Active Experimental Design For Tsunami Modeling

par

Chloé Pasin, Alexandre Robicquet, Maxime Stauffert

encadrés par

Nicolas Vayatis, David Buffoni, Themistoklis Stefanakis

Mémoire

Centre de Mathématiques et Leurs Applications

Juillet 2012

Résumé

On conçoit aisément que la localisation d'une île conique devant une plage puisse offrir une certaine protection contre les vents ou vagues de l'océan. Cependant, d'après l'étude de différentes données post-tsunamis, cette intuition s'avère fausse pour des tsunamis. Ceci nous motive donc à étudier l'amplification du run-up, c'est-à-dire le rapport entre la hauteur de la vague lorsqu'elle atteint la plage derrière l'île conique et la hauteur de la vague lorsqu'elle atteint un point latéral, indépendant de l'île. Le logiciel Volna nous permet d'effectuer des simulations de tsunamis à partir de configurations spatiales fixées au préalable. A l'aide de ces simulations, on souhaiterait déterminer la fonction, notée f, qui détermine le run-up (hauteur de la vague lorsqu'elle atteint la plage) en fonction de l'environnement (les paramètres physiques) dans lequel évolue le tsunami. Le temps computationnel de Volna étant élevé, on souhaiterait utiliser les points déjà existants pour estimer la fonction f. On peut donc traiter ce problème grâce à l'apprentissage, qui en théorie apporte des résultats optimaux lorsque le nombre d'expériences est élevé. Cependant, on est limité par le nombre de données, et on souhaite donc apprendre plus efficacement, en sélectionnant à l'avance les exemples qui seront étiquetés. L'apprentissage actif intervient donc ici, et après l'étude théorique des algorithmes déjà existants, nous avons testé leur efficacité. Plusieurs problèmes se sont alors présentés, puisque l'on n'a que très peu de points, et que contrairement à ce qui a déjà été étudié dans la littérature, on a affaire à une régression et non une classification, et l'on ne souhaite pas ajuster la fonction apprise à la fonction théorique, mais seulement déterminer son maximum. Après avoir adapté des programmes déjà existants à notre problème, nous nous sommes attaqué au problème plus épineux qu'est la conception d'algorithmes de régression en apprentissage actif. Algorithmes dont l'efficacité à ainsi pu être validé à l'aide d'expériences sur plusieurs bases de données réelles et artificielles.

Table des matières

1 Présentation du problème

1.1 Phénomènes physiques étudiés : tsunamis

Les tsunamis sont de longues vagues qui peuvent provoquer des dégâts considérables en atteignant les littoraux.
Ils peuvent être engendrés par divers phénomènes physiques [**?**] :

- les tremblements de terre : ils provoquent un déplacement vertical du sol marin qui peut s'étendre sur plusieurs kilomètres le long de la rupture ; l'eau située au dessus de ce sol marin est également déplacée de haut en bas, créant ainsi une vague plus ou moins longue.
- les glissements de terres côtières, créant de grandes vagues lors de la chute de ces terres dans l'eau : ils peuvent être dûs à de fortes pluies faisant s'écrouler la terre dans l'océan, ou à des tremblements de terre provoquant des détachements de terrain.
- les glissements de terrain sous-marins : la terre située sur les zones hautes du relief sous-marin peut s'écrouler à la base de ce relief, créant ainsi une vague
- les volcans (près de la côte, ou peu profonds) : les éruptions volcaniques peuvent non seulement générer des glissements de terrain sur les flancs du volcan, mais également des ondes de chocs créant un petit tsunami.
- les météorites : elles peuvent aussi créer des tsunamis lors de l'impact dans l'eau, mais restent assez rares.

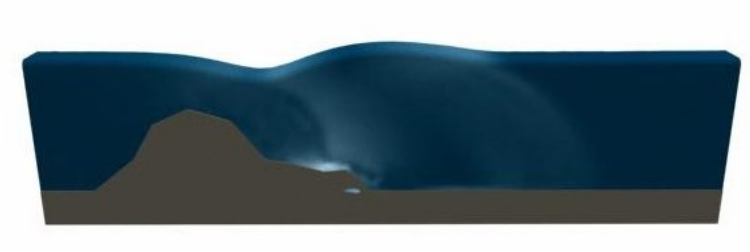


FIGURE 1 – Formation d'une vague due à un glissement de terrain sous-marin

Les tsunamis les plus répandus sont en général produits par des tremblements de terre sous-marins. Le schéma suivant (Figure **??**) explicite la formation de la vague en trois étapes, lors d'un tremblement de terre provoqué par une subduction entre 2 plaques tectoniques convergentes.

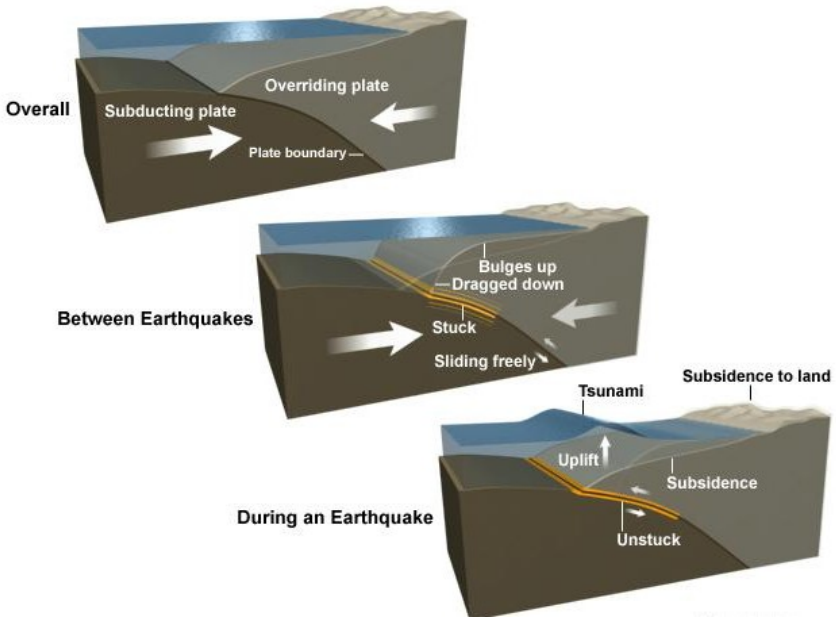


FIGURE 2 – Étapes de formation d'un tsunami

Les tsunamis peuvent être caractérisés grâce au run-up, la hauteur maximale de la vague (par rapport au niveau de l'eau) lorsqu'elle atteint la plage. Plusieurs tsunamis se sont succédés durant les dernières années [**?**] ; pendant certains de ces tsunamis, des habitants se pensaient protégés par des îles se situant devant la plage, mais ont au contraire observés un run-up plus élevé à cet endroit. Certaines études ont alors confirmé que lors de la présence de l'île devant la plage, le run-up derrière l'île était plus élevé que ce que l'on attendait [**?**] [**?**] [**?**] [**?**]. Nous souhaitons donc étudier la configuration suivante (Figure **??**), pour déterminer l'impact apporté par l'île. Pour cela nous allons étudier l'amplification du run-up, le rapport entre le run-up en un point sur la plage derrière l'île, et celui en un point latéral indépendant de l'île.

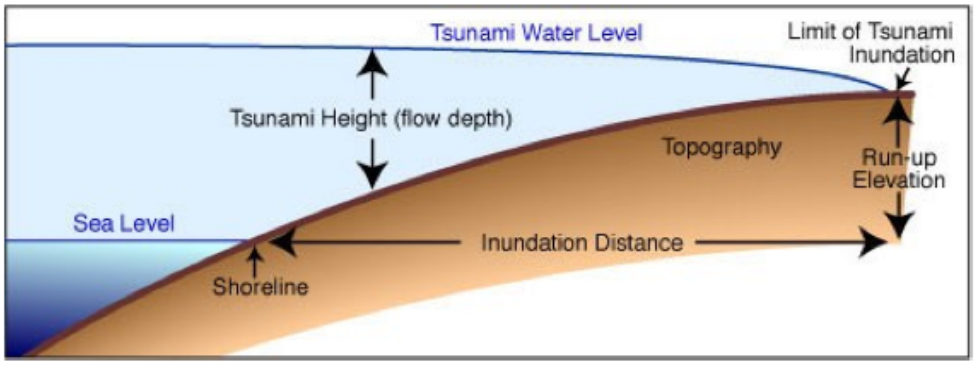


FIGURE 3 – Illustration du run-up

La vague du tsunami est modélisée par une "single wave" [**?**] [**?**], c'est-à-dire un seul déplacement de l'eau au dessus du niveau de la mer. La hauteur de la vague dépend de l'espace et du temps et peut être formellement modélisée par :

$$\eta(x_0, t) = A_0 sech^2(wt - 2.6) \text{ avec } A_0 = 1.5m$$

où w est la fréquence de la vague.

On s'intéresse à l'influence de 5 paramètres physiques sur l'amplification du run-up :

Notation	**Paramètre**	**Domaine**
is	pente de l'île	0.05 - 0.2
bs	pente de la plage	0.05 - 0.2
d	distance entre l'île et la plage	0 - 5000 m
h	profondeur de l'eau	100 - 1000 m
w	fréquence de la vague	0.01 - 0.1 rad/s

Le schéma suivant représente la situation dans laquelle nous nous plaçons pour l'étude de l'amplification du run-up.

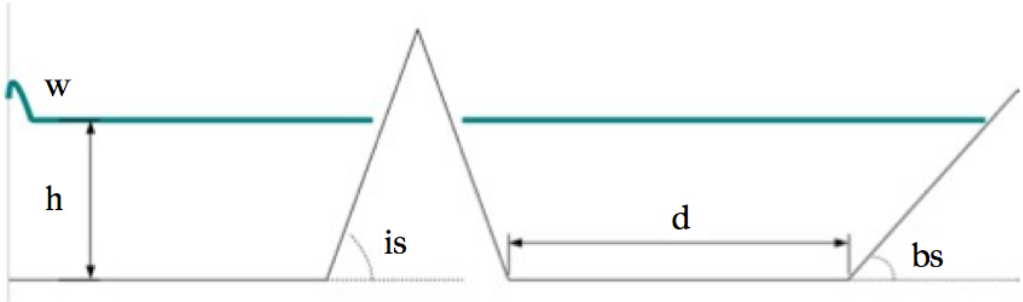


FIGURE 4 – Schématisation des paramètres d'étude

Afin d'étudier l'influence de ces 5 paramètres sur l'amplification du run-up, il est nécessaire d'effectuer, à l'aide du logiciel VOLNA [**?**],différentes simulations de tsunami. On obtient après

un certain nombre de simulations une base de données définissant les valeurs d'amplification du run-up en fonction des 5 paramètres pré-définis. Nous allons présenter plus en détails le logiciel VOLNA.

1.2 Simulation de tsunami à l'aide de VOLNA

VOLNA est un code numérique permettant de modéliser entièrement les tsunamis (origine, propagation et run-up sur la côte). Il consiste en la résolution des Equations Non Linéaires en Eau Peu Profonde (on considère que la profondeur de l'eau et la hauteur de la vague sont beaucoup plus faibles que la longueur de la vague).

On utilise les notations suivantes : a_0 l'amplitude de la vague, h_0 la profondeur moyenne de l'eau, l_0 la longueur caractéristique de la vague. H est la hauteur totale de l'eau (donc $H = h + \eta$) et $\vec{u} = (u, v)(\vec{x}, t)$ est la vitesse horizontale (en moyenne sur la profondeur).

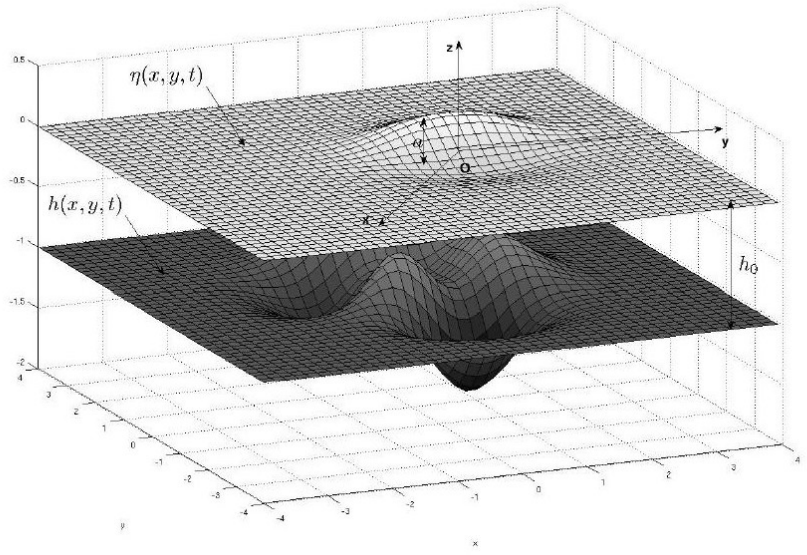


FIGURE 5 – Schéma du domaine où se développe le tsunami : en haut, la surface de l'eau et en bas, le sol sous-marin

Les équations sont les suivantes :

$$\begin{aligned} \frac{\partial H}{\partial t} + \nabla(H\vec{u}) &= 0 \\ \frac{\partial (H\vec{u})}{\partial t} + \nabla(H\vec{u} \otimes \vec{u} + \frac{g}{2}H^2) &= gH\nabla h \end{aligned}$$

Pour résoudre ces équations de façon discrète, la méthode de résolution utilisée est celle des Volumes Finis et celle de discrétisation du temps est la méthode de Runge-Kutta. La résolution est effectuée sur un maillage triangulaire non uniforme [**??**], de taille 500m et 2m dans les zones qui nous intéressent, c'est-à-dire la zone de la plage située derrière l'île et celle où se situe le point latéral de référence, indépendant de l'île (la plage est située à droite sur le maillage suivant). Ce maillage triangulaire non uniforme nous permet d'obtenir une meilleure précision là où c'est nécessaire, et d'éviter une précision (et donc des calculs effectués par la logiciel) inutile dans les zones non étudiées.

