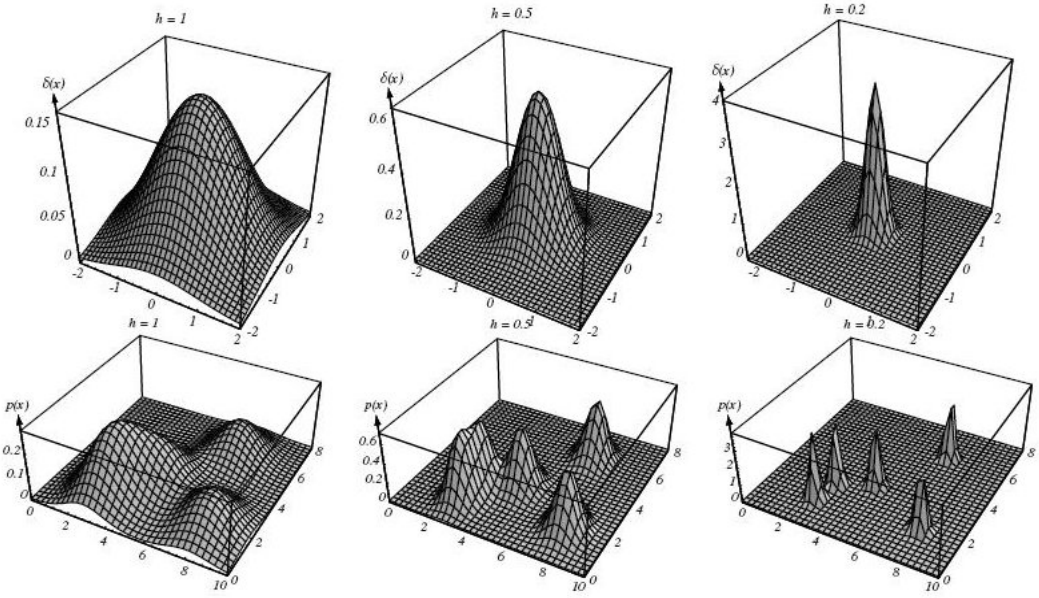
Maxime Chesnel - 4ème année Électronique et Informatique Industrielle

RAPPORT DE STAGE

Développement d'une expérimentation pour la recherche

Détermination du paramètre d'une fenêtre de Parzen dans le cadre de l'apprentissage actif



Maître de stage :
M. Lemaire Vincent
France Telecom R&D

Remerciements :

Je remercie chaleureusement Vincent Lemaire de m'avoir proposé et d'avoir encadré ce stage, qui s'inscrit dans le contexte de la recherche réalisée par le laboratoire TSI (Traitement statistique de l'information) dans le centre de Recherche et Développement de France Telecom à Lannion. Je veux également remercier Alexis Bondu pour sa participation active dans ce projet. Je n'oublie pas non plus l'aide efficace des autres collègues en particulier Pascal Gouzien et Carine Hue.

SOMMAIRE :

Introduction

Les expériences et les enseignements de ce stage appartiennent au domaine de la statistique décisionnelle ou apprentissage statistique. C'est une partie de la statistique dont la finalité est de prendre des décisions. A partir de bases de données, on prédit la valeur de variables non observées. On parle plutôt 'd'apprentissage' statistique. En effet, la statistique décisionnelle est d'un intérêt majeur pour les recherches et développements en intelligence artificielle. Elle permet la reproduction d'un apprentissage humain par un apprentissage artificiel. L'apprentissage statistique comporte de nombreux algorithmes et voies d'apprentissage dont certaines seront vues dans ce rapport.

Il est important d'introduire également le 'data mining' (fouille de données) qui serait l'un des dix grands enjeux du XXIe siècle selon la revue scientifique MIT Technological. Il s'agit d'une application de l'apprentissage statistique. Sa vocation est d'exploiter des bases de données afin d'en extraire des connaissances à usages professionnels. Les bases de données des entreprises comportent un général un très grand nombre de données. A la différence de l'apprentissage statistique, aucune hypothèse n'est faite sur les données. Ainsi, le logiciel de data mining doit déterminer lui même les corrélations et caractéristiques intéressantes des données qu'il explore.

Le stage, et ses objectifs se situent à la fois dans ces deux domaines. En effet, il concerne l'apprentissage actif, une voie d'apprentissage statistique que l'on veut munir d'algorithmes d'apprentissages en les évaluant sur un grand nombre de données. L'apprentissage actif a été formalisé en 1992 par des chercheurs américains. Il consiste à déterminer les données les plus instructives pour l'apprentissage. Dans un premier temps, nous verrons l'apprentissage actif, le

cadre de ce stage et la fenêtre de Parzen, outil qui nous fournira plusieurs algorithmes d'apprentissages statistiques. Nous aborderons ensuite la conception des procédures expérimentales et les choix réalisés. Nous terminons par les justifications faites sur chaque algorithme et les résultats obtenus.

Χηαπιτρε Ι– Fenêtre de Parzen dans le cadre de l'apprentissage Actif

I.1.Apprentissages et Apprentissage Actif

Les méthodes d'apprentissage exploitent la base d'apprentissage pour produire des règles (prédictions d'exemples, valeurs de certains paramètres etc.). Il existe plusieurs 'modes' d'apprentissages. Certains sont automatiques et passif, les apprentissages ne nécessitent alors pas l'intervention d'un opérateur. Ces stratégies s'opposent à l'apprentissage actif où l'opérateur intervient de manière optimale dans le processus d'apprentissage. Les apprentissages passifs peuvent être soit supervisés soit non supervisés.

I.1.1Apprentissage supervisé

En apprentissage supervisé, la base d'apprentissage contient des exemples de données déjà traitées. On supervise l'apprentissage en exploitant ces exemples pour en apprendre de nouveaux. La base d'apprentissage est dans ce cas de figure un ensemble de *N* couples entrées-sorties:

$$(x_n, y_n)_{1 \leq n \leq N}$$

Un algorithme d'apprentissage supervisé a pour but de généraliser sur les nouvelles entrées ce qu'il a appris des couples entrées-sorties fournis par la base. Dans notre cas, les entrées $x_n \in X$, sont toutes contenues dans un même hyperespace. Chaque élément d'entrée est un vecteur appelé 'instance'. Les sorties quant à elles, appartiennent à l'espace des réels $Y \subset \mathbb{R}$ (le couple est alors un couple de valeurs explicatives et valeurs cibles) ou aux entiers naturels $Y = \{1, \ldots, I\}$ (les sorties sont appelées des classes).

I.1.2Apprentissage non supervisé

En apprentissage non supervisé et contrairement à l'apprentissage supervisé (cf. sous-section précédente), il n'y a pas de 'sorties' au sens des couples entrées-sorties vu précédemment. L'apprentissage non supervisé consiste donc à trouver un autre moyen d'extraire des règles.

I.1.3 Apprentissage Actif

L'apprentissage actif est une méthode d'apprentissage statistique qui nécessite l'intervention d'un opérateur expert : 'l'oracle'. Cette méthode a pour vocation de mieux s'apparenter à l'apprentissage humain en optimisant la rapidité de l'apprentissage.

L'école active

A la manière d'autres méthodes de l'intelligence artificielle telles que les réseaux de neurones, l'apprentissage actif simule un moyen d'apprentissage humain ou naturel. L'idée vient du pédagogue suisse Adolphe Ferrière au XXe siècle. Il a énoncé le concept d'une école active où on peut faciliter l'apprentissage des enfants. Pour cela, on leur fait choisir judicieusement des éléments de leur vécu puis en leur enseigne des connaissances associées. Les élèves sont ainsi amenés à créer leurs propres connaissances de manière participative donc active.

L'Apprentissage Actif

L'apprentissage statistique actif est à l'image de l'école active. L'enfant est simulé par la partie automatique qui recherche parmi les exemples à traiter, les plus judicieux pour son apprentissage. Le professeur est représenté par l'oracle. Il fournit à la procédure automatique, les sorties des exemples à traiter qu'elle lui a demandé.

Contrairement aux autres méthodes d'apprentissage, il y a une interaction entre le modèle et son environnement. C'est une stratégie 'active' par opposition aux stratégies 'passives' où tous les exemples à apprendre sont choisis avant l'expérience.

Mais avant d'en venir à une méthode d'apprentissage actif, il se pose le problème de l'échantillonnage des données.

Échantillonnages et Échantillonnage Sélectif

Un échantillonnage de données est la sélection d'un ensemble de données extrait d'un ensemble plus grand. Dans le cas de l'apprentissage actif, l'échantillonnage peut prendre deux formes que sont l'échantillonnage sélectif et adaptatif.

L'échantillonnage sélectif est comme son nom l'indique un échantillonnage qui se contente d'une sélection parmi les données possédées. Ces données peuvent avoir une forme brute (image, son, etc.) ou peuvent être représentées par des descripteurs, des vecteurs de données. Dans le cas sélectif, on est certain que les données sélectionnées existent puisqu'elles proviennent directement des données.

L'échantillonnage adaptatif permet quant à lui l'exploitation entière de l'existence des instances de données ou descripteurs. En effet, les descripteurs forment un espace qui contient toutes les données réelles. Il est donc possible de s'interroger sur des données virtuelles crées par une variation des descripteurs ne correspondant à aucune données brutes. L'intérêt d'une telle pratique, est lorsque l'on possède peu de données et qu'on accepte de s'interroger sur n'importe de fausses données. Elle possède l'inconvénient d'inventer des données qui ne représentent plus la réalité ce qui va poser des problèmes pour leur(s) traitement(s). Par exemple, il peut être impossible de classifier de telles données.

Notre cas est celui du Groupe France Telecom où les données clients sont très nombreuses et par conséquent, nous n'avons pas besoin d'en inventer. L'échantillonnage est donc sélectif.

Le lecteur pourra se reporter avantageusement à l'Etat de l'art réalisé sur les méthodes statistiques d'apprentissage actif [Bondu et .al] pour apprendre sur les possibilités d'échantillonner et de traiter des données dans le cas de l'apprentissage actif.

Un modèle d'apprentissage actif par échantillonnage sélectif

L'état de l'art [Bondu et .al] nous apporte l'algorithme d'un modèle d'apprentissage actif par échantillonnage sélectif qui a été formulé par Muslea [Muslea]. Comme le dit cette état de l'art, cet algorithme met en jeu une fonction d'utilité, Utile(u,M), qui estime l'intérêt d'une instance u pour l'apprentissage du modèle M. Grâce à cette fonction, le modèle présente à l'oracle les instances pour lesquelles il espère la plus grande amélioration de ses performances. Ci-dessous l'algorithme présenté dans l'état de l'art.

Étant donnés :

- M un modèle prédictif muni d'un algorithme d'apprentissage L
- Les ensembles *Ux* et *Lx* d'exemples non étiquetés et étiquetés
- n le nombre d'exemples d'apprentissage souhaité.
- L'ensemble d'apprentissage *T* avec $||T|| < n$
- La fonction Utile : $X \times M \rightarrow \Re$ qui estime l'utilité d'une instance pour l'apprentissage d'un modèle.

Répéter

(A) Entraîner le modèle M grâce à *L* et *T* (et éventuellement *Ux*).

(B) Rechercher l'instance $q = \text{argmax}_{u \in Ux}$ Utile(u,M)

(C) Retirer q de *Ux* et demander l'étiquette f(q) à l'oracle.

(D) Ajouter q à *Lx* et ajouter (q, f(q)) à *T*

Algorithme 1 : Apprentissage Actif selon Muslea

L'apprentissage Actif dans cette étude

L'apprentissage actif est un axe contemporain d'exploration de la recherche. Le projet que je réalise contribue à cette recherche en lui apportant les connaissances d'un outil statistique: la fenêtre de Parzen. Les expérimentations de cet outil, au lieu de se faire dans les conditions contraignantes de l'apprentissage actif, se feront plutôt et d'abord en apprentissage passif afin de comparer sur de grands jeux de données les performances des différentes méthodes. Le projet consiste donc à déterminer notre fenêtre de Parzen dans un apprentissage supervisé ou non supervisé. Comme nous verrons que notre fenêtre de Parzen ne possède qu'un seul paramètre, sa détermination est finalement le but en pratique de nos expérimentations d'apprentissages sur les données.

I.2.Modèles prédictifs et Fenêtres de Parzen

I.2.1Qu'est ce qu'un modèle prédictif

Principe des prédictions sur les données

Un modèle prédictif est un outil. Sa finalité est de réaliser les prédictions nécessaires à un ou des apprentissages. Ainsi, le modèle forme un noyau pour un algorithme d'apprentissage. Le modèle est déterminé par l'expérience qui est faite et peut dépendre du jeu de données utilisées comme exemple. Ainsi si un modèle prédictif est déterminé par une méthode de prédiction, chaque expérience peut définir un modèle différent pour chaque jeu de données. Le modèle prédictif doit pouvoir donner une réponse au problème à traiter et également estimer la fiabilité de sa réponse.

Schéma de principe

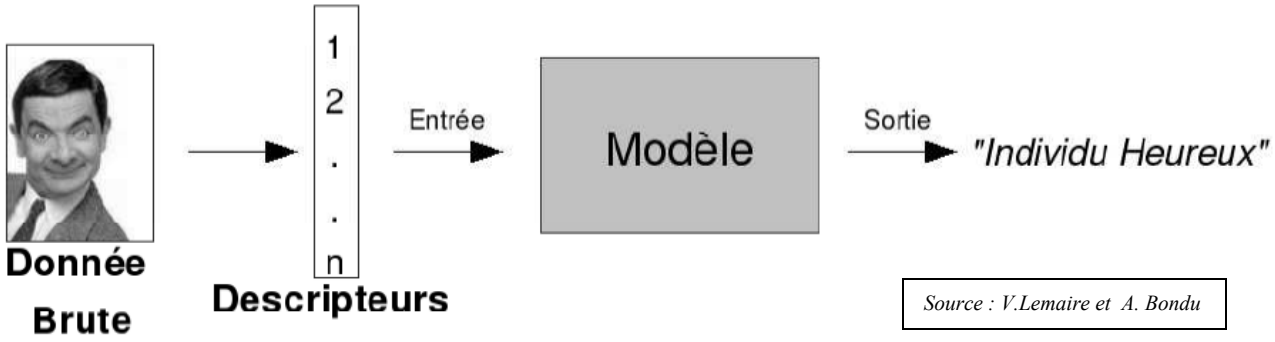


Figure 1 : Chaîne de traitement d'un modèle prédictif

Comme la montre la chaîne de traitement d'un modèle prédictif, le modèle intervient à la suite d'un échantillonnage rassemblant un certain nombre d'entrées et permet une prédiction en sortie. Beaucoup de modèles possèdent un ou plusieurs paramètres qui vont permettre le fonctionnement puis l'optimisation du modèle. Certaines méthodes de prédictions utilisent des exemples d'apprentissages pour leurs modèles. Ainsi le modèle de prédiction est fonction du réglage et des exemples d'apprentissage comme dans le schéma ci-dessous :

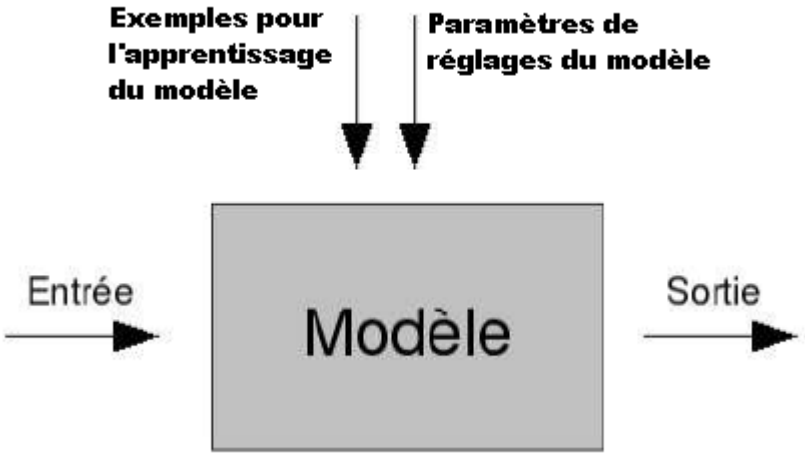


Figure 2 : Le modèle prédictif

I.2.2Un modèle prédictif : la fenêtre de Parzen

La fenêtre de Parzen est à la fois un outil statistique et un modèle prédictif muni d'un paramètre. Mais le fonctionnement et la détermination de la fenêtre se basent sur des exemples de données qu'on lui propose. Ainsi, à un jeu de données correspond un modèle, alors que la méthode de prédiction reste la même.

Concept de Fenêtre de Parzen

La fenêtre de Parzen est une méthode de prédiction suggérée par M. Parzen. Elle s'intitule fenêtre car en effet elle propose de ne regarder dans un espace de données que les éléments situés dans un hyper-volume autour de l'élément à prédire. Ce volume serait par conséquent un cadre ou une fenêtre dans un espace à deux dimensions.

La fenêtre de Parzen permet un apprentissage par voisinage et à noyau. Elle est donc proche de la méthode des k-ppv (les k plus proches voisins – k-nn K-Nearest-Neighbors en Anglais). Elles permettent toutes les deux, de réaliser une prédiction quelconque sur un élément en prenant en compte les éléments (les instances) dont la proximité sera jugée suffisante. La différence réside dans le fait que les k plus proches voisins définissent à partir de données, une fenêtre qui les encercle tandis que la fenêtre de Parzen est une fenêtre que l'on définit avant de considérer les proches voisins situés à l'intérieur.

La fenêtre de Parzen est munie d'un algorithme qui dépendra du contexte considéré (classification, régression, estimation de densité...). On utilise un poids fournit par l'algorithme de Nadaraya-Watson (NW) formulée simultanément et indépendamment par Nadaraya et Watson en 1964. Ce poids sera utilisé différemment en fonction qu'il s'agit d'un problème de classification ou de régression. Mais dans tous les cas, il s'agira de sommer les poids afin d'obtenir une prédiction probable en sortie.

$$W_i(x) = \frac{K(x, x_i)}{\sum_{\ell=1}^{n} K(x, x_\ell)}$$

Équation 1 : Poids de l'estimateur de Nadaraya-Watson (1964)

Remarque : On utilisera toujours la convention 0 / 0 = 0 avec cet estimateur.

Voici en résumé, un schéma récapitulatif explicitant le modèle prédictif de la fenêtre de Parzen.

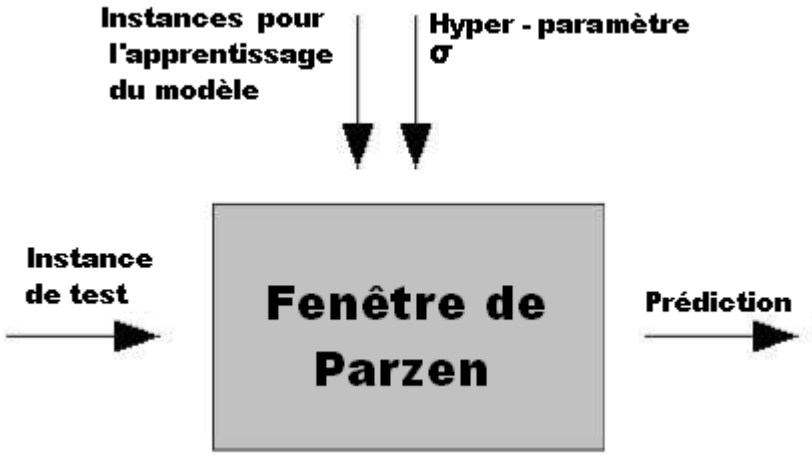


Figure 3 : Le modèle prédictif de la fenêtre de Parzen

Concept du noyau

Un noyau est une fonction scalaire de deux variables (des vecteurs). En effet, à partir de deux variables, il fournit une valeur sans fournir autre chose (par définition d'une fonction scalaire). Dans un algorithme, un noyau peut être placé là où ou se trouve un produit scalaire ou une distance euclidienne. Son but est de traiter les instances de données de manière non linéaire et prendre en compte la non linéarité des données [Louradour p.27].

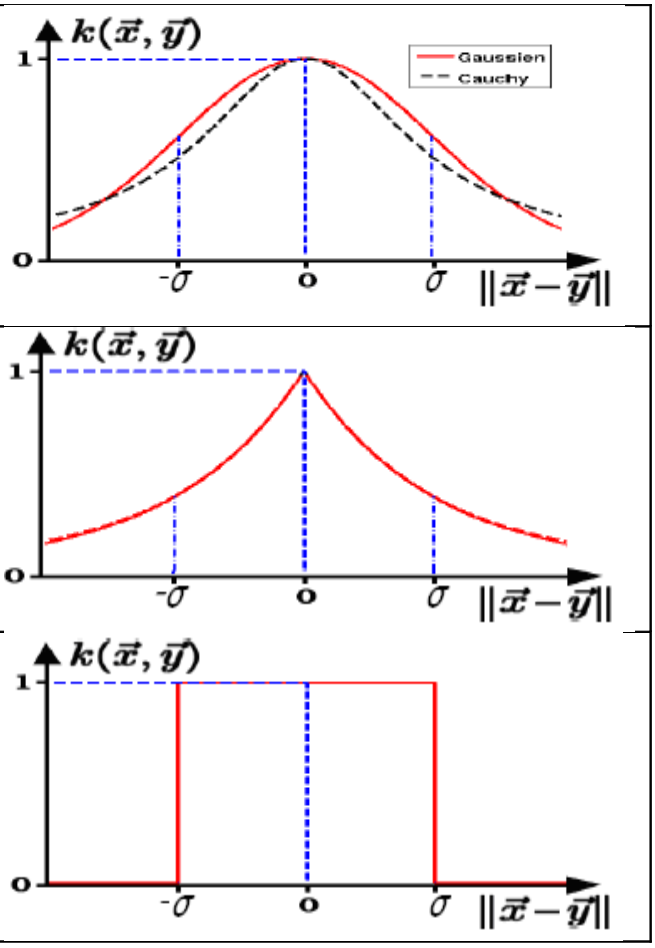
Concept du noyau radial

Il existe deux types de noyau, le noyau projectif et radial (ou métrique). Le noyau projectif se base sur le produit scalaire de deux vecteurs. Le noyau radial s'appuie sur le produit scalaire par lui-même de la différence de deux vecteurs, soit la distance euclidienne au carré ou norme L2 au carré.

Notre objectif est d'obtenir une mesure de distance entre deux instances de données à l'aide d'un noyau non linéaire. Le noyau que l'on doit utiliser est donc un noyau métrique ou radial. Le calcul du carré de la distance euclidienne sera donc un intermédiaire de calcul pour la détermination des noyaux.

Ci-dessous, quelques exemples de noyaux radiaux sont illustrés (pour tout σ positif)

Tableau 1 : Illustration de noyaux radiaux

Noyau Gaussien	$k(\mathrm{x},\mathrm{y}) = e^{-\frac{\|\mathrm{x}-\mathrm{y}\|^2}{2\sigma^2}}$	
Noyau de Cauchy	$k(\mathrm{x},\mathrm{y}) = \frac{1}{1+\frac{\|\mathrm{x}-\mathrm{y}\|^2}{\sigma^2}}$	
Noyau de Laplace	$k(\mathrm{x},\mathrm{y}) = e^{-\frac{\|\mathrm{x}-\mathrm{y}\|}{\sigma}}$	
Noyau radial uniforme	$k(\mathrm{x},\mathrm{y}) = 1_{\|\mathrm{x}-\mathrm{y}\|\leq\sigma}$	

Extension de la fenêtre à l'infini: le noyau gaussien

Pour la définition de la fenêtre de Parzen, on doit faire le choix d'un 'noyau', il en existe une grande variété. Notre choix a été fait sur un noyau particulier, le noyau gaussien L2. Il s'agit d'une extension de la fenêtre de Parzen qui donne un poids à tous les éléments de l'espace de données mais ce poids tend très rapidement vers 0 lorsque l'on s'éloigne du point d'où on considère la fenêtre. L2 désigne la norme L2 qui permet d'exploiter la distance euclidienne entre 2 points

Si les points *A* et *B* ont pour coordonnées respectives $(x_A; y_A; z_A)$ et $(x_B; y_B; z_B)$, alors leur distance euclidienne est :

$$\|\overrightarrow{AB}\| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

Équation 2 : Distance euclidienne d'un vecteur à 3 dimensions

.

Le carré de la distance euclidienne est le produit scalaire de la différence de deux vecteurs. Le Noyau Gaussien utilise ce carré de la distance euclidienne, il est donc un noyau radial.

Il n'est par contre pas évident que la forme du noyau gaussien ne possède qu'un seul paramètre comme la formule du tableau 1. En réalité, notre noyau est inclus dans un espace du même nombre de dimensions que les données que ce noyau utilise. Donc il correspond un paramètre pour toutes les dimensions d'un vecteur d'entrées. Dans le cas général et pour toutes les bases génératrices de l'espace vectoriel correspondant à l'espace des données, on peut définir une matrice de covariance qui définit un noyau gaussien "multi variée". Ci-dessous figure une écriture du noyau gaussien multi variée. Σ est la matrice de covariance et |Σ| son déterminant. La formule comporte également un terme de normalisation. Ainsi dans R^n :

$$K_\Sigma(x - x_i) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - x_i)^T \Sigma^{-1}(x - x_i)\right)$$

Équation 3 : Noyau Gaussien L2 multi variée (avec son terme de normalisation)

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_2\sigma_1 & \cdots & \cdots & \sigma_n\sigma_1 \\ \sigma_1\sigma_2 & \sigma_2^2 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \sigma_{n-1}^2 & \sigma_n\sigma_{n-1} \\ \sigma_1\sigma_n & \cdots & \cdots & \sigma_{n-1}\sigma_n & \sigma_n^2 \end{pmatrix}$$

Équation 4 : Matrice de covariance du noyau gaussien multi-variée

La diagonalisation de la matrice doit permettre d'obtenir une matrice diagonale dans cette base et d'interpréter une matrice de covariance dans ce cas le plus générale Le cas où la matrice est diagonale dans la base orthonormé directe de l'espace est un cas particulier.

$$\Sigma = \begin{pmatrix} \sigma_1^2 & & \mathbf{0} \\ & \sigma_2^2 & \\ \mathbf{0} & & \ddots \\ & & & \sigma_n^2 \end{pmatrix}$$

Équation 5 : Matrice de covariance diagonale du noyau gaussien multi variée

Lorsque toutes les variances qui forment la diagonale de la matrice de covariance sont égales. Alors les propriétés du noyau gaussien sont les mêmes selon toutes les directions de l'espace. Pour cette raison, la matrice de covariance spécifique est appelée iso-tropique. Elle est également appelée sphérique car les courbes de niveau du noyau forme des hyper sphères. Ce n'est que dans ce cas très spécifiques que le noyau gaussien ne possède qu'un seul paramètre : σ.

$$\Sigma = \sigma^2 I_n$$

Équation 6 : Matrice de covariance du noyau iso-tropique définie à partir de la matrice identité

Le noyau gaussien L2 s'apparente alors à la distribution gaussienne que l'on utilise fréquemment dans des calcule de probabilités et de statistiques avec σ^2 la variance (carré de l'écart type) et μ la moyenne.

$$\mathcal{N}_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Équation 7 : Distribution Gaussienne ou Normale

Le noyau gaussien L2 peut être définit sans son terme de normalisation comme suit:

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

Équation 8 : Noyau Gaussien L2

Ci-dessous une illustration du noyau gaussien dans un espace de dimension 2 :

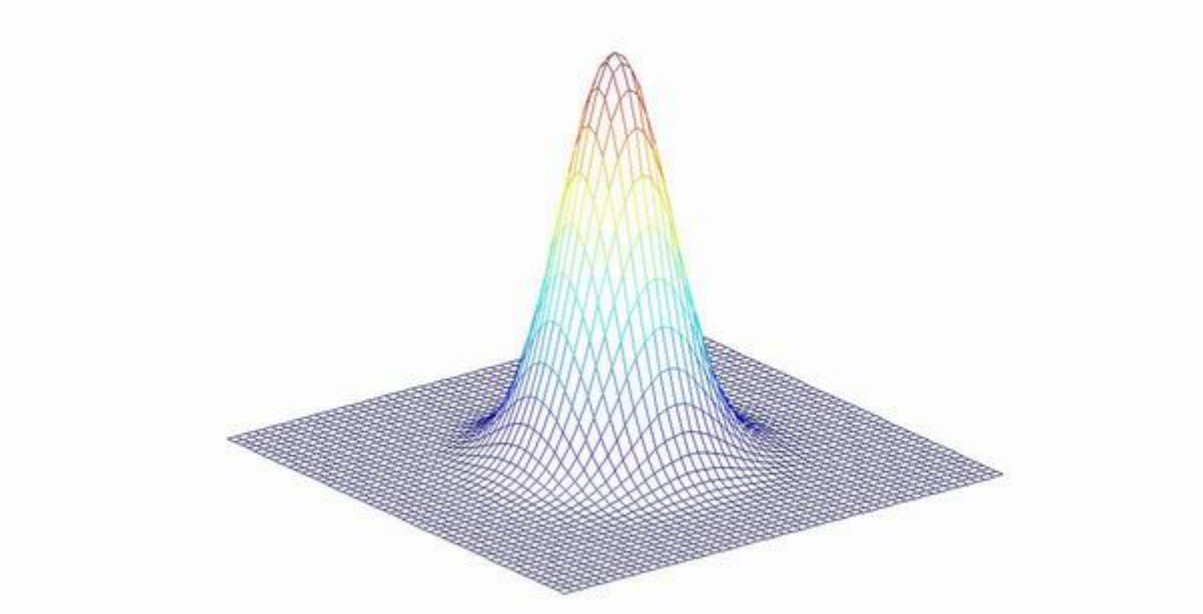


Figure 4 : Représentation d'un Noyau Gaussien L2 en 2 dimensions

Nous adoptons le noyau gaussien L2 pour toutes les expériences. Celui-ci nous crée un enjeu pour cette étude. Il s'agit de régler correctement le paramètre de ce noyau. Ce qui équivaut au bon réglage de la fenêtre de Parzen.

