République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Ghardaïa

Faculté des Sciences et Technologie

Département des Mathématiques et Informatique

Projet de fin d'étude présenté en vue de l'obtention du diplôme de

LICENCE

Domaine : Mathématiques et Informatique

Spécialité : Informatique

THEME:

Noyaux sur les arbres

Développement d'une boite à outils

PAR:

Chebaki hadjer

Bendjeddou Wahiba

Jury:

M^{r}: Slimane Bellaouar	Maitre Assistant A Univ. Ghardaïa	**Encadreur**
M^{r}: Oulednaoui Slimane	Maitre Assistant A Univ. Ghardaïa	**Examinateur**

ANNEE UNIVERSITAIRE: 2013/2014

Remerciement

Avant toute chose, on remercie le bon dieu tout puissant de nous avoir permis de mener à terme de ce mémoire qui est pour nous le point de départ d'une merveilleuse aventure , celle de la recherche, source de remise en cause permanant et de perfectionnement perpétuelle. On tient tout d'abord à remercier ***Mr Bellaouar Slimane****, notre encadreur de mémoire, pour tout le soutien, l'aide, l'orientation, la guidance qu'il nous a apporté durant tout l'année de notre cursus universitaire ainsi que pour ses précieux conseils et ses encouragements lors de la réalisation de notre mémoire*

Nous remercions également les membres de jury qui nous ont fait l'honneur en accepter d'évaluer notre travail

Nous remercions tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé nos réflexions et ont accepté à nous rencontrer et répondre à nos questions durant nos recherches

Nous remercions aussi nos deux familles pour leur soutien durent la réalisation de ce projet

À tous ces intervenants, nous présentons nos remerciements, notre respect et nos gratitudes.

Résumé :

Avec le phénomène de l'explosion de données, l'apprentissage automatique est devenu de plus en plus une discipline indispensable dans le domaine d'informatique. Cependant, dans de nombreuses applications dans le monde réel, les données ne sont pas toujours linéairement séparables. Par conséquence, les méthodes d'apprentissages traditionnelles ne peuvent pas être appliquées. Pour faire face à ces limitations, les méthodes à noyaux ont été proposées.

Dans ce mémoire on s'intéresse précisément aux noyaux sur les arbres qui ont proliféré dans des vastes domaines grâce à leur forme hiérarchique.

Nous avons présenté l'astuce noyau ainsi que les méthodes à noyau d'arbre ST (Subtree kernel) et SST (Subset tree kernel) ainsi que l'analyse de leur complexité.

Par ailleurs nous avons commencé au développement d'une boite à outils comportant l'implémentation des algorithmes suscités.

Mot clés : arbre, noyau, apprentissage, …

Abstract:

With the phenomenon of the explosion of data, machine learning is increasingly become a necessary discipline in the field of informatics. However, in many applications in the real world, the data are not always linearly separable. As consequence, the traditional learning methods cannot be applied. To cope with these limitations, the kernel methods have been proposed.

In this thesis we are precisely interested in kernels trees which have proliferated in vast domains thanks to their hierarchical forms

We have presented the kernel trick as well as the kernel tree methods ST (Subtree kernel) and SST (Subset tree kernel) as well as the analysis of their complexity.

In addition, we have begun the development of a toolbox with the implementation of the generated algorithms.

Key words: tree, kernel, machine learning…

Table des matières

Introduction :

Les méthodes d'apprentissage automatique forment une classe de techniques attrayantes pour l'accomplissement des tâches d'extraction de connaissances évoquées. Bien choisis, ces outils peuvent être amenés à accompagner, voire à remplacer l'opérateur humain.

Les algorithmes d'apprentissage automatique ont été appliqués à divers domaines, notamment le traitement du langage naturel et de la parole, la reconnaissance de l'écriture manuscrite, la vision robotisée, la fouille de données, les moteurs de recherche sur Internet, le diagnostic médical, la bioinformatique,… etc. Les techniques d'apprentissage ont ainsi joué un rôle crucial dans des applications qui vont de la mise au point de médicaments à l'analyse de grands réseaux de télécommunication.

Le problème principal des algorithmes d'apprentissage est la non-séparation linéaire des données d'entrées. Pour cette raison les méthodes à noyaux on été proposés comme une solution alternative des limitations des algorithmes d'apprentissage traditionnelles appliquées uniquement sur les données linéairement séparables. Elle consiste à projeter les données brutes dans un espace de redescription de haute dimension ou des relations non linéaires peuvent êtres découvertes.

Les noyaux d'arbres ont été proposés comme un outil efficace pour appliquer les technique d'apprentissage sur des données structurées et semi-structurées.

Dans le cadre de ce mémoire, nous nous intéressons principalement à l'étude des problématiques liées au traitement de données structurées et semi structurés par des approches à base de noyaux sur les arbres. Puis analyser la complexité de quelques algorithmes de construction de ces noyaux. Et en fin procéder à l'implémentation sous forme d'une boite à outils

Le mémoire s'organise en trois chapitres :

Le 1er chapitre constitue un tour d'horizon sur le domaine de l'apprentissage automatique et des méthodes à noyaux.

Le 2emechapitre présente quelques méthodes à noyaux sur les arbres. Nous avons introduit les méthodes à noyau d'arbre ST (Subtree kernel) et SST (Subset tree kernel) ainsi que l'analyse de leur complexité.

le 3emechapitre se focalise sur l'implémentation des deux algorithmes étudiés dans le chapitre 2 sous forme d'une boite à outils en utilisant le langage MATLAB.

chapitre 1:
l'apprentissage à noyaux

1. Apprentissage Automatique :

L’apprentissage automatique (machine learning) est la discipline scientifique concernée par le développement, l'analyse et l'implémentation de méthodes automatisables qui permettent à une machine (au sens large) d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques.
De tels modèles d'apprentissage ont notamment permis de développer des systèmes de reconnaissance vocale, de détection de fraude par cartes de crédit, de diagnostique médical, d'analyse de la personnalité des utilisateurs des réseaux sociaux (1) (2)

1.1. Définition :

L'apprentissage automatique consiste à utiliser des ordinateurs pour optimiser un modèle de traitement de l'information selon certains critères de performance a partir d’observations, que ce soit des données-exemples ou des expériences passées

1.2. Les types de l’apprentissage automatique :

Les algorithmes d'apprentissage peuvent se catégoriser selon le mode d'apprentissage qu'ils emploient :

1.2.1. Apprentissage supervisé :

L'apprentissage supervisé est une technique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage contenant des *« exemples »* (en général des cas déjà traités et validés) qui se présentent sous forme d'un couple(x, y) (entrée-sortie).
Le but d'un algorithme d'apprentissage supervisé est donc de généraliser pour des entrées inconnues ce qu'il a pu «apprendre » grâce aux données déjà traitées par des experts, ceci de façon « raisonnable ».

Exemple : reconnaissance des pièces

Dans cette exemple nous avons un ensemble de pièces de monnaies (échantillons) sont déjà classées dans des classes connus (10, 1, 5 et 25) en terme de masse et de taille (figure.1), et quand il s'agit d'une nouvelle pièce nous pouvons connaitre le nom de la classe qui lui appartient c.à.d. quand on aura une nouvel pièce et on veut distinguer sa classe on peut la connaitre selon les critères précédent la taille et la masse et on la classifie (figure.2) (3)

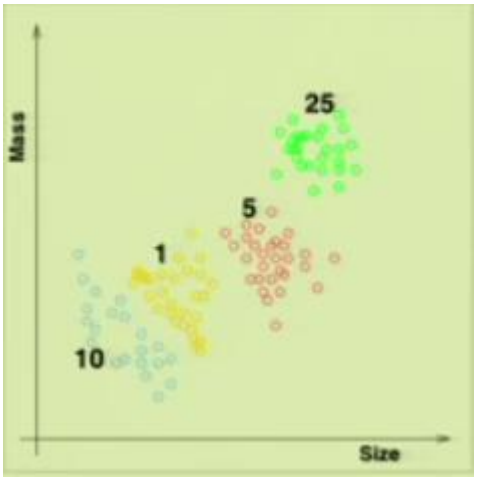


Figure 1 échantillons de monnaies classés

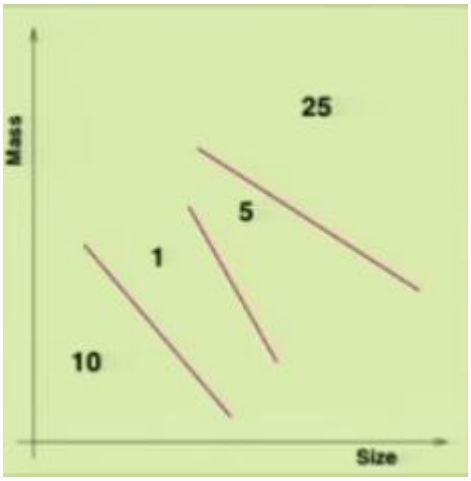


Figure 2 les classes des échantillons

On distingue en générale trois types de problème aux quels l'apprentissage supervisé est appliqué. Ces taches se diffèrent essentiellement par la nature des paires (entrée, sortie) qui y sont associées :

- **Classification** :

  Dans les problèmes de classification, l'entrée correspond à une instance d'une classe, et la sortie qui y est associée indique la classe. Par exemple pour un problème de reconnaissance de visage, l'entrée serait l'image bitmap d'une personne telle que fournie par une caméra, et la sortie indiquerait de quelle personne il s'agit (parmi l'ensemble de personnes que l'on souhaite voir le système reconnaître) (4)

  - Exemple de la classification binaire :

  L'exemple sera Approbation de crédit, la ressource qu'on peut travailler avec sont :

  Les données $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_d}\}$ sont des attributs d'un client pour possède une carte de crédit.

  L'algorithme de classification binaire prend ces attributs et il donne pour chaque $\boldsymbol{x_i}$ un poid $\boldsymbol{w_i}$ selon l'importance d'atrribut par exemple si $\boldsymbol{x_i}$ est le salaire alors $\boldsymbol{w_i}$ est un nombre important et si $\boldsymbol{x_i}$ est une Dette impayée alors $\boldsymbol{w_i}$ est un petit nombre qui peut être négative, alors cette algorithme donne un ensemble $\boldsymbol{W} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_d}\}$ et il les ajoute avec cette forme linéaire :

$$\begin{cases} \boldsymbol{si} \sum_{\boldsymbol{i=1}}^{\boldsymbol{d}} \boldsymbol{w_i x_i} > \boldsymbol{seuil\ alors\ Approbation} \\ \boldsymbol{sinon\ Rejet} \end{cases}$$

Donc On peut l'écrire comme :

$$\boldsymbol{h(x) = sign((\sum_{i=1}^{d} w_i x_i) - seuil)}$$

Pour une coordination artificielle Supposons que $w_0 = -\text{seuil}$ alors :

$$\boldsymbol{h(x) = sign((\sum_{i=1}^{d} w_i x_i) + w_0)}$$

Dans ce cas $x_0 = 1$ Et on obtient :

$$\boldsymbol{h(x) = sign(\sum_{i=0}^{d} w_i x_i)}$$

Dans une forme vectorielle on obtient :

$$\boldsymbol{h(x) = sign(W^T X)}$$

tel que : $\boldsymbol{W = \{w_1, w_2, \dots, w_d\}}$et $\boldsymbol{X = \{x_1, x_2, \dots, x_d\}}$

Alors l'algorithme sera :

L'algorithme de perceptron

On a :

$$\boldsymbol{h(x) = sign(W^T X)}$$

Les données de cette algorithme est l'ensemble de donnée d'apprentissage :

$\boldsymbol{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)}$ $\Longrightarrow$Il existe des clients x_i avec la décision y_i

