

Méthodes à noyaux appliquées aux textes structurés

Sujeevan Aseervatham, Emmanuel Viennet

Université de Paris-Nord, LIPN - UMR CNRS 7030
99, avenue Jean-Baptiste Clément
93430 Villetaneuse, France
{Prénom.Nom}@lipn.univ-paris13.fr

Résumé. Cet article ébauche un état de l'art sur l'utilisation des noyaux pour le traitement des données structurées. Les applications modernes de la fouille de données sont de plus en plus confrontés à des données structurées, notamment textuelles. Les algorithmes d'apprentissage doivent donc être capables de tirer parti des informations apportées par la structure, ce qui pose d'intéressants problèmes de représentation des données. L'une des approches possibles consiste à utiliser les noyaux de Mercer. Ces noyaux permettent de calculer la similarité entre deux données de type quelconque, et peuvent être utilisés par une large gamme d'algorithmes d'apprentissage (Machines à Vecteur de Support, ACP, Analyse Discriminante, Perceptron, etc). Nous présentons dans cet article les principaux noyaux proposés ces dernières années pour le traitement des structures telles que les séquences, les arbres et les graphes.

1 Introduction

Les techniques d'apprentissage statistique sont généralement conçues pour travailler sur des données vectorielles ; chaque mesure est représentée par un ensemble de données numériques de taille fixe. Pendant plusieurs décennies, les recherches en statistique se sont centrées sur des problèmes comme la normalisation des données, le traitement des valeurs manquantes, etc. Depuis une dizaine d'années, sous la pression des applications, nous sommes confrontés à des problèmes dans lesquels la structure des données porte une information essentielle : textes en langage naturel, documents XML, séquences biologiques, analyse de scènes (images), analyse des réseaux sociaux. Pour attaquer ces problèmes, il est nécessaire trouver un moyen de traiter l'information structurelle, par exemple en calculant une mesure de similarité entre deux structures.

De nombreux systèmes d'apprentissage numérique des données textuelles utilisent une représentation du texte en "sac de mot". Ce type de codage, qui a l'avantage de la simplicité, n'utilise que les fréquences d'apparition des mots dans les documents et perd toute information liée à l'ordre des éléments (ordre des mots, structure en paragraphes ou sections, etc).

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 vecteur de support (SVM) (Vapnik, 1995), 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. Or, de nombreuses méthodes d'apprentissage statistique peuvent se formuler en ne recourant qu'à des calculs de produits scalaires entre les éléments (exemples d'apprentissage ou nouveaux points). L'utilisation d'une fonction noyau permet alors d'utiliser ces méthodes bien connues et performantes (analyse discriminante, analyse en composantes principales, perceptron, etc) avec un traitement intégré qui peut être non linéaire et incorporer des connaissances sur l'application (par exemple sur la structure des données). Récemment, l'utilisation de noyaux spécifiques pour le traitement de données textuelles structurées a commencé à faire l'objet de recherches.

Dans le cadre du projet InfoM@gic (pôle IMVN Cap Digital), nous travaillons sur l'application des méthodes à noyaux au traitement de données textuelles structurées, et cet article présente un bref état de l'art dans ce domaine. Nous passons en revue les principaux types de noyaux proposés ces dernières années pour le traitement des séquences et plus généralement des données structurées (arbres, graphes, etc).

2 Le noyau de convolution

Le noyau de convolution appelé R-noyau (Haussler, 1999) permet de définir un cadre général pour les noyaux appliqués aux données structurées telles que les arbres et les graphes. Les données structurées sont des objets pouvant être décomposés en sous-objets jusqu'à atteindre une unité atomique. L'idée du R-noyau est de calculer la similarité entre deux éléments x et y en effectuant une décomposition de x et de y . Plus formellement, soit x un élément appartenant à un ensemble \mathcal{X} d'éléments structurés de même type, x peut être décomposé en sous éléments (x_1, \dots, x_D) où x_d peut être un élément structuré ou non appartenant à \mathcal{X}_d . On définit la relation binaire

$$R : \mathcal{X}_1 \times \dots \times \mathcal{X}_D \rightarrow \mathcal{X}$$

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

$$R^{-1} : \{\bar{x} = (x_1, \dots, x_D) \mid R(\bar{x}, x)\}$$

qui retourne pour x l'ensemble de toutes les décompositions possibles. Le noyau de convolution (R-noyau) pour deux éléments x et y de \mathcal{X} est alors :

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

avec k_i un noyau calculant la similarité entre les éléments x_i et y_i de même structure i.e. $x_i, y_i \in \mathcal{X}_i$. Il est facile de montrer que si les k_i sont des noyaux valides alors k_R 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 ($x_i, x \in \mathcal{X}$). 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 contre-partie, un temps de calcul non

négligeable. Ainsi, il est nécessaire de spécialiser ce noyau selon le type de structure afin de réduire la complexité.

3 Les noyaux pour les séquences de caractères

Les séquences de caractères sont considérées comme faisant partie des données structurées car d'une part une séquence peut être décomposée en sous-partie ainsi les séquences possèdent la propriété des données structurées vue dans le paragraphe précédent et d'autre part les caractères de la séquence sont ordonnés.

Les séquences sont généralement rencontrées dans les domaines liés à la bioinformatique mais aussi dans les documents en langage naturel. En effet, on peut considérer tout un document comme étant une séquence. Ainsi, deux documents peuvent être considérés comme proches s'ils partagent un nombre important de sous-séquences identiques. Cette approche permet alors de tenir compte des mots composés comme par exemple "économie" et "microéconomie" qui ne peuvent être traités par l'approche en sac de mots. 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.

Bien que cette approche donne de meilleurs résultats que l'approche classique en sac de mots, elle n'en reste pas moins coûteuse en temps de calcul.

3.1 Le noyau p -Spectrum

Le noyau p -spectrum (ou n -gram) (Leslie et al., 2002; Lodhi et al., 2002) est le noyau le plus simple pour le traitement de séquences. Il permet d'évaluer le nombre de sous-séquences contiguës de taille p (ou n) que deux documents ont en communs. Plus le nombre de sous-séquences en commun est important et plus la similarité des deux documents sera importante. Soit l'alphabet Σ , l'espace associé au noyau p -spectrum sera de dimension $\text{card}(\Sigma)^p$. Le $u^{\text{ème}}$ composant du vecteur $\Phi^p(s)$ associé à la séquence s avec $u \in \Sigma^p$ est :

$$\Phi_u^p(s) = \text{card}(\{(v_1, v_2) | s = v_1 u v_2\})$$

Avec $v_1 u v_2$ désignant la concaténation des séquences v_1 , u et v_2 .

Le noyau p -spectrum est alors :

$$k_p(s_1, s_2) = \langle \Phi^p(s_1), \Phi^p(s_2) \rangle$$

La complexité de ce noyau est $O(p|s_1||s_2|)$. Toutefois, il est possible de réduire cette complexité à $O(p \times \max(|s_1|, |s_2|))$ en utilisant la programmation dynamique et des structures de données appropriées comme les arbres de suffixes (Shawe-Taylor et Cristianini, 2004).

Le noyau p -Spectrum a été utilisé par Leslie et al. (2002) pour la classification de séquences de protéines avec l'algorithme SVM (Séparateur à Vaste Marge). Les résultats obtenus sur la base de données SCOP (*Structural Classification of Proteins*) ont montré que la classification par SVM avec le noyau p -Spectrum donne des résultats semblables aux méthodes génératives basées sur les modèles de Markov cachés. Cependant, la méthode SVM avec le noyau Fisher reste la plus performante pour la classification de séquences de protéines.