I.3.Le réglage d'une Fenêtre de Parzen

I.3.1L' hyper-paramètre σ

Comme nous l'avons vu, la fenêtre de Parzen utilise une brique, l'algorithme de Nadaraya-Watson, différemment selon les méthodes d'apprentissage. Cet algorithme contient un noyau gaussien. C'est ce noyau qui donne des paramètres à régler pour la fenêtre de Parzen Si on adopte le cas le plus simple du noyau iso-tropique, alors il n'y a qu'un unique paramètre de ce noyau qui est σ. Celui-ci quantifie la largeur du "dôme". C'est un hyper-paramètre puisqu'il intervient sur toutes les dimensions du noyau dans l'hyperespace considéré. Ce paramètre est positif car sinon le noyau n'aurait pas de sens "métrique". Il tendrait également vers l'infini lorsque la distance euclidienne tendrait lui aussi vers l'infini. Enfin, ce paramètre est également le paramètre unique de la fenêtre de Parzen via l'algorithme de Nadaraya-Watson. C'est donc sur lui que se porte toute la connaissance de notre fenêtre de Parzen en vue de son réglage (dans notre but d'apprentissage actif). Mais, nous pouvons d'ores et déjà, raisonner sur le rôle de σ.

Dans les méthodes d'apprentissages utilisant la fenêtre de Parzen, on considèrera une somme de noyaux gaussiens tous réglées avec le même paramètre σ et calculés dans le voisinage de l'instance de test où une prédiction est à faire.

Influence du paramètre sur l'apprentissage

- Sur - apprentissage

Lorsque que σ tend vers 0, les courbes tendent vers des pics ou Dirac. Ainsi, au voisinage d'une instance d'apprentissage, les noyaux calculés seront très grands par rapport aux autres noyaux. Ils n'auront pas d'impacts dans le cas considéré où l'on somme les noyaux. La conséquence est que seul ce noyau sera pris en compte. En dessous d'une valeur de σ petite, des noyaux sont σ à proximité sont suffisamment distancées pour ne pas être pris en compte. Si un nombre significatif de ces noyaux étaient importants également pour la prédiction de la fenêtre de Parzen. On doit observer une diminution de la performance pour les petites valeurs de σ. On dit que l'on a un phénomène de sur-apprentissage. En effet on a sur-appris les données, c'est-à-dire que l'on a apprit avec trop de précision au point de négliger de l'information. Le noyau gaussien devient ou s'approche dans ce cas à un noyau de Dirac. [Laroudour]

Dans l'exemple de classification ci-dessous, les 2 classes attribuées aux données d'apprentissage représentées par des '+' et des 'o' ont permis de tracer une frontière dans l'espace des données. Ce tracé est dans le cas du sur – apprentissage trop minutieux

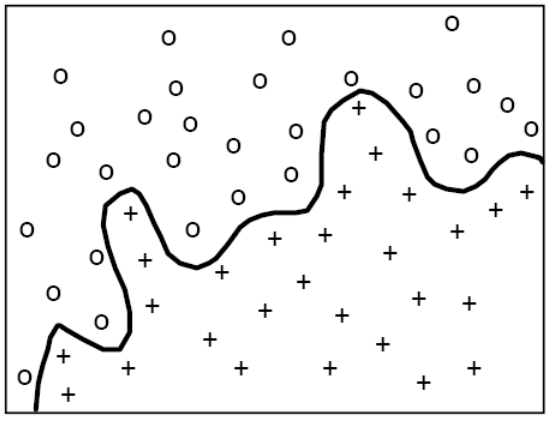


Figure 5 : Exemple de sur- apprentissage en 2 dimensions

- Lissage

Lorsque que σ tend vers ∞, on a un phénomène de lissage. Toutes les gaussiennes deviennent plates. La fenêtre de Parzen ne parvient plus à extraire de l'information à partir des distances séparant les instances dans l'hyper-espace. Beaucoup d'informations sont donc comme effacées à l'intérieur du modèle de prédiction. Par exemple, le modèle donnera toujours la classe majoritaire dans un problème de classification et la moyenne des valeurs dans un problème de régression. Le noyau gaussien devient ou s'approche dans ce cas à un noyau constant. [Laroudour]

Sur le dessin représenté plus haut

Source : www.ulb.ac.be

- Optimalité

Tandis que le lissage et le sur-apprentissage apportent normalement une diminution observable de la performance. Nos algorithmes d'apprentissages devront s'en écarter afin de trouver un bon ou le meilleur paramètre pour notre fenêtre de Parzen. Mais comme il s'agit pour nous d'améliorer nos connaissances de la fenêtre de Parzen muni du noyau gaussien. Nous allons rechercher le paramètre qui maximise les performances de la fenêtre, en fait, le σ optimal. Ceci va nous permettre de comparer plusieurs méthodes de prédictions entre elles.

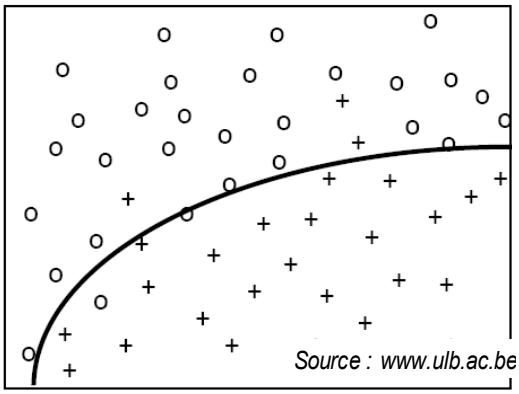


Figure 6 : Illustration d'un meilleur apprentissage en 2D

I.3.2Contexte bibliographique

Description du contexte

Le réglage d'une fenêtre de Parzen a été traité par d'autres scientifiques mais pas au point de la recherche d'une véritable d'optimalité du paramètre. En effet, en 1999, Schölkopf [Schölkopf] a proposé une solution simple pour le réglage du noyau gaussien. Ce réglage a été repris ensuite par Chapelle et par d'autres. La méthode semble marcher mais aucune justification n'a été apportée à cet état de fait. Par conséquent, on se demande si la méthode marche pour tous les jeux de données et si d'autres méthodes pourraient apporter de meilleurs résultats.

Travaux de Schölkopf

Bernhard Schölkopf chercheur à l'institut Max Planck en Allemagne a fait part de ses études sur le sujet des algorithmes à noyaux. Comme nous l'explique Jérôme Louradour dans sa thèse, Schölkopf utilise à la fois la moyenne quadratique des écarts-types de chaque composante vectorielle et la dimension des données pour régler sa fenêtre de Parzen. Le calcul de l'écart type moyen des données est le suivant :

x_u correspond aux variables d'entrées. d est la dimension du jeu de données.

$$\overline{\sigma} = \frac{1}{d}\sum_{u=1}^{d}\left(\mathrm{E}[x_u^2] - \mathrm{E}[x_u]^2\right)$$

Équation 9 : Calcul de l'écart-type des données

Le calcul du σ qui en résulte est:

$$\sigma \approx \sigma_0 = \sqrt{d}\overline{\sigma}$$

Équation 10 : Paramètre choisi par Schölkopf

D'après eux, à condition que toutes les variables d'entrées aient la même importance, on peut normaliser les données et se contenter de prendre pour σ:

$$\sigma = \sqrt{d}$$

Équation 11 : Simplification du paramètre de Schölkopf

Dans l'expérimentation du jeu de données USPS, Schölkopf utilise donc 256, la dimension du jeu de données et 0.5 un bruit gaussien qu'il a appliqué sur les données USPS.

Jeu de donnée "USPS"	
Dimension	256
Variance σ^2	$256.0{,}25 = 256.(0{,}5)^2$
Paramètre σ	$\sqrt{256}.0{,}5=16.0{,}5$

Tableau 2 : Paramètres du jeu de données USPS d'après Schölkopf

C'est pour quoi il a utilisé le paramètre σ réglé selon le tableau ci-dessus.

Travaux de Chapelle

Dans sa publication "Active Learning for Parzen Window Classifier" (Apprentissage actif pour la fenêtre de Parzen de classification), Olivier Chapelle également de l'institut Max Planck explicite l'utilisation de la fenêtre de Parzen dans un algorithme d'apprentissage actif. Son but est de déterminer la fonction qui détermine l'espérance de l'erreur de test qui s'apparente à la fonction utile vu à la sous - section Apprentissage Actif (Chap. I). Nous avons observé comment Chapelle règle la largeur de sa fenêtre de Parzen de classification. Chapelle exploite 2 jeux de données que sont le Damier (un jeu de donnée artificiel) et l'USPS (un jeu de donnée provenant d'un problème réel). Dans les 2 cas, il règle σ^2 selon la valeur de la dimension des données: comme dans le tableau suivant: Ces résultats proviennent bien des travaux de Schölkopf.

Jeu de donnée "Damier"		Jeu de données "USPS"	
Dimension	2	Dimension	256

Variance σ^2	2	Variance σ^2	$256 .(0,1)^2$
Paramètre σ	$\sqrt{2}$	Paramètre σ	$\sqrt{256} .(0,1)$

Tableau 3 : Paramètres du jeu de données USPS d'après Chapelle

I.3.3 La problématique du stage

Motivations

Les expériences réalisées dans ce projet ont pour but d'obtenir une fenêtre de Parzen paramétrée convenablement pour l'Apprentissage Actif. Le paramétrage de la fenêtre de Parzen s'est fait jusque là d'une manière simple et judicieuse certes. Mais nous avons besoin de réaliser un réglage précis dans l'optique de bien s'éloigner des mauvais réglages vus à la sous-section de l'hyper-paramètre (Chapitre I - 2). Il s'agit alors de trouver d'autres méthodes judicieuses que l'on va confronter à la méthode de Schölkopf dans le but de trouver la meilleure méthode de paramétrisation.

Problématique(s)

Pour confronter différentes méthodes d'apprentissage, nous avons besoin de critères. 2 critères vont entrer en compte: la performance ou éventuellement la non performance (l'erreur) du paramètre choisi et la capacité du modèle à généraliser son apprentissage sur d'autres données. Comme on voudrait savoir l'impact de la quantité des données sur l'apprentissage, on analysera l'impact de la variation du nombre de ces données en fonction de la méthode d'apprentissage utilisée. On regardera également les résultats sur tous les jeux de données afin de nous assurer que les performances ne s'effondrent pas sur certains jeux.

Dans cette étude, nous nous interrogerons donc sur les questions : Quelles sont les méthodes (parmi les méthodes passives vues dans cette étude) qui permettent le mieux de déterminer le σ optimal d'un jeu de données? Quelles sont les hypothèses quantitatives (quantité d'instances et de dimensions des ensembles utilisés) et qualitative (indépendance, distribution) sur les jeux de données qui ont une influence sur la détermination du paramètre? L'aboutissement de l'expérience peut être l'utilisation de certaines de ces méthodes dans de nouvelles expériences testant cette fois-ci, des méthodes d'apprentissage actif.

Sujet de l'étude

Nous allons donc réaliser des expériences dont l'objectif est d'obtenir l'optimalité de σ. Ces expériences sont 3 méthodes d'apprentissage auxquelles pourront s'ajouter par la suite d'autres méthodes. L'illustration suivante rassemble parmi de nombreux éléments vus dans ce chapitre, les 3 méthodes d'obtention du σ optimal.

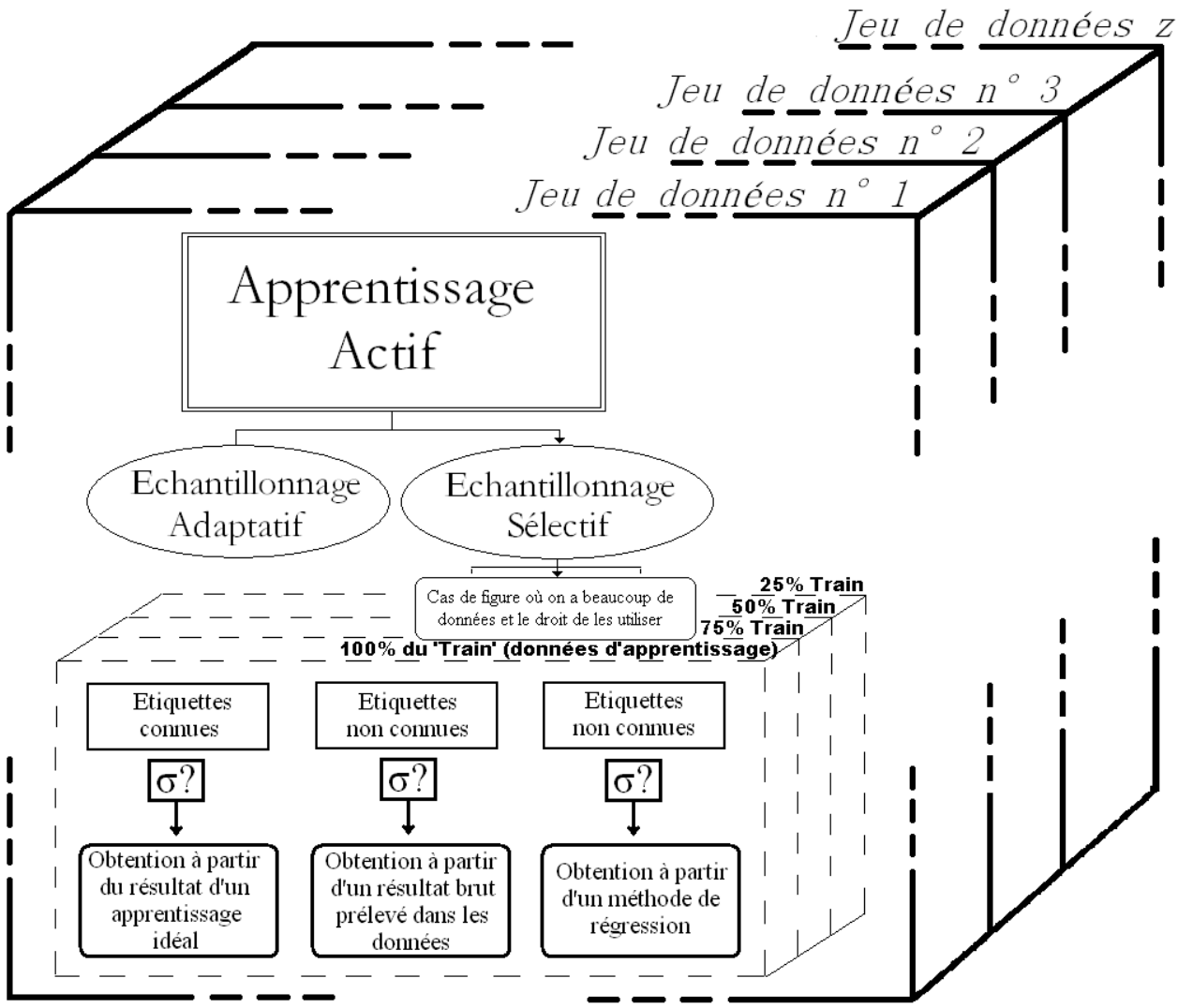


Figure 7 : Schéma des expérimentations

Le sujet du stage se situe dans le domaine du 'Data mining' ou 'Fouille de données'. On cherche à y développer des méthodes d'apprentissage appropriées aux données dont on dispose. L'apprentissage actif permet une sélection judicieuse d'instances de données pour acquérir de nouvelle information et un gain d'efficacité. Comme vu à la sous - section 'Echantillonnage Sélectif (Chapitre I -1), on se place dans le cas de l'apprentissage sélectif où l'on possède beaucoup de données et le droit de les utiliser. Pour cet apprentissage on utilisera la fenêtre de Parzen qui est un outil judicieux pour l'apprentissage de données, ce qui nous demande des expériences pour la maîtriser. Pour cela 3 expériences sont proposées.

Description des expériences

Comme il l'a été dit à la sous-section de la problématique du stage (Chapitre I-3). Nous proposons 3 expériences différentes visant à déterminer le σ optimal et qui seront confrontés selon la quantité de données et les jeux de données utilisées (cf. figure précédente).

Nous traiterons d'abord une méthode d'apprentissage idéal de classification (chapitre associé: chapitre III). C'est elle qui doit nous montrer les meilleures performances parmi toutes les méthodes. Elle ne pourra pas cependant pas être retenue puisqu'elle utilise les étiquettes des données. En effet, cette méthode revient à exploiter un problème de classification résolue pour

paramétrer ce même problème de classification. C'est un mode 'triche' qui nous sert de témoin à des fins d'évaluations des performances des autres méthodes.

Ensuite, nous essaierons la méthode de Schölkopf (chapitre associé: chapitre IV) en choisissant directement la racine de la dimension comme valeur du σ optimal (cf. figure). Nous cherchons bien entendu une méthode plus performante qu'un réglage issu de ce simple choix.

Enfin, nous développerons une méthode dite de régression (chapitre associé: chapitre V). C'est une méthode d'apprentissage qui va chercher à déterminer notre paramètre sur un problème déjà résolue sans que celui-ci soit un problème de classification et que les étiquettes soient apparentes. Cette méthode pourra donc être utilisée pour l'apprentissage actif.

D'autres méthodes comme l'estimation de densité donneront probablement lieu à d'autres expériences du même type à l'avenir. Ces 3 méthodes seront communément traitées dans le chapitre II selon leurs points de similarité, ceux contenus dans les jeux de données, les fenêtres de Parzen et les mesures de performances utilisées. Puis elles seront vues individuellement dans chacun de leur chapitre respectif (III, IV et V) selon leur implémentation, les résultats obtenus et les interprétations réalisées.

Χηαπιτρε II– Protocole expérimentale et données utilisateurs

II.1.Jeux de données utilisées

II.1.1Descriptions

Jeux de données

L'information peut avoir dans la nature des formes très diverses : sonore, visuelle, numérique etc. Echantillonnées d'un contexte réel et quelconque puis traitées par l'informatique, l'information est transformée en vecteurs de données numériques (ou instances). Les instances et leur étiquette sont réunies dans un tableau contenu dans un fichier.

Pour les expériences à réaliser, nous allons utiliser au moins 10 jeux de données issus de "l'UCI machine learning repository" sur le site de l'université de Californie. C'est un lieu de dépôt

pour des dizaines de jeux de données issues des travaux de groupes scientifiques. (www.ics.uci.edu/mlearn/MLSummary.html).

Jeu de données	Dimension	Instance de Train	Instance de Test
Damier	2	200	400
Iris	4	90	60
Glass	9	107	107
Wine	13	119	59
Pima	8	354	354
Australian	14	345	345
Segment	19	310	1998
Ionosphere	34	240	111
Sinx3	2	2000	30000
USPS2	256	317	1998
Emovoc	20	3783	1622
Letters	16	13400	6600

Tableau 4 : Jeux de données utilisés

Données

Les données se trouvent sous forme d'instances, des vecteurs d'une dimension unique (car d'un même espace). On caractérise l'appartenance de ces vecteurs à une classe grâce à une étiquette. Cette étiquette décrit le groupe d'origine (un point qui est 'rouge' ou 'bleu', un portrait qui représente 'une personne' ou 'un objet', un son 'd'une voix' ou 'd'un choc' etc.).

Pour faciliter notre tâche et les expériences, toutes les étiquettes sont remplacées par des classes (1, 2, 3…). Ainsi plutôt que de manipuler des chaînes de caractères, on manipulera des nombres.

Des caractéristiques intrinsèques

Les données sont diverses. Issue du monde réel, ou logique, celles-ci ne sont à priori pas aléatoires. Dans le cas contraire, il serait impossible de faire un apprentissage sur la localisation des vecteurs de données. En effet, des données aléatoires signifieraient que les classes sont attribuées aléatoirement. On suppose que les données que l'on possède doivent permettre de faire un apprentissage.

Les données ont des caractéristiques facilement exploitables comme le nombre de dimension, le nombre de classes possibles, le maximum, le minimum, la moyenne, et l'écart type des valeurs. On peut s'intéresser également aux caractéristiques relatives à chaque dimension. En effet, chaque dimension d'un jeu de données possède un maximum, minimum, moyenne et écart type. Les dimensions ne seront a priori pas aussi informatives les unes que les autres. Mais par simplicité, nous supposerons que toutes les dimensions ont la même importance pour l'apprentissage. Cette hypothèse nous est nécessaire pour utiliser le réglage d'une fenêtre de Parzen avec le paramètre calculé comme la racine carré de la dimension des données normalisées.

Les données sont enfin sujettes au bruit. Il en existe de plusieurs sortes (gaussien, blanc…). Ces défauts des données sont des contraintes pour les différentes fenêtres de Parzen. Le réglage

devra se protéger pour cela du sur - apprentissage cf. sous - section sur l'hyper - paramètre σ (Chapitre I – 2).

Segmentation préétablie

Nous avons opéré à un formatage particulier facilitant la lecture des fichiers. Ce formatage a permis de segmenter les données en 2, créant ainsi un ensemble d'apprentissage et de test.

Illustration avec le jeu de données 'damier.data'

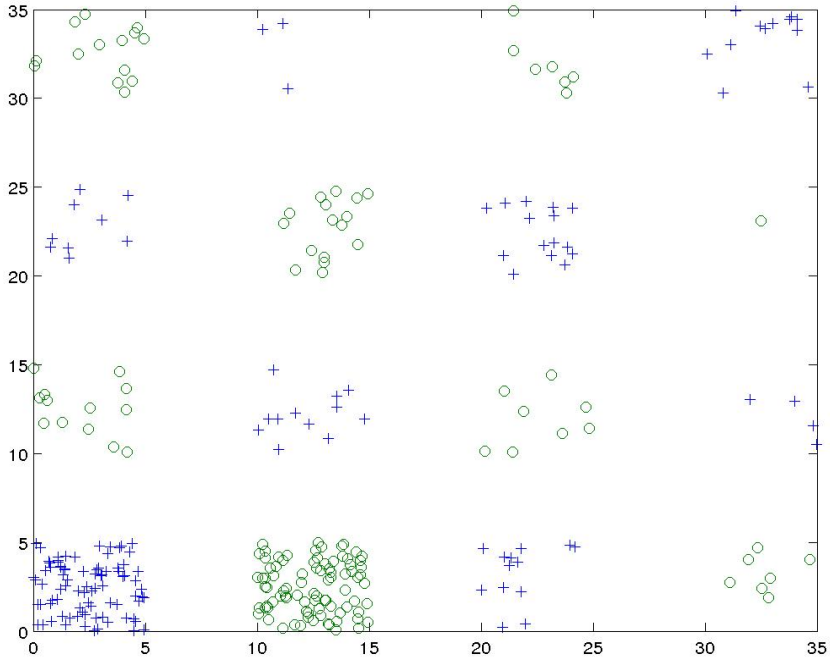


Figure 8 : Représentation de l'ensemble d'apprentissage et de ses 2 classes

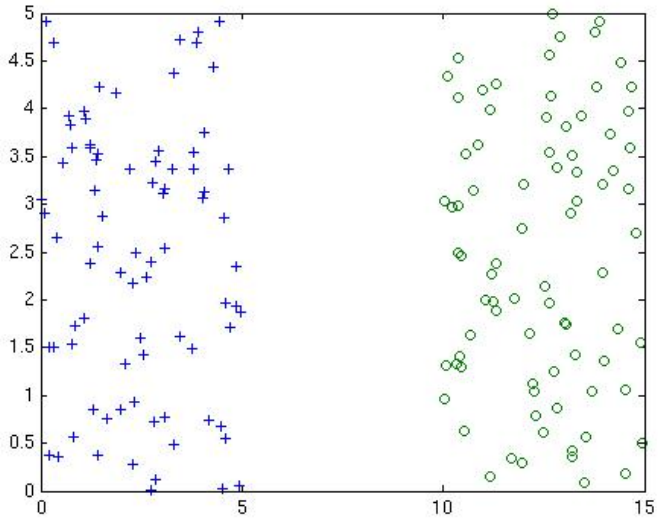


Figure 9 : Représentation de l'ensemble de test et de ses 2 classes

II.1.2Normalisation

Normalisation

On choisit de centrer et réduire toutes nos données selon leur dimension. Ainsi, sur chaque dimension, on doit obtenir une distribution des données répartie selon une loi normale. Pour réaliser cette normalisation, on exploite les instances des données en laissant les classes ou étiquettes. On calcule pour chaque dimension, la moyenne et l'écart type. On retranche d'abord la moyenne à toutes les valeurs et ensuite on les divise par l'écart type. La normalisation nous permet de faire l'obtention du paramètre simplement à partir de la dimension des données (2ème méthode qui sera vu au Chapitre IV). Elle a un intérêt pratique pour la fenêtre de Parzen car certains jeux de données contiennent des données de valeurs trop importantes. Le noyau contenant la fonction exponentielle demanderait un calculateur maniant plusieurs dizaines de chiffre après la virgule.

Algorithme de la normalisation

L'algorithme 2 résume les opérations qui ont été faite dans nos expérimentations pour les normaliser. On constate que les normalisations se font à partir des moyennes et écart-types extraites des données d'apprentissage et s'applique aux données d'apprentissage et de test.

Étant donnés:
2 ensembles de même dimension sans leur classe
- L'ensemble d'instances de test
- L'ensemble d'instances d'apprentissage

Pour toutes les dimensions**, faire**
Calculer la moyenne des données d'apprentissage de la dimension choisie

Calculer l'écart type des données d'apprentissage de la dimension choisie à partir de la moyenne calculée

Centrer et réduire les données d'apprentissage de la dimension choisie

Centrer et réduire les données de test de la dimension choisie

Algorithme 2 : Normalisation des données

II.2.Utilisation des Fenêtres de Parzen

II.2.1En classification

Un apprentissage supervisé

La classification correspond à une première utilisation d'une fenêtre de Parzen dans un contexte supervisé. En effet, cet apprentissage consiste à utiliser un apprentissage sur des données dont la classe est connue afin de réaliser des prédictions et classer d'autres données. Le fait de pouvoir comparer sorties prédites et sorties réelles pour améliorer le modèle implique que l'on se situe dans un apprentissage supervisé cf. description des expériences (Chapitre I - 3.3).

Classification

En apprentissage, la classification est l'attribution de classe à des instances. Lorsque les classes sont des étiquettes, on dit que l'on étiquete des instances. Cette attribution est un choix qui se fait parmi les classes du problème. Pour que ce choix se fasse correctement, il faut que toutes ces

classes se trouvent dans attribuées à des données d'apprentissage et que ces instances soient représentatives.

Dans notre contexte de classification, toutes les classes sont disponibles. On cache donc les classes des instances de test temporairement pour pouvoir réaliser les prédictions. Lorsque celles –ci sont terminées, on compare les prédictions avec la classe masqué. On évalue alors la performance de la classification.

Algorithme de Nadaraya-Watson en classification

L'algorithme de Nadaraya-Watson correspond à un poids qui est utilisé pour les prédictions de la fenêtre de Parzen (chapitre I 2.2). Dans le cas de la classification, l'algorithme de la méthode de prédiction est une somme de plusieurs poids d'une classe sachant l'instance de test choisie. L'algorithme permet d'estimer une prédiction des probabilités qu'une instance de test ait chacune des classes. Donc sachant l'instance de test, il calcule à partir des 'l' instances d'apprentissage, une probabilité pour toutes les classes. Donc pour une instance de test choisie, ces probabilités se somment à 1. Ci-dessous le prédicateur de la fenêtre de Parzen qui sera utilisé pour la classification :

$$\hat{P}(y|x) = \frac{\sum_{i=1, y_i=y}^{n} K(x, x_i)}{\sum_{\ell=1}^{n} K(x, x_\ell)} = \sum_{i=1, y_i=y}^{n} W_i(x)$$

Équation 12 : Prédicateur de la Fenêtre de Parzen de Classification

On observe que l'algorithme calcule la proportion des proximités des points d'une classe parmi toutes les classes.

II.2.2En régression

Un apprentissage supervisé

La régression est une méthode d'apprentissage supervisé puisqu'elle utilise des couples entrées et sorties. Les estimations de sorties et leurs comparaisons avec les valeurs réelles vont ainsi permettre l'amélioration du modèle de prédiction.

Régression

On ne peut plus prévoir les classes des instances de test puisque celles-ci ne sont plus apparentes dans notre ensemble de données. Au lieu de prédire les probabilités de ces classes avec des instances de données comme en classification, on estime directement une valeur cible à partir de valeurs explicatives. Les valeurs explicatives sont des instances de données réduites d'une dimension qui nous sert de valeurs cibles. On peut faire le choix de prédire un ensemble de valeurs cibles. Pour cela toute une dimension des instances est enlevée des valeurs explicatives et devient un ensemble de valeurs cibles. On va chercher à les prédire à l'aide des autres dimensions de valeurs explicatives et également les valeurs cibles des données d'apprentissage et ce au niveau d'une instance de test. Mais à la différence des classes du problème de classification, nos valeurs cibles

s'étalent sur l'espace des réels. Il n'est donc pas possible de classifier une instance de test selon une certaine valeur – cible. Cette classification en valeur réelle n'a pas d'intérêt car on devrait considérer quasiment autant de classes que d'instances de test. On cherche donc à 'régresser' l'instance de test selon les valeurs cibles (choisies parmi les valeurs explicatives) et la proximité des instances d'apprentissages. L'algorithme de NW utilisé par notre fenêtre en régression est formulé ci-dessous : (n est le nombre d'instances de l'ensemble d'apprentissage).

Algorithme de Nadaraya-Watson en régression

$$y = \hat{g}(x) = \sum_{i=1}^{n} \left(\frac{K(x, x_i)}{\sum_{\ell=1}^{n} K(x, x_\ell)} \right) y_i$$

$$= \sum_{i=1}^{n} W_i(x).y_i$$

Équation 13 : Estimateur de la Fenêtre de Parzen de Régression

On observe que cette formulation est bien une 'moyenne' des valeurs cibles où la pondération utilisée n'est pas 1 mais la proximité (ou distance au sens de Parzen) des instances de l'ensemble d'apprentissage. On retrouve comme une moyenne classique, la somme de toutes les pondérations utilisées au dénominateur. Au lieu de fournir une valeur de probabilité pour différentes classes, la fenêtre de Parzen en régression fournit une valeur régressée parmi les valeurs cibles pondérées.

II.2.3 Gamme de valeur du paramètre σ

Principe

Le paramètre σ va servir dans le cas des apprentissages supervisés à superviser l'apprentissage. En effet, on cherchera à obtenir la valeur du σ qui maximise les performances de notre méthode de prédiction. C'est celui là qui sera élu comme le σ optimal de la méthode.

Pour ce faire on va tester un grand nombre de valeur de σ. Dans nos expériences, ce nombre de valeur est un paramètre utilisateur. Une procédure permet le calcul de σ minimal et de σ maximal afin d'éviter à la procédure d'expérimenter des valeurs de σ inintéressantes. La gamme définit par ces 2 extrema est ensuite segmenter régulièrement selon le nombre de valeurs à examiner. Les paramètres σ minimal et σ maximal dépendent uniquement du jeu de données et ne change pas selon les méthodes. Ces deux paramètres peuvent aussi être choisis arbitrairement évitant ainsi une procédure automatique de leur détermination.