1. L'algorithme essayez de choisir un point classé incorrectement
    - Un point (x_n, y_n)est classé incorrectement : cela signifie que w n'a pas fait le bon emploi pour faire une bonne classification c.-à-d. que :

$$\boldsymbol{sign(w^T x_n) \neq y_n}$$

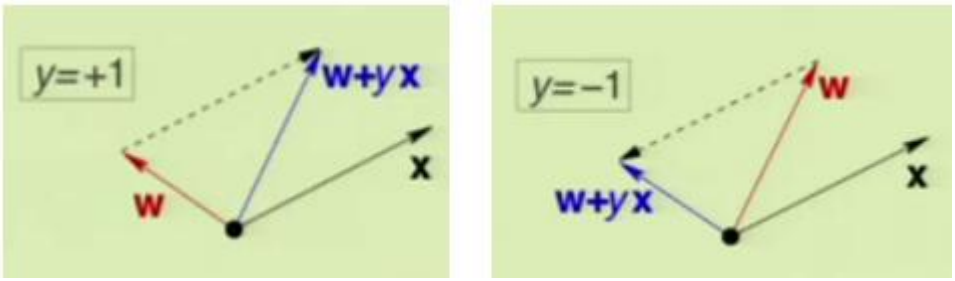


Figure 3 le changement de w

2. Si $(\boldsymbol{x_n}, \boldsymbol{y_n})$ un point qui est classé incorrectement il va changer le poids $\boldsymbol{w}$ de tels sort que

$$\boldsymbol{sign}(\boldsymbol{w^T x_n}) = \boldsymbol{y_n}$$

Alors selon les figures précèdent Le changement de $\boldsymbol{w_n}$sera comme ca :

$$\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{y_n x_n}$$

Alors le code de perceptron sera

Répéter Choisir un point classé incorrectement Changer leur poids avec $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{y_n x_n}$ Jusqu'à (aucun point est classé incorrectement)

- **Régression** :

  Dans les problèmes de régression, l'entrée n'est pas associée à une classe, mais dans le cas général, à une ou plusieurs valeurs réelles (un vecteur). Par exemple, pour une expérience de biochimie, on pourrait vouloir prédire le taux de réaction d'un organisme en fonction des taux de différentes substances qui lui sont administrées. (4)

- **Série temporelle** :

  Dans les problèmes de séries temporelles, il s'agit typiquement de prédire les valeurs futures d'une certaine quantité connaissant ses valeurs passées ainsi que d'autres informations. Par exemple le rendement d'une action en bourse. . . Une différence importante avec les problèmes de régression ou de classification est que les données suivent typiquement une distribution non stationnaire. (4)

1.2.2. L'apprentissage non supervisé :

Quand le système ou l'opérateur ne disposent que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé ou *clustering*. Aucun expert n'est requis. L'algorithme doit découvrir par lui-même la structure plus ou moins *cachée* des données. Le partitionnement de données, *data clustering* en anglais, est un algorithme d'apprentissage non supervisé.

Exemple illustratif sur ce type d'apprentissage :

Comme l'exemple précédent dans l'apprentissage supervisé mais ici l'ensemble de pièces de monnaies ne sont pas classer (figure 4), on peut seulement dire que chaque groupe se différent de l'autre groupe, le système sépare les pièces de monnaies selon des groupe et chaque groupe porte le nom type (1.2.3 ou 4) (figure 5) (3)

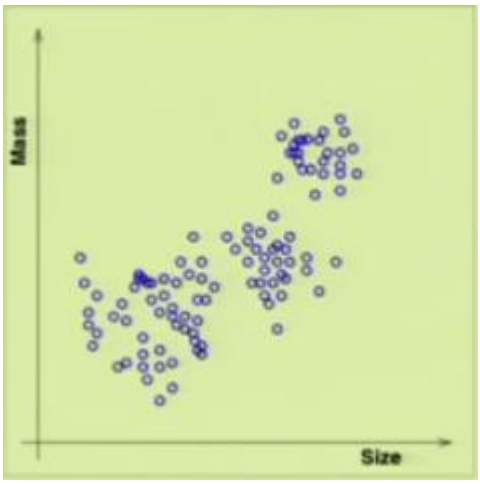


Figure 4 ensemble de pièces de monnaie

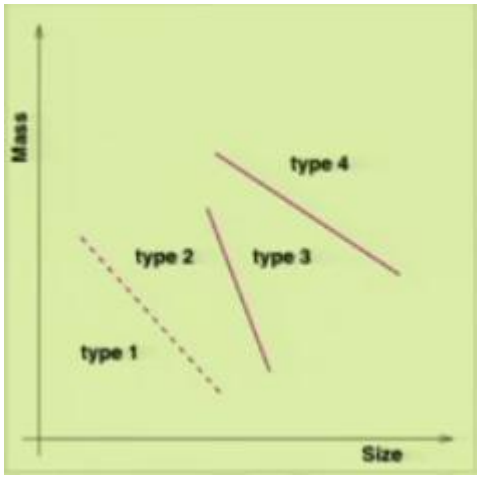


Figure 5 classement avec type

On distingue en générale trois types de problèmes dans l'apprentissage non supervisé :

- **Estimation de densité** :
  Dans un problème d'estimation de densité, on cherche à modéliser convenablement la distribution des données. L'estimateur obtenu f(x) doit pouvoir donner un bon estimé de la densité de probabilité à un point de test x issu de la même distribution (inconnue) que les données d'apprentissage. (4)

- **Partitionnement (clustring)** :
  Le problème du partitionnement est le pendant non-supervisé de la classification. Un algorithme de partitionnement tente de partitionner l'espace d'entrée en un certain nombre de "classes" en se basant sur un ensemble d'apprentissage fini, ne contenant aucune information de classe explicite. Les critères utilisés pour décider si deux points devraient appartenir à la même classe ou à des classes différents sont spécifiques à chaque algorithme, mais sont très souvent liés à une mesure de distance entre points. L'exemple le plus classique d'algorithme de partitionnement est l'algorithme K-Means

  - Algorithme k-Means :

    Méthode des K-moyennes *aussi appelée méthode des centres mobiles*

    - *Choisir K élément intiaux "centres "des K groupes*

- *Placer les objets dans le groupe de centre le plus proche*
- Recalculer le centre de gravité dechaque groupe
- *Itérer l'algorithme jusqu'à ce que les objets ne changent plus de groupe*

❖ Etapes de l'algorithme :

- Fixer le nombre de clusters : k
- Choisir aléatoirement k tuples comme graines (centres)
- Assigner chaque tuple à la graine la plus proche
- Recalculer les k graines
- Tant que des tuples ont été changés
  * réassigner les tuples
  * recalculer les k graines

- C'est l'algorithme le plus utilisé

- exemple de k-means =2

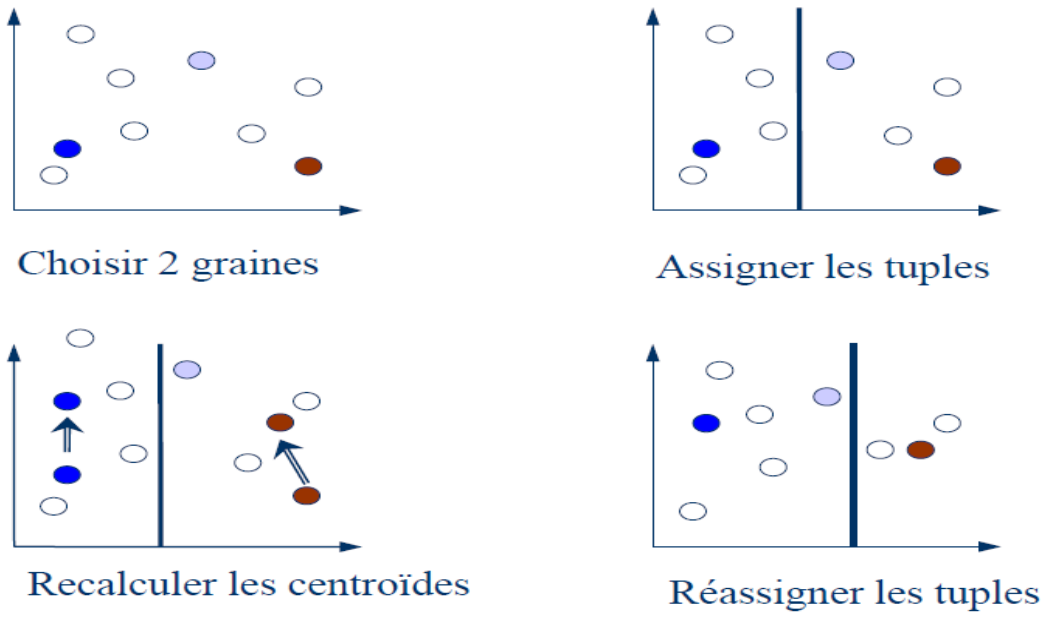


Figure 6 étape de l'algorithme k-means=2

Le schéma suivant présente la trajectoire des centres

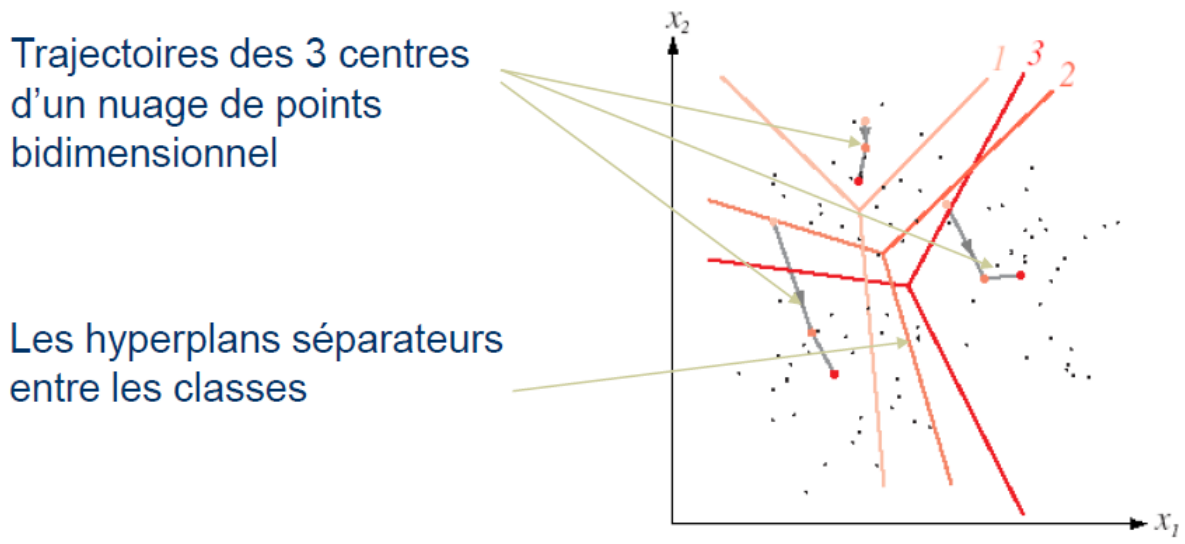


Figure 7 trajectoire des centres

K-Means : Exemple

A= {1,2,3,6,7,8,13,15,17}. Créer 3 clusters à partir de A

- On prend 3 objets au hasard. Supposons que c'est 1, 2 et 3. Ca donne C1= {1}, M1=1, C2= {2}, M2=2, C3= {3} et M3=3
- Chaque objet O est affecté au cluster au milieu duquel, O est le plus proche. Est affecté à C3 car dist(M3,6)< dist(M2,6) et dist(M3,6) <dist(M1,6)

  On a

  C_1= {1}, M_1=1,

  C_2={2}, M_2=2

  C_3= {3, 6, 7, 8, 13, 15, 17}, M_3=69/7=9.86

  - $dist(3, M\mathbf{2}) < dist(3, M\mathbf{3}) \rightarrow$ 3 passe dans $C2$. Tous les autres objets ne bougent pas.

$$C\mathbf{1} = \{1\}, M\mathbf{1} = 1, C\mathbf{2} = \{2,3\}, M\mathbf{2} = 2.5, C3 = \{6,7,8,13,15,17\}\ et\ M\mathbf{3} = 66/6 = 11$$

