_{i \leq n} |f(X_i)| \mathbb{1}_{X \text{ parmi les } k-\text{ppv de } X \text{ dans } \{X_1, \dots, X_n\}}\right] \\ &= \mathbb{E}[|f(X)| \sum_{i \leq n} \mathbb{1}_{X \text{ parmi les } k-\text{ppv de } X_i \text{ dans } \{X_1, \dots, X_{i-1}, X, X_{i+1}, \dots, X_n\}}] \\ &\leq \mathbb{E}[|f(X)| k\gamma_p] = \mathbb{E}|f(X)| k\gamma_p \end{aligned}$$

*Fin de la preuve.*

Nous sommes maintenant en mesure de démontrer la consistance du  $kNN$  sous de bonnes hypothèses.

**Théorème 9** *Supposons que  $k_n$  une constante dépendant du nombre d'observation de notre jeu de données tel que :*  $\begin{cases} k_n \rightarrow +\infty \\ k_n/n \rightarrow 0 \end{cases}$

*Alors la règle de classification  $\Phi^{kNN}$  est universellement consistante.*

*Preuve du théorème :*

On commence par poser la fonction de poids vérifiant :

$$\forall x \in \mathbb{R}^p, W_{i,n}(x) = \begin{cases} \frac{1}{k_n} & \text{si } X_i \text{ est parmi les } k - ppv \text{ de } x \\ 0 & \text{sinon} \end{cases}$$

Puisque  $k_n \rightarrow +\infty$  la condition (3) du théorème de Stone est évidente. Pour la condition (2) toujours puisque  $k_n \rightarrow +\infty$  on a :

$$\begin{aligned} \mathbb{E}\left[\sum_{i \leq n} W_{i,n}(X) \mathbb{1}_{\|X_i - X\| > \epsilon}\right] &= \mathbb{E}\left[\sum_{i \leq k_n} \frac{1}{k_n} \mathbb{1}_{\|X_{(i)} - X\| > \epsilon}\right] \\ &= \frac{1}{k_n} \sum_{i \leq k_n} \mathbb{P}(\|X_{(i)}(X) - X\| > \epsilon) \\ &\leq \max_{i \leq k_n} \mathbb{P}(\|X_{(i)}(X) - X\| > \epsilon) \end{aligned}$$

Or on rappelle le lemme qui nous dit que si  $x \in Supp(\mu)$  alors si  $k_n/n \rightarrow 0$  on a :

$$\|X_{(k)}(x) - x\| \rightarrow 0$$

Donc par convergence dominée :

$$\begin{aligned} \lim_{n \rightarrow +\infty} \mathbb{P}(\|X_{(i)}(X) - X\| > \epsilon) &= \lim_{n \rightarrow +\infty} \int_{\mathbb{R}^p} \mathbb{P}(\|X_{(i)}(x) - x\| > \epsilon) d\mu(x) \\ &= \int_{Supp(\mu)} \lim_{n \rightarrow +\infty} \mathbb{P}(\|X_{(i)}(x) - x\| > \epsilon) d\mu(x) \\ &= 0 \end{aligned}$$

Pour finir le dernier lemme nous montre que la condition (1) du théorème de Stone est vérifiée. On en conclut  $\Phi^{kNN}$  est universellement consistante.

*Fin de la preuve.*

Ainsi, on a démontré que si l'on choisissait un hyperparamètre  $k_n$  tel que :  $\begin{cases} k_n \rightarrow +\infty \\ k_n/n \rightarrow 0 \end{cases}$  notre algorithme était en théorie sensé nous donner des résultats optimaux. Afin de s'assurer que l'algorithme fonctionne de manière optimale sur nos données, il faudrait que l'on choisisse un  $k_n = u(n)$  tel que  $u$  soit coercive et négligeable devant  $n$ . La première intuition qui nous vient en tête serait de tester le  $kNN$  avec  $k_n := \lceil n^{1/j} \rceil$  en regardant les résultats obtenus selon le  $j > 1$  choisi.

### 3.4.5 Complexité du kNN

Le kNN fait partie de l'ensemble des *algorithmes d'apprentissage paresseux* car la procédure d'apprentissage ne consiste qu'en une seule opération : le stockage des données d'entraînement dans la mémoire, soit une complexité constante  $\mathcal{O}(1)$ .

En revanche procédure de prédiction se fait en 2 étapes :

(1) : Calcul de la distance entre la nouvelle observation et chaque observation du jeu de données, soit une complexité de  $\mathcal{O}(np)$ .

(2) : Trouver les  $k$  plus petites distances, soit  $\mathcal{O}(n \log(k))$ .

### 3.4.6 Résultats pratique

#### i Premier test

Les premiers tests sans optimisation manuelle des hyperparamètres nous donne une précision d'environ 0.90. Les hyperparamètre sont sélectionné de manière optimale en fonction de la forme du *training set*. Nous allons tenter de nous approcher de cette précision à l'aide des théorèmes que l'on a démontré précédemment.

#### ii Sélection des hyperparamètres

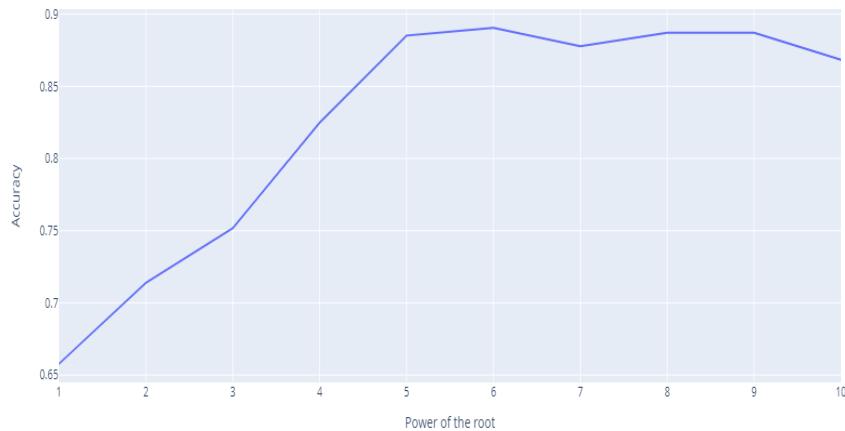


FIGURE 7 – Graphique de l'évolution de la précision du modèle en fonction de la racine nième de  $1/n$ .

On peut voir que l'arbre a l'air de donner des résultats optimaux pour la racine 6ème racine de  $1/n$ . Afin de tester la généralisation du modèle nous allons utiliser la technique de 5-validation croisée stratifiée afin de vérifier la généralisation de notre algorithme et de s'assurer de la représentativité des *fold*.

#### iii Généralisation du modèle

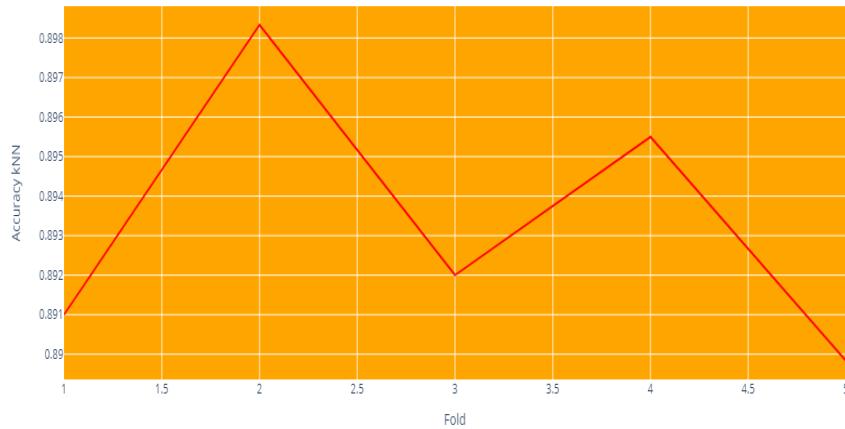


FIGURE 8 – Graphique de l'évolution de la précision du modèle en fonction du fold sur lequel il a été testé.

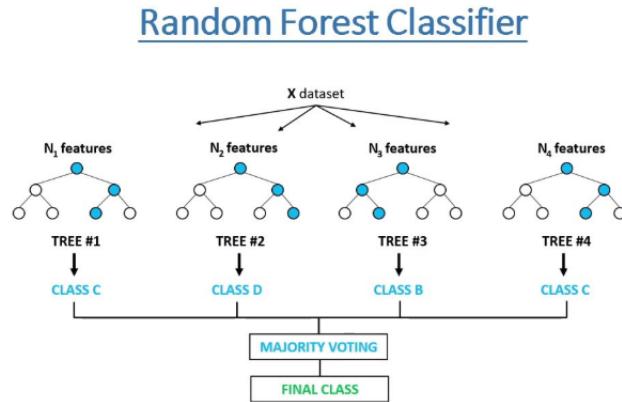
On voit qu'on obtient en moyenne une précision d'environ 0.89, c'est à dire que nous avons réussi à atteindre des performances quasi optimales en ne modifiant qu'un seul hyperparamètre.

### 3.5 Forêts aléatoires

Le code utilisé pour obtenir les résultats de cette section est dans le notebook 'RandomForest.ipynb'

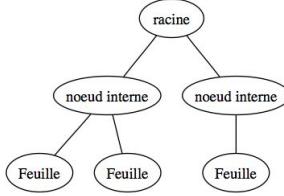
#### 3.5.1 Théorie sur les forêts aléatoires

Une forêt aléatoire est composée de plusieurs arbres aléatoires qui vont chacun proposer une réponse au problème de régression ou de classification. Puis on obtient un résultat unique par la majorité.



Comment un arbre permet d'obtenir une prédiction ?

Un arbre est constitué d'une racine de noeud intermédiaire et de feuille. Chaque noeud divise l'espace des variables en 2 ou plus sous-espaces selon une fonction qui dépend des données. Chaque branche représente un scénario de décision et un résultat de la variable d'intérêt, dans notre cas il s'agit de la variable "winner" qui vaut 0 ou 1 en fonction du gagnant du match.