σ minimal

Une procédure permet la détermination du σ minimal qui a priori ne doit pas être plus petit que le plus petit écartement entre les données. On obtient le σ minimal en mesurant tous les écartements entre les instances d'apprentissage deux à deux et en choisissant le plus petit. Après des premières expériences, nous avons constaté que le σ minimal était tout de même pris trop grand. Il est donc arbitrairement divisé par $32=2^5$.

σ maximal

Pour déterminer le σ maximal, il a été choisi de trouver le plus grand écartement entre une instance et la moyenne de ces instances sur l'ensemble des données d'apprentissage. Cette moyenne est l'origine de l'espace de données puisque les données ont été normalisées. L'écartement est mesuré par la distance euclidienne. Un des 2 points étant l'origine, on mesure donc la norme L2 des instances et on cherche le maximum.

II.3.Mesures des performances

II.3.1Critère de mesure de résultats

Principe

Lorsqu'on utilise une fenêtre de Parzen en classification ou en régression, on obtient effectivement des valeurs qui correspondent au résultat d'un apprentissage réalisé sur les données. On peut effectivement comparer prédictions de classes et classe, valeurs cibles estimées et réelles. Mais lorsqu'on utilise de grandes quantités de données, des méthodes existent pour calculer des performances de l'apprentissage comparables entre les différents essais et entre tous les jeux de données. Nous utiliserons une méthode appropriée à la classification, l'AUC et une autre à la régression, l'AOC.

Utilisation AUC et AOC

L'AUC est l'Area Under the ROC Curve c'est-à-dire l'aire sous la courbe de ROC. L'AOC est quant à lui, l'aire sur la courbe de REC (Area Over the REC Curve). Ce sont toutes deux des méthodes de quantification des performances des prédictions. L'un, l'AUC est approprié à la classification cf. figure 8, l'autre à la régression cf. figure 9. On remarque que l'AOC fournit une erreur au lieu d'une performance. On peut se ramener à une performance en prenant le complément à 1 de l'erreur. Mais les méthodes de calcul étant très différentes, nous avons préféré garder ce calcul d'erreur qui permet de ne pas faire de confusion ou de comparaison dangereuse des résultats de l'AOC et l'AUC. Nous verrons par la suite, et dans les méthodes de classification et de régression, que l'on cherchera plutôt le paramètre σ qui maximise l'AUC ou qui minimise l'AOC.

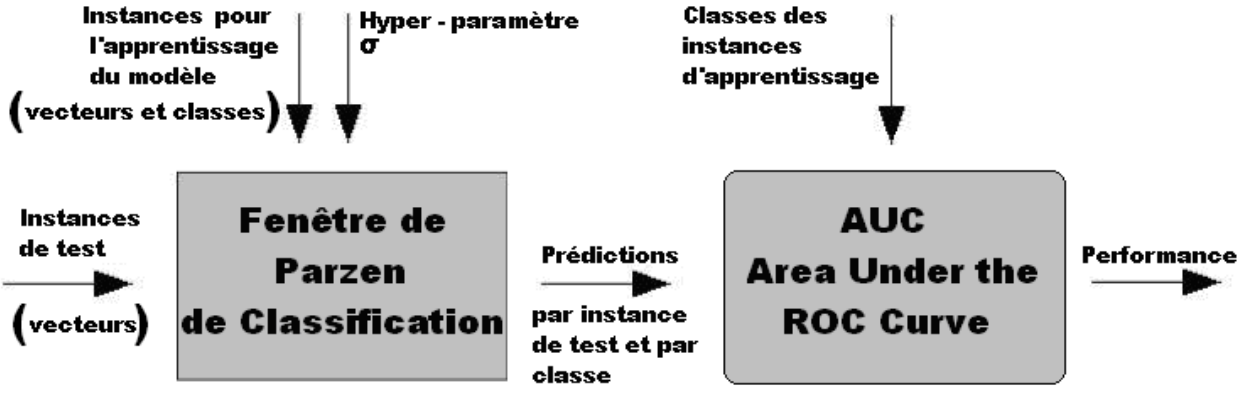


Figure 10 : Traitement des prédictions par l'AUC en classification

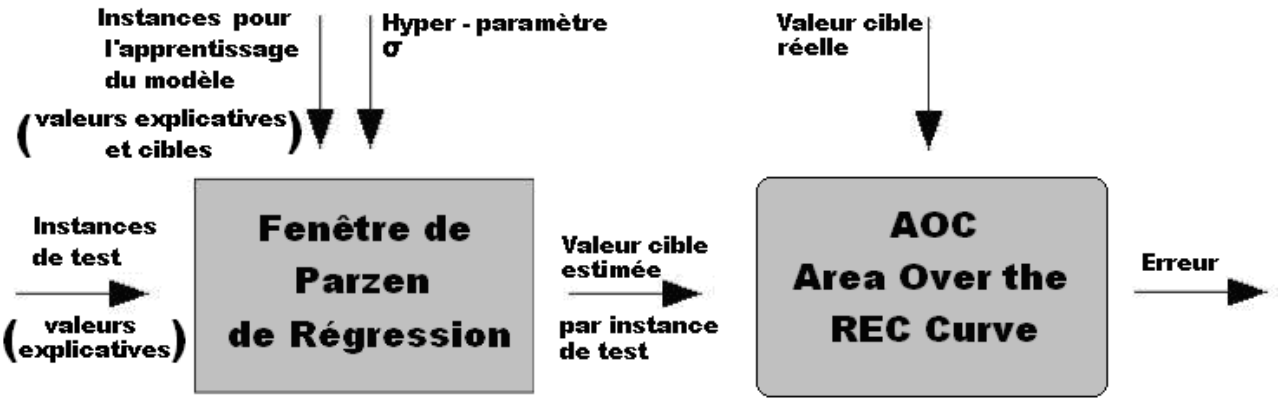


Figure 11 : Traitement des estimations par l'AOC en régression

II.3.2Validation croisée

Principe et Objectifs

Toutes les prédictions réalisées sont exploitées par une mesure de performance interprétable. Cependant, ce n'est pas suffisant car on s'intéresse à connaître la capacité de généralisation du modèle sur de nouvelles instances. Pour cela, on va estimer l'erreur de généralisation à l'aide d'une méthode de validation croisée.

k - fold cross validation

Nous avons choisi la k-fold cross validation comme méthode de validation croisée. Comme toute méthode de validation croisée, elle consiste à répéter la procédure, ici les prédictions sur l'ensemble de données d'apprentissage. A chaque répétition, on divise différemment cet ensemble en un sous ensemble d'apprentissage et de test. Dans la k-fold cross validation, il y a k partitions où k est un paramètre constant pour toutes les expériences que l'on voudra pouvoir comparer.

Selon cette méthode, on répète k fois une découpe de l'ensemble de données en ensemble de test et d'apprentissage. La découpe de l'ensemble d'instances se fait en k sous-ensembles que l'on réunie ensuite en (k-1) ensembles selon les 1 parmi k combinaisons possibles. Les (k-1) ensembles d'instances forment un grand ensemble d'apprentissage. La k-ième ensemble d'instances devient un petit ensemble de test. Une illustration de ce partitionnement et regroupement et représenté ci-contre.

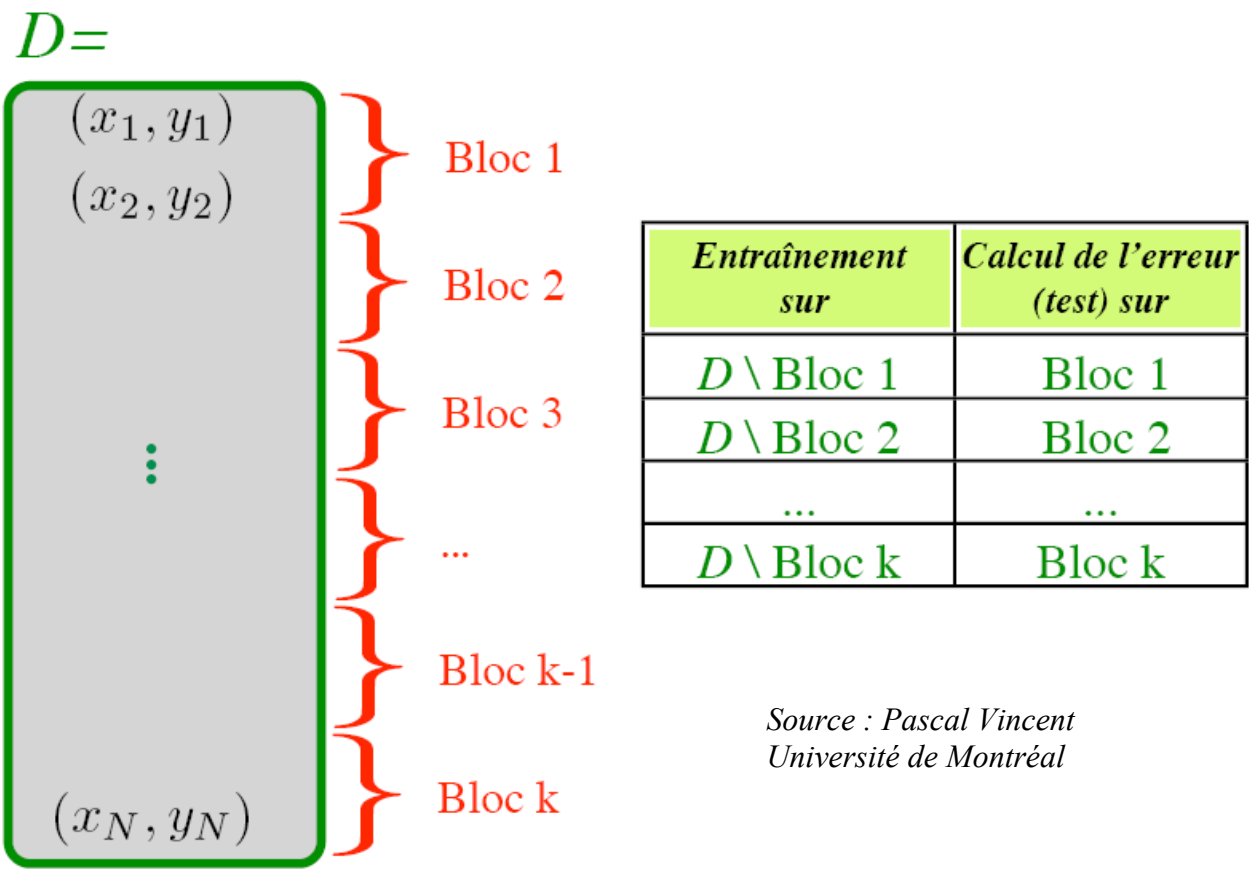


Figure 12: k-fold cross validation - Partitionnement et regroupement des ensembles

La répétition des expériences va permettre d'obtenir des valeurs moyennes et des écarts types des performances. Ce sont ces écarts type qui donne un estimé de l'erreur de généralisation.

Algorithme de la k - fold cross validation

Dans la réalisation faite, la k-fold cross validation construit de nouveaux ensembles d'apprentissage et de test à partir d'une matrice de destination. Celle –ci permet d'attribuer à tour de rôle les partitions d'apprentissage et la partition de test. Chaque ligne correspond à une des k regroupements possibles. 1 désigne la partition qui constituera à elle seule l'ensemble de test et 0 désigne les partitions qui seront regroupés dans l'ensemble d'apprentissage. Ci-dessous un exemple de matrice de répartition et ci-contre l'algorithme utilisé pour la k - fold cross validation.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Équation 14 : Matrice de destinations k x k dans le cas k = 5

Étant donnés:

- Le paramètre k
- Un ensemble d'instances et si besoin leurs classes (si classification)

Construire une matrice k *x* k de répartitions des instances pour le partitionnement avec des 0 et des 1 avec 0 pour les partitions destinées à l'ensemble d'apprentissage et 1 destinées à l'ensemble de test

Construire une liste de k nombre d'instances à mettre dans les k partitions.

Pour toutes les lignes de la matrice**, faire**
    Pour toutes les partitions**, faire**
        Se placer au niveau du 1er élément de cette partition

        Lire le nombre d'instances à récupérer dans la liste (voir plus haut)

        Si la valeur associée dans la matrice de répartition est 0 (et la destination est l'ensemble d'apprentissage)
            Mettre le nombre d'instances calculées dans l'ensemble d'apprentissage
            Mettre si besoin les classes correspondantes dans l'ens. des classes d'apprentissage
        Sinon la valeur associée dans la matrice de répartition est 1 (et la destination est l'ensemble de test)
            Mettre le nombre d'instances calculées dans l'ensemble de test
            Mettre si besoin les classes correspondantes dans l'ens. des classes de test
        Fin si

        Opérer sur ces nouveaux ensembles de test et d'apprentissage.
        (Selon la méthode : Fenêtre de Parzen, ... AUC, ..., etc.)
    Fin Pour
Fin Pour

Algorithme 3 : k-fold cross validation

II.3.3 Estimation de la performance finale

Principe

Nous devons évaluer les performances et comparer les différentes valeurs des σ obtenues. On utilise donc un test final pour tous les σ optimaux. Ce test est identique quelque soit les méthodes de détermination de ces σ.

Utilisation de Fenêtre de Parzen de Classification

Choisie comme test final, la fenêtre de Parzen de classification est utilisée une dernière fois pour valider le σ optimal obtenu par la méthode choisie. Son but est de réaliser des prédictions sur l'ensemble de test que l'on lui propose à partir de l'ensemble d'apprentissage. Cet ensemble de test n'a pas été exploité jusqu'à présent.

Utilisation de l'AUC

La valeur AUC est calculée suite aux prédictions réalisées par la fenêtre de Parzen de classification. Cette valeur permet la comparaison de toutes les expérimentations ayant visé la détermination d'un σ optimal avec leur jeu de données et leur quantité de données d'apprentissage initialement utilisée

II.4.Résultats

II.4.1Synthèse des méthodes:

Algorithme commun à toutes les méthodes

Le réglage de σ commence par des étapes préalables indispensables. Il faut d'abord lire les 2 ensembles d'apprentissage et de test qui sont préalablement établis dans le fichier contenant le jeu de données. Ces 2 ensembles sont ensuite normalisés, on normalise l'ensemble de test à partir des moyennes et écart – types obtenus sur les dimensions de l'ensemble d'apprentissage.

L'expérience commence par l'extraction d'un pourcentage de l'ensemble d'apprentissage par un tirage sans remise. Ensuite vient la détermination du σ optimal qui utilise la méthode de prédiction demandée. Dans la méthode d'obtention par la dimension des données, il n'y a pas d'opérations de prédictions pour déterminer σ optimal. Elle est donc faite au tout début de la procédure. Dans les autres méthodes, une expérience contient 2 étapes : cette détermination et le test final utilisant le paramètre déterminé. Le test final calcule la performance du σ optimal sur la fenêtre de Parzen de Classification.

L'expérience est réitérée suivant tous les pourcentages à examiner et le nombre d'itérations demandées. Le nombre d'itérations a pour but de nous assurer que lorsque les pourcentages de l'ensemble d'apprentissage sont petits, les performances ne soient pas exagérément basses ou hautes selon le choix mauvais ou bons des données extraites.

Etant donnés:

- le jeu de données sous un formatage particulier
- des pourcentages « n » choisis (pour le tirage des données du train)
- un nombre d'itérations choisis pour chaque pourcentage
- un paramètre k choisi de la k-fold-cross-validation
- le nombre de valeurs de σ à examiner

Lire le jeu de données séparé en 2 ensembles, le premier d'apprentissage 'Train' et l'autre de 'Test'

Normaliser les données.

Pour les pourcentages n choisis**, faire**
    Pour j variant de 1 à J, **faire**
        Extraire des données de l'ensemble d'apprentissage par un tirage sans remise

        Évaluer la fenêtre de Parzen de classification
        - réglée avec le σ optimal:
        - sur les instances de l'ensemble d'apprentissage: Train
        - pour les instances de l'ensemble de test: Test
        Remplir un tableau de prédiction par classe et instance de test

        Pour toutes les classes
            Construire la courbe de ROC
            Calculer l'aire sous la courbe de ROC (AUC)
        Fin Pour

        Calculer l'espérance des AUC (pondérant par la proportion des classes)
    Fin Pour
Fin Pour

Obtention d'un AUC moyen et de sa variance pour le choix de σ optimal (selon la stratégie) et pour chaque pourcentage

Algorithme 4 : commun à toutes les méthodes

Algorithme commun aux méthodes 1 et 3

D'après les Sections "Utilisation des Fenêtre de Parzen" et "mesure de performances" dans le chapitre courant, les méthodes 1 et 3 ont de très nombreux aspects communs. Les algorithmes qui leur ont été proposés sont donc assez proches. Leur partie commune est écrite ci-dessous. On observe dans cet algorithme que le jeu de données a un formatage particulier pour faciliter sa lecture. Il est formaté de telles matières que les premiers ensembles d'apprentissage ("Train') et de test (qui sert au test finale uniquement) soient figées pour toutes les expériences. Autre remarque:

Etant donnés:

- le jeu de données sous un formatage particulier
- des pourcentages « n » choisis (pour le tirage des données du train)
- un nombre d'itérations choisis pour chaque pourcentage
- un paramètre k choisi de la k-fold-cross-validation
- le nombre de valeurs de σ à examiner

Lire le jeu de données séparé en 2 ensembles, le premier d'apprentissage 'Train' et l'autre de 'Test'

Normaliser les données.

Déterminer le σ minimal (plus petite distance entre 2 instances) et maximal (plus grande distance avec le barycentre des instances)

Calculer le pas de σ à partir du min et du max et du nombre de valeurs de σ

Pour les pourcentages n choisis**, faire**
    Pour j variant de 1 à J, **faire**
        Extraire n% de l'ensemble d'apprentissage par un tirage sans remise

        Pour toutes les combinaisons possibles des k partitions en 2 partitions
            Regrouper les k partitions en 2 /s-ensembles (New_Train%,New_Test%)
                avec k-1 partitions pour New_Train% et 1 pour New_Test%

            Pour σ variant de σ_{min} à σ_{max} (avec un pas σ_{pas}), **faire :**
                Évaluer la performance du σ courant selon la méthode de classification ou régression
            Fin Pour
        Fin Pour

        Évaluer la performance moyenne et sa variance (erreur de généralisation) en fonction de la valeur de σ

        Choisir un σ optimal (celui qui maximise la performance)

        Évaluer la fenêtre de Parzen de classification
        - réglée avec le σ optimal:
        - sur les instances de l'ensemble d'apprentissage: Train
        - pour les instances de l'ensemble de test: Test
        Remplir un tableau de prédiction par classe et instance de test

        Pour toutes les classes
            Construire la courbe de ROC
            Calculer l'aire sous la courbe de ROC (AUC)
        Fin Pour

        Calculer l'espérance des AUC (pondérant par la proportion des classes)
    Fin Pour
Fin Pour

Algorithme 5 : Commun aux méthodes 1 et 3

Rôles des validations

On observe dans l'algorithme 4 que la k-fold cross validation permet d'obtenir plusieurs valeurs de performance pour un même σ. Ces valeurs permettent d'obtenir effectivement une moyenne et un écart type, écart type qui pour rappel donne l'estimé de l'erreur de généralisation.

La partie 'test final' qui comprend l'exécution d'une fenêtre de classification réglé avec le σ optimal et de l'AUC se retrouve également dans la 2ème méthode dite de 'Schölkopf'. Ainsi toutes les méthodes de prédiction et/ou de détermination du σ optimal peuvent être comparées selon leur performance à régler le paramètre pour la fenêtre de Parzen de classification.

II.4.2 Format des résultats

Ci-dessous un récapitulatif des trois résultats que l'on se propose de 'récolter':

'Résultat n°1' (pour les méthodes de classification et de régression uniquement)

Pour une expérience faite sur un jeu de données, avec un pourcentage de l'ensemble d'apprentissage choisi,
Déterminer la performance des prédictions de la fenêtre de Parzen et ce en fonction de σ.

'Résultat n°2' (σ optimal)

Pour une expérience faite sur un jeu de données, avec un pourcentage de l'ensemble d'apprentissage choisi, et à partir du 'Résultat n°1',

Désigner parmi toutes les valeurs de σ, celui dont la performance est la meilleure.

'Résultat n°3'

Selon les expériences faites sur des jeux de données, pour tous les pourcentages de l'ensemble d'apprentissage choisis, et à partir du 'Résultat n°2',
Déterminer les performances du σ optimal obtenue pour la fenêtre de Parzen de classification

Χηαπιτρε ΙΙΙ- Réglage du σ à l'aide d'une méthode de classification

III.1. Introduction

Un apprentissage Idéal

La méthode d'obtention du σ optimal par une méthode de classification permet de régler cette même méthode de classification. C'est donc une méthode 'idéal' au sens où elle se trouve dans un cas idéal : on possède déjà les classes de nos instances. Il ne sera à priori pas possible de faire mieux que cette méthode. Mais il sera possible d'approcher voire d'égaler ses performances.

Attentes vis à vis des expériences

La méthode de classification devrait nous permettre d'évaluer assez rapidement l'efficacité des autres méthodes. En effet, grâce au test final, on compare dans les mêmes conditions les performances obtenues par les σ optimal réglant la fenêtre de Parzen de classification.

III.2. Implémentation

Particularités de la procédure

On détermine avant l'expérience les paramètres utiles telle que le σ min et max (cf. Gamme de valeur du paramètre σ - Chapitre II 2.3). D'autres paramètres sont nécessaires mais sont fournis à la procédure au préalable tel que k le paramètre de k – fold cross validation, le nombre de valeurs de σ à examiner, les pourcentages etc.

Une expérience contient 2 étapes : la détermination du σ optimal et le test final utilisant ce paramètre. La détermination du σ optimal utilise la méthode de classification avec la fenêtre de Parzen de classification et l'AUC espérée sa mesure de performance.

Cette première classification est répétée selon 2 boucles. L'une permet le changement des partitions de test et d'apprentissage à l'aide de la k – fold cross validation. L'autre fait varier σ selon la gamme de σ définie par ces extrema et selon le nombre de σ.

Le σ optimal est la valeur de σ qui maximise l'AUC moyen. On rappelle que l'AUC moyen est la moyenne des AUC espérée obtenue pour chaque σ et grâce à la répétition de la classification grâce à la k – fold cross validation.

Le test final calcule la performance du σ optimal sur la fenêtre de Parzen de Classification et fournit la valeur d'AUC espérée correspondante.

L'expérience est réitérée suivant tous les pourcentages à examiner et le nombre d'itérations demandées. Le nombre d'itérations a pour but de nous assurer que lorsque les pourcentages de l'ensemble d'apprentissage sont petits, que les performances ne soient pas exagérément basses ou hautes selon le choix mauvais ou bons des données extraites. On obtient donc au final plusieurs valeurs d'AUC pour chaque pourcentage de l'ensemble d'apprentissage extrait. Ceci permet d'en calculer une moyenne et un écart type.

Algorithme

Étant donnés:

- le jeu de données sous un formatage particulier
- des pourcentages « n » choisis (pour le tirage des données du train)
- un nombre d'itérations choisis pour chaque pourcentage
- un paramètre k choisi de la k-fold-cross-validation
- le nombre de valeurs de σ à examiner

Lire le jeu de données séparé en 2 ensembles, le premier d'apprentissage 'Train' et l'autre de 'Test'

Normaliser les données.

Déterminer le σ minimal (plus petite distance entre 2 instances) et maximal (plus grande distance avec le barycentre des instances)

Calculer le pas de σ à partir du min et du max et du nombre de valeurs de σ

Pour les pourcentages n choisis**, faire**
Pour j variant de 1 à J, **faire**
Extraire n% de l'ensemble d'apprentissage par un tirage sans remise

Pour toutes les combinaisons possibles des k partitions en 2 partitions
Regrouper les k partitions en 2 /s-ensembles (New_Train%,New_Test%)
avec k-1 partitions pour New_Train% et 1 pour New_Test%

Pour σ variant de σ_{min} à σ_{max} (avec un pas σ_{pas}), **faire :**
Évaluer la fenêtre de Parzen de classification
- réglée avec le σ courant:
- sur les instances de l'ensemble d'apprentissage New_Train%
- pour les instances de l'ensemble de test New_Test%
Remplir un tableau de prédiction par classe et instance de test

Pour toutes les classes
Construire la courbe de ROC
Calculer l'aire sous la courbe de ROC (AUC)
Fin Pour

Calculer l'espérance des AUC (pondérant par la proportion des classes)
Fin Pour
Fin Pour

Évaluer l'AUC moyen et sa variance en fonction de la valeur de σ

Choisir un σ optimal

Évaluer la fenêtre de Parzen de classification
- réglée avec le σ optimal:
- sur les instances de l'ensemble d'apprentissage: Train
- pour les instances de l'ensemble de test: Test
Remplir un tableau de prédiction par classe et instance de test

Pour toutes les classes
Construire la courbe de ROC
Calculer l'aire sous la courbe de ROC (AUC)
Fin Pour

Calculer l'espérance des AUC (pondérant par la proportion des classes)
Fin Pour
Fin Pour

Obtention d'un AUC moyen et de sa variance pour le choix de σ optimal (selon la stratégie n°1) et pour chaque pourcentage

Algorithme 6 : Méthode 1 – Obtention du σ optimal à partir d'une méthode de classification

Fenêtre de Parzen de classification

La fenêtre de Parzen de classification récupère 2 ensembles d'instances : un ensemble d'apprentissage et un ensemble de test. Les instances y sont toutes définies dans une même espace. Pour chaque instance, la fenêtre va calculer les prédictions de chaque classe. Ces prédictions sont complémentaires à 1 et permettraient d'ores et déjà de déterminer la classe prédite pour une instance. Cependant, l'analyse de la performance par la courbe de ROC et l'AOC n'utilisera pas les classes prédites mais les valeurs de prédictions pour toutes les classes. Ainsi toute l'information créée par notre modèle de prédiction est exploitée par la suite dans l'analyse de performance du modèle lui-même. On remarque que dans une utilisation plus pratique de la fenêtre de Parzen de classification, l'algorithme indiquerait un tableau des classes les plus probables pour l'ensemble de test au lieu des prédictions sur toutes les classes.

Algorithme

Étant donnés:

- Une valeur choisie pour le paramètre sigma du noyau gaussien
- Les ensembles d'instances de test et d'apprentissage
- Les classes de l'ensemble d'apprentissage
- Le nombre de classe du problème

Pour toutes les instances de test**, faire**
    Pour toutes les classes du problème**, faire**
        Initialiser le numérateur et le dénominateur de la fenêtre de Parzen

        Pour toutes les instances d'apprentissage**, faire**
            Calculer la distance euclidienne entre instance de test et d'apprentissage

            Calculer la distance au sens de Parzen(Noyau) à l'aide de la distance euclidienne

            Si la classe de l'instance d'apprentissage est la classe courante
                Ajouter le Noyau calculé au numérateur.
            Fin Si

            Ajouter le Noyau calculé au dénominateur
        Fin Pour

        Calculer le rapport Numérateur et Dénominateur

        Mettre cette valeur dans un tableau de prédictions par classe et instance de test
    Fin Pour
Fin Pour

Algorithme 7 : Fenêtre de Parzen de Classification

AUC

L'AUC signifie « area under the curve » soit l'aire en dessous de la courbe, où la courbe désignée est la courbe de ROC (Receiver Operating Characteristics). Alors que la courbe de ROC est un outil de visualisation de la performance d'un classifieur, l'AUC permet de quantifier la performance à partir de l'aire sous la courbe. Pour construire une courbe de ROC, on représente la bonne ou mauvaise classification d'un certain nombre d'éléments dans une classe donnée. On exploite pour cela les prédictions (faites par un modèle de prédiction) pour tous ces éléments et pour la classe donnée. L'AUC correspond à une valeur de performance. Mais c'est en réalité l'espérance des AUC de toutes les classes qui est calculée. Elle est obtenue en faisant une sorte de moyenne des AUC mais en pondérant chaque valeur par la proportion de la classe dans l'ensemble d'apprentissage utilisé.

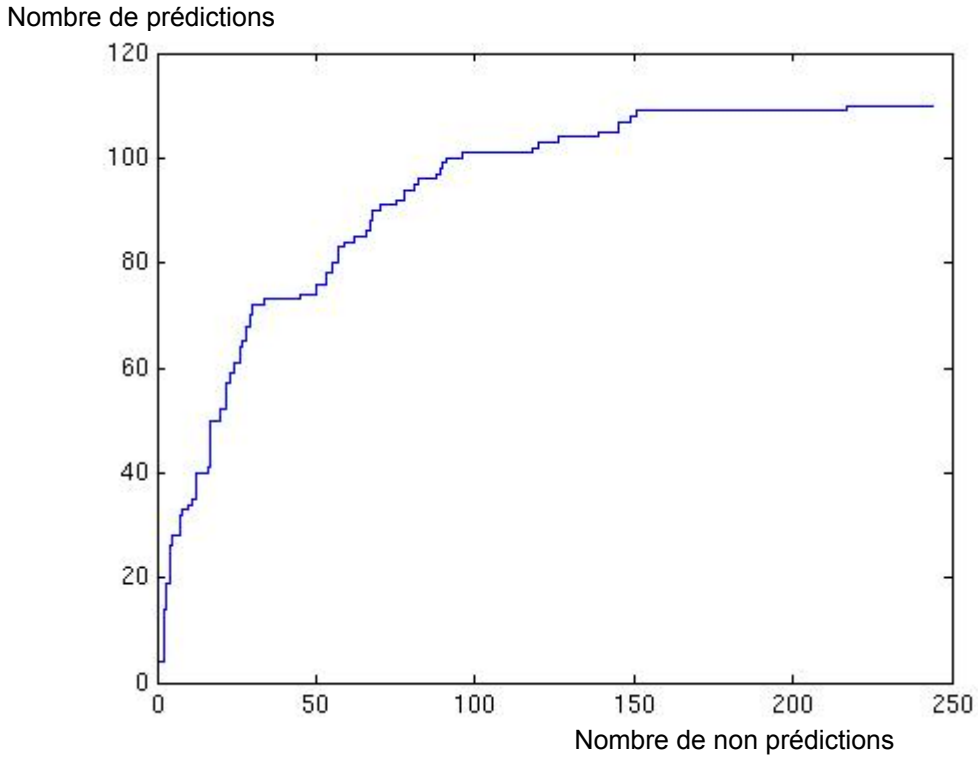


Figure 13 : Exemple de courbe de ROC réalisée sur le jeu de données 'Pima' (Test final)

Algorithme

Pour comprendre l'implémentation de l'algorithme de l'AUC et en particulier la construction d'une courbe de ROC, on pourra se reporter au tutorial de Tom Fawcett [Fawcett].