- $dist(6, M\mathbf{2}) < dist(6, M3) \rightarrow$ 6 passe dans C2. Tous les autres objets ne bougent pas. $C\mathbf{1} = \{1\}, M\mathbf{1} = 1, C\mathbf{2} = \{2,3,6\}, M\mathbf{2} = 11/3 = 3.67, C\mathbf{3} = \{7,8,13,15,17\}, M\mathbf{3} = 12$

- $dist(2, M\mathbf{1}) < dist(2, M\mathbf{2}) \rightarrow$ 2 passe en $C1$. $dist(7, M\mathbf{2}) < dist(7, M3) \rightarrow$ 7 passe en $C2$. Les autres ne bougent pas. $C\mathbf{1} = \{1,2\}, M\mathbf{1} = 1.5, C\mathbf{2} = \{3,6,7\}, M\mathbf{2} = 5.34, C3 = \{8,13,15,17\}, M\mathbf{3} = 13.25$

- $dist(3, M\mathbf{1}) < dist(3, M2) \rightarrow$ 3 passe en 1. $dist(8, M2) < dist(8, M3) \rightarrow$ 8 $passe\ en$ 2 $C\mathbf{1} = \{1,2,3\}, M\mathbf{1} = 2, C\mathbf{2} = \{6,7,8\}, M\mathbf{2} = 7, C\mathbf{3} = \{13,15,17\}, M\mathbf{3} = 15$

- Maintenant plus rien ne bouge

- **Réduction de dimensionnalité** :

  Le but d'un algorithme de réduction de dimensionnalité est de parvenir à "résumer" l'information présente dans les coordonnées d'un point en haute dimension **($\boldsymbol{x} \in \boldsymbol{R^n}$,n** grand_)par un nombre plus réduit de caractéristique ($y = f(x)$,$y \epsilon\ R^m, m < n$) Le but espéré est de préserver l'information "importante", de la mettre en évidence en la dissociant du bruit, et possiblement de révéler une structure sous-jacente qui ne serait pas immédiatement apparente dans les données d'origine en haute dimension. L'exemple le plus classique d'algorithme de réduction de dimensionnalité est l'Analyse en Composantes Principales (ACP) (4)

1.2.3. Apprentissage semi supervisé :

L'apprentissage semi-supervisé se situe entre l'apprentissage supervisé qui n'utilise que des données étiquetées et l'apprentissage non-supervisé qui n'utilise que des données non-étiquetées, (Peu de données étiquetées sont disponibles, c'est pour cette raison qu'elles seront combinées avec des données non étiquetées), cette combinaison permet l'étiquetage des données qui ne sont pas étiquetées

A titre d'exemple, soit une collection de pièces de monnaie, dont certains sont classés en types connus en termes de poids et de taille (figure 8). Nous voulons libeller les données non étiquetées selon les données étiquetées (figure.9). (3)

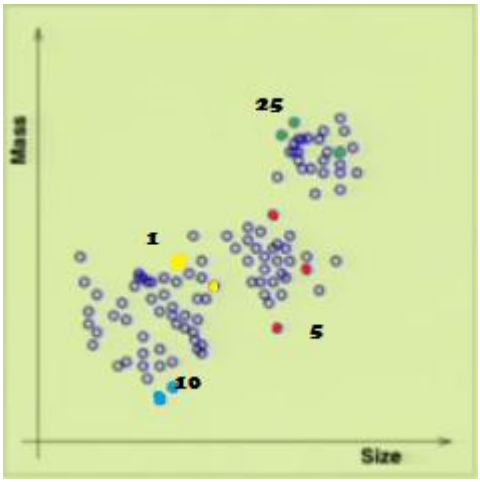


Figure 8 collection de pièces de monnaie

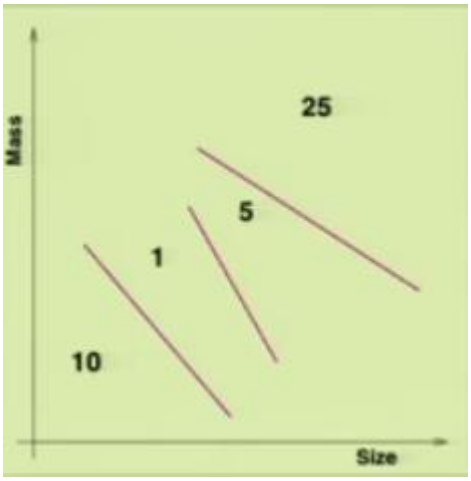


Figure 9

1.2.4. Apprentissage par renforcement :

l'apprentissage par renforcement (AR) consiste à apprendre quoi faire, comment associer des actions à des situations, afin de maximiser quantitativement une récompense. On ne dit pas à l'apprenant quelle action faire, mais au lieu de cela il doit découvrir quelles actions donnent le plus de récompense en les essayant.
son principe consiste à établir un système artificiel capable de juger des actions qui seraient les plus performantes en fonction de son état interne donc L'apprentissage par renforcement est différent de l'apprentissage supervisé. Ce dernier nécessite un superviseur qui dit à l'**agent** quelle action est correcte dans telle situation. Dans l'AR, l'**agent** n'a pas d'oracle à sa disposition, il interagit avec l'environnement qui lui donne un retour quantitatif sur les valeurs de ses actions (figure.10) (5)

* (**Un agent** est toute entité qui perçoit son environnement a travers des capteurs et qui agit sur son environnement a travers des effecteurs) (6)

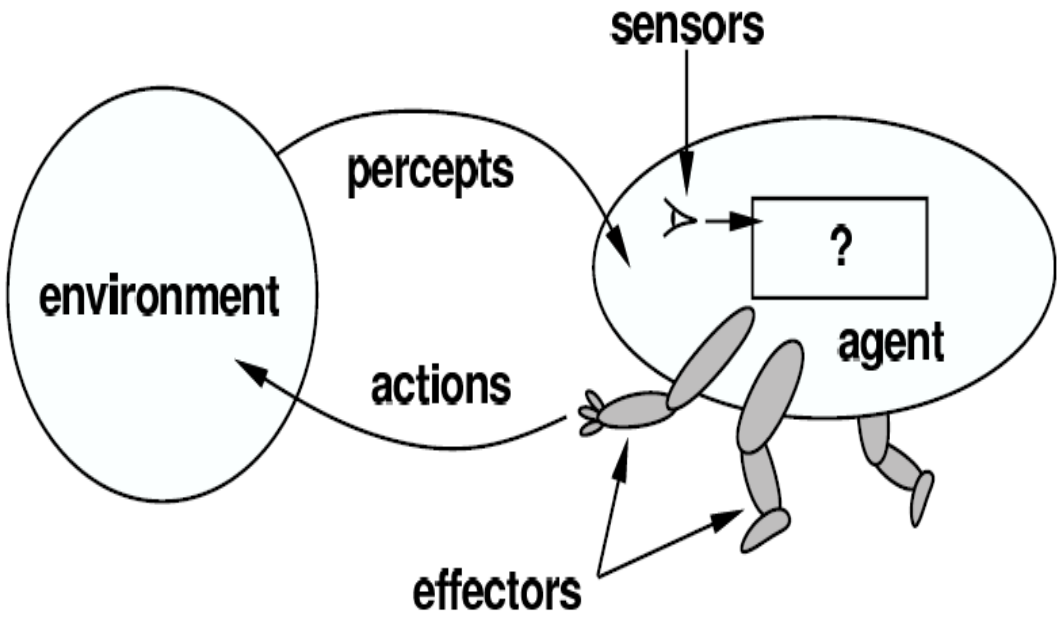


Figure 10

Exemple :

But : Passer le plus vite possible de O à G

Actions : {haut, bas, gauche, droite}

Perceptions : Triplet (x ; y ; r) ou (x ; y) est la position de l'agent, et r est une pénalité tant que l'agent n'a pas trouvé la sortie :

$$r = \begin{cases} 0 \; si \; (x,y) = (x_G, y_g) \\ -1 \; sinon \end{cases}$$

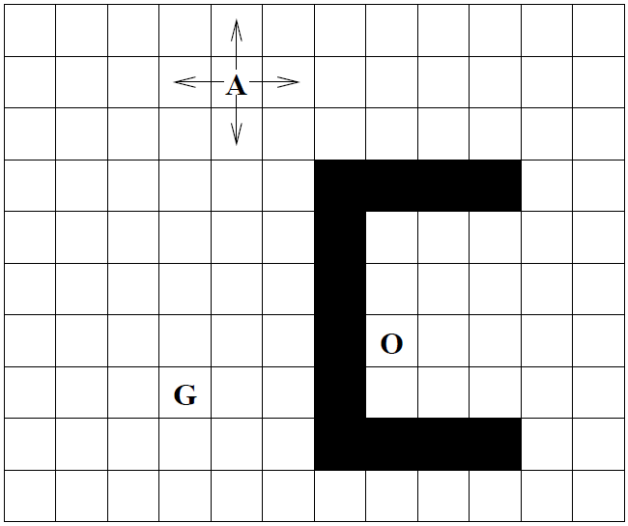
Condition initiale : Les perceptions valent (x0,y0,-1).

Problème : Maximiser la récompense totale suivant le temps (retour) :

$$V(x) = \sum_{t=0}^{\infty} r_t$$

Quand on a observé la suite de perceptions $\{(x_0, y_0, r_0), (x_1, y_1, r_1), \ldots\}$.

Moyen : Trouver une politique $\pi : S \rightarrow A$ ou S dénote l'ensemble des perceptions et A l'ensemble des actions, telle que si on commence dans n'importe quel état et que l'on suit ensuite, on atteint toujours G en un nombre minimum détapes. (6)



Après avoir vus les type d'apprentissage automatique voici un tableau comparatif entre les trois type majeures (supervisé, non supervisé et par renforcement) : (6)

Type d'apprentissage	But de l'agent	Les perceptions contiennent
Apprentissage supervisé	Faire la bonne action	L'action qu'il fallait faire
Apprentissage non supervisé	Structurer ses perceptions	Aucun indice
Apprentissage par renforcement	Maximiser ses perceptions	Une récompense ou une punition

Tableau 1 comparaison entre les types d'apprentissage

les algorithmes utilisé dans l'apprentissage automatique rencontre des problèmes et parmi ces problèmes la linéarité, la majorité des données ne sont pas linéairement séparable à vrai dire aucun modèle d'un système réel n'est vraiment linéaire donc on peut dire que , la linéarité est assez particulière D'autre part, la détection des relations linéaires a été l'objet de nombreuses recherches dans les statistiques et l'apprentissage automatique depuis des décennies après des recherches ils ont arrivé a une méthode qui est « le noyau » si un problème est non-linéaire, au lieu d'essayer d'adapter un modèle non linéaire, on utilise les noyaux

2. Les noyaux :

Depuis une petite dizaine d'années, une nouvelle famille d'algorithmes d'apprentissage basés sur la notion de noyaux, fait l'objet d'intenses recherches. Les noyaux, proposés par V.Vapnik pour les machines à vecteurs de support (SVM), permettent de définir des mesures de similarité non linéaires. En simplifiant, la fonction noyau calcule un produit scalaire (utilisé comme mesure de similarité) entre deux éléments à traiter.

2.1. La terminologie du mot noyau :

Le terme ‘’noyau’’ est dérivé d'un mot qui peut être retracée autour de l'année 1000 qui signifie à l'origine une graine (contenue dans un fruit) ou moelleux (habituellement comestible) contenue dans la coque d'une noix ou de fruits à noyau. L'ancien sens est

maintenant obsolète. Il a été utilisé la première fois en mathématiques quand il a été de nie pour les équations intégrales dans lequel le noyau est connue et l'autre fonction(s) inconnu, mais il a maintenant plusieurs significations en mathématiques. Dans l'apprentissage automatique le terme astuce noyau a été utilisé la première fois en 1998. (7)

2.2. La fonction noyau :

Les noyaux constituent une nouvelle famille d'algorithme d'apprentissage utilisable pour les données qui ne sont pas linéairement séparable. Elles correspondent à une mesure de similarité entre des entrées x et x'.