### 3.5.2 classification supervisée

On a choisi la variable réponse ou variable d'intérêt Y dans un espace discret  $\mathcal{Y} = \{0,1\}$  et X dans  $\mathcal{X}$  (souvent  $\mathcal{X} = \mathbb{R}^p$ ) les variables d'entrée. On cherche ici à résoudre un problème de classification supervisée mais l'algorithme CART peut également être appliquée aux problèmes de régression (où Y dans  $\mathbb{R}$  )

En classification, on cherche à estimer :  $\forall y \in \mathcal{Y}, P(Y = y | X = x)$  La probabilité que Y appartienne à chacune des classes qui forment Y, conditionnellement à X.

### 3.5.3 Algorithme CART : Classification and Regression Trees

Le principe général de l'algorithme CART consiste à partitionner récursivement l'espace d'entrée  $\mathcal{X}$  de façon binaire (on pourrait le diviser en plus de groupes mais c'est typiquement ce qu'on choisit) puis on détermine la sous partition optimale pour la prédiction.

On procède en 2 étapes : D'abord la construction de l'arbre maximal puis l'élagage.

**3.5.3.1 Construction de l'arbre maximal** La racine de l'arbre est associée à l'espace tout entier, elle contient toutes les observation de l'échantillon d'apprentissage. Puis, à chaque pas du partitionnement on découpe l'espace du noeud en deux sous-parties.

Par exemple :  $\{X^j \leq d\} \cup \{X^j > d\}$  Ici on a fait en sorte d'avoir des variables numériques mais on pourrait tout à fait l'appliquer à des variables catégorielles en divisant les valeurs possibles pour cette variable en 2 ensembles et en appliquant :  $\{X^j \in d\} \cup \{X^j \in d'\}$

A chaque noeud le nombre de découpes possibles du noeud t contenant  $N_t$  éléments en n subset donné par le nombre de Stirling :

$$S(N_t, n) = \frac{1}{n!} \sum_{n=0}^N \binom{j}{n} (-1)^{n-j} j^{N_t}$$

Choisir la meilleure découpe revient à prendre celle qui réduit le plus l'impureté et donc qui permet d'acquérir le plus d'information.

En classification, l'impureté est le plus souvent calculé à partir de l'indice de Gini d'un noeud t qui est donné par la formule :

$$\Phi(t) = \sum_{k=1}^L p_t^k (1 - p_t^k)$$

où L est le nombre de classes et  $p_t^k$  est la proportion d'observations de classe k dans le noeud t.

Diminuer l'indice de pureté de Gini revient à augmenter l'homogénéité des noeuds obtenus. L'homogénéité parfaite correspondrait à un arbre dans lequel chaque noeud ne contient que des observation de la même classe.

L'arbre se développe jusqu'à une condition d'arrêt. En ce qui concerne l'arbre maximal, on ne lui impose pas de condition il se développe jusqu'à ce que chaque feuille contienne qu'une observation ou alors des observations identiques.

Une règle d'arrêt classique est d'imposer un nombre d'éléments minimal dans le noeud en dessous duquel on ne divise plus l'ensemble en deux sous-parties.

Dans les deux cas extrêmes : L'arbre maximal a une très grande variance et un biais faible, il surapprend les données d'apprentissage, et l'arbre "vide", constitué uniquement de la racine a une très grande variance et un biais élevé, on est alors dans un cas de sous-apprentissage. Pour trouver le juste milieu, on procède donc à l'élagage.

**3.5.3.2 Élagage** Dans cette étape, on recherche le sous-arbre optimal à partir de l'arbre maximal. C'est-à-dire celui qui minimise un critère des moindres carrés pénalisé.

Pour être admissibles, les sous-arbres doivent contenir la racine.

On cherche le sous-arbre optimal dans une sous-suite d'arbres "emboîtés". C'est une suite qui démarre par l'arbre maximal et qui est construite de manière itérative.

L'erreur d'ajustement d'un sous-arbre T élagué de l'arbre maximal est :

$$\text{erreur}(T) = \frac{1}{n} \sum_{\text{feuilles de } T} \sum_{\text{observations } (x_i, y_i) \text{ dans } t} (y_i - \bar{y}_t)^2$$

A partir de l'arbre maximal  $T_{max}$  : Pour tout noeud interne t, on note  $T_t$  la branche de T contenant tous les descendants de t.

- Initialisation :  $\alpha_1 = 0, T_1 = \operatorname{argmin}_{T \text{ filiale de } T_{max}} \text{erreur}(T)$
- $T = T_1$  et  $k = 1$
- Itération : Tant que  $|T| > 1$ , Calculer

$$\alpha_k + 1 = \min_{t \text{ noeud interne de } T} \frac{\text{erreur}(t) - \text{erreur}(T_t)}{|T_t| - 1}$$

Elaguer toutes les branches de  $T_t$  telles que  $\text{erreur}(T_t) + \alpha_k + 1|T_t| = \text{erreur}(t) + \alpha_k + 1$

Prendre  $T_k + 1$  le sous arbre obtenu et itérer pour  $T = T_k + 1$  et  $k = k + 1$

- On obtient ainsi une suite d'arbre emboîtés au sens de l'élagage les un des autres telles que  $T_1 > \dots > T_K$  et une suite de paramètres  $\alpha_1; \dots; \alpha_K$

### 3.5.4 Implémentation de l'algorithme

Une forêt aléatoire n'est pas difficile à implémenter. On construit l'algorithme comme suit :

**3.5.4.1 Etape 1 : Construction de X et Y** On choisit Y : la variable que notre forêt va devoir prédire. Ici Y est la variable binaire 'winner' qui vaut 1 si le player 1 gagne le match et 0 sinon. On choisit X qui regroupe les variables dont dispose notre algorithme en entrée. Ici il s'agit de toutes les variables sauf 'winner'.

**3.5.4.2 Etape 2 : Split des données entre train et test** On divise X et Y entre les données que l'on va utiliser pour entraîner les arbres de notre forêt et celles que l'on va utiliser pour tester l'efficacité de notre forêt aléatoire.

**3.5.4.3 Etape 3 : Train & test** On entraîne l'algorithme sur une partie des données puis on tente de prédire Y à partir du reste des données. Nous pouvons dès lors obtenir un résultat. Lors de ce premier test en prenant un nombre d'arbres et une profondeur arbitrairement nous avons obtenu une accuracy de 0.69.

### 3.5.4.4 Etape 4 : Choix des hyperparamètres

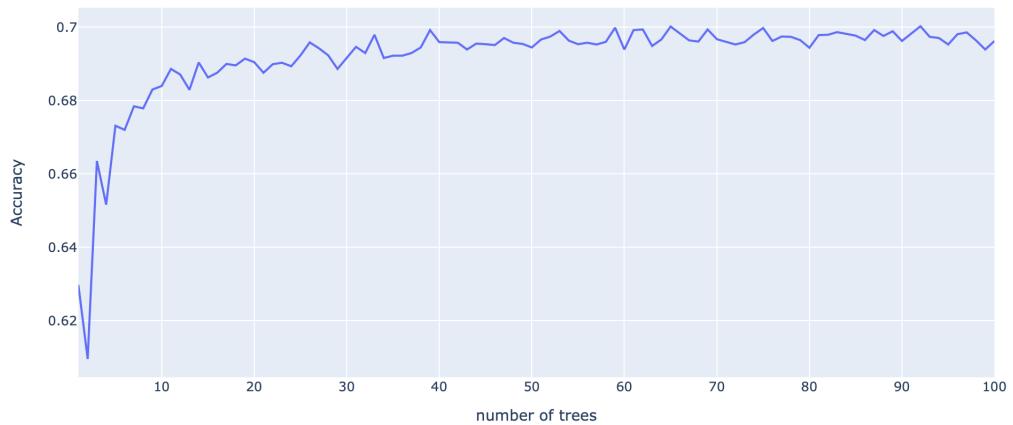
Cette partie est la plus intéressante de l'algorithme car elle permet de faire évoluer la performance de notre forêt sans avoir plus de données. Nous devons choisir les hyperparamètres de notre forêt qui permettent d'obtenir les meilleurs résultats. Il s'agit des paramètres fixés avant l'exécution de l'algorithme. En particulier, nous nous intéressons au nombre d'arbres que doit contenir notre forêt et la profondeur maximale que peut atteindre chaque arbre. Pour trouvé les hyperparamètres optimaux, plusieurs méthodes existent.

On effectue le réglage de ces hyperparamètres par l'expérimentation et non la théorie. Il est quasiment impossible de trouver les hyperparamètres optimaux avant d'exécuter l'algorithme de forêt aléatoire.

Il faut ajuster ces paramètres en faisant plusieurs fois tourner l'algorithme mais en s'assurant de ne pas faire de l'overfitting. On choisit donc un ensemble d'hyperparamètres et on test les résultats obtenus (la précision du modèle) pour chacun, afin d'estimer les paramètres optimaux.

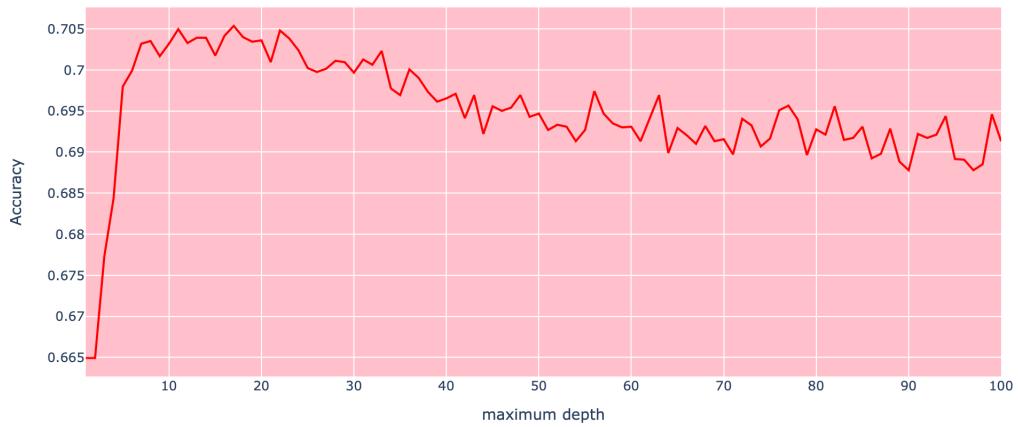
En fixant la profondeur des arbres à 50 noeuds et en calculant la précision du modèle pour différents nombres d'arbres. On obtient les résultats illustrés dans le graphique suivant qui montre une très nette évolution de la précision mais qui cesse à partir d'un certain nombre d'arbres. (il est inutile d'en mettre plus)

Graphique de l'évolution de la précision du modèle en fonction du nombre d'arbres



Ensuite en fixant environ à 35 arbres la taille de la forêt, nous avons cherché la profondeur optimale à fixer pour chaque arbre. On trace de nouveau l'évolution de la précision et on obtient le graphique suivant.