Étant donnés:

- Un tableau de prédictions par éléments (instances prédites par le modèle) et par classe
- Les 'vraies' classes des instances

Pour toutes les classes**, faire**

Trier les éléments à classer dans l'ordre décroissant de leurs prédictions de la classe courante

Commencer à tracer la courbe de ROC en partant de l'origine

Initialiser l'aire sous la courbe de ROC à 0

Pour l'élément trié suivant (D'abord Pour le 1er élément de la liste triée)**, faire**

Mémoriser le point de la courbe où l'on se trouve

Si la prédiction est bonne (si la classe prédite/courante est bien la classe de l'élément)
Alors on ajoute 1 à l'ordonnée de la courbe
Sinon (si la prédiction est mauvaise) on ajoute 1 à l'abcisse de la courbe
Fin Si

Pour tous les éléments triés suivants ayant la même valeur de prédiction

Si la prédiction est bonne (classe courante est la 'vraie' classe de l'élément)
Alors on ajoute 1 à l'ordonnée de la courbe
Sinon (si la prédiction est mauvaise) on ajoute 1 à l'abcisse de la courbe
Fin Si
Fin Pour

Calculer l'aire du trapèze sous segment [point courant de la courbe et point mémorisé]

Ajouter cette aire à l'aire sous la courbe de ROC

Fin Pour

Si l'abcisse de la courbe de ROC est nulle
l'AUC est égale à 1
Fin Si

Si l'ordonnée et l'abcisse de la courbe de ROC sont non nulles
Normaliser l'AUC en divisant l'aire par l'abcisse et l'ordonnée
Fin Si

Calculer la proportion de la classe courante parmi les 'vraies' classes de tous les éléments

Fin Pour

Calculer l'espérance des AUC pondérée par la proportion des classes.

Algorithme 8 : AUC esperée – Area Under the 'ROC' Curve

III.3. Résultats

La 'méthode 1' nous donne la validation de la performance de la méthode de classification en fonction des jeux de données et la portion de l'ensemble d'apprentissage utilisée. Nous exploiterons d'autres résultats intermédiaires intéressants tels que la performance de la méthode de prédiction (Fenêtre de Parzen en régression) en fonction de son paramètre σ ou encore le σ optimal toujours selon le jeu de données et la portion d'instances apprises.
Ci-dessous un récapitulatif des trois résultats que la méthode produit :

'Résultat n°1'

Pour une expérience faite sur un jeu de données, avec un pourcentage de l'ensemble d'apprentissage choisi,
Déterminer la performance des prédictions de la fenêtre de Parzen en classification et ce en fonction de σ.

'Résultat n°2' (σ optimal)

Pour une expérience faite sur un jeu de données, avec un pourcentage de l'ensemble d'apprentissage choisi, et à partir du 'Résultat n°1',
Désigner parmi toutes les valeurs de σ, celui dont la performance est la meilleure, donner ce σ et sa performance

'Résultat n°3'

Selon les expériences faites sur des jeux de données, pour tous les pourcentages de l'ensemble d'apprentissage choisis, et à partir du 'Résultat n°2',
Déterminer les performances d'une fenêtre de Parzen de classification muni du σ optimal selon la fenêtre de Parzen de classification

Les représentations graphiques utilisant les moustaches utilisent 2 fois l'écart type au dessus et en dessous de la moyenne.

Performance des prédictions en classification

Les expériences ont été nombreuses mais limitées à des jeux de données d'une certaine taille, ainsi le jeu de donnée "Letters" n'a pas été testé. Un choix de paramètres des expériences a été préalablement réalisé. Celui-ci a été de réaliser 5 itérations des 4 valeurs de pourcentage (100%, 75%, 50%, 25%). De plus le paramètre de la k fold cross validation a été fixé à 4. La plupart des jeux de données ont été testés pour 2 ou 3 valeurs du nombre de σ parmi 100, 300 et 1000.

Pourcentage extrait des données	σ moyen obtenu	AUC Moyen	Ecart type de l'AUC	Commentaires
	Australian			
100%	1,867185	0,925783	0,001598	**Grands σ instables et résultat stable**
75%	1,495261	0,924653	0,003041	
50%	2,059145	0,926675	0,000404	
25%	2,059145	0,926675	0,000404	
	Damier			
100%	0,011405	1	0	**Très Petits σ et résultat parfait**
75%	0,011405	1	0	
50%	0,009173	1	0	
25%	0,011405	1	0	
	Glass			
100%	0,44374768	0,712085	0,652254	**Moyens σ et résultat stable et très faible**
75%	0,70673341	0,720105	0,653934	
50%	1,364	0,726	0,658304	
25%	1,06857192	0,711119	0,63994	
	Ionosphere			

100%	1,535436	0,984066	0,004618	Grands σ instables et résultat stable
75%	1,642865	0,976374	0,003132	
50%	0,955319	0,99011	0,003556	
25%	2,77087	0,96456	0,026749	
	Iris			
100%	0,463015	0,997667	0,000697	Moyens σ instables et résultats stable
75%	0,035623	0,993583	0,00744	
50%	0,568111	0,9955	0,002327	
25%	0,02161	0,991375	0,008472	
	Pima			
100%	2,996571	0,838256	0,004417	Grands σ instables, résultat stable et plus faible
75%	1,953352	0,841833	0,001847	
50%	5,970155	0,83813	0,006176	
25%	1,731565	0,8388	0,006168	
	Segment			
100%	0,568717	0,963547	0,000374	Moyens σ instables et résultat stable
75%	0,634072	0,962301	0,001077	
50%	0,111229	0,960218	0,002421	
25%	0,372651	0,961246	0,001216	
	Sinx3			
100%	0,060185	0,998553	2,53E-05	Petits σ stables et résultat stable
75%	0,064997	0,998307	0,000279	
50%	0,086651	0,998299	0,000155	
25%	0,093869	0,99774	0,000927	
	USPS - 300 (au lieu de 1000) valeurs de σ et 4 itérations			
100%	2,28397088	0,930452	0,003925	Grands σ assez stables, bon résultat stable
75%	2,34362497	0,931946	0,00203	
50%	2,88051174	0,933304	0,000513	
25%	2,10500862	0,923981	0,00613	
	Wine			
100%	1,192748	0,99351	0,003419	Moyens σ instables et résultat stable
75%	0,72963	0,98744	0,006867	
50%	0,237924	0,989077	0,008977	
25%	0,266512	0,979175	0,002812	

Optimalité du Paramètre de la fenêtre

Les résultats sont globalement bons et stables, ce qui signifie que les données utilisées peuvent être apprises facilement dans un but de classification. Comme il l'a été dit précédemment, ces résultats sont a priori les meilleurs que l'on puisse obtenir.

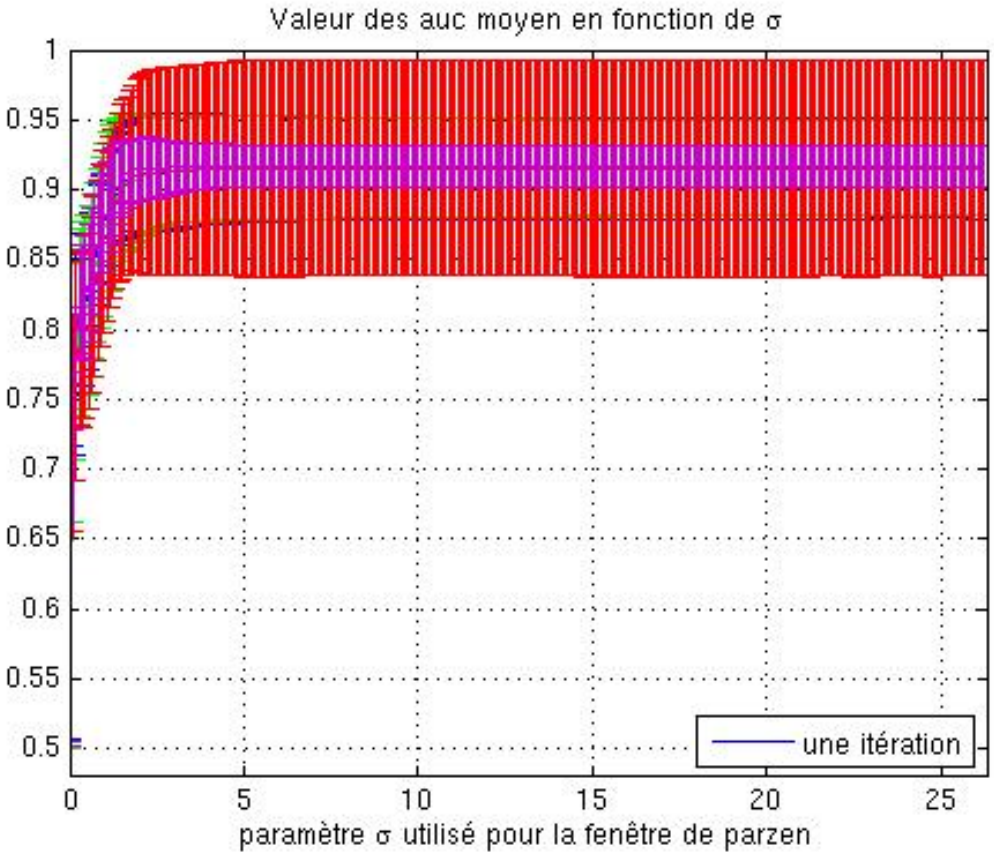


Figure 14 : AUC fonction de σ sur 'emovoc' et effet du sur-apprentissage

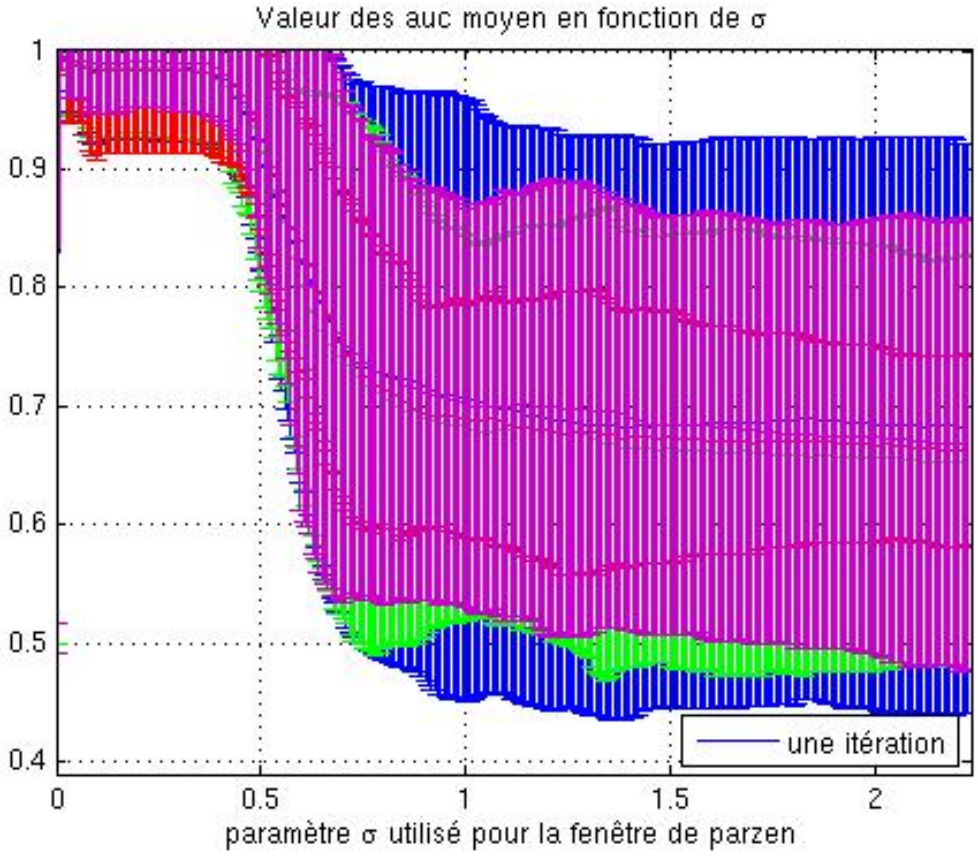


Figure 15 : AUC fonction de σ sur 'damier' et effet du lissage

Influence de la quantité de données

Lorsque la quantité des données, les résultats se maintiennent bien. La méthode de classification doit rester là encore la plus performante. Il se peut par contre que dans des cas de pourcentage de données inférieur à 100%, on observe de meilleurs résultats sur d'autres méthodes. Ceci est dans l'hypothèse où un tirage bon ou excellent se produise d'un côté et un mauvais ou très mauvais tirage de l'autre.

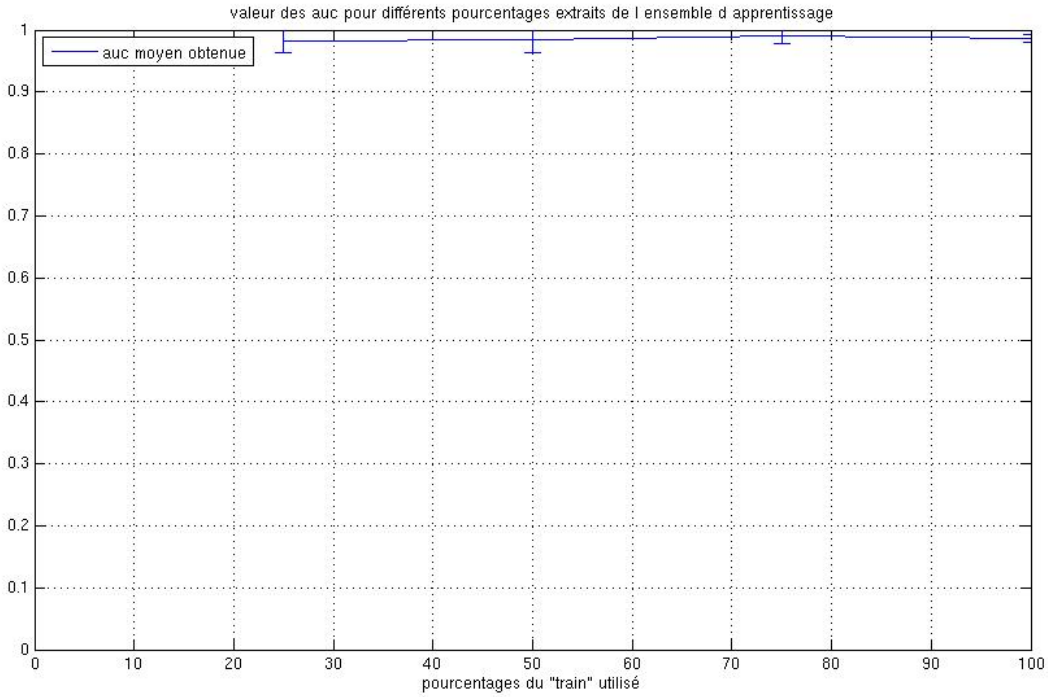


Figure 16 : Variations des performances selon la quantité de données sur Wine

Influence du choix du Jeu de données

'Pima' et 'Glass' semble être les jeux les plus difficiles à apprendre : On peut constater cette difficulté sur les graphes des résultats '1' réalisées sur ces 2 jeux de données sur lesquels on observe soit un effondrement précoce pour les grands σ et courbe plus basse (Pima)

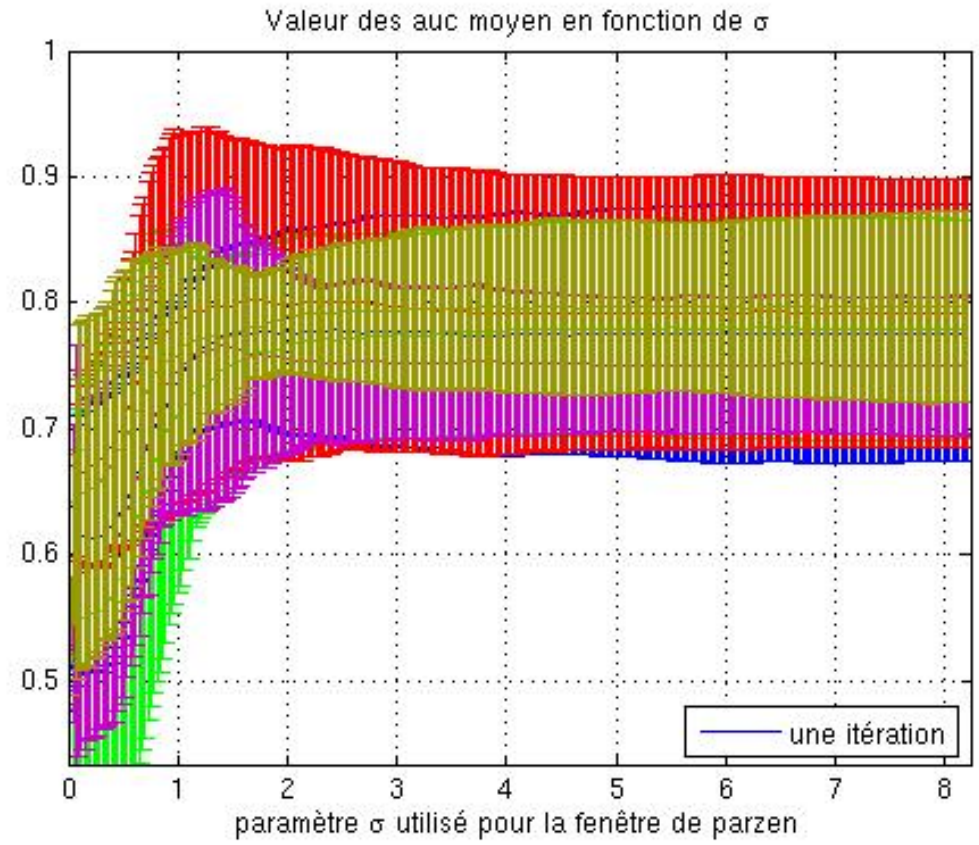


Figure 17 : AUC fonction de σ sur le jeu de données 'Pima'

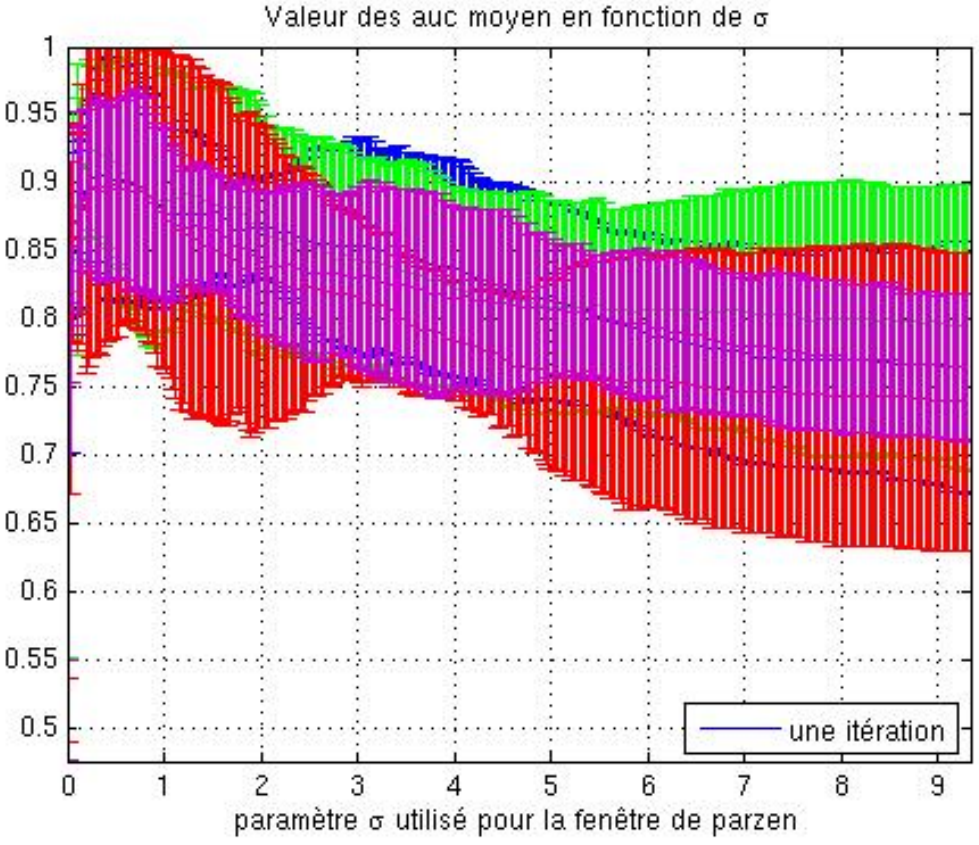


Figure 18 : AUC fonction de σ sur le jeu de données 'Glass'

III.4. Discussion

La méthode de classification propose des valeurs de σ optimales auxquelles devront se rapprocher les autres méthodes. Ces valeurs sont diverses car il y a un facteur 100 entre la plus petite valeur obtenue et la plus grande. On a observé également une forme typique des courbes de performance en fonction de la valeur de σ. Cette courbe est un effondrement à une performance d'environ 0.5 pour les valeurs suffisamment inférieures au σ optimal. Ce qui signifie que le risque de sur-apprentissage est à la fois grand et conséquent. La courbe maintient relativement bien ses performances pour une gamme de valeur de σ au dessus du σ optimal. L'effet du lissage des "gaussiennes" fait son effet très progressivement. Ce qui pose provoque un effondrement ou non des valeurs pour de grandes valeurs de σ. Le risque du lissage est quasi – inexistant. On est donc amené à penser que les méthodes qui fourniront une relativement grande valeur de σ seront plus fiable que celles qui prendront des risques dans les petites valeurs de σ.

Χηαπιτρε Ιϛ– Réglage du σ à l'aide des variances des données

Ιϛ.1. Introduction

Rôle et attentes vis à vis des expériences

Cette méthode utilise le réglage de Schölkopf pour trouver le meilleur σ de la fenêtre de Parzen de classification. On peut s'attendre à ce qu'elle ait de bonnes performances puisqu'elle a plusieurs été fois utilisée jusque là. Mais on pense qu'elle pourrait avoir aussi des limites dans certains cas de figure vu sa simplicité.

Ιϛ.2. Implémentation

Particularités de la procédure

Pour des soucis de similarités avec les autres méthodes, la méthode d'obtention par la dimension prend en paramètre exactement les mêmes arguments que les autres méthodes. Une extraction par un tirage sans remise est également réalisée mais elle est toujours de 100%.

On détermine avant l'expérience le σ optimal comme la racine carré de la dimension des données normalisées.

Une expérience ne contient qu'une seule étape : le test final utilisant ce paramètre. Le test final calcule la performance du σ optimal sur la fenêtre de Parzen de Classification et fournit la valeur d'AUC espérée correspondante.

L'expérience est réitérée suivant tous les pourcentages à examiner et le nombre d'itérations demandées. On obtient donc finalement plusieurs valeurs d'AUC pour chaque pourcentage de l'ensemble d'apprentissage extrait. Ceci permet d'en calculer une moyenne et un écart type.

Algorithme

Étant donnés:

- le jeu de données sous un formatage particulier
- des pourcentages « n » choisis (pour le tirage des données du train)
- un nombre d'itérations choisis pour chaque pourcentage
- un paramètre k choisi de la k-fold-cross-validation
- le nombre de valeurs de sigma à examiner

Lire le jeu de données séparé en 2 ensembles, le premier d'apprentissage 'Train' et l'autre de 'Test'

Normaliser les données.

Choisir sigma comme la racine carré du nombre de dimensions

Pour les pourcentages n choisis**, faire**
    Pour j variant de 1 à J, **faire**
        Extraire 100% de l'ensemble d'apprentissage par un tirage sans remise

        Évaluer l'AUC moyen et sa variance du sigma choisi

        Choisir un sigma optimal

        Évaluer la fenêtre de Parzen de classification
        - réglée avec le sigma optimal:
        - sur les instances de l'ensemble d'apprentissage: Train
        - pour les instances de l'ensemble de test: Test
        Remplir un tableau de prédiction par classe et instance de test

        Pour toutes les classes
            Construire la courbe de ROC
            Calculer l'aire sous la courbe de ROC (AUC)
        Fin Pour

        Calculer l'espérance des AUC (pondérant par la proportion des classes)
    Fin Pour
Fin Pour

Obtention *d'un AUC moyen et de sa variance pour le choix de sigma optimal (selon la stratégie n°2) et pour chaque pourcentage*

Algorithme 9 : Méthode 2 - Obtention du σ optimal à partir d'un résultat 'brut' prélevé sur les données

Iç.3. Résultats

La 'méthode 2' nous donne la validation de la performance de la méthode de Schölkopf en fonction des jeux de données et la portion de l'ensemble d'apprentissage utilisée. Il n'y a cette fois-ci pas de performance de la méthode à examiner en fonction des valeurs de σ puisque celui-ci est déjà choisi. Ci-dessous un récapitulatif des résultats de cette méthode :

'Résultat n°2' (σ optimal)

Pour une expérience faite sur un jeu de données,
Désigner la valeur de σ et sa performance

'Résultat n°3'

Selon les expériences faites sur des jeux de données, pour tous les pourcentages de l'ensemble d'apprentissage choisis, et à partir du 'Résultat n°2',
Déterminer les performances d'une fenêtre de Parzen de classification muni du σ optimal selon la le choix de Schölkopf

Optimalité du Paramètre de la fenêtre

La valeur de σ déterminée par la dimension des données semble convenir. Hormis avec le jeu Damier où le résultat est très mauvais, tous les jeux de données donnent de bon résultats. Le réglage réalisé n'est peut être pas optimal mais il s'en approche vraiment en donnant pour toutes les données normalisées, un σ forcément supérieur ou égal à $\sqrt{2}$. Il évite tout risque de sur-apprentissage. Par contre, il ne peut éviter le risque de lissage qui représente un risque très modéré sauf dans le cas de certains jeux de données comme Damier. En effet les données y sont regroupés par des paquets de tels sorte que s'y on donne beaucoup d'importances aux paquets voisins (en augmentant σ à $\sqrt{2}$ par exemple), les performances s'effondrent. On pourra se donner une idée en regardant la figure 8 qui représentent les données non normalisées de Damier.

Les résultats de la méthode 2 ont été rassemblés dans le tableau ci-dessous :

σ obtenu	AUC estimée	σ obtenu	AUC estimée
Australian		**Pima**	
3,74	0,93	2,00	0,97
Damier		**Segment**	
1,41	0,45	4,36	0,94
Glass		**Sinx3**	
3,00	0,77	1,41	0,96
Ionosphere		**USPS**	
5,83	0,89	16,00	0,83
Iris		**Wine**	
0,34	1,00	3,61	1,00

Tableau 5 : σ et AUC correspondants obtenues sur chaque jeu de données

Influence de la quantité de données

Contrairement aux autres méthodes, cette méthode ne tient pas compte de l'apprentissage préalable sur un pourcentage extrait des données d'apprentissage. Il y'a donc un unique résultat comme sur le graphique ci-dessous contrairement aux autres méthodes. Cette représentation est donc inutile.

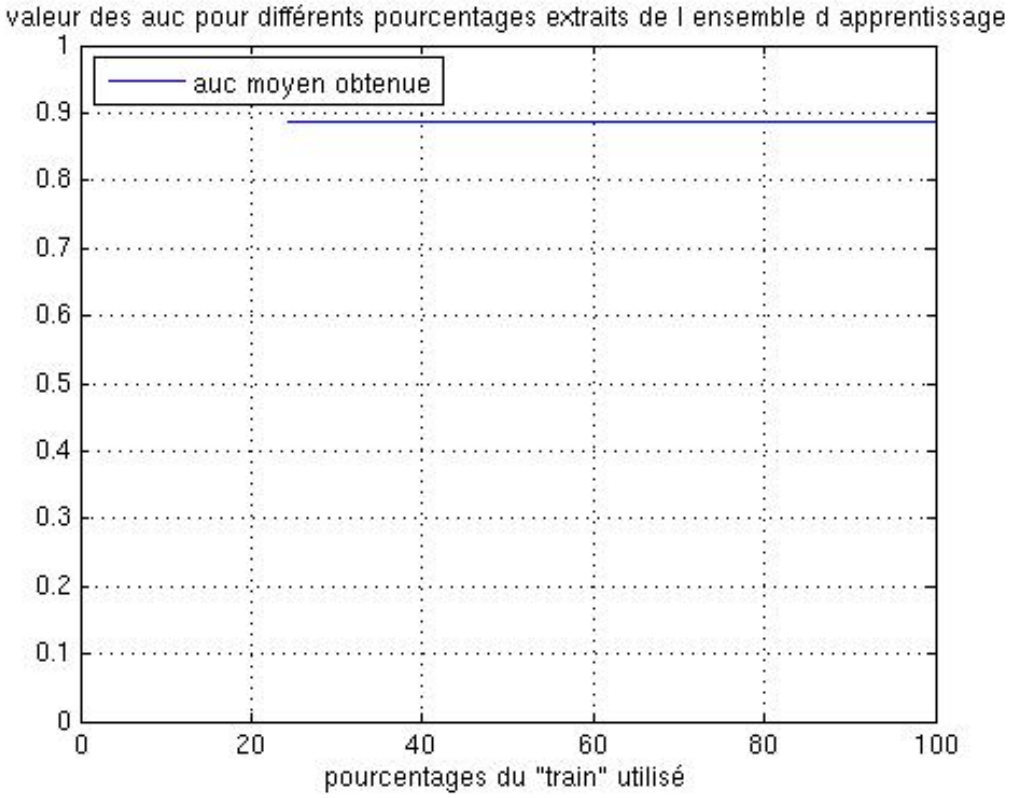


Figure 19 : Valeur des performances quelque soit les pourcentages sur Ionosphere

Influence du choix du Jeu de données

Le choix du jeu de données importe semble t-il dans la répartition des classes de données comme nous le montre l'expérience sur Damier. Les regroupements de ce jeu de données ont eu semble t-il eu un effet très néfaste sur le résultat. Peut être, un apprentissage par clustering pourrait aider à repérer des regroupements comme celui et à corriger la valeur de σ en conséquence.