Afin de déterminer le run-up assez précisément, on positionne des wave-gages (capteurs) linéairement espacés sur la plage entre 0 et 5.5m de hauteur. Ceux-ci vont détecter la hauteur de l'eau (la valeur de η) en fonction du temps, à l'endroit où ils sont situés. Voici un exemple de ce que peut renvoyer un des wave-gages (Figure **??**).

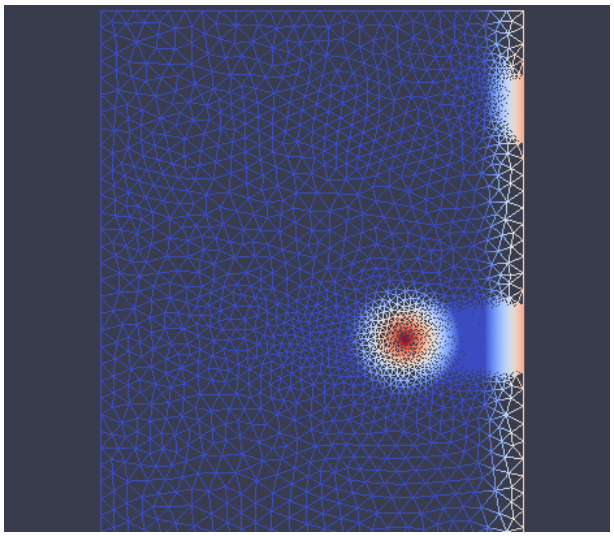


FIGURE 6 – Schéma du maillage

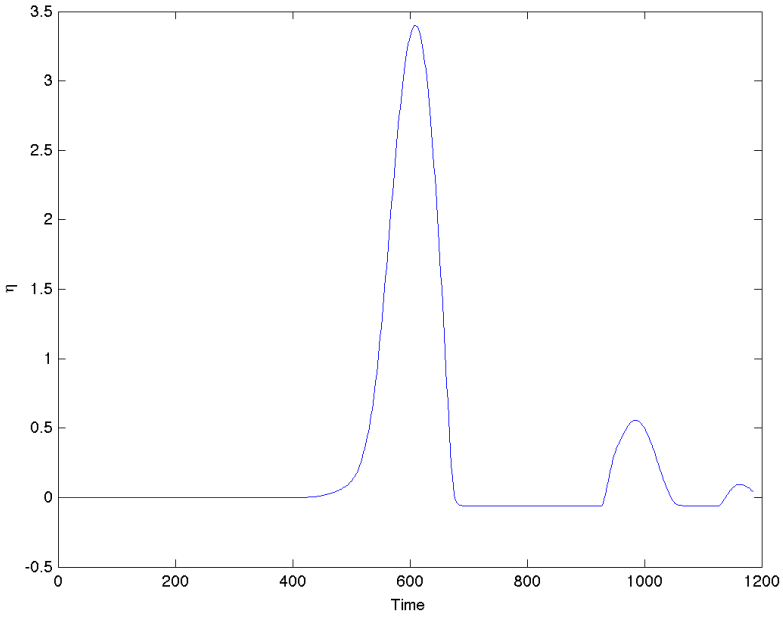


FIGURE 7 – Évolution d'η en un wave-gage au cours du temps

L'inconvénient de VOLNA est qu'une simulation nécessite beaucoup de temps, de l'ordre de 1h, 1h30. On souhaite donc obtenir le maximum d'information avec un nombre fixé de points calculés. On a donc utilisé la méthode LHS (Latin Hypercube Sampling) pour déterminer les 5 paramètres physiques (coordonnées d'un point X) de chaque situation qui sera simulée. Cette méthode permet de positionner dans l'espace un nombre fixé de points, de façon à maximiser un certain critère, ici leur distance réciproque. Ceci permet une bonne occupation de l'espace, dite "space-filling". De plus, Fricker et Urban ont montré dans leur article [**?**] que la méthode Latin Hypercube Sampling est plus efficace pour obtenir des points éparpillés dans l'espace que choisir des points aléatoires.

Suivant l'ensemble de points généré par la méthode LHS, nous avons lancé VOLNA sur cet ensemble de simulations. Les simulations effectuées vont nous permettre d'étudier le run-up et les paramètres physiques qui le déterminent.

1.3 Nature des données

Avant de s'intéresser précisément aux simulations $(x_i, y_i, i = 1..200)$ données par VOLNA, on essaie d'avoir des informations globales sur le problème. On a donc réalisé un *scatterplot*, qui représente chaque x_i en fonction de x_j, $i \neq j, 1 \leq i, j \leq 5$, et colore les points obtenus selon la valeur de l'amplification du run-up correspondante (bleu pour un run-up faible et rouge pour un run-up élevé). On a seulement sélectionné ici 2 graphiques parmi les 10 obtenus pour avoir une meilleure visibilité.

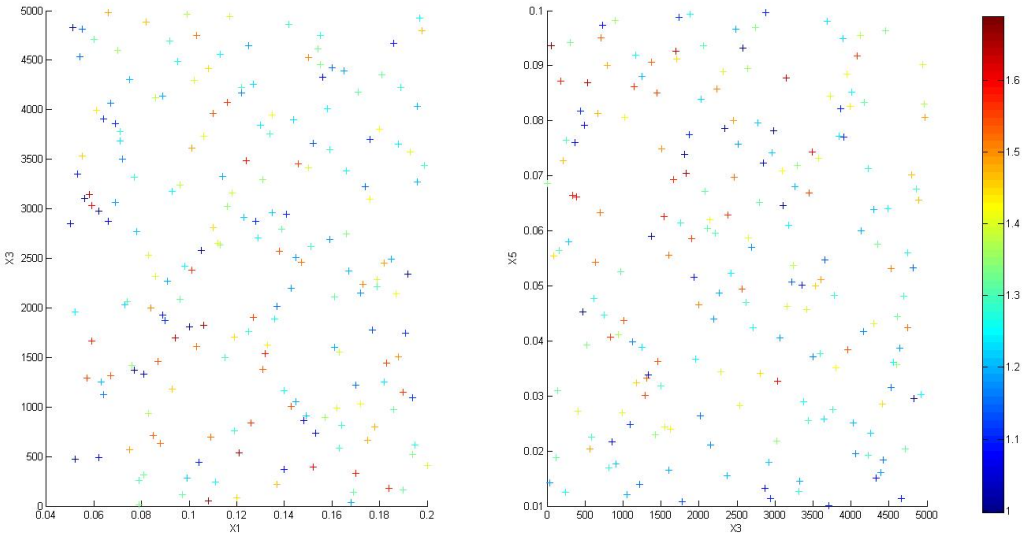


FIGURE 8 – Scatterplots : x_3 en fonction de x_1 et x_5 en fonction de x_3. La couleur des points correspond à la valeur de l'amplification du run-up associée, donnée par le dégradé de couleur

Les deux scatterplots de la figure 8 sont semblables à tous les autres obtenus. On remarque qu'il ne semble pas y avoir de dépendance linéaire entre l'amplification du run-up et les 5 paramètres physiques étudiés. Si l'on veut déterminer une fonction qui détermine le run-up selon ces 5 paramètres, cette fonction sera donc vraisemblablement non linéaire, et donc difficile à déterminer.

Après cette première étude, nous pouvons soulever plusieurs problèmes à aborder :

- Tout d'abord, nous souhaitons déterminer quelles sont les conditions qui engendrent la pire situation possible lors d'un tsunami, afin de pouvoir protéger les habitants dans ce cas. Il faudrait donc déterminer le maximum de l'amplification du run-up, pour en trouver les paramètres physiques qui le causent.

- De plus, il faudrait idéalement donner un modèle empirique de l'amplification du run-up en fonction des 5 paramètres physiques, via une fonction $f(is, bs, d, h, w) = f(x) =$run-up amplification. Cela permettrait de prévoir l'amplification du run-up dans n'importe quelle situation.

- Formellement, on souhaite donc déterminer $\text{argmax}_x\,(f(x))$ dans un domaine discret de valeurs, et pour lequel on connait f partiellement grâce aux données initiales.

- Nous allons donc avoir recours à l'apprentissage statistique, pour essayer de déterminer cette fonction f telle que, pour X les données et Y les résultats, $f(X)$ soit le plus proche possible de Y.

- Il devient également nécessaire de trouver différentes méthodes afin de diminuer le temps computationnel au cours d'une expérience, soit en d'autres termes, choisir intelligemment

les points que l'on demande à VOLNA. Nous allons donc avoir recours à l'apprentissage actif.

Nous allons donc dans un premier temps nous pencher sur l'aspect théorique de l'apprentissage statistique afin de mieux cerner les outils mathématiques que nous utiliserons par la suite. Nous arriverons dans un second temps à l'étude de l'apprentissage actif en classification et en régression pour ensuite pouvoir aussi bien adapter notre problème à des algorithmes déjà étudié que créer notre propre algorithme de régression actif. Nous exposerons enfin nos résultats expérimentaux, sur plusieurs bases de données réelles et artificielles, afin de souligner l'efficacité et la pertinence de certaine méthode à l'instar de certaines autres.

2 Apprentissage statistique

L'apprentissage statistique est un ensemble de méthodes statistiques très utilisé pour analyser la relation d'une variable par rapport à une ou plusieurs autres. Le développement de cette approche commença dans les années 1960, suite à l'apparition des premiers ordinateurs capables de calculer des problèmes multidimensionnels de la vie de tout les jours [**?**].
Le principe de l'apprentissage est de déterminer la relation entre X, où X représente les paramètres en entrée, qu'ils soient physiques (équations, conditions aux limites) ou numériques (géométrie du maillage), et Y, où Y correspond au résultat obtenu (ici le run-up, ou l'amplification du run-up). On cherche donc une fonction $f : \mathbb{R}^d \to \mathbb{R}^d$ telle que $Y \approx f(X)$, alors qu'on observe partiellement f à partir d'un échantillon $\{(x_i, y_i), i = 1..n\}$. Dans notre cas, l'échantillon est de taille 200 et chaque $x_i = (x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}, x_{i,5})$.

Dans le cas de l'approche statistique, on suppose que $(X, Y) \sim P$, avec P loi de probabilité inconnue [**?**]. On pose le modèle statistique $Y = f(X) + \epsilon$, où ϵ est le "bruit", la perturbation autour de la fonction f, qui est une variable aléatoire sur $\mathbb{R}$, d'espérance nulle et indépendante de X.

Pour Y, deux cas sont possibles :
- le cas de la régression : Y est une variable aléatoire sur $\mathbb{R}$
- le cas de la classification (cas spécifique de régression) : Y est une variable aléatoire contrainte sur le domaine $\{-1, 1\}$ (ou $\{0, 1\}$).

2.1 Apprentissage pour la régression

Y est une variable aléatoire réelle avec $Y = g(X) + \epsilon$ et on suppose $\epsilon \sim N(0, \sigma^2)$. On cherche à estimer g à partir d'un échantillon $\{(x_i, y_i), i = 1..n\}$ tel que les $\{(x_i, y_i)\}$ suivent la même loi de probabilité que (X,Y). On peut estimer g de deux manières possibles :

- Soit avec l'estimateur du maximum de vraisemblance (EMV) qui maximise la fonction de vraisemblance. Pour un échantillon $\{(\theta_i, z_i, i = 1..n)\}$, avec $z_i = (x_i, y_i)$ et en notant g la fonction recherchée :

$$\mathcal{L}(g, z_1, .., z_n|\theta_1..\theta_n) = \prod_{i=1}^{n} f(z_i, \theta)$$

  où $f(z, \theta)$ est la fonction de densité $f_\theta(z)$.

- Soit avec l'estimateur des moindres carrés (EMC). Pour un échantillon $\{(x_i, y_i, i = 1..n)\}$, en supposant que $\{(x_i, y_i)\}$ suivent la même loi de probabilité que (X, Y) et avec la fonction recherchée telle que Y = $f(X) + \epsilon$, l'estimateur des moindres carrés minimise $\sum_{i=1}^{n}(y_i - f(x_i))^2$

.

Dans ce cas, avec $\epsilon \sim N(0, \sigma^2)$, on peut calculer la fonction de vraisemblance :

$$\mathcal{L}((g, \sigma^2), y_1, ..y_n|x_1, .., x_n,) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y_i - g(x_i))^2}$$

donc

$$\ln(\mathcal{L}((g,\sigma^2),y_1,..y_n|x_1,..,x_n,) = -n\ln(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - g(x_i))^2$$

D'où,

$$\max_g \mathcal{L} = \min_g \sum_{i=1}^{n}(y_i - g(x_i))^2$$

donc ici, les estimateurs sont équivalents. On s'intéresse dans notre cas à l'estimateur des moindres carrés, adapté pour la régression. Avec cet estimateur, on a une solution explicite du problème, ce qui est plus simple que trouver le modèle (la loi de probabilité) qui génère les x.

2.2 Apprentissage pour la classification

La classification est un cas particulier de la régression, où $Y \in \{0,1\}$. Soit g : $\mathbb{R}^d \to \{0,1\}$ un classifieur.

Formellement, on peut définir l'erreur de classification de g,

$$L(g) = P(Y \neq g(X)) = \mathbb{E}[\mathbb{1}_{\{Y\neq g(X)\}}]$$

.