3.2 Le noyau *All-SubSequences*

Le noyau *All-SubSequences* (Shawe-Taylor et Cristianini, 2004) permet de tenir compte des sous-séquences non contiguës de toutes tailles. Ainsi, la fonction $\Phi^A(s)$ permet de plonger la séquence s dans un espace vectoriel dans lequel le $u^{\text{ème}}$ composant de $\Phi^A(s)$ indique la fréquence d'occurrence de la séquence u dans s . On dira que u est une sous-séquence non contiguë de s , s'il existe un ensemble $I = \{i_1, \dots, i_{|u|}\}$ tel que $\forall j \in \{2, \dots, |u|\}, i_{j-1} < i_j$ et $\forall j \in \{1, \dots, |u|\}, u(j) = s(i_j)$ ($u(j)$ désignant le $j^{\text{ème}}$ élément de u). On notera $s[I] = u$ pour désigner le fait que chaque élément de $u(j)$ est identique à l'élément $s(i_j)$ avec $i_j \in I$.

$$\Phi_u^A(s) = \sum_{I:s[I]=u} 1$$

D'où :

$$k_A(s, t) = \sum_{(I_1, I_2):s[I_1]=t[I_2]} 1$$

Il est possible de définir ce noyau de manière récursive. En effet, il suffit de remarquer que toute sous-séquence de s contenant le dernier caractère a de s , tel que $s = s'a$, ne peut apparaître dans t qu'entre le premier caractère de t et la dernière occurrence de a dans t . Ainsi, on a :

$$\begin{aligned} k_A(s, \epsilon) &= 1 \\ k_A(s'a, t) &= k_A(s', t) + \sum_{k:t[k]=a} k_A(s', t[1 \dots k-1]) \end{aligned}$$

L'avantage de ce noyau est qu'il est capable de capturer tous les motifs communs à deux séquences. Le désavantage est que l'espace de projection est de très haute dimension entraînant un temps de calcul important.

3.3 Le noyau *p-Fixed length SubSequence*

Le noyau *p-Fixed length SubSequence* (Shawe-Taylor et Cristianini, 2004) est un compromis entre le noyau *p-Spectrum* et le noyau *All-SubSequences*. Il permet de limiter la recherche de sous-séquences à des sous-séquences non contiguës de taille p . Ainsi, la fonction de projection $\Phi^F(s)$ sera composée des éléments $\Phi_u^A(s)$ tels que $|u| = p$.

De même que précédemment, on pourra définir ce noyau par récurrence en notant que la récurrence sera définie sur la séquence, en retirant à chaque étape le dernier élément de la séquence, mais aussi sur la taille p du motif. En effet, si le dernier caractère du motif a été fixé, le préfixe du motif ne peut être constitué que de $p - 1$ éléments.

$$\begin{aligned} k_0(s, t) &= 1 \\ k_p(s, \epsilon) &= 0 \text{ pour } p > 0 \\ k_p(s'a, t) &= k_p(s', t) + \sum_{k:t[k]=a} k_{p-1}(s', t[1 \dots k-1]) \end{aligned}$$

3.4 Le noyau *String Subsequence* (SSK)

L'un des inconvénients des noyaux traitant les sous-séquences non contiguës vus précédemment est qu'ils ne tiennent pas compte de la distance séparant les éléments non contiguës. En effet, prenons l'exemple de deux séquences "aaab" et "aab", la séquence "ab" est une sous-séquence des deux premières mais elle est plus similaire à la deuxième qu'à la première. Or, les noyaux *All-SubSequences* et *p-Fixed length SubSequence* attribueront la même valeur aux couples ("aaab", "ab") et ("aab", "ab").

Le noyau *String Subsequence* (Lodhi et al., 2002) permet de tenir compte de la discontinuité dans le calcul de la similarité en pondérant les séquences en fonction de leur taille. Pour une séquence s , la fonction de projection $\Phi_u^{SSK}(s)$ sera défini pour tout $u \in \Sigma^n$ par :

$$\Phi_u^{SSK}(s) = \sum_{I:u=s[I]} \lambda^{\text{card}(I)}$$

Le noyau SSK, de paramètre n , pour deux séquences s et t est alors :

$$k_{SSK}^n(s, t) = \sum_{u \in \Sigma^n} \sum_{I:u=s[I]} \sum_{J:u=t[J]} \lambda^{\text{card}(I)+\text{card}(J)}$$

Comme pour les noyaux précédents, en utilisant la programmation dynamique, on peut réduire la complexité à $O(n \cdot |s| \cdot |t|)$.

Des expérimentations ont été menées dans (Lodhi et al., 2002) pour évaluer les noyaux SSK, *p-Spectrum* et le noyau standard *Bag Of Words* (Joachims, 2002). La base de données utilisée est la base *Reuters-21578* contenant des documents en langage naturel. L'expérience consistait à effectuer un classement binaire des documents après avoir effectué un apprentissage sur les données prévues à cet effet. Les documents ont été pré-traités en éliminant les mots d'arrêts et les signes de ponctuations. Les résultats ont montrés que les *string kernels* sont plus performants que l'approche standard du *Bag Of Words*. De plus, le noyau SSK est le plus performant lorsque la valeur de pondération est choisie judicieusement. De même, lorsque la taille p est choisie convenablement, le noyau *p-Spectrum* donne les meilleurs résultats. Les deux inconvénients pour l'utilisation de ces noyaux sont d'une part le temps de calcul et d'autre part le choix des paramètres qui doit être fait de manière spécifique à chaque application.

3.5 Le noyau Séquence marginalisée

Le noyau marginalisé pour les séquences a été introduit par Kashima et Tsuboi (2004) afin d'étiqueter des structures complexes tels que les séquences, les arbres et les graphes. Le problème de l'étiquetage consiste à affecter à une donnée $\mathbf{x} = (x_1, \dots, x_T)$ un groupe d'étiquettes $\mathbf{y} = (y_1, \dots, y_T) \in \Sigma_y^T$. Par exemple, en langage naturel \mathbf{x} peut représenter une phrase, et le problème consiste à attribuer à chaque mot de \mathbf{x} une étiquette désignant son groupe grammatical (*Part Of Speech tagging*). La figure 1(a) montre un exemple de couple (x, y) où les nœuds noirs représentent les éléments x_i et les nœuds blancs les étiquettes y_i associées à x_i .

Pour résoudre le problème de l'étiquetage, l'approche standard consiste à attribuer à \mathbf{x} la séquence d'étiquettes \mathbf{y} tel que : $\mathbf{y} = \operatorname{argmax}_{\mathbf{y}} \sum_i \log P(y_i | x_i)$. Il s'agit ici de maximiser la probabilité que la séquence entière soit correcte. Une autre approche consiste à maximiser individuellement la probabilité qu'une étiquette y_i corresponde à l'élément x_i . Soit :

$$y_i = \operatorname{argmax}_y P(y_i = y | x_i) = \operatorname{argmax}_y \sum_{\mathbf{y}: y_i=y} P(\mathbf{y} | \mathbf{x})$$

Partant de cette dernière approche, Kashima et Tsuboi (2004) proposent une méthode pour étiqueter une séquence. Cette méthode se base sur l'utilisation d'un perceptron à noyau (Shawe-Taylor et Cristianini, 2004).

Afin d'étiqueter les éléments individuellement, une séquence (\mathbf{x}, \mathbf{y}) est décomposée en triplets (\mathbf{x}, t, y_t) où y_t est le $t^{\text{ème}}$ élément de \mathbf{x} . Pour cela, le perceptron à noyau est entraîné avec des séquences étiquetées auxquelles ont été ajoutés des exemples négatifs. Ces exemples négatifs sont obtenus pour chaque couple (\mathbf{x}, \mathbf{y}) en modifiant les valeurs de \mathbf{y} . Ainsi pour ce couple on obtient un ensemble de positifs $\{(\mathbf{x}, i, y_i) : 1 \leq i \leq \dim(\mathbf{x})\}$ et un ensemble de négatifs $\{(\mathbf{x}, i, z) : 1 \leq i \leq \dim(\mathbf{x}) \forall z \in \Sigma_y\}$. Après apprentissage, on affecte à un élément u_t d'une séquence \mathbf{u} l'étiquette y qui maximise le score calculé par le perceptron avec un noyau marginalisé.