Iç.4. Discussion

La méthode dite de Schölkopf offre un réglage d'une extrême simplicité. Et pourtant, les résultats obtenus sont bons dans l'ensemble et souvent très bons. L'apprentissage des données par la variance et la dimension semble être concluant. Peut être faudrait-il compléter cet apprentissage par un troisième paramètre tenant compte du regroupement par paquets des données comme Damier nous le suggère.

Il parait d'ores et déjà non simple de trouver une méthode équivalente en efficacité puisque cette méthode cumule performance, rapidité et simplicité.

Χηαπιτρε ς– Réglage du σ à l'aide d'une méthode de régression

ς.1. Introduction

Rôle et attentes vis à vis des expériences

On teste une méthode de prédiction (une fenêtre de Parzen en régression) sur des jeux de données que l'on a démunit d'étiquettes. Cette méthode est ainsi plus réaliste. On espère obtenir de meilleurs résultats avec cette méthode d'obtention qui utilise la fenêtre de Parzen plutôt que une méthode utilisant seulement un résultat brut des données. On espère s'approcher des performances de la fenêtre de Parzen de classification.

ς.2. Implémentation

Particularités de la procédure

On détermine avant l'expérience les paramètres utiles telles que ε, utilisé pour le calcul de l'AOC,σ min et max (cf. Gamme de valeur du paramètre σ - Chapitre II 2.3). Les autres paramètres sont nécessaires mais sont fournis à la procédure au préalable tel que k le paramètre de k – fold cross validation, le nombre de valeurs de σ à examiner, les pourcentages etc.

L'expérience contient 2 étapes : la détermination du σ optimal et le test final utilisant ce paramètre. La détermination du σ optimal utilise la méthode de régression avec la fenêtre de Parzen de classification et l'AOC sa mesure de performance.

Cette première classification est répétée selon 3 boucles. La première permet de choisir à tour de rôle la dimension qui doit servir comme valeur cible. La seconde fait varier σ selon la gamme de σ définie par ces extrema et selon le nombre de σ. La dernière permet le changement des partitions de test et d'apprentissage à l'aide de la k – fold cross validation.

Le σ optimal est la valeur de σ qui minimise l'AOC moyen. L'AOC moyen est une double moyenne de valeurs AOC obtenue pour chaque σ.

Le test final calcule la performance du σ optimal sur la fenêtre de Parzen de Classification et fournit la valeur d'AUC espérée correspondante.

L'expérience est réitérée suivant tous les pourcentages à examiner et le nombre d'itérations demandées. Le nombre d'itérations a pour but de nous assurer que lorsque les pourcentages de l'ensemble d'apprentissage sont petits, que les performances ne soient pas exagérément basses ou hautes selon le choix bons ou mauvais des données extraites. On obtient donc au final plusieurs valeurs d'AUC pour chaque pourcentage de l'ensemble d'apprentissage extrait. Ceci permet d'en calculer une moyenne et un écart type utile à la comparaison de cette méthode par rapport aux autres

Algorithme

Étant donnés:

- le jeu de données sous un formatage particulier
- des pourcentages « n » choisis (pour le tirage des données du train)
- un nombre d'itérations choisis pour chaque pourcentage
- un paramètre k choisi de la k-fold-cross-validation
- un sigma minimal, maximal et un pas de calcul

Lire le jeu de données séparé en 2 ensembles, le premier d'apprentissage 'Train' et l'autre de 'Test'

Normaliser les données.

Déterminer le σ minimal (plus petite distance entre 2 instances) et maximal (plus grande distance avec le barycentre des instances)

Calculer le pas de σ à partir du min et du max et du nombre de valeurs de σ

Déterminer ε, paramètre de l'AOC (en majorant l'écart entre les données et 0)

Pour les pourcentages n choisis**, faire**
Pour j variant de 1 à J, **faire**
Extraire n% de l'ensemble d'apprentissage par un tirage sans remise

Pour toutes les combinaisons possibles des k partitions en 2 partitions
Regrouper les k partitions en 2 /s-ensembles (New_Train%,New_Test%)
avec k-1 partitions pour New_Train% et 1 partition pour New_Test%

Pour σ variant de σ_{min} à σ_{max} (avec un pas σ_{pas}), **faire :**
Pour toutes les dimensions des instances
Enlever une dimension des instances qui devient valeur cible

Évaluer la fenêtre de Parzen de régression
- réglée avec le sigma courant:
- sur les instances de l'ensemble d'apprentissage New_Train%
- pour les instances de l'ensemble de test New_Test%
- avec la dimension choisie comme valeur à estimer
Remplir un tableau de valeur prédite par instance de test

Construire la courbe de REC
Calculer l'aire sous la courbe de REC (AOC)
Fin Pour

Calculer la moyenne des AOC sur les dimensions
Fin Pour
Fin Pour

Évaluer l'AOC moyen et sa variance en fonction de la valeur de sigma

Choisir un sigma optimal qui minimise l'AOC moyen

Évaluer la fenêtre de Parzen de classification
- réglée avec le sigma optimal:
- sur les instances de l'ensemble d'apprentissage: Train
- pour les instances de l'ensemble de test: Test
Remplir un tableau de prédiction par classe et instance de test

Pour toutes les classes
Construire la courbe de ROC
Calculer l'aire sous la courbe de ROC (AUC)
Fin Pour

Calculer l'espérance des AUC (pondérant par la proportion des classes)
Fin Pour
Fin Pour

Obtention d'un AUC moyen et de sa variance pour le choix de sigma optimal (selon la stratégie n°3) et pour chaque pourcentage

Algorithme 10 : Méthode 3 – Obtention du σ optimal à partir d'une méthode de régression

Fenêtre de Parzen de régression

La fenêtre de Parzen de régression: est selon la 'méthode de régression', la méthode utilisée pour déterminer le σ optimal dans le but de la performance de la fenêtre de Parzen de la classification. La fenêtre de Parzen de régression cherche à déterminer directement (c'est une estimation) une valeur dans l'espace des réels. C'est une fonction adaptée au problème de la régression en apprentissage non supervisé. Dans ce cas de figure, on cherche à apprendre de chaque dimension en la désignant comme « ensemble de valeurs cibles ». Comme la classification, ce sont les instances d'apprentissage qui vont permettre le calcul de la valeur cible la plus probable pour une dimension choisie et une instance de test. Contrairement à la classification, n'ayant plus aucune classe à disposition, on ne s'y intéresse plus. Le calcul d'une valeur cible est une moyenne des valeurs d'apprentissage pondérée par les noyaux (distances au sens de Parzen) à calculer pour les instances correspondantes et réduites d'une dimension. L'analyse de performance qui se fera ultérieurement à l'aide de la courbe de REC et l'AOC, opérera directement sur les estimations de valeurs cibles en les comparant avec leurs vraies valeurs.

Algorithme

Étant donnés:
- Les ensembles d'instances de test et d'apprentissage réduits d'une dimension choisie
- Un ensemble de valeur cibles d'apprentissage
- Une valeur choisie pour le paramètre sigma du noyau gaussien

Pour toutes les instances de test**, faire**
    Initialiser le numérateur et le dénominateur de la fenêtre de Parzen

    Pour toutes les instances d'apprentissage**, faire**
        Calculer la distance euclidienne entre instance de test et d'apprentissage

        Calculer la distance au sens de Parzen(Noyau) à l'aide de la distance euclidienne

        Ajouter le produit du Noyau calculé et de la valeur cible d'apprentissage au numérateur.

        Ajouter le Noyau calculé au dénominateur
    Fin Pour

    Calculer le rapport Numérateur et Dénominateur
    Mettre cette valeur dans un tableau des estimations par instance de test
Fin Pour

Algorithme 11 : La Fenêtre de Parzen de Régression

AOC

L'AOC signifie « Area Over the Curve » c'est à dire l'aire au dessus de la courbe, où la courbe désignée est la courbe de REC (Regression Error Characteristics). La courbe de REC est construite à partir des erreurs réalisées par les estimations en régression. Ces erreurs sont classées par ordre croissant puis réparties régulièrement en ordonnée. Chacune des erreurs quantifie, la distance horizontale entre l'emplacement de cette erreur sur l'axe des ordonnées et la courbe de REC. Pour la normalisation, l'axe des ordonnées permet la représentation des erreurs mais uniquement sur le segment entre 0 et 1, donc il n'y a pas à normaliser l'aire selon l'axe des ordonnées. Pour l'axe des abscisses, il y a nécessité de normaliser indépendamment de l'écart maximal des valeurs d'erreurs calculées. Pour cela, pour un jeu de données et avant toutes les expériences de régression nous amenant à utiliser le calcul de l'AOC, on détermine l'epsilon maximal qui correspond à la distance maximale d'une instance par rapport à la moyenne des instances et parmi toutes les instances d'apprentissage. C"est donc ce paramètre qui borne la courbe de REC à droite et c'est par celui-ci également que l'on divise l'AOC afin de la normaliser. Pour mieux comprendre la construction de la courbe de REC, on peut se référer au tutorial fait par Messieurs Bi et Bennett [Bi et al.].

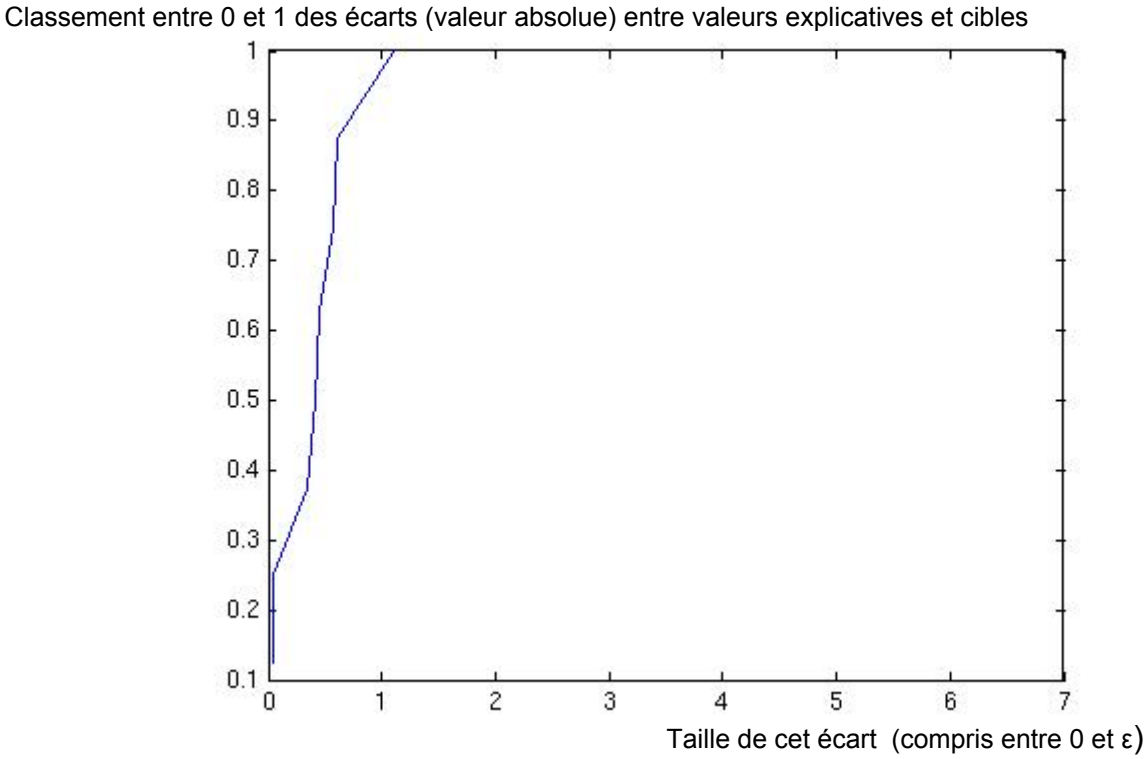


Figure 20 : Exemple de courbe de REC réalisée sur le jeu de données 'Pima'

Algorithme

Étant donnés:
- Un tableau d'estimations des valeurs cibles pour toutes les instances de test
- Les 'vraies' valeurs cibles des instances
- Le paramètre epsilon maximal

Pour toutes les instances de test**, faire**
    Calculer l'erreur comme la valeur absolue de la différence entre vraie valeur cible et estimée

    Ajouter cette valeur au tableau d'erreurs
Fin Pour

Trier le tableau d'erreur dans l'ordre croissant

Commencer à tracer la courbe de REC en partant de l'origine

Initialiser l'aire sous la courbe de REC à 0

Pour le 1er élément de la liste triée puis l'élément suivant jusqu'à la fin de la liste**, faire**
    Mémoriser le point de la courbe où l'on se trouve

    Ajouter l'inverse du nombre d'instance de test à l'ordonnée de la courbe

    Prendre comme abscisse l'erreur correspondant à l'élément courant

    Pour tous les éléments triés suivants ayant la même valeur d'erreur
        Passer à l'élément suivant

        Ajouter l'inverse du nombre d'instance de test à l'ordonnée de la courbe

        Prendre comme abscisse l'erreur correspondant à l'élément courant
    Fin Pour

    Calculer l'aire du trapèze sous segment [point courant de la courbe et point mémorisé]

    Ajouter cette aire à l'aire sous la courbe de REC
Fin Pour

Normaliser l'aire sous la courbe de REC en la divisant par le paramètre epsilon maximal

Calculer l'AOC comme le complémentaire à 1 de l'aire sous la courbe de REC normalisé

Algorithme 12 : l'AOC – Area Over the 'REC' Curve

ç.3. Résultats

La 'méthode 3' doit nous permettre d'obtenir avant tout la validation de la performance de l'expérience en fonction des jeux de données et d'un autre critère : la portion de l'ensemble d'apprentissage utilisée. D'autres résultats intermédiaire sont intéressants tels que la performance de la méthode de prédiction (Fenêtre de Parzen en régression) en fonction de son paramètre σ ou encore le σ optimal toujours selon le jeu de données et la portion d'instances apprises.
Ci-dessous un récapitulatif des trois résultats que l'on s'est proposé de 'récolter' :

'Résultat n°1'

Pour une expérience faite sur un jeu de données, avec un pourcentage de l'ensemble d'apprentissage choisi,
Déterminer la performance des prédictions en régression de la fenêtre de Parzen en régression et ce en fonction de σ.

'Résultat n°2' (σ optimal)

Pour une expérience faite sur un jeu de données, avec un pourcentage de l'ensemble d'apprentissage choisi, et à partir du 'Résultat n°1',
Désigner parmi toutes les valeurs de σ, celui dont la performance est la meilleure.

'Résultat n°3'

Selon les expériences faites sur des jeux de données, pour tous les pourcentages de l'ensemble d'apprentissage choisis, et à partir du 'Résultat n°2',
Déterminer les performances d'une fenêtre de Parzen de classification muni du σ optimal selon la fenêtre de Parzen de régression.

Performance des prédictions en régression

Pourcentage extrait des données	Sigma moyen obtenu	AUC Moyen	Ecart type de l'AUC	Commentaires
	Australian			
100%	0,01716565	0,518804	0,009537	Très petits σ stables et résultat très mauvais
75%	0,01476615	0,514539	0,01168	
50%	0,01476615	0,514539	0,01168	
25%	0,01956516	0,556967	0,123483	
	Damier			
100%	0,22610678	1	0	Petits σ peu stables et résultat qui s'effondre
75%	0,47651772	0,784094	0,295877	
50%	0,41179475	0,813158	0,253776	
25%	1,53038625	0,552119	0,251132	
	Glass			
100%	0,33183887	0,706047	0,001777	Petits σ peu stables et relativement bon résultat stable
75%	0,28334504	0,705295	0,001612	
50%	0,35608578	0,70342	0,007827	
25%	0,49224151	0,712869	0,030689	
	Ionosphere			
100%	0,76409512	0,994505	0	Moyens σ stables et bon résultat stable
75%	0,78558094	0,994505	0	
50%	0,85863271	0,994231	0,000614	
25%	0,39024194	0,96348	0,029035	
	Iris			
100%	0,36492457	0,9975	1,24E-16	Assez petit σ, bon résultat stable
75%	0,36982907	0,9975	1,24E-16	
50%	0,36072072	0,997	0,000456	
25%	0,4097657	0,993667	0,009512	
	Pima			
100%	0,75077515	0,813368	0,003033	Moyens σ assez instable et résultat bon et stable
75%	0,80006109	0,817623	0,001816	
50%	2,70249809	0,83813	0,006176	
25%	0,96599039	0,824858	0,00324	
	Segment			
100%	0,3770078	0,963212	6,74E-05	Moyens σ un peu instables et bon résultat stable
75%	0,41622103	0,963128	0,00012	
50%	0,49464747	0,963077	0,000189	
25%	0,64278632	0,963611	0,000846	
	Sinx3			

100%	**2,40362988**	**0,95898**	**0**	**σ max et bon résultat stable**
75%	**1,67557401**	**0,962403**	**0,006853**	
50%	**2,02155694**	**0,961465**	**0,005557**	
25%	**1,73283683**	**0,965528**	**0,014006**	
	USPS - 300 (au lieu de 1000) valeurs de σ et 4 itérations			
100%	**2,79103061**	**0,933765**	**0**	**Grands σ stables, bon résultat stable**
75%	**2,8506847**	**0,933747**	**2,05E-05**	
50%	**2,91033878**	**0,933729**	**0**	
25%	**2,96999287**	**0,933714**	**1,69E-05**	
	Wine			
100%	**1,00406996**	**0,986919**	**0,00172**	**Assez grands σ et bonne performance**
75%	**1,04981001**	**0,989657**	**0,00195**	
50%	**1,11498957**	**0,992214**	**0**	
25%	**1,20418266**	**0,992858**	**0,000695**	

Optimalité de Paramètre de la fenêtre

La méthode fonctionne et fournit des valeurs de σ convenables mais très diverses, souvent petites et parfois assez grandes. Aucun lien évident n'apparaît entre ces valeurs et celles de la classification. Il apparaît donc parfois des divergences. Celles-ci provoquent des conséquences néfastes sur l'apprentissage de certains jeux de données et aussi parfois selon la variation de la quantité d'instances exploitées.

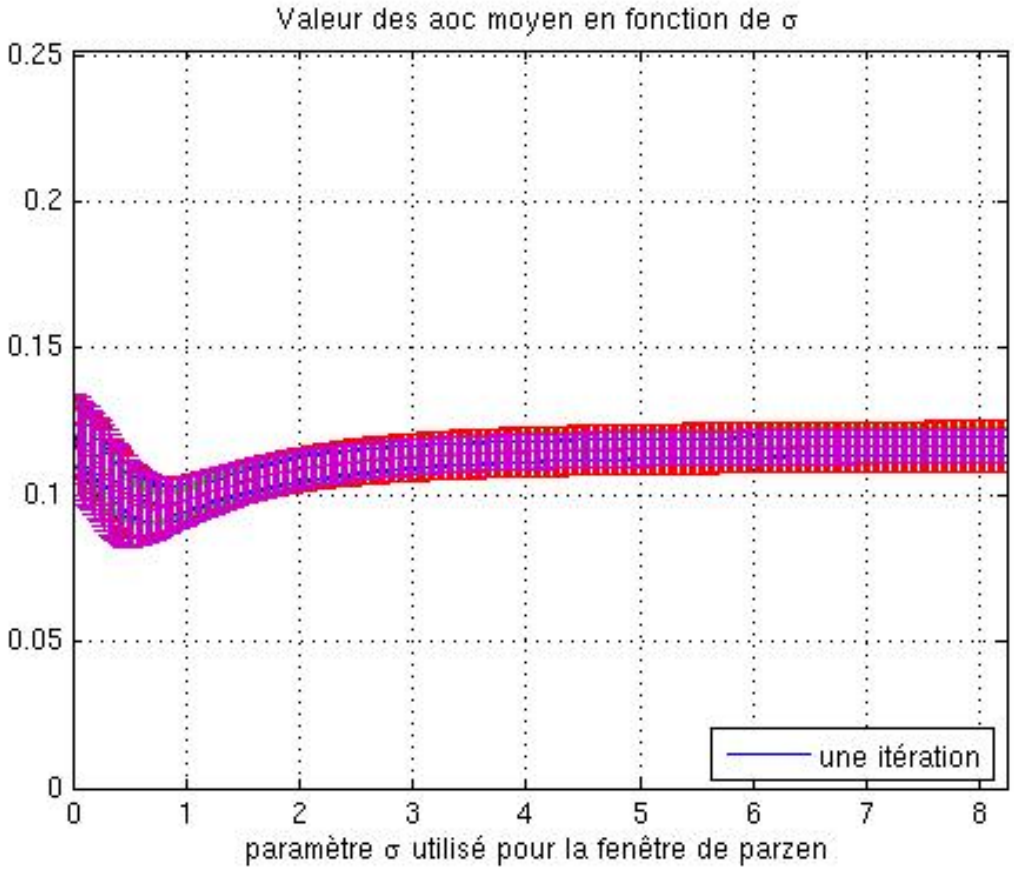


Figure 21 : AOC fonction de σ sur le jeu de données 'Pima'

Sur la figure 19, on observe que l'optimalité du paramètre σ est nettement mise en évidence par la forme de la courbe en cuillère.

Influence de la quantité de données

La diminution du pourcentage de données utilisées joue un rôle parfois positif. En effet, en obtenant de plus petits échantillons, on augmente σ et on arrive à mieux compenser les premiers effets du sur-apprentissage (qui provoque d'après la méthode de classification un effondrement des performances de classification). Ceci n'est probablement vrai que pour les pourcentages choisis et moins pour les très petits pourcentages. Par contre dans d'autres jeux de données comme Damier qui nécessite une toute petite valeur de σ, cette augmentation provoque l'effondrement de la performance de la classification.

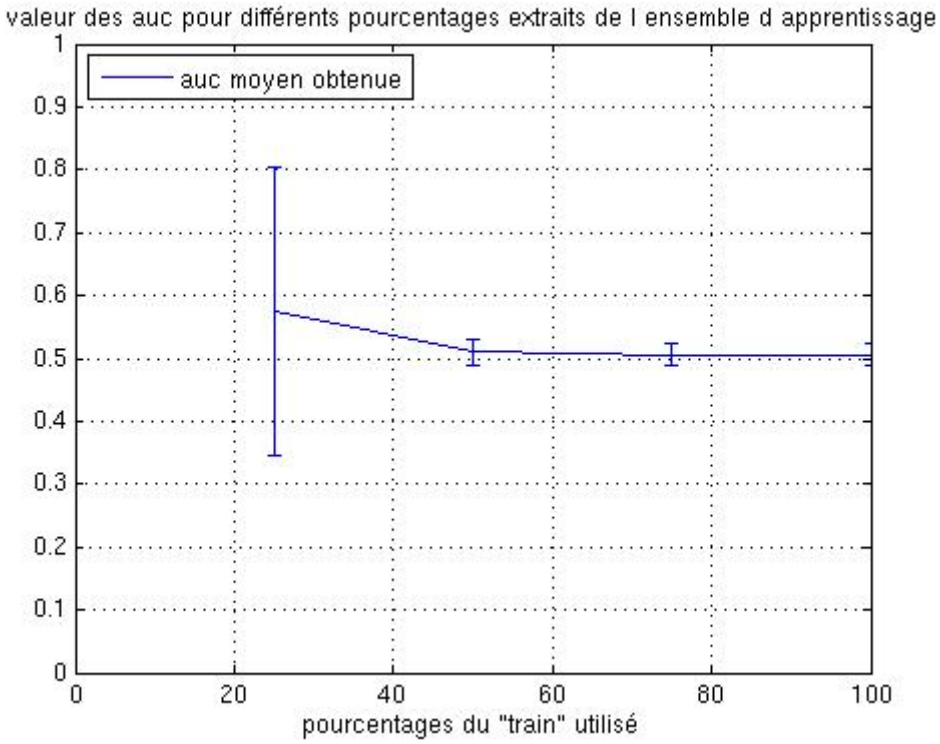


Figure 22 : Performance du réglage sur 'Australian'

Influence du choix du Jeu de données

On observe que dans le cas de jeux de données Australian et Damier, les valeurs de σ ne sont pas optimales ou risque de le devenir avec la diminution de la quantité de données utilisées. Pima et Glass, jeux de données les plus difficiles à apprendre en classification maintiennent bien leurs performances à partir d'un réglage en régression. Les autres jeux de données voient un maintien ou une légère diminution de la performance de leur apprentissage en classification.

Dans les figures 21, 22 et 23 on observe 3 courbes différentes qui s'éloignent de la 'cuillère' obtenue avec Pima. Elles révèlent les difficultés à trouver la valeur de σ qui minimise l'AOC. Dans la figure 23, la courbe est tellement plate que la détermination du σ est quasi – aléatoire à l'intérieur d'une certaine gamme de σ.

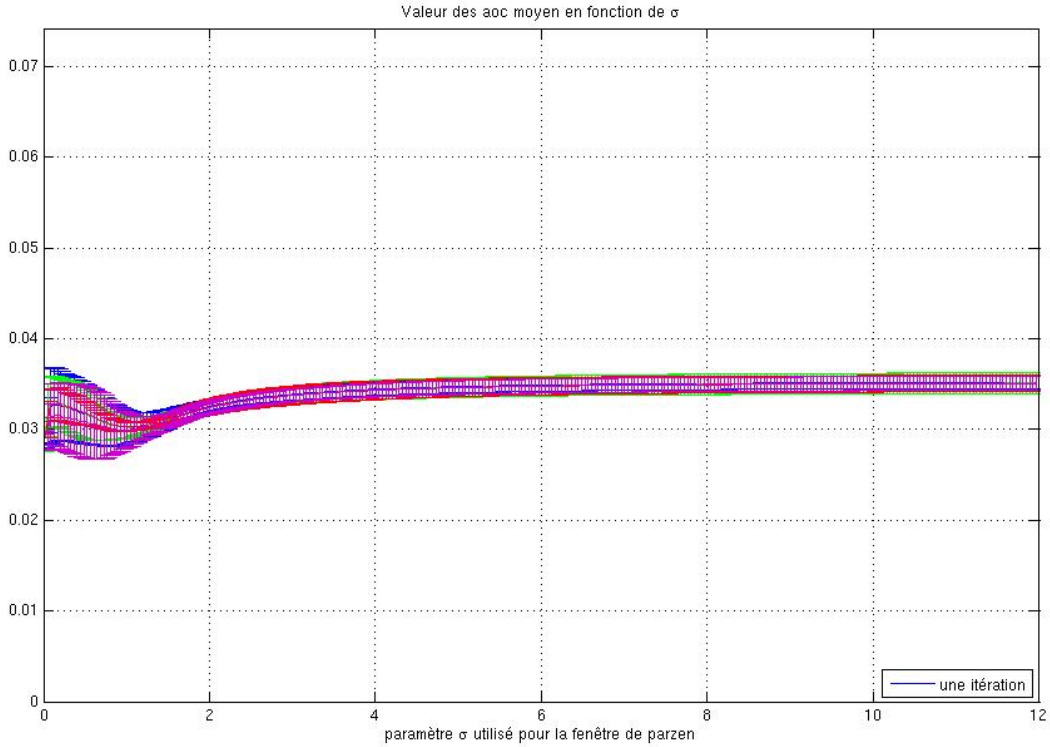


Figure 23 : AOC en fonction de σ sur 'Australian'

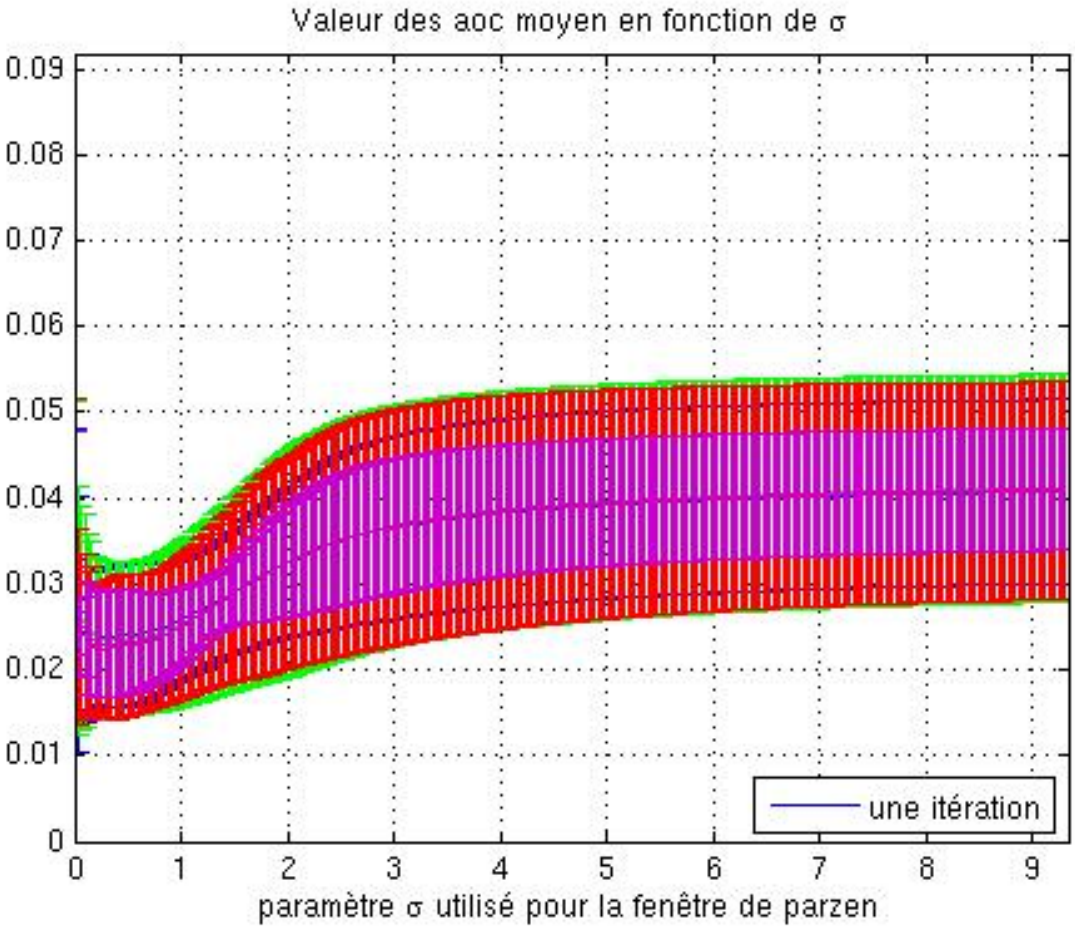


Figure 24 : AOC en fonction de σ sur 'Glass

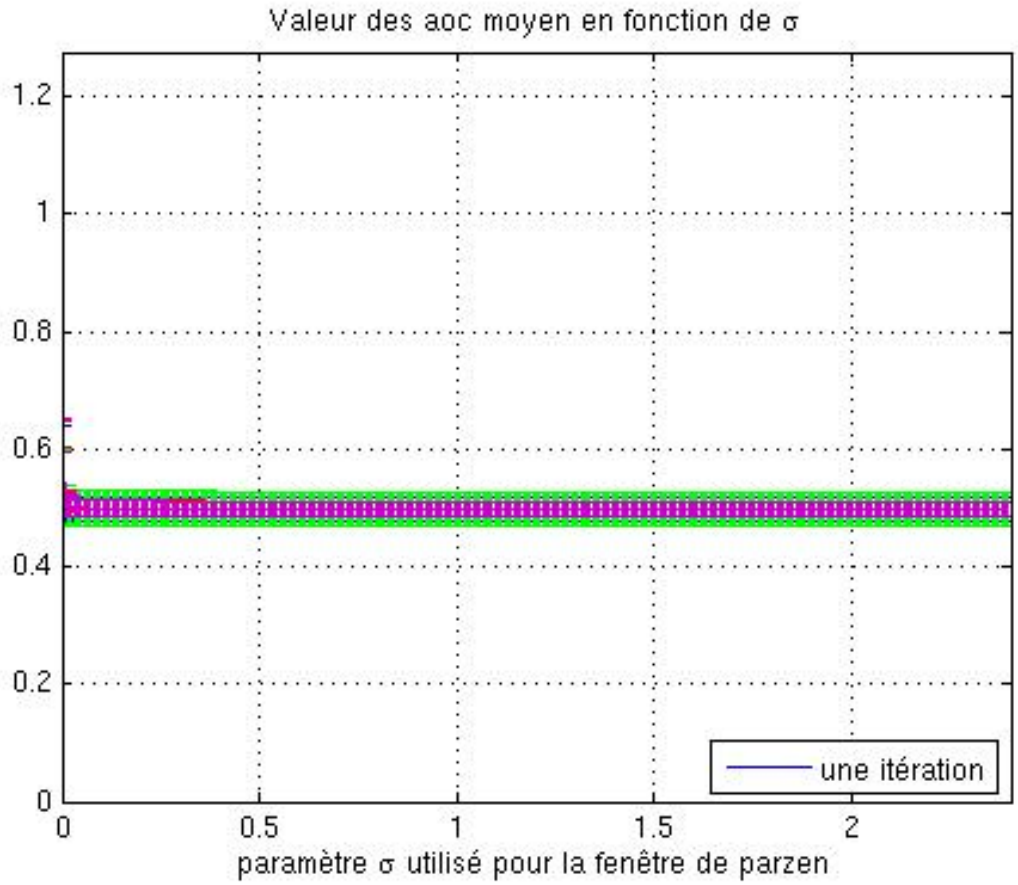


Figure 25 : AOC en fonction de σ sur 'Sin x 3'

ç.4. Discussion

La méthode de régression fournit des valeurs de σ une peu moins optimales pour la méthode de classification. Elle présente de grands inconvénients. Un premier est le fait de ne pas avoir de lien apparent avec la méthode de classification pour la détermination du σ, ce qui s'explique en bonne partie car on ne fait pas du tout le même apprentissage sur les données. Le second est la difficulté que l'on a à mettre en évidence l'optimalité. On observe dans la figure 23, que parfois la méthode de régression ne peut donner aucune précision dans sa détermination de σ car l'optimalité ne correspond plus à une valeur mais à une gamme. Ainsi, les faiblesses de cette méthode viennent de la méthode de classification et de la méthode de régression, elle-même. Ceci nous apprend qu'une méthode de détermination du σ pour la fenêtre de Parzen de classification doit absolument éviter de s'approcher de la gamme de sur-apprentissage d'une part et d'autre part elle doit éviter d'être plus ou moins aléatoire. C'est ce qui arrive parfois avec la méthode de régression lorsque la courbe AOC en fonction de σ forme un graphe plat (figure 23) plutôt qu'une cuillère (figure 19).