On cherche

$$\begin{aligned} g^* &= \underset{g\in\mathcal{G}}{\text{argmin}}\ \mathbb{E}[\mathbb{1}_{\{g(X)\neq Y\}}] \text{ donc } g^* \text{ minimise } L(g) \\ &= \underset{g\in\mathcal{G}}{\text{argmin}}\ P(g(X)\neq Y) \end{aligned}$$

En notant $\eta(x) = \mathbb{E}[Y|X=x]$, on peut déterminer théoriquement g^* :

$$\begin{aligned} L(g) &= \mathbb{E}[\mathbb{E}[\mathbb{1}_{\{Y\neq g(X)\}}|X]] \\ &= \mathbb{E}[\mathbb{E}[\mathbb{1}_{\{Y=1\}}\mathbb{1}_{\{g(X)=0\}} + \mathbb{1}_{\{Y=0\}}\mathbb{1}_{\{g(X)=1\}}|X]] \\ &= \mathbb{E}[\eta(X)\mathbb{1}_{\{g(X)=0\}} + (1-\eta(X))\mathbb{1}_{\{g(X)=1\}}] \end{aligned}$$

Donc pour tout X, on a

$$g^*(X) = \begin{cases} 1 & si\ \eta(X) \geq \frac{1}{2} \\ 0 & sinon \end{cases}$$

Le problème est que l'on a un nombre fini de données, sans connaître la loi de probabilité de (X,Y). De plus, l'erreur $L(g)$ dépend de la loi donc elle est inaccessible. On sait aussi, d'après la loi des grands nombres que :

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\{Y_i\neq g(X_i)\}} = \mathbb{E}[\mathbb{1}_{\{Y\neq g(X)\}}]$$

On propose donc un estimateur dépendant des données :

$$\widehat{g}_n = \underset{g\in\mathcal{G}}{\text{argmin}}\ \widehat{L}_n(g)$$

où $\widehat{L}_n(g) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\{Y_i\neq g(X_i)\}}$ est le risque empirique.

Cependant, $\mathcal{G}$ peut contenir une infinité de fonctions g, et il y a donc un risque de sur-apprentissage (ou "overfitting" en anglais) : la fonction qui modélise Y en fonction de X mémorise seulement les données en les apprenant par cœur et ne permet pas de généraliser le modèle. Par conséquent, pour des données inconnues, la prédiction risque d'être imprécise.

Pour éviter le sur-apprentissage, deux méthodes sont envisageables :

– Empirical Risk Minimization : on restreint la classe $\mathcal{G}$ (en un ensemble de fonctions plus régulières par exemple) puis on détermine $\widehat{L}_n$ sur cette classe.

– Structural Risk Minimization : les algorithmes qui utilisent cette solution consistent en l'introduction d'une pénalisation qui dépend de la taille de la classe de fonctions. On suppose qu'il existe une suite infinie $\{\mathcal{G}_d, d = 1, 2, ...\}$ de classes de fonction de taille croissante. On définit alors : $\widehat{g}_n = \text{argmin}_{g\in\mathcal{G}}\ \widehat{L}_n(g) + Pen(d, n)$ où $Pen(n, d)$ donne la préférence aux fonctions appartenant aux plus petites classes. On peut donc définir :

$$\widehat{g}_n = \underset{g\in\mathcal{G}_d}{\text{argmin}}\ \widehat{L}_n(g) + \lambda \|g\|^2 \tag{1}$$

On peut ensuite estimer $L(\widehat{g}_n) - L(g^*)$, pour déterminer sous quelles conditions sur P et $\mathcal{G}$ on a

$$L(\widehat{g}_n) \xrightarrow[n\to+\infty]{p.s} L^* = L(g^*)$$

Plusieurs méthodes sont possibles pour estimer Y (en donnant une forme spécifique à la fonction f), et elles sont décrites plus en détails dans "The Elements of Statistical learning" [**?**].

2.3 Espace d'hypothèses linéaire en régression

2.3.1 Modèle de régression simple

Le cas de la régression simple est celui dans lequel on suppose que $f(X) = X\theta$ avec $\theta \in \mathcal{M}_{d,1}(\mathbb{R}) = \mathbb{R}^d$. On prédit Y grâce à $\widetilde{Y} = X\theta$. L'estimateur des moindres carrés minimise l'erreur de prédiction, c'est-à-dire qu'il est tel que :

$$\widehat{\theta} = \underset{\theta\in\mathbb{R}^d}{\text{argmin}} \|Y - X\theta\|^2$$

La solution analytique est la suivante :

$$\widehat{\theta} = ({}^tXX)^{-1}{}^tXY$$

En effet : $\widehat{\theta}$ est l'unique vecteur annulant la différentielle de $h : \theta \to ||Y - X\theta||^2$ qui a pour expression :

$$Dh(\theta)(H) = -2{}^tH({}^tXY - {}^tXX\theta)$$

car

$$\begin{aligned} h(\theta + tH) - h(\theta) &= \langle Y - X(\theta + tH), Y - X(\theta + tH)\rangle - \langle Y - X\theta, Y - X\theta\rangle \\ &= -2t\,\langle XH, Y - X\theta\rangle + o(t) \\ &= -2t\ {}^tH{}^tX(Y - X\theta) + o(t) \end{aligned}$$

2.3.2 Modèle de régression Ridge

Dans le cas où X est telle que ${}^tXX$ n'est pas inversible, le $\widehat{\theta}$ trouvé précédemment n'existe pas. Pour éviter ce genre de problème, Hoerl et Kennard proposent, dans leur article [**?**], de modifier un peu la matrice ${}^tXX$ afin de la rendre inversible. On va donc considérer :

$$\widehat{\theta}(\lambda) = ({}^tXX + \lambda I_d)^{-1}{}^tXY$$

La deuxième approche, plus théorique, est de diminuer l'espace des prédicteurs en imposant une pénalité. Elle permet aussi d'éviter le sur-apprentissage (**??**). $\widehat{\theta}$ est l'unique solution du problème :

$$\underset{\theta\in\mathbb{R}^d}{\text{argmin}} \left(\|Y - X\theta\|^2 + \lambda \|\theta\|^2\right)$$

D'autres possibilités de pénalisation sont envisageables. On peut par exemple considérer l'estimateur Lasso où la pénalité est d'une norme 1 (détaillée dans "The Elements of Statistical learning" [**?**]) :

$$\underset{\theta\in\mathbb{R}^d}{\operatorname{argmin}} \left(\|Y - X\theta\|^2 + \lambda \|\theta\|_1^2 \right)$$

Dans certains cas, la fonction à modéliser est non linéaire, et les régressions ne suffisent pas à en donner une bonne estimation. Plusieurs méthodes sont envisageables pour résoudre ce problème et nous allons en décrire ici deux. La première, les Kernels, est théorique et permet de combler les lacunes de la régression, tandis que la suivante, les arbres de décision, permet de s'affranchir de la régression et est beaucoup plus visuelle et intuitive.

2.4 Espace d'hypothèses non linéaire

2.4.1 Kernel

Représentation des données

Soit $\mathcal{X}$ un ensemble. On peut définir un objet comme un point $x \in \mathcal{X}$, et on note l'ensemble de données à analyser $S = \{x_1, .., x_n\}$

Une première représentation des données est une représentation dite "simple" et s'exprime de la manière suivante : On considère un algorithme A défini sur un espace F. Pour traiter S, il nous faut soit $\mathcal{X} = F$, soit définir une application $\Phi : \mathcal{X} \to F$ afin de travailler sur l'ensemble

$$\Phi(S) = \{\Phi(x_1), .., \Phi(x_n)\} \in F^n$$

Cependant, il existe une autre représentation des données appelée *représentation par comparaison*, dès lors, plutôt que de représenter chaque objet $x \in \mathcal{X}$ individuellement par $\Phi(x) \in F$, on utilise une *fonction de similarité* :

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

On peut alors représenter S par la matrice carré de similarité $n \times n$:

$$[K]_{ij} := K(x_i, x_j)$$

Noyaux définis positifs. *Un noyau défini positif (n.d.p) sur l'ensemble $\mathcal{X}$ est une fonction $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ symétrique telle que :*

$$\forall (x, x') \in \mathcal{X}^2, K(x, x') = K(x', x)$$

et qui satisfait pour tout $N \in \mathbb{N}, (x_1, .., x_N) \in \mathcal{X}^N et (a_1, .., a_N) \in \mathbb{R}^N$:

$$\sum_{i,j=1}^{N} a_i a_j K(x_i, x_j) \geq 0$$

Theorème. *Si K est un n.d.p sur un espace $\mathcal{X}$ quelconque, alors il existe un espace de Hilbert H muni du produit scalaire $\langle ., . \rangle_H$ et une application*

$$\theta : \mathcal{X} \to H$$

tels que :

$$\forall (x, x') \in \mathcal{X}^2, K(x, x') = \langle \theta(x), \theta(x') \rangle_H$$

Ainsi, les n.d.p peuvent s'écrire sous forme de produits scalaire dans un espace de Hilbert bien choisi. Cependant, il devient important de mieux connaitre ou définir l'espace dans lequel nous souhaitons travailler. Nous pouvons donc introduire dans un premier temps les Rkhs et par conséquent les noyaux reproduisant :

Définition. *Soit $\mathcal{X}$ un espace quelconque, et $(H, \langle ., . \rangle_H)$ un espace de Hilbert de fonction ($H \subset \mathbb{R}^{\mathcal{X}}$).*
Une fonction $K : \mathcal{X}^2 \to \mathbb{R}$ est appelée un noyau reproduisant (noté n.r) si et seulement si :

- *H contient toutes les fonctions de la forme*

$$\forall x \subset \mathcal{X}, K_x : t \to K(x, t)$$

- *Pour tout $x \in \mathcal{X}$ et $f \in H$, on a :*

$$f(x) = \langle f, K_x \rangle_H$$

Si un n.r existe, H est appelé un espace de Hilbert à noyau reproduisant (rkhs).

Il importe alors d'expliquer concrètement en quel sens l'usage des noyaux ou kernels serait intéressant pour notre problème. La puissance des noyaux définis positifs sur un espace $\mathcal{X}$ provient de deux résultats ayant d'importantes implications pratiques :

- L'**astuce noyaux** ou plus communément appelée **Kernel Trick**, basée sur la représentation d'un n.d.p comme produit scalaire.
- Le **théorème du représentant** (representer theorem) basé sur la fonctionnelle de régularisation définie par un n.d.p.

L'Astuce Noyau est une modification de l'espace sur lequel on travaille découlant de la proposition suivante :

Proposition. *Tout algorithme pour vecteurs qui puisse ne s'exprimer qu'en termes de produits scalaires entre vecteurs peut être effectué implicitement dans un espace de Hilbert en remplaçant chaque produit scalaire par l'évaluation d'un n.d.p sur un espace quelconque.*

Grâce à l'astuce noyau, il est alors possible de :

- Rendre non-linéaires dans l'espace initial des méthodes linéaires .
- Plonger l'espace initial dans un espace plus grand et y travailler implicitement.

De plus, notre problème de recherche de maxima d'une base se repose en réalité sur une régression, où en d'autres termes sur une minimisation de l'erreur : $\sum_{i=1}^{n}(Y_i - \hat{\theta}(X_i))^2 + \lambda \left\|\hat{\theta}\right\|_2$ Cependant, ayant modifié notre base d'apprentissage à l'aide des Kernel, ce n'est que grâce au théorème du représentant [**?**] que nous pouvons affirmer que ce problème de minimisation d'erreur peut s'effectuer tout aussi bien dans un *Rkhs* à l'aide de noyaux définis positifs :

Théorème du représentant (Kimeldorf et Wahba). *Soit $\mathcal{X}$ un ensemble muni d'un noyau défini positif K, H_K le rkhs et $S = (x_1, ..., x_n) \subset \mathcal{X}$ un ensemble fini d'objet. Soit $\mathcal{X}$ un ensemble muni d'un noyau défini positif K, H_K le rkhs et $S = (x_1, ..., x_n) \subset \mathcal{X}$ un ensemble fini d'objet. Soit $\Psi : \mathbb{R}^{n+1} \to \mathbb{R}$ une fonction de $n + 1$ arguments strictement croissante par rapport au dernier argument. Alors toute solution au problème :*

$$\min_{f \in H_K} \Psi \left(f(x_1), ..., f(x_n), \|f\|_{H_K} \right)$$

admet une représentation de la forme

$$\forall x \subset \mathcal{X}, f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$$

L'utilité des Kernels est ainsi démontré. En effet, au delà de permettre une extension implicite de notre espace de travail, nous venons de voir que nous pouvions également effectuer des régressions dans cet espace à l'aide de noyaux définis positifs.

Les kernels que nous utiliserons lors de la construction de nos bases d'apprentissages ou de validations (bases nécessaires dans la validation croisée que nous expliciterons par la suite) seront du types Polynomial ou Gaussien.

– *Kernel Polynomial*

$$K(x, x') = \langle x, x' \rangle^d = \left({}^t x x'\right)^d$$

Où l'on doit définir le degré d du Kernel.

– *Kernel Polynomial Non-homogène*

$$K(x, x') = \left(\langle x, x' \rangle + c\right)^d = \left({}^t x x' + c\right)^d, c > 0$$

– *Kernel Gaussien*

$$K(x, x') = \exp\left(\frac{\|x - x'\|^2}{2\sigma}\right)$$

Où l'on doit prédéfinir la variance σ.

2.4.2 Arbres de décision

Un arbre de décision est un outil qui permet de classifier ou régresser des données (X) et de représenter cette procédure sous la forme graphique d'un arbre. Il est dit arbre de régression lorsque le résultat $Y \in \mathbb{R}$ et arbre de classification lorsque Y$\in \{0, 1\}$. Les feuilles de l'arbre contiennent la valeur de Y (par exemple 0 ou 1 en classification, ou bien un réel en régression), tandis que les nœuds de l'arbre contiennent un test sur une des données $(X_i, i = 1..5)$, avec sur chaque branche partant du nœud un résultat différent de ce test.