Le noyau marginalisé proposé est le suivant :

$$k(\mathbf{x}, \mathbf{x}', \tau, t, y_\tau, y'_t) = \sum_{\mathbf{z}: z_\tau=y_\tau} \sum_{\mathbf{z}': z'_t=y'_t} P(\mathbf{z} | \mathbf{x}) P(\mathbf{z}' | \mathbf{x}') \langle \Phi(\mathbf{x}, \mathbf{z}; \tau), \Phi(\mathbf{x}', \mathbf{z}'; t) \rangle \quad (1)$$

y_τ et y'_t étant les étiquettes respectives des éléments x_τ et x'_t ; $\phi_f(\mathbf{x}, \mathbf{y}; t)$ indiquant la fréquence d'occurrence de f dans (\mathbf{x}, \mathbf{y}) incluant la $t^{\text{ème}}$ position.

La combinatoire induite par le noyau de l'équation 1 peut-être diminuée en effectuant une recherche bidirectionnelle au sein d'une séquence. En effet, la composante $\phi_f(\mathbf{x}, \mathbf{y}; t)$ du vecteur $\Phi(\mathbf{x}, \mathbf{y}; t)$ indique le nombre d'occurrence de f dans (\mathbf{x}, \mathbf{y}) incluant la $t^{\text{ème}}$ position de (\mathbf{x}, \mathbf{y}) $((x_t, y_t))$. Il est alors possible de décomposer f en f_u et f_d tel que $((x_t, y_t))$ soit le dernier élément de f_u et le premier élément de f_d . Le calcul se limitera alors à évaluer les deux ensembles $F_u = \{(x_i, x_{i+1}, \dots, x_t) | 1 \leq i \leq t\}$ et $F_d = \{(x_t, x_{t+1}, \dots, x_k) | t \leq k \leq \dim(\mathbf{x})\}$. Les autres composantes de l'espace seront obtenues par combinaison de F_u et de F_d i.e. $F = F_u \times F_d$.

La figure 1 illustre ce principe : le couple de séquences (a) peut être obtenu en combinant (b) et (c).

Afin d'effectuer la décomposition d'une séquence en fonction de la position t , on supposera que :

$$P(\mathbf{y} | \mathbf{x}) = \prod_t P(y_t | x_t)$$

On posant $\mathbf{x}_u(t) = (x_0, \dots, x_t)$ et $\mathbf{x}_d(t) = (x_t, \dots, x_T)$, de même pour $\mathbf{y}_u(t)$ et $\mathbf{y}_d(t)$, on obtient :

$$P(\mathbf{y} | \mathbf{x}) = P(\mathbf{y}_u(t) | \mathbf{x}_u(t)) \cdot \frac{P(y_t | x_t)}{(P(y_t | x_t))^2} \cdot P(\mathbf{y}_d(t) | \mathbf{x}_d(t))$$

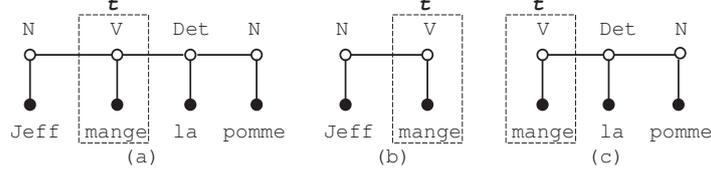


FIG. 1 – (a) Exemple de couple de séquences (x, y) . (b) sous-séquences de (a) où les x_i avec $i > t$ ont été éliminés. (c) sous-séquences de (a) obtenues en éliminant les x_i de (a) pour $i < t$.

En décomposant, l'équation 1 on obtient :

$$k(\mathbf{x}, \mathbf{x}', \tau, t, y_\tau, y'_t) = k_u(\mathbf{x}, \mathbf{x}', \tau, t) \cdot k_p(\mathbf{x}, \mathbf{x}', \tau, t, y_\tau, y'_t) \cdot k_d(\mathbf{x}, \mathbf{x}', \tau, t)$$

avec :

$$\begin{aligned} k_u(\mathbf{x}, \mathbf{x}', \tau, t) &= \sum_{\mathbf{y}_u(\tau)} \sum_{\mathbf{y}'_u(t)} P(\mathbf{y}_u(\tau) | \mathbf{x}_u(\tau)) P(\mathbf{y}'_u(t) | \mathbf{x}'_u(t)) \\ &\quad \cdot \langle \Phi(\mathbf{x}_u(\tau), \mathbf{y}_u(\tau); \tau), \Phi(\mathbf{x}'_u(t), \mathbf{y}'_u(t); t) \rangle \\ k_d(\mathbf{x}, \mathbf{x}', \tau, t) &= \sum_{\mathbf{y}_d(\tau)} \sum_{\mathbf{y}'_d(t)} P(\mathbf{y}_d(\tau) | \mathbf{x}_d(\tau)) P(\mathbf{y}'_d(t) | \mathbf{x}'_d(t)) \\ &\quad \cdot \langle \Phi(\mathbf{x}_d(\tau), \mathbf{y}_d(\tau); \tau), \Phi(\mathbf{x}'_d(t), \mathbf{y}'_d(t); t) \rangle \\ k_p(\mathbf{x}, \mathbf{x}', \tau, t, y_\tau, y'_t) &= \frac{P(y_\tau | x_\tau) P(y'_t | x'_t) \cdot \langle \Phi(x_\tau, y_\tau; \tau), \Phi(x'_t, y'_t; t) \rangle}{(\sum_{z_\tau} \sum_{z'_t} P(z_\tau | x_\tau) P(z'_t | x'_t) \cdot \langle \Phi(x_\tau, z_\tau; \tau), \Phi(x'_t, z'_t; t) \rangle)^2} \quad (2) \end{aligned}$$

La complexité pour l'évaluation de ces noyaux peut être ramenée à $O(T.T')$ avec T et T' la taille des séquences en utilisant la programmation dynamique. On obtient alors les noyaux suivants :

$$\begin{aligned} k_u(\mathbf{x}, \mathbf{x}', \tau, t) &= \begin{cases} 0 & \text{si } \tau = 0 \text{ ou } t = 0 \\ c^2 k(x_\tau, x'_t) (k_u(\mathbf{x}, \mathbf{x}', \tau - 1, t - 1) + 1) \end{cases} \\ k_d(\mathbf{x}, \mathbf{x}', \tau, t) &= \begin{cases} 0 & \text{si } \tau > \dim(\mathbf{x}) \text{ ou } t > \dim(\mathbf{x}') \\ c^2 k(x_\tau, x'_t) (k_d(\mathbf{x}, \mathbf{x}', \tau + 1, t + 1) + 1) \end{cases} \\ k(x_\tau, x'_t) &= \begin{cases} 0 & \text{si } x_\tau \neq x'_t \\ \sum_y P(y | x_\tau) P(y | x'_t) \end{cases} \\ k_p(\mathbf{x}, \mathbf{x}', \tau, t, y_\tau, y'_t) &= \begin{cases} 0 & \text{si } (x_\tau, y_\tau) \neq (x'_t, y'_t) \\ \frac{c P(y_\tau | x_\tau) P(y'_t | x'_t)}{(c^2 k(x_\tau, x'_t))^2} \end{cases} \end{aligned}$$

La constante c est utilisée pour pondérer les termes ϕ_f selon la taille de f . Il est aussi possible de permettre des discontinuités dans les sous-séquences comme dans le cas du noyau *String Subsequence*. Il suffit alors simplement de modifier k_u et k_d (voir (Kashima et Tsuboi, 2004) pour plus de détails).

Méthodes à noyaux appliquées aux textes structurés

Ce noyau, combiné au noyau polynomial de degré deux, a été utilisé avec un perceptron pour résoudre un problème de reconnaissance d'entités nommées et un problème d'extraction d'information. La loi uniforme est utilisée pour modéliser $P(y_t|x_t)$. Les expériences ont été menées en utilisant la validation croisée à 3 blocs.