Χηαπιτρε ςI- Interprétation des résultats

Les performances et l'évolution de celles-ci avec les pourcentages extraits ont été rassemblées dans le tableau 6 et ce pour la plupart des jeux de données. Elles ont été réalisées dans une expérience exploitant les 3 méthodes avec les mêmes paramètres et 300 valeurs de σ pour la classification et la régression.

Comme prévu, la méthode 1 rassemble globalement les meilleures performances, c'est un apprentissage idéal parce que cette méthode se donne les réponses au problème de classification alors que les classes ne devraient pas y être disponibles. Les performances des méthodes 2 et 3 ne sont pas loin de la méthode 1.

On s'attendait d'abord à une plus grande pertinence de la méthode 3 qui réalise sur les données un apprentissage approfondi. En réalité elle n'est pas beaucoup plus pertinente que la méthode 2 puisque celle-ci ne sait pas toujours proposer un paramètre optimal. Elle est même parfois en mesure de désigner une valeur de σ parmi toute une gamme de valeurs convenables (cf. figure 25). A noter toutefois que la méthode de classification a elle-même régulièrement une grande gamme de valeurs convenables pour σ. Mais ce défaut semble être aggravé dans le cas de la méthode de régression.

Malgré tout sur le plan des performances, on constate que les méthodes 2 et 3 offrent des résultats quasiment équivalents. Les résultats sont souvent bons. Les méthodes subissent le sur-apprentissage et le lissage, phénomènes qui dépendent largement de la répartition des données dans l'espace des données du jeu. La méthode 2 ne subit que le lissage tandis que la méthode 3 subit l'un ou l'autre. Mais le sur-apprentissage est, vu nos expériences plus dangereux que le lissage cf. figure 14 et 17; La méthode de régression prend donc davantage de risques dans l'obtention de σ.

La précision des σ choisi n'est bonne pour aucune des méthodes mais cette tâche est assez difficile. La méthode de classification a en effet aussi cette grande gamme de valeurs convenables pour σ. De toute façon, la précision des σ c'est-à-dire la capacité d'une méthode à fournir le même σ que la méthode de classification importe peu à priori. L'essentiel est la performance et la pertinence des σ choisies.

Etant à performance et pertinence à peu près égal égale, les méthodes 2 et 3 se départagent sur le critère de la rapidité, notre réalisation de la méthode de régression devait contenir 5 boucles de calcul tandis que la méthode 2 n'en contenait que 2. Les expériences de la méthode 2 offrent des résultats rapidement de l'ordre de l'heure au maximum tandis que la méthode 3 peut prendre des semaines sur des grands jeux de données. La méthode 2 assure donc une suprématie sur la méthode 3 malgré une pertinence limitée.

	Méthode 1		Méthode 2		Méthode 3	
	σ (100%→5%)	AUC (100%→ 25%)	σ	AUC	σ (100%→5%)	AUC 100%→25%)
Damier	0,04→0,03	100%→100%	1,41	45,00%	0,24→0,2	100%→100%
Iris	0,5→0,25	100%→100%	2	97,00%	0,35→0,4	100%→100%
Glass	0,4→1,5	70%→72%	3	77,00%	0,33→0,4	70,5→71%
Wine	0,95→0,14	99%→98%	3,61	100,00%	1→1,2	99%→99%
Pima	2,3→4	83%→83%	2,83	83,00%	0,75→1	81%→83%
Australian	2,24→3	93%→91%	3,74	93,00%	0,01→0,01	50%→50%
Segment	0,7→1,2	96%→96%	4,36	94,00%	0,4→0,7	96%→96%
Ionosphere	1,3→1,5	99%→95%	5,83	89,00%	0,75→1	99%→99%
SinX3	0,05→0,09	100%→100%	1,41	96,00%	2,38→2	96%→96%
USPS2	3,1→3	93%→92,5%	16	83,00%	2,9→2,9	93%→93%
Emovoc	5,2→8	91%→91,5%	4,47	91,00%	0,8→0,01	89%→80%

Tableau 6 : Comparaison des performances des méthodes

Conclusion

L'apprentissage statistique à noyaux offre de bonnes possibilités d'apprentissage en particulier avec la fenêtre de Parzen. Cet outil permet quand il est bien réglé de fournir des performances optimales d'apprentissage en classification, en régression, en estimation de densité etc. Mais l'optimalité du réglage n'est pas aisée. La recherche d'une méthode qui fournirait le même apprentissage du paramètre σ que la classification supervisée est maintenant bien entamée. La méthode d'obtention de σ par les variances et la dimension des données parait satisfaisante mais peut être incomplète, car elle ne convient pas à tous les jeux de données (Damier en particulier). Peut être, faudrait-il l'accompagner d'un apprentissage des regroupements des données. En attendant, l'apprentissage actif qui permet une nouvelle forme d'apprentissage plus efficace se voit conforter dans le réglage d'un de ces outils, la fenêtre de Parzen de classification. Par cette étude, on sait que parmi toutes les méthodes testées, la méthode de Schölkopf prévaut pour son réglage.

Annexe : Développement du projet sous Matlab

Langage Matlab

Le langage utilisé est Matlab. C'est un langage interprété qui propose de nombreuses fonctions de haut niveau offrant beaucoup de libertés à l'utilisateur. Par exemple, il possède des fonctions permettant de trier un tableau de valeurs tout en fournissant les indices des éléments correspondant (sort.m) et ce de manière optimisée (algorithme 'quicksort'). Fonction qui a été utile dans le code de l'AOC. D'autres fonctions permettent de gérer très efficacement les entrées et sorties du système permettant par exemple l'export de données dans de multiples formats de fichiers. Un autre avantage, Matlab possède une excellente précision de calcul de l'ordre de $1.\ 10^{-18}$.

Structure du prototype

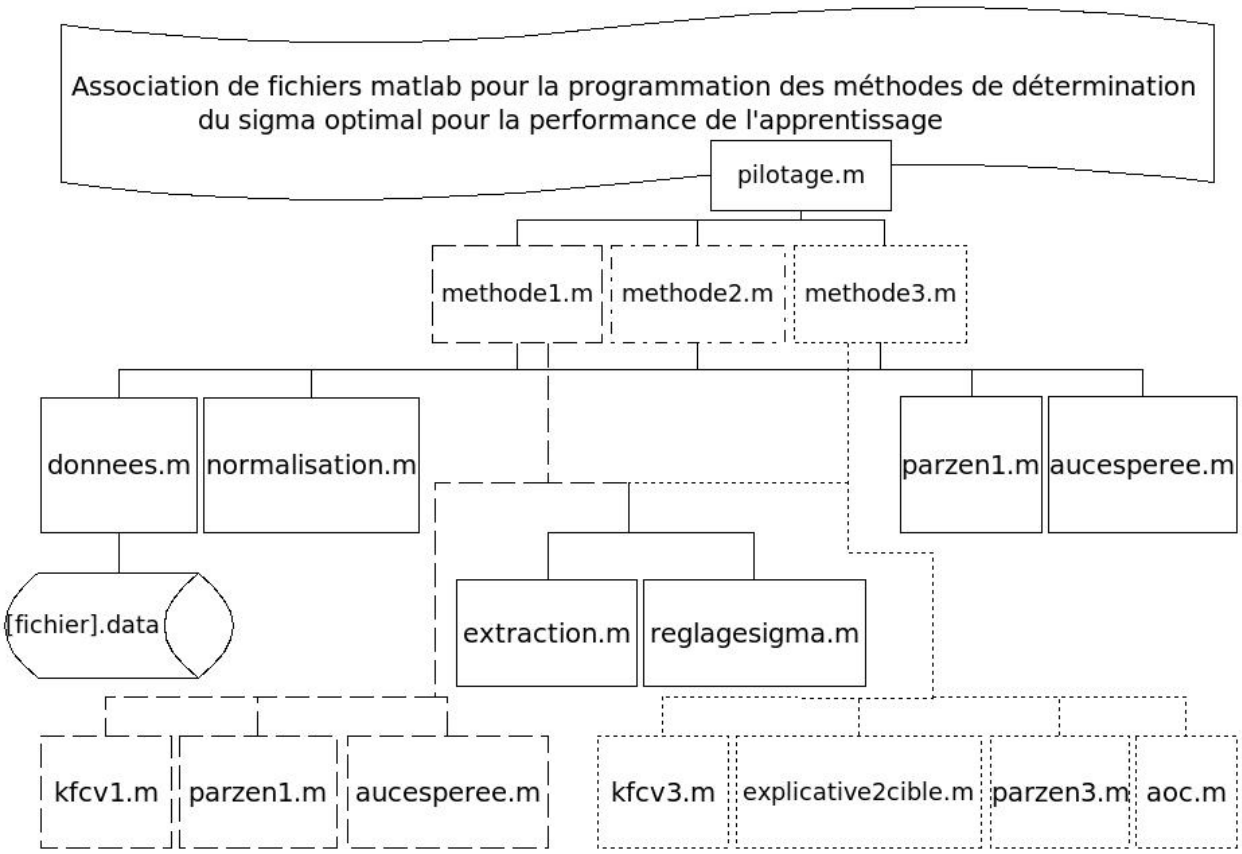


Figure 26 : Structure des fichiers Matlab

Note importante concernant le code

Les fichiers Matlab sont regroupés dans un même dossier Parzen

Conception de la méthode 3

La méthode de régression telle qu'elle a été envisagée est composé de 5 fonctions incontournables qui assurent les calculs de l'expérience sur nos données : à savoir la fenêtre de Parzen en régression, en classification, la construction/le calcul de l'aire sous la courbe de ROC (AUC), la construction/le calcul de l'aire sous la courbe de REC (AOC) et enfin la k-fold cross validation. Ces fonctions ne sont pas seulement des fonctions de traitement des données mais elles permettent également d'extraire de l'information sur les données.

Ces fonctions de premier plan sont accompagnées de fonctions qui permettent seulement un traitement mais les hypothèses qui leur sont faites sont tous aussi significatives pour le sens de l'expérience, et déterminantes pour les résultats. Ces autres fonctions sont l'extraction, le réglage de sigma la normalisation et une fonction qui transforme une dimension de valeur explicative en valeurs cibles. Enfin une ou des fonction(s) devront gérer les entrées et sorties du système permettant une lecture exacte des données et un affichage complet et efficace des résultats.

Commentaires pour le code du fichier pilotage.m

'pilotage.m' est le ficher principal du prototype. C'est de celui-ci que se lancent les expériences. Ce fichier comporte un en-tête destiné à sa présentation, une partie où sont initialisés tous les paramètres des expériences jugés importants une autre qui permet la mise en place d'une interface de dialogue et la réception des volontés de l'utilisateur. Enfin une dernier partie permet l'exécution des méthodes par des appels aux fichiers methode1.m methode2.m et methode3.m comme indiqué dans la figure 26.

```
% Pour réinitialiser l'interface matlab
clc
% Paramètre de l'expérience à définir:
pourcentages =[100,75,50,25];% pourcentage de train utilisé (mettre des entiers)

param_iteration=5; % une valeur ou un vecteur de même taille que pourcentages

% Modes utilisateur
test=0;
sauvegarde_automatique_sans_affichage=1;

k=5; % Paramètre de la k fold cross validation
% Paramètre de la boucle sur sigma.
sigma_auto = 1; % les sigmas max et min seront déterminés automatiquement
sigma_min = 0.5; % Mettre une valeur quelconque dans le cas automatique
sigma_max = 3; % idem
nb_sigma=1000;
```

Figure 27 : pilotage.m (1) Initialisation des paramètres

Les paramètres à régler manuellement sont d'abord les pourcentages utilisés pour toutes les expériences. Ce pourcentage est celui que l'on extrait de l'ensemble d'apprentissage. Il n'est pas possible d'exécuter plusieurs méthodes avec des pourcentages différents entre les méthodes. Bien qu'ils soient quasi inutiles concrètement dans la méthode 2, il est toujours indispensable de mentionner un tableau de valeurs de pourcentage comme ci-dessus. Sa taille peut être variable et les valeurs de pourcentages quelconques. Si un pourcentage très petit est utilisé, le prototype doit fonctionner normalement mais affichera un message d'erreur s'il ne peut plus remplir correctement ses partitions lors de la k-fold cross validation.

'param_iteration' est soit une valeur soit un vecteur de même nature et taille que 'pourcentages'. Lorsqu'il est une valeur, il permet un nombre d'itérations constant pour toutes les valeurs de pourcentages extraits. Sinon il peut être variable, ce qui peut être judicieux si on veut éviter des itérations sur une extraction de 100% qui ne nécessite pas d'itérations (Rappel: le but de l'itération est d'obtenir le résultat de différents tirages pour atténuer les relativiser les tirages trop chanceux ou malchanceux.

2 modes utilisateurs sont disponibles, et sont égaux à 0 si désactivés ou 1 si activés. Le 'test' exécute des fonctions de calculs de performances auctest.m et aoctest.m semblables aux aucesperee.m et AOC. Leur but est d'observer certains paramètres et surtout elle permet la

visualisation des courbes de REC et de ROC ce qui permet d'avoir une idée en préalable sur l'apprentissage d'un jeu de données en classification et en régression. C'est ce mode qui a permis la réalisation des figures 13 et 18 en fin d'expérience. Attention, l'utilisation de ce mode est coûteuse en temps, il vaut mieux la réaliser sur de petits jeux de données sous peine de ne pas arriver à finir une expérience.

Le mode sauvegarde_automatique_sans_affichage doit être de préférence mis à 1. C'est un mode automatique qui permet l'exécution d'un grand nombre d'expérience avec sauvegarde des données au fur et à mesure (graphiques .fig et tableaux .wk1) et ce sans interruption. Par défaut, tout ce qui peut être sauvegardé 'semi automatiquement' par le programme l'ait automatiquement dans ce mode.

Si ce mode est à 0, on opte pour des interruptions et des interrogations des choix de l'utilisateur à chaque fin d'expérience. Les sauvegardes se vont semi automatiquement car l'utilisateur contrôle ce qu'il veut sauvegarder. Dans la fenêtre de dialogue Matlab peuvent être réalisés des sauvegardes. Mais a priori toutes autres extractions de données (copies manuelles du contenu des variables) doit revenir à modifier le code du prototype, ou mieux à n'exécuter qu'une seule méthode à la fois afin de profiter du contenu des variables toujours existantes.

Ensuite, il vient le choix du paramètre de la k-fold cross validation qui doit au minimum égal à 2 et peut être aussi bien pair qu'impair.

Pour finir avec les paramètres, le réglage de la gamme du sigma demande d'abord à l'utilisateur si ce réglage doit se faire automatiquement. Si oui (1), peu importe les valeurs données de sigma_min et sigma_max, car elles ne seront pas pris en compte. Ces valeurs seront réglées dans les méthodes selon les méthodes vues au Chapitre II.

Sigma_min et sigma_max peuvent aussi être réglé manuellement et pour cela sigma_auto devra être égal à 0. La précision du nombre de sigma est obligatoire pour l'expérience.

Le renseignement des paramètres est obligatoire et doit être sauvegardés avant la compilation et l'exécution des expériences. La durée de l'expérience sera a priori proportionnelle avec la valeur de k, le nombre de sigma, et avec le nombre d'itérations. Par contre l'utilisation d'un pourcentage double peut multiplier par quatre la durée d'une expérience munie d'un pourcentage.

```
% Chargement du nombre d'expériences à réaliser
prompt = {'Combien de jeux de données à examiner ?                    '};
dlg_title = 'Expérimentation des performances des méthodes sur des données';
def = {'1'};
num_lines=1;
answer = inputdlg(prompt,dlg_title,num_lines,def);

if(numel(answer)~=0)
   answer2 = str2mat(answer);
   % Initialisation du tableau des méthodes
   methodes = zeros(str2double(answer2),1);
   fichiers='';% Pour initialisation de la chaine de caractère du fichier à
récupérer
    % initialisation du tableau des longueurs des noms de fichiers
   longueurs = zeros(str2double(answer2),1);

   % initialisation du tableau des durées
   durees = zeros(str2double(answer2),1);
```

Figure 28 : pilotage.m (2) Interface graphique et demandes d'ordres

La partie suivante (cf. figure 28) du code de pilotage.m concerne l'interface graphique et la sauvegarde des ordres de l'utilisateur dans un tableau. La 1ère sous partie est une demande à l'utilisateur "Combien de jeux de données à examiner ?". Cette demande doit récupérer un chiffre qui en convertit en chaîne de caractère puis en nombre.

Un 'If' permet de s'assurer que rien ne sera exécuté si aucun élément n'est récupéré. Ils sont alors crées 1 tableau numérique et 1 chaîne de caractère qui permettent la sauvegardes des méthodes et des noms de fichiers qui leurs correspondent. Le but est d'ainsi connaître à l'avance toutes les fichiers à exécuter et sur une méthode (parmi 1,2 ou 3). Le tableau est de la taille du nombre d'expériences à réaliser qui est déjà indiqué par l'utilisateur. La chaîne de caractère est vide au départ et sera réalisé par concaténation. Un tableau 'longueurs' tient une trace de la longueur du nom de fichier. Ceci est d'un intérêt pratique puisque le tableau de noms de fichiers a la largeur du plus grand nom de ces fichiers. Il faut donc sauvegarder la longueur pour extraire correctement un nom de fichier de ce tableau. Un dernier tableau qui est alors crée est un tableau de durées des expériences qui permet à la fin d'une série de dizaines d'expériences de connaître la contribution des expériences dans le temps d'exécution. Ceci afin de mieux gérer le choix des paramètres ou par la suite de programmer des expériences d'une certaine durée.

```
% Boucle sur le nombre d'expériences à réaliser.
 for i=1:str2double(answer2)

      % Procédure de récupération du nom du fichier avec interface.
      d = dir;
      str = {d.name};
      [s,v] = listdlg('PromptString','Select a file:',...
                      'SelectionMode','single',...
                          'ListString',str);
      filename=str{s};

      % Récupération de la longueur du nom  du fichier
      longueurs(i) = numel(filename);

      % Pour concatener verticalement les noms des fichiers
      fichiers=strvcat(fichiers,filename);

      % Choix de la méthode par boîte de dialogue
      prompt = {'Quelle méthode doit être utilisée ?                    '};
      dlg_title = 'Choix de la méthode';
      def = {'1'};
      answer = inputdlg(prompt,dlg_title,num_lines,def);

      % Si il y a eu une réponse, récupérer la méthode et la mettre dans
      % un tableau
      if(numel(answer)~=0)
          answer3 = str2mat(answer);
          methodes(i) = str2double(answer3);
      end
 end
```

Figure 29 : pilotage.m (3) Interface graphique et demandes d'ordres

La Procédure de récupération du nom du fichier et de sa méthode (cf. figure 29) est incluse à l'intérieur d'une boucle qui fait appel à 2 boîtes de dialogue à chaque tour de boucle. Le nom de fichier est d'abord concaténé verticalement à la chaîne de caractère existante (ce qui fait qu'une ligne correspondra bien à un fichier). La longueur du nom de fichier est ajoutée au tableau de longueurs. Puis, le numéro de la méthode est ensuite ajouté au tableau des méthodes

Pour finir (cf. figure 31), une boucle est réalisée sur le nombre d'expériences et exécute en fonction du numéro de méthode, la méthode appropriée avec tous les paramètres nécessaires (un parallélisme a été souhaité et conservé entre les méthodes). Une fois l'exécution terminé, la boucle se termine et un chronomètre qui a été lancé se termine enregistrant dans le tableau de durée, celle de l'expérience terminée.

```
   fichiers % Pour faire apparaitre le nom des fichiers sélectionnées
   methodes % Pour faire apparaître le numéro des méthodes choisies pour chaque
fichier
   for i=1:str2double(answer2)
       tic % Début du chronométrage d'une expérience
       if (methodes(i)==1)
            % exécution de la méthode 1 avec ses paramètres
            methode1(pourcentages,param_iteration,k,sigma_auto,nb_sigma,sigma_m
in,sigma_max,test,sauvegarde_automatique_sans_affichage,fichiers(i,1:longueurs(
i)))
       elseif(methodes(i)==2)
            % exécution de la méthode 2 avec ses paramètres
            methode2(pourcentages,param_iteration,k,sigma_auto,nb_sigma,sigma_m
in,sigma_max,test,sauvegarde_automatique_sans_affichage,fichiers(i,1:longueurs(
i)))
       elseif(methodes(i)==3)
            % execution de la méthode 3 avec ses paramètres
            methode3(pourcentages,param_iteration,k,sigma_auto,nb_sigma,sigma_m
in,sigma_max,test,sauvegarde_automatique_sans_affichage,fichiers(i,1:longueurs(
```

Figure 30: pilotage.m (3) Execution des méthodes

```
       end
       durees(i)=toc; % Fin du chronométrage d'une méthode
       durees % Pour faire apparaître les durées de toutes les méthodes
executées
    end
end
% Affichage d'un signal de fin des expériences
fprintf('fin de toutes les expériences demandées');
```

Figure 31 : pilotage.m (4) Exécution des ordres

Code du fichier methode1.m

```
function []=
methode1(pourcentages,param_iteration,k,sigma_auto,nb_sigma,sigma_min,sigma_max,
test,sauvegarde_automatique_sans_affichage,filename)

% Choix du nombre d'itérations par pourcentage en
% fonction que l'argument fourni est un tableau ou une valeur
if (numel(param_iteration)<numel(pourcentages))
```

```
    iterations = param_iteration*ones(numel(pourcentages));
else
    iterations = param_iteration;
end

% Extraction des données du fichier
[trb,cl,tsb,cs,nb_clas] = donnees(filename);

% recopie du nombre de sigma à tester
b2max = nb_sigma;

% Préparation de la réception des résultats dans 3 tableaux différents.
resultats1 = zeros (b2max,3,(numel(pourcentages)),param_iteration);% Stockage de
toutes les valeurs de sigma, d'auc, de variance
resultats2 = zeros (numel(pourcentages),param_iteration,2); % Stockage des sigma
optimaux et des auc que l'on a obtenues avec.
resultats3 = zeros (numel(pourcentages),2); % stockage des auc et de leur
variance pour chaque pourcentage de train utilisé.

% Préparation du tableau des combinaisons
[listes] = eye(k); % Création du tableau des combinaisons de destinations
possibles 1 parmi k

% Normalisation des données
[tr,ts]=normalisation(trb,tsb);

% si reglages automatiques de sigma_min et sigma_max
if(sigma_auto==1)
    [sigma_min, sigma_max] = reglagesigma(tr)
end

% Préparation de l'expérience
sigma_pas = ((sigma_max-sigma_min)/(nb_sigma));

% 1ère boucle sur les pourcentages du train utilisé pour l'expérience
for i=1:numel(pourcentages),

    pourcentage_en_cours = pourcentages(i) % Pour l'affichage du poucentage en
cours
    moyenneauc=0; % on initialise la moyenne des auc pour chaque pourcentage

    for j=1:iterations(i),% 2nd boucle sur le nombre d'itérations attribué à
chaque pourcentage

        iterations_en_cours = j % Pour l'affichage du numéro d'itération en
cours.
        [tr2,cl2] = extraction(tr,cl,pourcentages(i)); % extraction d'un
pourcentage des ensembles

        % initialisation du maximum des auc moyens à 0
        maxaucmoy = 0;

        % Préparations de la boucle sur les k listes
        b1 = 1;
        b1max1 = size(listes);
        b1max2 = b1max1(1);

        % on initialise tous les sigma de 'résultat' à la valeur de sigma_min
        sigmaxaucmoy = sigma_min;

        % on crée un tableau qui va permettre de recueillir les valeurs
        % d'auc qui sont utiles pour le calcul de la variance.
        tableauauc = zeros (b2max,b1max2);
```

```
        %  Initialisation du tableau de variance permettant le calcul de
        %  l'écart type
        var= zeros (b2max,1);

        while (b1~=(b1max2+1)) % Boucle sur les listes de k partitions

            l = listes(b1,:) % Affichage de la séquence de la k fold cross
validation en cours
            [app_tab, test_tab, app_clas, test_clas] = kfcv1(tr2,cl2,l);

            sigma = sigma_min; % on initialise le sigma au sigma courant

            % Initialisation du paramètre du boucle sur les valeurs de
            % sigma
            b2=1;

            while (b2~=(b2max+1)) % Boucle sur les valeurs de sigma


                % on réalise des prédictions grâce à la fenetre de parzen
                % de classification pour le sigma courant
                [predictions] =
parzen1(app_tab,test_tab,app_clas,nb_clas,sigma);

                % Pour obtenir une visualisation des courbes d'auc, on
                % donne le choix d'executer une fonction auc test

                if (test==1)
                    [auccourant] = auctest(test_clas,predictions);
                else
                    [auccourant] = aucesperee(test_clas,predictions);
                end
                % On remplit à ce stade un tableau de valeurs
                resultats1(b2,1,i,j)=sigma; %Chaque ligne correspond à une
valeur de sigma
                tableauauc(b2,b1)=auccourant;
                resultats1(b2,2,i,j)=resultats1(b2,2,i,j)+auccourant/b1max2; %
on fait la moyenne des k valeurs des auc.

                if (resultats1(b2,2,i,j)>maxaucmoy) % pour tous les sigma, on
teste la moyenne qui se construit
                    maxaucmoy=resultats1(b2,2,i,j); % maximas obtenu pour toutes
les partitions
                    sigmaxaucmoy=sigma;
                end
                sigma=sigma+sigma_pas;

                b2=b2+1;
            end % Fin de la boucle sur les valeurs de sigma

            b1=b1+1;

        end

        % Création du tableau des variances

        % Préparation de la boucle sur les listes de k partitions
        b1=1;
        while (b1~=(b1max2+1)) % Boucle sur les listes de k partitions

            % Préparation de la boucle sur les valeurs de sigma
            b2=1;
            while (b2~=(b2max+1)) % Nouvelle boucle sur les valeurs de sigma
afin de calculer la variance des auc sur les k partitions
```

```
                var(b2)= var(b2) + ((resultats1(b2,2,i,j)-
tableauauc(b2,b1)).^2)/b1max2; % calcul de la variance à partir de la moyenne

                b2=b2+1;
            end
            b1=b1+1;
        end

        % Création du tableau des écarts types

        % Préparation de la boucle sur les valeurs de sigma
        b2=1;

        while (b2~=(b2max+1)) % Nouvelle boucle sur les valeurs de sigma afin de
calculer l'écart type

            resultats1(b2,3,i,j) = sqrt(var(b2));% calcul de l'écart type à
partir de la variance

            b2=b2+1;
        end
        resultats2(i,j,1)=sigmaxaucmoy; % Pour chaque itérations, on enregistre
le sigma optimal obtenu

        [predictions] = parzen1(tr,ts,cl,nb_clas,sigmaxaucmoy); % on execute la
fenêtre de parzen de classification réglée
                                                                % à l'aide du
sigma optimal

        [auctestfinal] = aucesperee(cs,predictions); % on mesure la performance
des prédictions obtenues grâce à l'auc

        resultats2(i,j,2)=auctestfinal;% enregistrement de cette valeur d'auc
dans un tableau d'auc associé à son sigma optimal
        moyenneauc = moyenneauc + auctestfinal/iterations(i);% calcul de
                        % la moyenne des auc pour le pourcentage courant
        %waitbar(((i-1)*iterations(i)+j)/(numel(pourcentages)*param_iteration))
    end

    % Calcul de la variance des données selon les itérations

    var2=0; % initialisation de la variance attribué à chaque pourcentage.

    for j=1:iterations(i),% 2nd boucle sur le nombre d'itérations attribué à
chaque pourcentage

        var2= var2 + ((resultats2(i,j,2)-moyenneauc).^2)/iterations(i);% calcul
de la variance à partir de la moyenne

    end

    resultats3(i,1)=moyenneauc; % enregistrement de la moyenne des auc pour
chaque pourcentage
    resultats3(i,2)=sqrt(var2); % enregistrement de la variance des auc pour
chaque pourcentage

end
```