Les méthodes à noyau recherchent des relations linéaires dans un espace de redescription . Les éléments d'entrée sont comparés par des produits scalaires de leur représentation dans un tel espace.

Les Fonctions noyaux sont utilisées dans de nombreuses applications pour fournir un simple pont entre la linéarité et la non-linéarité pour les algorithmes. En reconnaissance de formes, nous voulons que nos algorithmes analysent et classer les données de manière efficace. Si la frontière entre deux ensembles de points est trop tordue, l'algorithme va prendre beaucoup de temps pour converger. Même quand il converge, il pourrait ne pas être la limite la plus optimale.

La principale caractéristique des fonctions noyaux dans l'apprentissage automatique est leur approche distincte à ce problème. Au lieu de prendre la route difficile de classer les données en dimension inférieure en mettant une ligne vraiment sinueuse, les fonctions noyau font une projection des données dans des espaces de dimensions supérieures, dans l'espoir que les données sont plus facilement séparés. aussi Il n'ya pas de contraintes sur la forme de cette projection, qui pourrait même conduire à des espaces de dimension infinie.

Maintenant, comment est-il facile de trouver cette fonction? La grande chose au sujet de cette fonction de projection est qu'elle est besoin d'être calculé, et pour la calculé avec il ya deux approches : (8)

2.2.1. La méthodes directe :

La méthode directe est une approche pour calculer la fonction de noyau.

comme un exemple on considère un espace de donnée X de deux dimensions $X \subseteq \mathbb{R}^2$ et un espace redescription

$$\varphi: x = (x_1, x_2) \rightarrow \varphi(x) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \in H = \mathbb{R}^3 \quad (1)$$

on calcule la fonction noyau de cet espace de description de la manière suivante :

$$\begin{aligned} k(x,z) &= < \varphi(x), \varphi(z) > \\ &= < (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2) > \quad (2) \\ &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2\, x_1.x_2\, z_1.z_2 \end{aligned}$$

On calcule $\varphi(x)$, puis $\varphi(z)$, ensuite on calcule leur produit scalaire.
Le problème dans cette méthode est le nombre de calcule. Si φ est une projection d'un espace d'entrée de dimension inférieure à un espace de fonction dimensions très élevé le calcule de $\varphi(x)$ et $\varphi(z)$ et leur produit scalaire peut nécessiter un temps très long pour terminer.
La fonction φ prend les données à partir d'espace à 2-dimensions à un espace à 3-dimensions de telle manière que les relations linéaires dans l'espace H correspondent à des relations quadratiques dans l'espace d'entrée X voire la figure11 (9)

Figure 11

2.2.2. L'astuce noyau :

Il s'agit d'une approche différente de calcul de la fonction noyau qui accélère considérablement le temps de calcul. Considérons le même espace de donnée X de deux dimensions $X \subseteq \mathbb{R}^2$ et la fonction φ (1). Ensuite, selon (2) la fonction du noyau est donné par :

$$k(x.z) = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1.x_2 z_1.z_2$$

Noter que :

$$x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1.x_2 z_1.z_2 = (x_1 z_1 + x_2 z_2)^2 = < x, z >^2$$

Et donc :

$$k(x.z) = < x, z >^2$$

Est une fonction noyau pour φ. Le dernier calcul est nettement plus rapide que le calcul des coordonnées de chaque vecteur dans l'espace des fonctions alors on prend le produit scalaire. L'astuce noyau nous permet d'éviter le calcul des coordonnées dans l'espace d'entrée de chaque vecteur d'entrée et elle donne le résultat directement dans l'espace de redescription

A noter également que l'espace de redescription n'est pas déterminée de manière unique par la fonction de noyau

Pour :

$$\varphi: x = (x_1, x_2) \rightarrow \varphi(x) = \left(x_1^2, x_2^2, \sqrt{2} x_1 x_2\right) \in H = \mathbb{R}^3$$

On a la même fonction noyau $k(x.z) = < x, z >^2$ Pour simplifier, nous choisissons habituellement une fonction noyau particulier et considérons l'un des φ pour cette fonction noyau comme notre espace de redescription. (9)

2.3. La matrice noyaux:

Pour simplifier la représentation des données et également de mieux visualiser les noyaux, nous faisons usage de la notation de la matrice du noyau. Étant donné un ensemble de données fini $S = \{x_1, \dots, x_l\} \subset \Re^n$, nous représentons toutes les valeurs possibles de la fonction de noyau pour l'ensemble de données dans la notation de la matrice du noyau suivantes: (Bucatanschi, 2006)

$$K = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & k(x_1,x_3) & \dots & k(x_1,x_l) \\ k(x_2,x_1) & k(x_2,x_2) & k(x_2,x_3) & \dots & k(x_2,x_l) \\ k(x_3,x_1) & k(x_3,x_2) & k(x_3,x_3) & \dots & k(x_3,x_l) \\ \dots & \dots & \dots & \dots & \dots \\ k(x_l,x_1) & k(x_l,x_2) & k(x_l,x_3) & \dots & k(x_l,x_l) \end{bmatrix}$$

Et $\boldsymbol{k_{ij} = k(x_i, x_j), ou\ i, j = 1..l}$

2.4. Les type noyaux :

Les noyaux les plus populaires sont :

2.4.1 Noyaux linéaire :

C'est l'une des fonctions les plus simples du noyau. C'est juste le produit scalaire <x,y> plus une constante c. (2012)

$$k(x,z) = < x,z > + c$$

2.4.2. Noyaux polynomial :

Ces noyaux sont utiles pour des problèmes où toutes les données d'apprentissage sont normalisées. Nous pouvons utiliser polynôme de n'importe quel ordre dépend du cas à la main

pour un degré d le polynome noyau définit comme suit :

$$k(x,z) = (< x,z > +R)^d$$

R et d son des paramètres (8)

2.4.3. Noyaux Gaussiennes :

Ceci vient dans la catégorie de fonctions de base radiales. Le paramètre 'sigma' détermine la variance de la distribution. Le bon accord de ce paramètre joue un rôle majeur dans l'exécution de ce noyau. Si elle est surestimée, l'exponentielle va se comporter de façon presque linéaire et la projection de dimension supérieure va commencer à perdre son pouvoir non-linéaire. D'autre part, si elle est sous-estimée, la frontière de décision sera très sensible au bruit dans les données d'apprentissage. Il ya quelques fonctions de base radiales comme exponentielle, Laplacian, ANOVA, etc Les formulations sont plus ou moins similaire à cela.

Gaussiennes sont des noyaux les plus utilisés ont été étudiées dans les champs voisins

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

2.5. Les noyaux pour les données structurées :

Les données structurées telles que les arbres et les graphes sont des objet pouvant être décomposés en sous objets jusqu'à atteindre une unité atomique, la comparaison de tels objet a été abordée depuis longtemps dans plusieurs domaines de recherche. la méthode a noyaux est renouvelé ces recherches dans la mesure o ces méthodes sont générique et peuvent s'appliquer à une grande variété de domaines (10)

2.5.1. Noyaux sur les mots :

Les séquences de caractères sont considérées comme faisant partie des données structurées car une séquence peut être décomposée en sous-partie ainsi que les séquences possèdent la propriété des données structurées vue dans le paragraphe précédent. Les séquences sont généralement rencontrées dans les domaines liés à la bioinformatique mais aussi dans les documents en langage naturel. De plus, les mots composés sont très présents dans le domaine de la chimie et les domaines connexes où il n'est pas rare d'avoir des noms de molécules composées.

Il existe plusieurs noyaux qui traitent les mots tels que (11)

- Le noyau p-Spectrum
- Le noyau All-SubSequences
- Le noyau p-Fixed length SubSequence
- Le noyau String Subsequence (SSK)

Le noyau Séquence marginalisée

2.5.2. Les noyaux sur les arbres :

Les arbres sont des structures de données permettant de représenter efficacement des données organisées de manière hiérarchique. Ainsi, ils sont communément utilisés dans de nombreux domaines. La majorité des documents structurés et semi-structurés, tels que les documents XML, sont représentés de manière arborescente. Ainsi, il peut être intéressant de tenir compte de cette structure dans l'évaluation des critères de similarités entre ces différents documents. Dans les chapitres qui suivent nous présentons plus de détail sur les noyaux d'arbre qui constituent l'objet de notre mémoire. (11)

2.5.3. Les noyaux sur les graphes :

Le graphe est une structure de donnée très utilisée dans le domaine informatique pour modéliser des informations structurées complexes. Les

séquences et les arbres cités précédemment peuvent être vus comme des graphes acycliques orientés (dans le cas d'une séquence, le graphe est de degré maximum 1).
La conception d'un noyau nécessite une définition de la similarité entre deux graphes.
Pour cela, deux approches ont été proposées

- La première consiste à déterminer si les deux graphes sont isomorphes (ils ne se distinguent que par l'ordre des nœuds) ou à déterminer le nombre de sous-graphes isomorphes communs. Cependant, il est connu que ce problème est fortement combinatoire.
- La deuxième approche consiste à projeter le graphe G dans un espace vectoriel où chaque dimension est indexée par un graphe H tel que la valeur de la projection de G sur cet axe représente la fréquence d'occurrence de H, en tant que sous-graphe, dans G.

Il est alors possible de concevoir un noyau qui identifie certaines propriétés dans les sous-graphes H. En particulier, ce noyau peut effectuer le produit scalaire dans l'espace vectoriel en se limitant aux sous-graphes qui sont des chemins hamiltoniens (H est un chemin hamiltonien de G si et seulement si H est un sous-graphe de G et si H est de même ordre que G, i.e. H contient tous les nœuds de G exactement une fois). (11)

chapitre 2:
les noayaux sur les arbres

1. Introduction :

Cette section aborde les structures de donnée sous forme arborescente dans les algorithmes d'apprentissage automatique.

Les Structures de données arborescentes sont utilisées pour modéliser les données de plusieurs domaines, Dans le traitement du langage naturel, des arbres syntaxiques sont modélisés comme des arbres étiquetés commandés. Dans la reconnaissance de formes, une image peut être représentée par un arbre dont les sommets sont associés à des composants d'image, tout en conservant les informations concernant la structure de l'image. Dans le raisonnement automatisé, de nombreux problèmes sont résolus par la recherche et l'espace de recherche est souvent représenté comme un arbre dont les sommets sont associés avec les Etats de la recherche et des bords représentent des étapes inférence. Les Données semi-structurées aussi comme des documents HTML et XML peuvent être modélisés par des arbres étiquetés commandés.

Dans ce chapitre, nous montrons que les noyaux d'arbres sont très utiles dans le traitement des données structurés et semi-structuré. Nous commençons par donner quelques définitions :

2. Définition :

2.1. un arbre :

un arbre selon est un graphe non orienté, connecté et sans cycle (acyclique) avec certains propriétés:

- ✓ chaque sommet (nœud) a un degré externe $d^-(r) = 1$ sauf la racine r qui a $d^-(r) = 0$
- ✓ les feuille sont des sommets avec $d^+(r) = 0$
- ✓ Les nœuds interne sont des sommets avec$d^+(r) \neq 0$.
- ✓ les nœuds connectés avec des nœuds internes (pères) connus comme leur fils.
- ✓ deux fils avec le même père sont des frères.

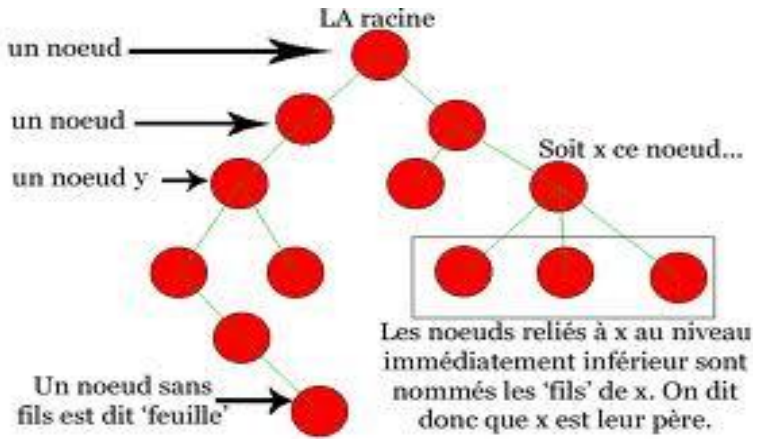


Figure 12 shéma d'un arbre

2.2.Un arbre ordonné :

un arbre ordonné est celle où les fils d'un nœud ont un ordre fixe, $fils_1(v) \dots, fils_{d^+(v)}(v)$ (12)

2.3. Arbre propre :

Un arbre propre est un arbre ayant une racine et au moins un nœud fils.