En outre, pour chaque expérience le noyau marginalisé est comparé à un perceptron utilisant le modèle de Markov caché (Collins, 2002). L'idée de base de cet algorithme est d'utiliser l'algorithme de Viterbi sur un modèle de Markov caché afin d'attribuer la meilleure séquence d'étiquettes à une séquence de termes. Le perceptron est utilisé pour attribuer un score à une séquence étiquetée. Ce score sera utilisé par l'algorithme de Viterbi pour trouver la séquence d'étiquettes optimale.

Pour la reconnaissance d'entités nommées, les données utilisées sont un sous-ensemble d'un corpus espagnol fourni par CoNLL2002. Ce corpus est composé de 300 phrases comprenant au total 8541 termes. L'objectif est d'attribuer à chaque terme l'un des neuf labels désignant le type d'entité nommée (l'un des neuf labels correspond au type "non-entité nommée"). Les résultats montrent que le noyau marginalisé a un taux de reconnaissance, tant au niveau de la précision que du rappel, supérieur à celui du modèle de Markov caché.

La deuxième expérience consistait à extraire des informations concernant l'utilisation de produits. A partir d'une base de 184 phrases (soit 3570 termes) en japonais, l'objectif est de reconnaître le nom du produit, le vendeur, le nombre de produits achetés, les raisons de l'achat etc. Ainsi, il s'agit d'attribuer à chaque terme l'une des 12 étiquettes correspondants aux informations citées.

Les termes ont été annotés en effectuant une analyse lexicale. En outre, un noyau marginalisé sur les arbres (voir la section 4) a été utilisé. Pour ce noyau, les données ont été structurées en arbre lexical de dépendance représentant la structure linguistique de la phrase en terme de dépendance entre les mots.

Les expériences montrent que les noyaux marginalisés sont plus performants que le perceptron utilisant le modèle de Markov caché. De plus, le noyau marginalisé sur les arbres obtient de meilleurs résultats que le noyau sur les séquences. Ce résultat peut être expliqué par le fait que le noyau sur les arbres tire avantage de l'information structurelle contrairement au noyau sur les séquences.

4 Les noyaux pour 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.

4.1 Le noyau *Tree kernel*

Le noyau *Tree Kernel* (appelé aussi *parse tree kernel*) (Collins et Duffy, 2002) a été défini pour le calcul de similarité entre les arbres grammaticaux (ou arbres syntaxiques). Un arbre syntaxique est obtenu à partir d'une phrase en la décomposant en groupe grammatical. La figure 2 montre l'arbre syntaxique associé à la phrase "Jeff mange la pomme".

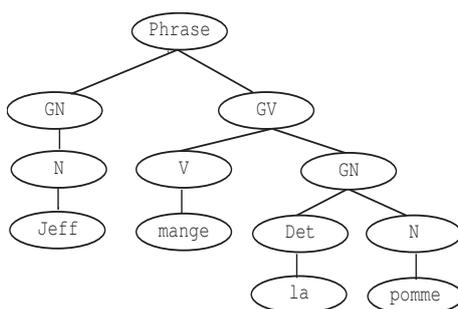


FIG. 2 – Arbre syntaxique pour la phrase "Jeff mange la pomme"

Definition 1 (arbre propre) Un arbre propre est un arbre ayant une racine et au moins un nœud fils.

Definition 2 (Sous-arbre complet) Un arbre S est dit sous-arbre complet d'un arbre T si et seulement si il existe un nœud n tel que l'arbre induit par n (de racine n) est égal à S . On notera $\tau_T(n)$ l'arbre complet induit par le nœud n de T .

Definition 3 (Sous-arbre co-enraciné) Un arbre S est dit sous-arbre co-enraciné d'un arbre propre T si et seulement si S les propriétés suivantes sont vérifiées :

1. S est un arbre propre,
2. la racine de S ($rac(S)$) est identique à la racine de T ($rac(T)$) (si S et T sont des arbres dont les nœuds sont étiquetés alors les étiquettes de $rac(S)$ et de $rac(T)$ doivent être identiques),
3. $\forall i, fils_i(rac(S)) = fils_i(rac(T))$,
4. S peut être obtenu à partir de T en supprimant des sous-arbres de $fils_i(rac(T))$.

La notion de sous-arbre co-enraciné permet de garantir la consistance des règles grammaticales. Par exemple, pour l'arbre T de la figure 2, il existe des arbres co-enracinés de T qui produisent : "N V la N", "Jeff mange GN", "GN V GN", ... Toutefois, il n'existe pas d'arbres co-enracinés de T produisant "Jeff mange Det", "GN mange N", ...

Soit Γ l'ensemble de tous les arbres propres possibles, un arbre T peut être plongé, par une fonction Φ^r , dans un espace vectoriel de caractéristique (*feature-space*). Le $u^{\text{ème}}$ composant

Méthodes à noyaux appliquées aux textes structurés

de Φ^r , associé à $S_u \in \Gamma$, donne le nombre de nœud n de T tel que S_u est un sous-arbre co-enraciné de $\tau_T(n)$ ($Rfreq_T(S_u)$), soit :

$$\Phi_{S_u}^r(T) = Rfreq_T(S_u) = \sum_{n \in T} I_{S_u}(\tau_T(n)) \quad (3)$$

Avec $I_{S_u}(\tau_T(n)) = 1$ si l'arbre S_u est un sous-arbre co-enraciné de $\tau_T(n)$ et 0 sinon. Le noyau permettant de calculer la similarité entre deux arbres T_1 et T_2 est :

$$\begin{aligned} k_{tree}(T_1, T_2) &= \langle \Phi^r(T_1), \Phi^r(T_2) \rangle \\ &= \sum_{S_u \in \Gamma} \Phi_{S_u}^r(T_1) \cdot \Phi_{S_u}^r(T_2) \\ &= \sum_{S_u \in \Gamma} \left(\sum_{n_1 \in T_1} I_{S_u}(\tau_{T_1}(n_1)) \right) \cdot \left(\sum_{n_2 \in T_2} I_{S_u}(\tau_{T_2}(n_2)) \right) \\ &= \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} \sum_{S_u \in \Gamma} I_{S_u}(\tau_{T_1}(n_1)) \cdot I_{S_u}(\tau_{T_2}(n_2)) \\ &= \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} k_{tree}^r(\tau_{T_1}(n_1), \tau_{T_2}(n_2)) \end{aligned} \quad (4)$$

$k_{tree}^r(T_1, T_2)$ indique le nombre de sous-arbres co-enracinés qu'ont en commun les arbres T_1 et T_2 . Cette fonction retourne 0 si 1) les racines sont différentes, ou 2) si le nombre de fils de T_1 et de T_2 ne correspondent pas ou 3) si $\exists i, fils_i(T_1) \neq fils_i(T_2)$. Dans les autres cas, on peut définir $k_{tree}^r(T_1, T_2)$ par récurrence. En effet, cette fonction sera égale au produit des nombres de sous-arbres co-enracinés communs à chacun des fils de T_1 et de T_2 . Il est à noter que si $k_{tree}^r(T_1, T_2) \neq 0$ mais que $k_{tree}^r(\tau_{T_1}(fils_i(rac(T_1))), \tau_{T_2}(fils_i(rac(T_2)))) = 0$, il existe un unique sous-arbre co-enraciné commun à T_1 et T_2 pour la partie du sous-arbre induit par $fils_i$. On peut ainsi définir $k_{tree}^r(T_1, T_2)$ dans le cas non nul :

$$k_{tree}^r(T_1, T_2) = \prod_i (k_{tree}^r(\tau_{T_1}(fils_i(rac(T_1))), \tau_{T_2}(fils_i(rac(T_2)))) + 1)$$

La complexité temporelle du noyau $k_{tree}(T_1, T_2)$ est $O(|T_1||T_2|)$ (Collins et Duffy, 2002) avec $|T|$ le nombre de nœuds dans T .