Graphique de l'évolution de la précision du modèle en fonction de la profondeur maximale des arbres



On observe que la précision n'augmente pas proportionnellement à la profondeur maximale des arbres. Il est préférable de fixer la profondeur aux alentours de 20.

Après avoir fixé les hyperparamètres ainsi, on obtient une précision (accuracy) de 0.70. Évidemment,

il existe d'autres hyperparamètres que nous pourrions fixer afin d'affiner le modèle. Cependant l'effet sur la précision ne serait pas forcément conséquent.

## 3.6 Réseau de neurones

### 3.6.1 Motivation

Nous avons pu voir précédemment, notamment dans le cadre de la régression logistique, que le choix de nos variables explicatives avait une grande influence sur la qualité de notre modèle. Est-il important d'inclure des termes polynomiaux, de nouvelles variables ou bien des combinaisons entre tels et tels variables... C'est autant de questions que nous nous sommes posé et auxquelles nous avons proposé des réponses en justifiant nos choix.

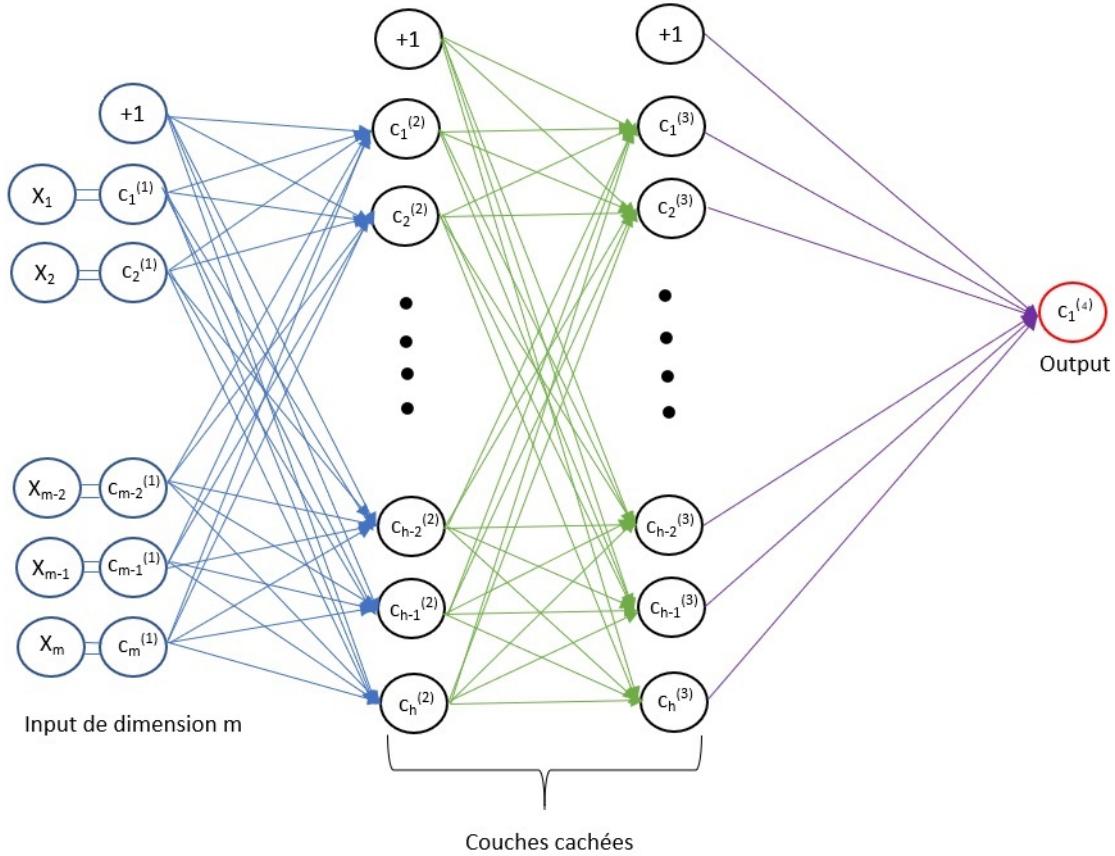
L'intérêt d'utiliser un réseau de neurones est d'apporter une dimension complémentaire à cette démarche. Un réseau de neurones ne nous permettra pas de justifier tel ou tel ajout d'une variable explicative, mais construira lui-même ses propres variables, à partir des données en input, qu'il déduira comme pertinentes lors de la phase d'apprentissage. C'est la fonction principale des couches cachées d'un réseau de neurones. Sachant cela, le seul moyen de s'assurer de faire mieux que le réseau de neurones, et donc rendre son utilisation inutile, serait d'inclure toutes les combinaisons polynomiales entre nos variables explicatives. Le problème est que cela aboutirait à un modèle en  $O(m^2)$  variables si on part de  $m$  variables explicatives. Un nombre beaucoup trop important de variables lorsque  $m$  est grand qui, en pratique, n'est pas valide à cause d'un coût de calcul trop important et d'un risque de sur-apprentissage.

Ainsi, proposer une "approche réseau de neurones" est un plus dans le cadre de notre problématique. Mais quelle structure choisir : peu de couches cachées ou un nombre important (Deep Learning) ?

Nous essayerons les deux. La première sera intéressante pour comprendre la structure et le fonctionnement d'un réseau de neurones d'un point de vue théorique, mathématique, c'est pourquoi nous construirons tout "à la main" et prendrons bien soin de tout démontrer. Dans la pratique, implémenter les parties qui suivent n'a pas grand intérêt puisqu'elles nécessiteront de laisser un ordinateur tourner longtemps pour des résultats sous-optimaux par rapport au Deep Learning avec Keras, qui pour le coût sera beaucoup plus optimal en terme de durée d'apprentissage du modèle. La deuxième approche sera donc intéressante pour la performance et la qualité de la prédiction qu'elle peut nous donner. C'est pourquoi au cours de cette partie "Deep Learning" nous nous focaliserons plutôt sur comment utiliser pertinemment les librairies permettant l'implémentation et l'optimisation de réseaux neuronaux profonds.

### 3.6.2 Réseau de neurones à deux couches cachées

Nous avons choisi 2 couches cachées car cela permet d'aller assez loin dans les calculs pour comprendre comment marche des principes essentiels comme la rétro-propagation, et aussi puisque que c'est la structure recommandée pour notre type de problème : classification binaire avec relativement peu de variables (comparé à un problème du type "computer vision", par exemple, où il est fréquent d'avoir des modélisations à plusieurs dizaines de milliers, voir millions de variables). Nous utiliserons la fonction sigmoid comme fonction d'activation afin de rester dans l'idée selon laquelle cette partie serait une sorte de suite améliorante du modèle logit. Nous appellerons  $\theta^{(i)}$  la matrice des poids associée à la  $i^{\text{ème}}$  couche. Par exemple les flèches bleus de l'illustration ci-dessous représentent la matrice des poids  $\theta^{(1)}$  :



$c_i^{(k)}$  représente l'activation du neurone  $i$  de la couche  $k$ . Les "+1" sont les termes de biais (qui peuvent se voir comme un intercept pour chaque couche). Par convention les couches cachées auront le même nombre de neurones :  $h$ . En général on choisit un  $h$  compris entre  $m$  et  $4*m$ .

### i Forward Propagation

Soit  $g$  la fonction d'activation et  $\#(k)$  le cardinal de la couche  $k$ , c'est à dire le nombre de neurones la composant (donc on ne compte pas le terme de biais dans  $\#(k)$ ). Suivant l'illustration ci-dessus, l'activation du  $i^{\text{ème}}$  neurones de la couche  $k$  est donnée par :

$$c_i^{(k)} = g \left( \theta_{i,0}^{(k-1)} + \sum_{j=1}^{\#(k-1)} \theta_{i,j}^{(k-1)} c_j^{(k-1)} \right) = g(\gamma_i^{(k)})$$

On remarque alors que  $\theta^{(k)} \in \mathcal{M}_{(\#(k+1)) \times (\#(k)+1)}(\mathbb{R})$ .

### ii Rétro-propagation

C'est la phase d'optimisation du réseau de neurones. Après avoir performé une "forward propagation", l'idée est de calculer l'erreur de prédiction du réseau puis de la repartir sur chaque neurone de chaque couche, en remontant le réseau, pour pouvoir modifier les matrices de poids associées grâce à une descente de gradient. D'où le nom de cette méthode : on "remonte" le réseau pour optimiser les poids responsables de l'erreur relativement à leur contribution à cette dernière. La première étape est donc d'expliquer la fonction de coût :

Nous allons appliquer une pénalisation Ridge à notre algorithme de descente de gradient durant la phase d'apprentissage des  $\theta^{(k)}$  afin d'éviter un sur-apprentissage du réseau.  $J^{\{R\}}$  sera alors défini de la

façon suivante, à l'image de (7) :

$$J^{\{R\}}(\theta) = \left( -\frac{1}{n} \sum_{i=1}^n [y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))] \right) + \frac{\lambda}{2m} \sum_{k=1}^3 \sum_{i=1}^{\#(k+1)} \sum_{j=1}^{\#(k)} (\theta_{i,j}^{(k)})^2$$

En pratique calculer le gradient et la fonction de coût sur toutes les observations, à chaque itération de notre descente de gradient, est coûteux en temps de calcul et sous optimal par rapport à une méthode de descente de gradient stochastique. C'est pourquoi nous allons introduire dans la section suivante cette façon de contourner ce problème.