Code du fichier methode2.m

```
function []=
methode2(pourcentages,param_iteration,k,sigma_auto,nb_sigma,sigma_min,sigma_max,
test,sauvegarde_automatique_sans_affichage,filename)
```

```
% Choix du nombre d'itérations par pourcentage en
% fonction que l'argument fourni est un tableau ou une valeur
if (numel(param_iteration)<numel(pourcentages))
    iterations = param_iteration*ones(numel(pourcentages));
else
    iterations = param_iteration;
end

% Extraction des données du fichier
[trb,cl,tsb,cs,nb_clas] = donnees(filename);

% recopie du nombre de sigma à tester
b2max = nb_sigma;

% Préparation de la réception des résultats dans 2 tableaux différents.
resultats2 = zeros (numel(pourcentages),param_iteration,2); % Stockage de racine
de d et des auc que l'on a obtenues avec.
resultats3 = zeros (numel(pourcentages),2); % stockage des auc et de leur
variance pour chaque pourcentage de train utilisé.

% Normalisation des données
[tr,ts]=normalisation(trb,tsb);

% si reglages automatiques de sigma_min et sigma_max
if(sigma_auto)
    [sigma_min, sigma_max] = reglagesigma(tr);
end

% Préparation de l'expérience
sigma_pas = (sigma_max-sigma_min)/(nb_sigma);

% Détermination du sigma comme la racine carrée de la dimension des données
d1 = size(tr);
sigma = sqrt(d1(2));

% 1ère boucle sur les pourcentages du train utilisé pour l'expérience
for i=1:numel(pourcentages),

    pourcentage_en_cours = 100 % Pour l'affichage du poucentage en cours

    moyenneauc=0; % on initialise la moyenne des auc pour chaque pourcentage

    for j=1:iterations(i),% 2nd boucle sur le nombre d'itérations attribué à
chaque pourcentage

        iterations_en_cours = j % Pour l'affichage de l'itération en cours

        [tr2,cl2] = extraction(tr,cl,100) ; % extraction de 100% des ensembles

        % on enregistre la valeur de sigma testée
        resultats2(i,j,1)=sigma;

        % on réalise une prédiction
        [predictions] = parzen1(tr2,ts,cl2,nb_clas,sigma);

        if (test==1)
             [auctestfinal] = auctest(cs,predictions);
        else
             [auctestfinal] = aucesperee(cs,predictions);
        end

        resultats2(i,j,2)=auctestfinal;% enregistrement de cette valeur d'auc
dans un tableau d'auc associé à son sigma optimal
```

```
        moyenneauc = moyenneauc + auctestfinal/iterations(i);% calcul de
        % la moyenne des auc pour le pourcentage courant

        % waitbar(((i-1)*iterations(i)+j)/(numel(pourcentages)*param_iteration))
    end

    var2=0; % initialisation de la variance attribué à chaque pourcentage.

    for j=1:iterations(i),% 2nd boucle sur le nombre d'itérations attribué à
chaque pourcentage

        var2= var2 + ((resultats2(i,j,2)-moyenneauc).^2)/iterations(i);% calcul
de la variance à partir de la moyenne

    end

    resultats3(i,1)=moyenneauc; % enregistrement de la moyenne des auc pour
chaque pourcentage
    resultats3(i,2)=sqrt(var2); % enregistrement de la variance des auc pour
chaque pourcentage

end
```

Code du fichier methode3.m

```
function []=
methode3(pourcentages,param_iteration,k,sigma_auto,nb_sigma,sigma_min,sigma_max,
test,sauvegarde_automatique_sans_affichage,filename)

% Choix du nombre d'itérations par pourcentage en
% fonction que l'argument fourni est un tableau ou une valeur
if (numel(param_iteration)<numel(pourcentages))
    iterations = param_iteration*ones(numel(pourcentages));
else
    iterations = param_iteration;
end

% Extraction des données du fichier
[trb,cl,tsb,cs,nb_clas] = donnees(filename);

% recopie du nombre de sigma à tester
b2max = nb_sigma;

% Préparation de la réception des résultats dans 3 tableaux différents.
resultats1 = zeros (b2max,3,(numel(pourcentages)),param_iteration);% Stockage de
toutes les valeurs de sigma, d'auc, de variance
resultats2 = zeros (numel(pourcentages),param_iteration,2); % Stockage des sigma
optimaux et des auc que l'on a obtenues avec.
resultats3 = zeros (numel(pourcentages),2); % stockage des auc et de leur
variance pour chaque pourcentage de train utilisé.

% Préparation du tableau des combinaisons
[listes] = eye(k); % Création du tableau des combinaisons de destinations
possibles 1 parmi k

% Normalisation des données
[tr,ts]=normalisation(trb,tsb);

% si reglages automatiques de sigma_min et sigma_max
if(sigma_auto)
    [sigma_min, sigma_max] = reglagesigma(tr);
end
```

```
% Préparation de l'expérience
sigma_pas = (sigma_max-sigma_min)/(nb_sigma);

% Détermination du paramètre epsilon chapeau nécessaire pour l'aoc, la méthode
consiste à
% trouver l'écart le plus grand avec un mauvais estimateur. qui donnerait
% tout le temps 0.
epsilon = max(max(max(max(tr)),max(max(ts))),-min(min(min(tr)),min(min(ts))))

% 1ère boucle sur les pourcentages du train utilisé pour l'expérience
for i=1:numel(pourcentages),

    pourcentage_en_cours = pourcentages(i) % Pour l'affichage du poucentage en
cours
    moyenneauc=0; % on initialise la moyenne des auc pour chaque pourcentage

    for j=1:iterations(i),% 2nd boucle sur le nombre d'itérations attribué à
chaque pourcentage

        iterations_en_cours = j % Pour l'affichage du numéro d'itération en
cours.
        [tr2,cl2] = extraction(tr,cl,pourcentages(i)) ; % extraction d'un
pourcentage des ensembles

        % Préparations de la boucle sur les k listes
        b1 = 1;
        b1max1 = size(listes);
        b1max2 = b1max1(1);

        % on initialise tous les sigma de 'résultat' à la valeur de sigma_min
        sigminaoc = sigma_min;

        % on crée un tableau qui va permettre de recueillir les valeurs
        % d'auc qui sont utiles pour le calcul de la variance.
        tableauaoc = zeros (b2max,b1max2);

        %  Initialisation du tableau de variance permettant le calcul de
        %  l'écart type
        var= zeros (b2max,1);

        while (b1~=(b1max2+1)) % Boucle sur les listes de k partitions

            l = listes(b1,:) % Pour l'affichage de la séquence des partitions
utilisées

            %Regroupement des ensembles de test et d'apprentissage en fonction
de la séquence des partitions utilisées
            [app_tab, test_tab] = kfcv3(tr2,l);

            sigma = sigma_min; % on initialise le sigma courant

            % Préparationde la boucle sur les valeurs de sigma
            b2=1;

            while (b2~=(b2max+1)) % Boucle sur les valeurs de sigma
                % Préparation de la boucle sur chacune des dimensions des
valeurs
                % explicatives
                b3=1;
                b3max1=size(app_tab);
                b3max2=b3max1(2);

                % Initialisation de l'aoc courant
                aoccourant = 0;
```

```
                while (b3~=(b3max2+1))

                    % on réduit les valeurs explicatives d'une dimension
                    % qui devient l'ensemble des valeurs cibles
                    [app_tab_reduit, test_tab_reduit, app_valeurs_cibles,
test_valeurs_cibles] = explicative2cible(app_tab,test_tab,b3);
                    % on réalise des prédictions grâce à la fenetre de parzen
                    % de régression pour le sigma courant
                    [estimations] =
parzen3(app_tab_reduit,test_tab_reduit,app_valeurs_cibles,sigma);

                    % Pour obtenir une visualisation des courbes d'aoc, on
                    % donne le choix d'executer une fonction 'aoctest'

                    if (test==1)
                        [valeur_aoc] =
aoctest(test_valeurs_cibles,estimations,epsilon);
                    else
                        [valeur_aoc] =
aoc(test_valeurs_cibles,estimations,epsilon);
                    end
                    aoccourant=aoccourant+valeur_aoc/b3max2;% calcul de l'aoc
moyen à partir des aoc.

                    b3=b3+1;
                end

                % réalisation d'un tableau de valeur comparatives
                resultats1(b2,1,i,j)=sigma;% pour un sigma donnée
                tableauaoc(b2,b1)=aoccourant;
                resultats1(b2,2,i,j)=resultats1(b2,2,i,j)+aoccourant/b1max2; %
on fait la moyenne des k valeurs des a0c.

                sigma=sigma+sigma_pas;
                b2=b2+1;
            end % Fin de la boucle sur les valeurs de sigma

            b1=b1+1;

        end

        minaoc = 1;
        b1=1;
        while (b1~=(b1max2+1)) % Boucle sur les listes de k partitions
            b2=1;
            while (b2~=(b2max+1)) % Nouvelle boucle sur les valeurs de sigma
afin de calculer la variance des aoc sur les k partitions
                var(b2)= var(b2) + ((resultats1(b2,2,i,j)-
tableauaoc(b2,b1)).^2)/b1max2; % calcul de la variance à partir de la moyenne
                if (resultats1(b2,2,i,j)<minaoc)
                    minaoc=resultats1(b2,2,i,j); % minimas obtenu pour la
partition en cours
                    sigminaoc=resultats1(b2,1,i,j);
                end
                b2=b2+1;
            end
            b1=b1+1;
        end

        b2=1;

        while (b2~=(b2max+1)) % Nouvelle boucle sur les valeurs de sigma afin de
calculer l'écart type
```

```
            resultats1(b2,3,i,j) = sqrt(var(b2));% calcul de l'écart type à
partir de la variance

            b2=b2+1;
        end
        resultats2(i,j,1)=sigminaoc;
        [predictions] = parzen1(tr,ts,cl,nb_clas,sigminaoc);
        [auctestfinal] = aucesperee(cs,predictions);
        resultats2(i,j,2)=auctestfinal;% enregistrement de cette valeur d'auc
dans un tableau d'auc associé à son sigma optimal
        moyenneauc = moyenneauc + auctestfinal/iterations(i);% calcul de
                        % la moyenne des auc pour le pourcentage courant
        % waitbar(((i-1)*iterations(i)+j)/(numel(pourcentages)*param_iteration))
    end

    var2=0; % initialisation de la variance attribué à chaque pourcentage.

    for j=1:iterations(i),% 2nd boucle sur le nombre d'itérations attribué à
chaque pourcentage

        var2= var2 + ((resultats2(i,j,2)-moyenneauc).^2)/iterations(i);% calcul
de la variance à partir de la moyenne

    end

    resultats3(i,1)=moyenneauc; % enregistrement de la moyenne des auc pour
chaque pourcentage
    resultats3(i,2)=sqrt(var2); % enregistrement de la variance des auc pour
chaque pourcentage

end
```

Code du fichier donnees.m

```
function [trb,cl,tsb,cs,nb_clas] = donnees(filename)
fid=0;

[fid, message] = fopen(filename, 'rt'); % fid est le 'file identifier'
if fid ==-1
    disp(message)
else
    fgetl(fid);
    fgets(fid,24);
    dim = fscanf(fid,'%g');
    fgets(fid,18);
    nb_clas = fscanf(fid,'%g');
    fgets(fid,34);
    nb_train = fscanf(fid,'%g');
    fgets(fid,26);
    nb_test = fscanf(fid,'%g');
    fgetl(fid);

    element = fscanf(fid,'%g');

    i=1;
    while (i~=(nb_train*(dim+1)+1)) % boucle sur l'ensemble des élements.
        j=1;
        while(j~= (dim+2)) % boucle sur chaque ligne
            if(mod(j,dim+1))
                trb((i-1)/(dim+1)+1,j)=element(i+j-1);
            else
                cl((i-1)/(dim+1)+1)=element(i+j-1)+1;
```

```
            end
            j=j+1;
        end
            i=i+dim+1;
    end
    fgetl(fid);
    element2 = fscanf(fid,'%g');
    fclose(fid);
    i=1;
    while (i~=(nb_test*(dim+1)+1))
        j=1;
        while(j~= (dim+2))
            if(mod(j,dim+1))
                tsb((i-1)/(dim+1)+1,j)=element2(i+j-1);
            else
                cs((i-1)/(dim+1)+1)=element2(i+j-1)+1; % Ajout de un pour
prévenir le décalage des classes.
            end
            j=j+1;
        end
        i=i+(dim+1);
    end
end
```

Code du fichier normalisation.m

```
function [app_tab, test_tab] = normalisation(app_tab, test_tab)

% Préparation de la boucle sur les dimensions
b1=1; % Paramètre de boucle
b1max1=size(app_tab);
b1max2=b1max1(2); % Obtention du nombre de dimensions

while (b1~=(b1max2+1)) % boucle sur les dimensions
    % Initialisation à 0 de la variance et de la moyenne pour chaque dimension
    moy=0;
    var=0;

    % Préparation de la boucle sur les instances de l'ensemble
    % d'apprentissage.
    b2=1; % Paramètre de boucle
    b2max1=size(app_tab);
    b2max2=b2max1(1); % Obtention du nombre d'éléments dans l'ensemble
d'apprentissage
    while (b2~=(b2max2+1)) % 1ère boucle sur les instances pour calculer la
moyenne
        moy = moy + app_tab(b2,b1)/b2max2; % incrémentation de la moyenne
        b2=b2+1; % incrémentation du paramètre de boucle
    end % While - fin de la boucle sur les instances d'apprentissage

    b2=1; % réinitialisation du paramètre de boucle
    while (b2~=(b2max2+1)) % 2ème boucle sur les instances pour calculer la
variance
        var= var + ((app_tab(b2,b1)-moy).^2)/b2max2; % incrémentation de la
variance
        b2=b2+1; % incrémentation du paramètre de boucle
    end % while - fin de la boucle sur les instances d'apprentissage

    b2=1; % réinitialisation du paramètre de boucle
    while (b2~=(b2max2+1)) % 3ème boucle sur les instances d'apprentissage pour
les normaliser
        if(var~=0)
```

```
            app_tab(b2,b1) = (app_tab(b2,b1)-moy)/(sqrt(var)); % normalisation
        end
        b2=b2+1; % incrémentation du paramètre de boucle
    end % while - fin de la 3ème boucle sur les instances d'apprentissage

    % Préparation de la boucle sur les instances de l'ensemble de test
    b3=1; % Initialisation du paramètre de boucle
    b3max1=size(test_tab);
    b3max2=b3max1(1); % Obtention du nombre d'éléments dans l'ensemble de test

    while(b3~=(b3max2+1)) % boucle sur les instances de test pour les normaliser
      if(var~=0)
            test_tab(b3,b1)=(test_tab(b3,b1)-moy)/sqrt(var); % normalisation
        end
        b3=b3+1; % incrémentation du paramètre de boucle
    end % while - fin la boucle sur les instances de test

    b1=b1+1; % incrémentation du paramètre de boucle
end % while - fin de la boucle sur les dimensions
```

Code du fichier parzen1.m

```
function [predictions] =
parzen1(ens_train,ens_test,classes_du_train,nbclasse,sigma)

% calcul du sigma au carré
sigma_carre = sigma.^2;

% préparation de la première boucle sur l'ensemble de test
b1=1; % Paramètre de boucle
b1max1=(size(ens_test));
b1max2=b1max1(1); % taille de l'ensemble de test en nombre d'instances.

% initialisation du tableau de prédiction: une valeur pour chaque couple
% d'instances de test et de classe
predictions = zeros(b1max2,nbclasse);

while (b1~=(b1max2+1)) % boucle sur l'ensemble de test
    % fprintf('ligne de test') résultat intermédiaire qui peut être affiché
    % ens_test(b1,:) résultat intermédiaire qui peut être affiché
    b2=1; % Paramètre de boucle

    while(b2~=(nbclasse+1)) % boucle sur les classes
        % nbclasse(b2); résultat intermédiaire qui peut être affiché
        rn = 0 ;
        rd = 0 ;
        % Préparation de la boucle sur l'ensemble d'apprentissage
        b3=1; % paramètre de boucle
        b3max1=(size(ens_train));
        b3max2=b3max1(1); % Obtention du nombre d'instance de l'ensemble
d'apprentissage

        while(b3~=b3max2+1) % boucle sur l'ensemble d'apprentissage
            % ens_train(b3,:) résultat intermédiaire qui peut être affiché
            normeL2 = norm(ens_train(b3,:) - ens_test(b1,:)); % norme L2
            dist_parzen = exp(-(normeL2.^2)./(2*sigma_carre)); % kernel/distance
au sens de parzen

            if(b2 == classes_du_train(b3)) % si égalité entre la classe courante
et la classe de l'instance d'apprentissage
                rn = rn + dist_parzen; % alors ajout de la distance au sens de
parzen au numérateur du résultat
```

```
            end % if - égalité entre les 2 classes

            rd = rd + dist_parzen; % ajout quelque soit l'élément de l'ensemble
d'apprentissage
                                   % de sa distance au sens de parzen au
dénominateur du résultat

            b3 = b3 + 1;
            % r=rn/rd  résultat intermédiaire qui peut être affiché
        end % while - fin de la boucle sur l'ensemble d'apprentissage
        if(rd==0)
            rd=eps;
        end
        predictions(b1,b2) = rn/rd; % enregistrement de la valeur obtenue dans
le tableau

        b2 = b2+1;
    end % while - fin de la boucle sur les classes

    b1=b1+1;
end % while - fin de la boucle sur l'ensemble de test.
```

Code du fichier aucesperee.m

```
function [auc] = aucesperee(classe_du_test,predictions)

% Préparation de la boucle sur les classes
b1=1; % Paramètre de boucle
b1max1 = size(predictions);
b1max2 = b1max1(2); % nombre de classes différentes

% Initialisation des résultats
auc =0 ;

while (b1~=(b1max2+1)) % bouble sur les classes

    [predictions_ordre_croissant,indice_prediction]= sort(predictions(:,b1)); %
Tri des données dans l'ordre croissant

    % Préparation de la boucle décroissante sur les prévisions
    b2max1=size(indice_prediction);
    b2max2=b2max1(1);
    b2=b2max2; % paramètre de boucle

    % Initialisation des paramètres de la courbe de roc dont on calcule l'aire
    x=0; % abscisse
    y=0; % ordonnée
    int=0; % aire sous la courbe de roc (calculé par la méthode des trapèzes)

    while (b2~=0) % Boucle sur l'ensemble de test
        xo=x;% trace de l'abscisse pour le calcul de l'aire du trapèze courant
        yo=y;% trace de l'ordonnée pour le calcul de l'aire du trapèze courant

        if(b1==classe_du_test(indice_prediction(b2))) % vérification de la
prédiction
            y=y+1; % incrémentation de l'ordonnée en cas de bonne prédiction
        else
            x=x+1; % incrémentation de l'abcisse en cas de mauvaise prédiction
        end % if - si bonne prédiction alors... sinon ...

        while (((b2-
1)~=0)&&(predictions_ordre_croissant(b2)==predictions_ordre_croissant(b2-1)))
```

```
                                        %  Boucle en cas d'égalité sur les
prédictions

             if(b1==classe_du_test(indice_prediction(b2-1))) % vérification de
la prédiction
                 y=y+1; % incrémentation de l'ordonnée en cas de bonne
prédiction
             else
                 x=x+1; % incrémentation de l'abscisse en cas de mauvaise
prédiction
             end % if - si bonne prédiction alors... sinon...

            b2 = b2 -1;
        end % While - Fin de la boucle en cas d'égalité sur les prédictions

        int=int+(x-xo)*(y+yo)/2; % ajout de l'aire du trapèze
        b2=b2-1;
    end % While - Fin de la boucle sur l'ensemble de test

    if((x*y)~=0) % vérification que l'aire du graphe n'est pas nulle
        int=int/(x*y); % normalisation de l'aire sous la courbe de roc
    end % if
    if(x==0)
        int=1;
    end %if
    % Préparation de la boucle sur les classes de test
    b4=1; % Paramètre de boucle sur l'ensemble des classes de test
    b4max1=size(classe_du_test);
    b4max2=b4max1(2); % Obtention de la taille de l'ensemble des classes de test

    proba_classe=0; % initialisation de la probabilité de la classe courante

    while (b4~=(b4max2+1)) % boucle sur les classes de test

        if (classe_du_test(b4)==b1) % si la classe de test est égale à la classe
courante
            proba_classe=proba_classe+1/b4max2; % incrémentation de la
probabilité de la classe courante
        end % if - si égalité des classes alors... sinon...

        b4=b4+1;
    end % While - Boucle sur les classes de test

    auc=auc + int*proba_classe; % calcul de l'espérance des AUC
                                % (pondération par la probabilité des
                                % classes de l'ensemble de test)
    b1=b1+1;
end % While - Fin de la boucle sur les classes
```

Code du fichier extraction.m

```
function [app_tab_reduit, app_clas_reduit] = extraction(app_tab, app_clas,
pourcentage)

% Préparation de la boucle sur les tirages de l'ensemble d'apprentissage.
b1=1; % Paramètre de boucle

nb_instance1=size(app_tab);
nb_instance=nb_instance1(1); % Obtention du nombre d'instances dans l'ensemble
d'apprentissage
tirage = floor(nb_instance*pourcentage/100);
controle = zeros(nb_instance);
```

```
while(b1~=(tirage+1))
    ind = floor(rand*nb_instance)+1;
    if(controle(ind)==0)

        % Préparation de la boucle sur les dimensions des instances
        b2 = 1;
        b2max1=size(app_tab);
        b2max2=b2max1(2);

        while (b2~=(b2max2+1)) % recopie des instances
            app_tab_reduit(b1,b2) = app_tab(ind,b2);
            b2=b2+1; % incrémentation du paramètre de boucle
        end % While - fin de la boucle sur les dimensions des instances
        app_clas_reduit(b1)=app_clas(ind);
        controle(ind)=1;
        b1=b1+1;
    end
end
```

Code du fichier reglagesigma.m

```
function [sigma_min, sigma_max] = reglagesigma(app_tab)

% Initialisation de sigma_min et sigma_max
sigma_min1 = 1000000000;
sigma_max = 0;

% Préparation de la boucle sur les instances de l'ensemble
% d'apprentissage.
b1=1; % Paramètre de boucle
b1max1=size(app_tab);
b1max2=b1max1(1); % Obtention du nombre d'éléments dans l'ensemble
d'apprentissage

while (b1~=(b1max2+1)) % 1ère boucle sur les instances pour calculer la moyenne
    normeL2 = norm(app_tab(b1,:)); % norme L2

    if(normeL2>sigma_max)
        sigma_max=normeL2;
    end

    % Préparation de la boucle sur les instances de l'ensemble
    % afin de les examiner deux à deux
    b2=b1+1;
    while(b2~=(b1max2+1))
        normeL2 = norm(app_tab(b1,:)-app_tab(b2,:)); % norme L2

        if((normeL2<sigma_min1)&&(normeL2~=0))
            sigma_min1=normeL2;
        end

        b2=b2+1;
    end

    b1=b1+1; % incrémentation du paramètre de boucle
end % While - fin de la boucle sur les instances d'apprentissage
sigma_min=sigma_min1/32;
```

Code du fichier kfcv1.m

```
function [app_tab, test_tab, app_clas, test_clas] = kfcv1(tab,clas,repartitions)
```

```
% Détermination de k qui est le nombre d'éléments du vecteur de répartitions
k= numel(repartitions);


% Détermination du nombre d'instances de l'ensemble d'instances
cardtab0 = size(tab);
cardtab = cardtab0(1);

% Afficher un message d'avertissement si le nombre d'élément de l'ensemble
% est inférieur au partitionnement

if (cardtab<k)

    fprintf('Attention, le nombre d instances de l ensemble fourni est inférieur
au paramètre de partitionnement k.')

end % if - fin de l'avertissement conditionnel

% Détermination du nombre d'instances minimale par partition
n = floor(cardtab/k);

% Création de la matrice d'effectifs par partitions
mat = ones(k) * n;

% Détermination du nombre d'instances restantes à distribuer entre les
partitions
m = mod(cardtab,k);

% Préparation de la boucle sur les m premieres partitions de mat
b1 = 1;

while (b1~=(m+1))

    mat(b1)=mat(b1)+1; % Répartition des instances restantes dans les partitions

    b1=b1+1; % passage à la partition suivante
end % While - Fin de la boucle sur les m premieres partitions

%     Création et remplissage de app_tab et de app_clas
%     -------------------------------------------------

% Préparation de la boucle sur le vecteur de répartition.
b1=1;

% Initialisation du repère de la ligne du tableau app_tab (écriture)
b3=1;

% Initialisation du repère de la ligne du tableau app_tab (écriture)
b6=1;

% Initialisation du repère de la ligne du tableau tab (lecture)
b4=1;

% Initialisation des tableaux de données
%t1 = zeros(1000,8)
%t2 = zeros(1000)
%t3 = zeros(1000,8)
%t4 = zeros(1000)

while (b1~=(k+1))

    % Préparation de la boucle b2 sur le nombre d'instances à transferer
    b2=1;
```

```
    while (b2~=(mat(b1)+1)) % boucle permettant le transfert d'un instance à
chaque tour

       if(repartitions(b1)==0); % si l'indice est égal à 0, on place l'élément
dans l'ensemble d'apprentissage

           b5=1;
           b5max1 = size(tab);
           b5max2 = b5max1(2);

           while (b5~=(b5max2+1))
                t1(b3,b5) = tab(b4,b5);

                b5=b5+1;
           end

           t2(b3) = clas(b4);

           b3=b3+1;  % passage à l'instance suivante en écriture
           b4=b4+1;  % passage à l'instance suivante en lecture
       else % sinon on le place dans l'ensemble de test
           b5=1;
           b5max1 = size(tab);
           b5max2 = b5max1(2);

           while (b5~=(b5max2+1))
                t3(b6,b5) = tab(b4,b5);

                b5=b5+1;
           end

           t4(b6) = clas(b4);

           b6=b6+1; % passage à l'instance suivante en écriture
           b4=b4+1; % passage à l'instance suivante en lecture
       end

       b2=b2+1;  % passage à l'instance suivante parmi les mat(b1) instances
    end

    b1=b1+1;
end

app_tab=t1;
app_clas=t2;
test_tab=t3;
test_clas=t4;
```

Code du fichier kfcv3.m

```
function [app_tab, test_tab] = kfcv3(tab,repartitions)

% Détermination de k qui est le nombre d'éléments du vecteur de répartitions
k= numel(repartitions);

% Détermination du nombre d'instances de l'ensemble d'instances
cardtab0 = size(tab);
cardtab = cardtab0(1);

% Afficher un message d'avertissement si le nombre d'élément de l'ensemble
% est inférieur au partitionnement

if (cardtab<k)
```

```
    fprintf('Attention, le nombre d instances de l ensemble fourni est inférieur
au paramètre de partitionnement k.')

end % if - fin de l'avertissement conditionnel

% Détermination du nombre d'instances minimale par partition
n = floor(cardtab/k);

% Création de la matrice d'effectifs par partitions
mat = ones(k) * n;

% Détermination du nombre d'instances restantes à distribuer entre les
partitions
m = mod(cardtab,k);

% Préparation de la boucle sur les m premieres partitions de mat
b1 = 1;

while (b1~=(m+1))

    mat(b1)=mat(b1)+1; % Répartition des instances restantes dans les partitions

    b1=b1+1; % passage à la partition suivante
end % While - Fin de la boucle sur les m premieres partitions

%     Création et remplissage de app_tab et de test_tab
%     -------------------------------------------------

% Préparation de la boucle sur le vecteur de répartition.
b1=1;

% Initialisation du repère de la ligne du tableau app_tab (écriture)
b3=1;

% Initialisation du repère de la ligne du tableau app_tab (écriture)
b6=1;

% Initialisation du repère de la ligne du tableau tab (lecture)
b4=1;

while (b1~=(k+1))

    % Préparation de la boucle b2 sur le nombre d'instances à transferer
    b2=1;

    while (b2~=(mat(b1)+1)) % boucle permettant le transfert d'un instance à
chaque tour

       if(repartitions(b1)==0); % si l'indice est égal à 0, on place l'élément
dans l'ensemble d'apprentissage

           b5=1;
           b5max1 = size(tab);
           b5max2 = b5max1(2);

           while (b5~=(b5max2+1))
                t1(b3,b5) = tab(b4,b5);

                b5=b5+1;
           end

           b3=b3+1;  % passage à l'instance suivante en écriture
           b4=b4+1;  % passage à l'instance suivante en lecture
```

```
       else % sinon on le place dans l'ensemble de test
           b5=1;
           b5max1 = size(tab);
           b5max2 = b5max1(2);

           while (b5~=(b5max2+1))
                t3(b6,b5) = tab(b4,b5);

                b5=b5+1;
           end

           b6=b6+1; % passage à l'instance suivante en écriture
           b4=b4+1; % passage à l'instance suivante en lecture
       end

       b2=b2+1;  % passage à l'instance suivante parmi les mat(b1) instances
    end

    b1=b1+1;
end

app_tab=t1;
test_tab=t3;
```