Collins et al. ont utilisé ce noyau pour associer à une phrase l'arbre syntaxique le plus plausible (*parsing*) (Collins et Duffy, 2002). La décomposition d'une phrase en arbre syntaxique est un problème difficile. En effet, l'ambiguïté sous-jacente au langage naturel entraîne plusieurs décompositions possibles pour une même phrase. L'objectif proposé par Collins et al. est de sélectionner l'arbre le plus probable par une approche discriminante.

Soit F une fonction permettant de générer un ensemble d'arbres syntaxiques pour une phrase, les données sont représentés par un couple $(s, F(s))$. Pour l'ensemble d'apprentissage, l'arbre syntaxique correct pour chaque s est connu dans $F(s)$. Un séparateur est alors "appris" en utilisant un perceptron. Pour une phrase s , on lui associe l'arbre T_{s_i} de $F(s)$ tel que :

$$T_{s_i} = \operatorname{argmax}_{T \in F(s)} (w^* \cdot \Phi^r(T))$$

Avec w^* le vecteur optimal associé à :

$$w = \sum_{s,j>1} \alpha_{s,j} (\Phi^r(T_{s_i}) - \Phi^r(T_{s_j}))$$

Avec s_i une phrase d'apprentissage, $s_1 \in F(s)$ l'arbre syntaxique correct de s et $s_j \in F(s)$. Les expériences sur le corpus *Penn treebank ATIS*, qui est un corpus anglais annoté sous forme arborescente, ont montré que l'utilisation de cette méthode améliore de près de 4% les résultats obtenus par une méthode conventionnelle stochastique (*Probabilistic Context Free Grammar*).

4.2 Le noyau *Tree kernel* généralisé

Le noyau *Tree Kernel* a été essentiellement développé pour traiter des arbres spécifiques telles que les arbres syntaxiques ; il se base sur les propriétés suivantes :

1. les descendants d'un nœud n'ont jamais les mêmes étiquettes que les nœuds ancêtres
2. le noyau utilise la notion de sous-arbre co-enraciné pour calculer la similitude entre deux arbres.

Une généralisation de ce noyau a été proposée dans (Kashima et Koyanagi, 2002) pour traiter des arbres complexes tels que les arbres XML et HTML. Toutefois, on impose que l'arbre soit étiqueté et ordonné (tel que c'était le cas pour les arbres syntaxiques). Le cadre général du noyau *Tree Kernel* reste valide. En effet, pour généraliser le noyau, il suffit de modifier la fonction $I_S(T)$ et de changer le noyau spécifique $k_{tree}^r(T_1, T_2)$. Dans le cas d'un arbre quelconque étiqueté et ordonné T , $I_S(T)$ retournera la fréquence d'occurrence du sous-arbre S dans T .

Definition 4 (Sous-arbre) *Un arbre S (possédant au moins un nœud) est un sous-arbre de T si et seulement si il existe un nœud n de T tel que $l(n) = l(rac(S))$ ($l(n)$ correspondant à l'étiquette du nœud n) et une liste ordonnée d'index $\{j_1, \dots, j_k\}$ tel que $\forall i, l(fil_s_i(rac(S))) = l(fil_{s_{j_i}}(\tau_T(n)))$ avec $i < j_i$ et $\tau_S(fil_{s_i}(rac(S)))$ soit, soit une feuille soit un sous-arbre de $\tau_T(fil_{s_{j_i}}(\tau_T(n)))$ partageant la même racine.*

Étant donnée cette définition de sous-arbre, le noyau $k_{tree}^r(T_1, T_2)$ peut être défini comme étant la fonction qui retourne le nombre de sous-arbres commun à T_1 et T_2 avec pour racine $rac(T_1)$, en tenant compte de la fréquence d'occurrence dans chaque arbre. Autrement dit, il s'agit de la somme, pour chaque sous-arbre possible S de racine $rac(T_1)$, des produits des nombres d'occurrences de S dans T_1 et dans T_2 .

Les arbres étant ordonnés, le noyau sur T_1 et T_2 peut être calculé en introduisant une récurrence sur le nombre de fils de T_1 ($nf(rac(T_1))$) et le nombre de fils de T_2 . Ainsi, la fonction $S_{T_1, T_2}(i, j)$ est introduite pour calculer $k_{tree}^r(T_{1i}, T_{2j})$ tel que T_{1i} est le sous-arbre de T_1 obtenu en supprimant tous les fils d'index supérieurs à i , ainsi que leurs descendants, de même pour T_{2j} .

$$k_{tree}^r(T_1, T_2) = \begin{cases} 0 & \text{si } l(rac(T_1)) \neq l(rac(T_2)) \\ S_{T_1, T_2}(nf(rac(T_1)), nf(rac(T_2))) & \end{cases} \quad (5)$$

Méthodes à noyaux appliquées aux textes structurés

avec

$$\begin{aligned}
 S_{T_1, T_2}(0, 0) &= S_{T_1, T_2}(i, 0) = S_{T_1, T_2}(0, j) = 1 \\
 S_{T_1, T_2}(i, j) &= S_{T_1, T_2}(i-1, j) + S_{T_1, T_2}(i, j-1) \\
 &\quad - S_{T_1, T_2}(i-1, j-1) \\
 &\quad + S_{T_1, T_2}(i-1, j-1) \cdot k_{tree}^r(\tau_{T_1}(fils_i(rac(T_1))), \tau_{T_2}(fils_j(rac(T_2))))
 \end{aligned} \tag{6}$$

On généralise ce noyau en s'appuyant sur deux idées ; d'une part utiliser la similarité entre les étiquettes, d'autre part représenter les sous-arbres non-contigus (Kashima et Koyanagi, 2002).

Soit Σ l'ensemble de toutes les étiquettes et $f : \Sigma \times \Sigma \rightarrow [0, 1]$ indiquant un score de "mutation" entre deux étiquettes tel que $f(e, a)$ indique la probabilité d'acceptation de la mutation de l'étiquette a vers e , la fonction de similarité entre deux étiquettes de deux nœuds n_1 et n_2 est :

$$Sim(l(n_1), l(n_2)) = \sum_{a \in \Sigma} f(l(n_1), a) \cdot f(l(n_2), a)$$

En introduisant la fonction de similarité dans l'équation 5, on obtient :

$$k_{tree}^r(T_1, T_2) = Sim(l(n_1), l(n_2)) \cdot S_{T_1, T_2}(nf(rac(T_1)), nf(rac(T_2))) \tag{7}$$

On peut encore généraliser en utilisant, au niveau du noyau, de la notion d'élasticité des sous-arbres. Ainsi, la définition de sous-arbre se voit élargie en permettant la non contiguïté au niveau des nœuds d'un chemin. Il n'est ici plus nécessaire qu'un chemin d'un sous-arbre apparaisse de manière contiguë dans un arbre. Cependant, la contrainte d'arbre ordonné reste valable.

Définition 5 (Sous-arbre non contigu) *Un arbre S (possédant au moins un nœud) est un sous-arbre de T si et seulement si il existe un nœud n de T tel que $l(n) = l(rac(S))$ ($l(n)$ et une liste ordonnée d'index $\{j_1, \dots, j_k\}$ tel que $\forall i, \tau_S(fils_i(rac(S)))$ soit un sous arbre de $\tau_T(fils_{j_i}(\tau_T(n)))$.*

Afin de prendre en compte la définition de sous-arbre non-contigu, il est nécessaire de généraliser la formule de $k_{tree}^r(T_1, T_2)$ (équations 5 et 7). En effet, cette formule retourne une valeur nulle (ou faible selon la similarité) si les étiquettes des racines sont différentes. En d'autre terme, tout sous-arbre commun à T_1 et T_2 doit être enraciné aux nœuds racines de T_1 et T_2 impliquant ainsi que les arbres possèdent la même racine. Or, dans le cas des sous-arbres non-contigus, un sous-arbre commun à T_1 et T_2 peut être construit par combinaison (ordonnée) à partir de sous-arbres enracinés à n'importe quels nœuds descendants de T_1 et T_2 .