### iii Descente de gradient stochastique

L'idée est d'effectuer une descente de gradient sur une seule observation tirée aléatoirement dans échantillon d'apprentissage, qu'on notera  $(x, y)$ , à chaque itération. L'avantage est que dans le cadre d'un "grand"  $n$  (taille de notre échantillon d'apprentissage), comme dans notre cas, on évalue une seule fois le gradient par itération ce qui rend l'algorithme beaucoup plus efficient puisqu'il est désormais indépendant de  $n$ . En pratique, avant de calculer directement le gradient de chaque poids de notre réseau de neurones, commençons par définir une notion qui nous sera utile pour simplifier nos expressions, mais également mieux les comprendre :

On appelle  $\delta_i^{(k)}$  "l'erreur" du neurone  $i$  de la couche  $k$  et on a alors :

$$\delta^{(4)} = c_1^{(4)} - y$$

Et en posant  $\delta^{(k)} = (\delta_0^{(k)}, \delta_1^{(k)}, \dots, \delta_{\#(k)}^{(k)})^T$  et  $\delta_{(-0)}^{(k)} = (\delta_1^{(k)}, \dots, \delta_{\#(k)}^{(k)})^T$ , par rétro-propagation on a :

$$\delta_i^{(k)} = (\theta^{(k)})^T \delta_{(-0)}^{(k+1)} \text{ pour } k \in \{3, 2\} \text{ et } i \in \{0, \dots, \#(k)\}$$

Bien sûr on ne cherche pas à calculer  $\delta^{(1)}$  puisque calculer l'erreur de la première couche, c'est à dire de l'input, n'a pas de sens. Intuitivement, la formule ci-dessus s'interprète par le fait que l'erreur d'un neurone  $i$  d'une couche  $k$  est responsable de l'erreur porté par tous les neurones de la couche supérieur proportionnellement aux poids qui lui sont rattachés, c'est à dire la  $i^{\text{ème}}$  colonne de la matrice  $\theta^{(k)}$  ou encore la  $i^{\text{ème}}$  ligne de la matrice  $(\theta^{(k)})^T$ . Maintenant que nous avons ce principe en tête, nous allons voir comment implémenter la rétro-propagation. Le but est d'appliquer une descente de gradient du type  $\forall k \in \{1, 2, 3\}, \forall i \in \{1, \dots, \#(k+1)\}, \forall j \in \{1, \dots, \#(k) + 1\}$  :

$$\theta_{i,j}^{(k)} := \theta_{i,j}^{(k)} - \alpha \frac{\partial}{\partial \theta_{i,j}^{(k)}} J^{\{R\}}(\theta)$$

afin d'optimiser chaque poids du réseau. Commençons par noter que notre fonction de coût s'écrit maintenant :

$$J^{\{R\}}(\theta) = -(y \log(h_\theta(x)) + (1 - y) \log(1 - h_\theta(x))) + \frac{\lambda}{2m} \sum_{k=1}^3 \sum_{i=1}^{\#(k+1)} \sum_{j=1}^{\#(k)} (\theta_{i,j}^{(k)})^2$$

C'est maintenant qu'intervient la dérivation en chaîne. Nous allons distinguer le cas  $j=0$ , qui correspond à l'intercept de chaque matrice, car nous verrons que le calcul diffère dans ce cas :

$$\begin{aligned}
\frac{\partial}{\partial \theta_{1,j \neq 0}^{(3)}} J^{\{R\}}(\theta) &= \frac{\partial J(\theta)}{\partial \gamma_1^{(4)}} \frac{\partial \gamma_1^{(4)}}{\partial \theta_{1,j \neq 0}^{(3)}} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \frac{\partial J(\theta)}{\partial c_1^{(4)}} \frac{\partial c_1^{(4)}}{\partial \gamma_1^{(4)}} \frac{\partial \gamma_1^{(4)}}{\partial \theta_{1,j \neq 0}^{(3)}} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \frac{\partial J(\theta)}{\partial c_1^{(4)}} \frac{\partial g(\gamma_1^{(4)})}{\partial \gamma_1^{(4)}} \frac{\partial \gamma_1^{(4)}}{\partial \theta_{1,j \neq 0}^{(3)}} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \frac{\partial J(\theta)}{\partial c_1^{(4)}} g'(\gamma_1^{(4)}) \frac{\partial \gamma_1^{(4)}}{\partial \theta_{1,j \neq 0}^{(3)}} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \frac{\partial J(\theta)}{\partial c_1^{(4)}} g'(\gamma_1^{(4)}) \frac{\partial [\theta_{1,0}^{(4-1)} + \sum_{p=1}^{\#(4-1)} \theta_{1,p}^{(4-1)} c_p^{(4-1)}]}{\partial \theta_{1,j \neq 0}^{(3)}} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \frac{\partial J(\theta)}{\partial c_1^{(4)}} g'(\gamma_1^{(4)}) c_j^{(3)} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m}
\end{aligned}$$

Or :

$$g'(x) = \frac{-(-\exp(-x))}{(1 + \exp(-x))^2} = \frac{1 + \exp(-x) - 1}{(1 + \exp(-x))^2} = g(x) - \frac{1}{(1 + \exp(-x))^2} = g(x) - g(x)^2 = g(x)(1 - g(x))$$

Et :

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial c_1^{(4)}} &= \frac{\partial J(\theta)}{\partial h_\theta(x)} = - \left( y \frac{1}{h_\theta(x)} + (1-y) \frac{(-1)}{1-h_\theta(x)} \right) \\
&= \left( \frac{-y(1-h_\theta(x))}{h_\theta(x)(1-h_\theta(x))} + \frac{(1-y)h_\theta(x)}{(1-h_\theta(x))h_\theta(x)} \right) \\
&= \left( \frac{-y+h_\theta(x)}{h_\theta(x)(1-h_\theta(x))} \right) \\
&= \left( \frac{-y+c_1^{(4)}}{c_1^{(4)}(1-c_1^{(4)})} \right)
\end{aligned}$$

Donc

$$\begin{aligned}
\frac{\partial}{\partial \theta_{1,j \neq 0}^{(3)}} J^{\{R\}}(\theta) &= \frac{\partial J(\theta)}{\partial c_1^{(4)}} g'(\gamma_1^{(4)}) c_j^{(3)} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \left( \frac{-y+c_1^{(4)}}{c_1^{(4)}(1-c_1^{(4)})} \right) g(\gamma_1^{(4)}) (1 - g(\gamma_1^{(4)})) c_j^{(3)} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= \left( \frac{-y+c_1^{(4)}}{c_1^{(4)}(1-c_1^{(4)})} \right) c_1^{(4)} (1 - c_1^{(4)}) c_j^{(3)} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= (c_1^{(4)} - y) c_j^{(3)} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m} \\
&= (\delta^{(4)}) c_j^{(3)} + \frac{\lambda \theta_{1,j \neq 0}^{(3)}}{m}
\end{aligned}$$

Et pour  $j=0$  on a (car on ne pénalise pas l'intercept) :

$$\frac{\partial}{\partial \theta_{1,0}^{(3)}} J^{\{R\}}(\theta) = (\delta^{(4)}) c_0^3$$

Pour les couches inférieures, nous allons voir qu'il existe une relation de récurrence à partir des poids et des activations calculés dans les couches supérieurs, toujours en utilisant la dérivation en chaîne nous avons :

$$\begin{aligned}
\frac{\partial}{\partial \theta_{i,j \neq 0}^{(2)}} J^{\{R\}}(\theta) &= \sum_{w=1}^{\#(3)} \frac{\partial J(\theta)}{\partial \gamma_w^{(3)}} \frac{\partial \gamma_w^{(3)}}{\partial \theta_{i,j \neq 0}^{(2)}} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} \\
&= \sum_{w=1}^{\#(3)} \frac{\partial J(\theta)}{\partial \gamma_w^{(3)}} \underbrace{\left[ \frac{\partial \theta_{w,0}^{(2)}}{\partial \theta_{i,j \neq 0}^{(2)}} + \sum_{d=1}^{\#(2)} \theta_{w,d}^{(2)} c_d^{(2)} \right]}_{c_j^{(2)} \mathbf{1}_{\{i=w\}}} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} \\
&= \frac{\partial J(\theta)}{\partial \gamma_i^{(3)}} c_j^{(2)} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} \\
&= \sum_{p=1}^{\#(4)} \frac{\partial J(\theta)}{\partial \gamma_p^{(4)}} \frac{\partial \gamma_p^{(4)}}{\partial \gamma_i^{(3)}} c_j^{(2)} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} \quad || \text{ or } \#(4) = 1 \\
&= \frac{\partial J(\theta)}{\partial \gamma_1^{(4)}} \frac{\partial \gamma_1^{(4)}}{\partial \gamma_i^{(3)}} c_j^{(2)} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} \\
&= \delta^{(4)} \frac{\partial \gamma_1^{(4)}}{\partial \gamma_i^{(3)}} c_j^{(2)} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m}
\end{aligned}$$

D'après les calculs effectués précédemment. Par ailleurs, on a que :

$$\frac{\partial \gamma_1^{(4)}}{\partial \gamma_i^{(3)}} = \frac{\partial \left[ \theta_{1,0}^{(3)} + \sum_{j=1}^{\#(3)} \theta_{1,j}^{(3)} \underbrace{c_j^{(3)}}_{g(\gamma_j^{(3)})} \right]}{\partial \gamma_i^{(3)}} = \theta_{1,i}^{(3)} g'(\gamma_i^{(3)}) = \theta_{1,i}^{(3)} g(\gamma_i^{(3)}) (1 - g(\gamma_i^{(3)}))$$

Et on obtient donc (en sachant que  $\theta^{(3)}$  n'a que  $\#(3+1)=\#(4)=1$  seule ligne) :

$$\begin{aligned}
\frac{\partial}{\partial \theta_{i,j \neq 0}^{(2)}} J^{\{R\}}(\theta) &= \overbrace{\delta^{(4)} \theta_{1,i}^{(3)} g(\gamma_i^{(3)}) (1 - g(\gamma_i^{(3)})) c_j^{(2)}}^{(*)} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} = \overbrace{\delta^{(4)} \theta_{1,i}^{(3)} c_i^{(3)} (1 - c_i^{(3)}) c_j^{(2)}}^{(*)} + \frac{\lambda \theta_{i,j \neq 0}^{(2)}}{m} \\
\frac{\partial}{\partial \theta_{i,0}^{(2)}} J^{\{R\}}(\theta) &= \overbrace{\delta^{(4)} \theta_{1,i}^{(3)} c_i^{(3)} (1 - c_i^{(3)}) c_0^{(2)}}^{(*)}
\end{aligned}$$

En passant à une écriture vectorielle, on retrouve bien l'expression par récurrence de  $\delta_i^3$  ici (\*), ce qui rend notre expression calculable en pratique par récurrence. On commencera par calculer les gradients des couches supérieures, puis on remontera le réseau pour calculer les gradients des couches inférieures. C'est tout le principe de la rétro-propagation permis par la règle de dérivation en chaîne.