2.4. un sous-arbre complet :

Un sous arbre complet d'un arbre T à un nœud n est l'arbre obtenu en prenant tous les nœuds et arcs à partir du nœud n . On notera $\mathcal{T}(n)$ l'arbre complet induit par le nœud n de T.

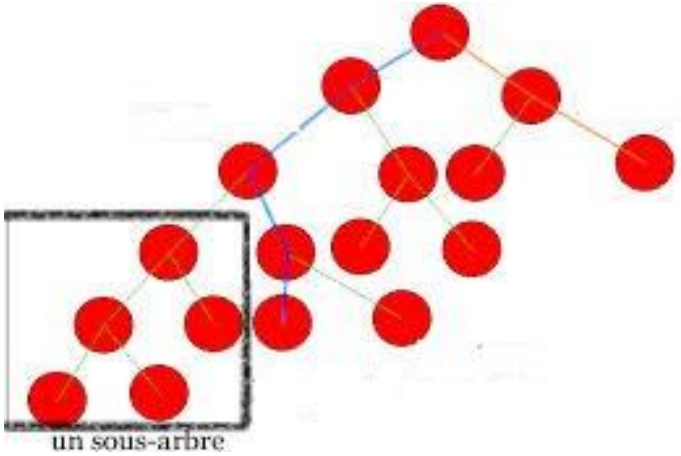


Figure 13 sous arbre d'un arbre T

2.5 Un sous-arbre Co-enracinée :

Un arbre S est dit sous-arbre Co-enraciné d'un arbre propre T si et seulement si :

- ✓ S est un arbre propre.
- ✓ la racine de S $(r(S))$ est identique à la racine de T $(r(T))$.
- ✓ Touts les nœuds de S sont des nœuds de T.
- ✓ S peut être obtenu à partir de T en supprimant des sous-arbres de $fils_1(r(T)$ (Sujeevan Aseervatham)

La figure.10 présente quelques arbres co-enracinée de l'arbre T :

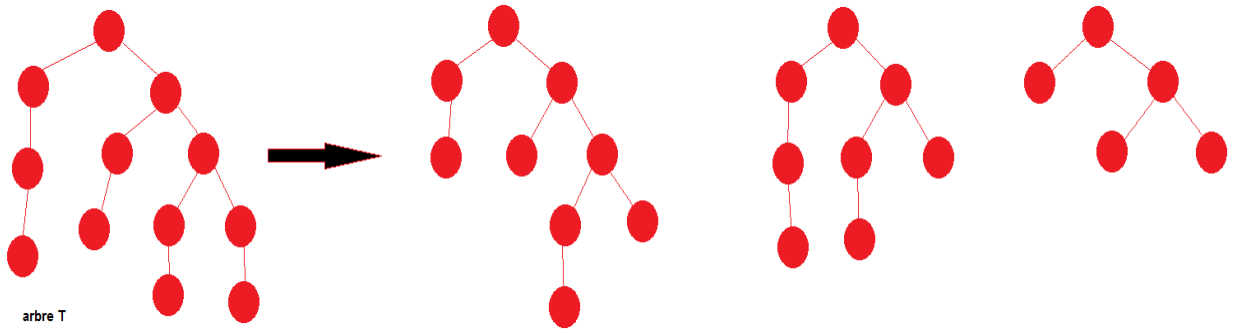


Figure 14 les arbres Co-enracinés de l'arbre T

La figure.15 présente quelques arbres qui ne sont pas Co-enracinée de l'arbre T :

Figure 15 arbres non Co-enracinées de l'arbre T

2.6. noyau de convolution :

Le noyau de convolution appelé R-noyau, il a été développé par D.Haussler. Il permet de définir un cadre général pour les noyaux appliqués aux données structurées telles que les arbres et les graphes. Il est une généralisation des noyaux sur les mots et les noyaux d'arbres qu'on va les présenter prochainement.

D'abord, Les données structurées comme on les a déjà présenté sont des objets pouvant être décomposés en sous-objets jusqu'à atteindre une unité atomique (comme les feuilles dans le cas d'un arbre). Donc L'idée du R-noyau est de calculer la similarité entre deux éléments x et y en les décomposant. Plus formellement, soit x un élément appartenant à un ensemble X d'éléments structurés de même type, x peut être décomposé en sous éléments $\overline{x} = (x_1, \dots, x_d)$ où x_d peut être un élément structuré ou non appartenant à X_d . On définit la relation binaire :

$$R: X_1 \times \dots \times X_D \to X$$

Qui associe les parties d'un élément x à x et la relation inverse :

$$R^{-1}: \{\overline{x} = (x_1, \dots, x_{D)} | R(\overline{x}, x)\}$$

Qui retourne pour x l'ensemble de toutes les décompositions possibles.

Le noyau de convolution (R-noyau) pour deux éléments x et y de X est alors :

$$k_R(x, y) = \sum_{\overline{x} \in R^{-1}(x)} \sum_{\overline{y} \in R^{-1}(y)} \prod_{i=1}^{d} k_i(x_i, y_i)$$

Avec k_i un noyau calculant la similarité entre les éléments $\boldsymbol{x_i}$ et $\boldsymbol{y_i}$ de même structure i.e. $x_i, y_i \in X_i$. Il est facile de montrer que si les k_i sont des noyaux valides alors k_R (R-noyau) est valide. En effet, si k_i est valide alors sa matrice de Gram est semi-définie positive et le produit et la somme de matrices semi-définies positives sont des matrices semi-définies positives

De plus, le R-noyau peut être défini par récurrence lorsque les parties x_i d'un élément x sont de même type de structure $x_i, y_i \in X$. Le critère d'arrêt est défini pour l'élément atomique (par exemple une feuille pour une structure arborescente). La possibilité de

décomposer le calcul de la similarité d'éléments permet de traiter aisément des structures complexes. Toutefois, elle nécessite, en contrepartie, un temps de calcul non négligeable. Ainsi, il est nécessaire de spécialiser ce noyau selon le type de structure afin de réduire la complexité. (2)

3. Les noyaux sur les arbres :

Comme on a déjà dite que les noyaux d'arbres sont prolongés dans plusieurs domaines grâce a leur forme hiérarchique qui nous permet de donner une bonne représentation des données.

Comme les noyaux d'arbre ont beaucoup d'intérêts ils ont quelques inconvénients parmi leur majeurs inconvénients est le temps d'exécution des algorithmes qui les traites il ont une complexité quadratique dépend sur le nombre des nœuds de l'arbre pour cela et pour rendre les noyaux d'arbres plus performantes deux algorithmes ont été proposé le 1^{er} « subtree »(ST) de Vishwanathan and Smola et le $2^{éme}$ « subset tree »(SST) de Collins and Duffy on va mieux expliquer et détaillé sur le fonctionnement récursif de ces deux algorithme selon (12)

Tous les noyaux présentés dans cette section peuvent être définies par un espace d'intégration explicite à partie l'espace de tous les arbres finis peut être étiquetés avec un ensemble A à un espace vectoriel F, dont les coordonnées sont indexées par un sous-ensemble I des arbres aussi peut être étiquetés ou non . Comme d'habitude, nous utilisons φ pour désigner l'espace de redescription

$$\varphi: T \rightarrow (\varphi_s(T))_{s \in I} \in F$$

L'objectif est de divisé l'espace de redescription pour lequel le noyau correspondant peut être évaluée à l'aide d'un calcul récursif qui procède de bas en haut à partir des feuilles à la racine des arbres. La relation de récurrence de base relie les valeurs de certaines fonctions d'un nœud donné avec les valeurs de fonction à ses fils. La base de la récursivité est définir les valeurs dans les feuilles, faire en sorte que le calcul est bien définie

À titre d'exemple pour comprendre le fonctionnement récursif voilà un exemple de comptage de sous arbre

Pour le calcule récursif sur un arbre considèrent l'évaluation de nombre N(T) de sous-arbres Co-enracines d'un arbre T.

Il est clair que pour un nœud feuille v il n'y a pas de sous-arbres Co-racines de $\mathcal{T}(v)$, Alors$N(\mathcal{T}(v)) = 0$.

Supposons que nous connaissons la valeur de $N(\mathcal{T}(v_i))$ pour chacun des nœuds $v_i = fils_i(r(t)), i = 1, \ldots, d^+(r(t))$ qui sont des fils de la racine $r(T)$ de T. Pour créer une sous-arbore Co-enracinés de T, nous devons inclure tous les nœuds $r(T), v_1, \ldots, v_{d^+(r(t))}$, mais nous avons la possibilité d'inclure l'un des Co-enracinée de $\boldsymbol{\mathcal{T}}(\boldsymbol{v_i})$ ou en laissant simplement le nœud $\boldsymbol{v_i}$ comme une feuille. Ainsi, pour les nœuds vi on a $N(\mathcal{T}(v_i)) + 1$ possibilités. Ces observations conduisent à la relation récurrence suivante de $N(T)$ pour un arbre propre T

$$N(T) = \prod_{i=1}^{d^+(r(T))} N(\mathcal{T}(fils_i(r(T)))) + 1)$$

Avec$N(\mathcal{T}(v)) = 0$, pour v est un feuille

Maintenant nous allons décrire les deux noyaux sur les arbres

3.1. Subtree kernel (ST) :

Pour mieux expliquer la figure(12) , définit un sous-arbre (ST) comme un nœud d'un arbre avec tous ses descendants par exemple:

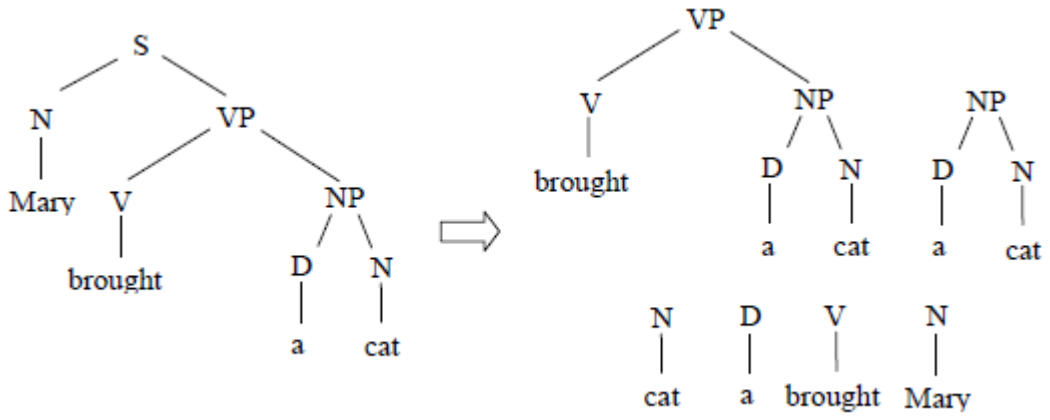


Figure 16 arbres sybtaxique avec ses sous-arbres (ST)

L'espace associée avec le noyau ST est indexée par $\boldsymbol{I} = \boldsymbol{\mathcal{T}}$ l'ensemble de tous les arbres propres avec la fonction de projection représenté par :

$$\varphi_S^r(T) = \begin{cases} 1 \ si\ S\ est\ uns\ sous\ arbre\ co-enracinnée\ de\ T \\ 0\ sinon \end{cases}$$