Code du fichier explicative2cible.m

```
function [app_tab_reduit, test_tab_reduit, app_valeurs_cibles,
test_valeurs_cibles] = explicative2cible(app_tab,test_tab,dimension)

% Préparation de la boucle sur les dimensions de lecture
b1=1;
b1max1=size(app_tab);
b1max2=b1max1(2);%nombre de colonnes-> dimensions de l'ensemble de test et
d'apprentissage

% Préparation du paramètre d'écriture des dimensions pour les 2 nouveaux
% ensembles
b2=1;

while (b1~=(b1max2+1)) % boucle de lecture sur les dimensions de l'ensemble de
test et d'apprentissage
    if(dimension==b1) % Si la dimension courante des valeurs explicatives est
celle qui est demandée alors:
        app_valeurs_cibles = app_tab(:,b1); % ajout de la colonne courante de
l'ensemble aux valeurs cibles
        test_valeurs_cibles = test_tab(:,b1);% ajout de la colonne courante de
l'ensemble aux valeurs cibles
    else % si c'est une autre dimension
        app_tab_reduit(:,b2) = app_tab(:,b1); % ajout de la colonne courante de
l'ensemble au nouvel ensemble
        test_tab_reduit(:,b2) = test_tab(:,b1); % ajout de la colonne courante
de l'ensemble au nouvel ensemble

        b2=b2+1; % incrémentation du paramètre d'écriture des nouveaux ensembles
    end % fin de la séparation de la colonne des valeurs cibles aux valeurs
explicatives

    b1=b1+1; % incrémentation du paramètre de lecture des ensembles
end % fin boucle sur la lecture des ensembles
```

Code du fichier parzen3.m

```
function [estimations] = parzen3(ens_train,ens_test,valeurs_cibles,sigma)

% calcul du sigma au carré
sigma_carre = sigma.^2;

% préparation de la première boucle sur l'ensemble de test
b1=1; % Paramètre de boucle
b1max1=(size(ens_test));
b1max2=b1max1(1); % taille de l'ensemble de test en nombre d'instances.

% initialisation du tableau de prédiction: une valeur pour chaque instance
% de test
% estimations = zeros(b1max2);

while (b1~=(b1max2+1)) % boucle sur l'ensemble de test
    % fprintf('ligne de test') résultat intermédiaire qui peut être affiché
    % ens_test(b1,:) résultat intermédiaire qui peut être affiché
    rn = 0 ;
    rd = 0 ;
    % Préparation de la boucle sur l'ensemble d'apprentissage
    b2=1; % paramètre de boucle
    b2max1=(size(ens_train));
    b2max2=b2max1(1); % Obtention du nombre d'instance de l'ensemble
d'apprentissage

    while(b2~=b2max2+1) % boucle sur l'ensemble d'apprentissage
        % ens_train(b2,:) résultat intermédiaire qui peut être affiché
        normeL2 = norm(ens_train(b2,:) - ens_test(b1,:)); % norme L2
        dist_parzen = exp(-(normeL2.^2)./(2*sigma_carre)); % kernel/distance au
sens de parzen

        rn = rn + dist_parzen*valeurs_cibles(b2); % ajout de la distance au sens
de parzen au numérateur du résultat
        rd = rd + dist_parzen; % ajout de la distance au sens de parzen au
dénominateur du résultat

        b2 = b2 + 1;
        % r=rn/rd  résultat intermédiaire qui peut être affiché
    end % while - fin de la boucle sur l'ensemble d'apprentissage
    if(rd==0)
        rd=eps;
    end
    estimations(b1) = rn/rd; % enregistrement de la valeur obtenue dans le
tableau

    b1=b1+1;
end % while - fin de la boucle sur l'ensemble de test.
```

Code du fichier aoc.m

```
function [valeur_aoc] = aoc(test_valeurs_cibles,estimations,epsilon)

% Préparation de la boucle sur les valeurs cibles
b1=1;
b1max1 = size(test_valeurs_cibles);
b1max2 = b1max1(1);

% Création du tableau des valeurs absolues de l'erreur entre les valeurs
% cibles test et leurs estimations.
while (b1~=(b1max2+1))
    erreur(b1)=abs(test_valeurs_cibles(b1)-estimations(b1));

    b1=b1+1;
```

```
end

[erreur_ordre_croissant]= sort(erreur); % Tri des données dans l'ordre croissant

% Initialisation des paramètres de la courbe de rec dont on calcule l'aire
x=0; % abscisse
y=0; % ordonnée
int=0; % aire sous la courbe de rec (calculé par la méthode des trapèzes)

% Préparation de la boucle croissante sur les erreurs
b2=1; % paramètre de boucle
b2max1=size(erreur_ordre_croissant);
b2max2=b2max1(2);

while (b2~=(b2max2+1)) % Boucle sur les erreurs
    xo=x;% trace de l'abscisse pour le calcul de l'aire du trapèze courant
    yo=y;% trace de l'ordonnée pour le calcul de l'aire du trapèze courant

    x=erreur_ordre_croissant(b2); % incrémentation de l'abscisse
    y=y+1/b2max2; % incrémentation de l'ordonnée

    while
((b2~=b2max2)&&(erreur_ordre_croissant(b2)==erreur_ordre_croissant(b2+1)))
                              %  Boucle en cas d'égalité sur les erreurs

       x=erreur_ordre_croissant(b2+1); % incrémentation de l'abscisse
       y=y+1/b2max2; % incrémentation de l'ordonnée

       b2 = b2 + 1;
    end % While - Fin de la boucle en cas d'égalité sur les erreurs

    int=int+(x-xo)*(y+yo)/2; % ajout de l'aire du trapèze
    b2=b2+1;
end % While - Fin de la boucle sur l'ensemble de test
xo=x;
x=epsilon;
int=int+(x-xo); % ajout de l'aire du trapèze final

int=int/(epsilon); % normalisation de l'aire sous la courbe de rec
valeur_aoc=1-int;
```

Code du fichier methode1.m (Partie affichage)

Les partie affichages de methode2.m et methode3.m se trouvent à la fin des fichiers. Les fichiers sont accessibles dans les répertoire 'Parzen Projet'.

```
pourcents = strrep(num2str(pourcentages),'  ','|');
% Préparation des noms de fichiers pour les sauvegardes
file = filename(1,1:length(filename)-5);
ladate = date;
% Préparation du dossier de sauvagarde
mkdir(strcat(file,ladate))

if (sauvegarde_automatique_sans_affichage==0)
    button = questdlg('Pour afficher valeurs/graphiques souhaités ou enregistrer
des données, cliquez sur OK.  Attention : Annuler quitte définitivement le
menu','Menu de récupération des données','Ok','Annuler','Ok');

    while (button(1) =='O')

        close(figure(1))
```

```
        pourcents = strrep(num2str(pourcentages),'  ','|');

        % Préparation des noms de fichiers pour les sauvegardes
        file = filename(1,1:length(filename)-5);
        ladate = date;
        % Préparation du dossier de sauvagarde
        mkdir(strcat(file,ladate))


        answer={1};
        while(numel(answer)~=0)

            prompt = {'Tableaux des sigma,auc moyen et écart type : (1 si oui, 0
sinon)','Tableaux des sigma optimaux et auc correspondants :(1 si oui, 0
sinon)','Tableaux des auc et écart types par pourcentages de train utilisées:(1
si oui, 0 sinon)'};
            dlg_title = 'Données';
            num_lines = 1;
            def = {'0','0','0'};
            answer = inputdlg(prompt,dlg_title,num_lines,def);

            if(numel(answer)~=0) % si un affichage de données est demandé
                answer2 = str2mat(answer);
                if(answer2(1)=='1')
                    fprintf('Tableaux des sigma,auc moyen et écart type :\n');
                    c=[];
                    while (isempty(c)~=0)
                        prompt = {strcat('Quel pourcentage à regarder parmi :
',pourcents,' ?              ')};
                        dlg_title = 'Précision du pourcentage';
                        def = {num2str(pourcentages(1))};
                        a = inputdlg(prompt,dlg_title,num_lines,def);
                        b = str2double(a);
                        c = find(pourcentages==b);
                    end
                    resultats1(:,:,c,:)

                    prompt = {'Enregistrement de la courbe : (1 si oui, 0
sinon)'};
                    dlg_title = 'Sauvegarde';
                    def = {'1'};
                    answer = inputdlg(prompt,dlg_title,num_lines,def);

                    if(numel(answer)~=0)
                        answer3 = str2mat(answer);
                        if(answer3(1)=='1')
                            savefile1 = strcat(pwd,'/',file,ladate,'/',file,lad-
ate(1,1:6),'-met1-
r1-',num2str(b),'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max)
,'.wk1');
                            wk1write(savefile1, resultats1(:,:,c,:));
                        end
                    end

                end
                if(answer2(2)=='1')
                    fprintf('Tableaux des sigma optimaux et auc correspondants
:\n');

                    c=[];
                    while (isempty(c)~=0)
                        prompt = {strcat('Quel pourcentage à regarder parmi  :
',pourcents,' ?              ')};
```

```
                        dlg_title = 'Précision du pourcentage';
                        def = {num2str(pourcentages(1))};
                        a = inputdlg(prompt,dlg_title,num_lines,def);
                        b = str2double(a);
                        c = find(pourcentages==b);
                    end

                    resultats2(c,:,:)

                    prompt = {'Enregistrement de la courbe : (1 si oui, 0
sinon)'};
                    dlg_title = 'Sauvegarde';
                    def = {'1'};
                    answer = inputdlg(prompt,dlg_title,num_lines,def);

                    if(numel(answer)~=0)
                        answer3 = str2mat(answer);
                        if(answer3(1)=='1')
                            savefile2 = strcat(pwd,'/',file,ladate,'/',file,lad-
ate(1,1:6),'-met1-
r2-',num2str(b),'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max)
,'.wk1');
                            wk1write(savefile2, resultats2(c,:,:));
                        end
                    end
                end
                if(answer2(3)=='1')
                    fprintf('Tableaux des auc et écart types par pourcentages de
train utilisées:\n');
                    resultats3
                    prompt = {'Enregistrement de la courbe : (1 si oui, 0
sinon)'};
                    dlg_title = 'Sauvegarde';
                    def = {'1'};
                    answer = inputdlg(prompt,dlg_title,num_lines,def);

                    if(numel(answer)~=0)
                        answer3 = str2mat(answer);
                        if(answer3(1)=='1')
                            savefile3 = strcat(pwd,'/',file,ladate,'/',file,lad-
ate(1,1:6),'-met1-
r3-',pourcents,'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max),
'.wk1');
                            wk1write(savefile3, resultats3);
                        end
                    end
                end
            end
        end

        answer={1};
        while(numel(answer)~=0)

            prompt = {'Graphique des auc moyen (+écart type) en fonction des
sigma : (1 si oui, 0 sinon)','Graphique des sigma optimaux et auc correspondants
: (1 si oui, 0 sinon)','Graphique des auc (+ écart types) par pourcentages de
train utilisées:(1 si oui, 0 sinon)'};

            dlg_title = 'Courbes';
            def = {'0','0','0'};
            answer = inputdlg(prompt,dlg_title,num_lines,def);

            if(numel(answer)~=0) % si un affichage de courbe est demandé
                answer2 = str2mat(answer);
```

```
                if(answer2(1)=='1')
                    close(figure(2))
                    figure(2);

                    c=[];
                    while (isempty(c)~=0)
                        prompt = {strcat('Quel pourcentage à regarder parmi  :
',pourcents,' ?              ')};
                        dlg_title = 'Précision du pourcentage';
                        def = {num2str(pourcentages(1))};
                        a = inputdlg(prompt,dlg_title,num_lines,def);
                        b = str2double(a);
                        c = find(pourcentages==b);
                    end

                    for j=1:param_iteration,
                        i=1:1:b2max;

                        color=[0 0 0;0 0 1;0 1 0;1 0 0; 0.8 0 0.8; 0.6 0.6 0;0
0.8 0.8;0 0 1];
                        errorbar(resultats1(i,1,c,j),resultats1(i,2,c,j),2*res-
ultats1(i,3,c,j),'Color',color(mod(j,7)+1,:))
                        aucminimum = min(resultats1(:,2,c,j))-
max(resultats1(:,3,c,j));
                        axis([0,sigma_max,aucminimum,1])
                        title('Valeur des auc moyen en fonction de \sigma');
                        xlabel('paramètre \sigma utilisé pour la fenêtre de
parzen');
                        legend('une itération',4);
                        hold on
                    end
                    grid;

                    prompt = {'Enregistrement de la courbe : (1 si oui, 0
sinon)'};
                    dlg_title = 'Sauvegarde';
                    def = {'1'};
                    answer = inputdlg(prompt,dlg_title,num_lines,def);

                    if(numel(answer)~=0)
                        answer3 = str2mat(answer);
                        if(answer3(1)=='1')
                            savefile1 = strcat(pwd,'/',file,ladate,'/',file,lad-
ate(1,1:6),'-met1-
r1-',num2str(b),'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max)
,'.fig');
                            saveas(figure(2),savefile1)
                        end
                    end
                end

                if(answer2(2)=='1')

                    c=[];
                    while (isempty(c)~=0)
                        prompt = {strcat('Quel pourcentage à regarder parmi  :
',pourcents,' ?              ')};
                        dlg_title = 'Précision du pourcentage';
                        def = {num2str(pourcentages(1))};
                        a = inputdlg(prompt,dlg_title,num_lines,def);
                        b = str2double(a);
                        c = find(pourcentages==b);
                    end
```

```
                    close(figure(3))
                    figure(3);

                    hold off
                    i=1:1:iterations(c);
                    plot(i,resultats2(c,i,1),'+',i,resultats2(c,i,2),'o');
                    title('valeur du \sigma optimal et de l`auc correspondant');
                    legend('\sigma optimal','auc correspondant');
                    axis([0,iterations(c)+1,0,max(sigma_max,1)])
                    xlabel('Numéro d`itération ')
                    grid;

                    prompt = {'Enregistrement de la courbe : (1 si oui, 0
sinon)'};
                    dlg_title = 'Sauvegarde';
                    def = {'1'};
                    answer = inputdlg(prompt,dlg_title,num_lines,def);

                    if(numel(answer)~=0)
                        answer3 = str2mat(answer);
                        if(answer3(1)=='1')
                            savefile2 = strcat(pwd,'/',file,ladate,'/',file,lad-
ate(1,1:6),'-met1-
r2-',num2str(b),'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max)
,'.fig');
                            saveas(figure(3),savefile2)
                        end
                    end
                end

                if(answer2(3)=='1')
                    close(figure(4))
                    figure(4);

                    hold off

                    % Calcul des 3 courbes de résultats; courbe d'auc moyenne,
courbe d'auc
                    % moyenne écartés de 2 fois l'écart type au dessus et en
dessus
                    i=1:1:numel(pourcentages);
                    errorbar(pourcentages(i),resultats3(i,1),2*resultats3(i,2))
                    axis([0,100,0,1])
                    xlabel('pourcentages du "train" utilisé');
                    title('valeur des auc pour différents pourcentages extraits
de l ensemble d apprentissage');
                    legend('auc moyen obtenue',2);
                    grid;

                    prompt = {'Enregistrement de la courbe : (1 si oui, 0
sinon)'};
                    dlg_title = 'Sauvegarde';
                    def = {'1'};
                    answer = inputdlg(prompt,dlg_title,num_lines,def);

                    if(numel(answer)~=0)
                        answer2 = str2mat(answer);
                        if(answer2(1)=='1')
                            savefile3 = strcat(pwd,'/',file,ladate,'/',file,lad-
ate(1,1:6),'-met1-
r3-',pourcents,'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max),
'.fig');
                            saveas(figure(4),savefile3)
                        end
```

```
                    end
                end
            end
        end

        button = questdlg('Cliquez sur OK pour afficher ou enregistrer des
données. Attention : Annuler quitte définitivement le menu','Menu de
récupération des données','Ok','Annuler','Ok');
    end
else % en cas de sauvegarde automatique
    close(figure(1))

    for c=1:numel(pourcentages)

        fprintf('Tableaux des sigma,auc moyen et écart type :\n');
        resultats1(:,:,c,:)

        savefile1 = strcat(pwd,'/',file,ladate,'/',file,ladate(1,1:6),'-met1-
r1-',num2str(pourcentages(c)),'%it',num2str(param_iteration),'k',num2str(k),'s',
num2str(b2max),'.wk1');
        wk1write(savefile1, resultats1(:,:,c,:));

        fprintf('Tableaux des sigma optimaux et auc correspondants :\n');
        resultats2(c,:,:)

        savefile2 = strcat(pwd,'/',file,ladate,'/',file,ladate(1,1:6),'-met1-
r2-',num2str(pourcentages(c)),'%it',num2str(param_iteration),'k',num2str(k),'s',
num2str(b2max),'.wk1');
        wk1write(savefile2, resultats2(c,:,:));
    end

    fprintf('Tableaux des auc et écart types par pourcentages de train
utilisées:\n');
    resultats3

    savefile3 = strcat(pwd,'/',file,ladate,'/',file,ladate(1,1:6),'-met1-
r3-',pourcents,'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max),
'.wk1');
    wk1write(savefile3, resultats3);

    for c=1:numel(pourcentages)
        close(figure(2))
        figure(2);
        for j=1:param_iteration,
            i=1:1:b2max;
            color=[0 0 0;0 0 1;0 1 0;1 0 0; 0.8 0 0.8; 0.6 0.6 0;0 0.8 0.8;0 0
1];
            errorbar(resultats1(i,1,c,j),resultats1(i,2,c,j),2*resultats1(i,3,c
,j),'Color',color(mod(j,7)+1,:))
            aucminimum = min(resultats1(:,2,c,j))-max(resultats1(:,3,c,j));
            axis([0,sigma_max,aucminimum,1])
            title('Valeur des auc moyen en fonction de \sigma');
            xlabel('paramètre \sigma utilisé pour la fenêtre de parzen');
            legend('une itération',4);
            hold on
        end
        grid;

        savefile1 = strcat(pwd,'/',file,ladate,'/',file,ladate(1,1:6),'-met1-
r1-',num2str(pourcentages(c)),'%it',num2str(param_iteration),'k',num2str(k),'s',
num2str(b2max),'.fig');
        saveas(figure(2),savefile1)

        close(figure(3))
```

```
        figure(3);

        hold off

        i=1:1:iterations(c);
        plot(i,resultats2(c,i,1),'+',i,resultats2(c,i,2),'o');
        title('valeur du \sigma optimal et de l`auc correspondant');
        legend('\sigma optimal','auc correspondant');
        axis([0,iterations(c)+1,0,max(sigma_max,1)])
        xlabel('Numéro d`itération ')
        grid;

        savefile2 = strcat(pwd,'/',file,ladate,'/',file,ladate(1,1:6),'-met1-
r2-',num2str(pourcentages(c)),'%it',num2str(param_iteration),'k',num2str(k),'s',
num2str(b2max),'.fig');
        saveas(figure(3),savefile2)

    end

    close(figure(4))
    figure(4);

    hold off

    % Calcul des 3 courbes de résultats; courbe d'auc moyenne, courbe d'auc
    % moyenne écartés de 2 fois l'écart type au dessus et en dessus
    i=1:1:numel(pourcentages);
    errorbar(pourcentages(i),resultats3(i,1),2*resultats3(i,2))
    axis([0,100,0,1])
    xlabel('pourcentages du "train" utilisé');
    title('valeur des auc pour différents pourcentages extraits de l ensemble d
apprentissage');
    legend('auc moyen obtenue',2);
    grid;

    savefile3 = strcat(pwd,'/',file,ladate,'/',file,ladate(1,1:6),'-met1-
r3-',pourcents,'%it',num2str(param_iteration),'k',num2str(k),'s',num2str(b2max),
'.fig');
    saveas(figure(4),savefile3)
end
```

Annexe : Structure des résultats

Les résultats sont contenus dans un sous dossier du dossier Parzen Projet. Un dossier de résultat porte le nom d'un jeu de données et la date de l'expérience (Attention c'est la date au moment de l'obtention des résultats à la fin de l'expérience!). Un dossier rassemble tous les résultats d'un jeu de données qui se sont terminées le même jour. Les données peuvent être des tableaux .wk1 ou des figures .fig En mode d'enregistrement automatique, les résultats contiennent exactement le même nombre de fichiers .fig et .wk1 puisque un .fig correspond à un .wk1.

Un nom de fichier contient le nom du jeu, le nom de la méthode, la nature du résultat codé par r1 r2 ou r3 le ou les pourcentages utilisées le paramètre k de la k-fold cross validation, le nombre d'itérations et enfin le nombre de valeurs de sigma. Attention, il réside un dysfonctionnement, pour la réalisation du tableau r3. Pour pouvoir exploiter les valeurs numériques il faut réunir les résultats r2 de tous les pourcentages utilisées et faire des calculs de moyenne et d'écart type dans un tableur.
Tableaux des Résultats R1 :

Ils sont constitués de séries de 3 colonnes mises ensemble. Si 5 itérations ont été réalisées, cela fera 3*5=15 colonnes. Ils contiennent les valeurs de sigma (1ère colonnne), les valeurs d'AOC moyen (c'est une moyenne de moyenne)/ moyenne des AUC espérée en 2ème colonne et enfin les écarts types des AOC moyen/AUC espérées en 3ème colonne. Ils y a autant de lignes que de sigma calculées.

Tableaux des Résultats R2 :

Ils n'y a qu'un seule ligne : Elle contient le sigma optimal (autant de fois que d'itérations), puis la valeur d'AUC calculée sur ce sigma optimal (là encore autant de fois que d'itérations). Les valeurs de sigma optimal sont mis ensemble et les valeurs d'AUC aussi.

Tableaux des Résultats R3 :

Partie non fonctionnelle. (il faut utiliser R2)

Tableaux des Figures R1 :

AUC et AOC fonction de sigma: et données contenues dans tableau R1 (voir tableau R1)

Tableaux des Figures R2 :

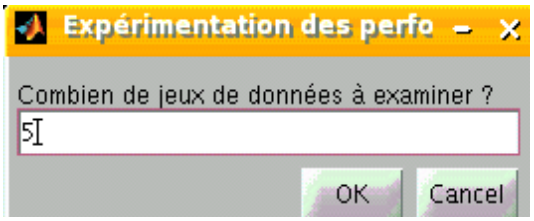
AUC optimaux et AUC calculées avec : données contenues dans tableau R2 (voir tableau R2)

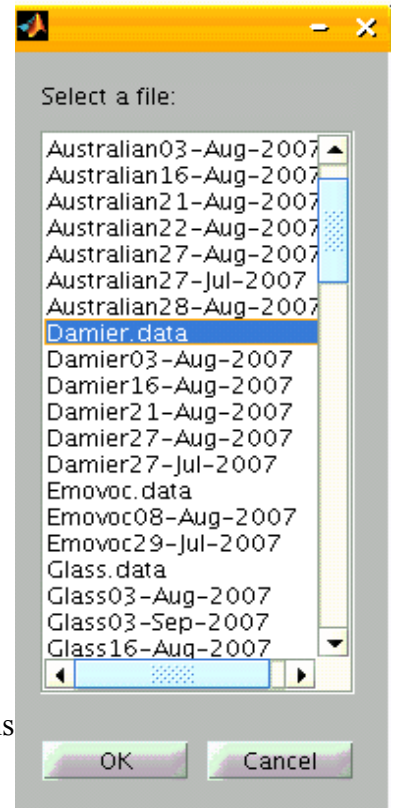
Tableaux des Figures R3 :

AUC moyen et écart type pour chaque pourcentage extraits de l'ensemble d'apprentissage.

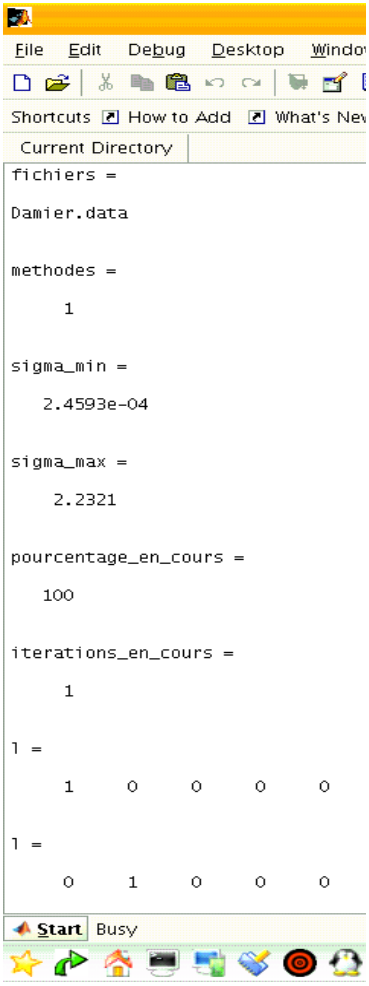
Annexe : Mode d'emploi

Préparer le nombre d'expériences et charger cette commande. Exemple
5 expériences – Pima - méthode 1 – Damier - méthode 2 – Iris méthode 3 – Australian – méthode 1 – Iris méthode 1. Ce qui donne :
5 OK Pima OK 1 OK Damier OK 2 OK Iris OK 3 OK Australian OK 1 OK Iris OK 1 OK
Annuler quitte tout. A la fin l'expérience peut commencer.



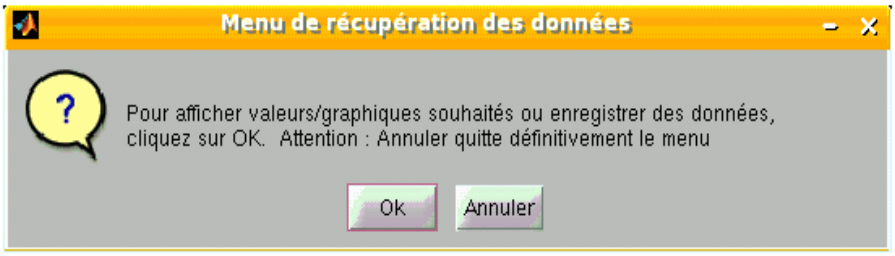


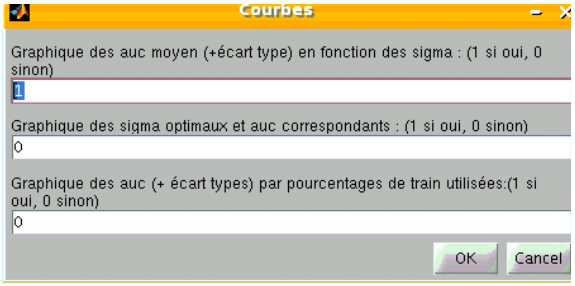
On peut suivre l'évolution de l'expérience avec l'affichage de certains paramètres comme le nombre d'itérations le pourcentage en cours, le partitionnement etc.

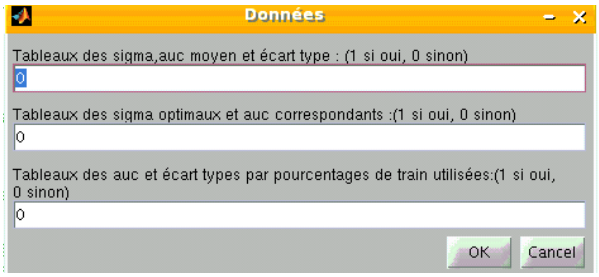


En mode non automatique, tout est enregistré à la fin. En mode non automatique, on peut faire 4 choses: Visualiser des données au choix dans la fenêtre de dialogue, Afficher une courbe au choix, enregistrer une courbe au choix, enregistrer un tableau au choix. Attention, pour enregistrer un tableau il faut le visualiser (une erreur s'est glissé: 'il est mis enregistrer une courbe'). De même, on

ne peut pas enregistrer une courbe sans la visualisée







Note Importantes :

La normalisation s'effectue uniquement avant et elle se fait sur tous les éléments de l'ensemble d'apprentissage quelque soit le pourcentage que l'on extrait. La normalisation se fait selon les dimensions de tous les éléments apprentissage et de test à partir de l'ensemble des données d'apprentissage. C'est une normalisation centrée réduit.

Un pourcentage extrait de l'ensemble d'apprentissage est utilisé pour l'obtention du sigma optimal mais l'ensemble des données d'apprentissage est en fait exploité pour le test final.

Dans la méthode de régression, le sigma optimal est obtenu en minimisant la moyenne des AOC moyen obtenu pour chaque sigma. En effet, pour chaque sigma et pour chaque dimension, il est calculé un AOC. L'ensemble des dimensions permet d'obtenir pour chaque sigma une valeur d'AOC moyen. Enfin la k-fold cross validation permet d'obtenir une moyenne de l'AOC moyen.

Références Bibliographiques

D.Cohn L.Atlas R.Ladner (1992) Improving Generalization with Active Learning
V. Lemaire A. Bondu (2006) Etat de l'art sur les méthodes statistiques d'apprentissage actif
Yoshua Bengio, Pascal Vincent (2004) Manifold Parzen Windows
Pascal Vincent (2003) Modèles à noyaux à structure locale
Bernhard Schölkopf (1999) Input Space Versus Feature Space in Kernel-Based Methods

Bernhard Schölkopf (2000) Statistical Learning and Kernel Methods
Olivier Chapelle "Active Learning for Parzen Window Classifier"
Jérome Louradour (2007) Noyaux de séquences pour la vérification du locuteur par Machines à Vecteurs de Support
Tom Fawcett (2003) ROC Graphs: Notes and Practical Considerations for Data Mining Researchers
Jinbo Bi, Kristin P. Bennett (2003) Regression Error Characteristic Curves

Autres sources

Wikipedia:http://fr.wikipedia.org/ Voir les sections: Apprentissage supervisé, Apprentissage non supervisé

www.ulb.ac.be

Présentations

www.stat.ucla.edu/~sczhu/Courses/UCLA/Stat_231/Lect_note_2005/Lect10_knn.pdf
www.stat.ucla.edu/~sczhu/Courses/UCLA/Stat_231/Lect_note_2005/Lect9_Parzen.pdf
http://www.iro.umontreal.ca/~vincentp/ift3390/index.html: knn parzen.pdf gaussienne evalperf.pdf