Terminons avec le calcul des gradients de la troisième et dernière matrice de poids ( $\theta^{(1)}$ ) :

$$\begin{aligned}
\frac{\partial}{\partial \theta_{i,j \neq 0}^{(1)}} J^{\{R\}}(\theta) &= \sum_{w=1}^{\#(2)} \frac{\partial J(\theta)}{\partial \gamma_w^{(2)}} \frac{\partial \gamma_w^{(2)}}{\partial \theta_{i,j \neq 0}^{(1)}} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \sum_{w=1}^{\#(2)} \frac{\partial J(\theta)}{\partial \gamma_w^{(2)}} \underbrace{\left[ \frac{\partial \theta_{w,0}^{(1)} + \sum_{d=1}^{\#(1)} \theta_{w,d}^{(1)} c_d^{(1)}}{\partial \theta_{i,j \neq 0}^{(1)}} \right]}_{c_j^{(1)} \mathbf{1}_{\{i=w\}}} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \frac{\partial J(\theta)}{\partial \gamma_i^{(2)}} c_j^{(1)} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \sum_{p=1}^{\#(3)} \frac{\partial J(\theta)}{\partial \gamma_p^{(3)}} \frac{\partial \gamma_p^{(3)}}{\partial \gamma_i^{(2)}} c_j^{(1)} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \sum_{p=1}^{\#(3)} \frac{\partial J(\theta)}{\partial \gamma_p^{(3)}} \frac{\partial \left[ \theta_{p,0}^{(2)} + \sum_{d=1}^{\#(2)} \theta_{p,d}^{(2)} \underbrace{c_d^{(2)}}_{g(\gamma_d^{(2)})} \right]}{\partial \gamma_i^{(2)}} c_j^{(1)} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \sum_{p=1}^{\#(3)} \frac{\partial J(\theta)}{\partial \gamma_p^{(3)}} \theta_{p,i}^{(2)} g'(\gamma_i^{(2)}) c_j^{(1)} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \sum_{p=1}^{\#(3)} \left( \delta^{(4)} \theta_{1,p}^{(3)} c_p^{(3)} (1 - c_p^{(3)}) \right) \theta_{p,i}^{(2)} g'(\gamma_i^{(2)}) c_j^{(1)} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
&= \sum_{p=1}^{\#(3)} \left( \delta^{(4)} \theta_{1,p}^{(3)} c_p^{(3)} (1 - c_p^{(3)}) \right) \theta_{p,i}^{(2)} c_i^{(2)} (1 - c_i^{(2)}) c_j^{(1)} + \frac{\lambda \theta_{i,j \neq 0}^{(1)}}{m} \\
\frac{\partial}{\partial \theta_{i,0}^{(1)}} J^{\{R\}}(\theta) &= \sum_{p=1}^{\#(3)} \left( \delta^{(4)} \theta_{1,p}^{(3)} c_p^{(3)} (1 - c_p^{(3)}) \right) \theta_{p,i}^{(2)} c_i^{(2)} (1 - c_i^{(2)}) c_0^{(1)}
\end{aligned}$$

Nous sommes donc maintenant en mesure de calculer les gradients associés à tous les poids de notre réseau. Bien que les quelques demis-journées qu'il nous a fallu pour calculer ces gradients voudrait que l'on soit certain de ces calculs, une vérification "de temps en temps" (disons une trentaine au total), paraît rassurante (et aussi parce que c'est la convention dans ce genre d'approche), et ce même pour éviter d'éventuelles erreurs de code dans la pratique. Nous allons utiliser le principe de différentiations finies étudier ce semestre en Analyse Numérique pour cela.

#### iv Vérification par différentiation finie de la Rétro-propagation

Cette méthode consiste à approcher le gradient de  $J$  en  $\theta$  par  $\frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$  (avec pour convention  $\epsilon = 10^{-4}$ ). Ainsi on aura le système, pour  $\theta \in \mathbb{R}^n$  :

$$\left\{
\begin{array}{lcl}
\frac{\partial}{\partial \theta_1} J(\theta) & = & \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon} \\
\frac{\partial}{\partial \theta_2} J(\theta) & = & \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \dots, \theta_n)}{2\epsilon} \\
& \vdots & \\
& \vdots & \\
\frac{\partial}{\partial \theta_n} J(\theta) & = & \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n-\epsilon)}{2\epsilon}
\end{array}
\right.$$

Preuve : En appliquant Taylor à l'ordre 2 on obtient que, avec  $\psi_1 \in [a; a+h]$  et  $\psi_2 \in [a-h; a]$  :

$$\begin{aligned} \frac{f(x+h) - f(x-h)}{2h} &= \frac{f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f^{(3)}(\psi_1)\frac{h^3}{6} - \left(f(x) - f'(x)h + f''(x)\frac{h^2}{2} - f^{(3)}(\psi_2)\frac{h^3}{6}\right)}{2h} \\ &= f'(x) + \frac{\frac{h^3}{6}(f^{(3)}(\psi_1) - f^{(3)}(\psi_2))}{2} \\ &= f'(x) + o(h^2) \end{aligned}$$

Et en appliquant ce résultat coordonnée par coordonnée on obtient le résultat souhaité.

Cette technique, bien qu'intéressante pour calculer le gradient, sera utilisée seulement pour vérifier que certains poids sont bien ceux supposés être trouvés, car l'utiliser pour calculer tous les poids serait trop coûteux en temps calcul.

Maintenant qu'on dispose d'un moyen de valider nos calculs nous pouvons avancer dans notre démarche et logiquement se poser la question à laquelle nous avons déjà été confrontée : comment choisir le taux d'apprentissage  $\alpha$  ?

La question que nous nous posons plus haut dans le cadre de notre régression logistique qui était : comment ne pas rester coincé dans un optima local se pose encore plus maintenant. D'autant plus que nous n'avons plus aucun moyen d'initialiser les poids selon une connaissance à priori puisque le principe d'un réseau de neurones est que nous ne savons pas exactement le rôle explicite qu'a chaque neurone dans la prédiction finale.

Il existe en fait une méthode pour contourner ce problème après une initialisation des matrices de poids pertinente : Momentum + RMSprop ! Nous allons y venir mais définissons en premier lieu l'initialisation de ces matrices  $\theta^{(1)}$ ,  $\theta^{(2)}$  et  $\theta^{(3)}$ .

#### v Initialisation des poids

Puisqu'on ne sait pas le rôle explicite de chaque neurone dans la prédiction finale, il n'est plus possible d'initialiser les poids selon une connaissance à priori. Il faudra donc le faire de manière aléatoire de la façon suivante, toujours sur recommandation de nos professeurs Dauphinois :

$$\theta_{i,j}^{(k)} = \frac{\mathcal{U}[0; 1]}{\sqrt{\#(k+1) \times (\#(k)+1)}} \quad (8)$$

Pour  $k \in \{1, 2, 3\}$ ,  $i \in \{1, \dots, \#(k+1)\}$  et  $j \in \{1, \dots, \#(k)+1\}$ , où  $\mathcal{U}[0; 1]$  est une uniforme sur  $[0; 1]$ .

**NB : A des fins pratiques, en terme de notation, dans toute la suite  $\theta_k$  désignera un poids d'une couche quelconque de la  $k^{\text{ième}}$  itération de notre descente de gradient stochastique. Ces méthodes se généraliseront ensuite à l'ensemble des poids du réseau.**

#### vi Algorithme de Polyak's Momentum - Amélioration de Nesterov

Momentum a pour objectif de répondre aux 2 problèmes que nous pouvons rencontrer en pratique lors de l'application d'une descente de gradient stochastique aux réseaux de neurones : la vitesse de convergence et le problème des minimas locaux. Intuitivement, elle s'inspire de deux principes physique : l'accélération et la vitesse ! L'idée est de penser le processus d'optimisation comme l'évolution d'une balle, se baladant le long de la courbe de la fonction de coût, à la recherche du minimum global. Si il y a assez de moments la balle ne restera pas coincé dans des optimas locaux mais atteindra l'optimum global ! Le principe est de faire bouger cette balle non plus en prenant seulement en compte la pente actuelle de la fonction de coût (c'est à dire  $\nabla J(\theta)$ ), ce qui intuitivement peut se s'interpréter comme son accélération, mais également sa vitesse, c'est à dire prendre en compte ses précédentes accélérations (les précédents gradients). Dans la pratique cela revient à actualiser la matrice de poids non plus seulement avec le gradient actuel, mais aussi avec les précédents. On initialise alors  $\theta_0$  suivant ce que nous venons de dire au cours de la sous-section précédente, ainsi que notre vitesse initiale à 0  $v_0 = 0$  et nous aurons, à chaque itération, deux paramètres inconnus : le taux d'apprentissage  $\alpha_k$  et le momentum  $\beta \in [0, 1]$  (par défaut

on le fixera à 0.9). L'algorithme sera alors défini par :

{ Pour  $k=1 \dots$  jusqu'à convergence }

$$v_{k+1} = \theta_k - \alpha_k \nabla J(\theta_k)$$

$$\theta_{k+1} = v_{k+1} + \underbrace{\beta_{k+1} (v_{k+1} - v_k)}_{(*)}$$

$$k \leftarrow k + 1$$

Puis retourner le dernier  $\theta_k$ . Cette méthode permettra d'améliorer la convergence, ainsi que la vitesse de convergence, du réseau de neurones. C'est pourquoi on la retrouve très souvent pré-implementée dans les librairies permettant de mettre en place une descente de gradient stochastique.