Exemple d'un arbre de classification réalisé avec Matlab sur la base de données de 200 points des tsunamis, avec la règle suivante :

$$\widetilde{Y} = \begin{cases} 1 & si\ Y \geq 1.5 \\ 0 & sinon \end{cases}$$

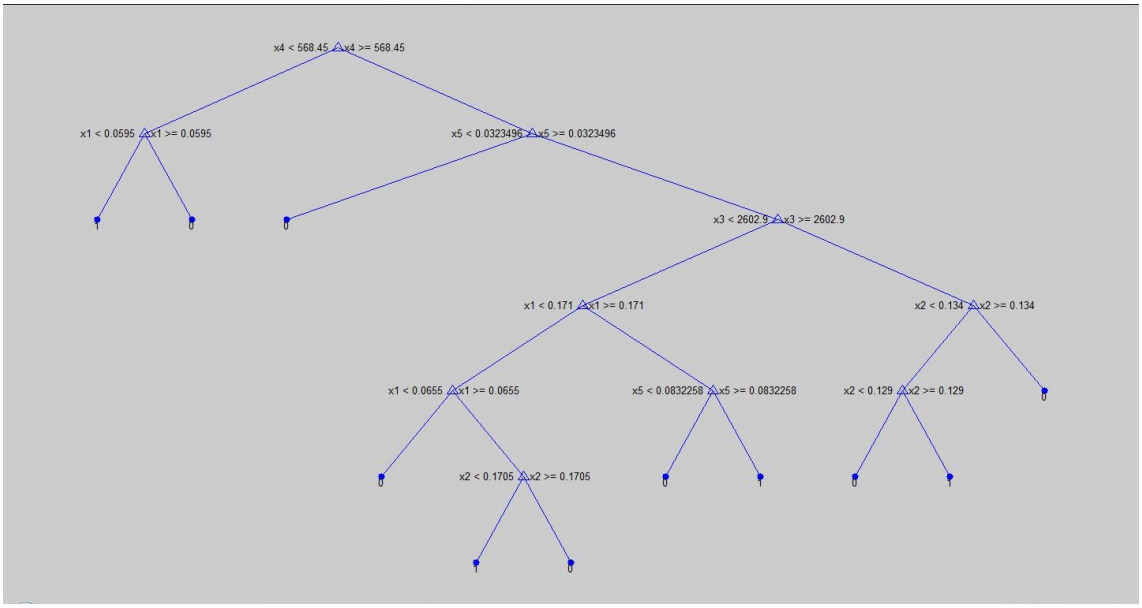


FIGURE 9 – Arbre de Classification

Le processus de réalisation d'un arbre est basé sur la dichotomie, comme décrit dans le livre d'Hastie et Tibshirani [**?**]. A chaque nœud, on souhaite diviser l'ensemble de données en 2 ensembles les plus homogènes possibles. On choisit donc le critère parmi les données X qui permet d'obtenir deux branches les plus pures possibles (pureté que l'on peut mesurer avec plusieurs méthodes, décrites ci-dessous) et on réitère ce procédé à chaque nœud de l'arbre. En général, le problème est de savoir à quel moment on arrête de construire l'arbre. Si l'on fait un arbre trop détaillé, il y a un fort risque de sur-apprentissage, où l'arbre colle totalement aux

données mais ne peut pas généraliser et donc faire une prédiction acceptable. On peut également fixer le nombre minimal de valeurs contenues dans les feuilles (nœud final). Si l'arbre est trop détaillé, on réalise ensuite un élagage pour enlever certaines feuilles.

Pour une classification où Y prend les valeurs $\{1, 2, .., K\}$, on note, pour un nœud m représentant une région R_m avec N_m observations :

$$\widehat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{y_i = k}$$

la proportion de la classe k dans le nœud m. Dans ce nœud, on donne comme valeur à $Yk(m) = \text{argmax}_k \ \widehat{p}_{mk}$.

Il existe différentes mesures de l'impureté de l'ensemble de données au nœud m :

Misclassification error : $\frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}_{y_i \neq k} = 1 - \widehat{p}_{mk}$

Gini index : $\sum_{k \neq k'} \widehat{p}_{mk} \widehat{p}_{mk'} = \sum_{k=1}^{K} \widehat{p}_{mk}(1 - \widehat{p}_{mk})$

Cross-entropy or deviance : $\sum_{k=1}^{K} \widehat{p}_{mk} \log(\widehat{p}_{mk})$

Algorithme ID3 :

On définit au préalable l'entropie de Shannon d'un ensemble S (avec Y qui prend n valeurs) :

$$Entropy(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

où p_i est la proportion de S appartenant à la classe i.

On définit aussi le gain d'un attribut A (ici les attributs sont les paramètres des données : les X_i pour $i = 1..d$ et Y) :

$$Gain(S, A) = Entropy(S) - \sum_{v \in valeurs \ de \ A} \frac{|S_v|}{|S|} Entropy(S_v)$$

où S_v est l'ensemble des éléments de S pour lesquels l'attribut A a pour valeur v.

L'algorithme est le suivant :

Pour que le programme donne une meilleure prédiction, on peut utiliser une technique de bagging : c'est un méta-algorithme permettant d'améliorer l'apprentissage en agrégeant plusieurs prédicteurs faibles et en effectuant leur moyenne. Ici pour rendre le programme plus robuste, on réalise une forêt aléatoire (random forest). On crée un certain nombre d'arbres avec S($\widetilde{X}$,Y), $\widetilde{X} = (X_i, X_j)$, $i \neq j$ pris aléatoirement dans $\{1..5\}$ (ou $\widetilde{X} = (X_i, X_j, X_k)$). Pour déterminer la prédiction d'un certain X, on fait passer X par tous les arbres de la random forest, puis on choisit Y comme la moyenne de toutes les valeurs données par chaque arbre.

Le principal inconvénient de l'apprentissage passif est que les résultats théoriques sont bons, mais seulement lorsque l'échantillon a une taille conséquente. Or, dans notre cas, on n'a que peu de points pour étudier la fonction f, et chaque exemple étiqueté est coûteux, puisqu'une simulation sur VOLNA dure au moins entre 1h et 1h30. On va donc avoir recours à l'apprentissage actif, qui,

Algorithme 1: ID3(R,C,S)

Input : S = (X,Y)
Input : R = colonnes 1 à d de S (paramètres)
Input : C = colonne d+1 de S (résultat)
while S_1,..,S_k *non vides* **do**
  if *S vide* **then**
    nœud simple et valeur “Failure”
  else if *S constitué de résultats ayant tous la même valeur* **then**
    feuille avec cette valeur
  else if *R vide* **then**
    arbre avec la valeur la plus fréquente qui apparaît dans les résultats de S
  else
    Soit A l'attribut avec le meilleur $Gain(A, S)$ parmi les attributs de R
    Soit $\{a_j, j = 1, 2, ..k\}$ les valeurs de A et $\{S_j, j = 1, 2..k\}$ les valeurs prises dans un sous-ensemble de S qui correspondent aux a_j

Output : Arbre avec la racine étiquetée par A et des branches étiquetées par les $a_j, j = 1, 2.., k$, et qui contiennent : ID3$(R\backslash\{A\}, C, S_1)$,..,ID3$(R\backslash\{A\}, C, S_k)$

au lieu d'utiliser tous les exemples étiquetés comme le fait le passif, sélectionne les exemples qui doivent être étiquetés selon un critère prédéfini. Ainsi, on peut en théorie sélectionner les points qui nous apportent le plus d'informations et s'affranchir des données qui en amènent le moins. L'intérêt principal de l'apprentissage actif réside donc en la diminution du temps computationnel tout en convergeant vers le prédicteur cible.

3 Apprentissage actif

À cause du manque de données étiquetées (en raison du temps computationel élevé), l'apprentissage passif n'est pas convaincant pour notre problème. L'apprentissage actif apporte une solution en explorant la base de données et en choisissant d'étiqueter ceux qui *a priori* sont les plus intéressants.

3.1 Classification

Ici on va voir principalement des algorithmes de classification, car il y a plus de résultats théoriques sur ces algorithmes que sur ceux de régression. Si l'on souhaite appliquer ces algorithmes à notre problème, on va donc devoir seuiller les cibles (en utilisant les fonctions indicatrices : $\mathbb{1}_{\{y>\text{seuil}\}}$) afin de se ramener à un problème de classification.

3.1.1 Cadre de travail

On travaille sur un ensemble de départ X et un ensemble d'arrivée Y. A chaque étape t du programme, on observe $x_t \in X$ et on doit décider si on demande ou non l'étiquette correspondante $y_t \in Y$. Z est appelé l'ensemble de prédiction, et on étudie alors les éléments de $H = \{h : X \to Z\}$. Le but est de trouver un classifieur (ou un prédicteur) $h \in H$ tel que $h(X)$ soit le plus proche possible de Y. Pour pouvoir comparer $h(X)$ et Y, on utilise des fonctions de perte $\ell : Z \times Y \to [0, \infty)$ et on calcule $\ell(h(X), Y)$. Voici plusieurs exemples de fonctions de perte :

$Y = Z = \{-1, 1\}$	
0-1 loss	$\ell(z, y) = \mathbb{1}_{\{y \neq z\}}$
$Y = \{-1, 1\}$ et $Z = \mathbb{R}$	
hinge loss	$\ell(z, y) = (1 - yz)_+$
logistic loss	$\ell(z, y) = \ln(1 + \exp(-yz))$
squared loss	$\ell(z, y) = (1 - yz)^2 = (y - z)^2$
absolute loss	$\ell(z, y) = \lvert 1 - yz \rvert = \lvert y - z \rvert$

On va étudier ici trois algorithmes dits de Query By Committee. On a au départ un espace de version contenant des hypothèses et on va, grâce aux nouveaux points de notre base et aux labels inférés ou demandés, réduire cet espace pour ne garder que les "bonnes" hypothèses.

3.1.2 Algorithme CAL

On suppose qu'il existe une hypothèse consistante pour notre problème, c'est-à-dire que nos données sont séparables. À chaque nouveau point, on élimine les hypothèses qui ne sont plus consistantes.

Principe de Cal :
Un exemple inconnu nous est fourni.
On apprend en supposant d'un côté que le label du nouveau point est $+1$ et de l'autre -1.
Si on prédit le même label dans les deux cas pour le nouveau point c'est qu'il n'est pas dans la région d'incertitude et on peut l'inférer.
Sinon on le demande et on recommence avec le point suivant.

On va considérer dans la suite une version légèrement modifiée de l'algorithme Cal [**?**], qui permet d'obtenir des résultats intéressants pour borner la complexité de l'algorithme (on veut savoir combien de points au plus on a besoin de demander pour obtenir une hypothèse qui a une erreur petite). Les notations sont les suivantes :

- Soit X une variable aléatoire sur $\mathcal{X}$ et Y une variable aléatoire sur $\{-1, 1\}$ telles que $(X, Y) \sim \mathcal{D}$.
- On suppose pour ces algorithmes que les données sont séparables et par conséquent on suppose l'existence de $h^* \in \mathcal{H}$ ($\mathcal{H}$ l'ensemble des hypothèses) telle que $\mathbb{P}(h(X) = Y) = 1$.
- On peut définir une pseudo-métrique sur $\mathcal{H}$: $\rho(h, h') = \mathbb{P}(h(X) \neq h'(X))$ et définir : $\mathcal{B}(h, r) = \{h' \in \mathcal{H} / \ \rho(h, h') \leq r\}$.
- Pour une hypothèse $h \in \mathcal{H}$ on peut définir son erreur : $\text{err}(h) = \mathbb{P}(h(X) \neq Y)$. Dans ce cas (existence de h^*) on a : $\text{err}(h) = \rho(h, h^*)$.
- Pour un ensemble d'exemples Z, on peut définir l'espace de version par rapport à $\mathcal{H}$: $\mathcal{V}(Z) = \{h \in \mathcal{H} / \ h(x) = y, \ \forall (x, y) \in Z\}$. On notera $\mathcal{V}_t = \mathcal{V}(Z_t)$.
- Pour un espace de version $\mathcal{V}$, on peut définir la région de désagrément : $\mathcal{R}(\mathcal{V}) = \{x \in \mathcal{X} / \ \exists h, h' \in \mathcal{V}, \ h(x) \neq h'(x)\}$. On notera $\mathcal{R}_t = \mathcal{R}(\mathcal{V}_t)$.
- Le coefficient de désaccord est définit comme suit :

$$\theta(\mathcal{H}, \mathcal{D}) = \sup \left\{ \frac{\mathbb{P}(X \in \mathcal{R}(\mathcal{B}(h^*, r))}{r} \ , \ r > 0 \right\}$$

L'algorithme est le suivant :

Le résultat sur cet algorithme qui apparait dans la thèse de Hsu [**?**] est le suivant :

Théorème 1. $\forall \epsilon, \delta \in]0; 1[$, *avec probabilité d'au moins* $1 - \delta$, *Phased* Cal *retourne une hypothèse* $h \in \mathcal{H}$ *qui a une erreur inférieure à* ϵ *après avoir demandé au plus* $O\left(\theta(\mathcal{H}, \mathcal{D}) \left(\ln\left(\frac{|\mathcal{H}|}{\delta}\right) + \ln\left(\ln\left(\frac{1}{\epsilon}\right)\right)\right) \ln\left(\frac{1}{\epsilon}\right)\right)$.