Soit k_{old}^r le noyau défini par l'équation 5, le noyau spécifique pour les arbres élastiques utilisés par le noyau *Tree Kernel* (équation 4) est :

$$k_{tree}^r(T_1, T_2) = \sum_{n_1 \in T_1} \sum_{n_2 \in T_2} k_{old}^r(\tau_{T_1}(n_1), \tau_{T_2}(n_2))$$

Cette formule peut être calculée efficacement de manière récursive :

$$\begin{aligned}
k_{tree}^r(T_1, T_2) &= k_{old}^r(T_1, T_2) + \sum_{n_i = \text{fils}_i(T_1)} k_{tree}^r(\tau_{T_1}(n_i), T_2) \\
&+ \sum_{n_j = \text{fils}_j(T_2)} k_{tree}^r(T_1, \tau_{T_2}(n_j)) \\
&- \sum_{n_i = \text{fils}_i(T_1)} \sum_{n_j = \text{fils}_j(T_2)} k_{tree}^r(\tau_{T_1}(n_i), \tau_{T_2}(n_j))
\end{aligned}$$

De même que dans le cas du noyau *parse tree kernel* vu dans la section précédente, la complexité du noyau *tree kernel* est $O(|T_1| \cdot |T_2|)$ dans les différents cas de généralisation.

Kashima et Koyanagi (2002) ont utilisé les noyaux généralisés élastiques et non élastiques, sans tenir compte des mutations d'étiquettes, pour la classification et l'extraction d'information à partir de documents HTML. Les performances ont été évaluées en utilisant la méthode de validation croisée *leave-one-out*. L'apprentissage a été effectué en utilisant l'algorithme du perceptron "kernelisé".

Pour la classification de documents, une base de données comprenant 30 documents HTML en japonais et 30 documents HTML en anglais a été construite en extrayant les documents aléatoirement sur les sites web américains et japonais d'IBM. La classification devant être purement structurelle, seules les balises HTML ont été préservées, éliminant ainsi les attributs et les données. De plus, les noyaux sur arbres ont été combinés avec le noyau polynomial. Les résultats ont montrés que le noyau non élastique était de 12% plus performant que le noyau élastique en atteignant près de 80% de bon classement. Ces résultats ont été obtenus avec un noyau polynomial d'ordre 4 (resp. 3).

L'extraction d'information consiste à apprendre et à reconnaître une information précise dans des documents HTML. Il s'agit ici du marquage des nœuds pertinents. Le problème du marquage consiste à apprendre à partir d'arbres correctement marqués puis à marquer les nœuds pertinents des arbres non traités. Ce problème peut être ramené à un problème de classification en effectuant une transformation du marquage. En effet, pour un nœud marqué, on insère entre le nœud concerné et le nœud père, un nœud portant une étiquette appropriée pour signaler le marquage. Puis, un ensemble d'arbres négatifs est générés à partir des arbres corrects en retirant le marquage et en les plaçant sur des nœuds non initialement marqués. Un apprentissage peut ensuite être effectué sur ces données. Pour le marquage sur un arbre, on effectue pour chacun de ses nœuds un marquage puis on le classe.

Pour l'expérimentation, une base a été créée à partir de 54 pages HTML extraites d'un catalogue de vente d'ordinateurs portables d'IBM Japon. L'objectif de l'expérience était de retrouver l'image de l'ordinateur à vendre. Pour cela, les nœuds contenant les images pertinentes ont été marqués et les données textuelles présentes dans les pages ont été éliminées. Les résultats ont montré que le noyau élastique a permis d'extraire l'information avec une précision de 99.3% et un rappel de 68.6% contre une précision de 11.9% et un rappel de 79.6% pour le noyau non élastique. Ces résultats ont été obtenus sans la combinaison avec le noyau polynomial. En effet, ce dernier n'a pas permis d'améliorer les résultats.

4.3 Le noyau *Tree kernel* marginalisé

Le noyau marginalisé pour les arbres a été introduit par Kashima et Tsuboi (2004) pour répondre aux problèmes d'étiquetages (voir la section sur le noyau marginalisé sur les séquences).

L'objectif de ce noyau est de permettre de calculer la similarité entre deux arbres étiquetés. Un arbre étiqueté étant simplement un arbre où chaque nœud représente un élément observable (un terme) et à chaque nœud est associé une étiquette. L'avantage de résoudre un problème d'étiquetage en utilisant un modèle de donnée arborescent, plutôt que séquentiel, est qu'il est possible d'exploiter l'information structurelle pour améliorer la discrimination.

Le noyau marginalisé sur les arbres est obtenu en intégrant le noyau *Tree kernel* généralisé dans le cadre théorique du noyau marginalisé défini par la série d'équation 2.

Ainsi, $k_d(T_1, T_2, \tau, t)$ est le noyau ne prenant en compte que les sous-arbres ayant la même racine $rac(T_{1_\tau})$ (T_{1_τ} indique ici le sous-arbre de T_1 induit par le nœud d'index τ). De même, $k_u(T_1, T_2, \tau, t)$ ne prend en compte que les sous-arbres ayant une feuille correspondant au nœud d'index τ de T_1 ($rac(T_{1_\tau})$).

Pour le calcul de k_d , on se base sur les équations 5 et 6. L'équation 6 calcule la somme des contributions des sous-arbres communs à T_1 et T_2 en explorant à chaque niveau les fils de droite à gauche. En modifiant cette équation on obtient :

$$\begin{aligned} S_F(T_1, T_2, \tau, t, 0, 0) &= S_F(T_1, T_2, \tau, t, i, 0) = S_F(T_1, T_2, \tau, t, 0, j) = 1 \\ S_F(T_1, T_2, \tau, t, i, j) &= S_F(T_1, T_2, \tau, t, i-1, j) + S_F(T_1, T_2, \tau, t, i, j-1) \\ &\quad - S_F(T_1, T_2, \tau, t, i-1, j-1) \\ &\quad + S_F(T_1, T_2, \tau, t, i-1, j-1) \cdot k_d(T_1, T_2, ch(T_1, \tau, i), ch(T_2, t, j)) \end{aligned}$$

Avec $ch(T_1, \tau, i)$ l'index, dans T_1 , du $i^{\text{ème}}$ fils de la racine de T_{1_τ} .

Le noyau k_d devient alors :

$$k_d(T_1, T_2, \tau, t) = c^2 k(T_{1_\tau}, T_{2_t}) \cdot (1 + S_F(T_1, T_2, \tau, t, nf(rac(T_{1_\tau})), nf(rac(T_{2_t}))))$$

Pour k_u , le calcul s'effectue de la feuille vers la racine. Ainsi, on utilise la fonction $pa(T_1, \tau)$ qui calculera l'index du père du $\tau^{\text{ème}}$ nœud dans T_1 . En outre, il faut aussi tenir compte des frères gauches et des frères droits du $\tau^{\text{ème}}$ nœud de T_1 . Pour la contribution des frères gauches, la fonction S_F pourra être utilisée. Quant aux frères droits, il faudra les explorer de la gauche vers la droite. On utilisera la fonction $n = chID(T_1, \tau)$ pour indiquer que le nœud d'index τ est le $n^{\text{ème}}$ fils de son père. On définit donc, une fonction symétrique à S_F :

$$\begin{aligned} S_B(T_1, T_2, \tau, t, i, j) &= 1 \text{ si } i \geq nf(rac(T_{1_\tau})) \text{ ou si } j \geq nf(rac(T_{2_t})) \\ S_B(T_1, T_2, \tau, t, i, j) &= S_B(T_1, T_2, \tau, t, i+1, j) + S_B(T_1, T_2, \tau, t, i, j+1) \\ &\quad - S_B(T_1, T_2, \tau, t, i+1, j+1) \\ &\quad + S_B(T_1, T_2, \tau, t, i+1, j+1) \cdot k_d(T_1, T_2, ch(T_1, \tau, i), ch(T_2, t, j)) \end{aligned}$$

L'expression de k_u est :

$$\begin{aligned}
 k_u(T_1, T_2, \tau, t) &= c^2 k(T_{1\tau}, T_{2t}) \cdot (1 + k_u(T_1, T_2, \tau, t)) \\
 &\quad \cdot S_F(pa(T_1, \tau), pa(T_2, t), \tau, t, chID(T_1, \tau) - 1, chID(T_2, t) - 1) \\
 &\quad \cdot S_B(pa(T_1, \tau), pa(T_2, t), \tau, t, chID(T_1, \tau) + 1, chID(T_2, t) + 1)
 \end{aligned}$$

Les expériences menées sur ce noyau sont décrites dans la section sur le noyau marginalisé pour les séquences.