Notons qu'il nous reste encore à déterminer  $\alpha_k$ . C'est ce sur quoi nous allons nous pencher à présent.

**Remarque :** (\*) Au lieu de  $(\theta_k - \theta_{k-1})$  dans Polyak's Momentum classique. Cela a pour but d'améliorer la convergence en terme de rapidité, et de qualité, dans les dernières itérations.

## vii Recherche linéaire

Une première intuition serait d'utiliser les méthodes définies dans la section "Regression Logistique" telle que "le gradient pas optimal", "le critère de Wolfe-Armijo", ou même "le gradient pas optimal + BFGS". Le problème est qu'au vu du nombre de gradients à calculer à chaque itération, et donc de taux d'apprentissage à calculer :  $(\#(2) \times \#(1) + 1) + (\#(3) \times \#(2) + 1) + (1 \times \#(3) + 1)$ , cela serait trop coûteux. Ainsi, on préférera en pratique fixer nous même le taux d'apprentissage une fois pour toute. Le problème est donc toujours le même : comment le déterminer ? Le problème est même plus profond : ne serait-il pas préférable d'avoir un taux d'apprentissage différent en fonctions du poids de telle ou telle couche que l'on cherche optimiser ? Et pour finir, ne serait-il pas également raisonnable de modifier ce taux d'apprentissage au cours des itérations de notre descente de gradient stochastique à l'image de ce qu'on a fait dans la partie "Régression Logistique" ? Ce sont autant de questions auxquelles nous proposerons une réponse au cours des sous-section ci-dessous.

## viii RMSprop

Commençons avec RMSprop qui a pour but d'apporter une réponse aux questions 2 et 3 ci-dessus. Nous verrons ensuite qu'une réponse naturelle à la première interrogation en découlera. L'idée ici est de diviser notre taux d'apprentissage, choisi en amont, par un terme d'actualisation qui s'adaptera à notre avancement et notre position dans la descente de gradient stochastique (itération à laquelle on est rendu, poids que l'on veut modifier, valeurs des derniers gradients et de l'actuel). Ce terme d'actualisation sera construit comme décroissant de la norme du gradient de notre descente afin qu'on ai bien, comme abordé précédemment, l'adaptation souhaitée aux différents poids des différentes couches. L'algorithme sera, suivant cela, donné par :

**Phase d'initialisation :**  $\theta_0$  sera toujours initialisé suivant (8), aucune raison de modifier cela,  $\epsilon = 10^{-8}$  par convention (seulement introduit pour une raison pratique : il permettra d'éviter une division par 0 synonyme d'explosion du taux d'apprentissage, ce qu'on veut éviter bien sûr) et on fixe un taux de décroissance  $\rho = 0.999$  par convention.

**Récurrence :**

$$\theta_{k+1} = \theta_k - \frac{\alpha}{\sqrt{C_k + \epsilon}} \nabla J(\theta)$$

$$C_k = \rho C_{k-1} + (1 - \rho)[\nabla J(\theta)]^2$$

Et pour répondre à notre dernière interrogation formulée précédemment, maintenant qu'on sait qu'il est juste nécessaire de fixer le taux d'apprentissage de la première itération, une fois pour toute, nous pouvons naturellement partir du principe que nous le déterminerons à l'aide du critère de Wolfe-Armijo, par exemple, ou alors en respectant la convention  $\alpha = 0.001$ .

## ix ADaptive Moment estimation - ADAM & ADAMax

Pour finir, il nous a paru pertinent de terminer avec cette méthode de part sa popularité au sein de la communauté Machine Learning/Deep Learning. ADAM est en effet très connue pour sa robustesse et son efficacité et n'est rien de plus que la combinaison des méthodes Momentum+RMSprop présentées précédemment ! Cette fusion donnera de ce fait l'algorithme suivant :

**Phase d'initialisation :**  $v_0 = c_0 = 0$  et comme précédemment  $\epsilon = 10^{-8}$ ,  $\beta_1 = 0.9$  et  $\beta_2 = 0.999$  par convention.

**Récurrence :**

$$v_k = \beta_1 v_{k-1} + (1 - \beta_1) \nabla J(\theta_{k-1})$$

On reconnaît Momentum, puis RMSprop :

$$c_k = \beta_2 c_{k-1} + (1 - \beta_2) [\nabla J(\theta_{k-1})]^2$$

On introduit ensuite une correction de ces termes avant la fusion

$$\tilde{v}_k = \frac{v_k}{1 - \beta_1^k}$$

Et

$$\tilde{c}_k = \frac{c_k}{1 - \beta_2^k}$$

Et relativement au précédentes sections on obtient :

$$\theta_k = \theta_{k-1} - \frac{\alpha}{\sqrt{\tilde{c}_k + \epsilon}} \tilde{v}_k$$

qui termine la définition d'ADAM ; ou alors

$$\theta_k = \theta_{k-1} - \frac{\alpha}{u_{k-1}} \tilde{v}_k$$

qui nous donne ADAMax avec  $u_k = \max\{\beta_2 u_{k-1}, |\nabla J(\theta_k)|\}$ . Ces deux algorithmes peuvent encore être légèrement améliorer grâce au petit plus pratique suivant...

## x Décroissance programmée du taux d'apprentissage

Ayant conscience que la descente de gradient stochastique aura du mal à converger vers le minimum de la fonction de coût dans les dernières itérations, contrairement à une descente de gradient classique, mais oscillera autour de ce dernier, l'idée est de progressivement diminuer  $\alpha$  au cours des itérations afin de garantir une meilleure convergence. On introduit alors la notion d'epoch : le nombre d'epoch correspond au nombre de fois qu'on a parcouru entièrement notre échantillon d'apprentissage. Nous utiliserons alors une décroissance dite "exponentielle", au vu de la taille de notre jeu de données, qui à chaque nouvelle epoch modifiera  $\alpha$  de la façon suivante :

$$\alpha := 0.95^{\text{"Nombre d'epochs"}} \alpha$$

Ce qui conclut notre section sur le réseau de neurones à deux couches cachées. Nous allons maintenant pouvoir nous plonger dans la pratique avec Keras !

## 4 Approche Deep Learning

Les résultats de code de cette section sont accessible dans le notebook 'DeepLearning.ipynb'

### 4.1 Introduction

Nous nous intéressons dans cette partie au deep learning qui va nous permettre d'implémenter un réseaux de neurones complets afin d'obtenir une prédiction optimale.

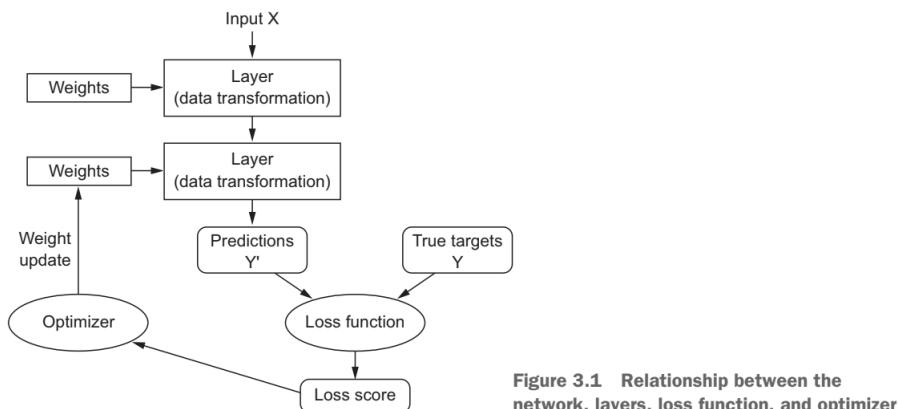
Le deep learning ou apprentissage profond est une des principales technologies utilisées en machine learning et aujourd'hui considérée comme la plus efficace dans les modèles pour lesquels on possède beaucoup de données. Elle définit l'utilisation de réseaux de neurones à plusieurs couches. La profondeur du réseau est déterminée par le nombre de couches c'est pourquoi on utilisé le terme 'deep' dans deep learning. Ces réseaux nécessitent une grande quantité de données en raison de l'important nombre de paramètres à fixer.

Le deep learning à pris de l'importance dans le monde de l'intelligence artificielle grâce au développement de la puissance de calcul des ordinateurs et l'augmentation de l'accès aux données. Ils sont capables de construire des modèles compétents en comprenant seuls les caractéristiques des données.

Les réseaux utilisent le "joint feature learning" : au cours de la rétro-propagation, lorsqu'ils modifient un poids ou un autre paramètre du réseau, tous les paramètres en lien avec celui-ci changent.

L'algorithme apprend donc les données en construisant couche par couche le réseaux qui se compléxifie au fur et à mesure mais ajuste simultanément toutes les caractéristiques du réseau pour optimiser l'apprentissage.

Voici un schéma récapitulatif pris du livre " Deep learning with Python" de Francois Chollet pour nous rappeler le fonctionnement global d'un réseaux de neurones :



Les connaissances acquises par le réseau sont représentées par les poids et les biais qui caractérisent chaque neurone. A l'aide de la bibliothèque Keras de python, nous souhaitons construire un modèle pour résoudre notre problème de classification binaire :

Lequel des joueurs 0 ou 1 gagne le match ?

## 4.2 Création d'un réseau de neurones avec Keras

Keras est une bibliothèque de Python créée à partir de tensorflow. Elle permet d'implémenter de réseaux de neurones plus facilement pour les utilisateurs. On crée un modèle Sequential() qui nous permet de créer un réseau couche par couche.

Tous les éléments construits à partir de la librairie Keras font partie de la classe des couches 'Layers' ou d'un objet très proche de celles-ci. Il existe différentes sortes de couches pour créer des réseaux. Keras utilise le plus souvent des couches 'densely connected' ou juste 'dense', et c'est ce que nous choisissons ici.