Algorithme 2: Phased CAL

Input : $Z_0 = \varnothing$
Input : $\mathcal{V}_0 = \mathcal{H}$
Input : $p = 0$
Input : $t_0 = 0$
for *t = 1,2...n* **do**
  Tirer X_t jusqu'à ce que $X_t \in \mathcal{R}(\mathcal{V}_{t_p})$
  Demander Y_t et on poser $Z_t = Z_{t-1} \cup \{(X_t, Y_t)\}$
  Poser $\mathcal{V}_t = \{h \in \mathcal{H} / \ h(X_i) = Y_i, \ \forall (X_i, Y_i) \in Z_t\}$
  Si $\mathbb{P}(X \in \mathcal{R}(\mathcal{V}_t)) \leq \frac{1}{2}\mathbb{P}(X \in \mathcal{R}(\mathcal{V}_{t_p}))$, poser $t_{p+1} = t$ et $p = p + 1$
Output : Tous les $h \in \mathcal{V}_T$

Démonstration. – Soit $p \in \mathbb{N}$, $\mathcal{H}_p := \{h \in \mathcal{V}_{t_p} / \ \rho(h, h^*) > r_p\} = \mathcal{V}_{t_p} \cap \mathcal{B}(h^*, r_p)^c$ qui sont les mauvaises hypothèses. $\forall h \in \mathcal{H}_p$, on a :

$$\mathbb{P}\left(h(X) \neq h^*(X) \mid X \in \mathcal{R}_{t_p}\right) = \frac{\mathbb{P}\left(h(X) \neq h^*(X)\right)}{\mathbb{P}\left(X \in \mathcal{R}_{t_p}\right)} = \frac{\rho(h, h^*)}{\mathbb{P}\left(X \in \mathcal{R}_{t_p}\right)} \geq \frac{r_p}{\mathbb{P}\left(X \in \mathcal{R}_{t_p}\right)} := c_p$$

Donc

$$\mathbb{P}\left(h(X) = h^*(X) \mid X \in \mathcal{R}_{t_p}\right) \leq 1 - c_p$$

Et comme les X_t sont IID, $(I_p = \{t_p + 1, ..., t_{p+1}\})$

$$\begin{aligned} \mathbb{P}\left(h(X_t) = h^*(X_t), \forall t \in I_p \mid X_t \in \mathcal{R}_{t_p}, \forall t \in I_p\right) &= \prod_{t=t_p+1}^{t_{p+1}} \mathbb{P}\left(h(X) = h^*(X) \mid X \in \mathcal{R}_{t_p}\right) \\ &\leq (1 - c_p)^{t_{p+1} - t_p} \end{aligned}$$

D'où

$$\mathbb{P}\left(\exists h \in \mathcal{H}_p / \ h(X_t) = h^*(X_t), \forall t \in I_p \mid X_t \in \mathcal{R}_{t_p}, \forall t \in I_p\right) \leq |\mathcal{V}_{t_p}|(1 - c_p)^{t_{p+1} - t_p} := \delta_p$$

δ_p représente la probabilité que l'on n'arrive pas à éliminer toutes les mauvaises hypothèses.

– On vient de voir qu'avec probabilité au moins $1 - \delta_p$, $\mathcal{V}_{t_{p+1}} \subseteq \mathcal{B}(h^*, r_p)$ et en posant $r_p = \frac{\mathbb{P}\left(X \in \mathcal{R}_{t_p}\right)}{2\,\theta(\mathcal{H}, \mathcal{D})}$, on a bien (avec cette probabilité) :

$$\begin{aligned} \mathbb{P}\left(X \in \mathcal{R}_{t_{p+1}}\right) &\leq \mathbb{P}\left(X \in \mathcal{R}\left(\mathcal{B}(h^*, r_p)\right)\right) \\ &\leq \theta(\mathcal{H}, \mathcal{D})\, r_p \qquad \text{(par définition de } \theta) \\ &\leq \frac{1}{2}\, \mathbb{P}\left(X \in \mathcal{R}_{t_p}\right) \end{aligned}$$

– Dans ce cas, $c_p = \frac{1}{2\,\theta(\mathcal{H}, \mathcal{D})}$ et $\delta_p = |\mathcal{V}_{t_p}| \left(1 - \frac{1}{2\,\theta(\mathcal{H}, \mathcal{D})}\right)^{t_{p+1} - t_p} \leq |\mathcal{V}_0|\, e^{-\frac{t_{p+1} - t_p}{2\,\theta(\mathcal{H}, \mathcal{D})}}$.
Les mauvaises hypothèses $\{\mathcal{H}_p, \ p \in \{0, 1, ..., P-1\}\}$ sont éliminées avec probabilité d'au moins :

$$\begin{aligned} \prod_{p=0}^{P-1} (1 - \delta_p) &\geq \prod_{p=1}^{P-1} (1 - \delta_p) - \delta_0 \underbrace{\prod_{p=1}^{P-1} (1 - \delta_p)}_{\leq 1} \\ &\geq 1 - \sum_{p=0}^{P-1} \delta_p \\ &\geq 1 - \sum_{p=0}^{P-1} |\mathcal{V}_0|\, e^{-\frac{t_{p+1} - t_p}{2\,\theta(\mathcal{H}, \mathcal{D})}} \end{aligned}$$

– Si on suppose que l'on a $\forall p \in \{0, 1, ..., P-1\}, t_{p+1} - t_p = \left\lceil 2\ \theta(\mathcal{H}, \mathcal{D})\ \ln\left(\frac{P|\mathcal{V}_0|}{\delta}\right)\right\rceil$, alors on a que les mauvaises hypothèses sur toutes les phases sont éliminées avec probabilité supérieure à $1-\delta$ après avoir demandé :

$$\sum_{p=0}^{P-2} t_{p+1} - t_p = O\left(\theta(\mathcal{H}, \mathcal{D})\ \ln\left(\frac{P|\mathcal{V}_0|}{\delta}\right) P\right)$$

– Enfin, pour que l'hypothèse retournée ait une erreur inférieur à ϵ, on pose :

$$P = \frac{\ln\left(\frac{\mathbb{P}(X \in \mathcal{R}_{t_0})}{\theta(\mathcal{H}, \mathcal{D}} \frac{1}{\epsilon}\right)}{\ln(2)} - 1 = O\left(\ln\left(\frac{1}{\epsilon}\right)\right)$$

En effet on a bien :

$$r_P = \frac{\mathbb{P}(X \in \mathcal{R}_{t_P})}{2\ \theta(\mathcal{H}, \mathcal{D})} \leq \frac{\mathbb{P}(X \in \mathcal{R}_{t_{P-1}})}{2^{1+1}\ \theta(\mathcal{H}, \mathcal{D})} \leq \frac{\mathbb{P}(X \in \mathcal{R}_{t_0})}{2^{P+1}\ \theta(\mathcal{H}, \mathcal{D})} = \epsilon$$

D'où, avec probabilité d'au moins $1-\delta$, Phased Cal retourne une hypothèse $(h \in \mathcal{V}_{t_{P+1}} \subseteq \mathcal{B}(h^*, r_P(\leq \epsilon)))$ qui a une erreur inférieure à ϵ après avoir demandé au plus $O\left(\theta(\mathcal{H}, \mathcal{D})\left(\ln\left(\frac{|\mathcal{H}|}{\delta}\right) + \ln\left(\ln\left(\frac{1}{\epsilon}\right)\right)\right)\ln\left(\frac{1}{\epsilon}\right)\right)$.

□

Si on suppose par exemple que $\theta(\mathcal{H}, \mathcal{D}) = O(1)$, ce qui n'est pas toujours le cas, alors pour une erreur inférieure à ϵ, le nombre de demande est de l'ordre de $O(\ln(1/\epsilon) \cdot \ln(\ln(1/\epsilon)))$ contre $\Omega(1/\epsilon))$ pour le passif. On a donc une amélioration exponentielle ! Mais les performances de cet algorithme dépendent du coefficient de désaccord qu'on ne sait pas calculer de manière simple *a priori*.

On va voir dans la suite un algorithme un peu plus évolué, qui tente d'inférer même dans la région d'incertitude, pour encore diminuer le temps computationnel.

3.1.3 Algorithme DHM

Ici on ne suppose pas que l'on est dans le cas séparable mais dans le cas agnostique, c'est-à-dire que on ne suppose plus qu'il existe une hypothèse consistante pour notre problème. On décompose les points en deux ensembles : ceux que l'on a inférés (sur lesquels on reste consistant) et ceux que l'on a demandés (sur lesquels on minimise l'erreur).

Principe de Dhm :
On commence comme dans Cal à apprendre les deux modèles sur un nouvel exemple qui nous est fourni.
Si le point n'est pas dans la région d'incertitude, on l'infère.
Sinon, si l'un des deux modèles a une erreur suffisamment plus grande que l'autre, on infère (ex : si le modèle positif a une erreur trop grande, on infère -1).
Enfin, si on n'arrive pas à inférer (on ne peut pas préférer une méthode à l'autre parce qu'elles ont des erreurs proches), on demande.

On a le théorème suivant qui nous donne une borne sur le nombre de points demandés pour Dhm :

Théorème. *Il existe* $C \in]0, 25[$ *tel que la probabilité que le nombre de points demandés après* n *itérations soit :*

$$1 + 2 \cdot \theta \cdot err(h^*) \cdot (n-1) + 4 \cdot \theta \cdot \sqrt{C \cdot err(h^*) \cdot \left(2d \ln \frac{2en}{d} + 2 \ln \frac{24n}{\delta}\right) \cdot (n-1) \cdot \ln n} + 4 \cdot C \cdot \theta \cdot \left(2d \ln \frac{2en}{d} + 2 \ln \frac{24n}{\delta}\right) \cdot \ln n$$

est supérieure à $1-\delta$.

La démonstration, très technique en soi, est détaillée dans la dissertation de Hsu [**?**] et utilise notamment un résultat de borne uniforme de Vapnik et Chervonenkis [**?**].

Algorithme 3: DHM

Input : $\tilde{S}_0 = \varnothing$
Input : $T_0 = \varnothing$
for *t = 1,2... n* **do**
  Tirer un nouveau point X_t
  $h_t = \text{LEARN}_{\mathcal{H}}(\tilde{S}_{t-1}, T_{t-1})$(c'est l'hypothèse dans $\mathcal{H}$ qui est consistante avec $\tilde{S}_{t-1}$ et qui minimise l'erreur sur T_{t-1})
  $h'_t = \text{LEARN}_{\mathcal{H}}(\tilde{S}_{t-1} \cup \{(X_t, -h_t(X_t))\}, T_{t-1})$
  Si $h'_t \neq \perp$ et $\text{err}(h'_t, \tilde{S}_{t-1} \cup T_{t-1}) - \text{err}(h_t, \tilde{S}_{t-1} \cup T_{t-1}) \leq \Delta(h'_t, h_t, \tilde{S}_{t-1} \cup T_{t-1}, \delta_{t-1})$, demander Y_t et poser $\tilde{S}_t = \tilde{S}_{t-1}$ et $T_t = T_{t-1} \cup \{(X_t, Y_t)\}$
  Sinon poser $\tilde{Y}_t = h_t(X_t)$, $\tilde{S}_t = \tilde{S}_{t-1} \cup \{(X_t, \tilde{Y}_t)\}$ et $T_t = T_{t-1}$
Output : $h_{n+1} = \text{LEARN}_{\mathcal{H}}(\tilde{S}_n, T_n)$

Biais d'échantillonnage

Malgré les résultats théoriques sur les algorithmes CAL et DHM, on remarque que ceux-ci ne convergent pas toujours vers le classifieur optimal. Ceci est du au biais d'échantillonnage, soulevé par Dasgupta [**?**], qui est un des principaux problèmes de l'apprentissage actif. Les algorithmes d'apprentissage actif commencent avec un ensemble de quelques points tirés au hasard qui sont donc *a priori* représentatifs de la distribution P inconnue. Cependant, lorsque l'apprentissage progresse, l'ensemble des points est de moins en moins représentatifs de cette distribution P, puisque l'on s'intéresse principalement aux points qui apportent le plus d'informations, et donc les plus marginaux. Un classifieur qui apprend sur cet ensemble ne sera donc pas un bon estimateur du classifieur optimal.

Expliquons ce problème à partir d'un exemple en une dimension, illustré par la figure suivante.

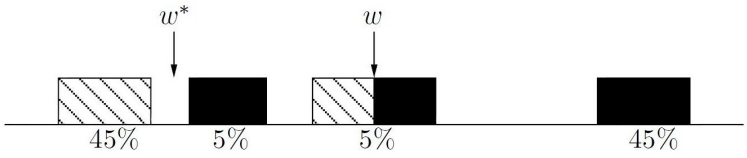


FIGURE 10 – Exemple de biais d'échantillonnage en une dimension

Dans ce cas, 90% des données est répartie dans les deux groupes aux extrémités ; ainsi, lors du tirage aléatoire initial, on a beaucoup de chances d'obtenir un ensemble contenant seulement des données appartenant à ces deux groupes. Ainsi, le classifieur optimal sera le suivant :

$$h_w(x) = \begin{cases} +1 & si\ x \geq w \\ -1 & si\ x \leq w \end{cases}$$

La première séparation sera donc située dans le groupe du centre, contenant 5% des données, moitié à +1 et moitié à -1. Le premier point demandé appartiendra donc au groupe central, donc la séparation sera inchangée. L'algorithme actif convergera donc vers le classifieur en w. Cependant ce classifieur à 5% d'erreur, alors que le classifieur en w^* n'a que 2.5% d'erreur. Le schéma d'apprentissage n'est donc pas consistant : pour un nombre infini de données, il ne convergera pas vers le classifieur optimal.

Le problème soulevé ici est que l'apprentissage actif peut laisser de côté certaines zones où se situent les données, et cette non-exploration de toutes les données peut empêcher un apprentissage correct et une convergence vers le classifieur optimal. Ce problème n'étant qu'un exemple en une

dimension, on peut imaginer que pour des dimensions plus élevées, il sera inéluctable.

L'algorithme IWAL a donc pour but d'éliminer ce problème de biais d'échantillonnage, en permettant d'explorer toutes les zones où se situent les données, en introduisant des probabilités pouvant être plus ou moins élevées.

3.1.4 Algorithme IWAL

L'algorithme IWAL (Importance Weighted Active Learning) a été développé par Beygelzimer, Dasgupta et Langford [**?**], afin de s'affranchir du biais d'échantillonnage, qui est une des principales difficultés de l'active learning, mais a aussi l'avantage non négligeable d'être consistant, c'est-à-dire qu'avec une infinité d'exemples (étiquetés et non étiquetés), il convergera vers le meilleur classifieur.