Le noyau associé est défini comme :

$$\mathrm{k_r}(\mathrm{T_1}, \mathrm{T_2}) = < (\varphi^{\mathrm{r}}(\mathrm{T1}), \varphi^{\mathrm{r}}(\mathrm{T2}) > = \sum_{S \varepsilon \mathcal{T}} \varphi_S^{\mathrm{r}}(\mathrm{T_1})\, \varphi_S^{\mathrm{r}}(\mathrm{T_2})$$

Si l'un des arbres T_1 ou T_2 est une feuille alors:

$$k_r(T_1, T_2) = 0$$

Aussi pour un arbre non propre :

$$\varphi^r(T) = 0$$

En outre, si $d^+(r(T_1)) \neq d^+(r(T_2))$ puis $k_r(T1, T2) = 0$ car un sous arbre Co-enracinée de $\boldsymbol{T_1}$ ne peut pas être un arbre Co-enracinée de $\boldsymbol{T_2}$.
Supposons donc que

$$d^+(r(T_1)) = d^+(r(T_2))$$

Maintenant, pour introduire le calcul récursif, supposons que nous avons évalué le noyau entre les sous-arbres complets sur les fils correspondant de $\boldsymbol{r}(\boldsymbol{T_1})$ et $r(T_2)$; c'est ce que nous avons calculé

$$\mathrm{k_r}(\mathcal{T}(\mathrm{fils_i}(\mathrm{r}(\mathrm{T1}))), \mathcal{T}(\mathrm{fils_i}(\mathrm{r}(\mathrm{T2})))),$$

$$\text{pour } i = 1, \ldots, d^+(r(T1))$$

Nous avons maintenant

$$k_r(T_1, T_2) = \sum_{S \in \mathcal{T}} \varphi_S^r(T_1)\varphi_S^r(T_2)$$

$$= \prod_{i=1}^{d^+(r\,(T_1))} \sum_{S_i \in \mathcal{T}_0} \varphi_{s_i}^r\left(\mathcal{T}\ \left(fils_i(r\ (T1))\right)\right)\varphi_{s_i}^r(\mathcal{T}\ \left(fils_i(r\ (T2))\right))$$

Où $\mathcal{T}_0$ désigne l'ensemble de tous les arbres, propre et non propre, puisque les sous-arbres Co- enracines de T sont déterminés par une combinaison de sous arbre Co-enracines de $\mathcal{T}\ \left(fils_i(r\ (T))\right), i = 1, \ldots, d^+(r(T_1))$

Comme il n'y a qu'un seul sous-arbre Co- enraciné, nous obtenons la récursivité suivante :

$$k_r(T_1, T_2) = \prod_{i=1}^{d^+(r\,(T_1))} (k_r\left(\mathcal{T}\left(fils_i(r\ (T1))\right)\right), \left(\mathcal{T}\left(fils_i(r\ (T2))\right)\right) + 1)$$

3.1.1 Pseudo algorithme récursive ST :

```
Fonction SubTree_Kernel(v1, v2 : racine d'arbre) : entier
Variable      kern :entier ;
debut;
  Si (d+(v1) ≠ d+(v2)) ou (d+(v1) = 0)  ou  ([1])  alors
        return 0;
 sinon
Kern=1
 pour  i  de  1   d+(v1)faire
          kern = kern ∗ (subTree_kernel(fils_i(v1), fils_i(v2)) + 1);
        SubTree_kernel:= kern
  fin ;
fin.
```

[1] Ou(l'étiquette (v_1)≠ l'étiquette (v_2)) en cas ou l'arbre est étiqueté on doit ajouter cette partie qui vérifie si le nœud v_1 correspond au nœud v_2

3.1.2. L'Analyse de l'algorithme:

la complexité de cet algorithme est au pire des cas O(min(|T1|,|T2|)) où |T| est le nombre de nœuds de l'arbre T . Lorsque le parcours récursive traite les nœuds d'arbres au plus une fois. Si les degrés des nœuds ne sont pas égaux alors leur sous-arbres ne seront pas visités et le temps de calcule sera accéléré.

3.1.3 Critique :

Comme la complexité de ce noyaux est linaire alors il est efficace mais selon (14) et (15) l'ensemble de noyau ST est faible car il utilise uniquement les sous arbres Co-enracinées donc moins expressif. D'ou la nécessité d'un autre noyaux plus riche en matière des sous-arbres

3.2. ALL Sub Tree Kernel (SST) :

Ce noyau utilise une structure plus générale que noyau ST, a été expliquer la différence entre un sous arbre (ST) et un sous arbre(SST) dans comme (16) :
Le noyau sous-arbre (ST) est caractérisée par des structures qui contiennent tous les descendants d'un nœud racine jusqu'à ce que les feuilles alors que les(SSTs) peuvent contenir des sous-arbres internes, sans feuilles.
Par exemple (figure 17) :

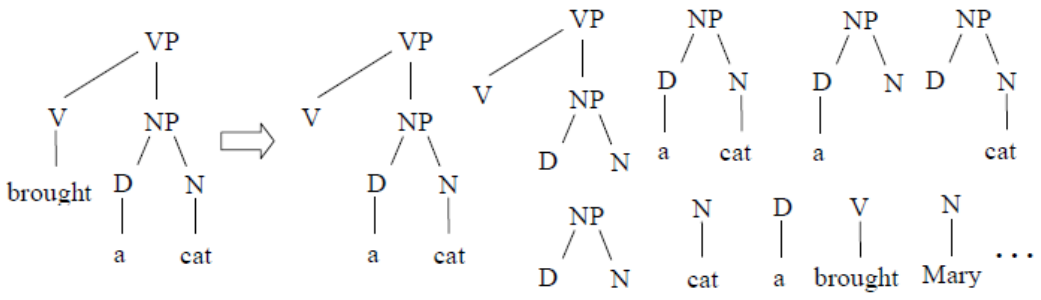


Figure17 : Un arbre syntaxique avec certains de ses arbres(SSTs)

L'espace de redescription de ce noyau représente tout les sous-arbre, il est indexé par $I = \mathcal{T}$, l'ensemble de tous les arbres propres avec :

$$\varphi_S(T) = \begin{cases} 1 \; si \; S \; est \; uns \; sous \; arbre \; de \; T \\ 0 \; sinon \end{cases}$$

Le noyau associé est défini comme étant :

$$K\,(T_1, T_2) \; = < \varphi(T_1)\,, \varphi(T_2) > = \sum_{S \in \mathcal{T}} \varphi_s\,(T_1)\varphi_s\,(T_2)$$

L'évaluation de ce noyau peut être réduite au cas de sous-arbres co-enracinés k_r en observant que :

$$K\,(T_1, T_2) = \sum_{v_1 \in T_1, v_2 \in T_2} k_r\,(\mathcal{T}(v_1)\,, \mathcal{T}(v_2)) \quad (3)$$

En d'autres termes le noyau SST peut être calculé par l'évaluation de noyau ST pour toutes les paires des nœuds dans les deux arbres. Cela conclu à partite de fait qui dit : quelque soit un sous-arbre d'un arbre T est un sous-arbre Co-enraciné du sous arbre complet $\mathcal{T}(v)$ pour un certain nœud v de T.

Plutôt que d'utiliser ce calcul, nous aimerions trouver une récurrence directe pour $K\,(T_1, T_2)$. Il est clair que si T_1 ou T_2 est un nœud feuille, nous avons :

$$k\,(T_1, T_2) \; = \; 0.$$

De plus, nous pouvons partitionner la somme (3) comme suit :

$$\begin{aligned} K\,(T_1, T_2) = & \sum_{v_1 \in T_1, v_2 \in T_2} k_r\big(\mathcal{T}\,(v_1), \mathcal{T}\,(v_2)\big) \\ & = k_r(T_1, T_2) + \sum_{i=1}^{d^+(r(T_1)} k\,(\mathcal{T}\,(fils_i\,(r\,(T_1)))\,, T_2) \\ & + \sum_{j=1}^{d^+(r(T_2)} k\left(T_1\,, \mathcal{T}\left(fils_j\big(r\,(T_2)\big)\right)\right) \\ & - \sum_{i=1}^{d^+(r(T_1)} \sum_{j=1}^{d^+(r(T_2)} k\,(\mathcal{T}(fils_i\,(r\,(T_1)))\,, \mathcal{T}(fils_j\,(r\,(T_2)))) \end{aligned}$$

Puisque les sous-arbres sont soit :

- des sous arbres Co-enracinés dans T_1 et T_2

- sous-arbres d'un fils de $r(T_1)$ ou un fils de $r(T_2)$

Cependant, ceux qui ne sont pas des sous arbres Co-enraciné avec T_1 ou T_2 seront comptés deux fois ce qui rend nécessaire de soustraire la somme finale.

Nous avons donc l'algorithme SST qu'on va le cité, en ce basant sur la construction d'une table indexée par les nœuds des deux arbres à l'aide de la programmation dynamique. On suppose que les nouds n_j de $v_1^j, \dots, v_{n_j}^j$ de l'arbre T_1 ont été ordonnée de manier que le parent d'un nœud a un indice après ce nœud. Alors, le dernier nœud $v_{n_j}^j$ est la racine de l'arbreT_j. Cet ordre sera utilisé pour indexer les tables $DP\ (i_1, i_2)$ et le $DP_r(i_1, i_2)$

Dans un premier temps,L'algorithme complète la table $DP_r(i_1, i_2)$ avec La valeur du noyau d'arbre Co-enraciné avant qu'il calcule le noyau tout sous-arbre dans table $DP(i_1, i_2)$

Nous supposons également que pour les fins de l'algorithme que $fils_k^j(i)$donne l'indice du k-nième fils du nœud indexé par i dans l'arbreT_j. De même,$d_j^+(i)$ est le degré externe du nœud indexé par ***i*** dans l'arbreT_j

3.2.1. Pseudo l'algorithme SST :

```
procedure     All_SubtreeKernel ((r1, r2: racine d'un arbre): tableau
Variable      supposons que les nœuds de Tj sont ordonnée : v1^j, ..., vnj^j
debut
  pour   i1  de 1  a   n1 faire
    pour   i2  de  1  a   n2 faire
      Si ( d1+(i1) ≠ d1+(i2)) ou (d1+(i1) = 0) ou (1) alors
                DPr (i1, i2) = 0
      sinon
                DPr (i1, i2) = 1
      pour k  de  1  a   d1+(i1) faire
        DPr (i1, i2) = DPr (i1, i2) * DPr (fils_k^1(i1), fils_k^2(i2) + 1);
      fin
```

[1] Ou l'étiquette de $(v_1) \neq$ de l'étiquette de (v_2) en cas ou les arbres sont étiquetés

```
fin
      fin
   pour   i1  de  1  a   n1
        pour   i2  de  1 a   n2
           si  ( d1+(i1) = 0) ou(d2+(i2) = 0)  alors
                    DP (i1, i2) = 0
           Else
                    DP (i1, i2) =  DPr (i1, i2)

           pour  j1  de1 a  d1+(i1) faire
              DP (i1, i2) = DP (i1, i2) + DP (fils2j1(i2), i2)
              pour  j2  de 1 a  d2+(i2) faire
                 DP (i1, i2) = DP (i1, i2) + DP (i1, fils2j2(i2))
                 pour  j1  de 1 a  d1+(i1) faire
                    DP (i1, i2) = DP (i1, i2) −  DP (fils2j1(i2), fils2j2(i2))
                 fin
           fin
        fin
    fin
```

3.2.2.L'Analyse de l'algorithme:

La structure de l'algorithme indique clairement que la complexité de l'évaluation du noyau peut être limitée par $O(|T_1||T_2|d_{max}^2)$, avec d_{max} est le degré externe maximum des nœuds dans les deux arbres.

3.2.3.critique:

La complexité de ce noyau n'est pas efficace car il est quadratique et malgré SST est plus riche d'information que le noyau ST. l'espace de redescription définie implicitement par ces noyaux est très éparse.(17)

chapitre3:
réalisation d'une boite a outils

1. Introduction :

Le présent chapitre se focalise sur la construction d'une boite à outil pour les méthodes de noyaux d'arbres. Nous commençons par l'implémentation des deux algorithmes décrits dans le chapitre précédent, à savoir les algorithmes ST et SST. Pour se faire, nous avons utilisé l'environnement MATLAB.