5 Les noyaux pour 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 étudiés précédemment peuvent être vus comme des graphes acycliques orientés (dans le cas d'une séquence, le graphe est de degré maximum 1).

Nous définirons un graphe étiqueté G par le triplé (V, E, σ) où V est l'ensemble des nœuds de G , σ l'ensemble des étiquettes tel que σ_i représente l'étiquette du nœud i (il est aussi possible d'étiqueter les arcs : on notera $\sigma_{(i,j)}$ l'étiquette de l'arc reliant le nœud i à j) et E , une matrice d'adjacence tel que $E_{ij} = 1$ si et seulement si il existe un arc reliant le nœud i au nœud j (afin de simplifier l'écriture on utilisera la même notation i, j pour désigner les indexes dans E que pour désigner les nœuds de V). L'une des propriétés de la matrice d'adjacence est que $[E^n]_{ij}$ indique le nombre de chemins de longueur n reliant le nœud i au nœud j .

La conception d'un noyau nécessite une définition de la similarité entre deux graphes. Pour cela, deux approches ont été proposées (Gärtner, 2003; Gärtner et al., 2006). 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). Comme pour la première approche, le problème du chemin hamiltonien est NP-difficile.

Afin de réduire la complexité dans l'évaluation de la similarité d'autres approches ont été explorées. L'approche la plus répandue consiste à calculer la similarité en se basant sur les chemins parcourus (Gärtner et al., 2003, 2006; Kashima et Inokuchi, 2002; Kashima et al., 2003). L'un des problèmes principaux de cette approche est qu'il existe une infinité de chemins possibles dès lors qu'il existe dans le graphe un cycle. On a alors le noyau :

$$k(G, G') = \lim_{n \rightarrow \infty} \sum_{i=1}^n \sum_{p \in P_i(G)} \sum_{p' \in P_i(G')} \lambda_i \cdot k_p(p, p') \quad (8)$$

Avec $P_l(G)$ l'ensemble des chemins de G de longueur l , λ_l un réel pondérant les chemins de longueur l (on fixera $\lambda_l = \lambda^l$) et k_p un noyau défini sur les chemins. Il existe plusieurs façons de définir k_p selon qu'on veuille tenir compte des étiquettes sur les nœuds, sur les arcs ou encore permettre des discontinuités (*gap*).

Dans (Gärtner et al., 2003, 2006), k_p est défini sur des chemins contiguës en tenant compte des étiquettes sur les nœuds et sur les arcs. Ainsi, le noyau sur les arcs revient à énumérer le nombre de chemins communs aux deux graphes.

De plus, l'équation 8 est réécrite plus élégamment en utilisant la propriété de la matrice d'adjacence. Pour cela, un nouveau graphe $G_\times : (V_\times, E_\times, \sigma_\times)$ est introduit en effectuant le produit direct des graphes $G : (V, E, \sigma)$ et $G' : (V', E', \sigma')$:

$$\begin{aligned} V_\times &= \{(v, v') \in V \times V' \mid \sigma_v = \sigma_{v'}\} \\ \sigma_{\times_{k=(v, v')}} &= \sigma_v \end{aligned}$$

Pour (i, j) correspondant à $((u, u'), (v, v')) \in V_\times^2$

$$\begin{aligned} [E_\times]_{i,j} &= \begin{cases} 1 & \text{si } [E]_{u,v} = [E']_{u',v'} = 1 \text{ et } \sigma_{(u,v)} = \sigma'_{(u',v')} \\ 0 & \text{sinon} \end{cases} \\ \sigma_{\times(i,j)} &= \sigma_{(u,v)} \end{aligned}$$

L'équation 8 devient :

$$k(G, G') = \lim_{n \rightarrow \infty} \sum_{i=1}^n \sum_{u,v}^{|V_\times|} \lambda^i \cdot [E_\times^i]_{u,v}$$

L'expression E_\times^i peut être simplifiée si la matrice E_\times est diagonalisable. Dans le cadre d'un graphe non-orienté, la matrice E_\times étant une matrice réelle et symétrique, elle peut être diagonalisée.

Ainsi, si E_\times peut être exprimé sous la forme $T^{-1} \cdot D \cdot T$ alors $E_\times^i = T^{-1} \cdot D^i \cdot T$. On peut alors réécrire le noyau sous la forme :

$$k(G, G') = \sum_{u,v}^{|V_\times|} (T^{-1} \cdot (\lim_{n \rightarrow \infty} \sum_{i=1}^n \lambda^i \cdot D^i) \cdot T)_{u,v}$$

Pour calculer la limite, Gärtner et al. (2003) propose d'utiliser une décomposition en série exponentielle ou en série géométrique.

La décomposition en série exponentielle se base sur l'égalité

$$e^{\beta \cdot E} = \lim_{n \rightarrow \infty} \sum_{i=0}^n \frac{(\beta E)^i}{i!}$$

Ainsi, en fixant $\lambda^i = \frac{\beta^i}{i!}$, on obtient :

$$k(G, G') = \sum_{u,v}^{|V \times|} (T^{-1} \cdot e^{\beta \cdot D} \cdot T)_{u,v}$$

De même, la décomposition en série géométrique se base sur, pour $\gamma < 1$:

$$\lim_{n \rightarrow \infty} \sum_{i=0}^n \gamma^i = \frac{1}{1 - \gamma}$$

En fixant $\gamma = \lambda \cdot D$ et en veillant à ce que $\lambda \cdot D < \mathbf{I}$, on a :

$$k(G, G') = \sum_{u,v}^{|V \times|} (T^{-1} \cdot (\mathbf{I} - \lambda \cdot D)^{-1} \cdot T)_{u,v}$$

Dans Kashima et Inokuchi (2002), les auteurs ont décomposé la similarité de deux graphes en une somme de similarité entre nœuds :

$$k(G, G') = \frac{1}{|V| \cdot |V'|} \sum_{v_i \in V, v_j \in V'} k_n(v_i, v_j)$$

Avec V et V' l'ensemble des nœuds de G et respectivement de G' .

La similarité entre deux nœuds sera d'autant plus importante qu'il existera des chemins longs communs aux deux graphes issus de ces nœuds. Pour assurer la terminaison du calcul, une probabilité $1 - \lambda$ est fixée pour terminer le chemin et une probabilité λ pour continuer vers un successeur du nœud courant. Ainsi, plus le chemin sera long et plus la probabilité de terminer le chemin deviendra importante. On obtient, ainsi, le noyau suivant :

$$k_n(u, u') = I(u, u') \cdot ((1 - \lambda) + \lambda \cdot \sum_{(v, v') \in A_G(u) \times A_{G'}(u')} \frac{I_A((u, v), (u', v'))}{|A_G(u)| \cdot |A_{G'}(u')|} \cdot k_n(v, v'))$$

Avec $A_G(u) = \{v \in V | E_{uv} = 1\}$, $I(u, u') = 1$ si les étiquettes des nœuds u de G et u' de G' sont identiques ou 0 sinon et de même pour $I_A((u, v), (u', v'))$ qui retourne 1 si l'étiquette de l'arc reliant u et v de G est identique à l'étiquette de l'arc reliant u' et v' de G' .