Le principe général d'IWAL est d'attribuer au temps t la probabilité p_t de demander y_t (en tirant $Q_t \in \{0,1\}$ tel que $\mathbb{E}(Q_t) = p_t$ et en demandant y_t si $Q_t = 1$), connaissant x_t et tout ce qui s'est passé avant, c'est-à-dire l'ensemble $\{x_i, y_i, p_i, Q_i, 1 \leq i < t\}$. A chaque itération, on a donc un ensemble d'exemples étiquetés auxquels on attribue des coefficients d'importance : si y_t est demandé, on ajoute $(x_t, y_t, \frac{1}{p_t})$ à l'ensemble des exemples. On peut donner une interprétation de la proportionnalité entre l'importance du couple (x_t, y_t) et $\frac{1}{p_t}$; en effet, les points ayant une faible probabilité d'être tirés (donc p_t faible) sont ceux dont on est presque sûr de connaître l'étiquette y_t. Si ils sont tirés, ils doivent donc être considérés comme importants (apportant une information considérable), d'où un poids inversement proportionnel à p_t.

La probabilité p_t de tirer y_t peut être déterminée par différents sous-programmes, dont un sera développé plus tard. Il est appelé *rejection-threshold*. On considère aussi qu'il existe un p_{min} tel que pour tout t, $p_t \geq p_{min}$. L'algorithme général est le suivant :

Algorithme 4: IWAL(sous-programme, p_{min})

Input : $S_0 = \emptyset$
for *t = 1,2... n* **do**
  p_t = *rejection-threshold* (x_t,passé $(x_i, y_i, Q_i, 1 \leq i < t)$)
  Tirer $Q_t \in \{0,1\}$ tel que $\mathbb{E}(Q_t) = p_t$
  Si $Q_t = 1$, demander y_t et poser $S_t = S_{t-1} \cup \{(x_t, y_t, \frac{p_{min}}{p_t})\}$
  Sinon $S_t = S_{t-1}$
Output : $h_t = \text{argmin}_{h \in H} \sum_{(x,y,c) \in S_t} c.\ell(h(x), y)$

On s'intéresse finalement à la consistance de l'algorithme IWAL. Pour cela, nous devons définir le risque de $h \in H$ par rapport à la distribution D de (X, Y) :

$$L_D(h) = \mathop{\mathbb{E}}_{(x,y) \sim S}[\ell(h(x), y)]$$

Définissons aussi "l'importance weighted estimator" au temps T :

$$L_T(h) = \frac{1}{T} \sum_{t=1}^{T} \frac{Q_t}{p_t} \ell(h(x_t), y_t)$$

Tout d'abord, on peut montrer que $\mathbb{E}[L_T(h)] = L(h)$, d'où l'algorithme n'est pas biaisé.

On peut démontrer cette égalité en conditionnant par rapport au passé $\{x_i, y_i, 1 \leq i < t, Q_i, 1 \leq i < t-1\}$:

$$\mathbb{E}[L_T(h)] = \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\frac{Q_t}{p_t} \ell(h(x_t), y_t)]$$

$$
\begin{aligned}
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\mathbb{E}[\frac{Q_t}{p_t}\ell(h(x_t),y_t)|\{x_i,y_i,1\le i<t,Q_i,1\le i<t-1\}]]\\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\mathbb{E}[Q_t|\{x_i,y_i,1\le i<t,Q_i,1\le i<t-1\}]\frac{\ell(h(x_t),y_t)}{p_t}]\\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[p_t\frac{\ell(h(x_t),y_t)}{p_t}]\\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\ell(h(x_t),y_t)]\\
&= \mathbb{E}[\ell(h(X),Y)]\\
&= L(h)
\end{aligned}
$$

Le théorème suivant permet de justifier la consistance de l'algorithme :

Théorème. *Pour toute distribution D, pour tout ensemble fini H de classes, si il existe une constante $p_{min} > 0$ telle que pour tout $1 \le t < T$, $p_{min} \le p_t$, alors pour tout $\delta > 0$,*

$$P(L_T(h) - L(h) > \frac{\sqrt{2}}{p_{min}}\sqrt{\frac{ln|H| + ln\frac{1}{\delta}}{T}}) < \delta$$

Démonstration. La démonstration utilise la suite de variables aléatoire $U_t = p_{min}(\frac{Q_t}{p_t}\ell(h(x_t),y_t) - L(h)$.

En notant $Z_t = \sum_{i=1}^{t} U_i$, on peut montrer qu'avec $Z_0 = 0$, $\{Z_t\}$ est une martingale (l'espérance de U_t sachant le passé est nulle).
Comme on a aussi $|U_t| \le 1$, puisque $p_{min} \le p_t$, $L(h) \le 1$ et dans le cas du 0-1 loss, $\ell(h(x_t),y_t) = 0$ ou 1. Ainsi

$$\forall t, |Z_t - Z_{t-1}| = |U_t| \le 1$$

On peut alors appliquer l'inégalité d'Azuma - Hoeffding, qui stipule que pour une martingale $\{X_k\}$ telle que $|X_k - X_{k-1}| \le c_k$ presque surement, alors

$$P(X_N - X_0 \ge t) \le exp(\frac{-t^2}{2\sum_{k=1}^{N} c_k^2})$$

Pour la martingale $\{Z_t\}$, on obtient

$$P(Z_t > \lambda\sqrt{T}) \le exp(-\frac{\lambda^2}{2})$$

Comme $L_T(h) - L(h) = \frac{Z_T}{Tp_{min}}$, et on veut $\frac{\lambda}{p_{min}\sqrt{T}} > \frac{\epsilon}{2}$, on obtient, pour tout les $h \in H$ simultanément :

$$P(L_T(h) - L(h) > \frac{\epsilon}{2}) \le |H|e^{-Tp_{min}^2\frac{\epsilon^2}{8}}$$

On pose ensuite $\delta = |H|e^{-Tp_{min}^2\frac{\epsilon^2}{8}}$, soit $\frac{\epsilon}{2} = \frac{\sqrt{2}}{p_{min}}\sqrt{\frac{ln|H| - ln|\delta|}{T}}$ et on obtient l'inégalité recherchée. □

Exemple d'un sous-programme permettant d'attribuer une valeur à p_t : *loss-weighting*

Définissons $\Delta_t = \sqrt{\frac{8}{t}ln\frac{t(t+1)|H_t|^2}{\delta}}$

En pratique, l'ensemble H des classifieurs n'est pas de cardinal fini (en dimension 2, c'est un ensemble infini de droites) et donc on ne peut pas calculer p_t comme étant un maximum sur les ensembles H_t (infinis aussi). L'algorithme théorique n'est donc pas implémentable et on utilise

Algorithme 5: Loss-weighting(x,history$\{x_i, y_i, p_i, 1 \leq i < t\}$)

Input : $H_0 = H$

for *t = 1,2... n* **do**

$$L^*_{t-1} = \min_{h \in H_{t-1}} \frac{1}{t-1} \sum_{i=1}^{t-1} \frac{Q_i}{p_i} \ell(h(x_i), y_i)$$

$$H_t = \{h \in H_{t-1} / \tfrac{1}{t-1} \sum_{i=1}^{t-1} \frac{Q_i}{p_i} \ell(h(x_i), y_i) \leq L^*_{t-1} + \Delta_{t-1}\}$$

Output : $p_t = \max_{\substack{f,g \in H_t \\ y \in Y}} (\ell(f(x), y) - \ell(g(x), y))$

plutôt un autre algorithme, basé sur des heuristiques comparables à ceux utilisés dans CAL [**?**].

En conclusion, on peut obtenir de bons résultats théoriques sur les algorithmes suivant l'approche Query By Committee, cependant ils ne sont pas forcément très efficaces pour notre problème. En effet, nous cherchons pas à ajuster la fonction de manière globale mais nous souhaitons plutôt déterminer ses maxima. Il est donc nécessaire d'adapter l'active learning à notre problème de régression.

3.2 Régression

3.2.1 Cadre de travail

Ayant dès lors mieux ciblé notre véritable problème, il est nécessaire d'adapter le raisonnement de l'*apprentissage actif* à la régression. L'algorithme ainsi utilisé peut se présenter selon trois principaux axes. Dans un premier temps, il s'agit d'apprendre sur une base de données initiale un estimateur de Y que l'on notera par la suite $\hat{\theta}$ ou $\hat{f}(X)$. Il s'agit ensuite de choisir le plus pertinemment possible l'ensemble des points à demander à VOLNA afin de pouvoir explorer la base tout en recherchant prioritairement à établir les différents maxima. Il nous importe enfin dans un troisième et dernier moment, d'établir un critère d'arrêt pour l'algorithme. Ce critère permettra ainsi de savoir quand notre estimateur est suffisamment performant ou en d'autres termes, de limiter le temps computationnel de VOLNA à l'identification des maxima qui nous intéressent. Les deux dernières parties de cet algorithme seront traitées plus en détails dans les sections 2 et 3.
Il s'est avéré après diverses expériences, que la régression la mieux adaptée à notre problème de recherche de maxima serait une régression de type Ridge.

L'algorithme est le suivant :

Algorithme 6: ActiveRegression

Input : $\mathcal{B} = \{(x_i, y_i)\}_{i \leq b}$ base de donnée initiale

Input : $\hat{\theta}_{prec}$

$S = \mathcal{B}$

while $E >$ *seuil* **do**

$\hat{\theta} = Learn(S)$

$X_q = \text{ARGMAX}(S)$

$Y_q = \text{VOLNA}(X_q)$

$S = S \cup (X_q, Y_q)$

$E = StoppingCriterion(\hat{\theta}, \hat{\theta}_{prec})$

$\hat{\theta}_{prec} = \hat{\theta}$

Output : $\hat{\theta}$

3.2.2 Sélection des requêtes

Comme nous l'avons introduit précédemment, il nous faut au cours de chaque itération demander certains points (le nombre est un paramètre à fixer) à VOLNA afin d'affiner notre régression et de trouver trouver nos maximum recherchés. Il semble évident que les points ne devraient être demandés de façon aléatoire, mais de manière réfléchie afin d'être le plus efficace possible et d'optimiser ainsi l'aspect *actif* de notre code. Il s'agit donc de combiner deux aspects distincts de demande de points : l'exploration de la base et l'exploitation.

Deux méthodes principales sont alors ressorties de nos différentes expériences. La méthode dite d'ARGMAX et une méthode plus mathématique dite MAXLIPSCHITZ.

ARGMAX : L'algorithme ARGMAX demande lors de l'itération $n+1$, les étiquettes des p premiers X vérifiant

$$X = \underset{X_i \in X}{\operatorname{argmax}}(\hat{Y}_i) \tag{2}$$

On note $\hat{Y}_i$ l'ensemble des prédictions effectuées sur chaque X_i et obtenues grâce à notre régresseur. il s'agit donc dans cet algorithme d'axer nos recherches sur les maximum dès le début du code sans explorer parallèlement l'ensemble de la base. Cet algorithme n'est efficace que si nous travaillons sur une base relativement simple ou lorsque les maxima sont aisément identifiables. Cette remarque sera d'ailleurs illustrée dans le compte-rendu de nos expériences.

MAXLIPSCHITZ : soit $\mathbb{L}_k$ l'ensemble des fonctions $k - lipschitziennes$ (soit dans notre cas, l'ensemble des fonctions de pentes limitées) et l'ensemble $\mathcal{V}_n$ tel que, lors de l'itération n, on ait :

$$\mathcal{V}_n = \{f \mid f \in \mathbb{L}\neg, f(x_i) = y_i, \forall (x_i, y_i) \in \mathcal{B}\}$$

La valeur de la constante k est quant à elle définie comme étant la pente maximale des fonctions affines permettant de relier deux points voisins de notre base initiale. L'algorithme MAXLIPSCHITZ consiste donc à demander à VOLNA les étiquettes des p points vérifiant :

$$X = \underset{x \in X}{\operatorname{argmax}} \max_{f \in \mathcal{V}_n} f(x) \tag{3}$$

La figure 1 montre un exemple en une dimension sur une base d'apprentissage de trois points : $\mathcal{B} = \{x_1, x_2, x_3\}$. Dans cet exemple, on ne considère que l'ensemble des fonctions au mieux 1-Lipschitzienne et on ne demandera qu'une unique étiquette, soit celle du meilleur x vérifiant (2) . On demande donc à VOLNA l'étiquette de x_{max} en tant que potentiel maximum de notre base.

L'algorithme MAXLIPSCHITZ allie donc de façon beaucoup plus concrète que ARGMAX l'exploration à l'exploitation. En effet, au delà de cibler les recherches sur les probables maxima indiqués par notre régression, il force l'exploration en maximisant les cibles potentielles $\hat{y} = f(x)$ où $f \in \mathbb{L}_k$.

3.2.3 Critère d'arrêt

Un des aspects de l'active learning jusqu'à aujourd'hui relativement négligé (et pourtant primordial), n'est autre que la définition d'un critère d'arrêt efficace. Bien que cette partie de l'algorithme soit évidemment extrêmement importante afin de pouvoir éviter toute saturation ou tout simplement de pouvoir limiter le temps de travail des machines, celui-ci reste obscur, ne se voyant pas décerné de théorème ou de loi d'utilisation.
Dépourvus de documents scientifiques, nous avons du créer notre propre critère d'arrêt. Étant basé sur un algorithme de régression afin de trouver les maxima d'une base au demeurant

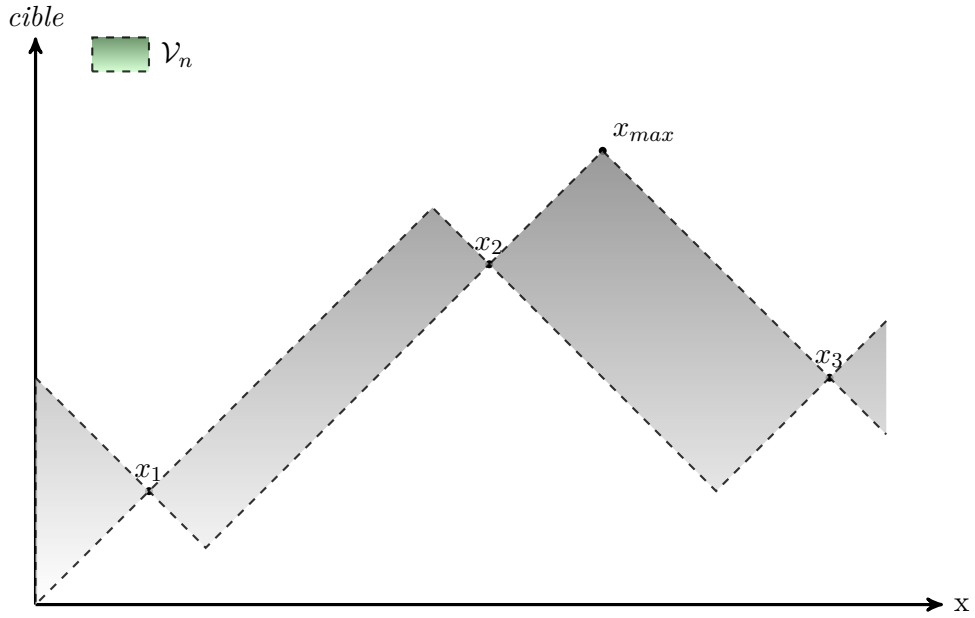


FIGURE 11 – Algorithme MAXLIPSCHITZ sur $\mathcal{B} = \{x_1, x_2, x_3\}$

inexplorée, nous avons jugé pertinent de fonder notre critère d'arrêt sur la 'classification' des points obtenus, en d'autres termes : étudier la différence de hiérarchisation des maxima entre deux itérations.