2. Le langage MALAB

2.1. Introduction :

MATLAB (**MAT***rix* **LAB***oratory*) est un environnement de programmation interactif pour le calcul scientifique, la programmation et la visualisation des données.

Il est très utilisé dans les domaines d'ingénierie et de recherche scientifique, ainsi qu'aux établissements d'enseignement supérieur. Sa popularité est due principalement à sa forte et simple interaction avec l'utilisateur mais aussi aux points suivants :

- ✓ **Sa richesse fonctionnelle** : avec MATLAB, il est possible de réaliser des manipulations mathématiques complexes en écrivant peu d'instructions. Il peut évaluer des expressions, dessiner des graphiques et exécuter des programmes classiques. Et surtout, il permet l'utilisation directe de plusieurs milliers de fonctions prédéfinie.
- ✓ **La possibilité d'utiliser les boites à outils (toolboxes)** : ce qui encourage son utilisation dans plusieurs disciplines (simulation, traitement de signal, imagerie, intelligence artificielle,...etc.).
- ✓ **La simplicité de son langage de programmation** : un programme écrit en MATLAB est plus facile à écrire et à lire comparé au même programme écrit en C ou en PASCAL.
- ✓ **Sa manière de tout gérer comme étant des matrices**, ce qui libère l'utilisateur de s'occuper de typage de données et ainsi de lui éviter les problèmes de transtypage.

A l'origine MATLAB était conçu pour faire principalement des calculs sur les vecteurs et les matrices d'où son nom '**Mat**rix **Lab**oratory', mais par la suite il a était amélioré et augmenté pour pouvoir traiter beaucoup plus de domaines.
MATLAB n'est pas le seul environnement de calcul scientifique existant car il existe d'autres concurrents dont les plus importants sont *Maple* et *Mathematica*. Il existe même des logiciels libres qui sont des clones de Matlab comme Scilab et Octave

2.2. Fonctionement du language MATLAB :

Il existe deux modes de fonctionnement:

1. **mode interactif**: MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'usager.

2. **mode exécutif**: MATLAB exécute ligne par ligne un "fichier.m" (programme en langage MATLAB).

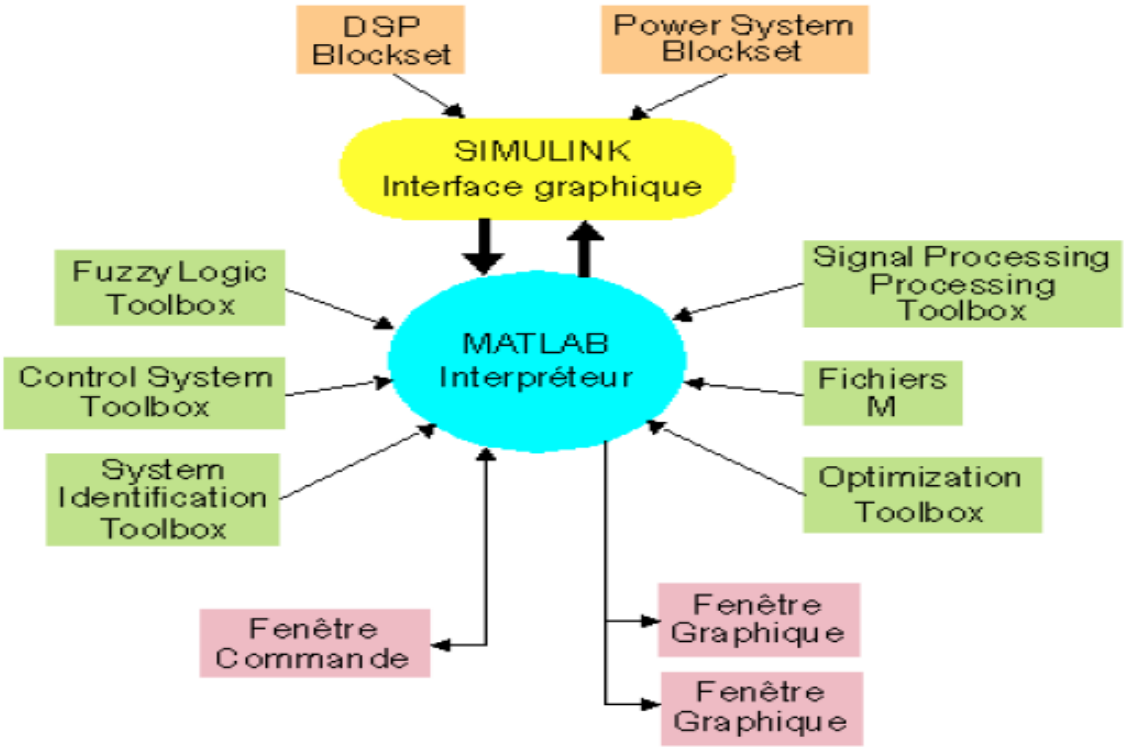


Figure 18 environnement MATLAB

- **Fenêtre Commande**: Dans cette fenêtre, l'usager donne les instructions et MATLAB retourne les résultats.
- **Fenêtres Graphique**: MATLAB trace les graphiques dans ces fenêtres.
- **Fichiers.m** : Ce sont des programmes en langage MATLAB (écrits par l'usager).
- **Toolboxes**: Ce sont des collections de fichiers.m développés pour des domaines d'application spécifiques (Signal Processing Toolbox, System Identification Toolbox, Control System Toolbox, u-Synthesis and Analysis Toolbox, Robust Control Toolbox, Optimization Toolbox, Neural Network Toolbox, Spline Toolbox, Chemometrics Toolbox, Fuzzy Logic Toolbox, etc.)
- **Simulink**: C'est l'extension graphique de MATLAB permettant de travailler avec des diagrammes en blocs.
- **Blocksets**: Ce sont des collections de blocs Simulink développés pour des domaines d'application spécifiques (DSP Blockset, Power System Blockset, etc.). (18)

2.3. Caractéristique du langage MATLAB :

Plusieurs caractéristiques de MATLAB en font un outil simple et puissant. MATLAB est un langage *interactif*, i.e. dès que vous entrez une commande, vous avez une réponse immédiate du logiciel. MATLAB s'occupe de déterminer le *type* et la *taille* des variables mises en mémoire, ce qui facilite la tâche du programmeur en lui permettant d'expérimenter et de se concentrer sur les problèmes à résoudre. Tous les types de MATLAB sont basés autour de la notion de matrice. Un scalaire est une matrice de taille 1×1, un vecteur est une matrice de taille $n \times 1$ ou $1 \times n$, etc. Une autre idée de base de MATLAB est que si vous vous demandez quel est le calcul qu'une commande exécutera, en général il s'agit de l'opération la plus naturelle.

Finalement, l'apprentissage de l'approche *vectorielle* de MATLAB sera utile pour ceux qui auront à utiliser des ordinateurs à architecture vectorielle ou parallèle. (18)

3. La boite à outils (ToolBox)

3.1. Définition :

Ensemble de programmes utilitaires. Dans le domaine de la programmation, l'expression "boîte à outils" désigne un ensemble de routines et/ou de bibliothèques proposées par un éditeur. Elles peuvent être intégrées à un programme en cours de développement au niveau du texte ou au niveau des modules objets.

3.2. Création d'une boite à outils en MATLAB :

Voici les étapes à suivre afin de créer une Toolbox :

- ✓ vérifier qu'aucune de vos fonctions ne porte le même nom qu'une fonction MATLAB avec

```
which-all<nom_fonction>
```

en remplaçant à chaque fois <nom_fonction> par le nom d'une de vos fonctions;

- ✓ vérifier que chaque fichier contient bien une fonction avec l'entête standard ;

```
function [<sorties>] = myfun(<entrées>)
%MYFUN <Description de myfun en une ligne>
%     MYFUN(<entrées>) explication des paramètres
d'entrée
%
%     [<sorties>] = MYFUN(...) explication des
paramètres de sortie%
%     See also <fonctions reliées>

% Auteur
```

```
% Date
% Version
%

% Code de la fonction
```

- ✓ réunir tous les fichiers et sous-dossiers au sein d'un même dossier ;

- ✓ créer le fichier Contents.m dans ce même dossier de la forme :

```
% NOM DE LA TOOLBOX
% <Version Toolbox> <version MATLAB minimum
requise> <Date>
%
% Titre1
%   fonction1_1                - Description
%   fonction1_2                - Description
%   fonction1_3                - Description
%   ...
%
% Titre2
%   fonction2_1                - Description
%   fonction2_2                - Description
%   fonction2_3                - Description
%   ...
```

cette étape peut être effectuée automatiquement en allant dans le menu de l'onglet Current Directory > Contents Report ;

3.3. l'utilisation de la boite à outils :

Lors de l'acquisition d'une nouvelle Toolbox MATLAB, il est nécessaire afin de pouvoir l'utiliser correctement, d'ajouter ses fonctions au PATH de MATLAB.

La Toolbox se présente sous forme de dossier unique.

✓ Ce dossier sera placé à l'emplacement de votre choix.

Le plus simple consiste souvent à utiliser le dossier retourné par :

```
fullfile(matlabroot,'toolbox')
```

Mais ceci comporte plusieurs points faibles :

- Vous ne pourrez pas copier les fichiers dans ce dossier si vous n'avez pas les droits en écriture sur le dossier d'installation de MATLAB.
- En cas de désinstallation de MATLAB, vous risquez de perdre les Toolbox installées lors de la suppression des dossiers.
- Si vous changez de version de MATLAB, il vous faudra penser à copier les Toolbox dans le nouveau dossier d'installation.

Donc il est judicieux de stocker vos toolbox dans un autre dossier.

✓ On ajoutera alors ce dossier au PATH de MATLAB avec la commande :

```
addpath( genpath(<chemin dundossier>))
```

✓ Si l'on souhaite qu'à chaque démarrage de MATLAB cette nouvelle Toolbox soit utilisable, on sauvegardera le nouveau PATH ainsi créé avec la fonction :

```
savepath
```

✓ Dans le cas contraire, on exécutera à nouveau l'étape 3) lorsque l'on voudra l'utiliser.

Remarque : l'étape 3 peut aussi se faire manuellement via :
- le bouton Start => Desktop Tools => Path ;
- le menu File => Set Path...

3.4. le PATH de MATLAB

Le PATH de MATLAB est la liste des chemins des dossiers auxquels MATLAB a accès.

Elle contient par défaut :

- les dossiers fournis avec MATLAB et les autres produits MathWorks installés. Ces dossiers se situent dans le chemin renvoyé par la ligne

```
fullfile(matlabroot,'toolbox')
```

- le chemin renvoyé par :

```
userpath
```

Cette liste est consultable en tapant la commande

```
path
```

On pourra rajouter des dossiers à cette liste (installation de nouvelles toolbox par exemple)

Tout fichier qui n'est inclus ni dans le "Current Directory" ni dans un dossier de cette liste ne sera pas visible directement par MATLAB, il faudra spécifier son chemin entier
(18)

4. Implémentation :

4.1 Structure de donnée utilisée :

Comme on a déjà définit dans le chapitre précédent l'arbre est un cas spéciale d'un graphe, un graphe orienté connexe sans cycle donc on va implémenter les arbres utilisé dans nos algorithmes comme un graphe

4.1.1 La représentation d'un graphe :

4.1.1.1 Définition :

Un graphe est un ensemble de sommets et un ensemble d'arêtes qui relient chacune une paire de sommets. Nous utilisons les noms 0 à V-1 pour les sommets dans un graphe à V-sommet. (19)

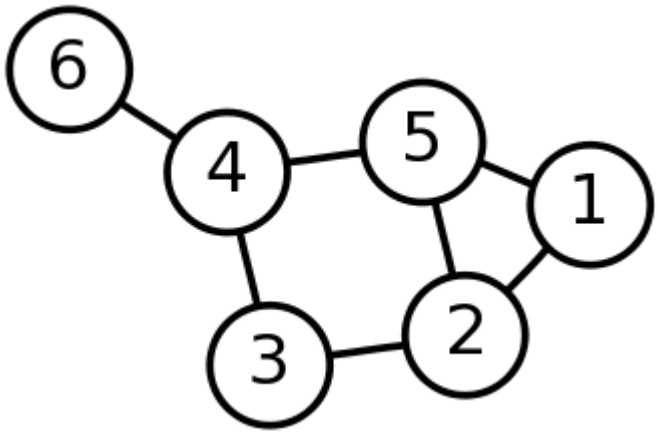


Figure 19 un graphe à 6-sommets

4.1.1.2 Représentation :

Nous utilisons la représentation des listes d adjacence, où nous maintenons un tableau de sommet indexées des listes de sommets reliés par une arête à chaque sommet.