On crée un modèle avec un nombre de couches et de neurones par couche défini. Nous devons également fixer d'autres hyperparamètres de sorte que le réseau soit le plus performant possible. Le réseau va initialiser tous ses paramètres( biais et poids de chaque neurone) avec des valeurs prises au hasard. Puis on entraîne le réseau à apprendre les données. A l'aide des données, il ajuste les paramètres dans le but de pouvoir ensuite être capable de prédire le résultat d'un match. Enfin nous devons vérifier sa performance.

### 4.2.1 Choix du nombre de couches du réseaux et de neurones dans chaque couche

Nous partons de notre data set construit comme expliqué dans la partie de traitement des données. Il contient suite au preprocessing des données 276 colonnes et 110 532 lignes.

Parmi les 276 colonnes se trouve la variable d'intérêt 'winner' qui indique 0 ou 1 en fonction du gagnant du match et les 275 variables contenant les informations connues avant les matchs.

#### i Pour la couche d'entrée

La première couche est formée de 275 neurones. Les valeurs "input" contenant toutes les informations sur le match que l'on possède dans le jeu de données.

#### ii Pour la couche de sortie

La dernière couche, sortie du réseau, indique la valeur "output" qui désigne quel joueur devrait gagné le match selon le réseau. Cette couche peut contenir 2 neurones qui correspondent aux deux output 'le joueur 1 gagne le match' et 'le joueur 0 gagne le match'. Grâce à l'utilisation de la fonction d'activation 'sigmoïde'(dont nous parlerons après) ces neurones indiquent des valeurs entre 0 et 1 qui correspondent à la probabilité que chaque joueur gagne le match. On peut tout à fait se contenter dans cette couche d'un unique neurones puisqu'on sait que la somme des valeurs des indiquées par les 2 neurones initialement créé dans cette couche sera toujours égale à 1.

#### iii Pour les couches cachées

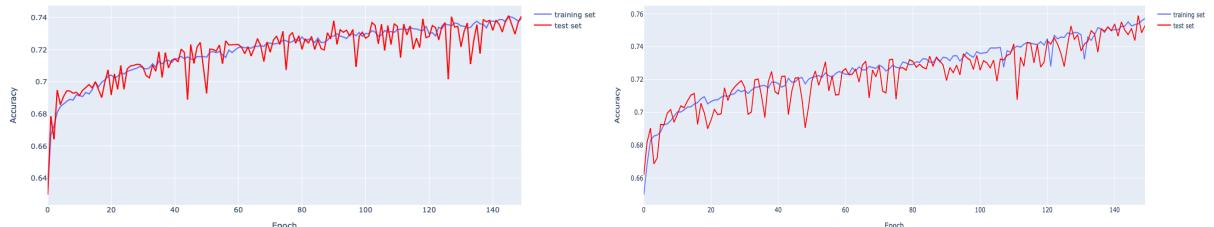
Pour déterminer le nombre de couches à créer, on doit expérimenter sur le jeu de données. On ne peut pas déterminer à l'avance ce qui va le mieux fonctionner. Ici, nous voulons au moins 3 couches cachées pour tenter de faire mieux, en terme de prédition, que notre précédent réseau de neurones.

Pour le nombre de couches on va créer plusieurs modèles : On teste, pour le même nombre d'epochs et de batchsize, 4 modèles contenant un nombre de couches cachées différent. On obtient les résultats ci-dessous :

En fin de compte, le modèle que nous avons retenu est celui avec 5 couches cachées ou moins. En effet, on peut voir sur les graphiques ci-dessus que lorsque le nombre de couche augmente on obtient une accuracy légèrement meilleure à la fin des 150 epochs mais la variabilité devient de plus en plus désastreuse. Il faut donc se contenter de 3 à 5 couches cachées, pas plus.

En ce qui concerne le nombre de neurones par couche, nous prenons entre 70 à 100 neurones par couche. Nous avons tester beaucoup d'alternatives.

A titre d'exemple, voici deux modèles parmi les nombreux que nous avons testé :



(a) 3 couches cachées

(b) 5 couches cachées



(c) 7 couches cachées

(d) 10 couches cachées

FIGURE 9 – Évolution de l'accuracy sur 150 epochs, avec un batch\_size fixé à 20 pour 4 réseaux de neurones ayant 100 neurones par couche cachée



(a) 70 neurones par couche cachée

(b) 600 neurones par couche cachée

FIGURE 10 – Évolution de l'accuracy sur 150 epochs, avec un batch\_size fixé à 20 pour 2 réseaux de neurones à 5 couches cachées

Tous ces choix peuvent évidemment être contestés car il existe plusieurs modèles qui conviendrait à notre problème.

Également comme il existe d'autres hyperparamètres (que nous verrons juste après) qui influencent la qualité de la prédiction. Le fait de devoir les choisir de manière plus ou moins arbitraire rend l'optimisation difficile.

#### 4.2.2 Choix d'autres hyperparamètres

Les hyperparamètres sont à optimiser "à la main". Ici nous verrons les trois principaux à fixer avec des tests sur notre jeu : Epoch et batch\_size.

Nous pouvons évidemment nous inspirer des choix qu'on fait d'autres utilisateurs pour des modèles similaires.

## i Epoch

Comme dit précédemment, cet hyperparamètre désigne le nombre de fois que le réseaux va parcourir l'intégralité des données pour mettre à jour les paramètres (poids et biais).

Pour choisir cet hyperparamètre, on peut faire apprendre notre réseaux de neurones avec un Epoch important et regarder l'évolution de l'accuracy ou d'une autre métrique.

On peut ensuite observer l'évolution de l'accuracy sur un graphique et on remarque qu'elle augmente avec le nombre d'Epoch jusqu'à atteindre un seuil.

Cela permet de voir à quel stade on commence à sur-apprendre les données.

Dans notre code nous avons fixé le nombre d'epochs à 150 pour voir une

Évidemment, plus le nombre d'Epoch et important plus l'apprentissage est long.

## ii Batch\_size

Cet hyperparamètre désigne le nombre de matchs utilisé par le réseau pour mettre à jour tous les paramètres au cours d'une itération.

Donc au cours d'une 'Epoch' les réseaux mettra à jour ses paramètres  $\frac{\text{nombre de match total}}{\text{batch size}}$  fois.

Plus le batch\_size est petit plus l'algorithme mettra de temps à réaliser chaque Epoch, ce paramètre à une grande influence sur la vitesse de calcul de l'algorithme et la vitesse de convergence.

En revanche s'il est trop grand le réseau ne sera pas capable d'ajuster les poids et les biais pour obtenir une performance satisfaisante.

Après de nombreux essaies avec la même méthode que pour choisir le nombre de couches et de neurones par couche, nous avons fixer le batch size à 30. Ce qui permet à l'algorithme de ne pas être trop lent mais d'atteindre une précision suffisante.

### 4.2.3 Choix des fonctions d'activation pour chaque couche et de la fonction de perte

#### i Fonction d'activation

Comme nous avons déjà décrit dans les parties précédentes du mémoire la fonction de perte nous nous concentrerons ici sur la fonction d'activation. Cette dernière doit être choisie pour chaque couche du réseau et correspond à l'opération faite sur les valeurs émises par une couche avant qu'elles ne soient reçues par la couche suivante.

Ces fonctions sont nécessaires car sans elles la  $(n + 1)^{i\text{ème}}$  couche du réseaux ne pourrait apprendre que des opérations linéaire entre les input qu'elles reçoit de la  $n^{i\text{ème}}$  couche et les output qu'elle émet. On aurait des calculs de la forme :

$$\text{output}^n = \text{dot}(\text{weights}^n, \text{output}^{n-1}) + \text{bias}^n$$

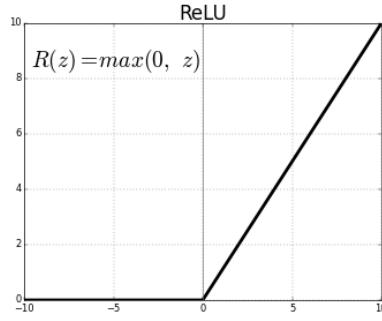
A la place, on applique une fonction d'activation  $\sigma$  entre deux couche et donc la couche effectue l'opération :

$$\text{output}^n = \sigma(\text{dot}(\text{weights}^n, \text{output}^{n-1}) + \text{bias}^n)$$

où sigma n'est plus linéaire et permet donc plus de souplesse au modèle.

#### ii Choix dans les couches cachées

Pour les fonctions d'activation des couches cachées du réseau, la fonction la plus utilisée est la fonction d'activation d'unité de rectification linéaire ('Rectified Linear Unit'), appelée ReLU.

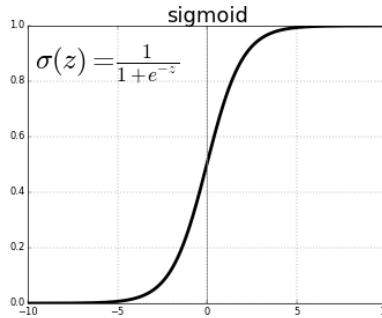


La fonction Relu permet au modèle d'apprendre plus rapidement. Elle ressemble à une fonction linéaire sur  $\mathbb{R}_+$ , une partie de son domaine de définition mais en réalité, elle ne l'est pas. C'est ce qui permet au réseau d'apprendre des relations bien plus complexe que les relations linéaires.

### iii Choix pour la dernière couche

Pour des problèmes classiques on connaît les fonctions à utiliser comme fonction de perte ('loss function') et comme dernière fonction d'activation ('last layer activation function'). Dans un problème de classification binaire comme c'est notre cas, on choisit la fonction sigmoïde comme dernière fonction d'activation :

$$\begin{aligned} \text{sigmoid} &: \mathbb{R} \rightarrow [0, 1] \\ x &\mapsto f(x) = \frac{1}{1 + \exp^{-x}} \end{aligned}$$



Ainsi les valeurs prises par le ou les neurones de la couche d'output seront comprises entre 0 et 1 et représenteront bien une probabilité de gagner pour chaque joueur.