Considérons deux itérations successives du code actif, on note Y_n et Y_{n+1} les prédictions des itérations respectives n et $n+1$. Au cours de chaque itération, nous demandons à VOLNA p nouveaux points choisis de manière spécifique afin d'optimiser la recherche des maxima de notre base. Étant donné que nous débutons l'algorithme en connaissant i points, on peut affirmer qu'à chaque nouvelle itération n nous connaissons $i + p \times n$ points.
Notre critère d'arrêt ne peut s'effectuer sur la base étudiée directement. En effet, à chaque itération, on demande de nouveaux points à VOLNA. Ceux-ci sont ensuite ajoutés à sa base d'apprentissage qui est donc dynamique. Les modifications de taille de la base d'apprentissage ne nous permettent donc pas d'obtenir des vecteurs de prédictions de tailles identiques d'une itération sur l'autre. On se propose donc de construire une base indépendante notée Γ avec la méthode LHS, et de voir comment les prédictions sur cette base réagissent suite aux modifications de la base d'apprentissage. Étant donné que nous cherchons à établir $\hat{\theta}$ nous permettant de trouver les maxima sur une base donnée, on peut vérifier que ce $\hat{\theta}$ obtenu, et ajusté à chaque itération, ordonne les points d'une base dite annexe de la même façon. L'intérêt d'utiliser une base annexe est d'obtenir des vecteurs Y_n et Y_{n+1} de taille identique d'une itération sur l'autre ainsi qu'une base de validation fixe. En d'autres termes, si les prédictions sont stables au cours des différents ajouts de données, on peut supposer que l'algorithme tend à converger vers une solution optimale.
Notre vecteur Y_n (de taille fixe d'une itération sur l'autre) est alors comparé au vecteur Y_{n+1} dans le sens ou l'on compare le classement des points de chaque vecteur. Ainsi d'une itération sur l'autre, on regarde si l'algorithme a classé les points obtenus avec la régression dans le même ordre.

Considérons les vecteurs $Y_n = (Y_{n,1}, .., Y_{n,k})$ et $Y_{n+1} = (Y_{n+1,1}, .., Y_{n+1,k})$ les vecteurs prédits par $\hat{\theta}_n$ et $\hat{\theta}_{n+1}$ sur la base $\Gamma = (\gamma_1, .., \gamma_k)$. On crée alors les vecteurs R_n (respectivement R_{n+1}) qui rendent compte des rangs de chaque point $Y_{n,i}$ (respectivement $Y_{n+1,i}$). On a donc $r_{n+1,i} \in \{1, 2, ..., k\}$. Pour illustrer, si $Y_{n,i}$ est le maximum, on aura $r_{n,i} = 1$, et si $Y_{n,i}$ est le minimum, on aura $r_{n,i} = k$. Il s'agit donc d'établir une formule rendant compte des variations de ces classements d'une itération sur l'autre. Pour cela nous nous sommes inspirés du DCG, une mesure très utilisée en recherche d'information(Cf **4.1.1**). Il nous faut donc une *mesure de désaccord* à l'itération n notée E_n, qui tend vers 0. N'ayant pas de formule canonique, il nous

a fallu tester cette formule sur différentes bases et avec différents algorithmes de sélection de points. On a choisi la formule heuristique suivante :

$$E_{n+1} = \sum_{j=1}^{k} \frac{|r_{n+1,i} - r_{n,i}|^2}{r_{n+1,i}^2}$$

Cette formule nous semble la meilleure puisque l'on se focalise réellement sur les premiers maxima. En effet, le numérateur compare les rangs. S'ils sont proches, l'erreur est négligeable. Le dénominateur joue également un rôle important car il nous permet d'accentuer l'erreur lorsque celle ci concerne les rangs des maxima (qui sont donc petits) tandis que les erreurs faites sur les points de rangs plus élevés deviennent elles très faibles.

L'algorithme utilisé est le suivant :

Algorithme 7: StoppingCriterion

Input : $(\hat{\theta}, \hat{\theta}_{prec})$
Input : $\Gamma \subset X$ base de validation utilisée pour le critère d'arrêt
$r_{prec} = rank(\left\langle \hat{\theta}_{prec}, \Gamma \right\rangle)$
$r = rank(\left\langle \hat{\theta}, \Gamma \right\rangle)$
$E = \sum_{j=1}^{k} \frac{|r_i - r_{prec,i}|^2}{r_i^2}$
Output : E

Il nous faut donc maintenant définir un seuil, heuristique une nouvelle fois, de l'erreur de classement (propre à chaque algorithme de demande) en dessous duquel on souhaite arrêter l'algorithme.

4 Résultats expérimentaux

4.1 Données simulées

L'étude des différents algorithmes s'effectue sur les deux bases suivantes :
- Tsunamis : base de 200 points obtenus grâce à VOLNA en cinq dimensions.
- Gaussienne : base de 1000 points en deux dimensions générés manuellement, composée d'une somme de 3 gaussiennes. L'intérêt de générer une telle base est de pouvoir tester la capacité des algorithmes à ne pas rester coincé sur un maximum local mais à bien explorer l'ensemble de la base.

4.2 Validation des méthodes

On a un ensemble de données constitué des exemples X et des résultats Y. On cherche une fonction f telle que $Y = f(X) + \epsilon$. Une des méthodes les plus utilisées dans l'apprentissage consiste à séparer l'ensemble des données en plusieurs échantillons ayant chacun un rôle différent.

Le premier ensemble de données (X_{train}, Y_{train}) permet l'apprentissage du modèle. On calcule $\widehat{f}$ de façon à ce que l'erreur entre $\widehat{f}(X_{train})$ et Y_{train} soit la plus faible possible.

Le deuxième ensemble (X_{valid}, Y_{valid}) permet de valider et d'ajuster les paramètres du problème, en gardant ceux qui donnent la plus petite erreur possible par rapport au modèle déterminé. On évite ainsi d'avoir un modèle biaisé par la base d'apprentissage.

Le dernier (X_{test}, Y_{test}) permet de tester le modèle en lui-même et d'évaluer l'erreur effectuée lors de l'apprentissage (en calculant l'erreur entre les Y_{test} et les Y prédits par le modèle appris sur le premier ensemble aux X_{test}).

Le problème principal est de savoir de quelle manière on va évaluer l'erreur. On note n la taille de l'échantillon X et on évalue l'erreur entre Y et $f(X)$. On définit dès lors la liste d'erreur d'évaluation suivante :

Type de méthode	Type d'erreur
Régression	Erreur quadratique moyenne : $\sum_{i=1}^{n}(Y_i - f(X_i))^2$
Classification	Taux d'erreur : $\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{f(X_i)\neq Y_i}$
Ranking	Spearman Correlation : 1 - 6 $\sum_{i=1}^{n}\frac{d_i^2}{n(n^2-1)}$ où d_i est, lorsqu'on a classé Y et $f(X)$ en suite décroissante, la différence de rang entre Y_i et $f(X_i)$
	Discounted Cumulated Gain : $DCG(Y,s) = \sum_{i=1}^{n}\frac{2^{Y_i}-1}{ln(1+i)}$ où s est la permutation ordonnant les $f(X_i)$ dans l'ordre décroissant
	NormalizedDCG : $NDCG(Y,s)=\frac{DCG(Y,s)}{\max_{s'} DCG(Y,s')}$
	DCG@k : calcul du DCG de 1 à k, c'est-à-dire que l'on ne s'intéresse qu'au k premières valeurs

FIGURE 12 – Erreur d'évaluation

Pour un ensemble de n données, le ranking attribue à chaque donnée son rang lorsque la base est classée par ordre décroissant. Cet ensemble est donc transformé en une permutation de l'ensemble $\{1..n\}$. Comparer $f(X)$ et Y en ranking permet donc de savoir si f attribue à X des valeurs qui sont classées dans le même ordre que Y. Cette manière de mesurer l'erreur peut être intéressante dans notre cas, puisque l'on cherche également les X qui engendrent le maximum de l'amplification du run-up (donc les $f(X)$ qui obtiennent le classement 1 en ranking). De bonnes performances sont reflétées par un DCG élevé.

Lorsque l'on n'a pas assez de données, la validation et le test ne sont pas représentatifs. On effectue alors une validation croisée, qui permet de mélanger les sous-ensembles de données, de façon à ce que chacun participe tour à tour à l'apprentissage, la validation et le test, et à faire une moyenne des erreurs obtenues. Nous avons utilisé deux types de cross validation : le 5-fold et le leave-one-out.

Dans le cas (plus général que le 5-fold) du k-fold, on divise l'échantillon en k sous-ensembles et on les répartit de la façon suivante :

60% pour l'apprentissage : par exemple dans le modèle de régression Ridge, on détermine le bon θ en fonction des λ.

20% pour la validation : elle permet, dans la régression Ridge de déterminer le λ qui minimise l'erreur, ou bien le degré du polynôme utilisé dans le Kernel.

20% pour le test : il permet d'évaluer l'erreur entre les données et ce que l'on obtient avec les paramètres déterminés.

On répète la cross validation k fois, afin que toutes les observations soient utilisées dans

l'apprentissage, la validation et le test.

Le cas du leave-one-out est similaire, mais on ne garde qu'un point pour la validation et un pour le test. On répète donc la cross validation autant de fois que l'on a d'éléments dans la base de données. Cette cross validation peut donc prendre plus de temps computationnel, mais elle peut être plus efficace lorsque l'on a une nombre faible de données.

4.3 Résultats pour l'apprentissage passif

On a effectué de l'apprentissage passif sur la base des 200 points obtenus avec Volna. On a seulement testé la méthode de régression Ridge, sans Kernel ou bien avec Kernel polynômial non homogène ou Gaussien. Les deux méthodes de cross validation utilisées pour cette expérience sont la 5-fold et la leave-one-out. On évalue l'erreur avec l'erreur quadratique moyenne. Les résultats obtenus sont les suivants :

Méthode \ Cross Validation	5-fold	Leave-one-out
Sans Kernel	0.0988	0.1198
Kernel polynômial non homogène	0.0086	0.0164
Kernel Gaussien	0.0081	0.0148

Figure 13 – Tableau récapitulatif des erreurs obtenues sur la base des 200 points, en régression Ridge.

La méthode de Régression Ridge avec Kernel Gaussien semble être la plus efficace dans notre cas ; cependant, il y a possibilité de sur-apprentissage ici puisque le Kernel Gaussien peut à peu près modéliser toutes les fonctions, et on n'a que très peu de points descriptifs de la fonction recherchée.

4.4 Résultats pour l'apprentissage actif

4.4.1 Classification

Les expériences ont été effectuées sur deux bases différentes : la base "Tsunamis" qui contient 200 points obtenues à partir de Volna en début de stage, décrits sur 5 dimensions, et la base "Gaussienne", fortement non convexe et contenant 1000 points décrits sur 2 dimensions. Ces expériences ont pour but de nous donner une idée des performances des algorithmes d'apprentissage actif par rapport à l'apprentissage passif, mais également de comparer entre eux ces algorithmes d'actif.

Les valeurs cibles de ces deux bases sont réelles, donc il faudrait effectuer une régression dans ces deux cas. Cependant, comme on l'a vu dans la section (**??**), les algorithmes existants ne sont applicables quand dans des cas de classification binaire. Afin de transformer la régression en classification, nous avons donc seuillé les valeurs cibles de la manière suivante : pour la base "Tsunamis" (respectivement "Gaussienne"), le seuil est placé afin d'avoir 85% (resp. 90%) d'exemples négatifs et 15% (resp. 10%) d'exemples positifs.

L'ensemble des valeurs de chacun des bases est alors divisé en deux parties, une qui permettra l'apprentissage, et l'autre qui servira au test. Cette répartition est effectuée de la manière suivante : pour la base "Tsunamis" (respectivement "Gaussienne"), nous avons utilisé 100 points (resp. 750) pour l'apprentissage et les 100 autres pour le test (resp. 250). Comme on s'intéresse aux performances des algorithmes d'apprentissage actif avec un nombre de paramètres variables minimum, nous avons au préalable fixé le terme de régularisation λ de la régression Ridge ainsi que les paramètres des noyaux gaussiens par validation croisée (**??**).

La procédure expérimentale pour chacun des algorithmes est la suivante. Tout d'abord, nous initialisons un modèle d'apprentissage avec un certain nombre d'exemples (20 pour chaque base).

Le reste de la base d'apprentissage est considéré comme non étiqueté (il reste donc 80 points pour la base "Tsunamis" et 730 points pour la base "Gaussienne") et sera étiqueté point par point dans le temps par l'algorithme. Celui-ci étiquettera chaque point, soit en demandant son étiquette (par l'intermédiaire de Volna pour la base "Tsunamis"), soit en l'inférant (grâce aux points connus jusqu'au moment de l'étiquetage. Le but ici est d'avoir un nombre de demandes d'étiquettes minimum faites par l'algorithme.