4.1.1.2.1 les listes d'adjacences :

Dans ce type de structure, on a un tableau de liste de sommet. Le tableau comporte autant de cases qu'il y a de sommets. Chacune des cases pointe vers une liste de sommet. Cette liste n'est rien d'autre que les successeurs du sommet considéré.

(**Adjacence** : deux arcs sont adjacents s'ils ont une extrémité commune; deux sommets sont adjacents s'il existe un arc, ou une arête, les reliant)

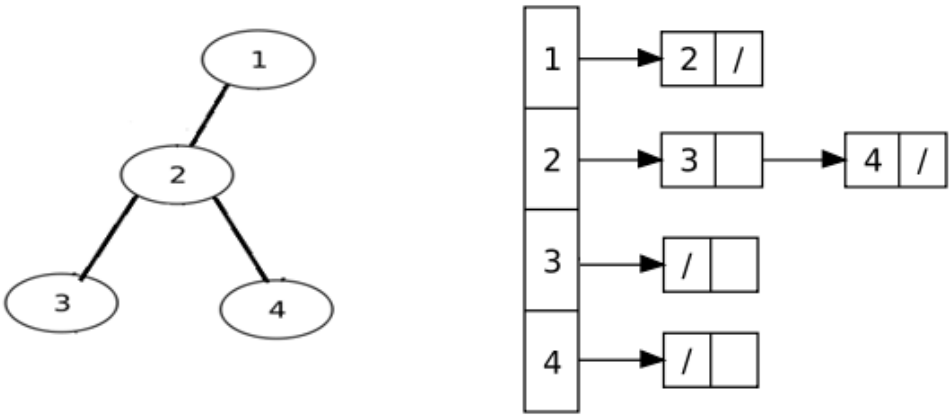


Figure 20 arbre avec sa représentation sous liste d'adjacence

4.1.1.2.2 Raison d'utilisation des listes d'adjacence :

- ✓ Cette représentation utilise un espace mémoire en $\Theta(n+p)$ pour un graphe avec **n** nœuds et **p** arcs
- ✓ Accès optimisé aux successeurs et prédécesseurs d'un nœud : $\Theta\ (d^{+})$ et $\Theta\ (d^{-})$
- ✓ Ajout et suppression de nœuds sans réallocation complète

4.2 Implémentation des algorithme en MATLAB :

4.2.1 L'algorithme ST :

```
function ST=subtree( r1, r2)
%subtree tree kernel function that compares two trees with co-rooted subtree
%       subtree (<r1 r2>) : r1 r2 are the roots of the tow trees
%
%       [<ST>] = subtree(...) the number of chared co-rooted trees

% Code de la fonction

   if size(getchildren(t1, r1))~= size(getchildren(t2, r2)) | size(getchildren(t1,
r1))==0 than
     ST=0;
     return;
   end
     ST=1;
  for i=1 : size(getchildren(obj, r1))

     ST=ST*subset(r1.size(getchildren(obj, r1))(i),r2.size(getchildren(obj,
r2))(i))
  end

end
```

4.2.2 L'algorithme SST :

```
function [dpi1,dpi2]=Subsettree(r1,r2)
%subsettree  kernel function that compares two trees with subtree
%       subsettree (<r1 r2>) : r1 r2 are the roots of the tow trees
%
%       [dpi1,dpi2] = subsettree(...)computes the all-subtree kernel in the array
[dpi1,dpi2]
```

```
% Code de la fonction
  v1=r1.breadthfirstiterator;
  v2=r2.breadthfirstiterator(;
    for i1=v1(1):v1(nnodes(r1))
     for i2=v2(1):v1(nnodes(r2))
        if size(getchildren(r1,i1))==0   ~= size(getchildren(r2, i2)==)0 |
size(getchildren(r1, v1))==0 than
        dpri1(1)=0
        dpri2(1)=0
        else
         dpri1(1)=1
         dpri2(1)=1
        end
         chilnoeud1=getchildren(i1)
         chilnoeud2=getchildren(i2)
         for k=1:size(chilnoeud1)
           dpri1(i1)=dpri1(i1)*dpri1(chilnoeud1(k)+1)
           dpri1(i2)=dpri1(i2)*dpri1(chilnoeud2(k)+1)
         end
     end
   end
  for i1=v1(1):v1(nnodes(r1))
    for i2=v2(1):v1(nnodes(r2))
      if size(getchildren(r1,i1))==0 | size(getchildren(r2, i2)==)0
          dpi1(1)=0
          dpi2(1)=0
      else
           dpi1(i1)=dpri1(i1)
           dpi2(i2)=dpri2(i2)
           for j1=1:size(chilnoeud)
            dpi1(i1)=dpi1(i1)+dpi1(chilnoeud1(j1))
            dpi2(i2)=dpi1(i2)+dpi2(i2)
           end
            for j2=1:size(chilnoeud2)
```

```
                dpi1(i1)=dpi1(i1)+dpi1(i1)
                dpi2(i2)=dpi1(i2)+dpi2(chilnoeud2(j2))
                end
                for j2=1:size(chilnoeud2)
                dpi1(i1)=dpi1(i1)+dpi1(chilnoeud1(j1))
                dpi2(i2)=dpi1(i2)+dpi2(chilnoeud2(j2))
                end
            end
          end
        end
      end
```

Fonction	Description
breadthfirstiterator	Fonction qui retourne un vecteur d'indices qui parcourt l'arbre en largeur
nnodes(obj)	Fonction qui return le nombre de nœud d' un arbre
getchildren(obj,ID)	Retourné la liste de ID du fils d'un nœud donné ID. Comme un vecteur

Tableau 2 description des finction utilis

Conclusion

Les méthodes des noyaux sur les arbres ont été proposés comme une solution alternative aux limitations des méthodes d'apprentissage traditionnelles. Ils ont trouvées plusieurs domaines d'application, à savoir le traitement automatique de langue naturelle, la bioinformatique, ... etc.
Nous avons décrits deux algorithmes concernant les noyaux d'arbres à savoir le noyau d'arbre ST (Subtree kernel) et SST (Subset tree kernel) ainsi que l'analyse de leur complexité.
Par ailleurs nous avons commencé au développement d'une boite à outils comportant l'implémentation des algorithmes suscités. Le développement n'est pas achevé par manque de temps.
Ceci dit, le mémoire nous a été bénéfique. D'une part, nous nous sommes impliqués dans la méthodologie de recherche académique et d'autre part nous avons eu l'occasion de découvrir un domaine qui ne cesse de progresser.

Bibliographie

1. **Martino, Giovanni Da San.** *Kernel Methods for Tree Structured Data.* Université di Bologna, Padova : s.n., 2009.

2. **ASEERVATHAM, Sujeevan.** *Apprentissage à base de Noyaux Sémantiques.* 2007.

3. **abu-mustafa, yaser.** *Kernel Méthode lecture 15.* caltech, 2012.

4. **Vincent, Pascal.** *Modèles à noyaux à structure locale.* 2003.

5. **Bouzy, Bruno.** *Apprentissage par renforcement (1/3).* 2013.

6. *Introduction à l'apprentissage par renforcement.* 2003.

7. **Sewell, Martin.** *Kernel Methods.* University College London : s.n., 2007.

8. http://prateekvjoshi.com/2012/09/01/kernel-functions-for-machine-learning/. *Kernel Functions For Machine Learning.* [En ligne] 2012.

9. **Bucatanschi, Dan George.** *Kernel Methods for Image Processing.* 2006.

10. **Miclet, Antoine Cornuéjols • Laurent.** *Apprentissage artificiel Concepts et algorithmes.* 2003.

11. **Sujeevan Aseervatham, Emmanuel Viennet.** *Méthodes à noyaux appliquées aux textes structurés.* Université de Paris-Nord : s.n.

12. **Shawe-Taylor, John.** *Kernel Methods for Pattern Analysis.* 2004.

13. **Sujeevan Aseervatham, Emmanuel Viennet.** *Méthodes à noyaux appliquées aux textes structurés.* Paris : s.n.

14. **Moschitti, Alessandro.** *Making Tree Kernels practical for Natural Language Learning.* Rome, Italy : s.n.

15. **Martino, Giovanni Da San.** *Route Kernels for Trees.* Padua, Italy : s.n.

16. **Moschitti, Alessandro.** *Efficient Convolution Kernels for Dependency.* Italy : s.n.

17. *"Kernelized" Self-Organizing Maps for Structured Data.* **Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti.** Padova - Italy : s.n.

18. [En ligne] matlab.developpez.com.

19. **Sedjwick, Robert.** [En ligne] http://algs4.cs.princeton.edu/41undirected/.

20. **Cristianini, John Shawe-Taylor Nello.** *Kernel Methods for Pattern Analysis.* cambridge : Cambridge University Press, 2004.

21. **Hofmann, Martin.** *Support Vector Machines — Kernels and the Kernel Trick.* 2006.

22. [En ligne] matlab.developpez.com.

décomposer le calcul de la similarité d'éléments permet de traiter aisément des structures complexes. Toutefois, elle nécessite, en contrepartie, un temps de calcul non négligeable. Ainsi, il est nécessaire de spécialiser ce noyau selon le type de structure afin de réduire la complexité. (2)

3. Les noyaux sur les arbres :

Comme on a déjà dite que les noyaux d'arbres sont prolongés dans plusieurs domaines grâce a leur forme hiérarchique qui nous permet de donner une bonne représentation des données.

Comme les noyaux d'arbre ont beaucoup d'intérêts ils ont quelques inconvénients parmi leur majeurs inconvénients est le temps d'exécution des algorithmes qui les traites il ont une complexité quadratique dépend sur le nombre des nœuds de l'arbre pour cela et pour rendre les noyaux d'arbres plus performantes deux algorithmes ont été proposé le 1^{er} « subtree »(ST) de Vishwanathan and Smola et le $2^{éme}$ « subset tree »(SST) de Collins and Duffy on va mieux expliquer et détaillé sur le fonctionnement récursif de ces deux algorithme selon (12)

Tous les noyaux présentés dans cette section peuvent être définies par un espace d'intégration explicite à partie l'espace de tous les arbres finis peut être étiquetés avec un ensemble A à un espace vectoriel F, dont les coordonnées sont indexées par un sous-ensemble I des arbres aussi peut être étiquetés ou non . Comme d'habitude, nous utilisons φ pour désigner l'espace de redescription

$$\varphi: T \rightarrow (\varphi_s(T))_{s \in I} \in F$$

L'objectif est de divisé l'espace de redescription pour lequel le noyau correspondant peut être évaluée à l'aide d'un calcul récursif qui procède de bas en haut à partir des feuilles à la racine des arbres. La relation de récurrence de base relie les valeurs de certaines fonctions d'un nœud donné avec les valeurs de fonction à ses fils. La base de la récursivité est définir les valeurs dans les feuilles, faire en sorte que le calcul est bien définie

À titre d'exemple pour comprendre le fonctionnement récursif voilà un exemple de comptage de sous arbre

Pour le calcule récursif sur un arbre considèrent l'évaluation de nombre N(T) de sous-arbres Co-enracines d'un arbre T.