Dans Kashima et al. (2003), un noyau marginalisé sur tous les chemins possible est proposé en ne considérant que des graphes orientés. Ainsi, le noyau est défini par :

$$k(G, G') = \lim_{L \rightarrow \infty} \sum_{l=1}^L \sum_{\mathbf{h}} \sum_{\mathbf{h}'} k_z(G, G', \mathbf{h}, \mathbf{h}') \cdot P(\mathbf{h}|G) \cdot P(\mathbf{h}'|G')$$

La probabilité a posteriori d'avoir un chemin \mathbf{h} de G de longueur l ($P(\mathbf{h}|G)$) est défini en fonction de la probabilité de débiter un chemin par un nœud h_1 ($P_s(h_1)$), la probabilité de

terminer ce chemin par un nœud h_l ($P_q(h_l)$) et les probabilités d'effectuer une transition d'un nœud h_i vers un nœud h_{i+1} ($P_t(h_{i+1}|h_i)$). D'où :

$$P(\mathbf{h}|G) = P_s(h_1) \cdot \prod_{i=2}^{l=|\mathbf{h}|} P_t(h_i|h_{i-1}) \cdot P_q(h_l)$$

Le noyau k_z effectue la comparaison des deux chemins \mathbf{h} et \mathbf{h}' des graphes respectifs G et G' . Pour cela, le noyau calcule le produit des similarités entre les étiquettes des nœuds et des arcs du chemins :

$$k_z(G, G', \mathbf{h}, \mathbf{h}') = \begin{cases} 0 & \text{si } |\mathbf{h}| \neq |\mathbf{h}'| \\ k_e(\sigma_{h_1}, \sigma'_{h'_1}) \prod_{i=2}^l k_e(\sigma_{(h_{i-1}, h_i)}, \sigma'_{(h'_{i-1}, h'_i)}) \cdot k_e(\sigma_{h_l}, \sigma'_{h'_l}) & \text{sinon} \end{cases}$$

On rappelle que σ_{h_i} indique l'étiquette du nœud h_i de G et que $\sigma_{(h_i, h_{i+1})}$ indique l'étiquette de l'arc reliant le nœud h_i à h_{i+1} . Étant données deux étiquettes e et e' , on peut définir k_e comme étant un noyau retournant 1 si $e = e'$ ou 0 sinon. Toutefois, on peut définir un noyau plus complexe si les étiquettes sont des réelles avec une certaine métrique. On pourrait alors définir un noyau gaussien qui tolérerait certaines différences entre les étiquettes.

Outre les graphes que nous venons de voir, Suzuki et al. ont introduit dans (Suzuki et al., 2003a,b) la notion de graphe acyclique orienté hiérarchique (*HDAG*). Les *HDAG* sont des graphes dont certains nœuds contiennent des graphes acycliques orientés. Cette structure a été proposée pour permettre la représentation de documents textuels ainsi que d'informations connexes. En effet, un document textuel peut subir de multiple pré-traitement et des informations grammaticales et sémantiques peuvent lui être ajoutées. Ces informations combinées entre elles forment des structures hiérarchiques complexes.

En outre, un noyau a été proposé pour calculer la similarité entre les *HDAG*. Ce noyau a été évalué sur un problème de classification multi-classe avec l'algorithme SVM et la méthode "un contre tous" (un classifieur SVM par classe). Une base de données de 3000 questions divisées en 148 classes a été utilisée pour l'expérimentation. Les questions ont été pré-traitées à l'aide d'un *parser*. En outre, les entités nommées ont été étiquetées et les informations sémantiques ajoutées.

Les résultats ont montrés que le noyau *HDAG* était plus performant que le noyau *SubString Kernel* et le noyau "sac de mots".

6 Conclusion

Nous avons présenté une collection de méthodes assez variées, adaptées aux différents cas rencontrés lors du traitement des données structurées. La diversité de ces méthodes rend pour l'instant délicate toute évaluation comparative de leurs performances respectives. Le caractère très récent de ces travaux fait qu'il n'existe pas pour l'instant d'étude expérimentale comparative sérieuse des comportements de ces différents algorithmes sur des données issues du monde réel et plus particulièrement du langage naturel.

Insistons sur le fait que la souplesse des méthodes à noyaux facilite la construction de méthodes ad-hoc adaptées à la structure du problème à traiter.

Références

- Collins, M. (2002). Discriminative training methods for hidden markov models : Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*.
- Collins, M. et N. Duffy (2002). Convolution kernels for natural language. In *NIPS : Advances in Neural Information Processing Systems 14*, pp. 625–632. MIT Press.
- Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explor. Newsl.* 5(1), 49–58.
- Gärtner, T., K. Driessens, et J. Ramon (2003). Graph kernels and gaussian processes for relational reinforcement learning. In *ILP*, pp. 146–163.
- Gärtner, T., Q. V. Le, et A. J. Smola (2006). A short tour of kernel methods for graphs.
- Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz.
- Joachims, T. (2002). *Learning to Classify Text Using Support Vector Machines : Methods, Theory and Algorithms*. Norwell, MA, USA : Kluwer Academic Publishers.
- Kashima, H. et A. Inokuchi (2002). Kernels for graph classification. In *ICDM '02 : Proceedings of the First International Conference On Data Mining, Workshop on Active Mining*.
- Kashima, H. et T. Koyanagi (2002). Kernels for semi-structured data. In *ICML '02 : Proceedings of the Nineteenth International Conference on Machine Learning*, San Francisco, CA, USA, pp. 291–298. Morgan Kaufmann Publishers Inc.
- Kashima, H. et Y. Tsuboi (2004). Kernel-based discriminative learning algorithms for labeling sequences, trees, and graphs. In *ICML '04 : Proceedings of the twenty-first international conference on Machine learning*, New York, NY, USA, pp. 58. ACM Press.
- Kashima, H., K. Tsuda, et A. Inokuchi (2003). Marginalized kernels between labeled graphs. In T. Faucett et N. Mishra (Eds.), *Proceedings of the 20th International Conference on Machine Learning*, pp. 321–328. AAAI Press.
- Leslie, C. S., E. Eskin, et W. S. Noble (2002). The spectrum kernel : A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, pp. 566–575.
- Lodhi, H., C. Saunders, J. Shawe-Taylor, N. Cristianini, et C. J. C. H. Watkins (2002). Text classification using string kernels. *Journal of Machine Learning Research* 2, 419–444.
- Shawe-Taylor, J. et N. Cristianini (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.
- Suzuki, J., T. Hirao, Y. Sasaki, et E. Maeda (2003a). Hierarchical directed acyclic graph kernel : methods for structured natural language data. In *ACL '03 : Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, Morristown, NJ, USA, pp. 32–39. Association for Computational Linguistics.
- Suzuki, J., Y. Sasaki, et E. Maeda (2003b). Kernels for structured natural language data. In *NIPS : Advances in Neural Information Processing Systems 16*. MIT Press.
- Vapnik, V. N. (1995). *The nature of statistical learning theory*. New York, NY, USA : Springer-Verlag New York, Inc.

Summary

This paper review the application of kernel methods to the mining of structured data. Modern applications of data mining must handle structured data, e.g. for text mining, and learning algorithms should benefit of the use of this structural information, which is an interesting challenge. One of the possible approach to this problem is the use of Mercer's kernels. These kernels compute a similarity measure between complex data, and can be used in a lot of learning algorithms (Support Vector Machines, PCA, Discriminant Analysis, Perceptron, etc). We present the most important kernels proposed during the last years to handle structured data like sequences, trees and graphs.