Comme les performances de l'algorithme dépendent des bases de test et d'apprentissage, l'expérience est réitérée 250 fois et l'erreur est moyennée sur ces expériences. Les modèles d'apprentissage utilisés sont des régressions Ridge dont la sortie a été seuillée. Nous comparons trois algorithmes actifs (Cal, Dhm et Iwal) avec l'algorithme passif (Témoin). Les performances sont évaluées en termes d'Erreur de classification (pondérée pour rééquilibrer les classes, afin d'avoir une erreur de 50% pour une sélection aléatoire), NDCG et NDCG@10 et sont moyennées sur l'ensemble des bases de test. Nous indiquons le nombre d'exemples demandés lors de l'apprentissage par chaque algorithme afin de mieux appréhender leurs performances intrinsèques. Nous reportons dans les Tables **??** et **??** les résultats obtenus sur les deux bases étudiées.

Algorithme	Erreur	NDCG	NDCG@10	Nb de points demandés
Témoin	0.374 (0.10)	0.756 (0.08)	0.505 (0.17)	100%
Cal	0.398 (0.11)	0.663 (0.09)	0.393 (0.16)	37.5% (0.05%)
Dhm	0.394 (0.11)	0.721 (0.08)	0.465 (0.16)	37.5% (0.05%)
Iwal	0.382 (0.11)	0.695 (0.10)	0.407 (0.18)	23.62% (0.03%)

Table 1 – Tableau des résultats sur la base de "Tsunamis" convertie en un problème de classification. Le seuil a été positionné respectant 85% de négatifs et 15% de positifs. Entre parenthèses, l'écart type.

Algorithme	Erreur	NDCG	NDCG@10	Nb de points demandés
Témoin	0.307 (0.08)	0.807 (0.05)	0.685 (0.14)	100%
Cal	0.400 (0.11)	0.645 (0.09)	0.432 (0.18)	11.6% (0.01%)
Dhm	0.343 (0.11)	0.706 (0.10)	0.422 (0.17)	9.96% (0.06%)
Iwal	0.332 (0.10)	0.724 (0.11)	0.465 (0.17)	8.3% (0.04%)

Table 2 – Tableau des résultats sur la base "Gaussienne" convertie en un problème de classification. Le seuil a été positionné en respectant 90% de négatifs et 10% de positifs. Entre parenthèses, l'écart type.

Globalement, on peut donc remarquer que pour des performances relativement égales, les algorithmes d'apprentissage actif permettent un gain de temps assez important. Ceci confirme donc que l'apprentissage actif peut nous permettre de faire des économies en terme de temps computationnel.

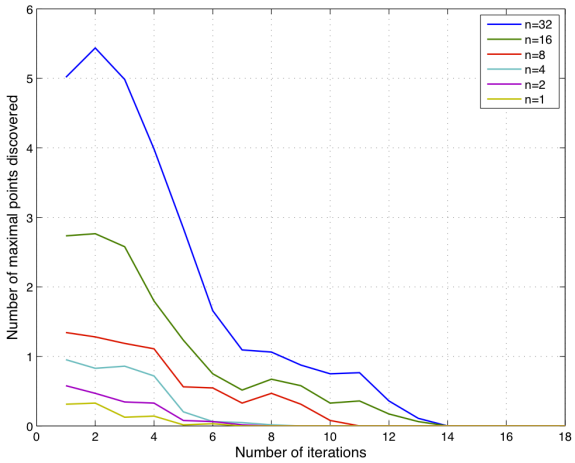
Pour les algorithmes d'apprentissage actif en eux-mêmes, on remarque qu'Iwal est plus performant que Dhm, lui même qui semble un peu plus efficace que Cal, ce qui illustre bien les améliorations successives de ces algorithmes depuis Cal. On sait de plus que par rapport à Dhm, Iwal permet d'éviter le biais d'échantillonnage, qui semble donc avoir une incidence assez importante sur les données des deux bases.

Les expériences menées nous permettent également de faire la remarque suivante : les performances des algorithmes d'apprentissage actif dépendent du paramètre λ de régularisation. En effet, plus λ est faible, plus l'algorithme va demander les étiquettes des points inconnus (à cause du sur-apprentissage). Dans nos expériences, nous avions préalablement fixé le terme λ, mais il faudrait qu'il soit fixé dynamiquement, en fonction du nombres d'exemples contenus dans la base d'apprentissage : on devrait attribuer à λ une valeur assez grande ($\lambda > 10$) lors des

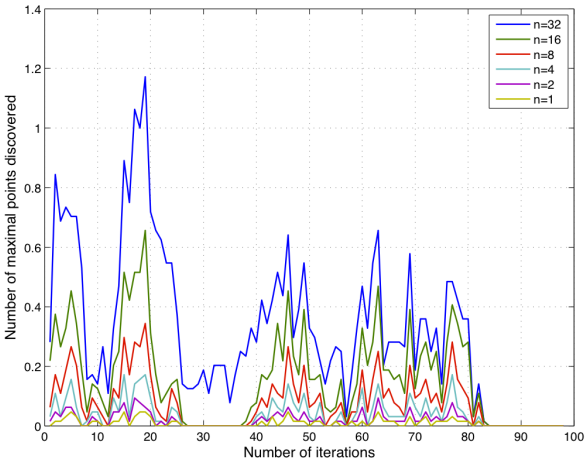
premières itérations, et le faire tendre vers des valeurs plus petites ($\lambda < 0.01$) quand un nombre conséquent (environ plusieurs milliers) d'exemples sont dans la base d'apprentissage.

4.4.2 Régression

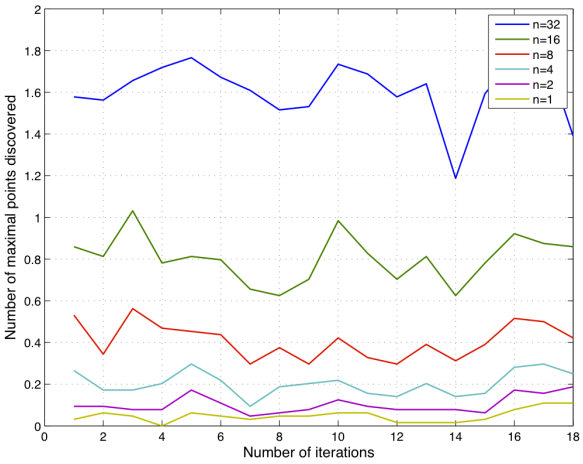
L'ensemble des expériences effectuées pour l'apprentissage actif en régression, ont principalement été réalisées afin de définir quel algorithme de sélection utiliser afin d'optimiser notre demande de point à Volna. On a donc travaillé sur différentes bases dont deux très différentes : la base dite "Tsunamis", soit 200 points obtenus à partir de Volna en début de stage, et la base "Gaussienne" plus délicate à étudier car fortement non-convexe.
Différents algorithmes de sélection de points ont alors été testés : Argmax, Random, MaxLipschitz. La Figure **??** permet d'observer à quelle vitesse ces différentes demandes de points nous permettent de trouver les n premiers maxima des bases respectivement étudiées. Sur chacune des sous-figures de la figure **??**, chaque couleur de courbe définie un nombre spécifique de maxima découvert au cours des itérations.
Comme on peut aisément le constater, l'algorithme qui consiste à demander aléatoirement des points à Volna ne permet pas d'obtenir un quelconque maximum, et ce, peu importe la base étudiée. Il est intéressant de remarquer cependant les différences entre les deux algorithmes Argmax et MaxLipschitz. Il ressort de ces expériences que l'algorithme MaxLipschitz obtient une meilleur convergence en général, et ce tout particulièrement sur une base compliquée telle que la base "Gaussienne". Cependant l'algorithme Argmax est également intéressant à considérer. En effet, Argmax, qui peut se définir comme un algorithme d'exploitation, obtient de meilleurs résultats de convergence sur une base telle que "Tsunamis" où le maximum est facilement repérable.
Néanmoins la différence d'efficacité entre les deux algorithmes n'étant que légère sur la base "Tsunamis", nous avons décidé d'utiliser de manière générale l'algorithme MaxLipschitz.



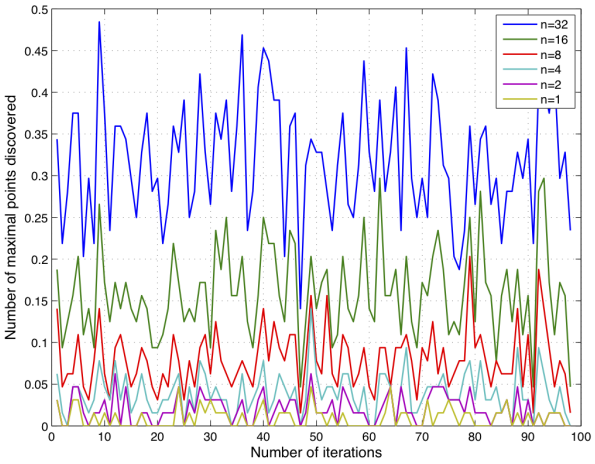
(a) Base : Tsunamis ; Algorithme de sélection :ARGMAX



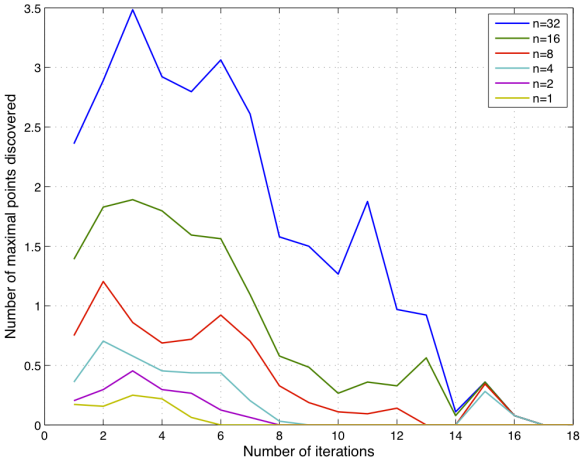
(b) Base : Gaussienne ; Algorithme de selection :ARGMAX



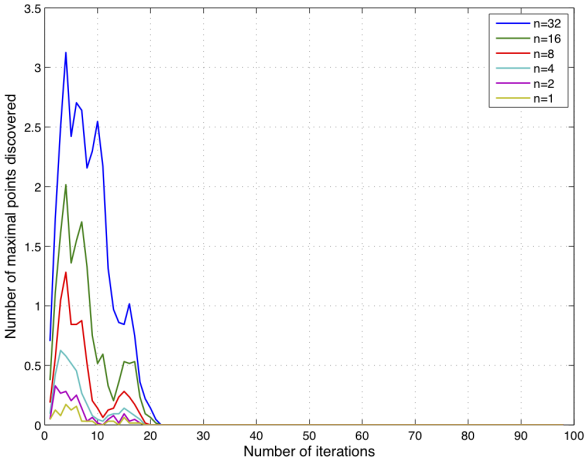
(c) Base : Tsunamis ; Algorithme de selection : RANDOM



(d) Base : Gaussienne ; Algorithme de selection : RANDOM



(e) Base : Tsunamis ; Algorithme de selection : MAXLIPSCHITZ



(f) Base : Gaussienne ; Algorithme de selection : MAXLIPSCHITZ

FIGURE 14 – Convergence de ACTIVE REGRESION

5 Conclusion

La simulation de tsunamis est aujourd'hui un problème important que l'on peut résoudre. Effectuées à l'aide du logiciel Volna, ces simulations sont néanmoins extrêmement couteuses d'un point de vue computationnel. Pouvant prendre jusqu'à plusieurs semaines, il était nécessaire de trouver une solution afin de limiter le travail de Volna. Le but principal de ce stage fut donc de rechercher dans le domaine du machine learning différentes alternatives afin d'obtenir les maxima d'amplification du run-up d'un tsunami tout en limitant la sollicitation de Volna.

Notre but ne fut donc pas de chercher à établir une fonction permettant de connaître l'ensemble des données, mais de pouvoir définir l'ensemble des maxima pour l'amplification du run-up en fonction des 5 paramètres d'entrée. Nous avons donc dans un premier temps étudié les différentes méthodes d'apprentissage afin d'identifier le meilleur modèle mathématique nous permettant d'approximer Volna. Nous nous sommes ensuite lancé dans un travail plus informatique, soit l'adaptation de ces différents modèles dans un cadre d'apprentissage passif.

Il fut cependant rapidement évident que l'apprentissage passif ne pouvait constituer une solution viable à notre problème. Nous avons donc orienté nos recherches vers l'apprentissage actif. Étant un domaine de recherche relativement récent, la plupart des algorithmes d'apprentissages actifs concernent les problèmes de classification. Nous avons donc commencé l'étude de différents algorithmes de classification active tels que Cal ou Dhm, dans l'espoir premier de pouvoir ramener notre problème d'amplification à un problème de classification. Parallèlement à cette étude, nous avons décider de créer un algorithme actif en régression, qui pourrait ainsi s'adapter beaucoup plus naturellement à notre recherche de maxima d'amplitude. Étant cependant confrontés à un manque cruel de documents scientifiques dans ce domaine, nous avons tenté de marier l'apprentissage actif à la régression en combinant une demande de point active à une régression Ridge. Il nous a également fallu définir un critère d'arrêt qui se focalise alors sur la capacité de notre régression à attendre une solution stable.

Bien que ce dernier algorithme de "régression active" nous permette d'obtenir des résultats très prometteurs, il est dénué de toute preuve formel. Contrairement à la "régression active", les différents algorithmes de classification sont quant à eux prouvés. Deux objectifs peuvent ainsi se présenter : démontrer formellement et optimiser les algorithmes de "régression active", ou trouver une solution afin de ramener notre problème de maxima à un problème de classification exploitable par les différents algorithmes étudiés.