Remarque : dans certains problèmes de classification tels que la classification multiclass single-label, c'est la fonction exponentielle normalisée, appelée softmax qui est utilisée.

En ce qui concerne la fonction de perte, la plus adapté est la fonction : 'binary\_crossentropy'.

$$\text{fonction de perte} = -\frac{1}{\text{taille de l'output}} \sum_{i=1}^{\text{taille de l'output}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

où  $\hat{y}_i$  correspond à la  $i$ -ème valeur dans l'output (ici  $i = 1, 2$  si 2 neurones),  $y_i$  correspond à la vrai valeur que devrait prendre le même neurone de l'output, enfin la taille de l'output est 1 ou 2 en fonction du nombre de neurones dans la dernière couche.

On peut utiliser cette fonction de perte dans beaucoup de problèmes de classification tant que les valeurs prises par les output sont binaires.

#### 4.2.4 Comment vérifier la performance de notre réseau de neurones ?

##### i Division du jeu de données entre train et test

On utilise la fonction 'train\_test\_split()' de scikit-learn qui nous permet de créer 4 dataframes : x\_train, x\_test, y\_train et y\_test

On peut ainsi entraîner le réseau grâce à la fonction 'model.fit(x\_train,y\_train, epochs, batch\_size)' de Keras sur les données 'train' puis vérifier son efficacité sur les données 'test' en utilisant la fonction 'model.evaluate(x\_test,y\_test, batch\_size)'.

Cette seconde étape permet de tester le modèle sur des données qui lui sont inconnues.

On peut aussi utiliser la fonction 'model.predict(x\_test[k])' qui nous renvoie le résultat du match k et de la fonction 'compare()' qui nous permet de comparer cette prédiction aux vraies valeurs.

Comme nous avons vu sur les schémas dans la partie 4.2.2 , l'accuracy de l'échantillon de test varie beaucoup plus que celle du train et est un bon indicateur pour choisir les hyperparamètres.

##### ii Validation\_split

C'est indicateur entre 0 et 1 de la fraction des données train à mettre de côté pour réaliser une validation. L'algorithme n'entraînera pas les données sur cette portion et pourra ensuite vérifier l'efficacité de l'algorithme sur ces données. Il évaluera la fonction de perte sur ces données ou tout autre métrique demandée.

Cela permet de réaliser une deuxième évaluation du modèle après avoir regarder l'efficacité sur nos données test.

##### iii Métriques

On utilise les Métriques pour déterminer l'efficacité de notre réseau. On utilise le tableau suivant pour définir les 'métriques' :

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Dans notre modèle on considère que les positifs correspondent aux cas 'joueur 0 gagne' et les négatifs à 'joueur 1 gagne'.

Les plus classiques sont :

1. Accuracy =  $\frac{TP+TN}{TP+TN+FP+FN}$  C'est la plus utilisée. Il s'agit du ratio des bonnes prédictions sur le nombre de prédictions totales.
2. Precision =  $\frac{TP}{TP+FP}$ . Correspond au ratio du nombre de fois que le réseau a bien prédit que le joueur 0 gagne sur le nombre total de fois où il a prédit que le joueur 0 gagnerait.
3. Recall :  $\frac{TP}{TP+FN}$ . Correspond au ratio du nombre de fois que le réseau a bien prédit que le joueur 0 gagne sur le nombre total de fois où le joueur 0 a vraiment gagné.

Il existe évidemment plein d'autres métriques qui permettent de vérifier cela.

Pour améliorer notre modèle, nous avons regardé l'accuracy. Comme lors de la construction du jeu de données nous avons rentrer le joueur 1 comme étant celui avec le meilleur rang des deux, il gagne plus souvent que le joueur 0 et nous devons donc vérifier que le réseau ne se contente pas de prédire que le joueur 1 gagne systématiquement.

On peut utiliser un modèle dit naïf qui prédit toujours que le gagnant est le joueur le mieux classé et le comparer à notre modèle.

#### 4.2.5 Conclusion

Tout ceci dans le but d'obtenir le modèle le plus performant. En raison de la quantité de données limitée, nous ne pouvons pas atteindre un modèle capable d'être efficace dans la vie de tous les jours. D'une

part parce qu'il ne connaît qu'un nombre limité de joueurs et d'autres part un nombre de paramètres manquants important.

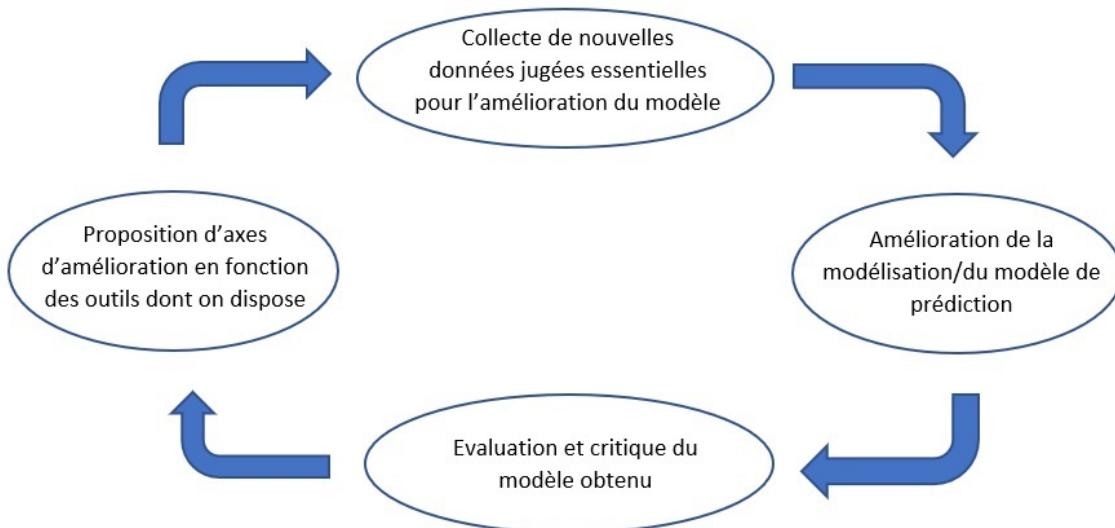
## 5 Conclusion

### 5.1 Conclusion à mi-parcours

La première leçon à tirer de ce projet est qu'il est contre-productif de passer trop de temps à collecter des données à corps perdu (ce que nous avons fait les premiers mois) avant de commencer à réfléchir à une quelconque modélisation. En effet, il semble préférable de traiter une étude de ce type de la façon suivante :

- Commencer par construire un modèle simple, avec quelques données, qu'il est possible d'implémenter rapidement pour voir ce qui se passe.
- Faire une première évaluation de ce modèle et identifier les problèmes auxquels nous pourrions être confrontés.
- Réfléchir aux outils théoriques et pratique que nous pourrions utiliser ainsi qu'à des aspects plus précis comme le nombre et comment sélectionner nos variables explicatives, qui est aussi important que la quantité de données à disposition
- Mise en oeuvre de ces conclusions et recherche ciblée de nouvelles données.
- Puis on "recommence ce schéma"...

En clair, l'idée est d'adopter une méthode de travail plus "cyclique" du type :



### 5.2 Conclusion finale

Nous n'avons pas eu le temps de nous pencher sur des techniques dites de "data mining" afin d'ajouter de nouvelles variables explicatives à notre modèle puisque nous avons jugé que nous possédons déjà un jeu de données suffisamment important pour implémenter les différentes méthodes que vous nous avons présenté. Nous avons donc fait le choix, puisque le mémoire de M1 a une visée plus pédagogique que de performance pure de notre prédiction en terme de résultat, de nous concentrer plus sur la présentation et la démonstration de différentes méthodes que sur une phase de collecte de données. Mais le "data mining" reste néanmoins une technique à retenir pour nos futures projets afin d'apporter des données pertinentes à la modélisation.

Dans un cadre plus théorique nous pouvons aussi faire une ouverture sur les algorithme à région de confiance et recuit simulé pour la régression logistique, qui sont d'autres techniques d'optimisation qui peuvent également être utilisées dans le cadre de notre problématique.

Pour finir, nous avons beaucoup appris de ce projet que nous avons pris comme un défis. Il nous a permis de développer notre intérêt pour le machine learning. Il a également renforcé notre gout pour le travail en équipe et notre capacité d'organisation.

## Références

- [1] Jean-Yves Audier. *Apprentissage statistique* . 2010
- [2] Sylvain Arlot. *Fondamentaux de l'apprentissage statistique* . 2010
- [3] Gerard Biau. *Statistique et apprentissage*. 2014
- [4] Trevor Hastie, Robert Tibshirani et Jerome Friedman. *The Elements of Statistical Learning*.2014
- [5] Shai Shalev-Shwartz, Shai Ben-David. *Understanding machine learning : From Theory to Algorithms*. 2014
- [6] Abdiansah, Retantyo Wardoyo. *Time complexity Analysis of Support Vector Machine (SVM) in LibSVM*. 2015
- [7] Pierre Gaillard. *Lecture Note on k-Nearest Neighbors*. 2018
- [8] Deep learning with Python Francois Chollet
- [9] Arbres CART et Forêts aléatoires Robin Genuer, Jean-Michel Poggi  
Cours de Machine Learning par l'Université de Stanford Coursera.  
<https://hal.archives-ouvertes.fr/hal-01387654v2/document>  
[https://perso.telecom-paristech.fr/angelini/master\\_spsiv/optimization/iup1.opti.chp6.pdf](https://perso.telecom-paristech.fr/angelini/master_spsiv/optimization/iup1.opti.chp6.pdf)  
<https://math.unice.fr/~dreyfuss/D4.pdf>  
<https://arxiv.org/pdf/1412.6980.pdf>  
<https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a>  
<http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.pdf>  
<https://ieeexplore.ieee.org/abstract/document/9042352>  
<https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-n>  
<https://towardsdatascience.com/a-step-by-step-implementation-of-gradient-descent-and-backpropagation>  
<http://papers.nips.cc/paper/681-a-parallel-gradient-descent-method-for-learning-in-analog-vlsi-neural>